QUALIDADE DE SOFTWARE

AULA 1

Prof.ª Maristela Regina Weinfurter Teixeira

CONVERSA INICIAL

Qualidade é uma área que há muito tempo é de grande importância para setor fabril, trazendo métodos, técnicas e ferramentas que garantem o controle de qualidade de produtos e serviços.

Dentro dos processos de desenvolvimento de *software* não é diferente. São métodos, técnicas e ferramentas, adaptadas ou não da área industrial, que permitem que o ciclo de vida de *software* tenha possibilidade de prover melhoria contínua para que processos e produto obtenham um grau de excelência.

A qualidade de *software* não visa somente tirar erros do código implementado, mas subsidia os requisitos, métricas utilizadas em gerenciamento, entre outras ferramentas e artefatos. O custo para correção de um produto de *software* quando já está nas fases finais do desenvolvimento é extremamente mais alto. Solucionar problemas nas fases iniciais (requisitos, análise e projeto) transformam os custos de manutenção a níveis menores. E há várias décadas funcionalidades de códigos estão comprometidas por falta de usabilidade, requisitos não atendidos, problemas na implantação e manutenção. Enfim, são vários momentos do processo que vêm comprometendo produto e processo. Em especial, o momento de manutenção e adaptação, o que gera um retrabalho enorme, onera tanto financeiramente quanto em termos de problemas que nunca se extinguem para engenheiros e desenvolvedores.

Todas as vezes que um *software* está inativo ou funcionando parcialmente acumulam prejuízos financeiros diretos e problemas com a imagem da empresa que não consegue manter um *software* funcionando adequadamente e satisfazendo as necessidades dos clientes.

A forma que temos para melhorar ou pelo menos minimizar tais situações é com a implantação de uma boa gerência e controle de qualidade. Iniciando-se desde o momento dos requisitos de um *software* até 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*.

A qualidade de *software* é parte integrante da engenharia de *software*, e pensando desta forma, a IEEE (Institute of Electrical and Electronics Engineering) em colaboração com a ACM (Association for Computing Machinery), elaboraram um guia para engenharia e qualidade de *software*, o

Swebok. Através dele conseguimos estabelecer melhores práticas e padrões que auxiliem na construção, validação e verificação de *software* durante todo o ciclo de vida do *software*.

Percebemos que erros, falhas e defeitos são características que ocorrem de longa data dentro do desenvolvimento de *software* e que precisam ser compreendidos para que todas as boas práticas de engenharia de *software* sejam aplicadas para garantir a qualidade.

Finalmente, os dois pontos fortes dentro de toda a garantia de qualidade de *software* se baseiam na verificação e na validação, pois elas se misturam aos dois processos: desenvolvimento e garantia da qualidade. Garantir a qualidade de *software* pode ser algo a princípio muito complexo, mas ela pode ser aplicada de forma evolutiva e iterativa ao processo de desenvolvimento, sendo que tudo pode começar por pequenas tarefas de verificações e testes.

TEMA 1 – PRINCÍPIOS EM QUALIDADE

Inicialmente vamos trabalhar conceitos inerentes à qualidade de bens e serviços de forma geral. Suas origens, filosofias e ideias que inevitavelmente foram transportadas para o conteúdo deste material: qualidade de *software*.

As mudanças não ocorrem ao acaso, nem tão pouco da noite para o dia. A qualidade precisa ser observada do ponto de vista do cliente e deve ser trabalhada para atingirmos o tão esperado produto ou serviço com a qualidade desejada por nosso cliente.

E a história das organizações, mais propriamente fábricas dos mais diversos produtos, demonstram que as empresas que alcançaram excelência em seus resultados foram exatamente as que priorizaram projeto, controle e melhoria contínua da qualidade.

Uma vez que as empresas levam em consideração a elevada satisfação do cliente, das partes interessadas e também dos colaboradores, isso permite que sustentem o desempenho de sua fabricação por um prazo bastante longo.

A primeira definição de qualidade que se tem notícias foi idealizada por Joseph M. Juran (Jurab; DeFeo, 2015) e diz que algo é apto a ser usado. Já o Dr. Deming utilizava o conceito de algo em conformidade com as exigências. Para Robert Galvin, ex-presidente do conselho da Motorola, o Seis Sigma era utilizado para distinguir o alto nível de qualidade relacionado aos defeitos. Alguns ainda falam que qualidade significa excelência de classe mundial. A ISO 8402

define o termo *qualidade* como todas as características de uma empresa que é capaz de satisfazer necessidades explícitas e implícitas dos clientes (ISO, 1994).

Antes de entrarmos nos assuntos propriamente ditos, é interessante compreendermos quem foram Juran e Deming, dois nomes muito difundidos na área de qualidade. Joseph M. Juran foi um consultor que atuou até meados dos anos 1990, tendo trabalhado para empresas americanas, japonesas européias, como Philips, Xerox, VolksWagen e Toyota. Criou os Instituto e Fundação Juran. (Juran; DeFeo, 2015). Enquanto que William E. Deming foi um professor universitário, estatístico, autor, palestrante e consultor americano que ficou reconhecido pela melhoria dos processos produtivos nos Estados Unidos durante a Segunda Guerra Mundial. No entanto, a partir de 1950 ficou conhecido pelo seu trabalho no Japão, onde ensinou altos executivos a melhoria de proejtos, qualidade de produtos, testes e vendas através da utilização de métodos e testes. Deming foi um estrangeiro no Japão que causou muito impacto na indústria e na economia, pois deixou um legado na fabricação de produtos inovadores de alta qualidade. Anos mais tarde K. Ishikawa formulou uma proposta de Ferramentas para Controle Estatístico de Qualidade, as quais são: folha de verificação, estratificação, diagrama de pareto, histograma, diagrama de Ishikawa (causa e efeito), diagrama de dispersão e gráfico de controle de processos.

Qualidade deve suprir a adequação do produto a um objetivo definido em função das necessidades do cliente, as quais estão compreendidas entre as necessidades dos vários clientes que possam auxiliar no projeto de bens e serviços adequados aos objetivos. Em outras palavras, que o produto ou serviço tenha as características certas (que satisfaçam as necessidades do cliente) e não tenha falhas.

