

ultima.

Porque a qualidade é importante

"O teste de software é a arte de realizar uma simulação em um software com o objetivo de encontrar defeitos." MYERS, G. J. *The Arts of Software Testing*



A qualidade não deve ser negligenciada por motivo de pressa. Qualquer descuido com a qualidade do produto pode resultar na desconfiança dos clientes, na redução das margens de lucro e, por fim, na incapacidade da empresa em atrair funcionários talentosos. Bons procedimentos de garantia de qualidade permitem que os proprietários e os funcionários se orgulhem de seu trabalho.

Reduz
Custos

Diminui
Riscos

Satisfação
do Cliente

Podemos notar um exemplo bem prático nos comentários abaixo:

The image displays three separate user reviews, each enclosed in a light gray box with a thin black border. Each review includes the user's name, a star rating, the date of the comment, and the text of their message. To the right of each message are icons for a thumbs-up (liking) and three dots (more options). The reviews are as follows:

- Vagner** (5 stars, 30 de dezembro de 2019): Cadastrei e não consegui entrar
- Clemilton** (5 stars, 26 de dezembro de 2019): Não estou conseguindo validar o acesso no email. O email de validação não chega, já olhei no spam e nada.
- Wellington** (5 stars, 23 de dezembro de 2019): Foi decepcionante. Baixar o app, fazer um cadastro confuso

Sabemos que a todo momento são lançados novos sistemas de aplicativos móveis e que existe um grande investimento em torno de sua concepção e criação. Porém, sem os testes apropriados para esses sistemas, como mostra o caso acima, os clientes ou usuários podem ter problemas para realizar funções básicas e isso afeta o projeto ou a empresa de forma negativa.

Aqui citamos algumas razões pelas quais os testes de software são muito importantes:

- 1. Ajudam a economizar dinheiro:** a relação custo-benefício do projeto é uma das principais razões pelas quais as empresas optam por serviços de teste de software;
- 2. Segurança:** é considerada a razão pela qual as pessoas procuram produtos bem testados e confiáveis;
- 3. Qualidade do produto:** para garantir que um produto específico ganhe vida, ele deve funcionar de forma completa e proporcionar uma experiência eficaz para o cliente;

- 4. Satisfação do cliente:** o principal objetivo de quem cria softwares é oferecer a melhor experiência para seus usuários. O mercado está bastante saturado atualmente e, pensando nisso, a primeira impressão é realmente importante. Caso não tenham uma experiência positiva, os usuários encontrarão outro produto que atenda às suas necessidades;
- 5. Aprimorar o processo de desenvolvimento:** os testadores de software devem trabalhar paralelamente com a equipe de desenvolvimento, o que é útil na aceleração do processo de desenvolvimento;
- 6. Facilidade para adicionar novos recursos:** os testes neutralizam a tendência de calcificação nos softwares, permitindo que os desenvolvedores adicionem novos recursos com mais facilidade;
- 7. Determinar o desempenho do software:** caso você introduza um software no mercado sem testá-lo e, após isso, seu desempenho não atender às expectativas ou requisitos dos clientes, fica mais difícil convencer as pessoas de que seu software é de qualidade.

Verdadeiro ou falso:

A garantia de qualidade é especialmente útil para pequenas empresas, pois permite que elas reduzam os custos extras de retestagem e de substituição e revenda de produtos defeituosos. Se os clientes não estão satisfeitos com determinados produtos, sua reação pode prejudicar a reputação da empresa, afetar negativamente futuros lançamentos e até mesmo afetar o negócio como um todo. A análise de testes também pode ajudar a empresa a reduzir despesas jurídicas, especialmente se o produto não estiver de acordo com os padrões do setor.

Opções de múltipla escolha

Sua resposta está correta!

- A.
- **Verdadeiro**
- B.
- Falso

Análise de testes de software



A análise de testes é um procedimento para garantir a qualidade dos produtos ou serviços de software fornecidos por uma organização a seus clientes. A garantia de qualidade se concentra em melhorar o processo de desenvolvimento do software até torná-lo o mais eficaz possível, de acordo com os padrões de qualidade definidos pelo setor.

Ela ajuda as empresas a atender às demandas e expectativas dos consumidores. Um produto ou serviço de alta qualidade gera confiança nos

clientes, o que, por sua vez, torna a empresa mais competitiva no mercado. Além de possibilitar redução de custos, a análise de testes corrige problemas iniciais e ajuda a definir e manter os padrões de qualidade. Investir em garantia de qualidade é indispensável para muitas indústrias hoje em dia. A análise de testes é mais eficaz quando está presente desde o início e, quando é bem feita, dá segurança e confiabilidade aos produtos que a empresa comercializa.

Podemos citar aqui alguns passos essenciais que envolvem a análise de testes de software:

Identifique os objetivos organizacionais

O papel da garantia de qualidade começa com um estudo sobre a missão, valor e a visão da empresa (ou projeto); como se dão as relações de trabalho nesse contexto e como será a atuação do testador. Funcionários e metas organizacionais são importantes no processo de garantia de qualidade.

Identifique os fatores de sucesso

O analista de testes precisa identificar quais são os fatores que melhoram o sistema de verificação de qualidade em uma empresa. Os fatores incluem produto com qualidade superior, processo de produção bem projetado, segurança financeira, suporte ao cliente, satisfação dos funcionários e/ou suporte técnico. Liste os principais fatores de garantia de qualidade para gerenciar os sistemas de forma recursiva e regular.

Feedback do cliente

No processo de garantia de qualidade, o feedback é o mais importante. O feedback contínuo dos clientes ajuda a detectar os problemas que surgem no controle de qualidade (inclusive antes mesmo que se tornem um problema mais sério).

Implemente melhorias contínuas

Os resultados obtidos nas pesquisas ou feedback dos clientes são usados continuamente para fazer mudanças nos produtos. Quando a empresa acompanha o feedback dos clientes, os produtos atingem números maiores no mercado.

Selecione o software de gerenciamento de qualidade

Ele ajuda a implementar um processo de garantia de qualidade melhor, mais detalhado e organizado.

Meça resultados

O principal objetivo da equipe de testes é que o produto satisfaça o cliente, e, para isso, é muito importante encontrar formas de analisar os resultados encontrados e trabalhar nas melhorias que podem ser feitas.

Quando a análise de testes deve começar?

Sua resposta está correta!

- A.
- O mais cedo possível
- B.
- Assim que o produto estiver pronto para lançamento

A diferença entre Garantia de Qualidade (Quality Assurance ou QA), Controle de Qualidade (Quality Control ou QC) e Testes



Frequentemente usados de forma intercambiável, os três termos referem-se a aspectos ligeiramente diferentes da gestão de qualidade de software. Apesar de terem o objetivo comum de entregar um produto com a melhor qualidade possível, tanto estrutural quanto funcionalmente, eles utilizam abordagens diferentes para essa tarefa.



Garantia da qualidade é um termo amplo, descrito no Google Testing Blog como “a melhoria contínua e consistente, e a manutenção do processo que permite o trabalho de controle de qualidade”. Segundo essa definição, a garantia da qualidade se concentra mais nos aspectos organizacionais da gestão da qualidade, monitorando a consistência do processo de produção.

Através do **controle de qualidade**, a equipe verifica a conformidade do produto com os requisitos funcionais. Conforme definido pela Investopedia, controle de qualidade é um processo através do qual uma empresa procura garantir que a qualidade do produto seja mantida ou aprimorada e os erros de fabricação sejam reduzidos ou eliminados. Esta atividade é aplicada ao produto finalizado, e é realizada antes de seu lançamento. É semelhante a analisar um item aleatório de uma linha de montagem para ver se ele está de acordo com as especificações técnicas da indústria.

O **teste** é a atividade básica destinada a detectar e resolver problemas técnicos no código-fonte do software e avaliar a usabilidade geral do produto, seu desempenho, segurança e compatibilidade. Tem foco muito específico e é realizado pelos engenheiros de teste paralelamente ao processo de desenvolvimento ou na fase de testes (dependendo da abordagem metodológica do ciclo de desenvolvimento de software).

Veja na tabela abaixo a comparação entre os conceitos de garantia da qualidade, controle de qualidade e testes:

	Garantia de qualidade	Controle de qualidade	Testes
Objetivo	Estabelecer processos adequados, introduzindo padrões de qualidade para prevenir erros e falhas no produto	Certificar-se de que o produto corresponde aos requisitos e especificações antes de ser lançado	Detectar e resolver erros e falhas
Foco	Processos	Produtos como um todo	Código fonte e design
O quê	Prevenção	Verificação	Detecção
Quem	A equipe, incluindo as partes interessadas (stakeholders)	A equipe	Analistas de testes, desenvolvedores

Quando	Ao longo do processo	Antes do lançamento	Na fase de teste ou junto com o processo de desenvolvimento
---------------	----------------------	---------------------	---

Se aplicado ao processo de fabricação de automóveis, ter métodos de **garantia da qualidade** adequados significa que cada membro da equipe entende os requisitos e realiza seu trabalho de acordo com as diretrizes comumente aceitas. Ou seja, esses métodos fazem com que cada ação seja executada na ordem certa, todos os detalhes sejam implementados adequadamente e os processos gerais sejam consistentes, para que não haja nenhum impacto negativo no produto final.

O **controle de qualidade** pode ser comparado a entrar em uma montadora de automóveis e escolher um carro aleatório para uma avaliação e um test drive. As atividades de teste, neste caso, referem-se ao processo de verificação de cada junta, de cada mecanismo separadamente, bem como de todo o produto, seja manual ou automaticamente, realizando testes de colisão, testes de desempenho e test drives reais ou simulados.

Devido à sua abordagem prática, as atividades de **teste de software** continuam sendo um assunto polêmico. Mas antes de entrarmos em detalhes, vamos definir os princípios básicos de teste de software.

Escolha a alternativa correta:

- 1. Garantir que um produto ou serviço atenda aos padrões mínimos do mercado, muitas vezes testando amostras de produtos já finalizados.**

Sua resposta está correta!

- A.

- Controle de qualidade
- B.
- Garantia da qualidade

2. Garantir que a qualidade seja entregue em todas as etapas do processo de produção, muitas vezes tornando-a uma responsabilidade de cada pessoa que faz parte desse processo.

Sua resposta está correta!

- A.
- Controle de qualidade
- B.
- Garantia da qualidade

O que é Garantia de Qualidade



Garantia de Qualidade (ou Quality Assurance) é um sistema que demonstra à liderança da empresa e ao público que um produto ou serviço está de acordo com os padrões estabelecidos pela indústria ou por órgãos reguladores. A garantia de qualidade é às vezes confundida com o controle de qualidade, embora este último seja mais relacionado ao produto pós-fabricação.

Analistas de garantia de qualidade — também conhecidos como analistas de localização e analistas de teste — testam programas, jogos e softwares para que sejam confiáveis, totalmente funcionais e fáceis de usar, antes de serem lançados para o público. Eles utilizam um plano de testes para inspecionar milhares de linhas de código, para que estejam totalmente livres de erros. Procuram falhas e pontos fracos no programa — como, por exemplo, uma interface pouco atraente — e relatam esses defeitos aos desenvolvedores. Podem também corrigir quaisquer problemas ou falhas do sistema e fazer sugestões de melhorias no funcionamento do software. Outra de suas funções é fazer com que o software seja adequado para o mercado no qual será lançado, traduzindo (localizando) expressões idiomáticas e realizando adaptações culturais.

Encontrar
Defeitos

Aumentar
Confiabilidade

Melhorar
Qualidade

Historicamente, o significado de qualidade de produto ou serviço teve várias interpretações: desde uma abordagem subjetiva baseada no usuário e contendo "os diferentes pesos que os indivíduos atribuem às características de qualidade"; até uma metodologia baseada em valor, pela qual os consumidores relacionam a qualidade ao preço e avaliam o produto com base nessa relação.

Essa disciplina tornou-se comum na indústria na década de 1920, como forma de definir e controlar a qualidade dos produtos. Na década de 1950, a garantia de qualidade tornou-se importante também para a saúde e a segurança públicas.

Hoje, são procedimentos que verificam os padrões de qualidade de um produto ou serviço antes que este seja lançado para o público, e também avaliam os métodos usados na fabricação de produtos e na prestação de serviços.

Quais são os deveres típicos de um analista de testes?

Sua resposta está correta!

- A.
- criar um plano de teste
- B.
- usar o plano para avaliar a funcionalidade, desempenho, confiabilidade, estabilidade e compatibilidade com outros sistemas
- C.
- usar o plano para procurar e corrigir bugs de software
- D.
- esperar o produto ser lançado para corrigir bugs

- E.
- procurar maneiras de evitar que os bugs ocorram

A carreira e os desafios do analista de testes



O analista de garantia de qualidade testa o software para assegurar que ele funcione de maneira adequada e eficiente, e faz sugestões de melhorias em seu desempenho — isso vale tanto para softwares novos como para os que precisam ser atualizados. A principal responsabilidade desse profissional é aprimorar os produtos e aplicativos, mantendo os prazos e orçamentos já definidos.

Os analistas geralmente trabalham no ambiente de teste em conjunto com a equipe de melhoria de qualidade, permitindo que a análise ocorra antes que o software seja lançado ao público. Quando um problema é identificado, o motivo de sua ocorrência deve ser entendido para que possa ser corrigido o mais rápido possível e evitado no futuro. É feito um relatório das novas alterações,

detalhando o programa, a avaliação, os métodos de teste e as melhorias realizadas no software.

Esses profissionais utilizam várias ferramentas para comparar o software com os requisitos dos usuários e garantir que ele funcione de acordo com esses requisitos. Após o teste, as alterações e melhorias do software são sugeridas aos desenvolvedores para que estes façam as mudanças apropriadas.

Como o trabalho de um analista de testes envolve grande conhecimento sobre o software desenvolvido, são também frequentemente chamados para criar materiais de treinamento e instruir os usuários quanto à sua correta utilização.

Quais são as principais habilidades de um analista de testes?

Sua resposta está correta!

- A.
- Conhecimento de uma ampla gama de aplicativos de software e de hardware e redes
- B.
- Uma boa compreensão de negócio
- C.
- Uma mentalidade criativa — capacidade de abordar um problema de forma criativa
- D.
- Boa capacidade de comunicação, tanto escrita como verbal

Material complementar

Principais Tipos de Teste de Software

Os testes de software são uma maneira de garantir a qualidade de um software. Com isso existem diversas maneiras de se testar um sistema, que podem variar de acordo com a sua natureza ou seu objetivo. Podemos classificar em:

- **Testes estruturais ou caixa-branca (*white-box testing*)**
- **Testes funcionais ou caixa-preta (*black-box testing*)**
- **Testes não-funcionais**
- **Testes de relacionados à mudança**



Os **testes estruturais ou caixa-branca** são realizados diretamente no código e geralmente são feitos pelo desenvolvedor do sistema nos níveis de: testes de unidade e testes de integração. No **teste de unidade**, cada função do sistema é testada a fim de garantir que elas funcionam independente da interação com as outras partes do sistema. Já o **teste de integração** tem o objetivo de verificar se cada parte, ao ser integrada com outras, funcionam corretamente.

Por outro lado, os **testes funcionais ou caixa-preta** geralmente são feitos pelos testador, cliente ou usuários, nos níveis de teste de sistema e teste de aceitação. Pois eles não têm contato direto com o código-fonte do sistema. Entende-se o sistema como uma caixa, onde ao inserir valores de entrada, retorna valores de saída. Em cada um desses níveis de teste, os testes funcionais mudam de objetivo.

Os **testes de sistema** são realizados por uma equipe específica de teste, que utiliza a especificação dada pelo cliente para fazer o roteiro de casos de teste. Já o **teste de aceitação** difere do teste de sistema, pois pode ser executado pelo cliente ou pelos usuários, cuja finalidade do teste é verificar se o sistema está de acordo com o que foi solicitado e se atende as necessidades dos usuários.

Os **testes não-funcionais** têm esse nome por se tratar de testes que verificam aspectos gerais da aplicação, independente das regras de negócio que há nele. Nesse caso, pode-se realizar **testes de desempenho**, **testes de segurança**, **testes de usabilidade**, entre outros.

Já os **testes de relacionados à mudança** são realizados quando o sistema sofrem alterações consideráveis, que podem gerar bugs. Geralmente é necessário re-executar o roteiro de teste criado para o teste funcional, nesse caso chamamos de **teste de regressão**. Para uma melhor eficiência nesse tipo de teste, uma estratégia seria automatizar para reduzir o custo de re-execução e conseguir realizar uma verificação mais rápida.

Uma ferramenta que pode ser usada para testes de regressão funcional é o **Selenium WebDriver**. Contudo, eles também podem ser feitos com técnicas estruturais (caixa-branca), pois qualquer execução de teste que valide se o sistema continua funcionando é um teste de regressão.

Uma métrica importante, que deve ser levada em consideração, é a **cobertura dos testes**, que tem a finalidade de verificar qual o percentual do sistema está sendo testada. No caso dos testes estruturais, seria analisado a cobertura de testes do código-fonte e no teste funcional, a métrica seria com relação à **cobertura de requisitos**. Há ferramentas que auxiliam na verificação dessa métrica, como por exemplo:

SonarQube:

1. OpenClover;
 2. IntelliJ IDEA;
 3. EclEmma.

Referência:

Referência: WILLIAMS, M., SUCCI, G. e MARCHESI, L. *Traditional and Agile Software Engineering*. Ch 8 — Black Box Testing. Ed. Addison-Wesley, 2003 Syllabus Foundation Level — BSTQB — https://www bstqb.org.br/uploads/syllabus/syllabus_ctfl_2018br.pdf

A vida de um Analista de Qualidade (QA) em um time de Alta Performance

Quality Assurance (QA) — Define-se como um conjunto de atividades para garantir a qualidade nos processos de desenvolvimento. Ser um analista QA, ao contrário do que muitos imaginam, não é apenas “testar”. Há uma diferença sutil, mas muito importante, entre o “Analista de Teste” e o “Analista de Qualidade”:

Analista de teste seria aquele envolvido especificamente na área de testes de software elaborando casos de testes e executando. Já o analista de qualidade seria aquela pessoa envolvida no trabalho que define um processo de desenvolvimento de software e depois certifica esse processo.

O trecho acima cita a diferença básica entre os dois cargos. Na [FCamara](#), por exemplo, há uma cultura de aprendizado diferenciada. Os gestores e líderes dos projetos sempre incentivam aos desenvolvedores e analistas a participarem de todo o processo e a entenderem o negócio, não só a parte técnica do produto, mesmo se a participação do indivíduo se inicie com o projeto já em andamento.

Para resumir bem as definições dos dois cargos:

Analista de Teste: Valida o produto

Analista de Qualidade: Valida o processo.

O que é preciso para ser um analista QA?

Além de conhecimentos técnicos sobre todo o processo de desenvolvimento de um software, tipos diferentes de testes e

ferramentas e metodologias de automação de testes, validação de processos, etc — falarei sobre estas em outro artigo — , o analista QA precisa ter características pessoais que são diferenciais para o sucesso de sua participação nesses projetos, como:

Ser detalhista. O analista QA precisa conhecer e entender o funcionamento de cada funcionalidade do sistema, por menor que ela seja. O entendimento do negócio desde a fase de idealização do projeto é crucial para o sucesso do seu trabalho. Quanto mais detalhista ele for, maior a chance de encontrar problemas ou até mesmo fazer sugestões de melhorias, tanto na aplicação quanto no código.

Seria curioso. “O que será que acontece se eu fizer isso? ”. É imprescindível que o QA saiba que, provavelmente, o “caminho feliz” da aplicação estará funcionando corretamente. Isto é, se uma função da aplicação foi desenvolvida para o usuário pedir uma cerveja, para validar que essa funcionalidade se comporta como esperado, o QA precisa ser curioso o suficiente para testar o comportamento da aplicação ao receber “pedidos inesperados”.

Pensar como técnico e usuário. É claro que o analista QA possui conhecimentos que lhe dão uma certa “vantagem” na hora de interpretar comportamentos de uma aplicação. No entanto, a depender do objetivo dessa aplicação, todo tipo de usuário a utilizará. E nesse momento o analista QA precisa “pensar como o usuário” para interpretar se a aplicação está agindo corretamente também para usuários que não possuem conhecimentos técnicos porque, na maioria das vezes, os desenvolvedores vão pensar única e exclusivamente no funcionamento da aplicação, e não em como o usuário vai se comportar com ela.

Ser comunicativo. O analista QA não pode ser tímido, introvertido. A comunicação é essencial para conseguir os envolvidos entendam suas

sugestões/críticas, e além disso, é essencial também para o engajamento entre os membros da equipe com o projeto

Se atualizar. Na área de tecnologia, todos os dias surgem novas ferramentas, metodologias, aplicações, etc, desenvolvidas para aumentar a produtividade do trabalho de desenvolvedores e analistas. O QA precisa estar sempre pronto para aprender e se profissionalizar com essas mudanças. Quanto mais diversificado é o seu conhecimento com todo tipo de ferramenta e metodologia de teste/desenvolvimento, maior será sua valorização em seus times.

Como é o trabalho do QA nesse processo?

Para entender a participação do QA nesta metodologia, você precisa perceber que, por se tratar de entregas de resultados de curto prazo (normalmente duas semanas), o tempo para entendimento do negócio, validação de processos, execução de testes e demais atividades consequentemente também é muito menor. Por isso, a objetividade na realização destas e a comunicação entre desenvolvedor e analista QA é extremamente importante desde o início do ciclo de desenvolvimento.

Um dos desafios do QA está em exatamente conseguir aplicar seu conhecimento e experiência no pouco tempo que lhe é disponibilizado. No entanto, um desafio maior pode estar em se adaptar a metodologias como essa e a rotina dos desenvolvedores do time, sem deixar que a performance do seu trabalho seja prejudicada. É extremamente importante que o QA participe de todo o processo para entender o funcionamento do sistema, e não só encontre “erros”, mas sugira e participe nas decisões das resoluções destes, e é um diferencial que tenha conhecimentos técnicos na linguagem utilizada no projeto para isso.

Por isso, a minha visão sobre meu trabalho como QA mudou bastante desde que comecei a participar deste time. Comecei a perceber que, para conseguir

acompanhar os resultados de alta performance, preciso me manter atualizado sobre processos de desenvolvimento X testes, inclusive não só como QA, mas também como desenvolvedor.

Após me engajar nessa ideia, eu fui apresentado a algumas metodologias de desenvolvimento, como TDD (Test Driven Development) e BDD (Behavior Driven Development), que mudaram minha visão sobre todo o ciclo de vida de um software, aumentando consideravelmente o meu potencial de participação desde o início desse ciclo.

Conclusão

Cada vez mais são criados processos e ferramentas para viabilizar melhorias nas fases do ciclo de vida de um software. Processos de automatização de testes, como o TDD e o BDD, são bons exemplos dessas mudanças. Cabe ao analista QA utilizar destas ferramentas, unidas ao seu conhecimento e experiência, para descobrir como otimizar o seu trabalho, inclusive em ambientes em que o “tempo disponível” não é o ideal para a prática de testes, afinal, raramente vai ser.

Conseguimos citar neste texto, alguns dos grandes desafios que um analista QA enfrenta todos os dias, principalmente em equipes que trabalham de forma parecida com o modelo “High Performance Team”. Mas definitivamente, seu maior desafio ainda está em “matar um leão por dia” para conseguir continuar provando o valor e a importância do seu trabalho, aumentando seu espaço e criando oportunidades de melhorias para si e para os times em que possa participar. Pois ainda há — e talvez sempre haverá — algum “recuo” de clientes quando se trata de disponibilizar tempo, o mais próximo possível do ideal, para a execução de testes em um projeto.

Concluindo, deixarei alguns links abaixo para que vocês possam ter informações mais detalhadas sobre as metodologias e ferramentas citadas

nesse texto, inclusive, de outras fontes que não foram citadas, mas que acredito ser de grande valia para quem quer seguir a carreira de QA nos dias atuais.

Qual o papel do teste na revisão do código?

Você é alguém que testa aplicações e já escutou a seguinte frase: “O analista de qualidade não pode ver código pois “vicia” para executar o teste.”; ou é um desenvolvedor e concorda com essa afirmação?

Em todos estes anos na área de testes, ouvi essa frase algumas vezes partindo de devs, gerentes e até colegas da profissão.

O Analista de Qualidade trabalhando em um contexto ágil pode executar o teste com base na técnica de revisão de código, gerando benefícios para garantir a qualidade do código, e pode usar uma lista de verificação para guiar seus testes de revisão.

Técnica de Revisão de Código

Quando o QA não possui, por exemplo, acesso aos requisitos do software, ele pode utilizar várias técnicas e abordagens para iniciar os testes. Uma dessas técnicas é por revisão de código.

O próprio nome “revisão de código” deixa claro o objetivo da prática. A ideia é que o código escrito por um desenvolvedor, antes de ser promovido ao ambiente de produção, seja revisado por outro membro da equipe. O revisor anota todos os problemas encontrados e devolve ao autor original daquele código. O autor então avalia os comentários recebidos e eventualmente os propaga para o código-fonte.

Quais os benefícios?

- Avaliar um software antes mesmo de ter uma versão para testes permite antecipar inconsistências.
- Permite também ao testador dizer como melhorar a cobertura de testes escritos pelos desenvolvedores.
- Acontece disseminação de conhecimento entre os membros da equipe.
- Redução de bugs, falhas.
- Melhoria da qualidade interna.

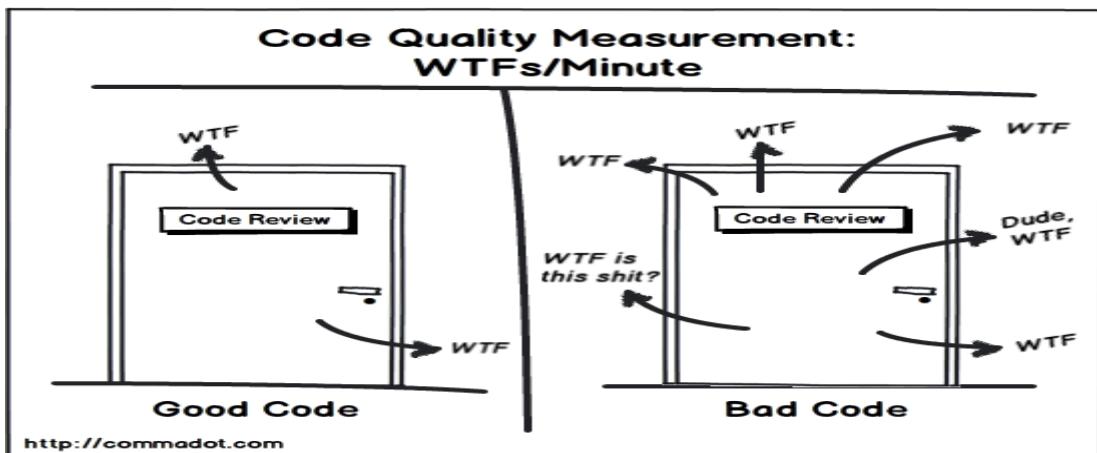
O revisor além de entender da parte técnica, é preciso que consiga ter uma perspectiva completa do que está revisando. É muito importante saber o objetivo do desenvolvedor com essa solução, seja ela a correção de um bug ou a entrega de um novo processo no sistema.

Não utilize apenas os casos de teste que o desenvolvedor criou, tente pensar em novos cenários para analisar a cobertura de testes. Nesse momento, podem surgir novas ideias que não haviam sido consideradas em tempo de desenvolvimento.

Ah, usar uma lista de verificação auxilia na revisão de código.

Exemplo do check-list revisão de código fonte

- O código obedece às convenções acordadas pelo time?
- Há código morto?
- Há algum comentário desnecessário?
- Todos os laços têm um final alcançável?
- Os testes contidos testam o que se propõe a testar?



QA, ainda está em dúvida se deve revisar o código?

Alguns exemplos reais:

1. Em uma revisão de código foi percebido que os parâmetros da função não estavam sendo testados para null, logo caso o serviço rest retornasse algum valor nulo, ocorreria um erro em tempo de execução.

Talvez esse erro nunca ocorresse, mas como estava consumindo um serviço de terceiros, era melhor fazer a verificação para que o aplicativo não desse crash.

2. Em outro cenário, o programador teria que codificar para exibir uma determinada mensagem que provinha de um arquivo de configurações, quando o tester abriu o PR já percebeu que a mensagem estava errada. Para validar essa mensagem depois da aplicação pronta, gastaria aproximadamente 30 minutos para criar massa de dados; analisando o código, antecipou a inconsistência.
3. Mais um exemplo, quando você revisa os testes unitários, analise se tem a devida cobertura de testes para aquele método.
Concluímos que é aceitável e possível o analista de teste revisar o código fonte, a técnica não serve só para melhorar a qualidade do código, mas também para disseminar conhecimento entre os membros da equipe, antecipar as inconsistências no ambiente de contexto iterativo, e existem ferramentas para auxiliar a revisão do código fonte, como o check list de verificação.

O que é Garantia de Qualidade



Ao comprar uma maçã, você avalia instantaneamente sua qualidade: o tamanho, a forma, a maturação e a ausência de manchas. Mas só quando você

der a primeira mordida é que será capaz de sentir se a maçã é realmente boa. Mesmo uma maçã extremamente bonita pode ter um gosto azedo ou uma larva dentro dela.

O mesmo se aplica a quase todos os produtos, seja um objeto físico ou um software. Um site pode parecer bom no início, mas à medida em que você continua a navegar por ele, é direcionado(a) a outra página ou o site pode começar a mostrar algumas falhas e erros de código ou design.

Isso torna a garantia de qualidade muito importante em todas as áreas nas quais são criados produtos para o usuário final. No entanto, uma maçã azeda não causará tantos danos quanto um carro autônomo com software de piloto automático de baixa qualidade. Um único erro pode colocar a vida de uma pessoa em risco. Da mesma forma, um site de comércio eletrônico com problemas de desempenho pode causar um enorme prejuízo ao proprietário.

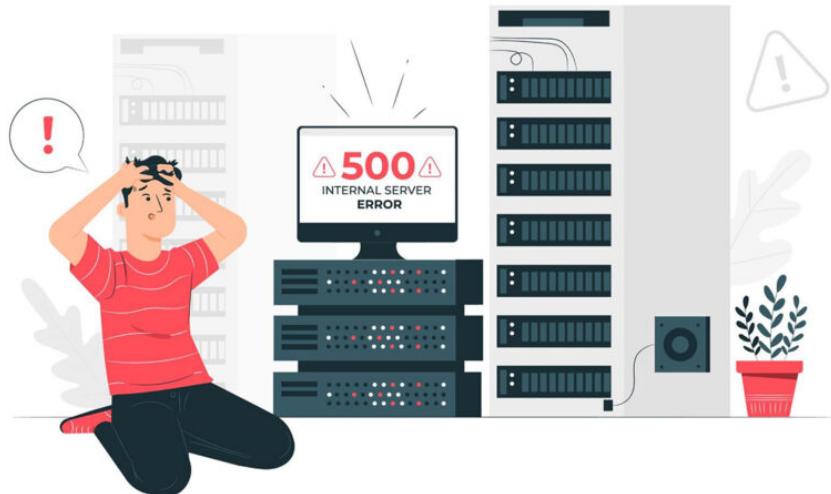
Verdadeiro ou falso:

“Garantia de Qualidade é parte da gestão focada em prover confiança de que os requisitos da qualidade serão atendidos. Garantia de qualidade está relacionada aos processos. Seu objetivo é garantir e verificar se todos os procedimentos estão de acordo com o sistema de gestão da qualidade. Uma das ferramentas para isso é o controle da qualidade.” (Fonte: ISO 9000)

Você respondeu a esta pergunta corretamente em sua primeira tentativa.

- A.
- Verdadeiro
Você selecionou esta opção inicialmente.
- B.
- Falso

Introdução ao conceito de qualidade



Embora errar seja humano, às vezes o custo de um erro pode ser muito alto. Ao longo da história tivemos muitos exemplos de situações em que falhas de software causaram prejuízos enormes — como cafeterias Starbucks sendo forçadas a dar bebidas grátis por causa de um mau funcionamento nas caixas registradoras; e até mesmo erros que custaram a vida de pessoas — como um avião militar F-35 com falhas em seu radar.

O conceito de qualidade de software foi introduzido para que um programa a ser lançado seja seguro e funcione conforme o esperado. É frequentemente definido como "o grau de conformidade a requisitos e expectativas explícitos ou implícitos". Essas chamadas expectativas explícitas e implícitas correspondem a dois níveis básicos de qualidade de software:

O conceito de qualidade de software foi introduzido para que um programa a ser lançado seja seguro e funcione conforme o esperado. É frequentemente definido como "o grau de conformidade a requisitos e expectativas explícitos ou implícitos". Essas chamadas expectativas explícitas e implícitas correspondem a dois níveis básicos de qualidade de software:

Funcional — a conformidade do produto com os requisitos funcionais (explícitos) e especificações de projeto. Este aspecto foca no uso prático do

software do ponto de vista do usuário: suas características, desempenho, facilidade de uso, ausência de defeitos.

Não funcional — características internas e arquitetura do sistema, ou seja, requisitos estruturais (implícitos). Isso inclui a capacidade de manutenção, compreensão, eficiência e segurança do código.

Combine os exemplos abaixo com seu respectivo tipo de requisito:

1. Consulta de saldo ou estoque

- A.
- Requisito funcional
- B.
- Requisito não funcional

2. O sistema deve ser implementado na linguagem Java

- A.
- Requisito funcional
- B.
- Requisito não funcional

3. O sistema deverá se comunicar com o banco SQL Server.

- A.

- **Requisito funcional**

- B.

- **Requisito não funcional**

4. Efetuar pagamentos de compra através de crédito ou débito

- A.

- **Requisito funcional**

- B.

- **Requisito não funcional**

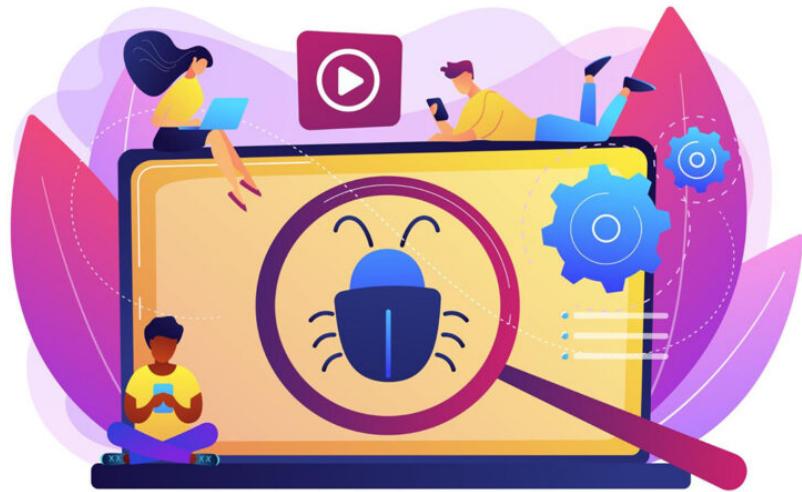
5. Consulta e alterações de dados pessoais

- **Requisito funcional**

- B.

- **Requisito não funcional**

Princípios básicos de teste de software



Formulados ao longo dos últimos 40 anos, os sete princípios de teste de software representam as regras básicas para o processo:

1. O teste mostra a presença de defeitos e não sua ausência

O teste visa detectar os defeitos em um software. No entanto, não importa o quanto o produto seja testado, nunca teremos absoluta certeza de que não possui defeitos. Só podemos usar testes para reduzir o número de problemas não encontrados.

2. Testes exaustivos são impossíveis

Não há como testar todas as combinações de entradas de dados, cenários e pré-condições em um aplicativo. Por exemplo, se uma única tela de aplicativo contiver 5 campos de entrada com 3 opções de valor possíveis cada, seu teste seria cobrir todas as combinações possíveis e criar centenas de cenários. E se o aplicativo contiver mais de 50 dessas telas? Para não perder semanas criando milhões de cenários pouco prováveis, é melhor se concentrar naqueles potencialmente mais significativos.

3. Testes iniciais economizam tempo e dinheiro

O custo de um erro cresce exponencialmente ao longo das etapas do ciclo de vida do desenvolvimento de software. Portanto, é importante começar a testar o software o quanto antes, para que os problemas detectados sejam resolvidos e não se transformem em uma bola de neve.

4. Defeitos se agrupam

Este princípio é muitas vezes referido como uma aplicação do princípio de Pareto ao teste de software. Isso significa que aproximadamente 80% de todos os erros geralmente são encontrados em apenas 20% dos módulos do sistema. Portanto, se um defeito for encontrado em um módulo específico de um programa de software, é provável que haja outros defeitos neste mesmo módulo.

5. Cuidado com o paradoxo do pesticida

Executar o mesmo conjunto de testes repetidamente não ajudará a encontrar mais problemas. Assim que os erros detectados são corrigidos, estes cenários de teste se tornam inúteis. Portanto, é importante revisar e atualizar os testes regularmente para adaptá-los e aumentar a probabilidade de encontrar mais erros.

6. O teste depende do contexto

Dependendo de sua finalidade ou setor, diferentes aplicativos devem ser testados de formas diferentes. Embora a segurança possa ser de importância primordial para um produto *fintech*, ela é menos importante para um site corporativo. Este último, por sua vez, privilegia a usabilidade e a velocidade.

7. A ausência de erros é uma ilusão

A completa ausência de erros em seu produto não significa necessariamente seu sucesso. Não importa quanto tempo você tenha gasto aprimorando seu código ou suas funcionalidades, se seu produto não for útil ou não atender às expectativas do usuário, ele não será adotado pelo público-alvo.

Embora os princípios listados acima sejam diretrizes indiscutíveis para todos os profissionais de teste de software, há mais aspectos a serem considerados.

Algumas fontes citam outros princípios além dos básicos:

- O teste deve ser um processo independente e realizado por profissionais imparciais;
- Os testes devem ser realizados tanto para os valores de entrada inválidos e inesperados, quanto para os válidos e esperados;
- Todos os passos do processo de teste devem ser registrados de forma abrangente para que se possa definir e verificar os resultados esperados.

Verdadeiro ou falso:

Ter zero defeitos não significa que o software resolva com sucesso os problemas do usuário final. O Linux sempre teve muito poucos bugs, enquanto o Microsoft Windows era (é?) notório por suas falhas. No entanto, a maioria das pessoas usava o Microsoft Windows como sistema operacional porque o achava mais fácil de usar e resolia seus problemas. O Linux está se tornando cada vez mais popular hoje, pois começou a se concentrar na experiência do usuário final.

- A.
- **Verdadeiro**
- B.
- Falso

O papel dos testes no ciclo de vida do desenvolvimento de software



De acordo com o ISTQB® (*International Software Testing Qualifications Board*), uma parte importante do papel do testador é estar familiarizado com os modelos de ciclo de vida de desenvolvimento de software mais comuns, para que as atividades de teste apropriadas possam ocorrer.

Em qualquer modelo de ciclo de vida de desenvolvimento de software, existem várias características para um bom teste:

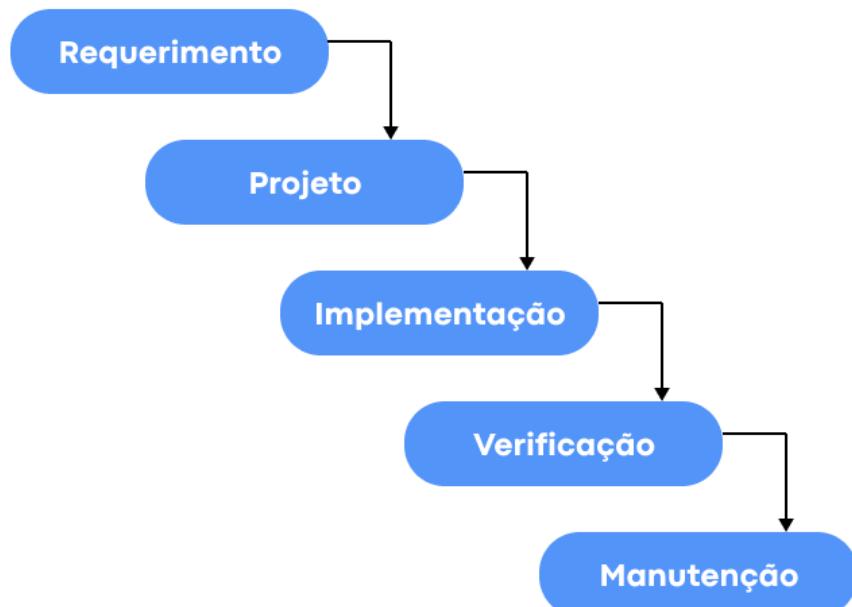
- Para cada atividade de desenvolvimento, existe uma atividade de teste correspondente;
- Cada nível de teste tem objetivos específicos;
- A análise e a modelagem para um determinado nível de teste começam durante a atividade de desenvolvimento correspondente;
- Os testadores participam de discussões para definir e refinar os requisitos e a modelagem, e estão envolvidos na revisão dos produtos de trabalho (por ex: requisitos, modelagem, histórias de usuários etc.) assim que os esboços estiverem disponíveis.

