Verificação, validação e testes de software

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Explicar o processo de verificação e validação de software.
- Definir verificação e validação estáticas e dinâmicas.
- Categorizar os tipos de testes de software.

Introdução

Neste capítulo, você vai aprender sobre verificação, validação e testes de software, aspectos fundamentais para que possamos garantir ou atrelar qualidade ao software sendo desenvolvido. Dessa forma, veremos que um software traz uma relativa complexidade, a qual exige que sejam aplicados diversos tipos de software, para que, no final, o produto execute com eficácia todas as funcionalidades estabelecidas e esperadas pelo cliente.

Verificação e validação de software

Quando adquirimos algum produto, independentemente do que seja, provavelmente ele passou por uma bateria de testes para que fosse colocado no mercado. Com o software, espera-se que aconteça o mesmo. Mas você já imaginou que uma verificação, validação ou teste de algo pode ser uma oportunidade e tanto para os problemas aparecerem? Isso porque essas etapas não estão na lista das atividades mais queridas de uma equipe de desenvolvimento. Imagine você ter horas de trabalho, deduzir que finalizou tudo, e depois descobrir diversos erros? Geralmente é isso que ocorre, pois um software é tão complexo que se torna praticamente impossível nenhum erro ser encontrado. A garantia de que o software execute pelo menos a maioria das funcionalidades definidas pelo usuário é sinônimo de qualidade. Basta imaginar que qualquer produto

que passe por uma série de testes, consequentemente, estará menos apto a apresentar algum defeito futuro.

Segundo Paula Filho (2009, p. 456), a garantia da qualidade, para o glossário do Institute of Electrical and Electronics Engineers (IEEE), é definida como "conjunto planejado e sistemático de ações necessárias para estabelecer um nível adequado de confiança de que um item ou produto está em conformidade com seus requisitos técnicos". Essas ações incluem ações preventivas, como o uso de processos e ferramentas adequadas e a capacitação das equipes no uso desses processos e ferramentas. A definição de garantia da qualidade constante do Capability Maturity Model Integration (CMMI) enfatiza a aderência aos processos: "conjunto planejado e sistemático de meios para garantir à gerência que os padrões, métodos, práticas e procedimentos definidos por um processo são aplicados".

A diferença entre validação e verificação, conforme Boehm (1979 apud SOMMERVILLE, 2011, p. 145), é o fato de na validação se perguntar se o produto certo está sendo construído, e na verificação a pergunta ser voltada para o processo de produção, ou seja, será que o produto está sendo construído da maneira correta? Inclusive é muito comum localizarmos na literatura referente ao assunto a validação e a verificação como "V&V". Para muitos, o desenvolvimento de um software é relativamente fácil, já que há aplicativos/software para quase tudo; realmente, a velocidade em que surgem as novidades no mercado tecnológico faz com que o processo pareça simples. Em contrapartida, quem está dentro dele sabe perfeitamente que é tudo muito complexo e minucioso. A etapa de testes e aplicação da validação e verificação é apenas parte de uma das diversas fases de desenvolvimento de um software.

Conforme Paula Filho (2009, p. 456), as apreciações intermediárias ou de trabalho compreendem as tarefas de verificação, definidas pelo IEEE como "processo de avaliar um sistema, produto ou componente para determinar se os resultados de um passo do respectivo processo de desenvolvimento satisfazem as condições impostas no início do passo". As apreciações finais ou de qualificação incluem as tarefas de validação, definidas pelo IEEE como "processo de avaliar um sistema, produto ou componente no final do processo de desenvolvimento para avaliar se ele satisfaz aos requisitos especificados". Dessa forma, podemos deduzir que verificação se refere ao conjunto de tarefas que garantem que o software implementa corretamente uma função específica. Já a validação se refere a um conjunto de tarefas que asseguram que o software foi criado e pode ser rastreado segundo os requisitos do cliente (PRESSMAN; MAXIM, 2016, p. 467- 468).