O Quadro 1 é uma adaptação de Juran e DeFeo (2015) e representa o significado de *qualidade*.

Quadro 1 – Significados de qualidade

Características que atendam às necessidades do cliente	**Isenção de falhas**
Uma qualidade superior permite que as organizações: Aumentem a satisfação do cliente Produzam produtos úteis Encarem a concorrência Aumentem sua fatia no mercado Gerem receitas Garantam melhores margens em seus preços Reduzam riscos	Uma qualidade de excelência permite que as organizações: 1. Reduzam níveis de erros 2. Reduzam retrabalho e desperdícios 3. Reduzam falhas de campo e custos com garantias 4. Reduzam a inssatisfação dos clientes 5. Abreviem o tempo para colocar novos produtos no mercado 6. Aumente o rendimento e a capacidade 7. Melhorem o desempenho nas entregas
O efeito principal recai na receita	O efeito principal recai nos custos
Maior qualidade custa mais	Maior qualidade custa menos

Não podemos deixar de comentar que a qualidade impacta nos custos. São problemas relacionados a defeitos de fábrica, falhas de campo, entre outros que significam maior qualidade, menos erros, menos defeitos e menos falhas. Clientes insatisfeitos se referem a falhas, defeitos e erros.

Dentro do avanço e aperfeiçoamento da área de qualidade surgiu um modo sistemático para redução do número de deficiências ou *custos de qualidade* para criação de um nível mais alto de qualidade aliado a baixos custos, o Seis Sigma.

Naturalmente a qualidade impacta na receita da empresa. Maior qualidade traz melhores entregas de bens ou serviços, com clientes satisfeitos e vendas crescentes.

Quando temos bens ou serviços deficientes, estes geram custos que são repassados ao fornecedor e ao cliente e desestimulam a repetição das vendas.

O ideal é que empresas busquem a promoção da melhoria e inovação contínuas, gestão voltada à qualidade, métodos e ferramentas de qualidade, equipes de trabalho multidisciplinares e garanta a motivação dos colaboradores durante todo o processo. Alguns programas de eficiência organização para garantia da qualidade de processos na fabricação de bens ou serviços: Seis

Sigma, Lean, Sistema Toyota de Produção e Gestão da Qualidade Total (TQM em inglês).

Uma gestão voltada para qualidade segue três processos gerenciais apresentadas no Quadro 2 adaptado de Juran e DeFeo (2015).

Tabela 2 – Gestão voltada para qualidade

Planejamento de qualidade	Controle de qualidade	Melhoria da qualidade
Estabelecimento de metas Identificação de clientes Determinação das necessidades dos clientes Desenvolvimento das características que atendam às necessidades dos clientes Desenvolvimento dos processos capazes de produção de produtos Estabelecimento de controles processuais Transferência de planos para operação	Determinação do foco de controle Medição do desempenho real Comparação do desempenho real com alvos e metas Utilização de medidas conforme as diferenças Medição contínua para permanência do desempenho	Comprovação da necessidade de estudo de caso do negócio Estabelecimento de infraestrutura de projeto Identificação dos projetos de melhoria Formação de equipes de projeto Oferecimento de recursos, treinamento e motivação das equipes para: 1. Diagnosticar as causas 2. Estimular ações corretivas Estabelecimento de controles para manter ganhos

Fonte: Elaborado por Teixeira, 2022 com base em Juran; DeFeo, 2015.

Outras duas contribuições de Juran ficaram conhecidas como Trilogia Juran para gestão da qualidade (JMS – Juran Management System) e os Sete princípios da gestão da qualidade.

A trilogia de Juran (Figura 1) é composta por:

1. Planejamento: qualidade desejada para planejar e projetar meios para atingir os objetivos no processo;
2. Controle: utilização de diagnóstico de erros e acertos no processo;
3. Aperfeiçoamento: proposta de níveis sempre mais apurados de qualidade.