Vamos conferir alguns desses modelos:

Modelo em cascata (*Waterfall*)

O modelo em cascata (ou *Waterfall*) representa um ciclo de vida de desenvolvimento de software tradicional e inclui 5 fases: requerimento, projeto, implementação, verificação e manutenção.

Este modelo determina que uma fase só é iniciada quando a anterior for finalizada. Neste contexto, antes da implementação de um software, as fases de requerimento e projeto precisam estar concluídas para que o software seja desenvolvido.

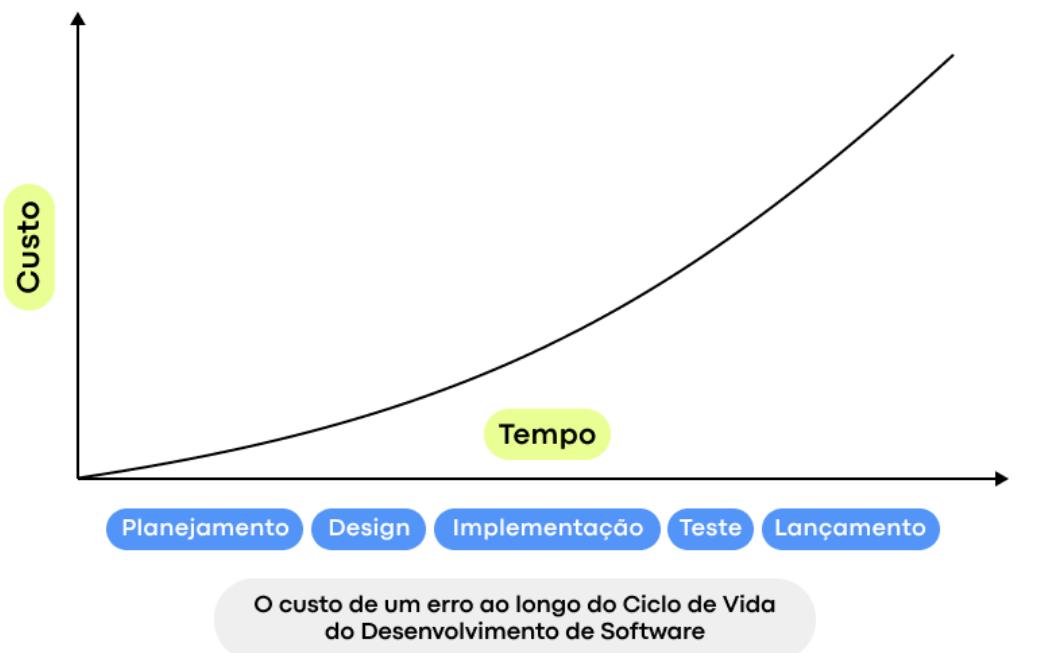


última.

Neste modelo, o analista de testes consegue atuar nas fases de requerimento, com a revisão da documentação do sistema; e de projeto, com a revisão dos modelos de dados, diagramas de classes, entre outros. Já na fase de implementação, testes automatizados podem ser realizados pelos engenheiros

de software. O analista de testes cria ferramentas para testes manuais, como: cenários de teste, roteiros de teste, casos de teste, entre outros.

Na fase de verificação, o produto completo, já projetado e codificado, é exaustivamente testado antes de seu lançamento. No entanto, a prática mostra que erros e defeitos de software detectados nesta fase podem custar muito caro para serem corrigidos, pois o custo de um erro tende a aumentar ao longo do processo de desenvolvimento de software.



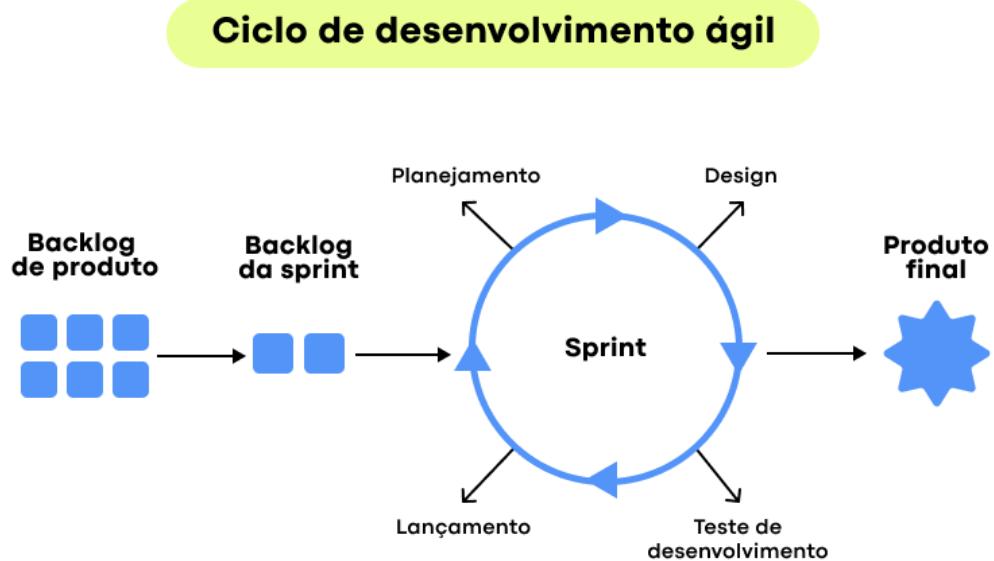
última.

Por exemplo, se houver um erro nas especificações, detectá-lo no início do planejamento não causaria prejuízos significativos ao seu negócio. No entanto, o dano cresce exponencialmente ao longo das etapas posteriores do processo. O mesmo acontece com os erros produzidos no processo de implementação. Se um recurso tem uma falha em sua lógica, construir mais funcionalidades em cima dele pode causar sérios danos a longo prazo. Portanto, é melhor testar todos os recursos enquanto o produto ainda está sendo construído. É aqui que os métodos ágeis iterativos se mostram benéficos.

A equipe de testes é composta, na maioria das vezes, por analistas de testes que têm como principal função elaborar casos de teste e também executá-los. Os membros da equipe têm funções definidas: os automatizadores são responsáveis por automatizar os testes, o líder de teste gerencia os analistas de teste e testadores, o analista de testes de performance é responsável pelos testes de performance, entre outros.

Metodologia ágil (Scrum)

Parte integrante do processo de desenvolvimento de software, a Ágil divide o processo de desenvolvimento em partes menores, iterações e *sprints*. Isto permite que os testadores trabalhem em paralelo com o restante da equipe durante todo o processo e que as falhas e erros sejam corrigidos imediatamente após sua ocorrência.



última.

O principal objetivo deste processo é entregar pequenas partes de um software de forma rápida e com a melhor qualidade possível. Portanto, essa abordagem é menos custosa: corrigir os erros no início do processo de desenvolvimento,

antes que mais problemas se acumulem, é significativamente mais barato e requer menos esforço.

As equipes Scrum são multidisciplinares, e seus membros utilizam boas práticas de qualidade que, dessa forma, os tornam responsáveis pelo desenvolvimento e entrega do produto. Além disso, a comunicação eficiente dentro da equipe e o envolvimento ativo das partes interessadas agilizam o processo e permitem decisões mais bem embasadas.

Na abordagem de teste ágil, o propósito é a construção de uma *prática* de controle de qualidade, em vez de somente contar com uma equipe de controle de qualidade. Em projetos ágeis, o analista de testes deve ser incorporado às equipes de desenvolvimento, uma vez que o teste e a qualidade não devem ser considerados apenas posteriormente, mas deve ser incorporada desde o início.

Assim, na fase de definição do backlog de produto e do backlog da sprint, o papel do analista de testes é realizar a revisão na documentação, investigar requisitos de sistema conflitantes e discuti-los nas reuniões de definição de novas funcionalidades do sistema.

Durante o tempo de uma sprint, que em geral dura cerca de 2 a 3 semanas, apenas uma parte do sistema é planejada, implementada e testada. Ao final da sprint, é lançada uma nova versão do sistema, denominada *release*. Após várias sprints, espera-se que o produto final, com todas as funcionalidades previstas no backlog do produto, seja desenvolvido e entregue.

Verdadeiro ou falso:

1. O modelo em cascata não é mais usado para o desenvolvimento do sistema, pois está obsoleto.

Sua resposta está correta!

- A.
- Verdadeiro
- B.
- Falso

2. Na fase de planejamento, a equipe começa a testar defeitos e erros. Eles continuarão a corrigir todos os problemas descobertos até que o produto atenda às especificações originais.

Sua resposta está correta!

- A.
- Verdadeiro
- B.
- Falso

3. Com o desenvolvimento ágil, testadores e desenvolvedores precisam trabalhar juntos e as funções podem ser intercambiáveis. Por isso, é importante que os requisitos estejam sólidos e a equipe esteja bem informada. Os testes entrarão em ação a partir do momento em que as histórias de usuário forem escritas. Tanto a equipe de desenvolvimento quanto a de teste precisam ter uma compreensão clara do escopo de trabalho esperado. A transparência é a regra básica do desenvolvimento ágil.

Sua resposta está correta!

- A.
- Verdadeiro

- B.
- Falso

Introdução aos níveis de teste de software



Imagen de storyset no Freepik, CC0

Um software é muito mais do que várias linhas de código. Geralmente é um sistema multicamada complexo, incorporando dezenas de componentes funcionais distintos e integrações com terceiros. Portanto, testes de software eficientes devem ir muito além de apenas encontrar erros no código-fonte. Normalmente, o teste abrange os seguintes níveis de software:



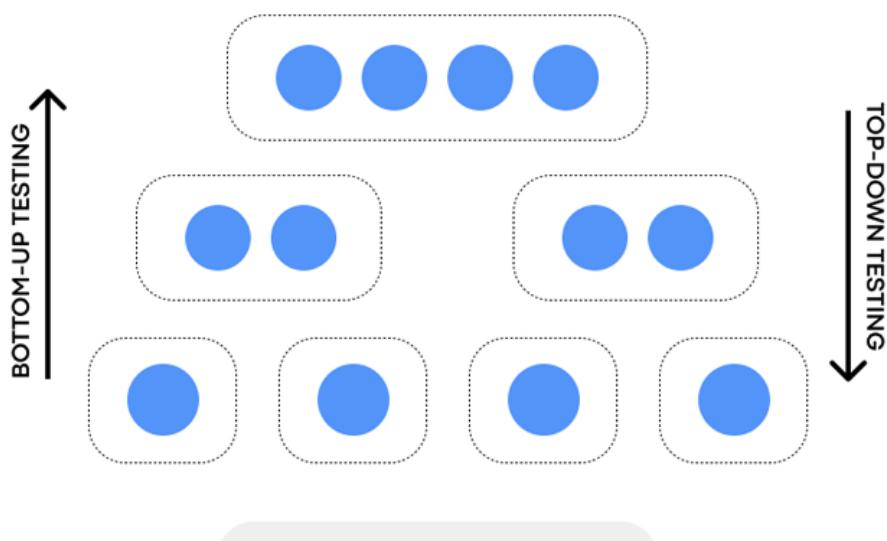
última.

Teste de componente/unidade

A menor parte testável do sistema de software é chamada de unidade. Portanto, este nível de teste visa examinar cada unidade de um sistema de software para garantir que ele atenda aos requisitos e funções originais conforme o esperado. O teste de unidade geralmente é realizado no início do processo de desenvolvimento pelos próprios engenheiros de software, e não pela equipe de teste.

Teste de integração

O objetivo do próximo nível de teste é verificar se as unidades combinadas funcionam bem em conjunto. O teste de integração visa detectar as falhas nas interações entre as unidades dentro de um módulo. Existem duas abordagens principais para este teste: os métodos de baixo para cima (bottom-up) e de cima para baixo (top-down). O teste de integração de baixo para cima começa com testes de unidade, aumentando sucessivamente a complexidade dos módulos de software sendo testados. O método de cima para baixo adota a abordagem oposta, concentrando-se primeiro nas combinações de alto nível e examinando as mais simples depois.



última.

Teste de integração: métodos bottom-up e top-down

Teste de sistema

Neste nível, um sistema de software completo é testado como um todo. Esta etapa serve para verificar a conformidade do produto com os requisitos funcionais, técnicos e com os padrões gerais de qualidade. O teste do sistema deve ser realizado por uma equipe de testes altamente profissional em um ambiente o mais similar possível ao cenário real de utilização.

Teste de aceitação

Esta é a última etapa do processo de teste, onde o produto é validado em relação aos requisitos do usuário final e quanto à precisão. Esta etapa final ajuda a equipe a decidir se o produto está pronto para ser lançado ou não. Embora pequenos problemas devam ser detectados e resolvidos no início do processo, este nível de teste se concentra na qualidade geral do sistema, cobrindo desde o conteúdo e a interface do usuário até os problemas de desempenho. A etapa de aceitação pode ser seguida por testes alfa e beta, que permitem que usuários reais selecionados experimentem o software antes deste ser lançado oficialmente.

Os níveis de teste de software em um modelo ágil

No desenvolvimento de software Ágil, o teste normalmente é um processo iterativo. Embora os níveis de teste geralmente se refiram ao produto completo, eles também podem ser aplicados aos recursos adicionados posteriormente. Neste caso, cada pequena unidade da nova funcionalidade será verificada. Em seguida, os engenheiros verificam as interconexões entre essas unidades, a forma como o recurso se integra ao restante do sistema e se a nova atualização está pronta para ser lançada.

Combine cada exemplo abaixo com o respectivo nível de teste:

1. Verificar se a comunicação entre os sistemas é realizada corretamente.

Opções de múltipla escolha (selecione um)

- A.
- **Teste de unidade**
- B.
- **Teste de integração**
- C.
- **Teste de sistema**
- D.
- **Teste de aceitação**

2. Em testes funcionais, verificamos se um recurso de login responde quando o usuário digita uma senha. E em testes não funcionais, verificamos quanto tempo leva para o usuário fazer login após a entrada da senha.

- A.
- **Teste de sistema**
- B.
- **teste de aceitação**
- C.
- **Teste de unidade**
- D.

- **Teste de integração**

3. Verificar se um componente está ou não cumprindo as funcionalidades esperadas. Por exemplo, conferir se uma calculadora realiza uma divisão quando os usuários clicam no sinal \div .

- A.
- teste de aceitação
- B.
- Teste de integração
- C.
- Teste de sistema
- D.
- Teste de unidade

4. O software finalizado é enviado aos usuários selecionados para que o usem e testem por determinado tempo, e então forneçam feedback.

- Teste de sistema
- B.
- Teste de integração
- C.

- teste de aceitação

- D.

- Teste de unidade

Métodos de teste de software



Vamos conhecer os principais métodos de testes de software:

Teste de caixa preta

Este método recebe esse nome porque o analista de testes se concentra nas entradas e nas saídas esperadas, sem saber como o aplicativo funciona internamente e como essas entradas são processadas. O objetivo deste método é verificar a funcionalidade do software certificando-se de que ele funciona corretamente e atende às demandas do usuário. Pode ser aplicado a qualquer nível de teste, mas é usado principalmente para testes de sistema e de aceitação..



última.

Teste de caixa branca

Ao contrário do teste de caixa preta, este método requer profundo conhecimento do código, pois envolve o teste de partes estruturais do aplicativo. Portanto, os desenvolvedores diretamente envolvidos na escrita do código são geralmente responsáveis por esse tipo de teste. O objetivo do teste de caixa branca é aumentar a segurança, o fluxo de entradas/saídas do aplicativo e aprimorar o design e a usabilidade. Este método é utilizado principalmente nos níveis de teste de unidade e integração.

Teste de caixa cinza

Este método é uma combinação dos dois anteriores, pois envolve o teste de partes funcionais e estruturais do aplicativo. Um testador experiente, com conhecimento parcial da estrutura interna do aplicativo, pode projetar casos de teste estruturais sob a perspectiva da funcionalidade. Este método é principalmente utilizado no nível de teste de integração.

Testes *Ad Hoc*

Este é um método de teste informal, pois é realizado sem planejamento ou documentação. Ao conduzir testes de maneira informal e aleatória, sem resultados formais esperados, o testador improvisa as etapas e as executa arbitrariamente. Embora os defeitos encontrados com este método sejam mais difíceis de reproduzir devido à ausência de casos de teste escritos, esta abordagem ajuda a encontrar defeitos importantes rapidamente, algo que não pode ser feito com os métodos formais.

Verdadeiro ou falso:

À medida que os softwares se tornam cada vez mais complexos e entrelaçados, e com a grande quantidade de plataformas e dispositivos diferentes que precisam ser testados, é mais importante do que nunca ter uma metodologia de testes robusta para garantir que os produtos/sistemas de software em desenvolvimento tenham suas falhas corrigidas. Dessa maneira, as empresas conseguem garantir que seus produtos atendam aos requisitos especificados e podem operar com sucesso em todos os ambientes previstos, com a usabilidade e a segurança necessárias.

Sua resposta está correta!

- A.
- Verdadeiro
- B.
- Falso

Arquitetura de software: MVC e microsserviços



Em uma economia cada vez mais competitiva, as empresas modernas precisam ser capazes de responder rapidamente aos desejos de seus clientes. Quando falamos de empresas onde a tecnologia é um fator diferencial, a agilidade torna-se ainda mais importante. Assim, ter uma arquitetura de software que permita à empresa reagir ao mercado com agilidade é essencial.

A arquitetura de software descreve como os diferentes elementos de um aplicativo são organizados e como eles interagem entre si. Este é um dos primeiros passos no desenvolvimento de software e ocorre durante a fase de projeto. No universo do desenvolvimento de software, existem algumas arquiteturas bem conhecidas, mas vamos nos focar em apenas duas delas: MVC e Microsserviços.

MVC (*Model-View-Controller* em inglês) é um padrão de arquitetura que controla o fluxo do código separando o aplicativo em três componentes interconectados. É um padrão utilizado principalmente em estruturas de desenvolvimento web. Os três principais componentes do MVC são os seguintes:

Modelo

O Modelo é responsável por armazenar os dados e responder a toda lógica relacionada aos dados que o usuário utiliza. Em outras palavras, o Modelo é usado para gerenciar os dados do aplicativo, transferindo-os entre a Visão e o Controlador.

Por exemplo, um usuário busca suas informações pessoais no banco de dados, faz alterações e essas informações são então atualizadas nesse banco de dados.

A adição e a manipulação dos dados é feita no componente Modelo.

Os componentes do Modelo respondem apenas às solicitações do Controlador. Quando o Controlador solicita dados específicos, o banco de dados não pode se comunicar diretamente com ele. Portanto, o Modelo se comunica com o Controlador e envia os dados necessários a ele.

Visão

Toda a lógica da interface do usuário é realizada no componente Visão. Em outras palavras, é nesse componente que o programador constrói a interface do usuário de um aplicativo. No caso de uma aplicação web, a Visão contém apenas a parte HTML e CSS do código.

Por exemplo, caixas de texto, formulários, listas suspensas e botões estão presentes no componente de exibição. O usuário interage com a interface do aplicativo. Quando o usuário solicita os dados, o Controlador se comunica com o Modelo, este recupera os dados e os envia para a Visão.

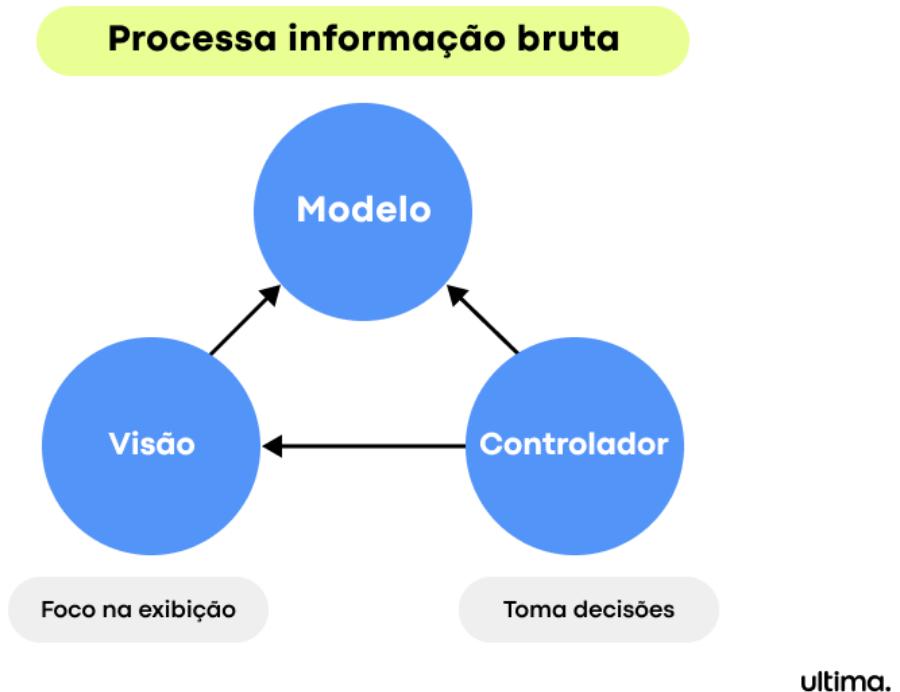
É importante notar, contudo, que a Visão não pode se comunicar diretamente com o Modelo.

Controlador

O Controlador é o principal componente da arquitetura MVC, sendo responsável por toda a lógica de negócio. Ele lida com todas as solicitações da Visão e atua como uma interface entre o Modelo e a exibição.

O Controlador recebe requisições da Visão. Na sequência, ele executa a lógica de negócio de acordo com a solicitação e manipula os dados.

Por exemplo, quando o componente Visão solicita os dados, o Controlador recupera os dados do Modelo, executa a tarefa requisitada e os atualiza no modelo de dados.

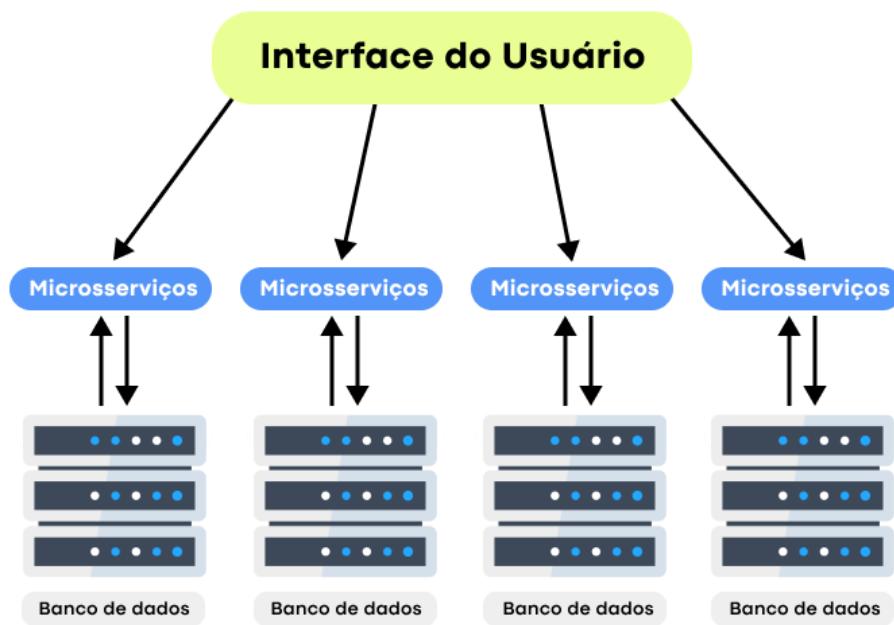


Como discutimos anteriormente, métodos ágeis preconizam iterações rápidas, com entregas frequentes de novos lançamentos, a fim de obter feedback e, se for preciso, efetuar mudanças de rumo. Porém, mesmo que uma empresa adote um método ágil, ela enfrentará um gargalo quando precisar fazer lançamentos de um produto de forma frequente em uma arquitetura MVC,

também conhecida como monolítica. Isso ocorre porque, nesse tipo de arquitetura, mudanças em um componente podem afetar diretamente outros componentes do sistema, causando defeitos ocultos e difíceis de rastrear.

Os microsserviços podem ser definidos como uma melhoria para a arquitetura MVC, e também como uma espécie de refinamento do que conhecemos como arquitetura orientada a serviços. Nesta arquitetura, um aplicativo grande é construído na forma de pequenos módulos monofuncionais. Logo, cada microsserviço é autônomo.

Os microsserviços não precisam compartilhar uma mesma camada de dados, pois cada serviço implementa uma funcionalidade do sistema. Assim, cada serviço pode ter seu próprio banco de dados sem comprometer a integridade dos outros serviços. Além disso, cada serviço pode ter seu próprio servidor de aplicação, permitindo que o sistema seja escalável, já que cada serviço pode ser disponibilizado em máquinas diferentes, sem compartilhar memória e processamento.



última.

Como os microsserviços são autônomos, eles também podem ser implementados em diferentes linguagens de programação. Por exemplo, um aplicativo de troca de mensagens que disponibiliza sua interface em uma aplicação Web e também em aplicativos para celulares Android e iOS, de forma sincronizada. Esse tipo de abordagem é possível quando separamos os serviços de negócio e os dados em um back-end em Java e os serviços de interface em vários front-ends que usam as linguagens JavaScript para web, Java para Android e Objective-C para iOS.

Dessa forma, para que todas as informações sejam transmitidas do back-end para cada front-end de forma sincronizada e em tempo real, independentemente da aplicação que o usuário esteja utilizando, é necessário que sejam implementadas APIs (Interface de programação de aplicativos). Assim, todos os serviços oferecidos pelo back-end ficam disponíveis, sem a necessidade de acessar o código-fonte diretamente, e eles podem ser consumidos por cada aplicação separadamente.

Como resultado, só é necessário criar um serviço separado, sem ter que reimplantar back-ends inteiros para cada uma dessas interfaces. Essa separação ajuda na manutenção do sistema e na redução de defeitos, já que não haverá código duplicado do back-end. Atualmente, há também frameworks que permitem criar um único serviço front-end para várias plataformas (desktop, web, Android e iOS), como é o exemplo do Flutter. Isto reduz ainda mais a necessidade de duplicação de código e de retrabalho nos front-ends.

Quais as semelhanças entre o MVC e os microsserviços?

- Ambos possuem uma arquitetura que separa cada parte do sistema em camadas;
- Seus componentes podem ser trabalhados de forma independente, o que ajuda no tempo de processamento.

Como eles se diferem?

MVC: sistemas monolíticos com divisão em três componentes de código: Modelo, Visão e Controlador. Este modelo está sendo usado por empresas como Microsoft, Dell e Marketwatch.

Microserviços: uma aplicação é dividida em um conjunto de serviços especializados que podem englobar partes das camadas MVC separadamente e interagem entre si por meio de APIs. Este modelo está sendo usado por empresas como Netflix, Spotify e eBay.

Verdadeiro ou falso:

A segregação do desenvolvimento é mais fácil em microserviços, pois a equipe de desenvolvimento pode dividir a parte de codificação conforme a necessidade de cada serviço. Também é possível incluir equipes de qualidade de software especializadas nas tecnologias de cada um deles.

Verdadeiro ou falso:

A segregação do desenvolvimento é mais fácil em microserviços, pois a equipe de desenvolvimento pode dividir a parte de codificação conforme a necessidade de cada serviço. Também é possível incluir equipes de qualidade de software especializadas nas tecnologias de cada um deles.

Sua resposta está correta!

A.

Verdadeiro

B.

Falso

Material Complementar

Qualidade, Qualidade de Software e Garantia da Qualidade de Software são as mesmas coisas?

Este artigo tem por objetivo mostrar as diferenças entre os termos Qualidade, Qualidade de Software e Garantia da Qualidade de Software. Com esse artigo, podemos esclarecer a diferença e até mesmo alguns relacionamentos entre estes três termos.

por Fábio Martinho Campos

Segundo a NBR ISO 9000:2005, "qualidade é o grau no qual um conjunto de características inerentes satisfaz aos requisitos". Ou seja, pode-se afirmar que se algum produto ou serviço atende aos requisitos especificados, este mesmo produto ou serviço possui a qualidade desejada.

A qualidade pode ser medida através do grau de satisfação em que as pessoas avaliam determinado produto ou serviço. No entanto, esse produto ou serviço pode ter qualidade para algumas pessoas e para outras nem tanto, ou seja, a qualidade é algo subjetivo.

Conceituar desta forma então o termo qualidade se torna uma tarefa muito difícil, pois elementos intrínsecos estão enraizados no intelecto de cada ser.

O termo TQM (*Total Quality Management*), amplamente usado nas organizações, também descreve uma abordagem para a melhoria da qualidade. De acordo com Kan (2002), "O termo tem tomado vários significados, dependendo de quem interpreta e como se aplica." (KAN, 2002, p. 30). Independente dos seus vários tipos de implementação, os elementos chave do TQM podem ser resumidos conforme Figura 1, abaixo:

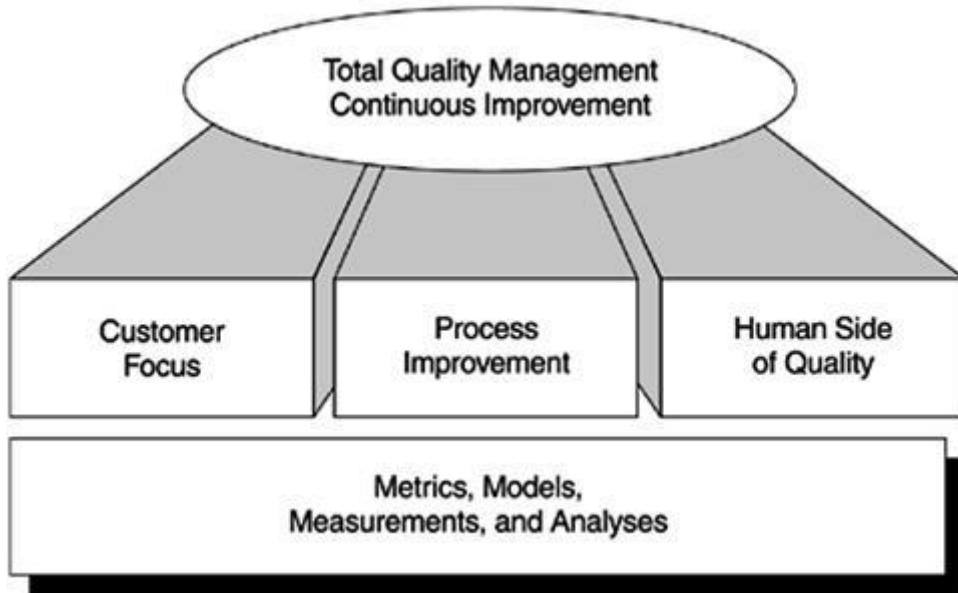


Figura 1 - Elementos chave do TQM.

1. Foco do Cliente (*Customer Focus*) - O objetivo é atingir a satisfação total do cliente. O foco do cliente inclui o estudo das necessidades e vontades do cliente, coleta de requisitos do cliente e a medição e gerenciamento da satisfação do cliente.
2. Melhoria de Processo (*Process Improvement*) - O objetivo é reduzir as variações de processo e atingir a melhoria da qualidade contínua. Este elemento inclui ambos os processos de negócio e o processo de desenvolvimento do produto. Através da melhoria de processo, a qualidade do produto será reforçada.
3. Lado Humano da Qualidade (*Human Side of Quality*) - O objetivo é criar a cultura de qualidade por toda a empresa. As áreas de foco

incluem liderança, apoio da alta gerência, participação total de todos os colaboradores da empresa, e outros fatores humanos como sociais e psicológicos.

4. Métricas, Modelos, Medições e Análises (*Metrics, Models, Measurement and Analysis*) - O objetivo é direcionar a melhoria contínua em todos os parâmetros da qualidade por um sistema de medição orientado a metas.

No contexto de sistemas de informação e softwares o conceito da ISO aplica-se na sua totalidade, sendo que os usuários finais sempre terão as expectativas de um alto padrão de qualidade para que suas tarefas sejam sempre executadas da maneira mais adequada possível.

Tian (2005) afirma que as expectativas de qualidade para sistemas de software estão relativamente ligadas em dois aspectos:

1. "O sistema de software deve fazer o que é suposto que se faça. Em outras palavras, eles devem fazer certo as coisas". (TIAN, 2005, p. 35)
2. "Eles devem realizar estas tarefas específicas corretamente e satisfatoriamente. Em outras palavras, eles devem fazer as coisas certas". (TIAN, 2005, p. 35)

Indiretamente, as duas afirmações de Tian (2005) estão com base na ISO 9000:2005 sendo que o sistema deve fazer o que se espera que ele faça, de acordo com seus requisitos levantados e especificados.

Contudo, a qualidade possui alguns princípios básicos, como:

- Tentar prevenir defeitos ao invés de consertá-los;

- Ter a certeza que os defeitos que foram encontrados, sejam corrigidos o mais rápido possível.
- Estabelecer e eliminar as causas, bem como os sintomas dos defeitos;
- Auditá o trabalho de acordo com padrões e procedimentos previamente estabelecidos.

Segundo ainda a NBR ISO 8402, o conceito de qualidade é "A totalidade das características de uma entidade que lhe confere a capacidade de satisfazer às necessidades explícitas e implícitas". As necessidades explícitas são aquelas expressas na definição formal de requisitos propostos pelo cliente. Esses requisitos definem as condições em que o produto ou serviço devem ser utilizados bem como seus objetivos, funções e o desempenho esperado. Já as necessidades implícitas são aquelas que, embora não expressas pelo cliente nos documentos de requisitos, são necessárias para o usuário. Estão incluídos nessas classes tanto os requisitos que não precisam ser declarados por serem óbvios como aqueles requisitos que não são percebidos como necessários no momento que o produto foi desenvolvido, mas que pela gravidade de suas consequências devem ser atendidos.

A qualidade, seja ela usada no contexto de software ou de produtos e serviços, hoje não mais é uma obrigação e um diferencial das empresas. A mesma se tornou um padrão em qualquer ramo de atividade e indústria sendo assim necessária para garantir a satisfação do cliente. Qualidade hoje em dia, não é apenas um diferencial de mercado para as empresas conseguirem vender e lucrar mais, é um pré-requisito que se deve conquistar para conseguir colocar o produto ou serviço no Mercado Global. De acordo com Jack Welch, "A qualidade é a nossa melhor garantia da fidelidade do cliente, a nossa mais forte defesa contra a competição estrangeira e o único caminho para o crescimento e para os lucros."

Qualidade de Software

Diante dessa complexidade na definição da palavra qualidade, Pressman (2005) sugere que a qualidade de software seja implementada e não somente uma idéia ou desejo que uma organização venha a ter. Para tanto, Pressman (2005) faz as seguintes colocações sobre qualidade de software:

1. "Definir explicitamente o termo qualidade de software, quando o mesmo é dito";(PRESSMAN, 2005, p. 193)
2. "Criar um conjunto de atividades que irão ajudar a garantir que cada produto de trabalho da engenharia de software exiba alta qualidade"; (PRESSMAN, 2005, p. 193)
3. "Realizar atividades de segurança da qualidade em cada projeto de software";(PRESSMAN, 2005, p. 193)
4. "Usar métricas para desenvolver estratégias para a melhoria de processo de software e, como consequência, a qualidade no produto final"; (PRESSMAN, 2005, p. 193)

Sendo assim, a busca constante pela qualidade não se faz apenas no começo do projeto ou no seu final realizando testes, mas sim e um processo que visa abranger toda a engenharia de software bem como a colaboração de todos os membros do time do projeto.

Uma possível definição mais abrangente e completa para qualidade de software seria a proposta por Bartié (2002): "Qualidade de software é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos". (BARTIÉ, 2002, p. 16)

Alguns modelos de qualidade de software também são citados por Pressman (2005). Há o que McCall e Cavano (1978) sugerem como métricas para qualidade de software. Conhecido como Fatores da Qualidade, estes fatores avaliam o software em três pontos distintos: Transição do Produto, Revisão do Produto e Operação do Produto. Na Figura 2 são mostrados os Fatores da Qualidade de McCall:

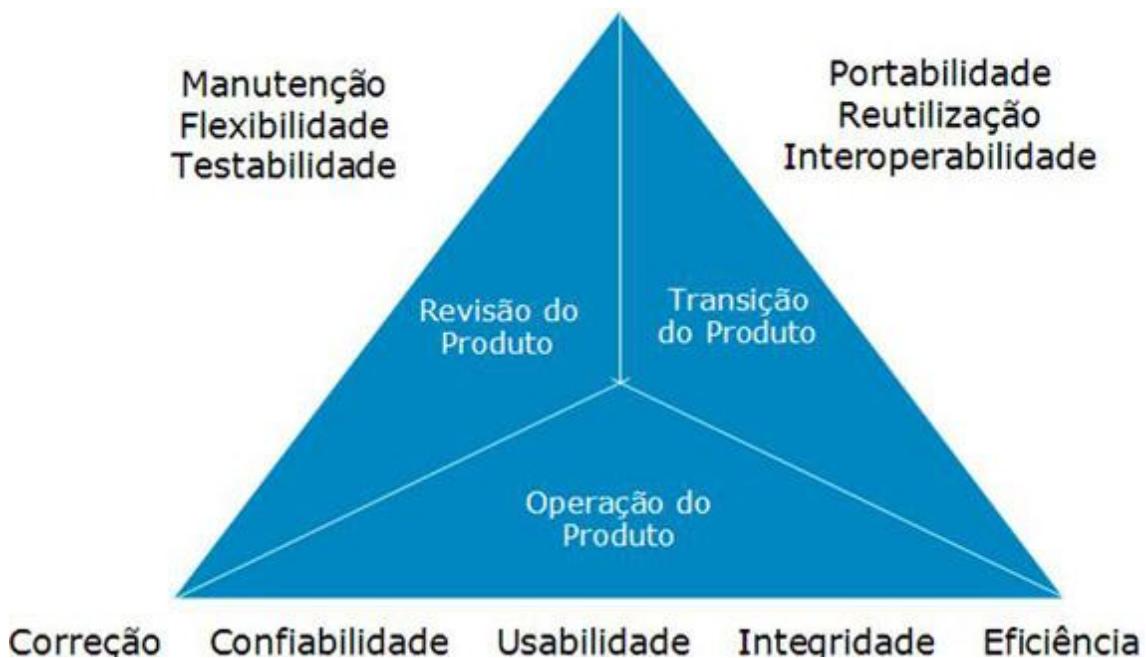


Figura 2 - Os Fatores da Qualidade de McCall.

Assim como o modelo proposto por McCall e Cavano (1978), "a Hewlett-Packard desenvolveu também um modelo que referencia fatores da qualidade de software e que primeiramente publicado por Grady and Caswell (1987), denominado FURPS: *Functionality, Usability, Reliability, Performance e Supportability*. Estes fatores estabelecem as métricas de qualidade de software para cada fase do processo de engenharia de software". (PRESSMAN, 2005, p. 539)

Além desses modelos de métricas para qualidade de software, nota-se que a constante busca pela mesma se tornou uma atividade essencial dentro das empresas. Colocando-se todos esses conceitos dentro do contexto apresentado, podemos dizer que "qualidade não é uma fase do ciclo de desenvolvimento de software ... é parte de todas as fases".(BARTIÉ, 2002, p. 16)

Portanto, é necessário um planejamento adequado para que a qualidade de software seja atingida, conforme a definição de qualidade que deverá ser alcançada. Para isso são necessários modelos, padrões, procedimentos e

técnicas para atingir essas metas de qualidade propostas. Para tanto, todas as etapas do ciclo de vida de engenharia de software devem ser contempladas com atividades que visam garantir a qualidade tanto do processo quanto do produto.

Software Quality Assurance

Segundo Lewis (2004), uma definição formal de *Software Quality Assurance* (SQA) é "atividades sistemáticas fornecendo evidências para o uso pretendido para o produto total de software".(LEWIS, 2004, p. 18)

Sendo assim, podemos ainda definir como *Quality Assurance* "o conjunto de atividades de apoio para fornecer confiança de que os processos estão estabelecidos e estão continuamente melhorados para produzir produtos que atendam as especificações e que sejam adequados para o uso pretendido".
(LEWIS, 2004, p. 18)

Com isso, o SQA envolve todo o processo de desenvolvimento de software fazendo as devidas monitorações e melhorias de processos pertinentes, fazendo com que os padrões, procedimentos acordados estão sendo seguidos e garantindo que problemas são encontrados e ações corretivas são tomadas. Esse tipo de ação é orientada a prevenção. O IEEE 610.12-1990 cita qualidade de software como "(1) Um padrão planejado e sistemático de todas as ações necessárias para fornecer confiança adequada que um item ou produto está em conformidade com os requisitos técnicos estabelecidos. (2) Um conjunto de atividades projetadas para avaliar o processo pelo qual produtos são desenvolvidos ou manufaturados".