Saiba mais

Conforme o CMMI (*Capability Maturity Model Integration* — em português, Modelo Integrado de Maturidade em Capacitação), as tarefas de verificação são definidas como "confirmação de que produtos de trabalho refletem corretamente os requisitos especificados para eles", e as de validação como "confirmação de que o produto, tal como será provido, atenderá ao uso pretendido" (PAULA FILHO, 2009, p. 456).

Link

O IEEE é a maior organização profissional técnica do mundo dedicada ao avanço da tecnologia em benefício da humanidade. Mais informações disponíveis em:

https://goo.gl/EIcJPs

Verificação e validação estáticas e dinâmicas

Complementando o que falamos anteriormente, segundo Pressman e Maxim (2016, p. 468), a verificação e a validação incluem uma ampla gama de atividades de SQA (*software quality assurance* — garantia da qualidade de software): revisões técnicas, auditorias de qualidade e configuração, monitoramento de desempenho, simulação, estudo de viabilidade, revisão de documentação, revisão de base de dados, análise de algoritmo, teste de desenvolvimento, teste de usabilidade, teste de qualificação, teste de aceitação e teste de instalação. Embora a aplicação de teste tenha um papel extremamente importante em V&V, muitas outras atividades também são necessárias. O teste proporciona o último elemento a partir do qual a qualidade pode ser estimada e, mais pragmaticamente, os erros podem ser descobertos.

O teste dinâmico (*dynamic testing*) traz atividades de planejamento, execução e controle do teste e necessita que o software seja executado. É o mais utilizado pelas empresas de desenvolvimento de softwares, e o custo das correções dos defeitos tende a ser mais alto. O teste estático (*static testing*) inclui atividades de revisão, inspeção e análise estática do código. A revisão

e a inspeção podem ser realizadas em qualquer documentação do projeto e do código-fonte durante a etapa de construção do software. Podemos afirmar que o teste estático traz a análise do código, enquanto o teste dinâmico testa o código que foi analisado, conforme se pode observar na Figura 1.

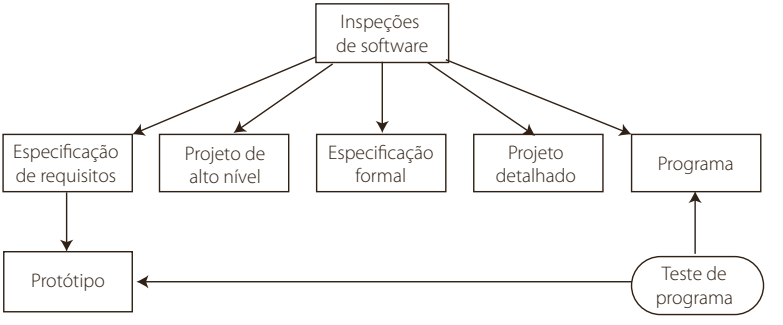

Figura 1. Verificação e validação dinâmica.
Fonte: Adaptada de Sommerville (2007, p. 342).

Segundo Paula Filho (2009, p. 351), podemos classificar os testes também como: testes de sistema e testes de desenvolvimento. Os primeiros têm por objetivo verificar a conformidade com os requisitos; para maior eficácia, devem ser especificados e executados por uma equipe de testes, independentemente da equipe de desenvolvimento. Os testes de desenvolvimento são procedimentos de verificação, executados pelos próprios desenvolvedores, com o objetivo de determinar se os resultados de determinadas atividades e tarefas satisfazem aos objetivos que pretendiam atingir.

É importante que os testes sejam bem planejados e desenhados, para conseguir-se o melhor proveito possível dos recursos alocados para eles. Testes irreproduzíveis e improvisados são quase inúteis, e devem ser usados, no máximo, para complementar os testes planejados. Durante e após a realização dos testes, os resultados de cada teste devem ser minuciosamente inspecionados, comparando-se resultados previstos e obtidos, pois nem sempre é óbvio quando um teste detectou um defeito (PAULA FILHO, 2009, p. 350).