Figura 1 – Trilogia de Juran

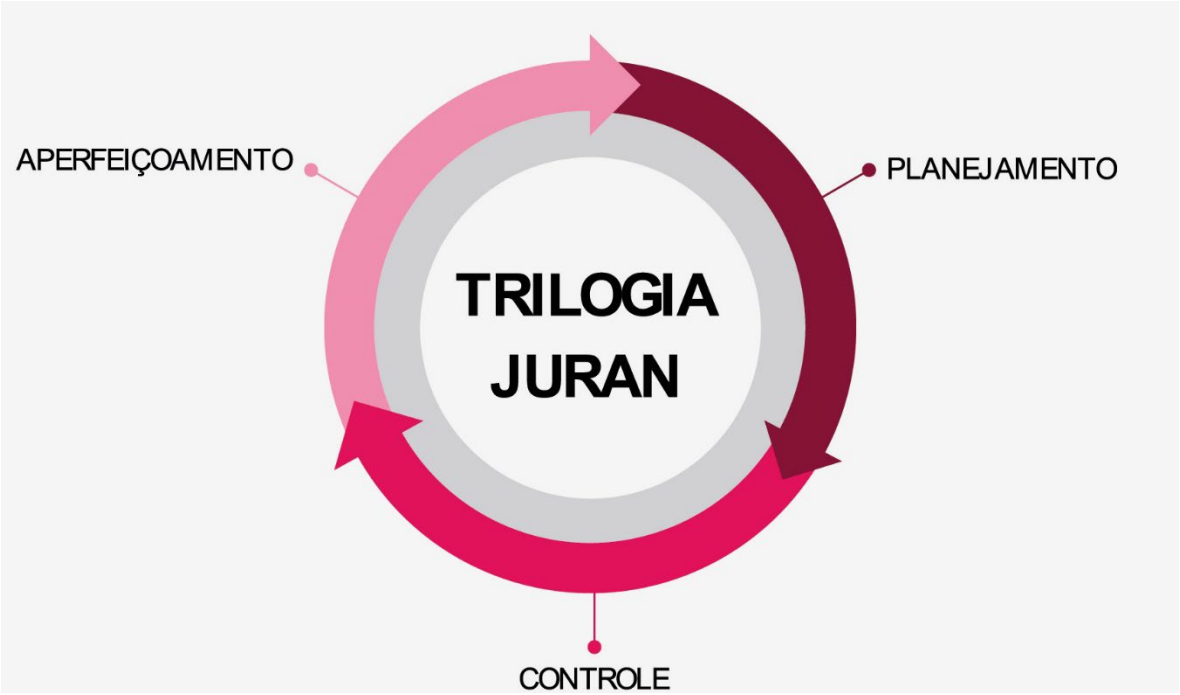
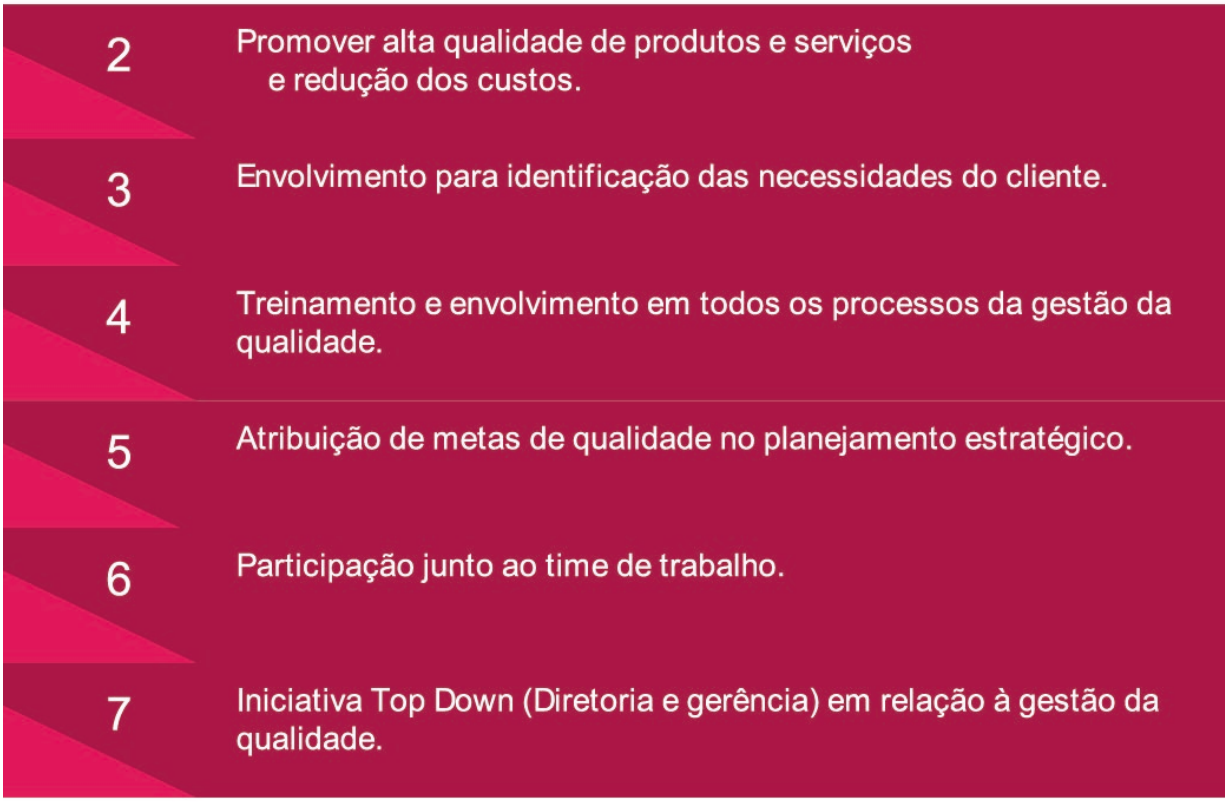


Figura 2 – Sete princípios de Juran



Fonte: Juran; DeFeo, 2015.

No final dos anos 1980 surgiram as normas ISO 9000 para regularização de relações comerciais e melhoria da gestão da qualidade. A ISO (International Organization for Standardization) é uma organização internacional que edita normas e diretrizes para garantia da qualidade. Com a implantação da ISO 9000, as empresas reduzem desperdícios, melhoria na eficiência dos recursos humanos e de equipamentos, análise de segurança e melhorias nas relações do mercado com aumento de produtividade e aumento dos lucros.

As teorias, *frameworks*, ferramentas e normas são utilizadas diariamente nas empresas que tenham como foco o desenvolvimento de produtos e serviços que atendam às necessidades dos clientes garantindo sua satisfação e contribuindo para melhoria nas vendas.

TEMA 2 – PRINCÍPIOS EM QUALIDADE DE *SOFTWARE* NA HISTÓRIA DA COMPUTAÇÃO

A engenharia de *software* é uma disciplina ou área de atuação muito jovem quando comparada às demais engenharias. Entre as décadas de 1950 e 1970 havia programadores desenvolvendo pequenas funcionalidades de forma isolada e um tanto quanto artesanais. É nesse contexto que iniciamos nossas primeiras soluções para área de TI e totalmente sem padrões, sem métricas, sem metodologias claras, mas resolvendo pontualmente problemas em empresas dos mais diversos mercados.

Mais especificamente, os primeiros encontros que impulsionaram a curiosidade pela área de computação ocorreram em 1928, segundo (The Origens..., S.d.). Neste período já existia a Big Blue, ou melhor dizendo, a IBM, que, por meio de seu presidente-executivo Thomas J. Watson, filho de um agricultor e madeireiro, o qual se formou apenas no ensino meio, iniciou encontros com o professor Benjamin Wood, professor da Universidade de Columbia. Nascia então a primeira disciplina científica na Universidade de Columbia, oferecendo um dos primeiros cursos com créditos acadêmicos em computação em 1946. A IBM ajudou a catalisar o ensino da ciência da computação. Neste mesmo tempo, outras universidades estavam na vanguarda da ciência da computação, tais como Universidade de Harvard (EUA), que projetou e construiu seu primeiro computador programável o Harvar Mark I, instalado na universidade em 1944. No ano de 1947 inicia o primeiro programa de graduação em ciência da computação da Universidade de Harvard. Já em

1953, a Universidade de Cambridge, na Grã-Bretanha, estabeleceu um programa de graduação em computação.