O SQA é também entendida e formada por um grupo de pessoas relacionadas e empregadas através de todo o ciclo de vida de engenharia de software que positivamente influenciam e quantificam a qualidade do software que está sendo entregue. Como já foi dito, o SQA não é somente uma atividade associada exclusivamente com atividades de desenvolvimento de software,

mas sim atividades que se expandem durante todo o ciclo de vida de desenvolvimento de software. Portanto, isso consiste em realizar a qualidade tanto do processo quanto ao produto. No processo, podemos quantificar a sua qualidade através de métricas para qualidade de software e no produto com as técnicas de verificação e validação. Essas atividades podem ser, por exemplo, avaliações como as citadas pela ISO 9000, auditorias, inspeções formais, teste de software, revisões. Ainda no processo podemos usar os métodos de garantia da qualidade no formato de auditorias e reportes para a alta gerência, além de avaliações constantes do processo e análise estatística de controle do processo. No produto os métodos de garantia da qualidade são revisões, inspeção formal e teste de software, além de revisão dos resultados do teste de software realizada por experts, auditorias do produto e testes realizados pelo cliente.

No entanto, empresas que não possuem o grupo ou processos de SQA tendem a mostrar os seguintes indicadores de falta de qualidade, conforme Lewis (2004):

1. O software que foi entregue freqüentemente apresenta falhas;
2. Inaceitáveis conseqüências de falhas de sistemas, desde financeiras até cenários reais de aplicação;
3. Sistemas não estão freqüentemente disponíveis para uso pretendido;
4. Sistemas são freqüentemente muito caros;
5. Custo de detectar e remover defeitos são excessivos.

Por outro lado, empresas que possuem o grupo ou processo de SQA implementados e a sua aplicação de maneira adequada e correta mostra que:

1. A remoção de erros acontece no momento em que se é barato corrigir;
2. Melhoria da qualidade do produto;

3. O SQA é um recurso para a melhoria de processo;
4. Estabelecimento de um banco de dados de métricas como:
planejamento, taxas de falhas e outros indicadores da qualidade;

Lewis (2004) cita as atividades mais comuns do SQA, sendo estas categorizadas como: Teste de Software (Verificação e Validação), Gerenciamento de Configuração de Software e Controle da Qualidade. Na Figura 3, podemos ver a relação entre essas três principais atividades juntamente com Padrões, Procedimentos, Convencões e Especificações:

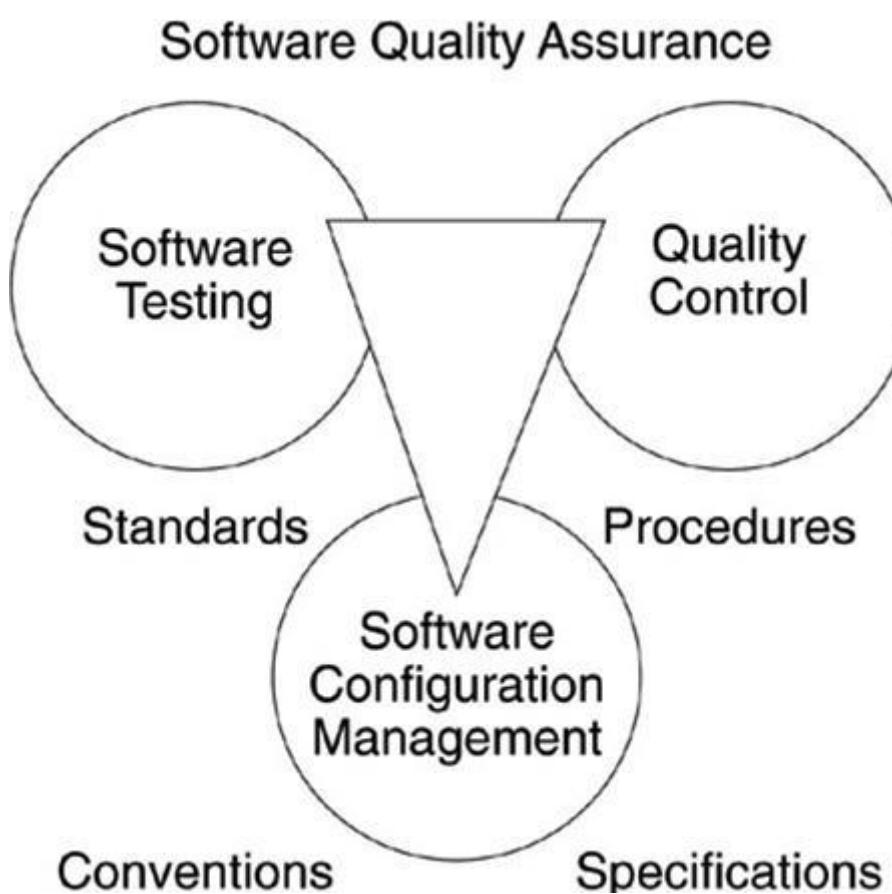


Figura 3 - Componentes do SQA.

1. **Teste de Software (*Software Testing*)**: Conforme Lewis (2004), "É uma estratégia popular para o gerenciamento de risco". (LEWIS, 2004, p. 19)
O teste de software é usado para verificar que requisitos funcionais

e não-funcionais foram devidamente implementados. A limitação dessa abordagem (Teste de Software), entretanto, é que na fase em que ele acontece, muitas vezes é difícil de se conseguir alguma qualidade no produto. Isso porque muitas empresas ainda usam o modelo Waterfall, ou modelo cascata, que foca justamente a atividade de teste de software somente no final do modelo, ou seja, caso algum defeito seja encontrado (erros em requisitos, por exemplo) todo ciclo deverá ser inicializado novamente. O teste de software na verdade, foca quase que exclusivamente as atividades de verificação e validação.

2. **Controle da Qualidade (*Quality Control*):** O controle da qualidade é definido como um processo e métodos usados para monitorar o trabalho e observar se os requisitos estão sendo satisfeitos. O foco é justamente em revisões e remoção de defeitos antes mesmo do envio dos produtos. O controle da qualidade deve ser de responsabilidade da unidade organizacional que produz o produto. No entanto, é possível ter o mesmo grupo que constrói o produto, que realize também o controle da qualidade, ou estabelecer um grupo de controle da qualidade separado ou departamento dentro da mesma unidade organizacional que desenvolve o produto.

O controle da qualidade consistem de *checklists* bem definidos em um produto que é especificado no plano de garantia da qualidade. Um exemplos clássico de controle da qualidade são as inspeções de software. A inspeção é o grau mais maduro e formal dentro das revisões, sendo necessária uma preparação prévia, participantes definidos adequadamente e critérios de entrado e saída bem definidos.

O controle da qualidade é projetado para detectar defeitos e corrigir esses defeitos encontrados, enquanto que a garantida da qualidade é orientada através da prevenção de defeitos.

3. Gerenciamento de Configuração de Software (*SCM - Software Configuration Management*):

O SCM é responsável por identificar, rastrear e controlar mudanças nos elementos do software de um sistema. O SCM controla a evolução do sistema de software , gerenciando versões dos componentes de software e seus relacionamentos. O propósito é identificar todos os componentes inter-relacionados do software e para controlar sua evolução através das várias fases no ciclo de vida de desenvolvimento de software.

SCM é uma disciplina que pode ser aplicada para atividades incluindo: desenvolvimento de software, controle de documentação, problemas de rastreamento, controle de mudanças e manutenção. O SCM ainda consiste de atividades que asseguram que arquitetura e codificação são definidas e não podem ser mudados sem uma revisão dos efeitos da mudança e sua documentação. Isso porque conforme definição do SCM é controlar o código-fonte e a sua documentação associada fazendo com que o código-fonte final e suas descrições estão consistentes e representam os itens que estavam revisados e testados.

Para que ainda estes três principais componentes funcionem corretamente, o sucesso do programa de garantida da qualidade de software também depende de uma coerente coleção de padrões, procedimentos, convenções e especificações, conforme Figura 3.

A combinação de todos esses componentes e suas melhores práticas é o que chamamos de *Software Quality Assurance*, e que por sua vez todo esse trabalho é realizado por pessoas, garantindo então a qualidade de software do produto final entregue ao cliente ou usuário final.

Conforme Figura 3 Componentes do SQA, o foco será exclusivamente o componente Teste de Software.

Para que também toda essa estrutura esteja devidamente documentada e aceita por todos os membros do time do projeto, é necessário um planejamento adequado. Esse planejamento é feito no *Software Quality Assurance Plan* ou Plano de Garantia da Qualidade de software. Segundo Lewis (2004), "O plano de garantia da qualidade de software é um resumo ou esboço das medidas de qualidade para garantir níveis de qualidade dentro do esforço do desenvolvimento de software".(LEWIS, 2004, p. 22)

O plano é usado como um *baseline* para comparar os níveis atuais de qualidade durante o desenvolvimento com os níveis planejados de qualidade. "O plano de SQA provê o framework e guias para o desenvolvimento de um código entendível e que seja de fácil manutenção"(LEWIS, 2004, p. 22).

Controle da Qualidade de Software versus Garantia da Qualidade de Software

Um dos grandes erros geralmente cometidos por pessoas e empresas é confundir os conceitos e aplicação dos termos Controle da Qualidade (*Quality Control*) e Garantia da Qualidade (*Quality Assurance*). Embora usados de maneira errônea em muitos lugares, ambos os termos têm propósitos totalmente diferentes.

A Tabela 1 - Diferenças entre Garantia da Qualidade e Controle da Qualidade mostra a diferença entre estas duas atividades:

Quality Assurance	Quality Control
1. Garantia da qualidade garante que o processo é definido e apropriado.	1. As atividades de controle da qualidade focam na descoberta de defeitos em i específicos.

2. Metodologia e padrões de desenvolvimento são exemplos de garantia da qualidade.	2. Um exemplo de controle da qualidade poderia ser: "Os requisitos definidos são os requisitos certos?".
3. Garantia da qualidade é orientada a processo.	3. Controle da qualidade é orientado a produto.
4. Garantia da qualidade é orientada a prevenção.	4. Controle da qualidade é orientado a detecção.
5. Foco em monitoração e melhoria de processo.	5. Inspeções e garantia de que o produto de trabalho atenda aos requisitos especificados.
6. As atividades são focadas no inicio das fases no ciclo de vida de desenvolvimento de software.	6. As atividades são focadas no final das fases no ciclo de vida de desenvolvimento de software.
7. Garantia da qualidade garante que você está fazendo certo as coisas e da maneira correta.	7. Controle da qualidade garante que os resultados do seu trabalho são os esperados conforme requisitos.

Fonte: Quality Assurance is not Quality Control.

Com isso, pode-se afirmar que o teste de software é uma das atividades de controle da qualidade, ou seja, o teste de software é orientado a produto e está dentro do domínio do controle da qualidade.

Qualidade de Software segundo o SWEBOK

O SWEBOK (*Software Engineering Body of Knowledge*) versão 2004 do IEEE (*Institute of Electrical and Electronics Engineers*) apresenta em uma das suas KA's (*Knowledge Areas*) a KA de *Software Quality*. Abaixo na Figura 4, pode-se ver a estrutura da KA's de Software Quality:

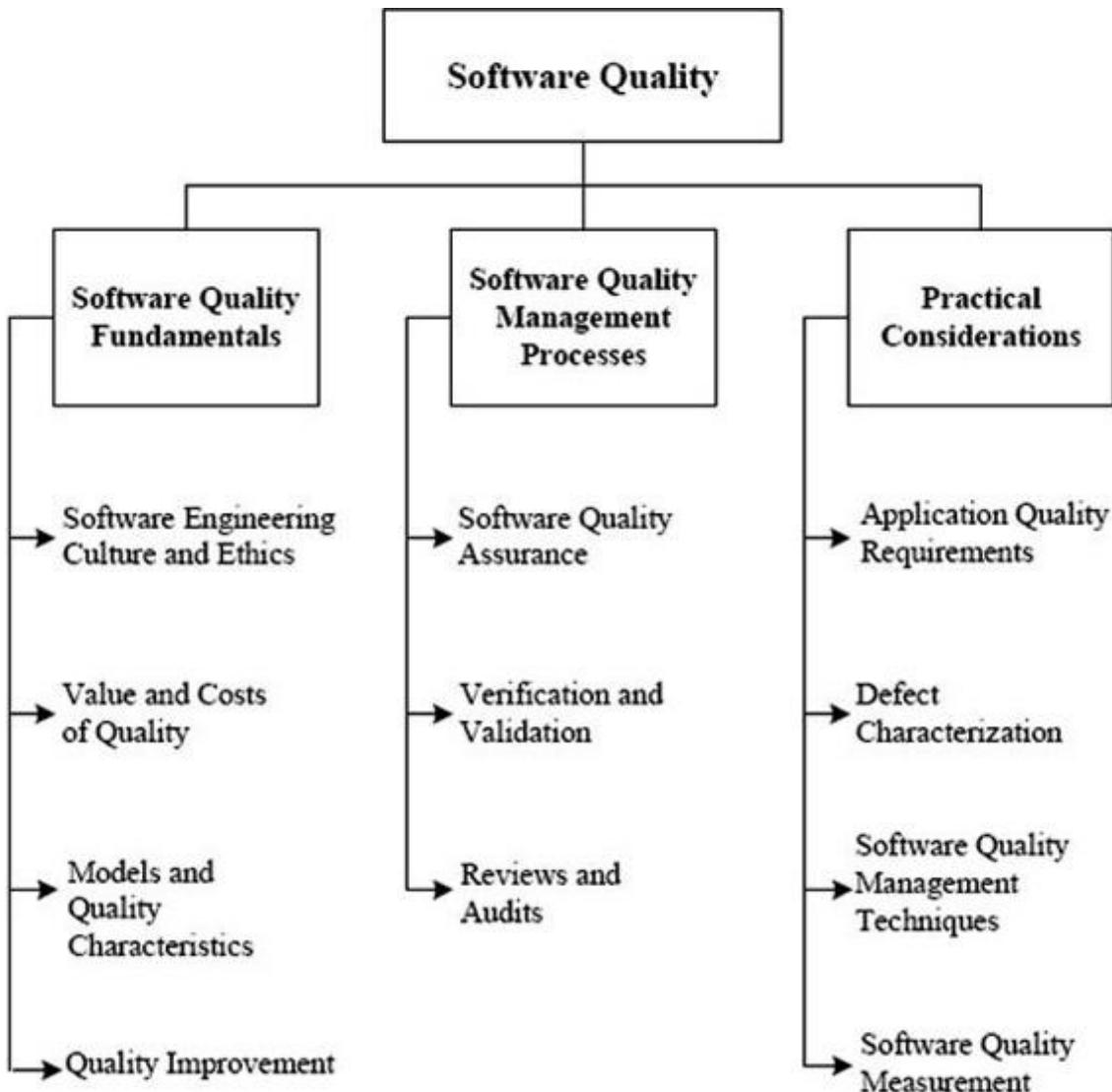


Figura 4 - SWEBOK - KA de Software Quality.

O SWEBOK descreve ainda inúmeras maneiras para se atingir a qualidade sendo que esta KA em específico cobre as técnicas estáticas para detecção de defeitos, não sendo requerido que o software esteja sendo executado, enquanto que as técnicas dinâmicas são cobertas pela KA de Software Testing.

Custo da Qualidade de Software

A qualidade não é totalmente algo que podemos obter de forma gratuita. Para tanto, investimentos financeiros, treinamentos, softwares e outras iniciativas precisam ser realizadas em adição para a busca da qualidade de software como um todo.

Segundo Bartié (2002), "Um dos maiores desafios a ser considerados é estabelecer um modelo de custos relacionados a implantação de um processo de garantia da qualidade de software." (BARTIÉ, 2002, p. 29)

A Figura 5 mostra um modelo de custo de qualidade de software proposto por Bertié (2002):

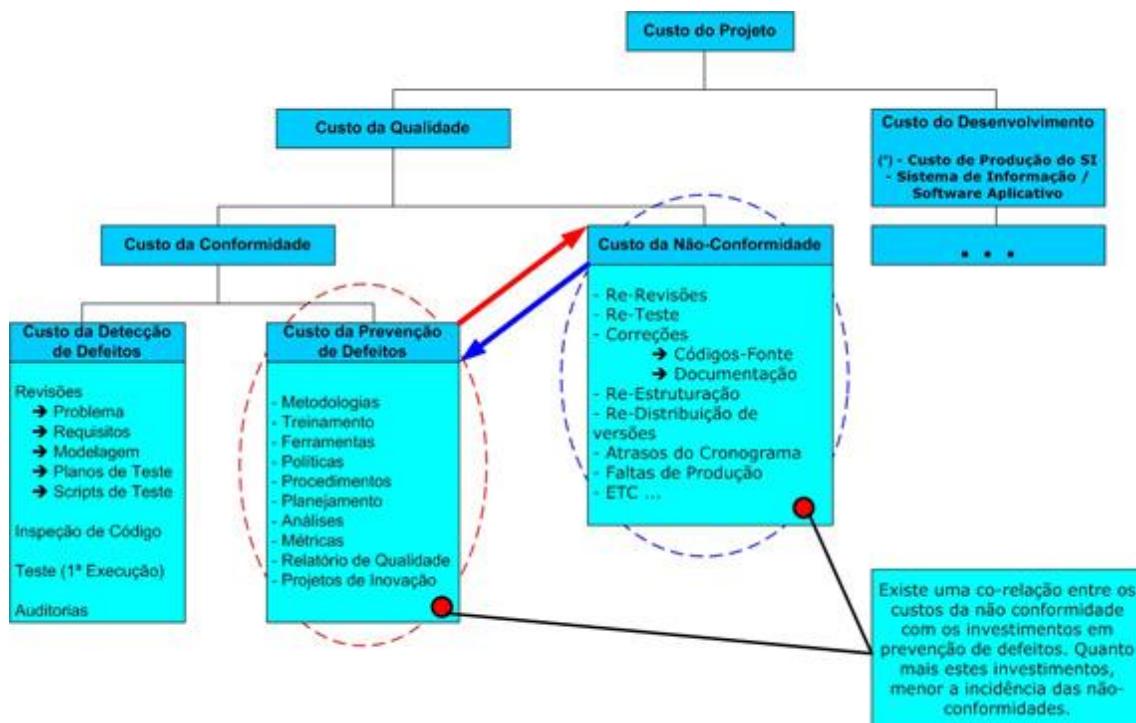


Figura 5 - Modelo de Custo de Qualidade de Software.

No modelo apresentado, Bartié (2002) propõe a criação de duas categorias separadas para os custos relacionados a conformidade e o custo da não-conformidade:

1. Custo da Detecção de Defeitos: Pode-se aqui fazer claramente referências para o termo controle da qualidade, ou seja, o foco está exatamente no produto. As atividades aqui realizadas são orientadas ao produto desenvolvido, e estas incluem:
 - Revisões de requisitos;
 - Revisões de Modelagem;
 - Revisões de Planos de Teste:

- Inspeções de código;
- Testes de Software.

2.

3. Custo da Prevenção de Defeitos: Assim como detecção de defeitos está associada ao controle da qualidade, a prevenção de defeitos está associada a garantia da qualidade, ou seja, o foco está exatamente no processo. As atividades aqui realizadas são orientadas ao processo, e estas incluem:

- Definição de Metodologias;
- Treinamentos;
- Ferramentas de apoio ao processo de desenvolvimento;
- Definição de Políticas;
- Procedimentos;
- Padrões;
- Especificações e convenções;
- Planejamento do SQA;
- Relatórios de Qualidade para melhoria de processo.

4.

5. Custo da Não-Conformidade: Por outro lado, o custo da não-conformidade está relacionado às perdas que o projeto terá, não optando pela detecção e prevenção de defeitos:

- Re-reviões;
- Re-testes;
- Correções de código-fonte e documentação muito constantes;
- Reestruturação;
- Redistribuição das versões do software;
- Atrasos no cronograma;
- Falhas na produção.

Com isso, "O modelo apresentado deverá ser associado a todas as atividades de um processo de engenharia de software. Em todos os projetos a serem construídos ou modificados, todas as atividades deveriam ter uma política de alocação de custos semelhante ao modelo apresentado." (BARTIÉ, 2002, p. 29)

Qualidade de Software segundo a ISO 9126-1

A ISO também apresenta as características da qualidade de software através da norma NBR ISO/IEC 9126-1. A antiga norma brasileira para as características da qualidade de software era a NBR 13.596. No entanto, a mesma se integrou a ISO, formando a NBR ISO/IEC 9126-1. Na Figura 6, são mostradas essas características juntamente com suas subcaracterísticas:

Características	Subcaracterísticas
Funcionalidade (satisfação das necessidades)	<ul style="list-style-type: none">• Adequação (execução do que é apropriado)• Acurácia (execução de forma correta)• Interoperabilidade (interação com quem deve)• Conformidade (aderência às normas)• Segurança de acesso (bloqueio de uso não autorizado)
Confiabilidade (imunidade a falhas)	<ul style="list-style-type: none">• Maturidade (freqüência das falhas)• Tolerância a falhas (forma de reação à falhas)• Recuperabilidade (forma de recuperação de falhas)
Usabilidade (facilidade de uso)	<ul style="list-style-type: none">• Intelegibilidade (facilidade de entendimento)• Apreensibilidade (facilidade de aprendizado)• Operacionalidade (facilidade de operação)
Eficiência (rápido e "enxuto")	<ul style="list-style-type: none">• Tempo (tempo de resposta, velocidade de execução)• Recursos (recursos utilizados)
Manutenibilidade (facilidade de manutenção)	<ul style="list-style-type: none">• Analisabilidade (facilidade de encontrar falha)• Modificabilidade (facilidade de modificar)• Estabilidade (baixo risco quando de alterações)• Testabilidade (facilidade de testar)
Portabilidade (uso em outros ambientes)	<ul style="list-style-type: none">• Adaptabilidade (facilidade de se adaptar a outros ambientes)• Capacidade para ser instalado (facilidade de instalar em outros ambientes)• Conformidade (aderência a padrões de portabilidade)• Capacidade para substituir (facilidade de ser substituído por outro)

Figura 6 - NBR ISO/IEC 9126-1 - Características da Qualidade de Software.

1. Funcionalidade (Satisfação das Necessidades): É a capacidade do produto de software de prover funcionalidades que satisfação as necessidades quando o software está em uso dentro das

condições especificadas.

2. Confiabilidade (Imunidade a Falhas): É a capacidade do produto de software de manter um nível especificado de performance quando o software está em uso dentro das condições especificadas.
3. Usabilidade (Facilidade de Uso): É a capacidade do produto de software de ser entendido, aprendido, usado e atrativo quando o software está em uso dentro das condições especificadas.
4. Eficiência (Rápido e "Enxuto"): É a capacidade do produto de software de prover performance apropriada, relativa ao conjunto de recursos usados quando o software está em uso dentro das condições especificadas.
5. Manutenibilidade (Facilidade de Manutenção): É a capacidade do produto de software de ser mudado. Modificações incluem correções, melhorias ou adaptações do software de mudar em um ambiente, e em requisitos e especificações funcionais.
6. Portabilidade (Uso em outros Ambientes): É a capacidade do produto de software de ser transferido de um ambiente para outro.

Além da NBR ISO/IEC 9126-1, existem ainda outras normas da série 9126, as quais são:

1. ISO/IEC 9126-2 - Métricas Externas: Podem ser aplicadas para um produto não executável durante os estágios de desenvolvimento. Medem a qualidade de produtos intermediários e predizem a qualidade do produto final.

2. ISO/IEC 9126-3 - Métricas Internas: Utilizadas para medir a qualidade do software através do comportamento do sistema ou de parte dele. Só podem ser usadas durante a fase de testes do ciclo de vida e durante a operação do sistema.

3. ISO/IEC 9126-4 - Métricas da Qualidade do Uso: medem se o produto atende ou não as necessidades dos usuários, fazendo-os atingir seus objetivos com efetividade, produtividade, segurança e satisfação. Só podem ser usadas no ambiente real ou em uma aproximação do ambiente real.

A Qualidade segundo o PMBOK do PMI

O PMBOK (*Project Management Body Of knowledge*) do PMI (*Project Management Institute*), na sua versão 2004 apresenta o gerenciamento da qualidade do projeto, conforme Figura 7 abaixo:

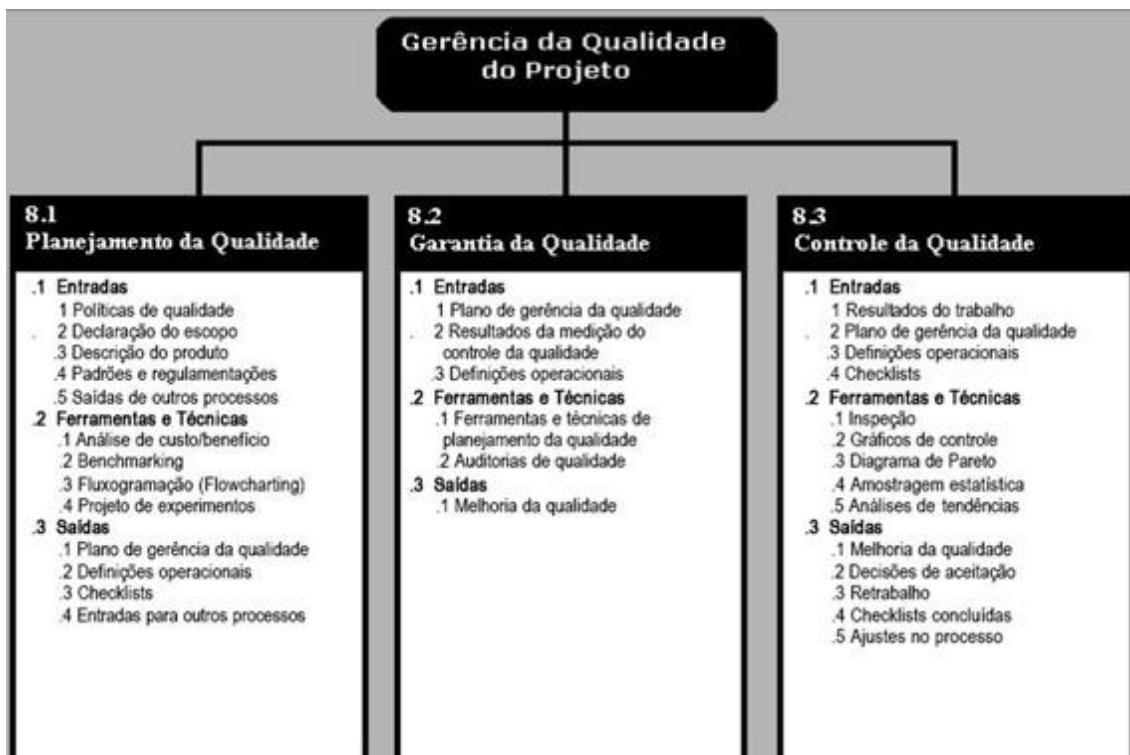


Figura 7 - Gerenciamento da Qualidade do Projeto segundo o PMBOK do PMI.

De acordo com o PMBOK (2004), "Os processos de gerenciamento da qualidade do projeto incluem todas as atividades da organização executora que determinam as responsabilidades, os objetivos e as políticas de qualidade, de modo que o projeto atenda às necessidades que motivaram sua realização. Eles implementam o sistema de gerenciamento da qualidade através da política, dos procedimentos e dos processos de planejamento da qualidade, garantia da qualidade e controle da qualidade, com atividades de melhoria contínua dos processos conduzidas do início ao fim, conforme adequado". (PMI, 2004, p.179)

Com isso os três principais processos são:

1. Planejamento da Qualidade: "Identificação dos padrões de qualidade relevantes para o projeto e determinação de como satisfazê-los." (PMI, 2004, p. 179)
2. Garantia da Qualidade: "Aplicação das atividades de qualidade planejadas e sistemáticas para garantir que o projeto emprega todos os processos necessários para atender aos requisitos.". (PMI, 2004, p. 179)
3. Controle da Qualidade: "Monitoramento de resultados específicos do projeto a fim de determinar se eles estão de acordo com os padrões relevantes de qualidade e identificação de maneiras de eliminar as causas de um desempenho insatisfatório." (PMI, 2004, p. 179)

Como se pode notar é evidente a semelhança entre os conceitos usados no PMBOK e os conceitos da própria ISO. Com isso, é possível ainda relacionar estes três processos do PMBOK com as definições de controle da qualidade e garantia da qualidade de software apresentados anteriormente.

- **Leia também**
- Entendendo o conceito por trás dos processos de Qualidade de Software
- *Qualidade e Testes*
- Entendendo Indicadores de Prazo e Custo de Projetos
- *Qualidade e Testes*
- Aplicação de QUALIDADE de processo de Software
- *Qualidade e Testes*
- Segurança: Item primordial
- *Qualidade e Testes*
- Qualidade de Software: Oculte seu código

entendendo o conceito por trás dos processos de Qualidade de Software

Introdução

Infelizmente, ainda é uma preocupação comum no dia a dia das equipes de implantação a publicação de novas realese de software. Portanto, fica fácil identificar que ainda não estamos dando a real importância para os processos de qualidade de software.

As equipes de desenvolvimento estão enfrentando uma mudança significativa no modelo de desenvolvimento de software com a transição de modelos tradicionais como cascata, por exemplo, para métodos ágeis de desenvolvimento, como o SCRUM, XP, FDD, entre outros. Os métodos ágeis estão cada dia obtendo mais atenção e sendo mais e mais adotados como modelos de desenvolvimento nas organizações e fábricas de software. Com isso, podemos perceber que o mercado ainda se preocupa mais com a velocidade do que com a qualidade, mas será que esse realmente é o caminho? Pare um minuto do seu dia e se faça as seguintes perguntas:

- A equipe realmente está sendo mais rápida?
- Quanto tempo minha equipe está perdendo com manutenção e correção de bugs que não foram identificados antes da implantação?
- Estou realmente entregando um produto de qualidade?
- Meu cliente está feliz com o meu produto?

Na tentativa de auxiliar as equipes de desenvolvimento de software e principalmente os profissionais de qualidade, este artigo aborda alguns conceitos básicos para planejamento um bom plano de teste e aplicação de processos estratégicos e eficientes para validação e garantia da qualidade de software.

Objetivo

O principal objetivo deste artigo é auxiliar os profissionais da área de qualidade de software. Para isso, será apresentado uma simples proposta de documentação de testes com base em casos de uso, além de trazer uma breve definição de várias categorias de testes e também apontar algumas das principais metodologias e ferramentas utilizadas no mercado de TI.

Metodologias

Mps.br: o Modelo de Processos de Software Brasileiro é uma proposta de solução brasileira compatível com o modelo CMMI que está em conformidade com as normas ISO/IEC 12207 e 15504, além de ser adequado à realidade brasileira. Para mais informações, acesse:

http://www.softex.br/mpsbr/_guias/default.asp

Papéis

1. Gerência da Qualidade
 - Gerencia da Qualidade de Software
 - Gerência de Teste de Verificação
2. Profissionais dos Testes de Verificação
 - Revisores de Documentos
 - Auditores da Qualidade
3. Profissionais dos Testes de Validação
 - Arquiteto de Teste
 - Analista de Teste
 - Automatizador de Teste

- Executor de Teste

Categorias de testes de Software

Teste de funcionalidade: teste que verifica se todos os requisitos do cliente foram satisfeitos, executando todas as funcionalidades de uma aplicação.

Teste de Usabilidade: tem por objetivo verificar a facilidade que o software <http://pt.wikipedia.org/wiki/Software> possui de ser claramente compreendido e manipulado pelo usuário <http://pt.wikipedia.org/wiki/Usu%C3%A1rio> na sua navegação(utilização).

Teste de Carga (Stress): teste que avalia um sistema simulando uma situação predeterminada, geralmente um grande volume de carga ou dados. Tem por objetivo identificar gargalos originados por baixo dimensionamento de link, falta de otimização da infraestrutura ou implementação ineficiente do aplicativo.

Teste de Recessão: teste que executa alguns ou todos os casos de testes já verificados, a fim de detectar erros gerados por alterações ou correções feitas durante o processo de desenvolvimento ou manutenção do software.

Teste de Volume: teste que avalia um sistema ou componente executando de modo que a demanda de recursos ocorre em quantidade frequente ou volume irregular.

Teste de Configuração: teste que verifica todas as configurações necessárias para garantir um bom funcionamento do software.

Teste de Compatibilidade: teste que garante a compatibilidade com todos artefatos

Teste de Segurança: teste de segurança visa verificar se todos os mecanismos de proteção embutidos no sistema eliminam as vulnerabilidades do software.

Teste de Desempenho: teste que avalia o tempo de resposta, os recursos utilizados e eficiência de cada função.

Teste de Instalação: teste que verifica se o processo de instalação está executando todos os passos requeridos para implementação do sistema no ambiente de utilização.

Teste de Confiabilidade: teste que valida à confidencialidade das informações, geralmente este teste é aplicado em conjunto com o teste de segurança.

Teste de Disponibilidade: teste que verifica o nível de disponibilidade do software, esta fase tem como objetivo garantir que o sistema estará disponível mesmo com pequenas falhas no ambiente de produção.

Teste de Recuperação: teste de sistema que força o software a falhar de diversas maneiras e verifica se a recuperação é adequadamente executada.

Teste de Contingência: teste também conhecido como plano de continuidade de negócios ou plano de recuperação de desastres, tem como intuito descrever as medidas a serem tomadas para restabelecer o sistema a seu estado funcional ou estado minimamente aceitável em caso de paralisação do sistema.

Checklist de verificação

- Checklist Verificação de Negócios
- Checklist da Proposta de Projeto de Software
- Checklist Verificação de Requisitos
- Checklist Diagrama de UML
- Checklist da Arquitetura
- Checklist Verificação, Análise e Modelagem
- Checklist do Código-Fonte
- Checklist do Banco de Dados
- Checklist Verificação da Implementação

Exemplo de checklist

Checklist do Modelo de Negócios		
Levantamento das Necessidades do Cliente		
- Todas as necessidades foram devidamente registradas.	SI M	NÃ O
- Todas as necessidades foram devidamente registradas.	SI M	NÃ O
Definição das Características do Software		
- Cada característica atende ao menos a uma necessidade identificada.	SI M	NÃ O
- Cada característica possui uma descrição clara.	SI M	NÃ O

- Cada característica possui exemplos que auxiliam seu entendimento.	SI M	NÃ O
- Existe uma rastreabilidade entre característica e necessidade.	SI M	NÃ O

Componentes de testes

Test-drivers: os controladores de teste são programas desenvolvidos com a finalidade de testar uma unidade de software. Esses controladores executam chamadas às unidades a ser validadas. Com isso, definindo uma ordem de execução e um conjunto de parâmetros de entrada que possibilitam testes mais variados em diferentes cenários.

Stubs: os simuladores são programas que simulam o comportamento de uma unidade de software ou hardware. Os simuladores substituem determinadas unidades reais que dificultam ou limitam determinados testes de software.

Reflexão

Segundo Leintz, a proporção de esforços dedicados para manutenção de software corresponde a 20% de corretivas, 25% adaptativas, 50% evolutiva e apenas 5% para preventivas. Esse resultado é com base em um estudo que aconteceu com 487 organizações da área de TI e testes.

Conclusão

Apesar de termos um processo que permite a organização e sequenciamento das atividades de teste de software, é muito importante investir na capacitação das pessoas que vão desempenhá-las. Existem diversas técnicas e práticas para realização de testes, mas é preciso conhecimento do assunto para escolher o que é melhor utilizar, levando em consideração o contexto.

Comunicado Importante

Espero que tenham gostado do conteúdo explorado neste artigo. Caso exista qualquer dúvida relacionada ao conteúdo ou interesse em conhecer um pouco mais sobre qualidade de software, informe seu interesse usando o recurso de "comentários".

Referência

Bartié, Alexandre . Garantia de Qualidade de Software, Campus, 2002.

Paul M. Duvall; Steve Matyas; Andrew Glover . Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley Professional, 2007.

Read more:

<http://www.linhadecodigo.com.br/artigo/3404/entendendo-o-conceito-por-tras-dos-processos-de-qualidade-de-software.aspx#ixzz7lr39iWyX>

Gerência - Qualidade e Testes

Entendendo Indicadores de Prazo e Custo de Projetos

Entendendo Indicadores de Prazo e Custo de Projetos

Cálculo do SPI (Schedule Performance Index) ou IDP (Índice de Desempenho de Prazos) – Project Manager Body Of Knowledge (PMBOK).

$$\text{SPI} = \text{BCWP} / \text{BCWS} \quad \text{OU} \quad \text{SPI} = \text{EV} / \text{PV}$$

BCWP (Budgeted Cost of Work Performed) ou EV (Earned Value)

Tradução: Valor Agregado (VA) – Custo Orçado do Trabalho Realizado (até o momento)

$$\text{BCWP} \text{ ou } \text{EV} = \% \text{Realizado} * \text{BAC}$$

BCWS (Budgeted Cost of Work Scheduled) ou PV (Planned Value)

Tradução: Valor Planejado (VP) – Custo Orçado do Trabalho Planejado (até o momento)

$$\text{BCWS} \text{ ou } \text{PV} = \% \text{Agendado} * \text{BAC}$$

Percentual Realizado

$$\% \text{Realizado} = \text{Andamento Realizado (h)} / \text{Previsão do Projeto (h)}$$

Andamento Realizado: número de horas realizadas no projeto até o momento.

Previsão do Projeto: duração estimada para o projeto em horas.

Percentual Agendado

$$\% \text{Agendado} = \text{Andamento Agendado (h)} / \text{Previsão do Projeto (h)}$$

Andamento Agendado: número de horas planejadas para o projeto até o momento (realizáveis).

Previsão do Projeto: duração estimada para o projeto em horas.

BAC (Budget at Completion) ou ONT (Orçamento No Término)

Orçamento total do projeto conforme baseline de custos (custo previsto total).

Cálculo do CPI (Cost Performance Index) ou IDC (Índice de Desempenho de Custo) – Project Manager Body Of Knowledge (PMBOK).

$$\text{CPI} = \text{BCWP} / \text{ACWP} \text{ OU } \text{CPI} = \text{EV} / \text{AC}$$

BCWP (Budgeted Cost of Work Performed) ou EV (Earned Value)

Tradução: Valor Agregado (VA) – Custo Orçado do Trabalho Realizado (até o momento)

BCWP ou EV = %Realizado * BAC

ACWP (Actual Cost of Work Performed) ou AC (Actual Cost)

Tradução: Custo Real (CR) – Custo Real do Trabalho Realizado (até o momento)

ACWP ou AC = (Custo Apurado do Projeto até o Momento)

Percentual Realizado

%Realizado = Andamento Realizado (h) / Previsão do Projeto (h)

Andamento Realizado: número de horas realizadas no projeto até o momento.

Previsão do Projeto: duração estimada para o projeto em horas.

BAC (Budget at Completion) ou ONT (Orçamento No Término)

Orçamento total do projeto conforme baseline de custos (custo previsto total).

Limitações

A utilização desses indicadores com metodologias ágeis poderá necessitar de adaptações visto as características de gestão e acompanhamento de projetos ágeis serem diferentes, logo, recomendo a utilização com cautela pois os números podem não apresentar a realidade do projeto, assim como, em alguns casos, mesmo na metodologia clássica de gestão de projetos o cálculo do valor agregado pode exigir uma interpretação além dos dados utilizados no cálculo (medidas de qualidade por exemplo).

Os indicadores SPI e CPI (Gerenciamento de Valor Agregado) são ferramentas da caixa de ferramentas do Gerente de Projetos.

www.iheringer.com.br

Segurança: Item primordial

Segurança, simples hábitos que nos podem poupar muita dor de cabeça

Veja neste artigo onde irei abordar um item muito importante, Segurança, simples hábitos que nos podem poupar muita dor de cabeça.

Durante algumas análises de segurança nos arquivos publicados em servidores aqui na empresa que presto serviço, identificamos várias falhas que talvez podem estar acontecendo com qualquer pessoa, talvez não seja do conhecimento de todos ou até hábitos adquiridos que depois são difíceis de abandonar é válido que todos saibam e mudemos a cultura desde já caso se enquadre na situação que encontrei aqui.