Outros termos bastante conhecidos são os testes de caixa branca e os testes da caixa preta. Sob o ponto de vista de Pressman e Maxim (2016, p. 509), o teste caixa-preta, também chamado de teste comportamental ou teste

funcional, focaliza os requisitos funcionais do software. As técnicas de teste caixa-preta permitem derivar séries de condições de entrada que utilizarão completamente todos os requisitos funcionais para um programa. O teste caixa-preta tenta encontrar erros nas seguintes categorias:

1. funções incorretas ou ausentes;
2. erros de interface;
3. erros em estruturas de dados ou acesso a bases de dados externas;
4. erros de comportamento ou de desempenho;
5. erros de inicialização e término.

No Quadro 1, podemos observar detalhes sobre os testes funcionais e suas respectivas características.

Quadro 1. Testes funcionais e suas características

Testes funcionais	Descrição do teste	Quando executar
Tratamento de erros	Verifica se o sistema valida todas as transações e se retorna todas as mensagens de erro (*feedback*) no caso de receber informações erradas.	Em todo o ciclo de desenvolvimento.
Requisitos	Verifica se o sistema executa todas as funcionalidades conforme o elicitado no documento de requisitos.	A cada iteração.
Regressão	Testa o impacto de novas funcionalidades sobre as já existentes e deve também ser realizada na documentação, principalmente se houver alterações.	A cada iteração.
Suporte manual	Testa o sistema de ajuda (*help*) sensível ao contexto e verifica se a documentação está completa e devidamente atualizada.	Na homologação (entrega do sistema).

(Continua)

(Continuação)

Quadro 1. Testes funcionais e suas características

Testes funcionais	Descrição do teste	Quando executar
Controle	Testa se o processamento corresponde ao esperado, principalmente em procedimentos alheios à aplicação, por exemplo, *backups* e recuperação de dados.	Pode ser executado a qualquer momento.
Interconexão	Garante a comunicação dos módulos desenvolvidos, bem como sua integração com outros sistemas (se houver).	Pode ser executado a qualquer momento.
Paralelo	Testa paralelamente o sistema em comparação ao sistema antigo, ou seja, verifica se atende às mesmas especificações, comparando resultados.	A cada iteração.

Fonte: Adaptado de Sbrocco e Macedo (2012, p. 187).

Para Pressman e Maxim (2016, p. 500), o teste caixa-branca, também chamado de teste da caixa-de-vidro ou teste estrutural, é uma filosofia de projeto de casos de teste que usa a estrutura de controle descrita como parte do projeto no nível de componentes para derivar casos de teste. Usando métodos de teste caixa-branca, o engenheiro de software pode criar casos de teste que:

1. garantam que todos os caminhos independentes de um módulo foram exercitados pelo menos uma vez;
2. exercitem todas as decisões lógicas nos seus estados verdadeiro e falso;
3. executem todos os ciclos em seus limites e dentro de suas fronteiras operacionais;
4. exercitem estruturas de dados internas para assegurar a sua validade.

No Quadro 2, a seguir, podemos ver algumas características dos ditos testes estruturais.

Quadro 2. Testes estruturais e suas características

Testes estruturais	Descrição do teste	Quando executar
Unidade	Testa as funções, classes e objetos do sistema, considerando a performance e a lógica utilizada.	Em todo o ciclo de desenvolvimento.
Execução	Analisa o desempenho do sistema com dados reais, testando a performance com múltiplos acessos simultaneamente.	No início e na homologação.
Estresse	Testa os limites máximo e mínimo do sistema, a fim de avaliar seu comportamento em condições adversas.	A cada iteração e na homologação.
Recuperação	Testa como um sistema pode recuperar-se de problemas físicos ou lógicos, desde falhas elétricas até de componentes de *hardware* e rede (contingência).	Na homologação ou a qualquer momento.
Operação	Avalia o funcionamento do sistema, comparando com os processos manuais da empresa.	A cada iteração.
Conformidade	Verifica se o sistema foi feito de acordo com as normas e os padrões previamente estabelecidos (*patterns*).	Em todo o ciclo de desenvolvimento.
Segurança	Teste de confiabilidade que assegura se o sistema está preparado para impedir acessos não autorizados ou invasões, protegendo seus dados quanto a isso.	Na homologação ou a qualquer momento.
Integração	Realizado pelo analista, conferindo e validando o que foi proposto nos casos de uso e o que foi projetado.	A cada iteração.