Figura 3 – The many colors of the IBM System/360

Crédito: Bernhaut/Picture-alliance/DPA/Imageplus.

Os primeiros departamentos de ciência da computação em faculdades americanas surgiram no início da década de 1960, tendo Frederick P. Brooks Jr, executivo da IBM que amava ensinar, como um dos pioneiros da área. Ele gerenciou o desenvolvimento do *mainframe* e do sistema operacional IBM System/360 (Figura 3), tendo sido professor voluntário na Universidade de Columbia e também no Systems Research Institute da IBM. Em 1963 aceitou o projeto para iniciar o departamento de Ciência da Computação da Universidade de North Carolina.

Foi então na década de 1970 que surgiu o termo *engenharia de software*, a qual inicia seus primeiros passos como disciplina e futuramente uma área consistente de atuação como a vemos na atualidade. E nessa mesma década é cunhado a expressão *crise do software*, a qual expressava literalmente grandes dificuldades no desenvolvimento de *software* diante da rápida demanda por *software*, bem como o crescimento de problemas e complexidade que precisavam ser resolvidos. Chamada de terceira era do *software*, os problemas mais comuns concentravam-se em baixa produtividade, baixa qualidade, prazos não cumpridos, custos altos e difícil manutenção (Pressman, 2011]. Mais de cinquenta anos já se passaram e muitos destes problemas continuam atuais. Mais de um terço dos projetos eram cancelados porque não atendiam aos requisitos, dois terços extrapolavam o orçamento. O custo de *hardware* em confronto com *software* era na medida de 8:2, enquanto hoje encontra-se em 1:9. Um exemplo clássico de falha no desenvolvimento de *software* foi o *software* Ariane 5 do projeto espacial da Agência Europeia, tendo custado oito bilhões de dólares e dez anos de desenvolvimento. A explosão encerrou com o projeto em 40 segundos após a decolagem, vindo a explodir e destruir o foguete e a carga avaliada em mais de quatrocentos e oitenta milhões de dólares.

É claro que estamos falando de um megaprojeto relacionado à corrida espacial, mas outros projetos menores não fugiam do mesmo problema.

Os tempos mudaram, mas continuamos com problemas em relação à qualidade de *software*, sejam eles aplicativos móveis, para Web, IoT, APIs, entre outros. Na realidade, aumentamos o nível de complexidade e dividimos a engenharia de *software* em subáreas que se preocupam com o desenvolvimento de *software*.

As possíveis soluções para melhoria da qualidade nos processos de desenvolvimento de software e do produto *software* traziam ideias sobre o uso de técnicas, ferramentas e processos sitematizados para produção.

Mais vinte anos se passaram, agora estamos na década de 1990, e estamos trabalhando com sistemas de informações como os ERPs (Enterprise Resource Planning), os quais acabaram por desperdiçar bilhões de dólares no desenvolvimento de *software* que não entregavam as características e as funcionalidades dos seus requisitos iniciais (Pressman, 2011].

Tempos difíceis para os desenvolvedores trouxeram novas ideias sobre a implantação da qualidade para o processo e para o produto. A complexidade e

a tarefa árdua de colocarmos em produção um *software* no prazo, com custos razoáveis, com requisitos atendidos e sem falhas, erros ou defeitos impulsionou a busca de mais critérios de melhorias. Esses critérios se originaram de mais modelos, regras, padrões, e principalmente de tudo aquilo que chamávamos de controle e garantia de qualidade dentro das fábricas.

Olhamos ao nosso redor e percebemos que tantas outras áreas de negócios já haviam passado por problemas similares e então buscamos tudo aquilo que pudesse nos ajudar a repensarmos o desenvolvimento de *software*, tais como ferramentas e *frameworks* lá das ideias de Juran e Deming. Sim, Kanban, Seis Sigma, gráficos, métricas e outras ideias que nos auxiliariam na melhoria de nossos projetos.

A engenharia de *software* então se reinventa, e se reposiciona como uma área mais consolidada e madura e proporciona o surgimento de normas e padrões ISO (9000-3, 9126, 15.504, 12.207, 1074, 1298), além de modelos CMM, CMMI e MPS.BR.

Hoje, mais de cinquenta anos depois que iniciamos nossos projetos de *software*, ainda nos primeiros *mainframes*, estamos trabalhando melhor com cultura e ética no processo de engenharia de *software*, bem como com custos de qualidade, custos de projetos, melhorias de processo e de produto, questões relacionadas à segurança de *software*, verificação e validação de *software*, revisão e auditoria, e dando um novo significado ao processo de engenharia de requisitos. Fazemos a distinção entre defeitos, erros e falhas e gerimos a qualidade de *software* através de métricas e ferramentas.

Além de métricas, metodologias, ferramentas e gestão da garantia da qualidade de *software*, um dos termos que mais nos remete à qualidade de *software* é Teste. Tudo é importante, mas testes de *software* se tornaram os queridinhos da qualidade de *software*. O que mais vemos em cursos, *sumits*, eventos e redes sociais relacionados à área de *software* são os testes de *software*. As empresas também perceberam que é um bom ponto de partida para implantação da qualidade em seus processos.

E falando em divulgação de qualidade de *software*, bem como da área toda de computação, há duas fontes muito importantes para estudantes e profissionais de TI: IEEE (Institute of Electrical and Electronics Engineering) e ACM (Association for Computing Machinery). O IEEE promove a engenharia de criação, desenvolvimento, integração, compartilhamento e conhecimento

aplicado a tudo que se refere à ciência e tecnologias da eletricidade e da informação. A ACM é uma sociedade que congrega educadores, pesquisadores e profissionais de computação. Hoje é a maior sociedade de computação do mundo, fortalecendo a ideia da profissão, promovendo padrões e reconhecimento da excelência técnica (IEEE, S.d.; ACM, S.d.]

A IEEE e a ACM colaboraram e patrocinaram a criação de um guia para processo e métodos em Engenharia de *Software*, o Swebok (Guide to the Software Engineering Body of Knowledge) (Swebok, S.d.]. Dentro do Swebok, encontramos a divisão da Engenharia de *Software* em dez áreas, e qualidade é o nosso destaque. A Figura 4 estabelece algumas subáreas da qualidade de *software*, segundo o Swebok.