Encontrei nos servidores arquivos renomeados vejam abaixo alguns exemplos:

- Excel_bkp08062011.asp;
- buscador.asp_bkp16082011;
- Participantes.bkp;
- App_Web_carrinhoconfirmacao.aspx..bkp;
- video2bkp.old;
- btt_fecharpedido.gif.mno;
- configuraca.bkp;
- configuraca.bak;
- configuraca.old;
- web.config.bak;
- web.config.homologacao;
- NOMEBANCO.dmp;

... e por ai uma infinidade de arquivos malucos.

Entendo que muitas vezes construímos sistemas para intranet, por isso não nos importamos com determinados hábitos, fazer backup dos arquivos antes de mandar os arquivos novos muitas vezes é uma prática salvadora, porém quando utilizamos esta alternativa descrita acima mesmo em sistemas internos estamos indo contra todo tipo de boa prática de programação e pior ainda quando nossos servidores estão com acesso externos.

Aqui na empresa ouvi a pergunta... -"Como alguém vai saber que eu tenho um arquivo .bak no servidor ou old ??" - Simples, para pessoas mal intencionadas é só fazer um teste abrindo uma página .asp e na sequência tentar mudar a extensão pra .old, .txt ou bak ou data ou qualquer outra

ferramenta. Eu que jamais procurei este tipo de ferramenta conheço umas duas ou três ferramentas e até mesmo o google se utilizado de forma procurar este tipo de falha ajuda bastante estas pessoas mal intencionadas.

A recomendação é que NUNCA, NUNCA, NUNCA, NUNCA, e NUNCA(5x nunca), deixemos os backup, arquivos renomeados no servidor. Principalmente arquivos que tem dados de conexão com banco de dados, informações de pagamento e todo e qualquer dado sigiloso.

O papel do QA nas diferentes metodologias de desenvolvimento

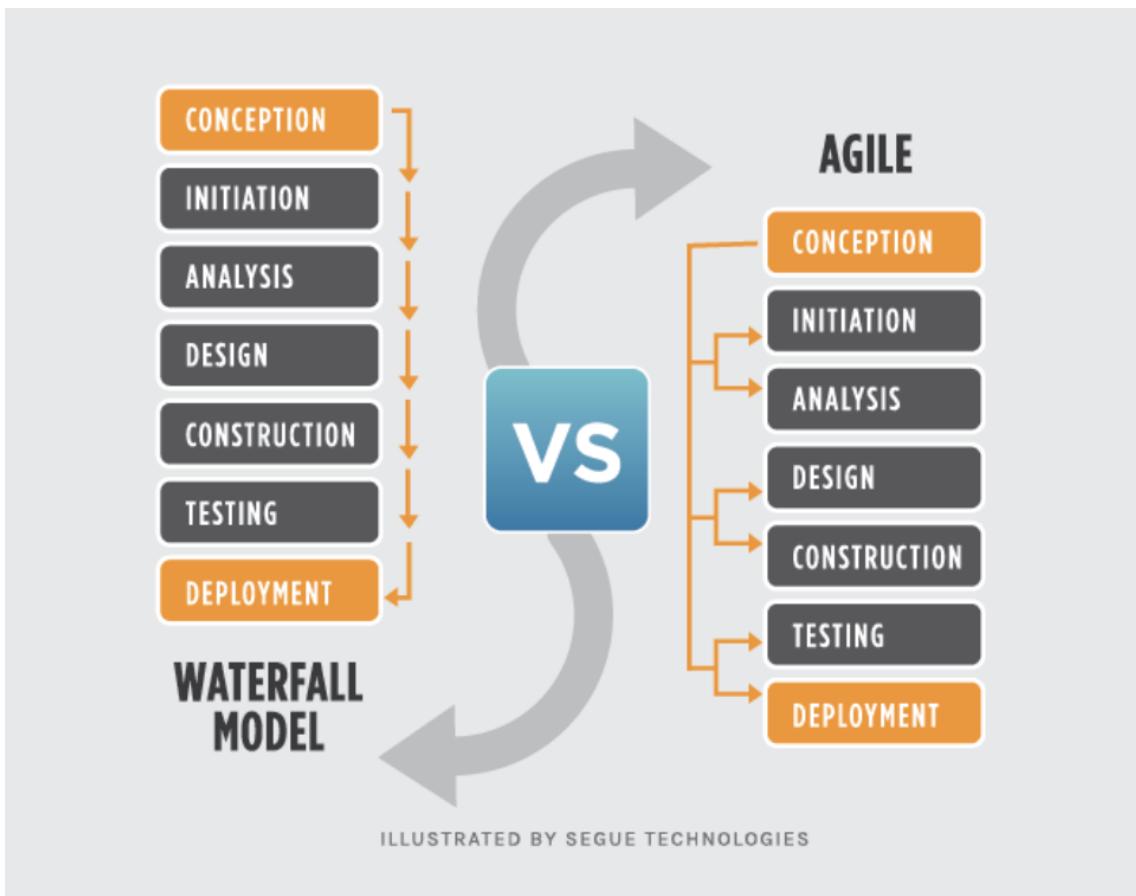
Nesta corrida frenética do mercado e avanço na tecnologia, empresas buscam incessantemente que os seus sistemas sejam funcionais, práticos, e tenham rentabilidade.

É cada vez mais comum que as mesmas utilizem boas práticas que possibilitem uma melhor gestão de projetos e controle em suas atividades.

Uma das metodologias utilizadas para este objetivo é o Scrum, seja em qualquer área ou segmento de trabalho, ela busca alcançar bons resultados no gerenciamento de projetos, mantendo-os dentro do tempo e dos custos pré-determinados.

Em poucas palavras, o Scrum propõe que um projeto seja dividido em pequenos ciclos de atividades, com reuniões rotineiras, alinhando a equipe no que vem fazendo e também pensar formas de melhorar o processo com agilidade.

Para a produção em si do sistema, há duas metodologias bem difundidas no mercado, a metodologia cascata ou metodologia tradicional, conhecida também com Waterfall, e a metodologia ágil. Veremos a seguir as suas características e as responsabilidades da qualidade em cada uma delas.



Metodologia Cascata

O modelo cascata é uma espécie de guia que segue de forma sequencial as etapas para o desenvolvimento de um projeto.

Nos meados da década de 1970, Winston Walker Royce apresentou o primeiro modelo de desenvolvimento de software denominada como modelo em cascata ou modelo de ciclo de vida clássico. Esse modelo é marcante por ser definido por etapas dependentes entre elas, tem como principal objetivo garantir que as etapas sejam analisadas sobre todos os aspectos, eliminando possíveis esquecimentos. Abaixo são apresentadas suas fases.

Requerimento > projeto > construção > integração > testes > implantação > manutenção.

Principais Vantagens

Uma das principais vantagens do modelo é que ele não permite pular etapas, podendo assim garantir que não haverá percalço de aplicação do modelo no decorrer do projeto.

Todas as atividades identificadas nas fases do modelo são fundamentais e estão na ordem certa;

O modelo rege por uma documentação rígida, idealmente completa em cada atividade do processo;

Principais Desvantagens

É possível visualizar partes do produto quando as fases estão avançadas e próximas ao final, veja que não há um retorno de informações entre as fases, não suporta modificações nos requisitos, não prevê a manutenção e não permite a reutilização devido ser excessivamente sincronizado e ter dependências entre as fases. No caso de acontecer um atraso em quaisquer etapas, todo o processo é afetado;

Papel do QA na metodologia

No desenvolver das fases, a equipe de qualidade atua na revisão dos documentos já criados e inicia a criação de artefatos, como: Plano de teste, Casos de teste, procedimentos de teste, entre outros.

O Time de Testes é composto na maioria das vezes por analistas de testes que tem como a principal característica criar testes como também executá-los, e geralmente a equipe tem funções definidas. Papéis como automatizadores, responsáveis por automatizar os testes, líder de teste, que gerencia os analistas de teste e testadores, o analista de teste de performance, que é responsável pelos testes de performance, entre outros, ou seja, cada um tem uma função determinada no time.

Os testes realizados pelos analistas e QA's são testes funcionais, não funcionais e “end-to-end”, que visa assegurar o sistema de uma forma mais completa, verificando se o fluxo de um software está sendo executado de acordo com o projeto, ao simular o ambiente real. Os desenvolvedores, ou alguns deles, poderiam desenvolver testes de unidade e

testes de integração. Interessante ressaltar que havia alguns percalços quando era aberto tarefas de inconsistências, visto que a visão dos times testes e desenvolvimento seria de terceirização, ou seja, não era comum uma mentalidade colaborativa.

Metodologia Ágil (Scrum)

No ano de 2010, a empresa VersionOne realizou uma pesquisa para identificar qual a metodologia é mais utilizada naquela época e chegou às seguintes conclusões:

Em disparada a metodologia Scrum é a mais utilizada, e é nela que iremos focar como metodologia ágil. Criado por dois desenvolvedores (Ken Schwaber e Jeff Sutherland), o Scrum é uma estrutura simples para trabalhar com projetos complexos, tem o objetivo principal reduzir o tempo de entrega de produtos e se ajustar a mudanças do projeto com maior facilidade durante as etapas.

Diferente da metodologia Cascata, a equipe não aguarda a entrega final do produto para avaliação do cliente, e sim faz as correções necessárias e pequenas entregas em tempos determinados, visto que o time consegue transitar por cada etapa do processo sem a necessidade de dependência.

Os times que aplicam essa abordagem são formados pelos papéis: Product Owner, é o dono do produto, ou seja, o cliente é quem dita todas as funcionalidades do programa e, esse cliente é Product Owner. Ele é quem entende do negócio. O Scrum Master, é um gestor do projeto e muito conhecedor do Scrum, é um facilitador de obstáculos em reuniões, e protege a equipe resguardando que não se comprometa com mais tarefas do que possa realizar em um determinado tempo, e o Scrum Team, é a equipe que vai desenvolver o programa, inclusive o QA está inserido nesse contexto.

Os valores do Scrum estão interligados aos valores descritos no manifesto ágil, o qual menciona quatro valores que, para gestão de projetos adaptativos são essenciais, são eles:

- Valorizar mais indivíduos e interações do que processos e ferramentas;

- Valorizar mais software em funcionamento do que documentação abrangente;
- Valorizar mais colaboração com o cliente do que negociação de contratos;
- Valorizar mais respostas às mudanças do que seguir um plano;

Papel do QA na metodologia

Em times Scrum todos são multidisciplinares, tomando boas práticas de qualidade que os tornam responsáveis pelo desenvolvimento e entrega do produto. Nesse contexto, existe um time de desenvolvimento (DEV Team) composto por engenheiros de software e são fluentes em testes, e não apenas o time, e sim a gestão tem que conhecer sobre testes. O QA é um grande convededor sobre testes de software e consegue ajudar seu time, monitorando que esteja sendo seguidas por todo o time, desde o levantamento dos primeiros requisitos do projeto, suporte aos desenvolvedores a explorar melhor os testes unitários, testes de integração, identificando possíveis riscos na revisão dos documentos, criação de testes tomo um todo, entre outros.

Há outro cenário híbrido que enquadram esses dois métodos, são empresas em fase de transição migrando do Cascata para o Ágil, então, como fica a visão da qualidade de software neste contexto?

Híbrido – Do Cascata para o Ágil

Enquanto no Cascata haviam funções bem definidas dos analistas de testes, e no Ágil onde todos são fluentes em testes, normalmente trabalhando com Squads, que em média tem cinco desenvolvedores e um ou dois QA('s), no início dessa transição o QA vai ter bem mais atribuições, visto que eles são responsáveis por testes de sistemas e de suportar desenvolvedores em testes de unidade e de integração, usuários em testes de aceitação e também os times em revisão de artefatos, e veja que ainda não foi falado de automação. Nessa fase o QA une forças para vender a ideia e filosofia de testes aos desenvolvedores, como por exemplo, nos testes unitários e de integração, trazendo assim mais responsabilidade no que foi produzido. Confesso que

essa transição não é fácil, e comumente vemos isso em empresas, mas isso é assunto para outro artigo.

Planning poker: A técnica baseada no consenso

Planning poker ou em português “Poker do planejamento”, é uma técnica baseada no consenso para estimar, é um jogo e ao mesmo tempo um exercício.

Através dessa técnica podemos estimar o esforço necessário para determinada quantidade de trabalho, tendo como base informações recolhidas do cliente, normalmente sendo, através de estórias de usuário.

O **planning poker** foi definido e nomeado pela primeira vez por James Grenning, em 2002, e mais tarde popularizado por Mike Cohn, no livro *Agile Estimating and Planning*.

De maneira resumida, no **planning poker** cada membro da equipe recebe um conjunto de cartas, com os valores de uma determinada sequência.

Em seguida, a cada estória de usuário analisada, cada membro da equipe joga uma carta com a face para baixo sobre a mesa, nela estará contido o valor numérico de pontos que o mesmo considera justo para que a estória seja concluída.

Caso haja grandes diferenças entre as cartas jogadas, os membros que jogaram as cartas de maior e menor valor explicarão suas razões e, então, com base em suas explicações, as cartas são jogadas novamente até que um consenso seja encontrado e uma estimativa seja definida.

A sequência de Fibonacci

Como citei anteriormente, as cartas possuem uma sequência de valores, que podem ser uma sequência simplificada (0, 1, 2, 3, 5, 8, 13, 20, 40, 100) ou uma sequência de Fibonacci. De acordo com a Wikipédia, a *sucessão de Fibonacci ou sequência de Fibonacci é uma sequência de números naturais, onde os dois primeiros números começam normalmente por 0 e 1, na qual, cada termo subsequente corresponde a soma dos dois anteriores*. Abaixo temos uma imagem que ilustra a lógica da sequência Fibonacci.

$$0, 1, 1, 2, \overset{+}{3}, 5, 8, 13, 21, 34, 55\dots$$

$\rightarrow 2 + 3 = 5$

A principal razão que nos leva a acreditar que a seqüência de Fibonacci é a melhor opção, é refletir na inerente incerteza que podemos ter em estimativas de itens maiores. Na imagem abaixo, podemos enxergar um conjunto de cartas que seguem a sequência Fibonacci. Você poderá encontrar várias versões diferentes de baralho para o **planning poker** através da internet, logicamente, você está livre para criar a sua própria versão e aplicá-la junto a sua equipe. Porém, como sugestão, entrego um exemplo que você poderá seguir.

?	0.5	0	1
2	3	5	8
13	21	...	

Entendendo o baralho

Você deve ter percebido que existe alguns elementos, vamos dizer, “confusos”, são eles: ?, 0, 0.5, [...] e uma xícara de café. Mas, o que eles significam e por que estão presentes no **planning poker**? Vou te explicar.

- **?** (**interrogação**): Significa que o membro não se sente confiante para atribuir um valor a tarefa;
- **0** (**zero**): Significa que a tarefa é absolutamente desnecessária e deveria ser descartada;
- **0.5** (**meio**): Significa que a tarefa necessita de um pequeno esforço para ser concluída;
- **...** (**infinito**): Significa que a tarefa é extremamente importante;
- **Xícara de café**: Significa uma pausa para refletir antes de tomar a decisão. Esta pausa é importante e deve ser respeitada quando solicitada, é muito provável que os membros não abusem dela.

Os outros números são a representação arbitrária do sentimento dos membros, quanto ao esforço necessário, para a realização da tarefa. Perceba que o **planning poker** é bastante simples e divertido, não há como um integrante se confundir ou não entender o jogo.

Planning poker em ação

Beleza, você agora já entende como é feita a sequência das cartas, conhece o significado dos elementos confusos e agora está em sua reunião de planejamento de sprint, com toda sua equipe (4 pessoas), com o ScrumMaster e com o Product Owner. Conseguiu imaginar? Show! O **planning poker** pode ser utilizado em dois momentos: na priorização de tarefas e na estimativa de esforço para concluir-las. Eu vou mostrar o cenário para uma estimativa de esforço, pois tanto para estimativa, quanto para priorização o jogo é o mesmo.

Suponha que vocês já tenham jogado um **planning poker** para a priorização de tarefas, sendo assim, já definiram quais tarefas deverão ser executadas primeiro e agora irão estimar o esforço para desenvolvimento delas. Todos (menos o ScrumMaster e o P.O.) receberão seus respectivos baralhos, cada um com os números 0, 0.5, 1, 2, 3, 5, 8, 13, 21 e os símbolos de interrogação, infinito e xícara de café. O ScrumMaster pede então que o Product Owner leia a primeira estória de usuário, podemos dizer que seria a seguinte: “Como usuário, quero acessar o site através do celular e usar todos os recursos, assim como no navegador do desktop”. O ScrumMaster pede que todos os participantes pensem um pouco sobre o esforço para desenvolver aquela estória e, em seguida, que todos mostrem suas cartas. O resultado da jogada é o seguinte:

[table caption="1° Rodada" width="300" colwidth="150|150" colalign="center|center" align="center"]

Membro, Carta do poker

Lucas,8

Ana,5

Alan,21

Carlos,3

Maria, 5

[/table]

Fica claro que Alan e Carlos fazem jogadas extremas. Nesta situação o ScrumMaster pergunta a ambos o motivo de suas escolhas, eles respondem:

Alan diz:

Teríamos que fazer um outro site para o ambiente mobile, um novo HTML, CSS, JS e integrar uma nova rotina à existente hoje. É necessário um grande esforço no design e na programação para que o site se torne mobile. Temos que considerar os plugins utilizados hoje e encontrar similares no mercado para customização.

Carlos diz:

Na verdade, poderíamos utilizar a estrutura já existente hoje e apenas ajustar o CSS e alguns elementos HTML, assim manteríamos tudo junto e o esforço para manutenção seria menor. Devemos levar em consideração que o usuário deseja ter os mesmos recursos, ou seja, seria uma replica. O site possui muitas páginas dinâmicas e isso faz com que o ajuste em uma, replique em várias.

Acontece então uma breve discussão e uma nova rodada é realizada. O resultado é o seguinte:

[table caption="2º Rodada" width="300" colwidth="150|150" colalign="center|center" align="center"]

Membro, Carta do poker

Lucas,8

Ana,5

Alan,8

Carlos,5

Maria,5

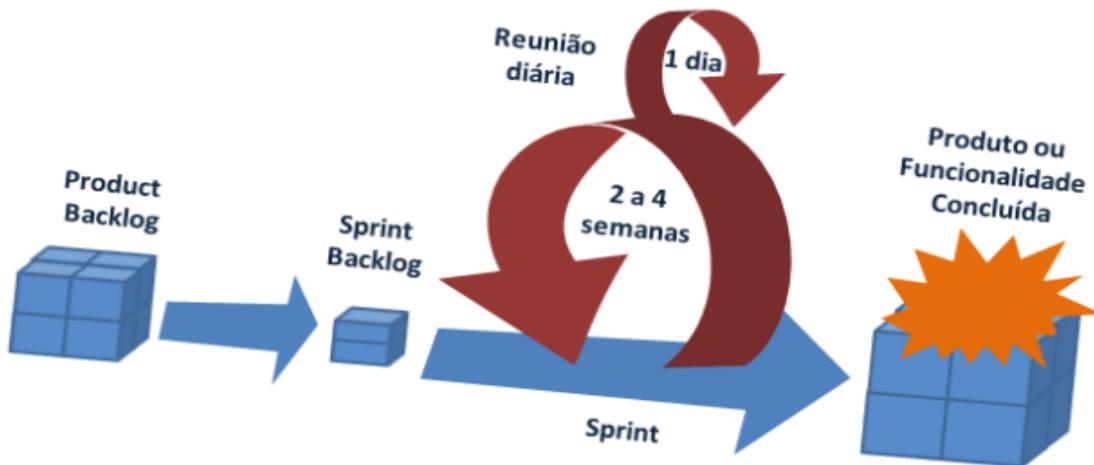
[/table]

Perceba que os valores estão mais uniformes, com esse cenário você pode assumir algumas opções, são elas:

1. Continuar a incentivar a discussão até que um consenso geral entre os membros seja obtido;
2. Fazer uma média dos valores, levando em consideração a proximidade entre eles;
3. Como os valores estão próximos, assumir o maior valor.

Em nosso caso optamos por escolher o maior valor, sendo assim, o número de pontos para essa tarefa (Story Points) seria de 8. Esse ciclo de jogadas é repetido em cada tarefa do sprint backlog (pode ocorrer também no product backlog) até que todas sejam concluídas. Com o tempo você facilmente saberá definir quantos Story Points conseguirá assumir em cada sprint, ajustando suas estimativas quanto ao desenrolar do projeto.

Método SCRUM — Um resumo de tudo o que você precisa saber



Scrum é um framework utilizado para gestão dinâmica de projetos, sendo muitas vezes aplicado para o desenvolvimento ágil de um software. É um processo **iterativo e incremental**, que possui **3 pilares centrais**:

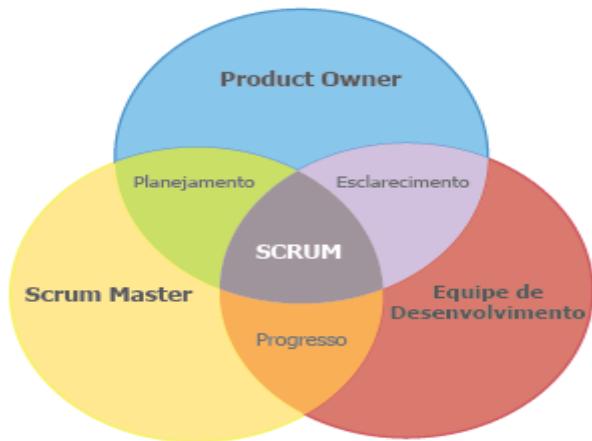
TRANSPARÊNCIA dos processos, dos requisitos de entrega e status. Todos os aspectos significativos do processo como um todo devem estar visíveis e alinhados com todos os responsáveis pelos resultados.

INSPEÇÃO constante de tudo o que está sendo feito.

ADAPTAÇÃO tanto do processo, quanto do produto, que podem sofrer mudanças que necessitam de adaptação. Também é importante adaptar o SCRUM para a realidade e cultura da empresa.

São **PRÁTICAS FUNDAMENTAIS** do SCRUM:

PAPÉIS



SCRUM MASTER: responsável por ajudar todo o time a entender e manter vivos os princípios e práticas do *SCRUM* no dia-a-dia. É como um coach, um facilitador para o trabalho, remove impedimentos, refina itens da próxima *Sprint* junto ao *Product Owner*.

PRODUCT OWNER: responsável pelos poderes de liderança sobre o produto. Decide quais recursos serão construídos, qual a ordem de prioridade de produção no *PRODUCT BACKLOG*.

TEAM: equipe desenvolvedora do projeto. É o *team* que define como as coisas serão feitas e quais e quantas tarefas são possíveis de entregar. O *team* se auto organiza para atingir as metas estabelecidas pelo *Product Owner*.

ARTEFATOS



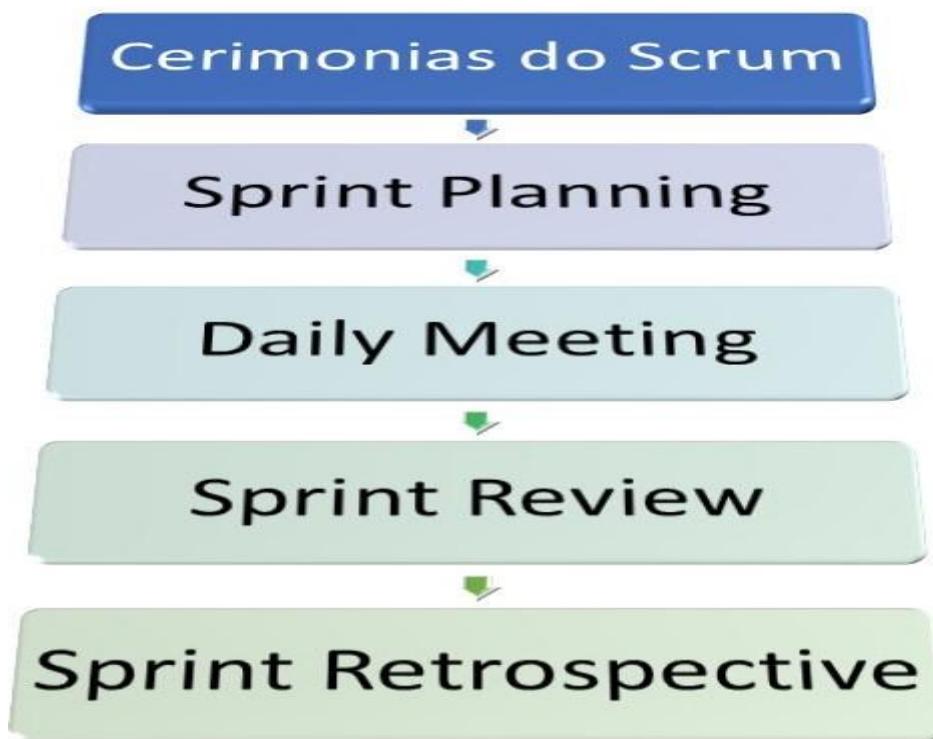
PRODUCT BACKLOG: o *Product Owner* descreve o que precisa e onde deve chegar através de *histórias*, que levam em conta 7 dimensões do produto: **ATORES, INTERFACES, AÇÕES, DADOS, REGRAS DE NEGÓCIO, AMBIENTE e QUALIDADE**. Essa visão é desmembrada em todas as funcionalidades que serão necessárias e irão compor o *Product Backlog*.

SPRINTS: As funcionalidades são organizadas por ordem de prioridade e o projeto é planejado em *Sprints* que são períodos de tempo onde os itens selecionados no *Product Backlog* serão construídos e entregues. As *Sprints* são planejadas respeitando o *time-boxed*: precisam ter a mesma duração fixa que varia entre 1 e 4 semanas, sendo mais utilizado o padrão de 2 semanas.

SPRINT BACKLOG: antes de cada *Sprint* começar é feita uma reunião de planejamento (*Sprint Planing*), onde é criado o *Sprint Backlog*, que leva em consideração as capacidades e velocidade de entrega do *team* para definir quantas histórias poderão ser construídas e entregues dentro de uma *Sprint*.

INCREMENTO / ENTREGA: ao término de uma *Sprint*, deve ser entregue um *Incremento* do produto (no caso de desenvolvimento, por exemplo, é importante que seja entregue uma parte do *software* funcionando, ainda que não esteja finalizada). Após a entrega do incremento, é realizada uma *Sprint Review*, momento em que o *Product Owner* verifica se serão inseridas mudanças ou alterações no produto. Essas mudanças são inseridas no *backlog*, também por prioridades.

CERIMÔNIAS



SPRINT PLANNING: reunião realizada antes do início de uma *Sprint*, onde é construído o *Sprint Backlog*.

DAILY SCRUM: reunião diária que dura em média 15 minutos, em que cada membro do team deve levar em conta 3 perguntas básicas a serem respondidas:

1 — O que eu fiz ontem?

2 — O que eu farei hoje?

3 — Existe algum impedimento para o andamento do que tenho que fazer?

Nesse momento, a *Transparência* é extremamente importante para que fique muito claro quais são os problemas e dificuldades para que soluções possam ser encontradas.

SPRINT REVIEW: reunião que acontece após o término de cada *Sprint* para validar e adaptar o produto que está sendo construído. É nesse momento em que o *Product Owner* sugere mudanças e alterações que serão inseridas no *product backlog* por ordem de prioridade.

RETROSPECTIVA: reunião final, em que todos podem e devem expor seu *feedback* do processo, verificando necessidades de adaptação a partir do que aconteceu de positivo e de negativo.

FERRAMENTAS UTILIZADAS

Existem algumas ferramentas que são utilizadas no *Scrum* para facilitar e organizar melhor os processos no dia-a-dia do *team*. Conheça duas delas:

BURNDOWN CHART

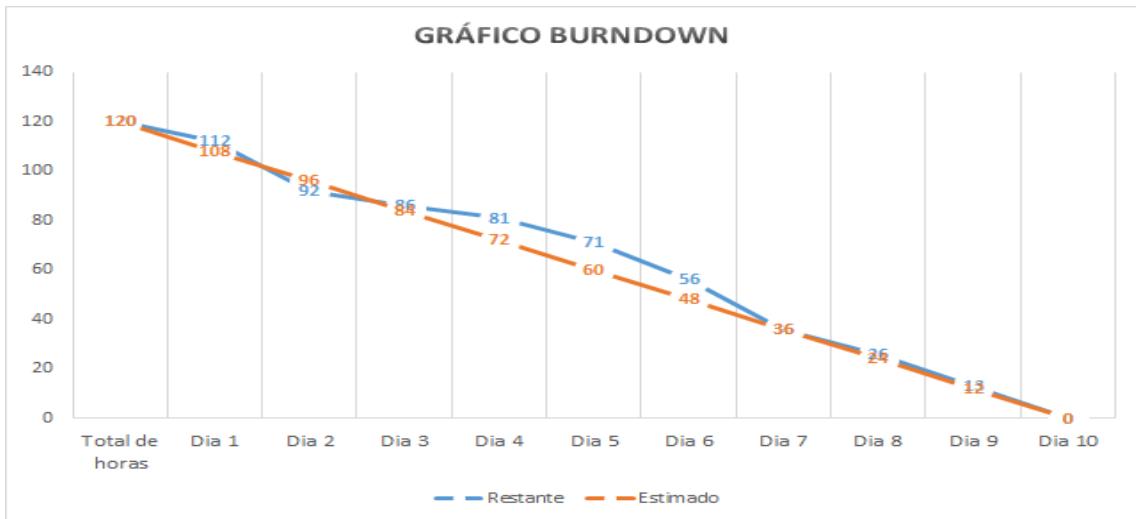
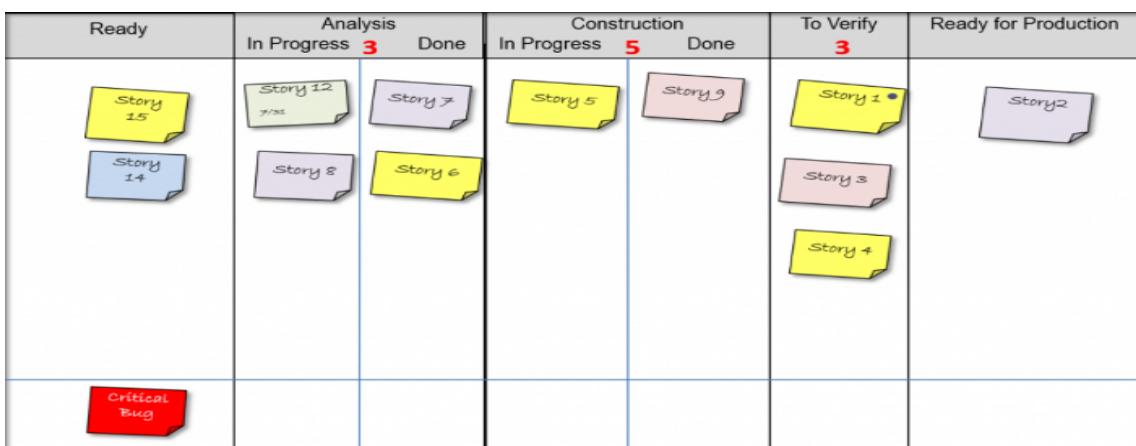


Gráfico que relaciona os itens a serem realizados com o tempo de entrega. Existe uma linha mestra que é o padrão desejado e sobre ela é possível acompanhar o andamento do processo através do fornecimento de informações sobre os itens que foram entregues pelo time e o tempo que foi levado para isso.

KANBAN BOARD



Quadro onde é possível visualizar o fluxo de trabalho que está sendo feito. Pode ser realizado através de um software (por exemplo, o

Trello) ou mesmo um quadro de parede com *post-its* que representam as histórias e funcionalidades do *product back log*. São listados os itens que precisam ser executados, que estão sendo executados e que estão prontos. Também são listados os impedimentos e outras informações.

Concluindo...

O *Scrum* é apenas um dos *frameworks* baseados em metodologias ágeis, existem muitos outros e suas aplicações são ilimitadas.

Ao utilizar o *Scrum*, podemos otimizar nossos processos ganhando cada vez mais agilidade para nos adaptarmos à mudanças e para construirmos prazos mais justos que levem em consideração não só as necessidades do cliente, mas principalmente a qualidade e viabilidade do desenvolvimento do produto final.

Método Kanban como usar?



O método Kanban foi concebido como um caminho alternativo à agilidade organizacional. Ao invés de grandes mudanças, o Kanban

propõe uma abordagem evolucionária, onde pequenas melhorias são integradas na prestação de um serviço ou desenvolvimento de um produto.

Nascido dentro do Sistema Toyota de Produção (TPS), o kanban (com k minúsculo) era um cartão utilizado para sinalizar conclusão de etapas do processo e tornar o fluxo puxado. Mais tarde, David J. Anderson criou o método Kanban (com K maiúsculo) e popularizou o seu uso no desenvolvimento de produtos e serviços. Neste guia detalharemos o funcionamento deste

segundo.

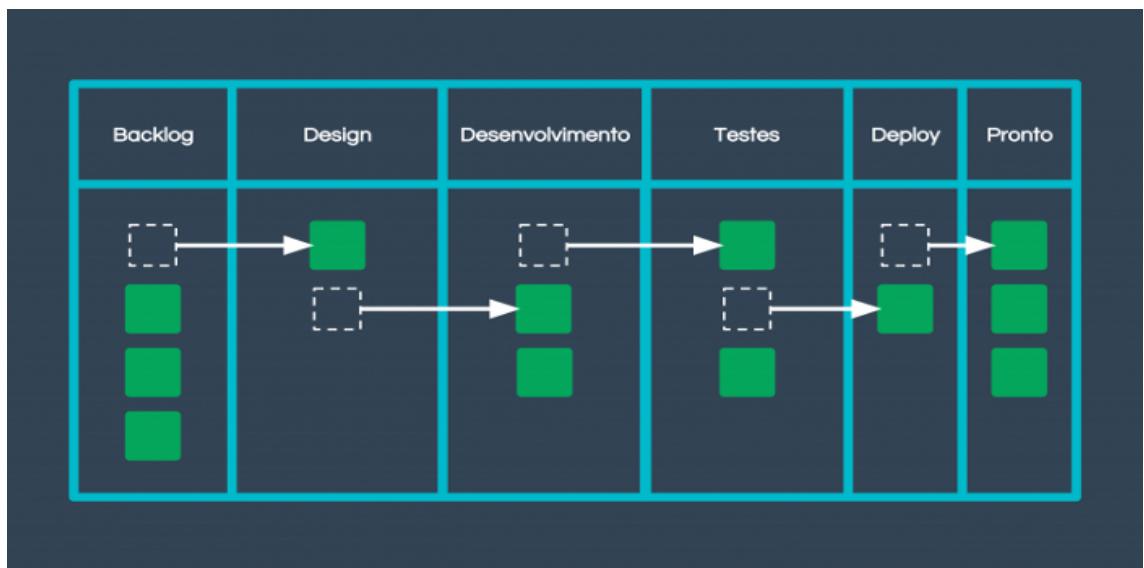
1. Entendendo um sistema Kanban
2. Sistema empurrado
3. Sistema puxado
4. Os princípios básicos do método Kanban
5. As 6 práticas gerais de um sistema Kanban
6. Quando usar o método Kanban?
7. Vantagens sobre o Scrum
8. Como implementar um sistema Kanban
9. Elementos de um sistema Kanban
10. Métricas de um sistema Kanban
11. Método Kanban & Autogestão (O2)
12. E tem muito mais que ainda não está aqui

ENTENDENDO UM SISTEMA KANBAN

Um sistema Kanban é composto por um fluxo de valor, onde unidades de trabalho trafegam da esquerda para a direita. Cada etapa do processo adiciona mais valor ao item, sendo que quando ele chega ao final do processo, ele está “concluído”. Esse fluxo de valor pode ser o

desenvolvimento de um software, a prestação de um serviço (como atendimento ao cliente) ou até a criação de um produto físico.

Vamos supor que fazemos parte de um time de desenvolvimento de um aplicativo. O nosso processo atual tem algumas etapas, que podemos mapear na forma de colunas “Backlog”, “Design”, “Desenvolvimento”, “Testes”, “Deploy” (implantação) e “Pronto”. Neste quadro Kanban, as unidades de trabalho representam coisas que geram algum benefício ao cliente/usuário final, como novas funcionalidades, correção de defeitos ou melhorias na interface do produto. O valor é gerado somente quando um item alcança a última etapa:



Apesar de conter etapas sequenciais e cartões, isso ainda não é um sistema Kanban. Vamos fazer mais duas mudanças para tornar o fluxo puxado:

1. Dividir as colunas intermediárias em dois estágios: “fazendo” e “pronto”;
2. Adicionar limites de trabalho em progresso

Feitas essas mudanças, o nosso quadro ficaria assim:

Backlog	Design	Desenvolvimento	Testes	Deploy	Pronto
3	1	3	2	1	
	Fazendo Pronto	Fazendo Pronto	Fazendo Pronto		
					

Vamos analisar como isso funcionaria na prática. Supondo que você é o especialista em Design do time e que você está trabalhando em uma funcionalidade. Após você concluir a sua atividade, você movimenta o cartão para a etapa “Pronto” da coluna “Design”. Se um desenvolvedor (que atua na etapa “Desenvolvimento”) está livre, ele pode perceber que você sinalizou a conclusão da sua parte e então “puxar” o cartão para a etapa “Fazendo” da coluna “Desenvolvimento”.

Repare que o limite “1” da coluna “Design” é válido tanto para “Fazendo” quanto “Pronto”. No caso acima, temos um item que foi concluído pelo Designer, mas ainda não foi puxado. O que o Designer faz nesse caso? Nada. Ele apenas aguarda alguém puxar o item para Desenvolvimento. Nesse caso o slot seria liberado e ele poderia trabalhar em outro item.

Claro que na prática se não houver slot disponível o membro do time não vai ficar simplesmente parado. Em um sistema Kanban, estimulamos as pessoas a olharem para o fluxo como um todo e buscarem formas de aumentar a vazão. Pode ser que o Designer possa ajudar alguém no Desenvolvimento. Pode ser que ele use esse tempo para pensar em melhorias. O que importa é não ficar simplesmente trabalhando na própria função, pois isso aumentaria o estoque e prejudicaria o fluxo.

Parece contra intuitivo, mas a teoria das filas comprova. A ociosidade no fluxo aumenta a vazão. Para entender os porquês, você vai precisar estudar um pouco de matemática (esse livro é um bom começo).

Como os itens são puxados apenas quando há capacidade disponível, raramente há sobrecarga. Isso é que o chamamos de sistema puxado. Ele é um contraste à forma tradicional de trabalhar, que chamamos de sistema empurrado. Vamos entender a diferença entre os dois para compreender melhor um sistema Kanban.

SISTEMA EMPURRADO

Um sistema empurrado é aquele em que a produção é baseada na demanda, sem respeito à capacidade do sistema. Em geral, no paradigma empurrado tenta-se produzir o máximo possível e em grandes lotes, sem considerar a real necessidade dos clientes. Neste tipo de abordagem, a estratégia de marketing e vendas também é focada em vender e “empurrar” os produtos para o cliente.

No nosso exemplo do time de desenvolvimento de um aplicativo, um gestor poderia atribuir diversas tarefas aos membros do time, exigindo

que todos se mantivessem ocupados e trabalhando 100% do tempo, cada um na sua função. Além disso, é provável que esse time desenvolvesse um grande número de funcionalidades inúteis e que não gerasse valor para o cliente final. Afinal, todos estariam preocupados em fazer bem a sua função, e não em entregar valor para o cliente.

Os principais efeitos colaterais de um sistema empurrado são sobrecarga, demoras nas entregas, grandes lotes e burnout das pessoas.

SISTEMA PUXADO

Um sistema puxado é aquele em que os participantes “puxam” o trabalho quando há capacidade disponível para executá-lo. Isso é possível quando atribuímos limites para as unidades de trabalho em progresso. Um sistema puxado nunca irá sofrer com sobrecarga se os limites forem estabelecidos corretamente. Nesse paradigma busca-se atingir um passo sustentável, ou seja, um equilíbrio entre a capacidade do time e o que é demandado dele.

O método Kanban é apenas uma forma de construir um sistema puxado. Outras variações também existem, como CONWIP, DBR e CapWIP.

Agora que conhecemos a diferença entre empurrado e puxado, vamos entender um pouco mais sobre os princípios e propriedades básicas de um sistema Kanban.

OS PRINCÍPIOS BÁSICOS DO MÉTODO KANBAN

Em seu livro Kanban: successful evolutionary change for your technology business, David J. Anderson descreve os princípios básicos do método Kanban:

Comece com o que você tem hoje. O método Kanban propõe uma abordagem evolucionária e incremental. Por mais que você esteja muito insatisfeito com a forma como as coisas são feitas no processo atual, não mude tudo logo no início. Fazer grandes mudanças pode aumentar a resistência das pessoas, além de ser muito arriscado para a organização.