(Continua)

(Continuação)

Quadro 2. Testes estruturais e suas características

Testes estruturais	Descrição do teste	Quando executar
Sistemas	Testa todo o sistema, utilizando cenários preestabelecidos a fim de confrontar resultados.	Na homologação ou a qualquer momento.
Aceitação	Testa a validação final do sistema com o cliente, liberando para o uso final.	Na homologação.

Fonte: Adaptado de Sbrocco e Macedo (2012, p. 188).

Tipos de testes de software

Um teste pode ser visto como uma coleção de procedimentos e casos de teste. Um procedimento de teste é um conjunto detalhado de instruções para execução de testes. Um caso de teste é uma especificação das entradas, dos resultados previstos e das condições de execução para um item a testar, em que um procedimento de teste pode ser invocado em vários casos de teste, de forma direta, ou por meio de outros procedimentos de teste. Os procedimentos de teste contêm uma sequência de ações que devem ser executadas para realizar um grupo de testes semelhantes. Tipicamente, cada procedimento de teste corresponde a um dos roteiros importantes que podem ser seguidos na execução de um fluxo de uma colaboração de uso, que por sua vez deriva de um fluxo de caso de uso. Um procedimento pode ser executado de forma manual ou automatizada (PAULA FILHO, 2009, p. 350).

Conforme Sommerville (2011, p. 144), o processo de teste tem dois objetivos distintos:

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

Miller (1977 apud PRESSMAN; MAXIM, 2016, p. 468) relaciona teste de software com garantia da qualidade afirmando que "a motivação que está por trás do teste de programas é a confirmação da qualidade do software

com métodos que podem ser econômica e efetivamente aplicados a todos os sistemas, de grande e pequena escala".

A seguir, na Figura 2, podemos visualizar alguns tipos de testes e como eles estão incluídos em uma estratégia de testes que deve ser utilizada em um processo de desenvolvimento de software.

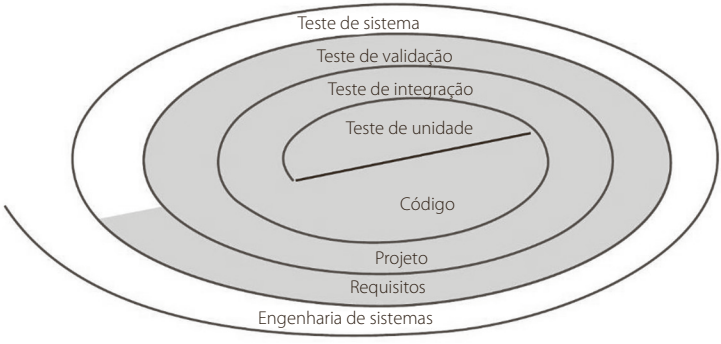

Figura 2. Estratégia de teste.
Fonte: Pressman e Maxim (2016, p. 470).

Teste de unidade

O teste de unidade focaliza o esforço de verificação na menor unidade de projeto do software — o componente ou módulo de software. Usando como guia a descrição de projeto no nível de componente, caminhos de controle importantes são testados para descobrir erros dentro dos limites do módulo. A complexidade relativa dos testes e os erros que revelam são limitados pelo escopo restrito estabelecido para o teste de unidade. Esse teste enfoca a lógica interna de processamento e as estruturas de dados dentro dos limites de um componente. Esse tipo de teste pode ser conduzido em paralelo para diversos componentes (PRESSMAN; MAXIM, 2016, p. 473).