Figura 4 – Pilares da qualidade de *software*

Fonte: Elaborada por Teixeira, 2022 com base em SWEBOK Guide v.3.0.

A qualidade de *software* traz consigo os primeiros conceitos em qualidade de forma geral: satisfazer às necessidades de nossos clientes ao entregarmos um produto que não contenha erros, defeitos ou falhas e que garanta que nossas vendas aumentem de forma virtuosa por meio de aplicativos móveis, IoT, aplicativos para Web, *software* embutido, entre tantas outras formas de construirmos um *software*. O desenvolvimento de *software* não foi nem

tampouco será uma tarefa fácil, até mesmo porque já discutimos que o nível de complexidade só aumenta, pois queremos que este funcione de forma simples, rápida, eficiente, multiplataforma, seguro e com zero falhas. Mas nossa experiência precisa considerar a melhoria contínua de nossos processos de desenvolvimento e *software* como produto.

TEMA 3 – *SWEBOK*: UM GUIA PARA ENGENHARIA E QUALIDADE DE *SOFTWARE*

O *Swebok* é um guia recomendado para promover a profissionalização da engenharia de *software*, auxiliando engenheiros de *software*, profissionais, estudantes e professores a clarificarem os limites desta.

Especificamente falando de qualidade de *software*, este manual promove conhecimento em relação à melhoria de produtos de *software* e dos processos de desenvolvimento e garantia da qualidade.

Os aspectos inerentes às características da qualidade de *software* foram desenhados em quatro itens importantes. A seguir, falaremos sobre cada um deles.

O primeiro item especifica *os fundamentos da qualidade de software*, no qual encontram-se aspectos relacionados a requisitos de qualidade, questões sobre ética, sociedade e legislação. A seguir elencamos subtópicos dos fundamentos da qualidade de *software*:

1. Engenharia de *software*, cultura e ética;
2. Custos e valores em qualidade;
3. Modelos e características da qualidade;
4. Melhorias em qualidade de *software*; e
5. Segurança de *software*.

O segundo item é representado pelos processos de gerenciamento de qualidade de *software*, que tratam sobre técnicas e procedimentos que suportam a identificação de erros em artefatos de software. Este item é representado pelo ISO/IEC 12207, que considera os seguintes tópicos:

1. Garantia da qualidade de *software*,
2. Validação e verificação e
3. Revisões e auditorias.

Dentro do tópico *Considerações Práticas* consideramos aspectos relacionados a técnicas, procedimentos, medições e monitoramento da qualidade do processo e do produto. Os subtópicos estão caracterizados por:

1. Requisitos de qualidade de *software*, que mencionam outros fatores que possam influenciar no comportamento dos requisitos, da segurança na execução do *software*, bem como em causas e consequências sobre falhas de *software*.
2. Caracterização de defeitos, a qual verifica se os requisitos estão em conformidade ou não.
3. Técnicas de gerenciamento de qualidade de *software*, são orientadas para revisões, auditorias, execução do *software* e técnicas analíticas por meio de métodos formais.
4. Métricas em qualidade de *software*, incluem medidas que determinam a qualidade do produto.

Finalmente, o tópico *Ferramentas de Qualidade de Software* faz menção à aplicação da qualidade de processos de desenvolvimento de *software* quanto no próprio produto (*software*) em questão.

Utilizarmos um guia que forneça uma visão geral de o que fazer e como fazer para garantirmos a qualidade de *software* é muito importante, pois o mesmo é baseado em experiências de profissionais que por sua vez, já passaram por problemas e conseguiram organizar e criar métodos, técnicas e ferramentas eficazes na melhoria contínua de *software*.

TEMA 4 – QUALIDADE DE *SOFTWARE* E OS *BUGS*

Falarmos sobre qualidade de *software* nos remete à área de testes de *software*, pois a qualidade está intimamente relacionada à solução de problemas que temos tanto no processo quanto no produto de *software*.

Diariamente ouvimos, após algumas reuniões de trabalho ou até mesmo quando ligamos para tentar resolver algum problema em bancos, comércio eletrônico, ou quaisquer outros serviços digitais dos quais somos usuários diretos ou indiretos, que houve algum problema no “sistema”.

São comentários conforme a lista abaixo:

- O sistema travou durante a produção;
- A área de TI cometeu um erro;

- Após uma revisão, encontramos um defeito no plano de teste;
- Encontrei um *bug* em um aplicativo hoje;
- O sistema quebrou;
- Uma falha foi relatada no subsistema de monitoramento.

Para identificação e correção ou melhoria em relação ao sistema, aplicativo, entre outros vocábulos que no final referem-se a um *software* a área de gestão da garantia da qualidade possui vários tipos de testes, verificações, validações, inspeções e auditorias, de acordo com a necessidade para cada situação.

Tais testes e outras formas de validações existem porque, de fato, podemos ter enganos, erros, defeitos e falhas no *software* (Pressman, 2011].