Busque mudanças incrementais e evolucionárias. Depois de partir do seu processo atual, busque pequenas mudanças. Formule hipóteses com base na sua observação do comportamento do sistema. Seja curioso e faça experimentos.

Vamos supor que o seu sistema possui uma coluna onde um grande documento é produzido que detalha o que deve ser feito. Talvez você desconfie que o nível de detalhe seja grande demais e que isso é um provável desperdício. Formule a hipótese: Se nós simplificarmos o documento, teremos uma redução do Lead Time do processo. Você pode estar certo ou não. Faça um experimento, meça os resultados e compare.

Respeite o processo atual, papéis, responsabilidades e títulos. É provável que na organização em que você está implementando o método Kanban existam cargos e autoridades definidas. Talvez essa estrutura esteja atrapalhando o fluxo, na sua visão. Mas ao mesmo tempo, é também muito provável que boa parte dos processos atuais simplesmente funcione. Por isso é importante respeitar o que já está posto e perseguir a melhoria contínua a partir disso.

AS 6 PRÁTICAS GERAIS DE UM SISTEMA KANBAN

David J. Anderson descreve 6 práticas gerais das implementações de sucesso do método Kanban. Podemos dizer que sem elas, você ainda não está fazendo “Kanban”. Por isso que dissemos que o quadro inicial (que não tinha limites de trabalho em progresso) ainda não poderia ser considerado um sistema Kanban: ele falhava em uma das 6 práticas gerais.

As 6 práticas são:

Visualize o fluxo de trabalho. Você não pode gerenciar o que não pode ver! E se tratando de trabalho do conhecimento, esta afirmação é ainda mais verdadeira. Na Toyota, o estoque excessivo de carros poderia ser facilmente visualizado, já que um carro é um produto físico. Mas e todas as funcionalidades de um sistema que foram começadas, mas não terminadas? Ou todos os tickets de atendimento que foram abertos, lidos, mas não respondidos? O trabalho do conhecimento é intangível. É por isso que precisamos estabelecer formas tangíveis de visualizar o fluxo e as unidades de trabalho.

Limite o trabalho em progresso. Se você não estabelecer limites para o trabalho em andamento, é muito provável que você ainda esteja operando um sistema empurrado. Em geral, todas as etapas em um fluxo são limitadas, com exceção da última. Estabelecer limites vai impedir o acúmulo de estoque e também vai manter o Lead Time estável. Em geral, quanto mais itens houverem em andamento, maior é o tempo de entrega.

Gerencie o fluxo. No Kanban, focamos os esforços na otimização do sistema como um todo. A gestão tradicional empurrada foca em manter todas as pessoas ocupadas e gerenciar o trabalho delas. Isso

não é feito no método Kanban. Se tentarmos otimizar uma parte do sistema (o trabalho de um desenvolvedor), isso provavelmente levará a uma situação sub

ótima globalmente. Esse fenômeno chama-se otimização local. Leia mais sobre como isso acontece com as metas.

Torne as políticas de processo explícitas. Ao trabalhar com um fluxo Kanban, você vai perceber que muitas regras de processos, papéis e definições não são claras para os participantes do fluxo. Vai surgir muita confusão sobre o significado de uma determinada coluna do fluxo, por exemplo. É importante que você aproveite essas oportunidades para esclarecer as políticas do processo e torná-las explícitas. Isso significa registrá-las em algum local acessível para todos. Parte importante de jogar o jogo é conhecer as regras.

Implemente ciclos de feedback. Nenhum processo pode evoluir sem ciclos de feedback. David J. Anderson propõe alguns rituais específicos para retroalimentar o sistema e permitir com que os participantes se adaptem comparando a situação atual do processo com as expectativas das partes interessadas.

Melhore colaborativamente. No final, as limitações de trabalho em progresso irão introduzir desconfortos e iniciar conversas sobre o fluxo. O que se busca em um sistema Kanban é um fluxo suave e constante. No entanto, não é isso que vai acontecer no início. É importante estabelecer um processo de melhoria colaborativa, onde o grupo constrói um entendimento compartilhado sobre os problemas encontrados e a partir daí propõe experimentos para melhorar o fluxo.

QUANDO USAR O MÉTODO KANBAN?

Praticamente qualquer processo de desenvolvimento ou prestação de serviço pode ser melhorado com o método Kanban. No entanto, quanto maior e mais complexa a cadeia de valor desse produto/serviço, maior é o benefício em aplicá-lo.

Cadeias longas geralmente possuem muitas pessoas envolvidas, handovers e fontes de desperdício. Isso significa maiores oportunidades de melhoria. Por outro lado, se o processo for extremamente simples, provavelmente o esforço de aplicar o método é maior do que o benefício obtido.

VANTAGENS SOBRE O SCRUM

Na comunidade ágil, o Scrum é o principal “concorrente” do Kanban. A palavra concorrente está entre aspas, porque a comparação não é justa. Primeiro, porque são coisas diferentes. O Scrum é um framework, enquanto que o Kanban é um método. O primeiro é muito mais prescritivo e o segundo muito mais flexível.

Deixando essa diferença de lado, quando aplicados em desenvolvimento de produtos digitais, o Kanban tem as seguintes vantagens sobre o Scrum:

Múltiplos focos. O método Kanban permite múltiplos focos de atuação. Se o seu time dá manutenção em um produto legado, e ao mesmo tempo desenvolve um novo, isso não é

problema para o Kanban. Já o Scrum exige um time dedicado para o desenvolvimento de um único produto, com um único Backlog.

Entregas constantes. O método Kanban trabalha com fluxo contínuo, enquanto que o Scrum usa Sprints com timebox. Ambas as abordagens permitem entregas constantes, mas o Kanban permite ciclos ainda menores que o Scrum.

Mais evolutivo. Por ser mais prescritivo, o Scrum pode exigir que a sua organização mude radicalmente. Por exemplo, um dos requisitos do time de desenvolvimento no Scrum é que ele seja pequeno, multidisciplinar e auto-organizado. Dependendo da sua estrutura organizacional, essa mudança pode exigir uma grande reengenharia. Já o método Kanban é bem enfático na melhoria incremental e diz: comece onde você está.

COMO IMPLEMENTAR UM SISTEMA KANBAN

Os princípios e as propriedades que vimos na seção anterior fornecem um caminho de implementação interessante. Segui-los em ordem já é um bom começo.

Por exemplo, o primeiro princípio é: comece de onde você está. Essa é uma boa dica. Não faça mudanças abruptas na implementação de um sistema Kanban logo de cara. Podemos ver também que a propriedade limite o trabalho em progresso vem depois de visualize o fluxo de trabalho. Isso também nos mostra que é importante primeiro enxergar o que está acontecendo na organização antes de sair aplicando as restrições, por mais que elas sejam “básicas”.

ELEMENTOS DE UM SISTEMA KANBAN

Um sistema Kanban possui diversos elementos e práticas atreladas.

Nesta seção, veremos os mais comuns. E o primeiro de todos é...

QUADRO KANBAN

Sim, o quadro. Ele é uma forma simples de visualizar o andamento do fluxo e o progresso dos itens de trabalho, mas não a única. No seu livro, David J. Anderson descreve alguns exemplos de sistemas Kanban sem uso de um quadro

Atenção: ter um quadro Kanban não significa ter um sistema Kanban.

Por exemplo, o uso de um quadro de tarefas (normalmente referido como quadro Kanban), é muito comum em times ágeis que praticam Scrum. Isso não implica necessariamente que há um fluxo puxado, com limites de trabalho em progresso e um processo de melhoria contínua estabelecido (o método Kanban).

CARTÕES

Dentro do quadro, temos cartões que simbolizam unidades de trabalho. Ao usá-los em trabalhos do conhecimento, conseguimos tornar visível o “estoque”, que é normalmente intangível.

No método Kanban é importante visualizarmos o trabalho em progresso, porque ele é desperdício em potencial. Por exemplo, todas as telas que

um designer desenhou, mas que ainda não foram implementadas no produto, podem se tornar “lixo” se a empresa mudar a sua estratégia de atuação e aquilo perder prioridade. Trabalho em andamento é dinheiro que já foi investido, mas que ainda não trouxe um retorno para a organização.

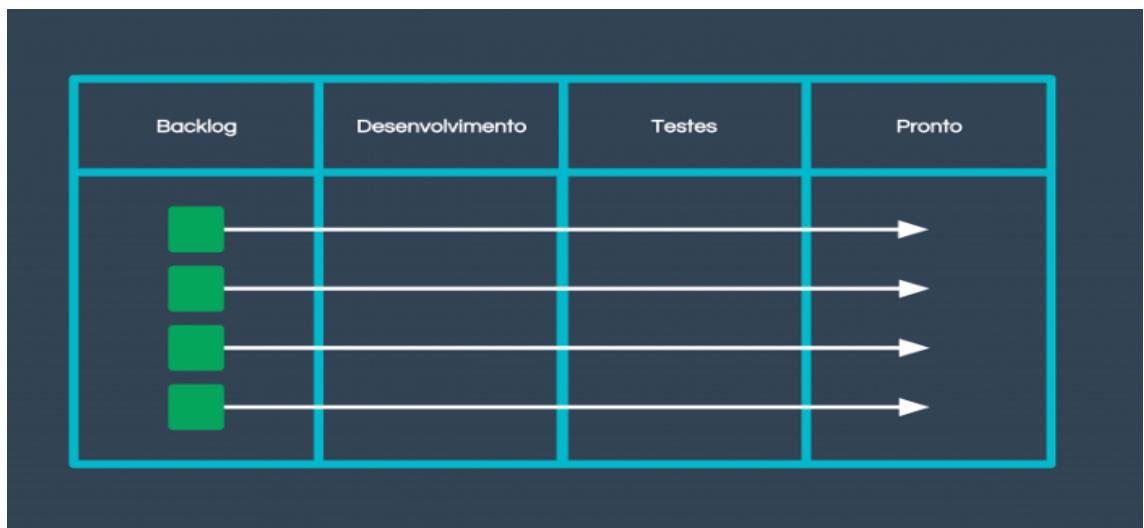
ATRIBUTOS DOS CARTÕES

Normalmente um cartão em um quadro Kanban possui alguns atributos, além do nome ou número de identificação.

O atributo mais comum é o tipo do item, que é normalmente classificado como valor, melhoria ou falha. No desenvolvimento de software, podemos usar tipos como nova funcionalidade, melhoria de interface e defeito, por exemplo. Os tipos de item permitem analisarmos as métricas de forma segmentada, além de nos darem uma visão mais clara sobre a composição do fluxo.

Os tipos de item são frequentemente representados usando um código de cores. Veja esse vídeo de exemplo:

COLUNAS DO QUADRO



Em um quadro Kanban, as colunas são normalmente utilizadas para representar as fases que um item de valor percorre, até ser concluído. Quanto mais a direita, mais valor e tempo foi investido.

Ao longo da sua jornada com o método Kanban, talvez você se depare com uma discussão se um cartão “deve voltar ou não” no fluxo. Imagine um cartão de uma funcionalidade X que está no fluxo acima. Vamos supor que ele está em “Testes”, mas o testador descobre que há um erro na funcionalidade que deve ser corrigido pelo desenvolvedor. Uma linha de pensamento defende que o item deve voltar para o estado que melhor representa. Outra diz que não, porque essas etapas não simbolizam “com quem está”, mas em qual estágio de valor o item se encontra.

Pessoalmente, prefiro a segunda opção por uma questão prática. Voltar o cartão tende a criar uma complicação no cálculo dos limites de trabalho em progresso. Veja esse vídeo do Buzon para mais detalhes.

RAIAS DO QUADRO

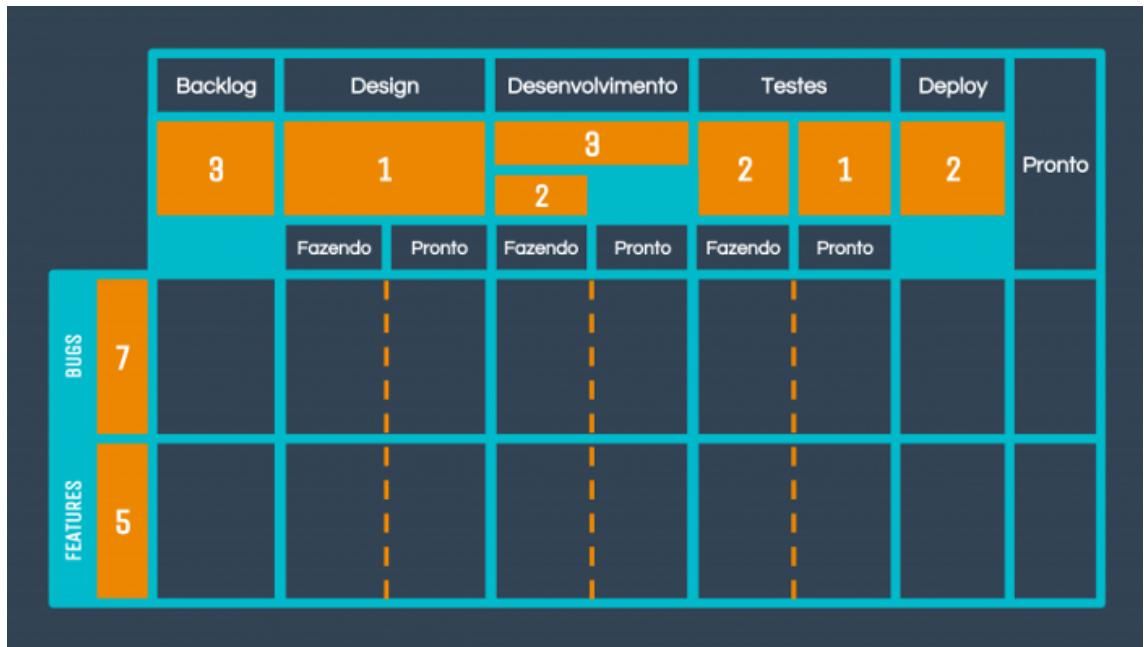
Além das colunas, que são divisões horizontais, muitos quadros possuem “raias”, que são o equivalente vertical. As raias costumam ser usadas para destacar visualmente uma classe de serviço ou para alojar capacidade por tipo de item.



No exemplo acima, utilizamos uma raia especial para destacar a classe de serviço urgente.

LIMITES DE TRABALHO EM PROGRESSO

Na introdução falamos sobre a importância dos limites de trabalho em progresso. Em termos de como montar o seu quadro, os limites podem ser estabelecidos por coluna, por subcoluna ou por raia. O quadro abaixo contém todos essas possibilidades.



A coluna “Design” possui um limite total de 1. Ou seja, podemos ter:

- 1 item em “Fazendo” e 0 itens em “Pronto”
- 0 itens em “Fazendo” e 1 item em “Pronto”

A coluna “Desenvolvimento” possui um limite total de 3 e um limite de 2 itens para a subcoluna “Fazendo”. A subcoluna “Pronto” não possui especificação de limite. Os seguintes cenários são permitidos:

- 2 itens em “Fazendo” e 1 item em “Pronto”
- 1 item em “Fazendo” e 2 itens em “Pronto”
- 0 itens em “Fazendo” e 3 itens em “Pronto”

Nessa forma de representação, tanto os limites para a coluna quanto a subcoluna devem ser respeitados simultaneamente. Ou seja, não poderíamos ter 3 itens em “Fazendo” (o limite da subcoluna é 2) e nem 2 em “Fazendo” e 2 em “Pronto” (o limite total é 3).

Note também que existe uma alocação por capacidade (falaremos mais abaixo), com limites especificados para as raias “Bugs” e “Features”.

Nesse caso os limites se aplicam a toda a linha. Ou seja, não poderemos ter mais do que 7 itens em toda a linha de bugs (independentemente da coluna) e 5 itens em features. A soma dos limites das linhas ($5 + 7 = 12$) deve ser igual à soma dos limites das colunas ($3 + 1 + 3 + 2 + 1 + 2 = 12$).

POLÍTICAS EXPLÍCITAS

Parte importante do processo de implementação de Kanban é tornar as regras do fluxo conhecidas por todos. Isso envolve explicitar muitas políticas que são normalmente implícitas nas organizações. Por explicitar, queremos dizer registrar (de forma escrita) e confirmar a compreensão com todos os envolvidos.

Esse processo dá a possibilidade da construção de um entendimento compartilhado sobre as restrições do sistema. Isso é muito importante, já que queremos envolver todos no processo de melhoria.

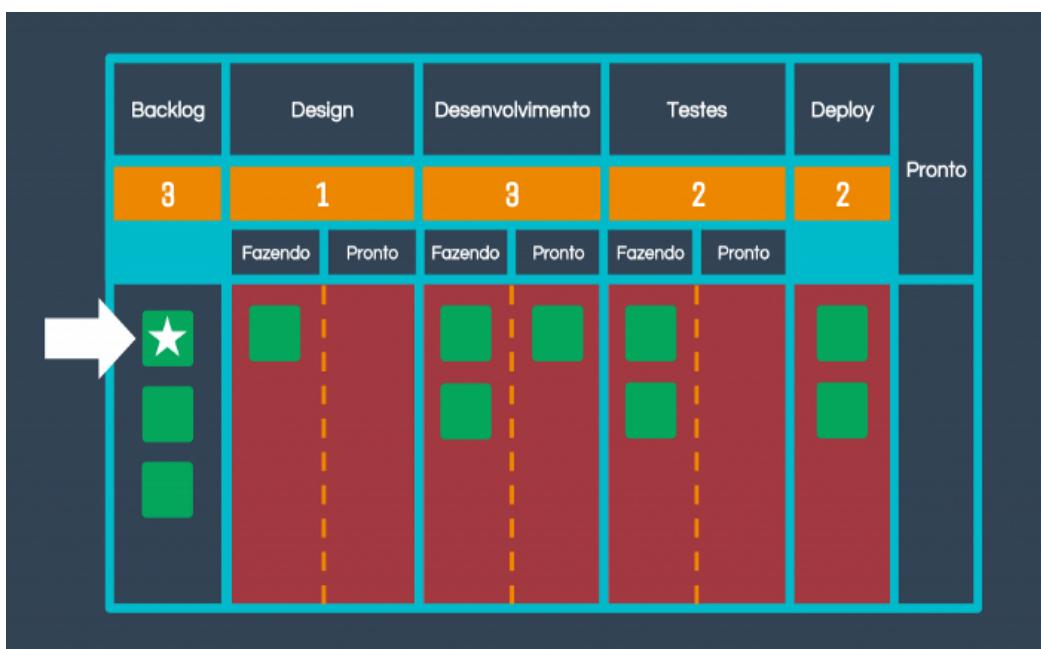
Não há um formato específico para escrita de políticas no Kanban, mas é possível trabalhar com os elementos papéis e restrições do O2 (mais informações na seção sobre autogestão).

Aqui vêm alguns exemplos de políticas explícitas:

1. Somente o Gerente de Produto pode promover um item de normal para urgente.
2. Os limites das raias e das colunas devem ser respeitados simultaneamente.
3. Se houverem itens urgentes, puxe-os antes dos normais.

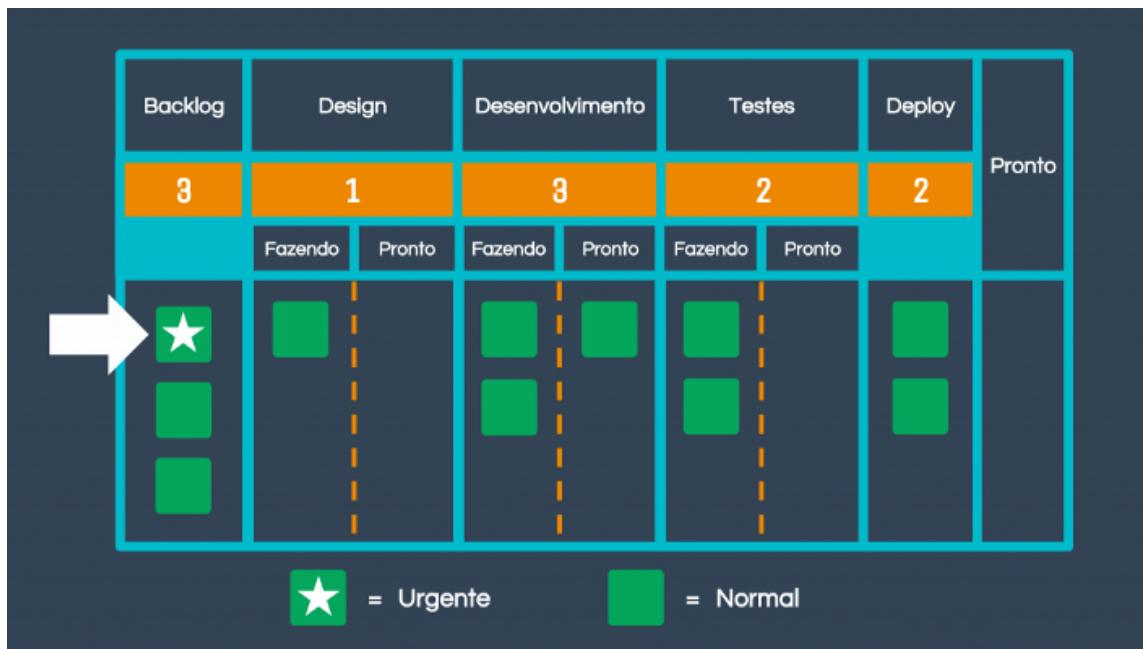
CLASSES DE SERVIÇO

Imagine a seguinte situação: o seu sistema Kanban está operando em capacidade máxima (todas as colunas atingindo os limites de trabalho em progresso) e de repente aparece um novo item com alto custo de oportunidade. Pode ser um defeito crítico, que se não for corrigido a tempo pode causar um grande prejuízo à organização. Ou pode ser uma nova funcionalidade, que se lançada antes do Natal (data de entrega), pode trazer um aumento de 20% na receita.



Se o seu fluxo está cheio, o que você faz? Mesmo que você coloque ele em primeiro lugar na sua fila de entrada, é provável que ele demore muito tempo para ser concluído. Você também não pode simplesmente aumentar os limites, porque isso iria contra as políticas já definidas.

Itens como os descritos acima normalmente aparecem nas piores horas, por isso ter uma forma de responder rapidamente é importante. É justamente essa a função das classes de serviço. Com elas, podemos oferecer tratativas diferentes a itens com essas características.



No quadro acima, temos uma marcação (a estrela) no item que define que ele possui a classe de serviço urgente. Como parte das políticas explícitas, poderíamos ter uma regra em que itens urgentes devem ser puxados primeiro e que podem exceder o limite de trabalho em progresso do sistema inteiro em 1. Poderíamos também incluir uma regra de que é possível ter no máximo 1 item urgente por vez, para não afetarmos muito o tempo de entrega dos outros itens.

Podemos também criar uma raia própria para as classes de serviço, para destacá-las visualmente:



Agora que já conhecemos os elementos básicos de um sistema Kanban, vamos falar de métricas.

MÉTRICAS DE UM SISTEMA KANBAN

Na seção sobre as 6 práticas gerais do método Kanban, vimos que o foco deve ser em gerenciar o fluxo, não as pessoas. Só podemos melhorar continuamente se pudermos melhor observar o que acontece no processo atual. Para isso, utilizamos algumas métricas e diagramas. A seguir estão as mais comuns e importantes métricas de um sistema Kanban.

Lead Time.

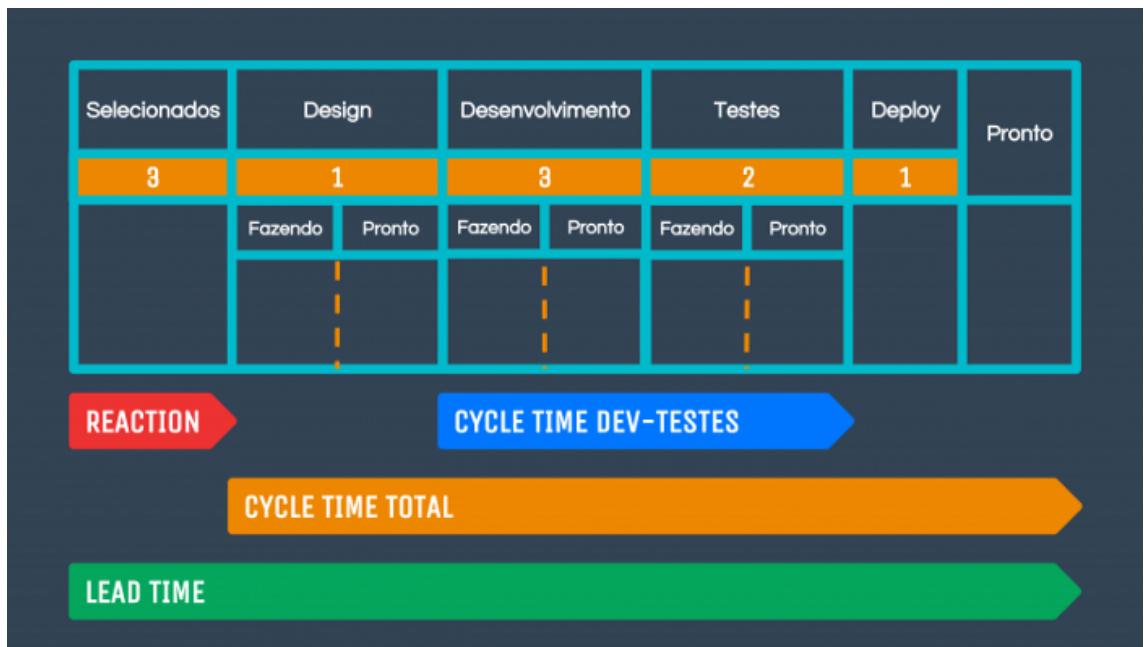
É o tempo decorrido desde que a demanda é registrada (item de trabalho) até a entrega final. No exemplo de desenvolvimento do

aplicativo, o lead time de um item seria a diferença de tempo entre o momento que ele entra na coluna “Selecionados” e que ele chega na coluna de “Pronto” final. Onde é o “início” e o “fim” do seu fluxo varia de acordo com o contexto. Em desenvolvimento de produtos, geralmente é medido desde a concepção da ideia até o momento em que o cliente final pode usufruir do seu benefício.

Esta é uma das principais métricas de um sistema Kanban. Em geral, buscamos os menores Lead Times possíveis, para entregar rápido e obter feedback o quanto antes.

Cycle Time e Reaction Time. O tempo de reação e ciclo são complementos ao lead time. Perceba que no fluxo abaixo há uma fila de entrada chamada “Selecionados”. Ali são colocados um lote de itens a serem consumidos ao longo dos próximos dias. O tempo de reação é a diferença entre o momento que você começa a capturar o lead time (nesse fluxo, o momento em que entra em “Selecionados”) até o momento em que o item é puxado pela próxima coluna. Quanto maior o seu lote de reabastecimento (limite da coluna “Selecionados”), mais alto é o tempo de reação. É interessante acompanhar essa métrica para entender se os itens

estão ficando muito tempo “parados” na fila de entrada. A diferença entre o tempo de reação e o lead time é tempo de ciclo. Veja abaixo:



Além disso, podemos medir o tempo de ciclo de um conjunto de colunas específicas, como indicado na figura (cycle time dev-testes).

Vazão (throughput). Outra métrica importante é o throughput, ou vazão. Você pode colocar um limite de apenas um item por vez no seu fluxo. Isso tornaria o seu lead time muito pequeno, mas às custas dessa outra métrica importante, que chamamos de vazão. A vazão corresponde à quantidade de itens finalizados em um determinado período (como uma semana). Em geral, buscamos obter a maior vazão com o menor lead time possível. Por isso é importante acompanhar essa métrica juntamente com o lead time.

O método Kanban não possui um ciclo pré-definido (como o Scrum possui a Sprint), pois o fluxo corre continuamente. Ainda assim, é uma prática comum definir um período de coleta das métricas, onde você pode também medir a vazão. Por exemplo, vamos supor que você quer medir tudo a cada duas semanas. Ao final desse período, você poderia contabilizar a quantidade de itens na coluna “Pronto”, calcular o lead time médio do período e assim por diante.

Composição do fluxo. Como vimos nos elementos de um sistema Kanban, você pode designar diferentes “tipos” para os itens do seu quadro. O mais comum é analisar a quantidade de “novos itens de valor” (funcionalidades) vs a quantidade de “itens de defeito”. Esses diferentes perfis determinam a composição do fluxo, que você pode acompanhar ao longo do tempo:

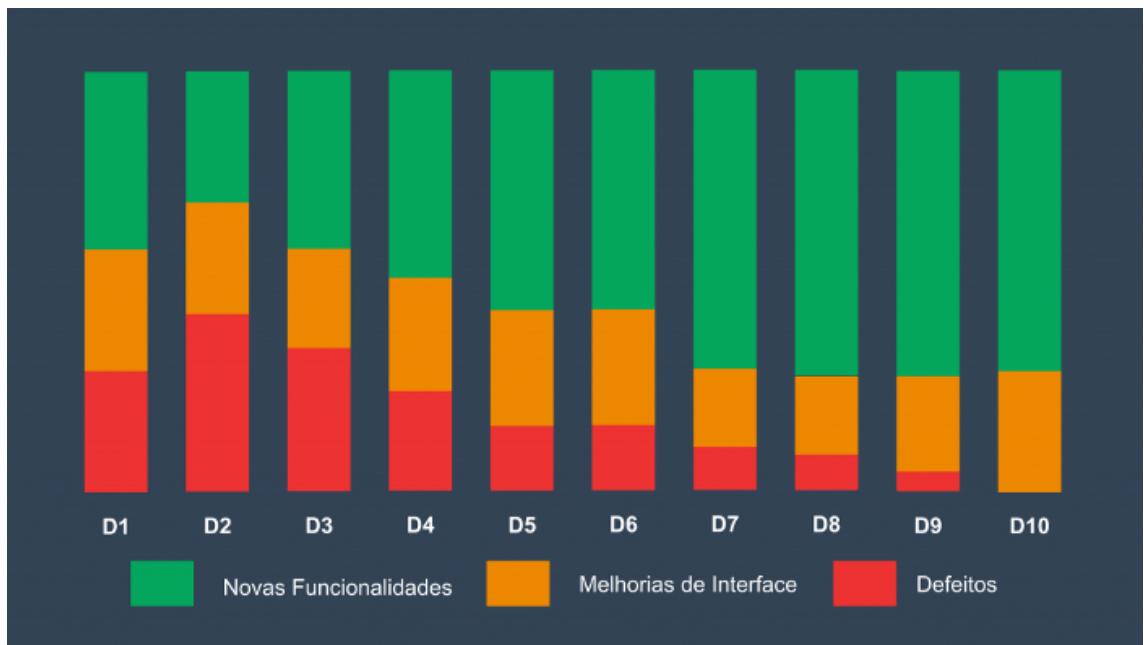


Diagrama de Fluxo Cumulativo (CFD). Outro diagrama fundamental para o seu fluxo é o CFD. Com ele você consegue identificar gargalos e olhar para a fluidez do seu processo. Leia esse post do Leonardo Campos sobre como ler e montar esse gráfico.

MEDINDO O SISTEMA, NÃO AS PESSOAS

Observe que nenhuma das métricas acima diz respeito ao trabalho dos indivíduos. O foco deve ser em melhorar o sistema como um todo, não as suas partes. Falando nisso, nem pense em utilizar essas métricas como parte de um sistema de avaliação de desempenho.

Isso certamente arruinaria a colaboração e a melhoria contínua, que são parte fundamental do método Kanban.

MÉTODO KANBAN & AUTOGESTÃO (O2) O método Kanban advoca uma evolução colaborativa e contínua. Isso tem tudo a ver com autogestão, que é um tema que gostamos muito aqui na Target Teal. Para não ter confusão, vamos definir melhor. Autogestão é:

Conjunto de práticas organizacionais que buscam distribuir a autoridade, dando clareza de responsabilidades e o máximo de autonomia a cada integrante da organização. Nesse caso, as pessoas deixam de reportar a um superior, porém seguem um conjunto de regras e acordos firmados coletivamente. Esses acordos formam uma estrutura organizacional que não exige que todos tenham o mesmo poder de decisão e autoridade, apenas deixa claro como isso é feito e impede a relação de chefe-subordinado.

O nosso método de autogestão favorito é o O2, também conhecido como Organização Orgânica(para saber mais sobre o tema, inscreva-se no nosso curso gratuito Fundamentos em Auto-Organização). O O2 possui alguns componentes básicos que são totalmente compatíveis com o método Kanban:

Papéis. No O2, buscamos esclarecer responsabilidades e expectativas através de papéis claros. Esse processo ajuda muito na hora de montar o quadro e entender como o seu processo funciona atualmente. Usamos o mesmo princípio do Kanban no O2. Começamos a partir da estrutura organizacional já existente. Além disso, o seu processo inevitavelmente terá uma série de papéis envolvidos nas diversas etapas. Tornar claras as responsabilidades de cada agente já gera ganhos no início.

Restrições. Uma das regras mais importantes do O2 é que tudo é permitido, a não ser que explicitamente proibido. Nos trabalhos de Kanban que fazemos, usamos o mesmo princípio na hora de definir as políticas. Colocamos todas elas na forma de restrições ou proibições. Por exemplo, ao invés de escrever:

Todos devem movimentar os cartões para a frente, sem voltar etapas.

Transformaríamos essa política em uma restrição do círculo, reescrevendo dessa forma:

Ninguém pode movimentar cartões para trás.

É uma mudança sutil, mas que faz toda a diferença. Se alguém perguntar: posso sinalizar um impedimento no cartão colocando uma bolinha colorida? A nossa resposta será: tudo é permitido, a não ser que explicitamente proibido. Então se não está proibido, siga em frente!

PAPEL DESIGNER DO SISTEMA

Quando começamos um trabalho de Kanban, também é comum assumirmos um papel de designer do sistema. Esse papel tem autoridade para realizar mudanças no processo de forma autocrática. Esse está longe de ser o estado que buscamos, pois sempre queremos nos tornar desnecessários o mais rápido possível. Mas antes disso acontecer, há um caminho que deve ser percorrido. O grupo passará por várias tentações no processo (aumentar os limites, por exemplo) que podem ser evitadas tendo alguém experiente no método com responsabilidade e domínio exclusivo sobre as regras do jogo.

Isso pode ser feito facilmente no O2, com um papel explícito com um artefato “Fluxo, políticas e restrições do Kanban”. Também vale colocar responsabilidade nos outros papéis de seguir as orientações

do designer do sistema.

Não é “feio” começar assim. Na realidade é mais funcional ter alguém com a “chave do quadro” no início do que pretender uma evolução colaborativa, que na prática você não vai executar.

KANBAN COMO RESTRIÇÕES DE UM CÍRCULO

Chega um momento que o grupo já tem conhecimento o suficiente para evoluir o sistema Kanban por conta própria. Nessa hora vale excluir o papel designer do sistema e transferir a responsabilidade para o grupo. Uma forma fácil de fazer isso com o O2 é simplesmente transformando todas as políticas do quadro em restrições do círculo. Isso fará com que estas restrições só possam ser alteradas mediante propostas no modo adaptar da reunião do círculo. Se isso for uma língua estranha para você, leia mais os nossos posts sobre O2.

Módulo 1 Semana 2

Conceito e objetivos de testes de software



Na primeira parte do módulo, trouxemos uma visão geral sobre a área de análise de testes e conhecemos um pouco dos princípios básicos de testes de software, assim como seus diferentes níveis e métodos. Nesta semana, vamos aprofundar nossos conhecimentos e entender alguns conceitos importantes de testes de software.

O que são testes de software?

Um software é todo programa utilizado nos diversos dispositivos disponíveis hoje em dia (computadores, celulares, televisores etc). Esses programas podem realizar diversas tarefas, como enviar mensagens (WhatsApp) ou ouvir músicas (Spotify), por exemplo. Também podemos pensar no software como uma ponte entre o computador e o humano que o utiliza.

Os softwares podem apresentar defeitos devido a muitas razões. Por isso, é muito importante realizar testes para verificar se os programas fazem o que deveriam fazer. O teste de software é o ato de examinar as partes e o desempenho do software para verificação e validação, consequentemente

permitindo que a empresa avalie sua qualidade e compreenda os riscos de sua implementação.

De maneira geral, os testes de software têm ganhado bastante importância no campo da tecnologia, pois ajudam a reduzir o custo total de um projeto de desenvolvimento de softwares. Caso o teste seja ignorado nos estágios iniciais de desenvolvimento, para economizar custos, ele pode acabar se tornando um quesito muito caro mais adiante, pois, à medida que se avança no processo de desenvolvimento, fica cada vez mais difícil rastrear defeitos. Além disso, corrigir um defeito pode gerar outra falha em alguma outra parte do software.

Atualmente é muito comum envolver a equipe de teste no processo de redação de especificações. Qualquer coisa que não esteja de acordo com o requisito é um defeito. Portanto, a equipe de teste deve ter uma ideia clara de como o software deve funcionar ao final do desenvolvimento. Na verdade, é importante começar a escrever casos de teste paralelamente à escrita de especificações, pois ajudará os testadores a analisar se todos os requisitos são testáveis ou não. Nas próximas semanas vamos estudar mais a fundo esses documentos, requisitos e casos de testes.

Quais opções abaixo são objetivos de testes de software?

- A.
- Descobrir o maior número possível de erros (ou bugs) em um determinado produto
- B.
- Aumentar o custo de desenvolvimento de um produto
- C.

- Demonstrar que um determinado software atende às suas especificações e requisitos
- D.
- Validar a qualidade de um software usando o mínimo de custo e esforço
- E.
- Gerar casos de teste de alta qualidade, executar testes eficazes e emitir relatórios de problemas para melhorias contínuas

Boas práticas de testes de software



Boas práticas de garantia de qualidade e de testes podem ajudar as empresas a reduzir consideravelmente os incidentes antes do lançamento de seus softwares. Listamos abaixo algumas práticas para tornar a realização de testes de software mais eficaz:

- Para estar à altura das expectativas do cliente, é importante programar cada passo com cuidado. Os testes de software devem ser planejados

considerando o orçamento, o cronograma e o desempenho, para assim alcançar os melhores resultados;

- Dentro deste planejamento, o testador (ou equipe de teste) segue um processo cujas etapas incluem: definir o ambiente de teste, desenvolver casos de teste, escrever scripts, analisar resultados de teste e gerar relatórios de defeitos;
- Os testes podem ser demorados e, dependendo do resultado, podem até atrasar o lançamento de um produto. Saber quais testes aplicar com maior eficiência em cada situação é uma das habilidades que os testadores precisam desenvolver. Testes manuais ou testes ad-hoc podem ser suficientes para pequenas compilações, porém, ferramentas para automatizar tarefas são frequentemente utilizadas para os sistemas maiores;
- Uma boa cobertura de testes abrange a interface de programação de aplicativos (API), a interface do usuário e os níveis do sistema. Além disso, quanto mais testes forem automatizados e executados antecipadamente, melhor;
- Testar continuamente à medida que novas funções do produto se tornam disponíveis permite que o software seja validado em diferentes fases e níveis — reduzindo os riscos de defeitos e bugs;
- Bons sistemas incluem autenticação de usuário e trilha de auditoria para ajudar as equipes a atender aos requisitos de conformidade com o mínimo de esforço administrativo;
- O monitoramento de defeitos é importante para que as equipes de teste e o desenvolvimento avaliem e aprimorem a qualidade dos softwares. Ferramentas automatizadas permitem que as equipes rastreiem defeitos, meçam seu escopo e impacto e descubram problemas relacionados;
- Relatórios e análises de defeitos permitem que os testadores compartilhem status, metas e resultados de testes. Com esses registros, as empresas podem incorporar as métricas do projeto e monitorar as relações entre teste, desenvolvimento e outros elementos de seus softwares. Vamos falar um pouco mais sobre esses documentos nos próximos slides.

Verdadeiro ou falso:

É importante escrever soluções de teste personalizadas para cada projeto, de acordo com as necessidades e seus possíveis usos. Um módulo em um aplicativo executado em um smartphone tem interface e componentes diferentes do ambiente do tablet, portanto, é importante planejar os testes para cada dispositivo.

• Falso

• B.

• Verdadeiro

•

Documento de requisitos de software



O documento de requisitos de software (também chamado de especificações de requisitos de software) compila e descreve os recursos e o desempenho

pretendido de um software. Em outras palavras, o documento de requisitos de software descreve o que o programa fará e como, além de cumprir com todos os objetivos dos stakeholders (usuários, empresa e mercado, por exemplo).

O que é um requisito?