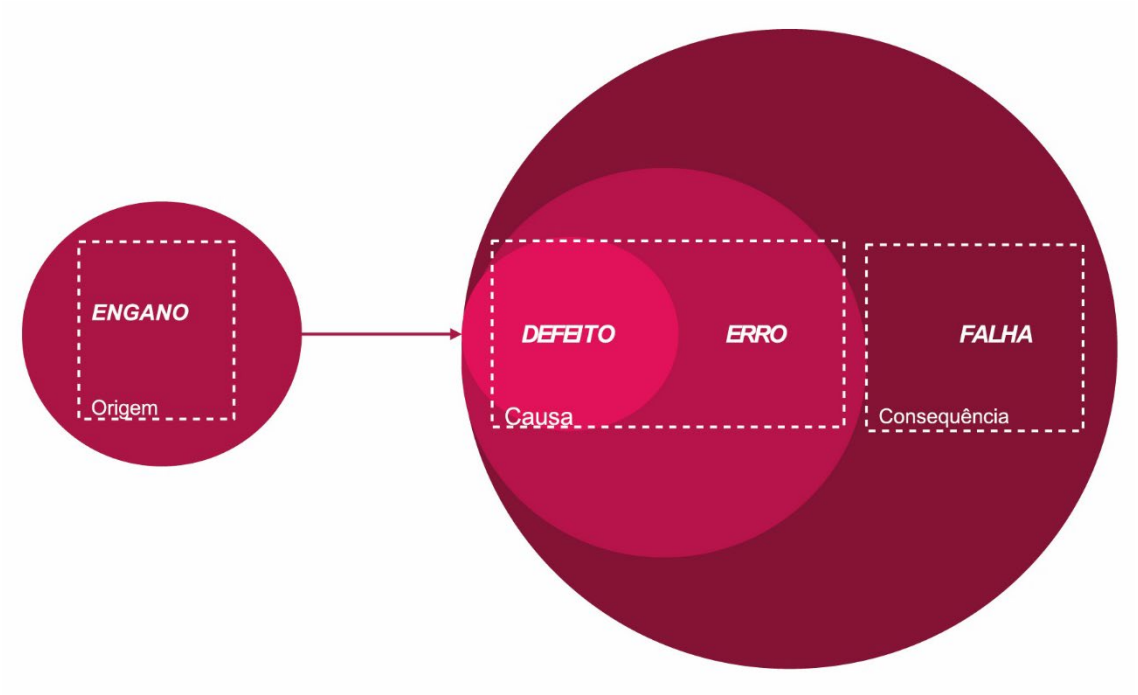
Vocábulos que a princípio parecem dizer a mesma coisa, porém cada um deles tem um significado diferente quando nos referimos a testes de *software*.

Vamos compreender então a diferenciação:

1. **Engano:** o engano é uma ação humana acidentalmente inclusa dentro de uma classe, uma função, ou outro elemento qualquer de um código de programa. Por exemplo, numa fórmula de cálculo de impostos sobre vendas, podemos incluir a fórmula de forma errada ou até mesmo a fórmula errada;
2. **Defeito**: o defeito é a consequência de um engano cometido em um trecho de código de programa, o qual resulta em saídas inesperadas ou inconsistentes;
3. **Erro**: o erro então se forma devido a um defeito causado por sua vez por um engano. E neste caso, o resultado do *software* é diferente do esperado;
4. **Falha**: a falha é uma consequência de um erro. Por exemplo, voltando ao cálculo dos impostos sobre vendas, que teve um defeito em decorrência da escrita de uma fórmula errada, gerou um erro que causou a falha na nota fiscal.

A Figura 5 demonstra esses conceitos de uma forma mais clara:

Figura 5 – Demonstração da relação entre enganos, defeitos, erros e falhas



No entanto a ISO 24765 diz que erro é uma ação humana que produz um resultado incorreto (ISO, 2017). Defeito é um problema ou falha que, se não for corrigido, pode fazer com que um aplicativo falhe ou produza resultados incorretos. Uma imperfeição ou deficiência em um *software* ou componente do sistema que pode fazer com que o componente não desempenhe sua função, por exemplo, uma definição de dados incorreta ou instrução de código-fonte, segundo a (ISTQB, S.d.). Um defeito, se executado, pode causar a falha de um software ou componente do sistema. E falha é o término da capacidade de um produto de desempenhar uma função requerida ou sua incapacidade de funcionar dentro de limites previamente especificados, conforme a ISO 25010 (ISO, 2011).

Segundo Pressman (2011], erros são as causas de problemas de qualidade de *software* e é necessário investigarmos suas causas, com a finalidade de prevenirmos e não vivermos continuamente corrigindo-os após as ocorrências.

Os erros podem ser divididos em nove tipos principais:

1. **Definição dos requisitos**: as definições incorretas acabam por gerar grande parte dos erros de *software*. Uma definição errada ou incompleta ocorre por falta de requisitos essenciais para construção do *software*;

2. **Falhas de comunicação:** a comunicação é algo essencial em qualquer processo empresarial, e no desenvolvimento de *software* não é diferente. Usuários e equipe técnica precisam construir pontes de comunicação para que os requisitos sejam elaborados levando em consideração as reais necessidades do *software*;
3. **Desvios nos requisitos de *software***: quando os desenvolvedores fogem do contexto dos requisitos, podem gerar problemas no *software*. Por exemplo, a omissão de funcionalidades em decorrência de prazos e orçamentos podem causar a aprovação de funcionalidades incompletas;
4. **Erros de projeto lógico**: definições no uso de bibliotecas erradas podem causar problemas no funcionamento de determinado *software*, e tais erros são introduzidos no *software* que acaba gerando uma sequência de erros;
5. **Erros de codificação**: este é o tipo de erro relacionado à lógica de programação e utilização de estruturas equivocadamente no *software*, podendo provocar a inclusão de erros durante a codificação;
6. **Não conformidade com documentação**: documentação com problemas e não em conformidade dificulta o êxito das equipes de testes, pois geram massas de testes equivocadas;
7. **Falhas no processo de testes**: Planos de testes precisam estar completos, bem como documentações e relatos de erros e falhas. Quando da ocorrência e detecção de falhas, as correções devem ser feitas o mais rápido possível;
8. **Erros de UI (User Interface)**: Interfaces mal projetadas, principalmente pela falta de observância de técnicas e heurísticas que conduzam uma interação centrada no usuário, também são causas de problemas no *software*;
9. **Erros de documentação**: erros encontrados na documentação, em manuais do *software*, ou em documentos integrados ao corpo do código são algumas situações deste tipo de erro.

Segundo a classificação anterior dos focos sobre inclusão de erros num *software*. Para que um requisito seja considerado com boa qualidade, ele deveria atender as seguintes características (Laporte, 2018):

- Estar correto;
- Estar completo;

- Ser claro para cada grupo de partes interessadas (por exemplo, o cliente, o arquiteto do sistema, testadores e aqueles que irão manter o sistema);
- Não ter ambiguidade, ou seja, mesma interpretação do requisito por parte de todas as partes interessadas;
- Ser conciso (simples, preciso);
- Estar consistente;
- Ser viável (realista, possível);
- Ser necessário (responde à necessidade do cliente);
- Ser independente do projeto;
- Ser independente da técnica de implementação;
- Ser verificável e testável;
- Poder ser rastreado até uma necessidade de negócios;
- Ser único.

Os erros também podem ocorrer devido a problemas na comunicação entre o pessoal de *software* e os usuários. Alguns exemplos de má compreensão entre usuários e desenvolvedores podem ser por exemplo:

- Má compreensão das instruções do cliente;
- O cliente quer resultados imediatos;
- O cliente ou o usuário não se dedique a ler a documentação que lhe foi enviada;
- Má compreensão das mudanças solicitadas aos desenvolvedores durante o projeto;
- O analista deixa de aceitar mudanças durante a fase de definição e desenho dos requisitos, uma vez que para determinados projetos 25% das especificações terão sido alteradas antes do final do projeto.

Outro aspecto de erros também pode ocorrer quando o desenvolvedor interpreta incorretamente os requisitos e desenvolve com base em sua própria interpretação. Deste tipo de situação, os desvios podem ser por:

- reutilizar o código existente sem fazer os ajustes adequados para atender aos novos requisitos;
- decidir abandonar parte dos requisitos devido a pressões orçamentárias ou de tempo;

- iniciativas e melhorias introduzidas por desenvolvedores sem verificação com os clientes.

A arquitetura do *software* também pode inserir problemas aos requisitos serem traduzidos de forma equivocada. Os erros de projeto típicos são os seguintes:

- Uma visão incompleta do software a ser desenvolvido;
- Papel pouco claro para cada componente da arquitetura de *software* (responsabilidade, comunicação);
- Dados primários não especificados e classes de processamento de dados;
- Um projeto que não usa os algoritmos corretos para atender aos requisitos;
- Sequência incorreta de processos de negócios ou técnicos;
- Desenho deficiente de critérios de regras de negócios ou processos;
- Um projeto que não rastreie os requisitos;
- Omissão de *status* de transação que representem corretamente o processo do cliente;
- Falha no processamento de erros e operações ilegais, o que permite que o *software* processe casos que não existiriam no setor de negócios do cliente – estima-se que até 80% do código do programa processe exceções ou erros.

Muitos erros podem ocorrer na construção do *software*. Erros podem ser por ineficiências comuns de programação, como:

- Escolha inadequada de linguagem de programação e convenções;
- Não abordando como gerenciar a complexidade desde o início;
- Má compreensão/interpretação de documentos de projeto;
- Abstrações incoerentes;
- Erros de *loop* e condição;
- Erros de processamento de dados;
- Erros de sequência de processamento;
- Falta ou validação deficiente dos dados na entrada;
- Desenho deficiente de critérios de regras de negócios;

- Omissão de status de transação que são necessários para representar verdadeiramente o processo do cliente;
- Falha no processamento de erros e operações ilegais, o que permite ao *software* processar casos que não existiriam no setor de negócios do cliente;
- Má atribuição ou processamento do tipo de dados;
- Erro no *loop* ou interferência no índice do loop;
- Falta de habilidade para lidar com ninhos extremamente complexos;
- Problema de divisão de inteiros;
- Má inicialização de uma variável ou ponteiro;
- Código-fonte que não remonta ao design;
- Confusão sobre um alias para dados globais (variável global passada para um subprograma).

A falta de documentação do *software* também pode ser causa de problemas, tais como:

- Quando os membros da equipe de *software* precisam coordenar seu trabalho, eles terão dificuldade em entender e testar software mal documentado ou não documentado;
- A pessoa que substituir ou manter o *software* terá apenas o código-fonte como referência;
- A garantia da qualidade encontrará um grande número de não conformidades (no que diz respeito à metodologia interna) em relação a este *software*;
- A equipe de teste terá problemas para desenvolver planos e cenários de teste, principalmente porque as especificações não estão disponíveis.

Observamos que há várias origens que por algum tipo de engano pode acarretar defeitos contendo erros e que fazem com que o *software* falhe. Não é tarefa fácil a garantia de que tais problemas possam ser prevenidos, porém, com a inclusão de atividades relacionadas à verificação (durante o processo de desenvolvimento) e validações (pós desenvolvimento de *software*) podem gerar bons resultados tanto a nível de *software* quanto de processo de desenvolvimento.

TEMA 5 – IMPORTÂNCIA DA VERIFICAÇÃO E VALIDAÇÃO NA QUALIDADE DE *SOFTWARE*

O padrão IEEE1012 (IEEE XPLORE, 2016) nos fornece uma visão dos objetivos da V&V (*validação* e *verificação*) em projetos de *software* para auxiliar as empresas na incorporação da garantia da qualidade ao longo do ciclo de vida de *software*.

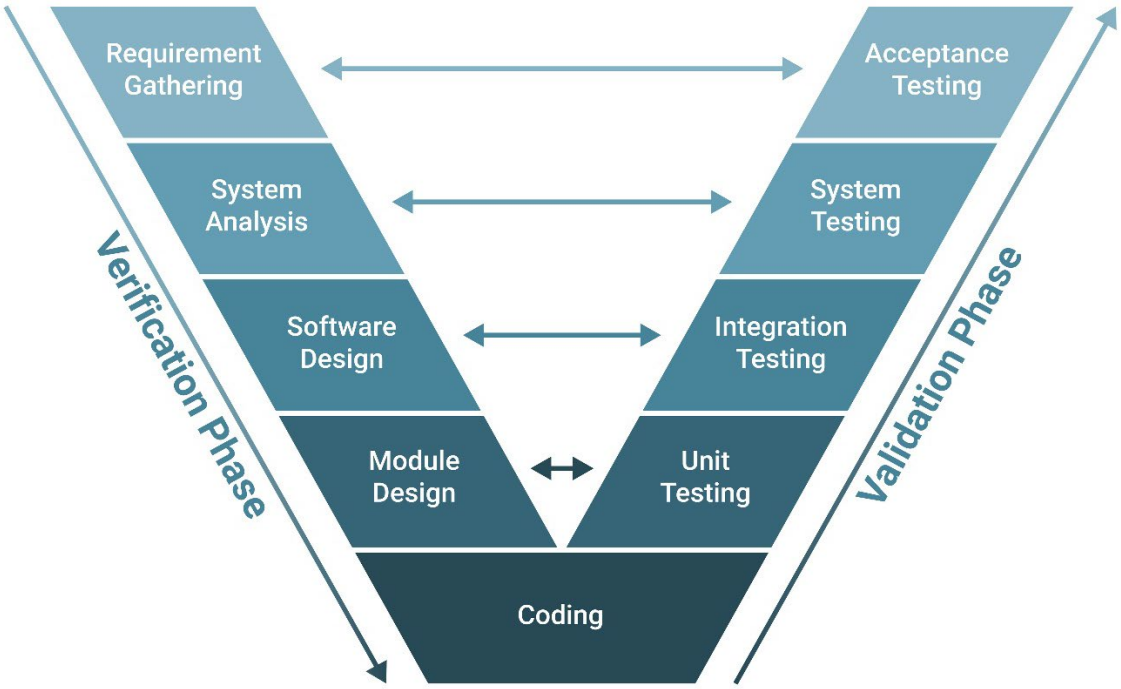
Este tema V&V é um assunto importante dentro do *pilar processos de gerenciamento de qualidade de software* e estão intimamente relacionadas com testes de *software*. Os vocábulos verificação e validação a princípio parecem que dizem respeito à mesma coisa, porém, dentro da garantia da qualidade de software nos trazem finalidades distintas. 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). Já 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 6 demonstra o modelo V na relação entre a verificação e a validação.