“Os requisitos são uma especificação do que deve ser implementado. São descrições de como o sistema deve se comportar, ou de uma propriedade ou atributo do sistema”

Segundo a definição adotada por Wiegers Karl E. e Beatty Joy

Ou seja, requisitos são a definição de todos os comportamentos que o sistema deve realizar.

Por que o documento de requisitos de software é importante?

As especificações de requisitos de software são vitais para ajudar as equipes de desenvolvimento a criarem o produto certo. Com um documento de requisitos de software claramente definido, a equipe saberá exatamente como o software deve funcionar. Em síntese, esses requisitos são especificados em estágios iniciais do projeto do sistema e reduzirão a necessidade de redesign nos estágios posteriores do processo de desenvolvimento do software.



última.

Como preparar um documento de requisitos do produto?

Vamos começar com a redação do documento. Como de costume, dê ao documento um título, um número de revisão, a data de criação, para quem foi preparado e quem o preparou.

Você pode preparar um esboço simples de como será a estrutura do documento, como por exemplo:

1. Introdução
 - 1.1 Objetivo
 - 1.2 Público-alvo
 - 1.3 Uso pretendido
 - 1.4 Escopo
 - 1.5 Definições e siglas

2. Descrição gera

- 2.1 Necessidades do usuário
- 2.2 Premissas e dependências

3. Recursos e requisitos do sistema

- 3.1 Requisitos funcionais
- 3.2 Requisitos não funcionais
- 3.3 Recursos do sistema

Mesmo que você já tenha um modelo oficial, é bom procurar inspiração online. Na internet você encontra ideias para adaptar às suas necessidades, para criar seus modelos próprios ou para incorporar a algum modelo já existente.

O modelo a seguir segue as diretrizes estabelecidas na norma IEEE 830, segundo as quais a especificação de requisitos de software deve conter a

descrição da funcionalidade da aplicação, o relacionamento com sistemas externos e os requisitos não funcionais como desempenho, disponibilidade, tempos de resposta e manutenção, entre outros.



última.

Tópicos a serem abordados em um documento de requisitos de software.

Crédito da imagem: Prof B Wadhwa @ National University of Singapore

Levantamento de requisitos

O processo começa com o levantamento das informações que farão parte do documento de requisitos, e que são coletadas mediante entrevistas, documentos, reuniões, workshops, prototipagem etc. Existem diversas abordagens para elencar todos os requisitos de um software, porém algumas são mais utilizadas. O método mais comum para levantamento de requisitos é a entrevista.

As entrevistas são fáceis de agendar e realizar e, portanto, são preferidas pela maioria dos profissionais. Uma boa prática é agendar tempo suficiente, mas sem exageros, pois desse modo os participantes podem perder o foco. Após cada sessão, reordene suas anotações e relate todas as especificações que foram discutidas, para que tudo fique registrado.

Nesse momento também é importante identificar os problemas que serão resolvidos pelo software e quais são as partes interessadas em seu desenvolvimento, os chamados stakeholders.

Em um artigo escrito para o Project Management Institute, Larry Smith (2000) diz que:

“Entender os atributos, inter-relações e interfaces entre defensores e oponentes do projeto nos auxilia em seu planejamento estratégico. Aqui reside uma grande parte do risco e viabilidade do nosso projeto e, em última análise, o apoio que devemos efetivamente obter e reter.”

As partes interessadas são sua principal fonte de informação: tudo o que fará parte da redação dos requisitos será proposto, revisado e aprovado por eles. Não envolver todos os participantes necessários na análise de software levará inevitavelmente ao fracasso do projeto.

Assegurar a participação de todos, porém, é tarefa complexa, uma vez que convidar todas as pessoas interessadas e envolvidas no projeto para todas as reuniões e atividades pode resultar em um investimento de tempo e dinheiro de que o cliente pode não dispor. Basta considerar que, na maioria das vezes, os usuários finais são os únicos que sabem como o sistema deve funcionar e o que é necessário para torná-lo funcional.

Requisitos funcionais e não funcionais

Existem dois tipos principais de requisitos de sistema que devem ser especificados em projetos de software. Esses requisitos podem ser classificados como **requisitos funcionais** ou **requisitos não funcionais**. Compreender a diferença entre os dois fará com que os desenvolvedores possam entregar um produto com o desempenho esperado.

Os **requisitos funcionais** podem ser definidos como **algo que o sistema deve fazer**. Se o sistema não atender a um requisito funcional, ele falhará. Isso ocorre porque ele não será capaz de operar corretamente. O conceito de requisito funcional também pode ser entendido através da revisão do sistema em termos de entradas e saídas. Os requisitos funcionais especificam o que o sistema deve fazer em resposta a diferentes entradas e o que deve produzir. De um modo geral, os requisitos funcionais são compostos por recursos do produto e requisitos do usuário.

Alguns exemplos de requisitos funcionais são:

- As interfaces externas do sistema;
- Especificações sobre o que o sistema deve fazer;
- Etapas que o sistema deve seguir na autenticação;
- Os requisitos de relatório do sistema;
- Especificidades relacionadas à conformidade legal ou regulatória;
- Descrição dos níveis de usuário e sua autorização;
- Detalhes de como as transações devem ocorrer.

Requisitos **não funcionais** em engenharia de software podem ser definidos como requisitos que **descrevem como o sistema funciona**.

Os requisitos não funcionais são focados no modo como o sistema realiza uma função específica. À primeira vista, eles podem ser considerados menos importantes do que os requisitos funcionais, mas ambos têm um papel a desempenhar em um bom sistema. Os requisitos não funcionais não afetam a funcionalidade do sistema, mas o desempenho dele. Em suma, os requisitos não funcionais têm tudo a ver com a usabilidade do sistema. Se esses requisitos não forem atendidos, os usuários podem ficar frustrados e sair do sistema.

Os exemplos de requisitos não funcionais ajudam a entender melhor seu conceito — alguns deles estão listados abaixo:

- Velocidade — a rapidez com que o sistema executa determinadas atividades;
- Disponibilidade — por quanto tempo o sistema está disponível. Por exemplo, se funciona durante a noite, todos os dias do ano, ou não;
- Capacidade — quais são os limites do que o sistema é capaz de administrar;
- Confiabilidade — nível de confiança gerado pelo sistema;
- Usabilidade — facilidade com que o cliente ou usuário final tem em usar o sistema.

Escolha o tipo de requisito com base em cada exemplo abaixo.

1. O tempo necessário para uma página específica carregar.

- A.
- Requisito funcional
- B.
- Requisito não funcional

2. A capacidade do sistema de relatar a quantidade de transações que foram processadas corretamente.

- A.
- Requisito funcional
- B.
- Requisito não funcional

3. O que o sistema faz quando um usuário seleciona um determinado botão e para onde ele é direcionado em seguida.

- A.
- Requisito funcional
- B.
- Requisito não funcional

4. Quantos usuários o sistema pode suportar simultaneamente.

- A.
- Requisito funcional
- B.
- Requisito não funcional

5. A maneira como um usuário é autenticado quando faz login no sistema.

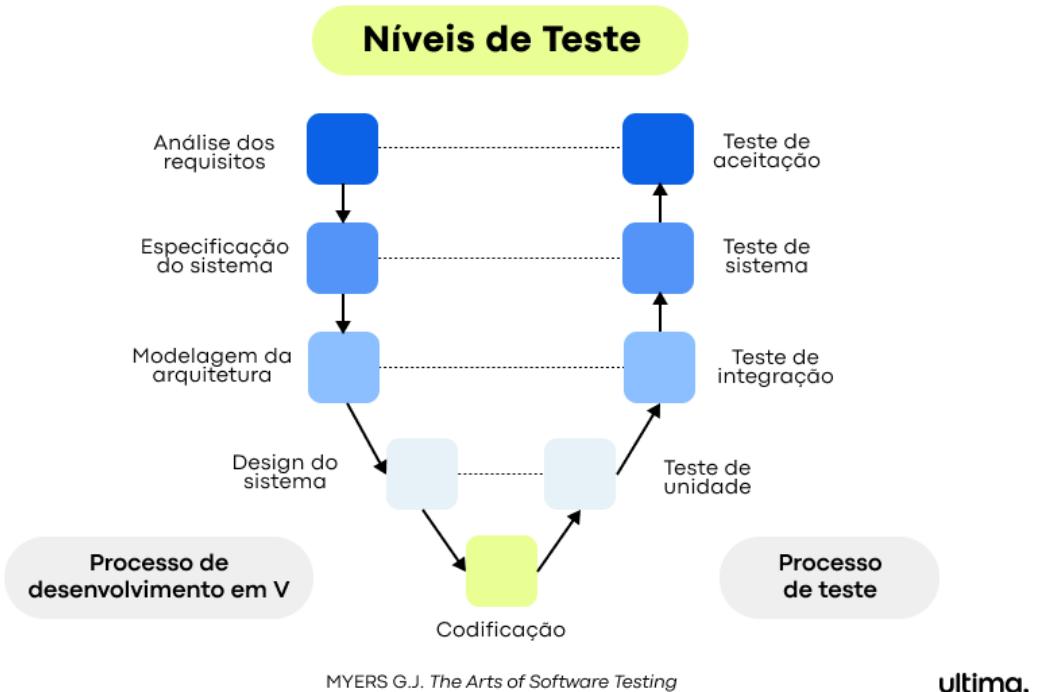
- A.
- Requisito funcional

- **B.**
- **Requisito não funcional**

Níveis de testes



Durante o ciclo de desenvolvimento de um software, ele passa por um processo completo de testes para garantir que esteja funcionando da maneira pretendida. Existem quatro níveis principais de teste de software: teste de unidade, teste de integração, teste de sistema e teste de aceitação.



Teste de unidade (ou teste de componente)

Durante esta primeira rodada de testes, o programa é submetido a avaliações que se concentram em unidades ou componentes específicos do software, para determinar se cada um deles está totalmente funcional. O principal objetivo deste esforço é determinar se o aplicativo funciona conforme projetado. Nesta fase, uma unidade pode se referir a uma função, programa individual ou até mesmo a um procedimento, e geralmente é usado um método de teste de caixa branca para realizar o trabalho. Um dos maiores benefícios desta fase de testes é que ela pode ser executada toda vez que uma parte do código é alterada, permitindo que os problemas sejam resolvidos o mais rapidamente possível. É bastante comum que os desenvolvedores de software realizem testes de unidade antes de entregar o software aos testadores para testes formais.

Teste de integração

O teste de integração permite combinar todas as unidades dentro de um programa e testá-las como um grupo. Este nível de teste é projetado para

encontrar defeitos de interface entre os módulos/funções. Isto é particularmente benéfico porque determina quão eficientes são as unidades funcionando em conjunto. Lembre-se de que, independentemente da eficiência de cada unidade, se elas não estiverem devidamente integradas, isso afetará a funcionalidade do software. Para executar estes tipos de testes, os profissionais poderão fazer uso de vários métodos, porém, o método específico a ser usado para realizar o trabalho dependerá muito da maneira como as unidades são definidas.

Teste de sistema

O teste de sistema é o primeiro nível em que o aplicativo é testado como um todo. O objetivo neste nível é avaliar se o sistema atendeu a todos os requisitos descritos e verificar se atende aos padrões de qualidade. Este teste é realizado em um ambiente que reflete a produção do software. O teste de sistema é muito importante porque verifica se o aplicativo atende aos requisitos técnicos, funcionais e de negócios definidos pelo cliente.

Teste de aceitação

O nível final, teste de aceitação (ou teste de aceitação do usuário), é realizado para determinar se o sistema está pronto para lançamento. Durante o ciclo de vida de desenvolvimento de software, as alterações de requisitos podem às vezes ser mal interpretadas, de uma maneira que não atende às necessidades dos usuários. Nesta fase final, o usuário testará o sistema para saber se o aplicativo atende às suas necessidades. Uma vez que este processo tenha sido concluído e o software tenha sido aprovado, o programa será então entregue à produção.

Como você pode ver, a extensão destes testes é mais uma razão pela qual é importante contar com seus testadores de software antecipadamente. Quando um programa é testado de forma mais detalhada, um número maior de bugs

pode ser detectado; em última análise, isto resulta em um software de maior qualidade.

Escolha a resposta que melhor completa cada nível de teste: Testes de unidade testam pequenas unidades de código, testes de integração examinam [1] e testes de sistema verificam [2].

- 1) alterações no código; 2) software em um ambiente de produção
- B.
- 1) objetos de software; 2) o trabalho dos testadores
- C.
- 1) a interação entre componentes; 2) sistemas completos
- D.
- 1) sistemas completos; 2) componentes

Tipos de testes



Testes manuais versus automatizados

Antes de nos aprofundarmos nos diferentes tipos de teste de software, vamos começar com a distinção básica entre testes manuais e automatizados.

Os testes manuais são realizados pessoalmente. O testador navega pelo aplicativo ou interage com o software e as APIs usando as ferramentas adequadas. Esse tipo de teste manual exige que as equipes configurem um ambiente e os executem por conta própria. Os testadores precisam fazer tudo manualmente e isso leva mais tempo, principalmente em projetos maiores, realizados em várias plataformas e navegadores. Quando se trata de projetos menores e mais curtos, o teste manual pode ser mais barato do que o teste automatizado. Naturalmente, como há humanos envolvidos no processo, este tipo de teste é propenso a erros — por exemplo, os testadores podem omitir etapas no script de teste ou cometer erros de digitação.

Ao contrário dos manuais, os testes automatizados são realizados por máquinas que executam um script de teste preparado previamente por testadores e/ou desenvolvedores (depende do tipo de teste). Também podem variar em complexidade. Enquanto alguns simplesmente verificam um único método em uma classe, outros garantem que uma sequência complexa de ações na interface do usuário produza os mesmos resultados no desempenho.

O teste automatizado oferece uma excelente oportunidade para dimensionar o processo de garantia de qualidade à medida que você adiciona novos recursos ao aplicativo.

Os tipos de teste de software são as diferentes maneiras e estratégias conduzidas para se verificar um software. Cada tipo se concentra em um objetivo específico e depende de vários fatores, incluindo requisitos do software, orçamento, cronograma, funcionalidades e acessibilidade.

Existem vários tipos de teste de software que ajudam as equipes de desenvolvimento a lançar produtos digitais de qualidade. Nem todos são iguais e sua escolha depende da abordagem utilizada pela equipe. Vale notar que tais testes não garantem que você tenha construído o produto certo, ou seja, um software que resolva o problema de seu público-alvo. Eles significam apenas que você construiu algo corretamente.

Agora vamos conhecer alguns tipos de testes. É sempre bom revisar este conteúdo ao fazer o seu planejamento.

Tipos de Testes

Teste Funcional

- Teste de Sistema
- Teste de Aceitação

Teste Estrutural

- Teste de Unidade
- Teste de Integração

Teste Não-Funcional

- Teste de Usabilidade
- Teste de Carga
- Teste de Segurança

Teste relacionado à mudança

- Teste de Manutenção
- Teste de Confirmação

última.

Teste funcional

Usado para testar se o software faz o que a empresa que o desenvolveu espera que faça, se funciona de acordo com suas especificações e com as expectativas das partes interessadas. Este tipo de teste também pode prever

cenários indesejados (por exemplo: perder todos os seus dados no momento em que a rede cair). Caso necessário, são incluídos testes exploratórios e/ou execução de cenários de teste (o teste exploratório permite que você pense fora da caixa e crie cenários que não são normalmente previstos).

- **Teste de sistema**

O maior teste de integração: quando tudo estiver montado, o sistema realmente irá funcionar?

Este teste é essencial caso grandes alterações sejam feitas no sistema. Está implícito em alguns outros testes, como por exemplo o teste realizado antes de lançar alterações, que deverá incluir a execução de um teste de sistema.

- **Teste de aceitação**

Construímos o sistema certo, ou seja, o sistema atende às necessidades do usuário? Este é o melhor teste para examinar o risco de valor antes do lançamento e também para validar o conceito do sistema com um grupo de usuários selecionados.

Teste estrutural

O teste estrutural é o tipo de teste realizado para verificar a estrutura do código. Ele também é conhecido como teste de caixa branca, e requer conhecimento do código do software. Este teste é mais focado no modo como o sistema realiza tarefas do que sua funcionalidade específica. O teste estrutural fornece uma cobertura mais abrangente que os demais testes. Por exemplo, para testar determinada mensagem de erro em uma aplicação, precisamos testar o momento em que ela ocorre. No entanto, pode haver outras condições em que a mensagem de erro ocorra, e é possível negligenciar uma dessas condições se apenas forem testados os requisitos elaborados no documento de requisitos de software. Porém, usando o teste estrutural, é mais provável que todas as

condições sejam testadas, pois este teste visa cobrir todos os nós e caminhos da estrutura do código.

O teste estrutural é complementar ao teste funcional. Usando esta técnica, os casos de teste elaborados de acordo com os requisitos do sistema podem ser analisados primeiro e, em seguida, outros casos de teste podem ser adicionados para aumentar a cobertura. Ele pode ser usado em diferentes níveis, como testes unitários, testes de componentes, testes de integração, etc. O teste estrutural auxilia na realização de testes completos no software.

- **Teste de unidade**

O código funciona como o desenvolvedor espera que funcione?

Este é um nível de teste em que unidades/componentes individuais de software são testados. O objetivo é validar se cada unidade do software funciona conforme foi projetada. Uma unidade é a menor parte testável de qualquer software, e geralmente tem uma ou mais entradas e uma única saída.

O teste de unidade aumenta a confiança na alteração ou manutenção do código, se os testes forem criados e executados todas as vezes em que o código é alterado.

Dessa forma, o desenvolvimento fica mais confiável e, além disso, o custo de correção de um defeito detectado durante o teste de unidade é menor em comparação com os custos de defeitos detectados em níveis mais altos.

- **Teste de integração**

Estes componentes funcionam em conjunto?

No teste de integração, vários módulos ou componentes são integrados e testados. Por meio de testes de integração, os testadores verificam se os

módulos individuais de código funcionam corretamente em conjunto. Muitos aplicativos modernos são executados em microsserviços — unidades independentes, projetadas para lidar com uma tarefa específica. Esses microsserviços devem poder se comunicar entre si ou o aplicativo não funcionará conforme o esperado. Por meio de testes de integração, os testadores podem verificar se esses componentes funcionam e se comunicam perfeitamente.

Um exemplo de teste de integração: uma empresa de cartão de crédito inclui uma página onde o cliente pode solicitar um aumento de crédito, que é uma base de código separada da funcionalidade de login. Os testadores podem realizar testes de integração para assegurar que o sistema se lembre do usuário depois que ele navegar para a página de aumento de crédito e novamente após uma solicitação bem-sucedida.

Teste não funcional

Os testes não funcionais verificam todos os aspectos não abordados nos testes funcionais. Estes aspectos incluem o desempenho, a usabilidade, a escalabilidade e a confiabilidade do software. Testes não funcionais são realizados para assegurar que os interesses do usuário final sejam levados em consideração.

- **Teste de usabilidade**

Os usuários conseguem entender o sistema e usá-lo para resolver seus problemas?

Um produto só vai gerar valor se os usuários puderem usá-lo para resolver seus problemas. Não importa que o software tenha sido bem projetado ou bem construído, se ele não atende às necessidades dos usuários.

O padrão para esse tipo de teste é o teste do usuário, no qual você pede a usuários reais (atuais ou em potencial) que tentem usar os novos recursos enquanto eles explicam o que estão pensando. Você não precisa esperar que o recurso seja construído antes de fazer o teste do usuário: o exemplo maior é o uso de protótipos de papel.

- **Teste de carga**

O sistema lidará com a carga necessária, qual é o limite do sistema (teste de carga)? Se sobrecarregarmos o sistema, de que maneira ele falha e se recupera (teste de estresse)?

O teste de carga é projetado para provar que um aplicativo pode lidar com a carga diária esperada. Trata-se de um teste que mede recursos e garante que vazamentos, quedas e picos de memória não ocorram ou então ocorram no momento pretendido.

- **Teste de segurança**

Somente usuários predefinidos podem obter acesso e o sistema está protegido contra exclusão ou corrupção accidental de dados?

O teste de segurança verifica se o sistema e os aplicativos da organização estão imunes a brechas. Este teste busca encontrar todas as fraquezas em potencial do sistema, ou seja, qualquer coisa que possa resultar em perda de informações nas mãos dos funcionários.

Teste relacionado a mudanças

Uma vez que o sistema é lançado, não significa que seja o fim de suas atualizações ou alterações. Na verdade, pode haver muitas iterações após o lançamento, como defeitos encontrados, novas funcionalidades etc. Mas o que são testes relacionados a mudanças?

O teste relacionado a mudanças é um tipo de teste com uma finalidade específica. Ele é projetado para verificar se uma alteração no sistema foi corrigida e se não existe nenhum efeito adverso no sistema como consequência disto.

- **Teste de regressão**

O teste de regressão é um teste para verificar se uma alteração de código no software não afeta a funcionalidade existente do produto. O teste de regressão ajuda a manter um produto estável enquanto as alterações são feitas nele. Testes de regressão são frequentemente automatizados.

Um exemplo de teste de regressão: um varejista de roupas adiciona a capacidade de pagar com pontos de recompensa em seu aplicativo para dispositivos móveis. Os testadores podem realizar testes de regressão em outras funcionalidades existentes, como a capacidade de pagar com cartões de crédito e cartões-presente, para confirmar que todas as formas de pagamento continuam funcionando corretamente.

- **Teste de confirmação**

Esta técnica confirma que a mudança pretendida atende à sua especificação.

Normalmente, os testadores relatam um bug quando um teste falha e os desenvolvedores lançam uma nova versão do software após a correção do defeito. Na sequência, a equipe de teste testará novamente para verificar se o bug relatado foi realmente corrigido ou não. Isto é conhecido como teste de confirmação, ou reteste.

Verdadeiro ou falso:

É essencial garantir que os usuários possam usar seu software conforme o esperado. Mas também é importante verificar se um sistema funcionará bem, mesmo que receba dados incorretos ou caso um usuário execute

uma ação inesperada. E estes são os objetivos de todos os diferentes tipos de testes de software listados acima.

Sua resposta está correta!

- A.
- Verdadeiro
- B.
- Falso

Testes estáticos e dinâmicos



Testes estáticos e testes dinâmicos são duas técnicas importantes relacionadas ao teste de software, e são utilizadas por testadores e desenvolvedores no ciclo de vida de desenvolvimento de software. Para tirar o máximo proveito de cada tipo de teste e escolher as ferramentas certas para uma determinada circunstância, é fundamental compreender as vantagens e limitações de cada um desses tipos de teste.

O **teste estático** é um tipo de teste no qual o código não é executado. É possível realizá-lo manualmente ou através de ferramentas de automação. Este tipo de teste verifica o código, as especificações e os documentos de design e insere observações de revisão nos documentos de trabalho. Com testes estáticos, tentamos descobrir os erros e defeitos de código no software. Ele começa em uma etapa anterior no ciclo de vida de desenvolvimento e, portanto, também é chamado de teste de verificação. O teste estático é utilizado em documentos de trabalho como código fonte, planos de teste, scripts de teste e casos de teste.

O **teste dinâmico** é realizado quando o código está no modo de tempo de execução, ou seja, enquanto o código está sendo executado; o resultado ou a saída do código é verificado e comparado com o resultado esperado. Com isso podemos observar a funcionalidade do software, verificar a memória do sistema, o tempo de reação da CPU, e o desempenho do sistema. O teste dinâmico também é chamado de teste de validação, avaliando o produto final. O teste dinâmico pode ser de dois tipos: teste funcional e teste não funcional.

Tanto o teste estático quanto o teste dinâmico são cruciais para a aplicação do produto.

Diferenças entre testes estáticos e dinâmicos

Considerando todos os aspectos, o teste estático é uma técnica de teste vital que inclui: revisão de requisitos de negócios, revisão de design, revisão de requisitos funcionais, orientações de código e revisão de documentação de teste. É um processo incessante e realizado não apenas por testadores.

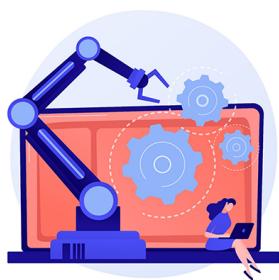
A validação, a parte do teste dinâmico, é mais abrangente e acontece no próprio produto e não em um documento ou representação do produto. Dentre as estratégias de teste dinâmico podemos destacar: o procedimento formal de identificação de caso/condição de teste, a contemplação de escopo, a execução e o relatório de erros.

Verdadeiro ou falso:

Testes precisam ser realizados em cada nível do ciclo de vida de desenvolvimento de software.

- **Verdadeiro**
- B.
- **Falso**

A pirâmide de testes automatizados

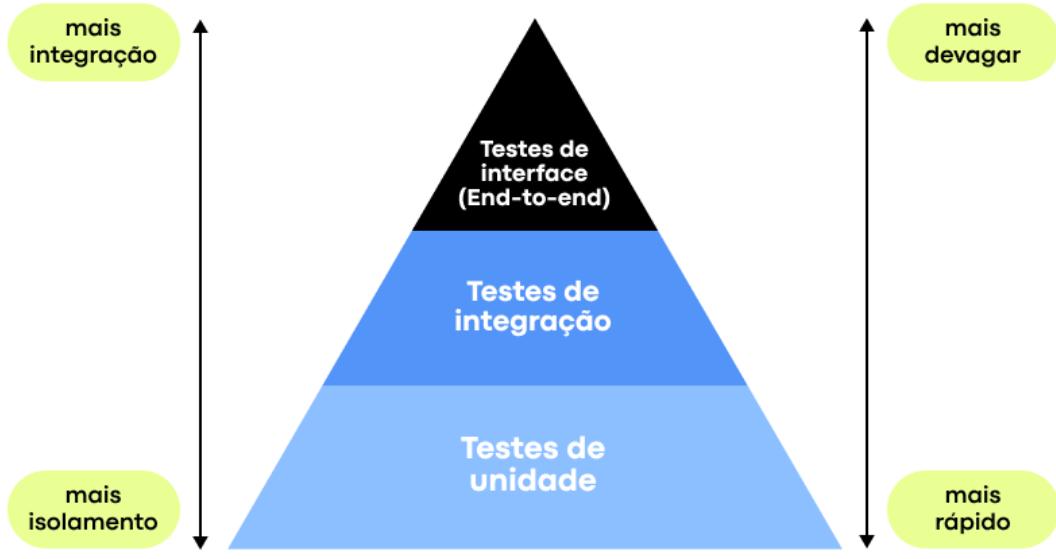


A função de uma pirâmide de testes é delinear os diferentes níveis de testes e fornecer a referência quanto à quantidade de testes que deve haver em cada um destes níveis. Embora o conceito da Pirâmide de Testes já exista há algum tempo, as equipes ainda têm dificuldade para colocá-lo em prática adequadamente.

À medida que a disciplina de desenvolvimento de software foi amadurecendo, as abordagens de teste de software também evoluíram. Ao invés de ter inúmeros testadores de software manuais, as equipes de desenvolvimento passaram a automatizar a maior parte de seus esforços de teste. A automatização dos testes permite que as equipes saibam se o software tem algum problema em questão de segundos e minutos, em vez de dias e semanas.

O ciclo de feedback drasticamente reduzido e alimentado pelos testes automatizados faz parte das boas práticas de desenvolvimento ágil e de entrega contínua. Ter uma abordagem eficaz de teste de software permite que as equipes trabalhem com rapidez e confiança.

Quando estudamos testes automatizados para software, a pirâmide de testes é um conceito-chave. Mike Cohn utilizou esse conceito em seu livro *Succeeding with Agile*. É uma ótima metáfora visual que nos leva a pensar em diferentes camadas de teste, e também informa quantos testes devem ser feitos em cada camada.



última.

A pirâmide de testes original de Mike Cohn é formada por três camadas, e define uma proporção da quantidade de testes que devem ser realizados (de baixo para cima):

1. Testes de unidade
2. Testes de integração (ou serviço)
3. Testes de interface do usuário (*End-to-end*)

No topo da pirâmide, temos os testes de interface (ou ponta a ponta — e2e).

Seu objetivo é reproduzir o comportamento do usuário final (seja uma pessoa, uma API ou qualquer outro tipo de cliente) em um aplicativo.

Na base da pirâmide temos os testes de unidade, nos quais verificamos o funcionamento da menor unidade de código testável do nosso aplicativo.

Entre estas duas camadas temos os testes de integração. Seu objetivo é verificar se um conjunto de unidades se comporta corretamente, em uma escala menor do que os testes de ponta a ponta.

Devido à sua simplicidade, a pirâmide de testes serve como uma boa regra quando se trata de estabelecer seu próprio conjunto de testes. Dois conceitos importantes da pirâmide de testes para você se lembrar:

- Escreva testes com granularidade diferente;
- Quanto mais alto o nível que você alcançar, menos testes você deverá ter.

Atenha-se à forma de pirâmide para criar um conjunto de testes robusto, rápido e de fácil manutenção: escreva muitos testes de unidade pequenos e rápidos. Escreva alguns testes mais grosseiros e escreva poucos testes de alto nível que testem seu aplicativo de ponta a ponta.

Testes de unidade

Os testes de unidade verificam o funcionamento da menor unidade de código testável de um software, independentemente de suas interações com outras partes do código. Os testes unitários formam a base da pirâmide de testes. Eles testam componentes ou funcionalidades individuais para validar se funcionam conforme o esperado em condições isoladas. É importante executar vários cenários em testes de unidade — caminho feliz, tratamento de erros etc.

Como este é o maior subconjunto, os testes de unidade devem ser criados para serem executados o mais rápido possível. Lembre-se de que o número de testes de unidade aumentará à medida que mais recursos forem adicionados. Este conjunto de testes precisa ser executado toda vez que um novo recurso é adicionado. Consequentemente, os desenvolvedores recebem feedback imediato sobre os recursos individuais que estão funcionando (ou não) como deveriam.

Uma boa maneira de construir um conjunto de testes de unidade robusto é praticar o desenvolvimento orientado a testes (TDD). Como o TDD exige que o teste seja escrito antes do código, o código acaba sendo mais simples, claro e livre de bugs.

Testes de Integração

Os testes de integração verificam pequenos trechos de uma base de código. Ou seja, para testar como esse código interage com outros códigos (que formam a totalidade do software), testes de integração precisam ser executados. Essencialmente, são testes que validam a interação de um trecho de código com componentes externos. Estes componentes podem ser bancos de dados, serviços externos (APIs) e similares.

Os testes de integração são a segunda camada da pirâmide de automação de testes. Isto significa que eles não devem ser executados com tanta frequência quanto os testes de unidade. Fundamentalmente, eles testam como um recurso se comunica com dependências externas. Seja com uma chamada para um

banco de dados ou com um serviço da Web, o software precisa se comunicar de forma eficaz e recuperar as informações corretas para funcionar conforme o esperado.

Lembre-se que, como os testes de integração envolvem interação com serviços externos, eles serão executados mais lentamente do que os testes de unidade. Eles também exigem um ambiente de pré-produção para serem executados.

Teste de ponta a ponta — E2E (ou teste de interface do usuário)

São testes que simulam um ambiente real, como: abrir o aplicativo em um navegador, preencher formulários, clicar em botões e ao final verificar se o resultado é o esperado. A diferença entre este tipo de teste e o que um cliente real faz é que um teste de ponta a ponta geralmente é realizado em um ambiente controlado e as ações são executadas por um *bot* (e não por um usuário real). Vale lembrar que quando mencionamos um "usuário", não estamos necessariamente falando de alguém que acessa sua página ou aplicativo de desktop — se seu software é uma API, seu usuário é o consumidor dessa API.

Usar a pirâmide de automação de testes é especialmente vantajoso para equipes ágeis. Os processos ágeis enfatizam velocidade e eficácia e a pirâmide de testes oferece a agilização do processo de testes: com uma progressão clara e lógica introduzida no pipeline de testes, o trabalho é feito mais rapidamente. Como a pirâmide é construída para executar os testes mais fáceis no início, os testadores gerenciam melhor o tempo, obtêm melhores resultados e isto facilita a vida de todos os envolvidos.

A pirâmide indica aos testadores as prioridades certas. Se os scripts de teste forem escritos com um foco maior na interface do usuário, é provável que a lógica de negócios principal e a funcionalidade de *back-end* não sejam suficientemente verificadas. Isto afeta a qualidade do produto e gera mais trabalho para a equipe. Além disso, como o tempo de retorno dos testes de

interface do usuário é longo, a cobertura geral de testes acaba sendo reduzida. Ao implementar a pirâmide, tais situações podem ser evitadas.

Em testes automatizados, ferramentas como o Selenium executam testes com script em um aplicativo ou componente de software para confirmar que estejam funcionando conforme o esperado. O objetivo principal é reduzir o esforço humano e a supervisão.

Verdadeiro ou falso:

A pirâmide de automação de testes pretende organizar e estruturar o ciclo de testes.

- A.
- **Verdadeiro**
- B.
- Falso

Teste manual de software em aplicativos web



O que é um teste manual

Durante as fases de desenvolvimento de um software, os testes podem ser realizados em diferentes níveis, como foi apresentado na semana anterior. A atividade de teste de sistema é realizada após a etapa de programação, quando já existe alguma interface a ser testada.

Em um teste de sistema manual, como o nome diz, um testador não utiliza nenhuma ferramenta automatizada. Esse tipo de teste é o coração do processo de qualidade de software, e costuma ser feito após o desenvolvimento de qualquer novo aplicativo, sempre do ponto de vista do usuário final.

O teste manual requer mais esforços, de modo geral, mas é necessário para verificar a viabilidade da automação de outros testes que ainda serão realizados. Além disso, a realização de testes manuais não requer conhecimento de nenhuma ferramenta ou linguagem de programação.

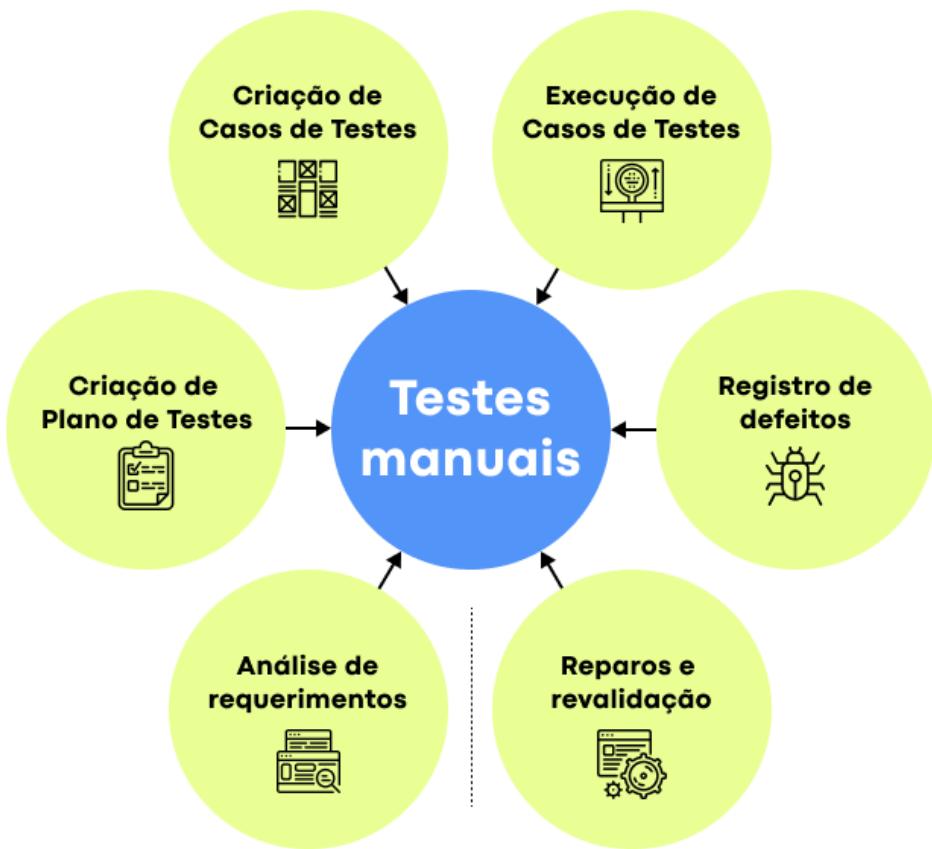
Teste manual de software em aplicativos web (ou: como testar um site)

Não basta ter um site. Ele precisa ser informativo, acessível, fácil de usar e de carregamento rápido. Para manter todas essas qualidades, o site deve ser bem testado, e esse processo é conhecido como teste de aplicativos (ou aplicações) web.

O teste manual de software em aplicativo web é uma prática para verificar se tudo está funcionando conforme pretendido ou de acordo com seus requisitos.

Em vez de serem instaladas no dispositivo do usuário (como no caso dos aplicativos móveis), as aplicações web são executadas em um navegador de internet. Além disso, os sites são executados em qualquer dispositivo que possa acessar a Internet, incluindo computadores desktop, tablets e telefones celulares. Não há necessidade de baixar ou instalar uma atualização, os novos recursos estão imediatamente disponíveis para os usuários quando um aplicativo é atualizado. Por esse motivo, é imprescindível testar aplicativos web extensivamente antes que eles entrem no ar.

Para isso, é necessário que o site passe por uma série de validações para verificar sua funcionalidade, usabilidade, acessibilidade, compatibilidade, desempenho e segurança.



última.

Veja um passo a passo para realizar testes manuais:

- Analise o documento de especificação de requisitos de software;
- Crie um plano com os tipos de testes que serão realizados;
- Escreva casos de teste que cubram todos os requisitos definidos no documento;
- Execute casos de teste e identifique quaisquer bugs;
- Analise os resultados e gerencie os reparos a serem realizados, e depois faça uma revalidação desses reparos.

Ferramentas de gerenciamento de projetos:

Durante o ciclo de desenvolvimento de um software, é criado um projeto de software, no qual as atividades de análise de requisitos, desenvolvimento, testes e o registro dos defeitos precisam ser definidas e gerenciadas. Um

projeto de software descreve o objetivo, o escopo, o cronograma, as atividades, o custo e os recursos necessários para realizá-lo. É comum haver uma pessoa com o papel de Líder de projeto ou Scrum Master que ficará responsável por verificar o andamento dessas atividades até que elas sejam finalizadas e o produto de software desenvolvido seja lançado em diferentes versões.

Para um melhor gerenciamento desses projetos de software, há ferramentas que ajudam a distribuir as atividades para cada participante do projeto e também registrar os defeitos (bugs) encontrados nos testes manuais e que precisam ser corrigidos pelos desenvolvedores. Dentre as ferramentas mais utilizadas para esse gerenciamento dos projetos temos: Mantis, Redmine, Gitlab, Github e Jira.

No próximo slide vamos conhecer um pouco mais sobre o Jira.

Verdadeiro ou falso:

considerando as vantagens e o serviço que as ferramentas de gerenciamento de projetos oferecem, as empresas podem realmente maximizar as entregas de suas equipes de desenvolvimento. Essas ferramentas podem otimizar todo o fluxo de trabalho, aumentar a produtividade do desenvolvedor e permitir o lançamento do aplicativo em menos tempo, o que acaba gerando mais lucros.

- A.
- Verdadeiro
Você selecionou esta opção inicialmente.
- B.
- Falso

Diferenças entre requisitos de sistemas e regras de negócios



Tanto as regras de negócios quanto os requisitos são necessários para definir o escopo completo de um sistema, mas qual é a diferença entre os dois? Vamos começar com algumas definições para que possamos ver claramente onde os requisitos e as regras de negócios diferem.

O que é uma regra de negócio?

Em uma empresa, os sistemas e processos internos devem ser projetados para apoiar a implementação de regras de negócios. Uma regra de negócio é uma restrição do próprio negócio que pode orientar o desenvolvimento do sistema que essa empresa quer criar. Muitas vezes, envolve critérios ou condições muito específicas, que devem ser seguidos.

Ao obter uma compreensão geral de quais são as regras de negócios, pode-se determinar quais requisitos podem ser extraídos. As regras de negócios são uma excelente fonte de requisitos, especialmente porque precisam ser seguidas pelos participantes do processo e aplicadas pelo sistema.

O que é um requisito?

Um requisito funcional descreve o comportamento mais abrangente do sistema, algo que o sistema fará para atender as necessidades dos seus usuários. Os requisitos contemplam todas as funcionalidades do sistema, conforme o que foi definido pelo cliente do software.

A principal diferença entre requisitos e regras de negócio é que geralmente os requisitos descrevem apenas o que o sistema deverá fazer, enquanto as regras de negócio definem como esses requisitos deverão funcionar. As regras incluem tanto o que é permitido como também o que não é permitido realizar nos sistemas.

Exemplo

Vamos esquecer os softwares por um minuto. Vejamos um exemplo que não envolve necessariamente tecnologia.