A fase de verificação aborda as etapas de requisitos, análise, projetos e início da implementação do código, ja a fase da validação inicia ainda na fase de implementação, testes unitários, testes de integração, testes de sistema e testes de aceitação.

De uma forma mais simples, podemos fazer as seguintes perguntas para distinguirmos verificação e validação:

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

Figura 6 – V&V – Relação entre *verificação* e *validação* no contexto de desenvolvimento e testes



Crédito: Kostiantyn/Adobestock

Dentro da garantia 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. De forma mais palpável e prática, existem alguns exemplos que se encaixam na V&V:

1. ***Code review*** (estilo verificação), identifica se o desenvolvimento seguiu boas práticas, convenções estabelecidas pela empresa. Esse tipo de análise é considerado estática, 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 TDD (*Test Driven Development*), esse 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 observam 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.

A verificação pode ser pensada seguindo alguns questionamentos, tais como:

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 a 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, podemos questionar:

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 todas as fases do ciclo de vida de software, 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.

A verificação e a validação são atividades principais dentro da garantia da qualidade, pois estabelece o que podemos fazer em cada uma das fases do desenvolvimento de *software*.

FINALIZANDO

A garantia da qualidade de *software* pode nos parecer um processo complexo e inatingível, porém o que precisamos compreender é que o mínimo que se faça dentro do ciclo de vida do projeto de *software* já é o início de um longo caminho a ser percorrido para implantação de validações, verificações, auditorias, inspeções e testes.

Um bom gerenciamento da qualidade de *software* preocupa-se em garantir que o número de erros caia sensivelmente e que a carga de trabalho em manutenções corretivas também tenha um declínio considerável. Antevermos problemas ao longo das etapas de requisitos, análise e projetos pode diminuir em muito os problemas que possam ocorrer nas fases finais do ciclo de vida.

Padrões e boas práticas são importantes nas fases de projeto e implementação, pois trazem consigo a construção de um código mais limpo e direcionado às funcionalidades reais.

Se a garantia de qualidade é o primo pobre do desenvolvimento de software, a validação tem a mesma relação com V&V. Embora as práticas de verificação, como testes, tenham um lugar muito importante na academia e na indústria, não podemos dizer o mesmo para as técnicas de validação. As técnicas de validação geralmente estão ausentes ou ignoradas pelos desenvolvedores e pelos processos de desenvolvimento obrigatórios. Algumas organizações validam os requisitos no início de um projeto. Infelizmente, eles realizarão alguma validação apenas no final. Ocasionalmente, encontramos

práticas de validação incorporadas em diferentes estágios do ciclo de desenvolvimento.

Medições e elaboração de métricas são importantes para gerirmos a garantia da qualidade, pois aquilo que não é medido não pode ser gerido. Por vezes, por questões de falta de mão de obra, ou até mesmo por questões de custos, empresas deixam a qualidade de *software* como segundo plano, porém, o mínimo sempre é mais. Mesmo que sejam pequenos testes unitários e algumas verificações entre desenvolvedores, o importante é tentarmos pensar em resultados mais próximos das necessidades dos usuários e sem erros.

A trajetória de implantação de garantia da qualidade é longa e trabalhosa, porém a recompensa com o retorno através de um *software* simples, usável, funcional, escalável, entre tantas outras características é incrível.

REFERÊNCIAS

ACM – Association for Computing Machinery. Disponível em: <https://www.acm.org/>. Acesso em: 28 set. 2022.

IEEE. Disponível em: <https://www.ieee.org/>. Acesso em: 28 set. 2022.

IEEE XPLORE. 1012-2016 - IEEE Standard for System, Software, and Hardware Verification and Validation. **IEEE**, 2016. Disponível em: <https://ieeexplore.ieee.org/document/8055462>. Acesso em: 28 set. 2022.

ISO. ISO 8402:1994. Quality management and quality assurance — Vocabulary. **ISO**, 1994. Disponível em: <https://www.iso.org/standard/20115.html>. Acesso em: 28 set. 2022.

_____. ISO/IEC/IEEE 24765:2017. Systems and software engineering — Vocabulary. **ISO**, 2017. Disponívle em: <https://www.iso.org/standard/71952.html>. Acesso em: 28 set. 2022.

_____. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. **ISO**, 2011.

ISTQB. Certified Tester Foundation Level (CTFL). **ISTQB**, S.d. Disponível em: <https://www.istqb.org/certifications/certified-tester-foundation-level>. Acesso em: 28 set. 2022.

JURAN, J.M.; DEFEO, J. A. **Fundamentos da qualidade para líderes**. Bookman: Porto Alegre, 2015.

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

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

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

SWEBOK – Software Engineering Body of Knowledge. Disponível em: <https://www.computer.org/education/bodies-of-knowledge/software-engineering>. Acesso em: 28 set. 2022.

THE ORIGINS of Computer Science. **IBM100**, S.d. Disponível em: <https://www.ibm.com/ibm/history/ibm100/us/en/icons/compsci/>. Acesso em: 28 set. 2022.