Requisito: as horas diárias trabalhadas dos funcionários da empresa devem ser registradas.

Regra de negócio: cada funcionário deve registrar sua entrada ao iniciar seu trabalho e sua saída ao final de seu turno.

Verdadeiro ou falso: as regras de negócios definem como um requisito deverá ser implementado em um software, incluindo as ações que o usuário não poderá realizar no sistema.

- A.
 - Falso
-
- B.
 - Verdadeiro

Como criar roteiros de teste



Os testes de software, quando realizados de forma manual, podem ser realizados de forma exploratória — quando dependem apenas da experiência do testador; ou de forma planejada — através da elaboração de um roteiro de teste.

No entanto, através da qualidade desses roteiros, é possível garantir que os testes realizados serão mais eficientes e que irão detectar mais falhas em um sistema. O uso de roteiros de teste é uma abordagem confiável para verificar a cobertura dos requisitos e regras de negócio do sistema, ou seja, se nada foi ignorado no processo de desenvolvimento do software e se os resultados são realmente os esperados pelas partes envolvidas.

Um roteiro de teste precisa ser organizado de forma que possa ser executado seguindo a sequência lógica de cada requisito do sistema. Já que muitas vezes há dependências entre os requisitos, pode ser necessário executar os testes de uma tela de cadastro de usuário antes dos testes de autenticação daquele usuário. Com isso, um roteiro de teste é composto por cenários e, por sua vez, cada cenário é composto de casos de teste.

A atividade de elaborar o roteiro de teste é criativa, com isso, é possível planejar diversos caminhos ou cenários de teste para cada tela do sistema,

levando em consideração a ordem de execução de cada tela conforme suas pré-condições e os objetivos de cada requisito do sistema. Esses cenários de teste são um tipo menos detalhado de documentação, no qual os objetivos gerais de teste são listados para serem testados. Uma maneira de organizar e detalhar esses cenários é criando casos de teste, que detalham cada execução no sistema e verificam se ele está em conformidade com as regras de negócio.

Casos de teste são uma descrição linha por linha de todas as ações a serem executadas para se testar jornadas específicas do usuário. É uma lista de cada passo que deve ser dado e os respectivos resultados esperados. Em seguida, os testadores podem testar sistematicamente cada etapa em uma ampla variedade de dispositivos.

Por exemplo, um cenário de teste pode informar: “Teste se os cupons de desconto podem ser aplicados em cima de um preço de venda”. Isso não menciona como aplicar o cupom ou se existem várias maneiras de aplicar o cupom. O teste real que cobrirá este caso pode variar de tempos em tempos. O testador usará um link para aplicar um desconto, ou inserirá um código, ou fará com que um representante de atendimento ao cliente aplique o desconto, ou seja, irá testar várias formas possíveis para se adicionar um desconto.

Os cenários de teste são mais abrangentes, eles descrevem apenas o objetivo do teste. Já os casos de teste definem como exatamente o teste deverá ser realizado. Logo, quando o sistema evoluir e as telas mudarem, pode ser que o cenário de teste não precise ser alterado, no entanto, os casos de teste devem mudar para se adequarem às novas mudanças das telas.

Exemplo de um caso de teste

Por exemplo, para verificar a função de login em um site, seu caso de teste deve conter os seguinte passos:

1. Carregue a página inicial do site e clique no link “login”;

2. Verifique se a tela de login e os campos “Nome de usuário” e “Senha” estão visíveis;
3. Em seguida, digite o nome de usuário “Charles” e a senha “123456”, identifique o botão “Confirmar” e clique nele;
4. É esperado que o sistema exiba o título: “Bem-vindo, Charles” na tela principal do site.

Se o texto do título estiver de acordo com a expectativa, deve se fazer um registro de que o teste foi aprovado. Caso contrário, informar que o teste falhou e registrar um defeito

Aqui estão algumas dicas importantes para criar bons roteiros de teste:

Claro: certifique-se que cada etapa do roteiro de teste é clara, concisa e coerente. Isso ajuda a manter o processo de teste tranquilo.

Simples: crie um roteiro de teste que contenha apenas uma ação específica para os testadores realizarem por vez. Isso garante que cada função seja testada corretamente e que os testadores não se percam nas etapas da execução do teste de software.

Bem pensado: para escrever o roteiro de teste, você precisa se colocar no lugar do usuário para decidir quais caminhos testar. Você deve ser criativo o suficiente para prever todos os diferentes cenários pelos quais os usuários passariam ao executar um sistema.

O tipo de documentação menos detalhada é o teste baseado em checklist, quando não há casos de testes a serem seguidos. É um teste que pode usar como base apenas uma lista de cenários que descrevem o objetivo que um usuário tem ao usar aquela tela do sistema. Um exemplo pode ser “Teste se o usuário pode sair com sucesso do programa”. Normalmente, um teste de cenário exigirá ações diferentes para garantir que o cenário foi coberto satisfatoriamente. Apenas com base nessa descrição, o testador pode optar

por fechar o programa por meio da opção de menu, encerrá-lo pelo gerenciador de tarefas etc.

Práticas recomendadas para escrever casos de teste



Como vimos anteriormente, um caso de teste especifica o procedimento a ser executado, os resultados esperados e as condições que um testador precisa verificar em um determinado software. Ele faz parte da documentação básica necessária para determinar se um software, ou um de seus recursos, está funcionando conforme planejado. Determinar se um produto ou seus recursos estão prontos para lançamento geralmente exigirá que vários casos de teste sejam escritos e executados.

Os casos de teste derivam dos requisitos especificados ou implícitos de qualquer software ou sistema. Esse processo de “tradução” de histórias de usuários para comandos de casos de teste é uma habilidade essencial a ser desenvolvida como testador de softwares.

Algumas perguntas podem surgir no processo de planejamento e redação dos casos de testes:

- Quão específico ou amplo deve ser um caso de teste?
- Quais casos de teste devem ser escritos primeiro?
- Quais aspectos da funcionalidade o caso deve cobrir?

Quando se trata de escrever casos de teste para software, anote os casos que lhe trarão o melhor retorno sobre o investimento (ROI), seguindo algumas práticas recomendadas:

Crie casos de teste com base em riscos e prioridades: priorize quais casos de teste escrever com base nos cronogramas do projeto e nos fatores de risco do seu software. Não existe uma fórmula de caso de teste, você precisará adequar suas ações ao seu contexto de trabalho.

Lembre-se da regra 80/20: 20% de seus testes cobrirão 80% de sua aplicação. Escrever e testar um caso, mesmo que restrito, pode revelar uma parte significativa de seus bugs. Por isso, é melhor começar com testes mais gerais e só então começar a cobrir recursos específicos com mais profundidade.

Certifique-se de que seus casos de teste possam ser concluídos por outras pessoas quando necessário: ao escolher qual teste escrever, concentre-se em como eles podem ser “terceirizados”. Por exemplo, escreva testes que você pode dar aos desenvolvedores para que eles mesmos executem enquanto você está ocupado testando

Dica: em alguns casos, será impossível escrever um único teste que atenda a todos os públicos que irão usar uma determinada aplicação. Você pode então considerar escrever 2 versões de um mesmo teste.outros pontos específicos do software que você não pode repassar a mais ninguém.

O caso de teste “bom o suficiente”: muitas vezes é melhor escrever casos de teste que sejam bons o suficiente para o momento, mas não deixe de revisá-los no futuro, quando possível. Essa abordagem ágil/iterativa também se aplica à escrita de casos de teste, e não apenas a tarefas de desenvolvimento de um software

Crie testes que serão relevantes em sprints/builds/releases futuros: se você torná-los muito específicos, sua relevância durará apenas para esta etapa do seu projeto. Assim, à medida que o sistema evoluir, alguns casos de teste poderão parar de funcionar e precisarão ser alterados.

Liste seus testes antes de escrevê-los: crie uma lista de tópicos e prioridades com base no risco. Isso ajudará a manter o foco no que você precisa ou deseja testar. Mesmo quando a lista não é final, você pode posteriormente dividir os testes ou mesclá-los.

Classifique os casos de teste com base nos cenários de negócios e na funcionalidade: isso permitirá que você veja o sistema de diferentes ângulos. A classificação também ajudará a organizar seus testes em um Repositório de Testes, para que você e sua equipe possam escolher quais testes podem ser executados com base nas necessidades do seu plano de testes no futuro.

Teste seus testes: os testes que você escreve também serão aperfeiçoados à medida em que você os executa. Certifique-se de documentar bem todas as versões e avaliar a eficiência dos testes em cada etapa do projeto.

Verdadeiro ou falso: aplicando boas práticas durante a elaboração de um roteiro de teste, é possível obter uma melhoria significativa em sua execução.

- A.
- Verdadeiro
- B.
- Falso

Material complementar

fundamentos de Técnicas e fundamentos de Testes

Introdução

Os testes representam uma etapa de extrema importância no processo de desenvolvimento de software, pois visam validar se a aplicação está funcionando corretamente e se atende aos requisitos especificados.

Nesse contexto existem diversas técnicas que podem ser aplicadas em diferentes momentos e de diferentes formas para validar os aspectos principais do software. Nos artigos abaixo você conhecerá conceitos fundamentais do teste de software e como aplicá-los.

Introdução aos diferentes tipos de teste

Esse tema é útil quando o objetivo dos envolvidos no projeto é agregar qualidade ao software, fugindo do “vício” habitual das empresas de executar apenas os testes funcionais mais comuns, aumentando assim a fidelização e a satisfação dos clientes.

Com o aumento da utilização de sistemas web, associado a uma busca contínua por mais qualidade, menos riscos e melhores resultados, adicionar o

Teste de Software ao ciclo de vida de desenvolvimento do software torna-se cada vez mais importann te.

O problema é que o Teste, uma área relativamente nova, ainda encara muitas barreiras. Isto se dá, às vezes, em função dos recursos humanos e financeiros reduzidos, outras, por causa do tempo e dos prazos limitados dos projetos, sem contar que existem situações em que o Teste é simplesmente deixado de fora do escopo do projeto por puro desconhecimento de causa ou até mesmo despreparo dos gestores envolvidos.

O que ocorre é que nos dias de hoje não dá mais para deixar o Teste como uma atividade irrelevante. É sabido que não é viável realizar uma centena de tipos de testes para garantir a qualidade de um software, mas que tal começar devagar, adicionando novos testes gradativamente, para que aos poucos essa atividade possa conquistar o seu devido espaço na empresa e a confiança do cliente?

É válido lembrar que cada projeto apresenta características distintas, que dependem do tamanho do software, da tecnologia utilizada para o seu desenvolvimento e de muitos outros fatores. Assim, a escolha adequada dos tipos de testes que serão adotados torna-se primordial.

Nesse artigo, o foco será dado aos testes de Usabilidade, Aceitação, Portabilidade e Confiabilidade. No entanto, antes de prosseguir, para auxiliar na compreensão do que representa cada um desses testes, vamos apresentá-los sucintamente, através dos seguintes questionamentos:

- Teste de Usabilidade: O software é fácil de usar?
- Teste de Confiabilidade: O quanto podemos contar com o correto funcionamento do software? Ele é tolerante a falhas?
- Teste de Portabilidade: É possível utilizar o software em diversas plataformas com pequeno esforço de adaptação?
- Teste de Acessibilidade: Qualquer usuário, deficiente ou não, consegue utilizar a aplicação?

Ao longo desse artigo, além dos testes já citados, serão apresentadas as vantagens e as ferramentas que viabilizam as suas respectivas execuções. Como referência principal, será utilizada a ISO-9126, uma norma que define características e subcaracterísticas voltadas para a qualidade de software.

Teste de Usabilidade

O Teste de Usabilidade tem como objetivo avaliar a usabilidade da aplicação, determinando até que ponto a interface do software é fácil e intuitiva de utilizar.

Esse tipo de teste possibilita detectar todas as ações dos usuários, analisar suas preferências, ajudando a determinar o que pode ser melhorado na aplicação.

Os problemas mais comuns que podem ser detectados com a execução de testes de usabilidade em aplicações web são:

- Irrelevância: informações exibidas na interface que não possuem nenhuma utilidade, sem pertinência às necessidades do usuário;
- Inadequação: abreviaturas usadas sem prévia explicação do termo completo;
- Resposta inesperada: ao acionar um link, a aplicação apresentar outra página totalmente diferente da solicitada;
- Cliques: efetuar muitos cliques para conseguir chegar à funcionalidade desejada.

Com a realização desse teste é possível analisar o tempo de resposta do servidor para cada requisição, e até mesmo a satisfação, mesmo que subjetiva, do usuário, em relação ao que está sendo apresentado.

Jakob Nielsen, um dos maiores especialistas na área de usabilidade, listou um conjunto de critérios básicos para analisar a usabilidade de um software. São eles:

- Intuitividade: o software deve ser fácil de ser utilizado, permitindo que mesmo um usuário sem experiência seja capaz de executar algum trabalho satisfatoriamente;
- Eficiência: o sistema deve ser eficiente em seu desempenho, viabilizando um alto nível de produtividade;
- Memorização: as interfaces da aplicação devem de fácil memorização, permitindo que usuários ocasionais consigam utilizá-lo mesmo depois de um longo intervalo de tempo;
- Erro: a quantidade de erros apresentados pelo software deve ser a menor possível. Além disso, a solução para os problemas deve ser simples e rápida, mesmo para usuários iniciantes. Erros graves ou sem solução não podem ocorrer;
- Eficácia: o software deve, sobretudo, cativar o usuário, sejam eles iniciantes ou avançados, possibilitando uma interação no mínimo agradável.

Analizar essas características de usabilidade em uma aplicação em desenvolvimento, ou até mesmo pronta, possibilita que muitos problemas sejam diagnosticados antes do software ser disponibilizado para o cliente.

Isso pode, no mínimo, evitar a insatisfação do cliente numa fase avançada do ciclo de vida do desenvolvimento do software, onde as correções se tornam mais caras e impactantes.

Como executar o teste de usabilidade

Muitas são as maneiras de medir a usabilidade de um software. A escolha depende de cada projeto e está diretamente ligada ao resultado que se deseja obter.

Executados em laboratório, em um ambiente controlado, ou em ambiente real, muitas técnicas de teste de usabilidade podem ser utilizadas. De uma maneira geral, o teste de usabilidade pode ser realizado seguindo quatro etapas principais:

- Planejamento: envolve determinar o objetivo do teste, bem como a metodologia que será utilizada. Define também o perfil dos participantes, como recrutá-los, o cenário e as tarefas que serão realizadas. Nessa primeira etapa, um ponto fundamental é saber exatamente o que será avaliado. Criar um pequeno checklist, como o exemplificado a seguir, pode facilitar essa etapa:
 - Qual o objetivo do teste?
 - Quando e onde o teste será executado?
 - Qual o tempo de duração para cada sessão de teste?
 - Qual ambiente e ferramentas necessárias para executá-lo?
 - Em qual estado o software deve estar no início do teste?
 - Quem serão os usuários envolvidos no teste?
 - Quais tarefas serão definidas para que os usuários executem?
 - Quais são as funcionalidades mais críticas?
 - Quais serão os dados coletados durante os testes?
 - Qual critério será utilizado para definir se um teste foi bem sucedido?
- Preparação: na segunda etapa, tudo deve estar pronto para por o teste em prática. É nesse momento que o usuário receberá uma explicação sobre como será executado o teste, e caso alguma ferramenta seja utilizada, um breve treinamento deve ser realizado;
- Execução: na terceira etapa é onde o teste é efetivamente executado. É aconselhável que antes de realizar o primeiro teste, um projeto experimental seja colocado em prática.
Nesse momento serão detectadas as informações primordiais para a avaliação posterior do resultado. É recomendável ainda utilizar um cronômetro, e é importante informar ao usuário que executará o teste o tempo limite para conclusão de cada tarefa. Outros pontos também devem ser seguidos, a saber:
 - Não interfira no teste, no momento da execução;
 - Documente os pontos de dificuldade citados pelos usuários;
 - Anote as sugestões do(s) usuário(s) participante(s).
- Análise: esta é a etapa final dos testes de usabilidade e tem como objetivo analisar os dados obtidos. É o momento de revisar os

problemas encontrados, definindo a frequência com que ocorrem, a gravidade e a prioridade para resolução dos mesmos.

Nessa etapa, é fundamental que todos os detalhes sejam avaliados, como por exemplo, a quantidade de cliques antes da conclusão de uma tarefa, o tempo de duração de cada atividade, quantas vezes o help foi acionado e, se for o caso, quantos erros foram apresentados.

Ferramentas

Nos últimos anos, houve um considerável aumento no número de ferramentas voltadas para o teste de usabilidade. Para exemplificar, apresentaremos três delas:

- Loop11: é uma ferramenta online voltada para a execução de testes de usabilidade não moderados (sem o acompanhamento presencial de um orientador) com usuários reais. Para utilizá-la, basta cadastrar algumas tarefas, solicitar que sejam realizadas e o Loop11 irá armazenar as ações.

Como resultado, através de relatórios você poderá rastrear a interação do usuário com a aplicação, o tempo de conclusão de cada tarefa, os caminhos percorridos e as falhas apresentadas;

- Ethnio: permite recrutar os visitantes da sua aplicação web para ajudá-lo com a pesquisa sobre a usabilidade do software. Ou seja, a ferramenta resolve o problema de tentar encontrar usuários reais para participar dos testes e ainda envia relatórios periódicos sobre os resultados obtidos;
- Morae: permite captar toda a interação do usuário, incluindo voz, vídeo, teclas, movimentos do mouse e ações na tela. Ou seja, essa ferramenta é um laboratório de testes de usabilidade completo, pois é capaz de obter uma sofisticada gama de informações, possibilitando uma rica análise da aplicação.

Vantagens do Teste de Usabilidade

A execução desse tipo de teste pode fornecer inúmeros benefícios ao software, a saber:

- Avaliação da qualidade das telas da aplicação;
- Identificação de problemas visando melhorias durante o desenvolvimento do software;
- Pode ser associada à prototipação, tornando a avaliação do protótipo mais efetiva.

Teste de Confiabilidade

A confiabilidade de um software é medida de acordo com a estabilidade e o desempenho da aplicação durante um determinado período de tempo, sob diferentes condições de teste.

O objetivo desse teste é garantir a integridade completa dos dados trafegados pelo software, monitorando e avaliando a capacidade que a aplicação tem de concluir as suas operações com sucesso, conforme especificado.

As informações obtidas ao longo dos testes de confiabilidade devem ser coletadas em todas as etapas do ciclo de vida do desenvolvimento de software, identificando sempre quando uma interrupção produzir uma falha.

Em suma, o teste de confiabilidade visa analisar a integridade, conformidade e interoperabilidade do software, incorporando os resultados de testes não funcionais, como os testes de segurança e estresse, aos testes funcionais.

Esse teste pode ser descrito através de algumas características, a saber:

- Recuperabilidade: o software é capaz de voltar a funcionar após uma falha?
- Maturidade: com que frequência a aplicação apresenta falhas inesperadas?
- Tolerância a falhas: como o software se comporta quando uma falha ocorre?

- Conformidade: a aplicação está de acordo com normas e convenções previstas em leis?

Como executar o teste de confiabilidade

O teste de confiabilidade é considerado caro, em comparação com os demais citados ao longo desse artigo. Isto porque pode envolver outras técnicas de teste. Sendo assim, torna-se primordial executá-lo de forma planejada e estruturada. Para isso, alguns passos podem ser seguidos, conforme apresentado a seguir:

- Estabeleça metas de confiabilidade: o objetivo dessa primeira etapa é determinar qual será o desempenho satisfatório do software em termos que sejam significativos para o cliente. O ideal é que todos os envolvidos no ciclo de vida do desenvolvimento estejam cientes da priorização dessas metas;
- Desenvolva perfis operacionais: definir quem irá realizar esses testes também é muito importante. Feito isso, é preciso descrever exatamente o que cada usuário irá realizar, juntamente com o resultado que se deseja alcançar. Os recursos de hardware necessários para execução dos testes também devem ser especificados;
- Planeje e execute os testes: esse terceiro passo visa levantar os dados necessários para execução dos testes, bem como verificar as exigências dos requisitos, atribuir as responsabilidades de acordo com os perfis operacionais e, por fim, registrar todas as informações obtidas ao longo da execução do teste de confiabilidade;
- Use os resultados do teste para tomar decisões: após a execução do teste, tem-se o momento de avaliar os resultados. Este passo é fundamental para definir qual será a priorização da solução para os problemas apresentados.

Vantagens do Teste de Confiabilidade

A execução desse tipo de teste pode fornecer inúmeros benefícios ao software, a saber:

- Determina se o software atende aos requisitos de confiabilidade definidos pelos usuários, proporcionando a confiança de cumprir com os objetivos pretendidos;
- Verifica a existência de pontos fracos na aplicação;
- Analisa até que ponto o software está apto a suportar as falhas apresentadas.

Saiba mais sobre Confiabilidade de Software.

Teste de Portabilidade

O Teste de Portabilidade tem como objetivo verificar o grau de portabilidade da aplicação em diferentes ambientes e situações, envolvendo desde o hardware até o software. Por exemplo, um grande desafio para quem desenvolve aplicações web é garantir que ela tenha o mesmo comportamento independente do navegador que o usuário esteja utilizando.

A execução desse tipo de teste consiste em avaliar algumas características fundamentais de uma aplicação. Dentre elas, destacamos:

- Adaptabilidade: É fácil adaptar o software a outras plataformas?
- Coexistência: A aplicação conseguirá compartilhar os recursos comuns com outros softwares independentes?
- Capacidade de substituição: É fácil substituir o software por outro especificado para o mesmo fim?

Esse tipo de teste pode ter o seu planejamento voltado para avaliar questões de hardware, browsers, de diferentes tipos, e sistemas operacionais, com suas várias versões e *service packs*.

Uma dica para testar a aplicação em diferentes plataformas é a utilização de máquinas virtuais (VMs). Através de emulação as VMs permitem que diferentes

sistemas operacionais sejam executados em uma mesma máquina, sem a necessidade de *dual boot*. Além disso, as VMs permitem salvar o estado da máquina ou restaurá-lo facilmente.

Como executar o teste de portabilidade

Executar um teste de portabilidade não é complicado e nem requer um grande planejamento, basta utilizar o software na arquitetura desejada. Para que seja viável a execução desse tipo teste, algumas pré-condições devem ser seguidas:

- Verificar se o software foi ou está sendo desenvolvido seguindo os requisitos de portabilidade;
- Confirmar se a integração entre todos os componentes foi realizada;
- Analisar se o ambiente está preparado, apto para a execução do Teste de Portabilidade.

Ferramentas

Para execução do teste de Portabilidade, não é necessário que haja um computador para cada teste. Existem ferramentas que simulam diferentes situações numa mesma máquina, por exemplo:

- IETester: disponível gratuitamente em sua versão *alpha*, é uma ferramenta simples e intuitiva que possibilita emular o navegador Internet Explorer e de uma só vez utilizar desde a versão 5.5 até a versão 10;
- Windows XP Mode: é uma ferramenta para o Windows 7 que, em conjunto com o Windows Virtual PC, propicia emular uma versão do Windows XP;
- VirtualBox: é uma ferramenta da Oracle que permite instalar e executar diferentes sistemas operacionais em um único computador. Pode ser instalado no Windows, Linux, Solaris e Mac.
- VMware Player: é um software gratuito que funciona como uma máquina virtual, assim como o VirtualBox. Permite a instalação e utilização de um

ou vários sistemas operacionais dentro de um mesmo computador. É possível até mesmo utilizar os dispositivos USB conectados à máquina física e a internet.

Vantagens do Teste de Portabilidade

A execução desse tipo de teste determina se o software pode ser portado para cada um dos ambientes especificados pelo usuário, além de avaliar características fundamentais para a sua correta utilização, como:

- Espaço necessário em disco;
- Velocidade do processador;
- Resolução do monitor;
- Navegadores e Sistemas Operacionais suportados.

Teste de Acessibilidade

O Teste de Acessibilidade tem como objetivo garantir que o software poderá ser utilizado por qualquer usuário, inclusive aqueles que possuam algum tipo de deficiência física. Esse teste verifica se as interfaces do software permitem uma navegação adequada para todos.

Visando as aplicações web, existem padrões, citados a seguir, que determinam se existe ou não acessibilidade no software. Essa questão é tão relevante que, para garantir que os serviços ou informações governamentais disponíveis na web sejam acessíveis a todos os usuários, em qualquer horário, local, ambiente ou dispositivo, o governo brasileiro definiu, em 2005, o “Modelo Brasileiro de Acessibilidade em Governo Eletrônico”, conhecido como e-Mag.

No mundo, com o objetivo de tornar a web acessível a um número cada vez maior de cidadãos, o World Wide Web Consortium (W3C), principal organização de padronização da Web, um consórcio internacional com quase 400 membros, criou o WAI (*Web Accessibility Initiative*), com a atribuição de manter grupos de trabalho elaborando conjuntos de diretrizes para garantir a

acessibilidade ao conteúdo da Internet a pessoas com deficiências, ou para os que acessam a rede em condições especiais de ambiente, equipamento, navegador e outras ferramentas Web.

Em suma, diante de tantas padronizações, algumas características são fundamentais quando se trata de testar a acessibilidade de uma aplicação web. Dentre elas, destacamos:

- Perceptível: a informação e os componentes da interface da aplicação devem ser apresentados de modo que os usuários possam perceber o que é exibido na tela;
- Operável: a interface do software não pode exigir uma interação que o usuário não consiga executar. Além disso, os componentes da tela precisam ser operáveis;
- Compreensível: a informação e a operação da interface da aplicação devem ser de fácil compreensão;
- Robusto: o conteúdo precisa ser sólido o suficiente para que ele possa ser interpretado por diferentes usuários, ferramentas e tecnologias assistivas (recursos e serviços que contribuem para proporcionar ou ampliar habilidades funcionais de pessoas com deficiência).

Lembre-se: Para o teste de acessibilidade, quanto mais autoexplicativa é a aplicação, mais facilmente ela será utilizada. Isto permitirá também que os sites de busca encontrem mais rapidamente o conteúdo da sua aplicação web.

Como executar o teste de acessibilidade

Para medir a acessibilidade de um software algumas condições precisam ser seguidas. Segundo o *checklist* do WCAG1.0 da W3C um documento que fornece uma listagem com todos os pontos de verificação previstos nas “Diretrizes para Acessibilidade ao Conteúdo da Web”, o primeiro passo é definir a prioridade da funcionalidade baseada no seu impacto. Essas prioridades podem ser divididas em três categorias:

- **Prioridade1:** pontos que têm que ser totalmente desenvolvidos na aplicação. Caso contrário, um ou mais grupos de usuários ficarão impossibilitados de acessar as informações contidas no software;
- **Prioridade2:** pontos que o software deve satisfazer. Caso contrário, um ou mais grupos de usuários terão dificuldades em acessar as informações contidas na aplicação;
- **Prioridade3:** pontos facultados, ou seja, que a aplicação pode ou não contemplar. Caso contrário, um ou mais grupos poderão se deparar com algumas dificuldades em acessar a aplicação.

Diante de tantas padronizações vindas dos mais distintos setores, desde os públicos até os privados, o teste de acessibilidade possibilita que sejam analisadas inúmeras características na aplicação, como por exemplo:

- Verificar se os campos da interface permitem uma tabulação na ordem da disposição dos campos na tela, visando à impossibilidade de utilizar o mouse;
- Analisar se é viável a navegação por toda a aplicação apenas com o teclado;
- Verificar a existência de texto informativo nas imagens (*hint*), possibilitando que outros softwares possam informar ao usuário o seu respectivo conteúdo;
- Analisar se é possível alterar o tamanho do conteúdo da interface, como textos, tabelas e imagens;
- Verificar se em tabelas de dados são identificados os cabeçalhos de linha e de coluna;
- Verificar se é fornecida uma descrição sonora das informações importantes veiculadas em trechos visuais de apresentações multimídia;
- Analisar se as informações estão divididas em pequenos blocos, facilitando o gerenciamento;
- Verificar se os mecanismos de navegação estão sendo utilizados de maneira coerente e sistemática;
- Verificar se está especificado por extenso cada abreviatura ou sigla ao longo da aplicação;

- Verificar se é identificado o principal idioma utilizado pela aplicação.

Ferramentas

Nos últimos anos, houve um considerável aumento no número de ferramentas voltadas para o teste de usabilidade. Para exemplificar, apresentaremos três delas:

- AChecker: é uma ferramenta que permite que você informe uma URL e teste uma aplicação web, verificando a conformidade de acordo com diversas orientações de acessibilidade;
- Wave Toolbar: é uma barra de ferramentas do Firefox que fornece um mecanismo para a execução de testes diretamente do navegador. Um fator interessante é que nenhuma informação é enviada para o servidor Wave, garantindo assim uma acessibilidade 100% privada e segura. A ferramenta consegue avaliar a versão renderizada da aplicação mesmo o conteúdo gerado dinamicamente;
- NVDA: NonVisual Desktop Access é uma ferramenta open source de leitura de tela. Ela permite que pessoas com algum nível de deficiência visual ouçam o conteúdo por meio de uma voz sintetizada. Suporta mais de 20 idiomas e não precisa ser instalada no computador, bastando utilizá-la diretamente a partir de um *pendrive*.

Vantagens do Teste de Acessibilidade

A execução desse tipo de teste pode fornecer inúmeros benefícios ao software, a saber:

- Disponibilizar uma aplicação para um público mais amplo permite melhoria na fidelização dos clientes;
- Todo usuário, independentemente de ser portador ou não de deficiência, terá acesso à aplicação;
- Reforça e amplia a acessibilidade como uma característica relevante do software.

Conclusão

Executar os testes de Usabilidade, Confiabilidade, Portabilidade e Acessibilidade não é uma atividade comum dentro de um Processo de Teste de Software. No entanto, à medida que são inseridos, de forma combinada ou não, podem contribuir significativamente para a ampliação do público alvo da aplicação.

Em termos de Portabilidade, hoje em dia estão acessíveis a praticamente todos os públicos os mais distintos sistemas operacionais e navegadores. Pensando na acessibilidade, estima-se que somente no Brasil, segundo o último censo do IBGE, 23,95% da população sofra de algum tipo de deficiência. Já a Usabilidade e Confiabilidade podem garantir, no mínimo, a fidelização do cliente, tornando o software mais eficaz, eficiente e comprehensível.

Executados sozinhos, esses testes podem até não garantir um software (quase) perfeito, mas agregados a outros tipos de teste e diante dos diferentes aspectos expostos ao longo desse artigo, bons resultados podem ser obtidos.

Para acrescentá-los ao ciclo de vida de desenvolvimento do software, não será necessário nem tanto investimento e, talvez, nem tanto tempo como se imagina, considerando a melhoria da qualidade que pode ser obtida na aplicação.

Entendendo o Teste de Software por Amostragem

A técnica de teste por amostragem tem como principal objetivo permitir a inferência estatística em atividades de teste. Devido à impossibilidade de considerar todos os dados de um sistema, é retirado um dado amostral que será utilizado como base e a partir dele serão realizadas conclusões prováveis em maior ou menor grau, dependendo do número de dados considerados e

como essa seleção foi feita, gerando assim conclusões incertas. Entretanto, quando essas informações são avaliadas seguindo certos princípios que regem uma pesquisa, o grau de incerteza da inferência estatística pode ser mensurado.

Para realizar a mensuração do grau de incerteza envolvido é utilizada a teoria da Probabilidade, que é uma importante área da matemática que tem por finalidade modelar a ocorrência de eventos potenciais de um experimento aleatório (resultados que não seguem um modelo padrão, mas estão todos presentes no espaço amostral).

Podemos utilizar a técnica de teste amostral de diversas formas, uma delas é se baseando nos casos de testes existentes, porém é preciso tomar cuidado, pois para atingir a expectativa é necessário que esses casos de testes sejam criados de forma eficiente e objetiva, senão a amostra não será representativa o suficiente. As técnicas de amostragem são utilizadas geralmente por motivo de auditoria ou teste de regressão, mas nada impede que seja implementada nas atividades diárias de um testador.

Pode ser utilizada para testar um caso de teste específico como, por exemplo, um caso de teste que avalia um determinado campo e este possui muitas variações. Nessa situação, as técnicas de amostragem auxiliam a determinar quais informações serão inseridas para que não seja necessário inserir todos os dados possíveis para realizar o teste.

Para utilizarmos de forma coerente a técnica de testes por amostragem é preciso ter certa experiência no software que será testado, ou seja, ao menos uma navegação deve ter sido feita na aplicação. Esse teste preliminar precisa acontecer para que tenhamos conhecimento das áreas mais críticas do sistema e então saibamos dividir nossa amostra de forma proporcional a atingir os pontos desejados tentando ser o menos incerto possível. Desta forma seremos capazes de saber como e onde tirar boas amostras do que é preciso testar.

Se tomarmos como base o que é feito em outras áreas de conhecimento, antes de investirem tempo e dinheiro em um determinado solo, água, etc.,

primeiramente é retirada uma pequena amostra do que precisa ser avaliado e a partir dessa informação analisa-se se o solo é fértil, se a água é impura e assim por diante, concentrando então seus esforços nos pontos avaliados, conforme o resultado obtido. Com os analistas de testes funciona da mesma forma; ao utilizar técnicas de amostragem tomando como base o risco, concentram o tempo de teste nesses pontos, otimizando assim seus testes. Isso é muito utilizado quando o tempo disponível é pequeno e a quantidade a ser testada é grande e com muitas variações gerando assim muitas combinações. Com base nos resultados ...

Ferramentas de suporte ao Teste de Software

Ao longo dos anos, à medida que o teste vem ganhando mais espaço no mercado, diversas ferramentas têm surgido visando à melhoria no gerenciamento e na execução das atividades voltadas para o Teste de Software.

A utilização de ferramentas é um importante aliado para avaliar se o software está exposto a vulnerabilidades. Independente da função, o objetivo gira em torno da simplificação das atividades, controle, e, principalmente, garantia da qualidade em cada uma das etapas de um Processo de Teste. Outro fator relevante é que elas permitem, sobretudo, minimizar os riscos, inconformidades e uma série de falhas que podem ser detectadas durante a execução dos testes.

Além de simplificar as atividades, a adoção dessas ferramentas pode aumentar consideravelmente o número de defeitos encontrados, e, consequentemente, agilizar os testes, aprimorando a qualidade das aplicações. Elas também podem levar a melhorias na confiabilidade do software, tornando os testes mais eficazes e produtivos.

Com base nesse contexto, ao longo deste artigo serão apresentadas algumas ferramentas de diferentes características e classificações voltadas para fornecer um amplo suporte ao teste de software.

Classificação das ferramentas

As ferramentas são classificadas de acordo com as atividades de teste que elas suportam. Seja para o gerenciamento ou para execução de um teste, elas são categorizadas de acordo com os tipos de teste que executam.

Os tipos de teste, alguns deles demonstrados na Tabela 1, possuem diferentes funções, e cada um deles visa avaliar uma determinada característica do software. Um fator interessante é que há ferramentas de suporte que contemplam praticamente todos os tipos de teste existentes.

Teste Unitário	Geralmente realizado pelo desenvolvedor, é um tipo de teste que tem como objetivo testar trechos do código do software.
Teste de Segurança	Verifica se a aplicação se comporta adequadamente mediante as mais diversas tentativas ilegais de acesso.
Teste de Sistema	Tem como objetivo verificar o comportamento de todas as funcionalidades do software, analisando possíveis falhas operacionais.
Teste de Ambiente	O objetivo deste teste é verificar se o ambiente está pronto para o início dos testes, de modo que a integridade dos mesmos seja mantida.
Teste de Integração	Visa garantir que os componentes da aplicação desenvolvidos separadamente funcionem perfeitamente quando integrados.
Teste de Desempenho	Tem como objetivo verificar o tempo de resposta e o rendimento da aplicação de acordo com o resultado esperado.

Tabela 1. Tipos de Teste de Software.

Hoje em dia, algumas ferramentas suportam várias atividades, como gestão dos testes, automação e controle dos defeitos, enquanto outras suportam apenas uma. Mesmo assim, apresentá-las intitulando uma ou outra como “a melhor” não é possível. Esse título vai depender de uma série de fatores como, por exemplo, investimento disponível, aplicabilidade, tipo de software a ser testado, tamanho da equipe e expectativa de retorno. Em função disso, é interessante analisar as opções disponíveis para que seja possível definir qual se enquadra melhor às suas expectativas.

Neste contexto, ao longo desse artigo serão apresentadas diferentes classificações de ferramentas de suporte ao teste, todas com um propósito único: garantir a qualidade do software.

Ferramentas de gerenciamento de testes

A gestão de testes possui um papel fundamental dentro de um Processo de Teste, afinal, é responsável pelo planejamento e controle de todas as atividades.

As ferramentas de gerenciamento podem ser utilizadas em todas as etapas do ciclo de vida do teste e têm como grande vantagem a centralização de todas as informações relacionadas à evolução dos testes realizados no software.

Geralmente elas não têm apenas uma função, podendo apresentar distintas aplicabilidades, como priorização dos testes, cronograma de atividades, registro dos resultados, rastreamento do progresso e gerenciamento dos incidentes dos testes.

O objetivo dessas ferramentas está voltado para o controle dos testes, sejam eles manuais ou automáticos. Elas permitem que informações sejam coletadas

e acompanhadas, possibilitando uma visão mais ampla para a tomada de decisão, fornecendo artefatos de acordo com os interesses e expectativas.

Processo de teste

Para evitar que o teste seja uma mera etapa do ciclo de desenvolvimento, a implantação de um processo relacionado a este garante um maior controle das atividades de teste e, consequentemente, mais qualidade ao software. Nos artigos a seguir você verá como a utilização de um Processo de Teste pode melhorar a efetividade dos testes, controlando as atividades e garantindo mais credibilidade e valor ao produto.

Processo de Teste de Software

De que se trata o artigo:

Explicar como a utilização de um Processo de Teste pode melhorar a efetividade dos testes, controlando as atividades e garantindo mais credibilidade e valor ao produto.

Em que situação o tema é útil:

Para evitar que o teste seja uma mera etapa do ciclo de desenvolvimento, a implantação de um processo relacionado a este garante um maior controle das atividades de teste e, consequentemente, mais qualidade ao software.

Resumo DevMan:

Nesse artigo apresentamos o Processo de Teste de Software com as suas principais etapas e respectivos artefatos gerados. Abordamos também os papéis e as responsabilidades de cada membro da equipe envolvida. Por fim, citamos algumas boas práticas que podem contribuir para obter sucesso na utilização de um Processo de Teste.

Autores: Renata Eliza e Vivian Lagares

Ao contrário do que muitos imaginam, testar um software vai muito além de executar testes explorando as funcionalidades de um sistema já desenvolvido. Na realidade, para que seja possível avançar diante desse estigma, o primeiro obstáculo a ser enfrentado é fazer com que o Processo de Teste e o Processo de Desenvolvimento sejam executados em paralelo, desde o início do ciclo de vida do software.

Um Processo de Teste de Software tem como objetivo estruturar as etapas, as atividades, os artefatos, os papéis e as responsabilidades do teste, permitindo organização e controle de todo o ciclo do teste, minimizando os riscos e agregando valor ao software.

A estruturação do processo tem o propósito de reduzir o número de erros apresentados no projeto. Mas para que isso seja possível, a definição dos objetivos do teste deve ser bem clara, as melhores técnicas devem ser selecionadas, e uma equipe de pessoas treinadas e qualificadas deve estar apta para desempenhar os respectivos papéis dentro do processo.

Após estruturar os objetivos, o próximo passo é implantar o processo. A implantação permite que o teste deixe de ser tratado como uma atividade secundária, passando a ser um processo próprio, condutor do fluxo das atividades do teste e gerador de artefatos que tornam possível a avaliação da qualidade do software.

Neste contexto, apresentaremos nesse artigo as principais fases do ciclo de vida do Processo de Teste, com os referentes artefatos gerados por cada uma delas. Abordaremos também a hierarquia e a definição dos papéis envolvidos no teste, com suas respectivas responsabilidades. Por fim, serão listadas algumas práticas que podem contribuir para a melhoria do software com a implantação do processo.

Ciclo de Vida do Processo de Teste

O ciclo de vida consiste em uma série de etapas dependentes, consideradas como o esqueleto do Processo de Teste, que visam estruturar as atividades definindo como os testes serão conduzidos no projeto.

Essas etapas podem variar de acordo com a metodologia utilizada, e conforme apresentado na Figura 1, serão demonstradas nesse artigo as principais etapas do ciclo de vida de um processo de teste.

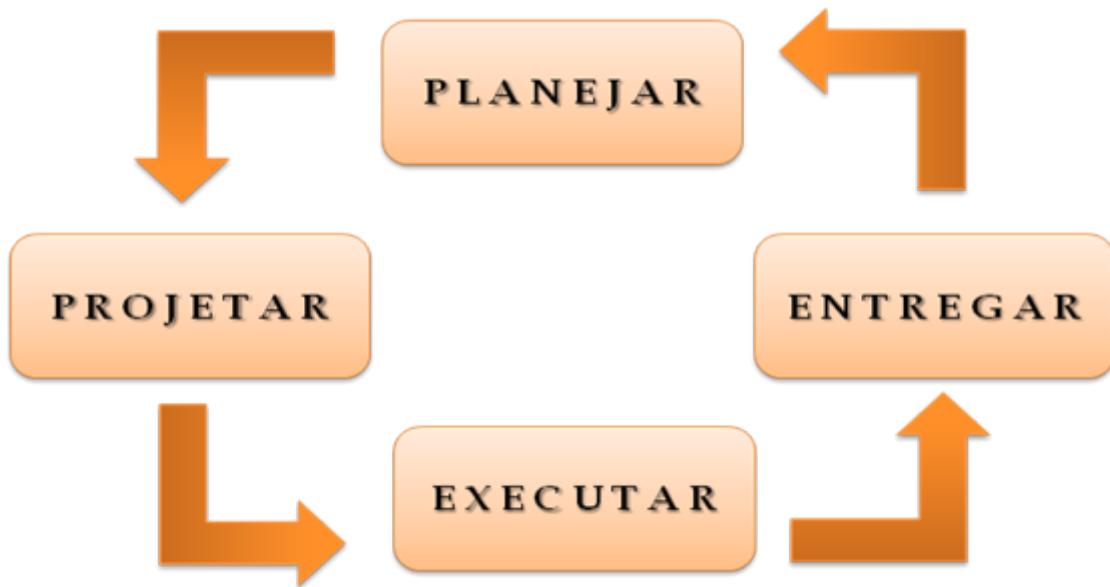


Figura1. Ciclo de Vida do Processo de Teste de Software.

Conforme indicado na Tabela 1, a execução de cada etapa do ciclo de vida tem um tempo estimado de duração. Assim, é recomendável seguir essa estimativa para que seja possível executar todas as fases do processo, minimizando riscos e consequentemente garantindo mais qualidade ao software.

Etapa Do Processo de Teste	Distribuição do Tempo Estimado
-------------------------------	-----------------------------------

Planejar Testes	10%
Projetar Testes	40%
Executar Testes	45%
Entregar	...

Desvendando o processo de Teste de Software

Em um projeto de software é natural que os envolvidos foquem no resultado final, esperando que a solução funcione corretamente.

Porém, antes de pensarmos no que seria uma solução funcionando corretamente, sugerimos primeiramente que você pense no contexto de negócio onde o sistema está envolvido.

Por questões estratégicas, agilidade na tomada de decisões, redução de custos, entre outras características, grande parte dos sistemas estão

integrados, o que aumenta a complexidade e consequentemente a necessidade de testes mais robustos para manter a qualidade.

Com base nisso, chegamos a um segundo pensamento que se faz necessário, apesar de já estar implícito: as soluções precisam funcionar, do ponto de vista técnico e de negócio, ou seja, não podemos nos preocupar apenas se o sistema está funcionando sem erros (visão técnica).

Devemos levar em consideração também se ele está atendendo ao propósito para o qual foi desenvolvido (visão de negócio). No entanto, durante um projeto de desenvolvimento de software, muitas vezes estas visões estão separadas, e é até natural que estejam, pois costumam ser exercidas por áreas diferentes.

Quando é indicado que um software apresenta problemas, não estamos nos referindo apenas à indisponibilidade de um site, lentidão do sistema ou a outra característica técnica. Podemos nos referir também ao fato de o software expor uma característica funcional/de negócio que não está correta.

Este é mais um ponto de atenção que é preciso ter para que no final do projeto possamos afirmar que o software possui qualidade. Antes de continuarmos, vale lembrar que erros, defeitos e falhas são conceitos distintos, porém, para simplificar o entendimento, iremos tratar como uma única definição: “inconformidade do software”.

São pelas razões mencionadas que existem os Testes de Software e as equipes de teste e qualidade. Com base no que foi exposto, ao adotar os testes e garantir uma visão unificada (técnica e funcional) do software, dificilmente os objetivos e resultados esperados da solução não serão alcançados.

Portanto, o objetivo deste artigo é ressaltar a necessidade dos testes e discorrer sobre conceitos importantes, relacionados à fase de planejamento, a qual representa a base para as demais etapas (análise, modelagem, implementação, etc.).

Como um exercício para os desenvolvedores que não gostam de testar ou que acham que a etapa de testes deve ser menor ou praticamente não existir,

sugerimos que refletem sobre os impactos que podem ocorrer quando um software falha. Independentemente da área de seguimento, podemos listar alguns problemas em comum:

- Prejuízos financeiros;
- Perda de credibilidade;
- Perde de qualidade e retrabalho;
- Perda de confiança;
- Ações legais.

A partir desses itens, podemos perceber o quanto os testes precisam ser levados a sério e colocados em um patamar de discussão onde a redução ou eliminação da fase de testes para atingir o prazo e custo desejados seja uma possibilidade.

Para desmitificar a questão de que é possível ter economia de custos ao reduzir a fase de teste, apresentamos nas Figuras 1 e 2 um cenário comum que podemos nos deparar por conta de retrabalho. Note que o cenário de propagação de erros por má/não execução de testes pode gerar custos e retrabalhos maiores ao longo do projeto, os quais certamente superariam os custos de uma fase de testes bem planejada e executada.

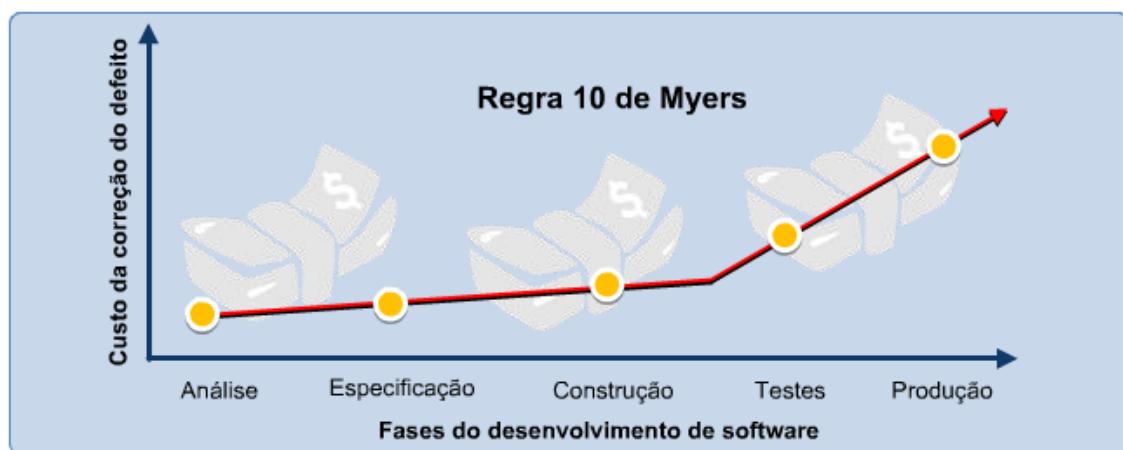


Figura 1. Fases de desenvolvimento de software x custo de correção

Nos requisitos	Na especificação	No código	Durante a execução do teste	Durante o teste de aceitação	Com o software em produção
- Alterar documento de requisitos.	- Alterar documento de requisitos. - Alterar a especificação.	- Alterar documento de requisitos. - Alterar a especificação. - Alterar o Código.	- Alterar documento de requisitos. - Alterar a especificação. - Alterar o Código. - Alterar o teste, reteste e teste de regressão.	- Alterar documento de requisitos. - Alterar a especificação. - Alterar o Código. - Alterar o teste, reteste e teste de regressão. - Refazer teste de aceitação.	- Alterar documento de requisitos. - Alterar a especificação. - Alterar o Código. - Alterar o teste, reteste e teste de regressão. - Refazer teste de aceitação. - Avisar usuários da falha e da solução de contorno. - O banco de dados do cliente pode necessitar de correções. - Montar um novo build do software pode levar horas.

Aumento do trabalho (Re-trabalho)



Figura 2. FAtividades a serem realizadas nas fases do projeto diante dos erros encontrados

Como esperado, este cenário não é aceitável em um projeto, ainda mais quando lidamos com soluções corporativas que exigem precisão cirúrgica de custos e prazos. Portanto, os testes de software não podem ser encarados apenas como mais uma fase do projeto, já que ela pode ser crucial para o sucesso ou fracasso do mesmo. Ao contrário do que muitos pensam, os testes começam muito antes da “simples” execução de testes em cima do código desenvolvido.

Talvez a seguinte definição nos ajude a ampliar um pouco a nossa visão sobre os testes: “Teste é o processo que consiste em todas as atividades do ciclo de vida, tanto estáticas quanto dinâmicas, voltadas para o planejamento, preparação e avaliação de produtos de software e produto de trabalhos relacionados, a fim de determinar se elas satisfazem os requisitos especificados e demonstra ...”

Conheça o MPT.Br: Melhoria do Processo de Teste de Software

Na globalização do mercado de software, cada vez mais competitivo, a busca por modelos de melhoria de processos está diretamente vinculada à necessidade da organização. Isto porque a gestão efetiva dos ativos organizacionais é crítica para o sucesso do negócio.

De acordo com o SEI, instituto que criou o Capability Maturity Model Integration – CMMI, “a qualidade de um sistema ou produto é amplamente influenciada pela qualidade do processo utilizado”. Além disso, um processo bem definido fornece uma estrutura para maximizar a produtividade da equipe, assim como, possibilita o uso de tecnologias para tornar a organização mais competitiva no mercado.

Dessa forma, os processos, provenientes de modelos de maturidade, têm por objetivo auxiliar as organizações na melhoria e no alcance dos resultados, através de uma melhor execução das atividades planejadas, além de minimizar os impactos quando da introdução e uso de novas tecnologias.

O modelo de Melhoria do Processo de Teste Brasileiro (MPT.Br) trata a melhoria do processo de teste através de melhores práticas relativas às atividades desenvolvidas ao longo do ciclo de vida de teste do produto.

O MPT.Br foi desenvolvido com o objetivo de melhorar a forma como os testes são executados, implementando níveis de maturidade e trazendo mais qualidade para os produtos entregues aos clientes, sem que haja um aumento significativo do custo.

Dentro deste contexto, este artigo apresenta os principais conceitos e estrutura do modelo MPT.Br, identificando e sugerindo a utilização das melhores práticas para a melhoria dos processos de teste de software.

O que MPT.Br?

O MPT.Br é um modelo de Melhoria do Processo de Teste idealizado para apoiar organizações através dos elementos essenciais para o desenvolvimento da disciplina de teste, inserida no processo de desenvolvimento de software. É utilizado para melhorar os processos de teste através de um conjunto de práticas relativas às atividades desenvolvidas ao longo do ciclo de vida de teste do produto.

A busca por modelos de melhoria está diretamente relacionada à demanda organizacional, visto que a efetiva gestão dos ativos organizacionais é crítica para o sucesso do negócio. Nesse contexto, os processos oriundos dos modelos de maturidade têm por objetivo auxiliar as empresas a aprimorarem seus processos e se tornarem mais maduras e eficientes, minimizando os impactos quando existir a introdução e uso de novas tecnologias.

Segundo o Guia de Referência do MPT.Br, os principais obje ...

Ainda nesse contexto, considerando o amplo uso das metodologias ágeis, é importante alinhar os testes ao processo de desenvolvimento e gestão como um todo.

Processo de teste ágil x tradicional

A área de testes, dentro da engenharia e desenvolvimento de software, vem ganhando cada vez mais espaço dentro de instituições de ensino, empresas e organizações, formando e qualificando profissionais, agregando conhecimento e contribuindo muito para melhoria da qualidade e valor de produtos, sejam eles de quaisquer tipos de sistemas.

Quando testadores participam de projetos onde os processos ou determinada metodologia escolhida não é a melhor, ou não flui de uma maneira eficiente, muitas vezes ficam desmotivados ou frustrados, pois o processo e a metodologia utilizados não são os melhores para a situação. Um processo mal

definido ou com má utilização contribui negativamente para fatores como alta rotatividade de pessoas, baixa eficácia de testes, atrasos ou demoras em entregas e, como resultado, temos um sistema instável, com muitos erros e clientes insatisfeitos.

Exemplificando essa situação, podemos pensar em um sistema cujo desenvolvimento não atinge o esperado nem pela empresa construtora, nem pela empresa contratante, a qual utiliza e paga pelo usufruto do sistema. Conclui-se que a empresa responsável pelas entregas ou desenvolvimento não possui processos bem definidos de testes e não trabalha na melhoria e evolução de seus produtos, escolhendo e gerenciando mal a metodologia utilizada. Em um mundo onde o mercado é cada vez mais competitivo, isso é inadmissível. Com a alta concorrência, jamais a empresa pode pecar em ter um produto razoável e de qualidade duvidosa porque a metodologia utilizada no ciclo de desenvolvimento é falha ou não suporta determinado tipo de projeto. Fazer uma escolha errada e não admitir alteração é pior do que fazer a escolha errada e reconhecer os erros. O mercado é muito exigente quando paga por um serviço e não recebe o acordado ou recebe com diversas falhas e erros.

Ao longo deste artigo, vamos mostrar e discutir sobre a importância de uma boa metodologia e sobre como defini-la para determinado projeto, uma vez que uma metodologia adequada para projetos pequenos e pouco complexos pode não se adequar bem para projetos maiores de alta complexidade.

O que é um projeto de software?

Podemos definir um projeto de software como sendo um empreendimento temporário dividido em ciclos de vida ou fases com o objetivo de criar um produto, serviço ou resultado único de forma a atender às especificações e expectativas de negócio. Os ciclos de vida de um projeto de software normalmente definem: qual trabalho técnico será realizado, quais entregas serão geradas em cada fase, como serão verificadas e validadas, quais as pessoas ou papéis estarão envolvidos em cada fase e como será realizado o controle e aprovação de cada fase.

Todo o processo de testes, metodologias, equipes e definições são realizados em projetos, por isso é crucial que o conceito esteja claro e bem definido.

Processo de testes de software

De forma geral, o processo de testes de software representa uma estruturação de atividades, etapas, artefatos, papéis e responsabilidades que buscam a padronização de equipes e trabalhos buscando uma forma de ampliar a organização, a qualidade e o controle dos projetos de teste.

Para melhor eficácia e eficiência, o processo de teste, como qualquer outro processo, deve ser revisto continuamente de forma a aperfeiçoar sua atuação buscando a melhoria contínua, além de analisar sua aplicabilidade e continuidade possibilitando aos profissionais envolvidos uma maior visibilidade e organização de seus trabalhos, o que resulta em uma maior agilidade e controle operacional dos projetos.

Independentemente das metodologias utilizadas nos projetos, o processo pode ser dividido em sete partes: planejamento dos testes, especificação dos testes, modelagem dos testes, disponibilização ou preparação do ambiente de testes, execução dos testes, análise dos resultados e encerramento do processo.

Segue um breve detalhamento da composição de cada parte:

- Planejamento dos testes: é onde será definido como os testes serão realizados tendo como base as restrições do cliente em relação a prazos, custos e qualidade esperada. Fazem parte das atividades de planejamento: estudar o projeto (quais serão os requisitos, alterações de arquitetura do sistema, avaliar custos, riscos, prazos e impactos), realizar uma análise interna e externa de qual será o esforço (especificar quais serão as métricas utilizadas e fazer uma estimativa do esforço), definir os cenários possíveis (lista de prioridades e riscos), aprovação do planejamento (por meio de aceite da diretoria e dos clientes), definir responsabilidades (qual o papel de cada um no projeto) e mapear os

artefatos (quais serão as documentações e onde será realizada a gestão do projeto).

- Especificação dos testes: nessa atividade são especificados os cenários e casos de teste considerando os requisitos especificados e a necessidade de cobertura dos testes a serem realizados. Casos de testes são escritos com base em requisitos ou documentos onde estão os critérios de aceite, casos de uso e checklists. Fazem parte das atividades de especificação: estudar os requisitos funcionais e não funcionais e demais documentos solicitados pelos clientes (com o objetivo de entender e identificar inconsistências ou pontos falhos), especificar adaptações da arquitetura dos testes (no que diz respeito às ferramentas utilizadas e exigidas, scripts de testes e de automação), identificar os casos de testes e cenários a serem validados (mapeando fluxos de casos de usos e garantindo que os requisitos estejam totalmente cobertos), refinamento e aceite dos casos de testes (por meio de uma reunião entre a equipe de teste, onde os casos de testes serão apresentados com o objetivo de encontrar pontos de melhoria e garantir a cobertura), definir as responsabilidades (qual o papel de cada membro na equipe de teste) e mapeamento dos artefatos (qual o padrão de casos de testes, quais ferramentas utilizar, onde reportar bugs).
- Modelagem dos testes: etapa onde é realizada a definição dos itens necessários para a implementação dos cenários de teste especificados. A tarefa central nessa etapa é a modelagem das massas de testes. Fazem parte das atividades de modelagem: criação dos roteiros de testes (identificação, especificação, organização e revisão dos roteiros e critérios de entrada/saída e de todos os procedimentos para execução dos testes), detalhamento da massa de testes (identificação dos pontos de validação, parametrização, detalhamento da massa de entrada/saída esperada), elaboração do plano de execução dos testes (identificação de quais sistemas estarão envolvidos, quais configurações de hardware, quais características e licenças de uso), quais artefatos e quais os papéis de cada membro da equipe também fazem parte da modelagem.
- Disponibilização ou preparação do ambiente de testes: atividades relacionadas à preparação do ambiente de testes da organização.

Fazem parte das atividades de preparação do ambiente: instalação ou atualização do aplicativo a ser testado (atualização do sistema com últimas alterações e implementações, que inclui scripts de banco de dados e códigos fonte), instalação e homologação da arquitetura de testes (caso existam pontos automatizados) e geração da massa de dados (criação via tela, banco de dados ou scripts de um conjunto de informações que possam ser utilizadas para validação das alterações). A definição de papéis de cada membro da equipe de testes também faz parte da preparação do ambiente.

- Execução dos testes: etapa na qual os testes planejados são executados. Nessa etapa a coleta de artefatos e informações é importante para a validação plena das funcionalidades, de forma a garantir que os critérios de aceite foram totalmente ou parcialmente cobertos. Fazem parte da execução as seguintes atividades: execução dos casos de testes (seguindo prioridade pré-definida, coleta e análise de evidências, validação de conformidade e não conformidade) e confirmação dos resultados (revalidação das não conformidades e conformidades, análise de impactos). O papel de cada membro dentro da equipe de testes também faz parte da execução dos testes.
- Análise dos resultados: nessa atividade os resultados da execução dos testes são comparados com os resultados esperados. Aqueles que não estiverem em conformidade devem ser reportados em detalhes para facilitar a identificação dos defeitos pela equipe de desenvolvimento. Fazem parte das atividades de análise de resultados: revisão dos resultados de não conformidade (avaliar evidências de testes e avaliar e priorizar a lista de bugs), formalizar defeitos encontrados (coletar, detalhar e classificar os defeitos por nível de severidade e priorização), reexecutar testes falhos (avaliar impactos, reexecutar casos de testes com falhas e possíveis casos de sucesso — dependendo da correção, um defeito pode ocasionar outro). Também faz parte da execução dos testes o papel de cada membro dentro da equipe e quais artefatos serão utilizados como forma de evidenciar os testes executados (como prints de telas, gravação, geração de scripts, passo-a-passo, etc.).

- Encerramento do processo: nesta atividade são analisados indicadores extraídos durante a execução dos testes para o trabalho realizado de forma quantitativa e qualitativa. O resultado dessa análise permitirá registrar as lições aprendidas durante a execução dos testes. Fazem parte da atividade de encerramento de processo: extração de indicadores (indicadores quantitativos, de produtividade, confiabilidade, financeiros e de nível de satisfação, individuais e do projeto), resumo do processo de testes (registrar e analisar os indicadores de testes, lista de defeitos, horas planejadas x horas executadas, indicadores de sucesso, lições aprendidas, caminhos críticos e realizar uma divulgação corporativa sobre os resultados obtidos), versionamento dos processos de testes (versionar scripts, casos de testes, ferramentas, códigos fontes utilizados na automação, configurações e ferramentas de produtividade utilizadas), avaliação final e melhoria do processo (avaliar riscos planejados x concretizados, performance, atualizar plano de melhoria contínua e sinalizar o encerramento do processo).

De um modo geral, de uma metodologia para outra, poucas atividades são alteradas para os testadores. Normalmente o que difere é a ordem e a forma como as atividades são feitas, mas geralmente o processo não sofre alteração na sua essência. As poucas diferenças existentes serão tratadas e exploradas nos tópicos a seguir.

Desenvolvimento de software

Desenvolvimento de software é uma tarefa muito complexa, difícil e de alto risco. Inúmeros problemas são enfrentados diariamente, e nós, testadores, não estamos excluídos nesse processo, aliás, muito pelo contrário, como somos praticamente o final do processo, somos a ligação entre desenvolvedor e usuário.

Entre alguns desses problemas enfrentados, podemos destacar: alto consumo de tempo que supera estimativas e cronograma, funcionalidades que não solucionam os problemas dos usuários, baixa qualidade dos sistemas

desenvolvidos, tempo para teste e qualidade reduzidos e, em alguns casos, até o cancelamento do projeto por inviabilidade.

Tendo como grande objetivo a redução de risco associado ao desenvolvimento de software, as metodologias definem uma maneira de trabalhar em projetos utilizando processos, sendo elas um conjunto de atividades e resultados associados que auxiliam no desenvolvimento e construção de softwares.

Teste de software no Scrum

O **Scrum** é utilizado no processo de desenvolvimento de software ágil. Mas ao invés de um processo completo ou metodologia, é um framework. Então, em vez de fornecer completas descrições detalhadas de como tudo deverá ser feito no projeto, muito é deixado para a equipe de desenvolvimento de software. Isso é feito porque a equipe que define a melhor forma de executar as tarefas, nunca podendo ser dirigida e/ou direcionada por alguém que não pertença ao time. Apenas o time tem a competência necessária para definir como "atacar" as tarefas e quem deve "atacá-las", sempre levando em consideração a habilidade de cada integrante, ou seja, se um membro do time tem uma maior experiência em uma determinada tarefa, muito provavelmente ele será o responsável por executá-la. Além disso, uma das técnicas mais amplamente aplicáveis introduzidas pelos processos ágeis é expressar os requisitos do produto sob a forma de histórias do usuário.

Scrum se concentra em fornecer recursos valiosos de negócios em iterações de duas a quatro semanas de desenvolvimento. Embora a metodologia ágil Scrum possa ser utilizada para o gerenciamento de qualquer projeto, o processo ágil é ideal para projetos com mudanças frequentes de requisitos ou para projetos curtos. Cada time Scrum possui três papéis:

1. **ScrumMaster**, que é responsável por garantir que o processo seja entendido e seguido;

2. **Product Owner**, que é responsável por maximizar o valor do trabalho que o time Scrum faz;
3. **Time**, que executa o trabalho propriamente dito. O time é formado por desenvolvedores, designers e analistas de teste com todas as habilidades necessárias para transformar os requisitos do Product Owner em um pedaço potencialmente entregável do produto ao final da Sprint.

No início de um Sprint os membros do time de Scrum se reúnem com o PO (Product Owner) ou dono do produto que representa o cliente, sendo ele responsável por definir, priorizar e organizar o backlog maximizando assim o valor do produto. Esse planejamento tem por objetivo fazer com que os membros da equipe entendam os requisitos desejados e em conjunto definam quais características da aplicação serão desenvolvidas naquele Sprint, assim como qual será a duração do Sprint. A partir dessa definição o projeto começa a ser executado e todos os membros iniciam sua atuação. É nesse momento que os analistas de testes se questionam: como garantir a qualidade de um produto em tão pouco tempo? Como escrever casos de testes e executá-los tão rapidamente? Quais serão os artefatos do projeto?

Modelagem e Casos de Testes

O propósito da técnica de modelagem de teste é identificar as condições e os casos de testes. Isso visa garantir que os testes serão executados com dados e formas que realmente simulam o comportamento real do software e validam todas as possibilidades de uso.

Técnicas de modelagem de teste na prática

Teste de software é um componente crítico da garantia de qualidade de software e representa uma revisão final da especificação, do projeto e do código construído. O aumento da visibilidade do software como um elemento

do sistema e os "custos" de manutenção associados a uma falha, são questões motivadoras para a execução de um teste rigoroso e bem planejado.

Não é raro uma empresa de desenvolvimento de software gastar entre 30% e 40% do esforço total do projeto na etapa de teste. A rigor, o teste de softwares que envolvem vidas (por exemplo, sistemas para medicina, monitoramento de reatores nucleares) pode custar de três a cinco vezes mais do que todas as outras fases de engenharia de software juntas.

Segundo Myers, "testar um programa é analisar um programa com a intenção de descobrir erros e defeitos". Ou seja, durante os testes certificamos se o software foi implementado corretamente, isto é, verificamos o software; e também certificamos se ele foi construído de acordo com seus requisitos, ou seja, validamos o software.

Após o desenvolvimento do código fonte, o software deve ser testado para encontrar (e corrigir) tantos erros quanto possíveis antes de ser entregue ao cliente.

O propósito do teste de software não é provar que um programa está bem escrito, mas sim descobrir todos os defeitos que por ventura existem nele.

Através dos testes não demonstramos a ausência de erros ou defeitos, apenas evidenciamos os existentes. Um bom teste é aquele que exercita diversas porções do software fazendo-o falhar.

Assim, dizemos que o teste de software é uma atividade de natureza "destrutiva" e não "construtiva". De igual importância a examinar se um software realiza as tarefas para as quais foi projetado, é identificar o que ele faz não intencionalmente.

Sabemos que o custo de reparos no software aumenta ao longo do tempo, portanto, quanto mais tarde um defeito for encontrado, maior será o custo da sua correção. Daí a importância da realização de testes desde o início de um projeto.

De uma forma simples, testar um software significa verificar através de uma execução controlada se o seu comportamento ocorre de acordo com o especificado.

O objetivo principal desta atividade é revelar o número máximo de falhas dispendo do mínimo de esforço, mostrando aos que desenvolvem se os resultados estão ou não de acordo com os padrões que foram estabelecidos.

Ao longo deste artigo, iremos discutir os principais conceitos relacionados às atividades de teste e as principais técnicas de modelagem de teste que podem ser utilizados para verificação ou validação de um produto, assim como exemplos práticos da aplicação de cada tipo de técnica.

Processo de Teste

Pode-se entender um programa como sendo uma função que descreve uma relação entre um elemento de entrada e um elemento de saída. O processo de teste é utilizado para garantir que este programa realize integralmente esta função.

Os elementos essenciais de um teste são:

- O software executável ou o código fonte;
- Descrição do comportamento esperado do software;
- Descrição do seu domínio funcional (entradas).

O processo de teste consiste, então, em obter um valor válido do seu domínio funcional (ou inválido), definir o comportamento esperado do software para o valor escolhido, executá-lo, observar seu comportamento e, por fim, comparar este comportamento com o esperado.

Se o comportamento atual não coincidir com o esperado, este teste descobriu (no sentido de tornar visível) um erro.

A completa verificação de um software pode ser obtida através de um teste exaustivo, isto é, a aplicação do processo de teste para todos os valores do domínio (entradas). No entanto, comumente as entradas são infinitas ou, quando finitas, são suficientemente grandes tornando inviável a quantidade de testes necessária para cobri-las. Por exemplo:

```
int FuncaoA(char ch);
```

```
int FuncaoB(int x, int y);
```

A partir da modelagem são gerados os casos de teste, que são conjuntos de regras e dados usadas para testar o software. Normalmente os casos de teste estão vinculados a requisitos e visam simular da forma mais fiel possível o uso do sistema, a fim de garantir que ele funcionará corretamente quando submetido à utilização real.

Casos de Teste: Importância da rastreabilidade entre requisitos

Teste pode ser definido como uma atividade que tem por objetivo verificar se o software produzido está de acordo com sua especificação e se satisfaz as expectativas do cliente. Teste de software é parte integrante do processo de Validação e Verificação (V&V) da Engenharia de Software.

Sendo que, verificação refere-se ao conjunto de tarefas que garantem que o software implementa corretamente uma função específica, e validação refere-se ao conjunto de tarefas que asseguram que o software foi criado e pode ser rastreado segundo os requisitos do cliente. A definição de V&V abrange muitas atividades de garantia da qualidade de software [4].

Das atividades de verificação e validação, a atividade de teste é considerada um elemento crítico para garantia da qualidade. Um problema na atividade de teste de software está relacionado a como determinar se um programa P foi suficientemente testado e pode ser liberado para os usuários com razoável confiança de que funcionará de modo aceitável.

O significado de aceitável varia para uma aplicação específica em função da criticidade das funções, consequências antecipadas de um mau funcionamento e da frequência de uso esperada. Apesar dos esforços na utilização de modelos de melhoria de processos em organizações de desenvolvimento de software, os padrões para determinar a adequação da atividade de teste são praticamente inexistentes. Um conjunto de problemas no processo de teste que devem ser evitados são:

- Cronogramas agressivos deixando a equipe de teste impossibilitada de completar os testes planejados devido à falta de recursos, equipe não qualificada e falta de tempo;
- Falta de rastreabilidade de casos de teste entre diferentes versões do sistema, dificultando o reuso e repetição dos testes após modificações nos requisitos;
- Testes manuais, pois há um grande esforço da equipe de testes a cada início de uma nova atividade de teste, consequentemente quando o software sofre manutenção e a documentação não é atualizada;
- Ambiente de teste diferente do ambiente de produção;
- Ausência de critérios para seleção dos casos de teste, definição da sua completude e estabelecimento de um ponto de parada.

Este artigo aborda a rastreabilidade de requisitos, enfatizando aspectos relacionados às informações de rastreabilidade entre requisitos e casos de teste ...

Casos de Teste: Aprimore seus casos e procedimentos de teste

Mentoring: Casos de teste são elementos essenciais para o sucesso das atividades de teste em um projeto de software. São eles que definem as entradas a serem informadas pelo testador (manualmente ou com apoio ferramental) e os resultados esperados a partir desta ação. Assim, eles nos permitem medir o quanto o software está sendo testado. Já um procedimento de teste define os passos/sequência necessários para executar os casos.

Estes visam apoiar na customização do esforço de teste em um projeto. Neste artigo, serão discutidas estratégias para melhor especificação de casos e procedimentos de teste em projetos de software. Para isso, será utilizado um exemplo simples e comum de ser encontrado em sistemas de informação, um formulário de cadastro de venda de produtos online com diversas regras de negócio para validação dos dados preenchidos, e realizaremos a especificação dos casos e procedimentos de teste de forma a torná-la mais completa e objetiva, apoiando assim testes manuais e automáticos.

Qualquer profissional da área de teste de software possivelmente conhece e/ou já precisou especificar casos e/ou procedimentos de teste.

Segundo Craig e Jaskiel (2002), estes artefatos do processo de teste de software podem ser definidos da seguinte forma:

- **Caso de Teste:** descreve uma condição particular a ser testada e é composto por valores de entrada, restrições para a sua execução e um resultado ou comportamento esperado.

- **Procedimento de Teste:** é uma descrição dos passos necessários para executar um caso (ou um grupo de casos) de teste.

Aqueles com um pouco mais de experiência profissional devem perceber que uma das dificuldades enfrentadas na atividade de teste é a especificação dos casos e procedimentos de testes, não apenas pela dificuldade natural exigida pela tarefa, mas pelas variações de formato de descrição destes artefatos em diferentes locais.

Apesar de existir um padrão internacional para a especificação de casos e procedimentos de teste publicado no padrão IEEE-Std-829 (roteiro de documentos para a atividade de teste em geral, não apenas casos e procedimentos de teste, cuja versão mais atual foi publicado em 2008), este padrão em geral serve apenas como um roteiro e normalmente é customizado pelas empresas, em certas ocasiões de forma correta, porém em outras, de forma indevida.

Na verdade, o padrão IEEE-Std-829 foi proposto exatamente com o intuito de servir como base para que empresas de software pudessem definir o seu roteiro de documentos para as atividades de teste de software. Cada organização possui suas especificidades e isso precisa ser refletido no roteiro utilizado para documentar suas atividades de teste.

Assim, se um determinado formato funciona adequadamente para a empresa X, é por que ele contém o conjunto de informações necessárias para que os demais membros da equipe de teste desempenhem de forma satisfatória suas atividades. Contudo, não há garantia de que mesmo este formato funcionando adequadamente em uma empresa, ele possa ser utilizado de forma satisfatória por outra com características ou perfil diferentes. Maiores ou menores níveis de detalhamento na descrição podem ser necessários.

Apesar de não existir um padrão universal ou um consenso entre empresas de software no que diz respeito à especificação, um conjunto de boas práticas tem sido estabelecido ao longo dos anos e que, se consideradas, podem contribuir positivamente para que a especificação dos casos e procedimento de teste

seja realizada de forma mais satisfatória. Neste contexto, satisfatória significaria poder ser utilizada por outros membros da equipe de teste (analistas de teste ou testadores, por exemplo) sem a necessidade de novas intervenções na especificação para que ela possa ser melhor compreendida.

Neste artigo, partiremos de um caso de uso simples para realizar a especificação de casos e procedimentos de teste para um formulário de compra online de produtos. Ao final, teremos uma especificação bem definida e que irá considerar várias das práticas que têm sido adotadas por equipes de teste de software.

Gerenciamento de testes

Para garantir que os testes serão realizados de forma eficiente, é necessário haver um gerenciamento sobre as atividades e artefatos relacionados a esse processo. No artigo a seguir você verá uma introdução ao assunto:

Testabilidade e planejamento de testes de software

A área de testes de software vem se tornando cada vez mais importante para que o desenvolvimento alcance um alto nível de qualidade e confiabilidade. Um software que apresenta erros e falhas após a sua entrega para o usuário final fatalmente não irá obter uma boa aceitação. Além disso, os desenvolvedores acabam gastando muito tempo para a correção de erros muitas vezes complexos que poderiam ter sido identificados na fase inicial. Tudo isso pode aumentar significativamente o custo do desenvolvimento. O ideal é que os testes de software sejam parte essencial desde o planejamento do projeto, passando por todas suas etapas e indo até a finalização e entrega do software totalmente funcional.

Estratégias e estruturas de testes adequadas devem ser criadas em cada etapa do desenvolvimento para que os testes sejam realizados no software. O

objetivo é procurar os possíveis erros de forma automatizada, quando viável, facilitando o trabalho do programador ou responsável pela execução dos testes e preparando o software para que esses sejam facilmente elaborados e efetuados.

No entanto, a utilização apropriada das técnicas de testes de software ainda não é uma prática muito difundida entre as equipes de desenvolvimento. Muitos não preparam rotinas de testes principalmente por não compreenderem a sua importância. A realização dos testes contribui claramente para o aumento da qualidade do programa e resulta na satisfação do usuário final e na conformidade com os requisitos que foram identificados no projeto.

Os testes de software e a garantia da qualidade

A principal finalidade de se aplicar os fundamentos da engenharia de software no desenvolvimento é a criação de softwares dentro dos prazos e custos planejados e, sobretudo, com alta qualidade. A qualidade de software é definida pela norma ISO/IEC 9126 como sendo a capacidade de um produto ou serviço apresentar funcionalidades e características que atendam totalmente às necessidades específicas ou implícitas dos usuários. A garantia da qualidade reúne processos que definem como alcançar a qualidade do software e como a equipe de desenvolvimento deverá se comportar para satisfazer o nível de qualidade exigido.

Verificação e Validação (V&V) é um dos processos de garantia de qualidade do software, e abrange várias atividades, entre elas o teste de software. A verificação deve garantir a consistência interna do produto, ou seja, certificar que suas especificações estejam sendo atendidas. Já a validação utiliza as referências externas do usuário final para assegurar que o produto construído satisfaz suas necessidades e expectativas.

Os testes de software são conjuntos de funções e atividades que são executadas com o objetivo de encontrar erros cometidos na construção de um software. Além de buscar e descobrir falhas no software, os testes têm como

meta comprovar que o produto atende aos requisitos e que está em conformidade com suas especificações. É importante citar que apesar dos testes contribuírem significativamente para a confiabilidade do software, eles não podem assegurar a qualidade de um produto, pois alcançar a qualidade de um software depende de outros fatores.

Estratégias de testes de software

Para executar os testes de software é preciso, inicialmente, definir uma estratégia de teste, fornecendo um roteiro que indica o caminho a ser seguido e definindo como e quando os passos referentes aos testes serão executados, bem como o tempo, esforço e recursos necessários. Uma estratégia de teste bem definida deve incluir o planejamento dos testes, o projeto dos casos de teste, a execução dos testes e, por fim, a coleta e avaliação de resultados.

O planejamento dos testes deve ser feito para garantir que se obtenham os resultados esperados, decidindo onde os erros podem ser encontrados e projetando os testes mais eficientes para localizá-los. Uma das maneiras de se criar um plano de teste é baseá-lo em riscos, ou seja, em uma parte do código em que a probabilidade de ocorrência de uma falha seja maior ou em um módulo específico onde o impacto do erro poderia comprometer o funcionamento do produto.

Um caso de teste deve documentar um teste com o objetivo de provar uma exigência. Deve haver pelo menos um caso de teste por requisito, mas algumas vezes são utilizados vários casos de testes para provar um único requisito. Também se pode utilizar o mesmo caso de teste em muitas situações a fim de verificar por completo uma determinada exigência.

A criação de projetos de casos de teste eficazes é um dos tópicos mais importantes relacionados aos testes de software, pois técnicas eficientes de casos de teste, utilizadas em conjunto, possibilitam a identificação da maioria dos erros. Entre essas técnicas estão:

- **Teste de caixa-preta:** seu objetivo é descobrir situações em que o software não se comporta de acordo com as suas especificações, sem se preocupar com os seus aspectos estruturais. Deve ser derivada de uma série de condições de entrada para que o programa seja testado com foco em seus requisitos funcionais, sendo que essas entradas podem ser válidas ou não;
- **Teste de caixa-branca:** testa cada caminho no software pelo menos uma vez utilizando técnicas específicas para executar decisões lógicas e c ...

Como elaborar um Roteiro de Testes

O Roteiro de Teste é uma maneira de realizar testes manuais em softwares, como por exemplo, em Testes Funcionais. Este roteiro é elaborado a partir dos documentos de especificação de um determinado caso de uso, como: especificação funcional, guia de interface e modelagem do banco de dados. O roteiro de teste também é conhecido como Projeto de Teste, Script de Teste ou Especificação de Teste. Ele é importante no momento da execução dos testes, pois o testador consegue realizar uma sequência de passos de forma prática, sem a necessidade de consultar todos os documentos de especificação no momento dos testes, podendo ficar focado apenas na execução dos testes.

Além do que é especificado pelo cliente, o roteiro de teste também possui procedimentos que testam a eficiência do sistema. Em geral um roteiro de teste é composto por um conjunto de Casos de Teste, mas além dos casos de teste há também as seções de Localização e de Objeto de Teste, descritas a seguir.

As seções de **Localização** do roteiro de teste servem para definir em qual tela do sistema será executado um sub conjunto de casos de teste, os quais fazem parte daquela localização. A **Localização** pode ser escrita da seguinte forma:
Tela Consultar Funcionários > Tela Manter Funcionários.



Estrutura de um Roteiro de Teste (Fonte: [GTSW](#))

Uma **Localização** pode conter 1 (um) ou mais **Objetos de Teste**, onde este pode ser definido como a ideia global de um conjunto de **Casos de Teste**. Por exemplo, na localização descrita anteriormente, Tela Manter Funcionários, um possível Objeto de Teste seria Cadastro de funcionários no sistema. Outro objeto de teste para esta mesma localização poderia ser Alteração de funcionários no sistema.

O **Objeto de Teste** pode conter 1 (um) ou mais Casos de Teste, pois são nos casos de teste onde estão inseridos os procedimentos necessários para a execução de um determinado teste no sistema.

Um **Caso de Teste** é composto por uma descrição, por uma pré-condição, pelo procedimento e pelo resultado esperado, que será definido a seguir.

- Na **Descrição** está descrita a ideia específica do caso de teste;

- A **pré-condição** é um requisito para o comportamento do sistema antes da execução do caso de teste;
- No **procedimento** estão os passos para a execução do caso de teste, este procedimento não deve fugir do foco descrito na descrição do caso de teste;
- O **resultado esperado** descreve como o sistema deveria se comportar após a execução do procedimento do caso de teste.
-