Teste de integração

O teste de integração consiste em uma técnica utilizada para construir a arquitetura de software e realizar testes para descobrir erros associados às

interfaces. Busca-se, assim, construir uma estrutura de programa determinada pelo projeto a partir de componentes testados em unidade (PRESSMAN; MAXIM, 2016). Conforme Schach (2010, p. 499), cada artefato de código novo tem de ser testado quando for adicionado àquilo que já foi integrado. Quando o produto tiver uma interface gráfica com o usuário, podem surgir problemas em relação aos testes de integração. Em geral, testar um produto normalmente pode ser simplificado por meio do armazenamento dos dados de entrada para um caso de teste em um arquivo. O produto é, então, executado, e os dados relevantes lhe são submetidos. Com o auxílio de uma ferramenta CASE, o processo todo pode ser automatizado, ou seja, é preparado um conjunto de casos de teste, junto com o resultado esperado de cada caso. A ferramenta CASE roda os casos de teste, compara os resultados reais com os esperados e informa ao usuário cada um dos casos. Estes são armazenados para uso em testes de regressão toda vez que o produto é modificado. SilkTest é exemplo de uma ferramenta desse tipo. É importante ressaltar que uma ferramenta CASE é toda ferramenta que contribui nos processos de desenvolvimento de software e consequentemente na engenharia de software, tendo em vista que, com o auxílio do computador, traz facilidades para a equipe desenvolvedora.

Saiba mais

Conforme Schach (2010, p. 499),

> Quando o processo de integração estiver completo, o produto como um todo é testado. Isso é denominado teste de produto. Quando os desenvolvedores estiverem confiantes em relação à correção de cada aspecto do produto, ele é passado para o cliente para o teste de aceitação.

Teste de validação

O teste de validação começa no momento em que termina o teste de integração, quando os componentes individuais já foram exercitados, o software está completamente montado como um pacote e os erros de interface já foram descobertos e corrigidos. No nível de validação ou de sistema, a distinção entre diferentes categorias de software desaparece. O teste focaliza ações visíveis ao usuário e saídas do sistema reconhecíveis pelo usuário. A validação pode ser definida de

várias maneiras, mas uma definição simples (embora rigorosa) é que a validação tem sucesso quando o software funciona de uma maneira que pode ser razoavelmente esperada pelo cliente. Nesse ponto, um desenvolvedor de software veterano pode protestar: "Quem ou o que é o árbitro para decidir o que são expectativas razoáveis?". Se uma Especificação de Requisitos de Software foi desenvolvida, ela descreve todos os atributos do software visíveis ao usuário e contém uma seção denominada Critérios de Validação, a qual forma a base para uma abordagem de teste de validação (PRESSMAN; MAXIM, 2016, p. 483).

Teste de sistema

Um problema clássico do teste de sistema é a "procura do culpado". Isso ocorre quando é descoberto um erro e os desenvolvedores de diversos elementos do sistema começam a acusar um ao outro pelo problema. Em vez de adotar essa postura sem sentido, é necessário se antecipar aos problemas de interface em potencial e, conforme Pressman e Maxim (2016, p. 486):

1. criar caminhos de manipulação de erro que testem todas as informações vindas de outros elementos do sistema;
2. executar uma série de testes que simulem dados incorretos ou outros erros em potencial na interface de software;
3. registrar os resultados dos testes para usar como "evidência" se ocorrer a caça ao culpado;
4. participar do planejamento e projeto de testes do sistema para assegurar que o software seja testado adequadamente.

O Quadro 3 apresenta uma tabela contendo informações resumidas e complementares sobre o universo tão vasto do teste de software.

Quadro 3. Classificação dos testes quanto ao papel nos processos

Termo	Versão em inglês	Definição
Teste de aceitação	*Acceptance testing*	Teste formal realizado para determinar se um sistema satisfaz a seus critérios de aceitação e capacitar um usuário, cliente ou outra entidade autorizada a determinar se aceita ou não o sistema.

(Continua)

(Continuação)

Quadro 3. Classificação dos testes quanto ao papel nos processos

Termo	Versão em inglês	Definição
Teste de desenvolvimento	*Development testing*	Teste formal ou informal realizado durante o desenvolvimento de um sistema ou componente, geralmente pelo desenvolvedor.
Teste de integração	*Integration testing*	Teste no qual componentes são combinados e avaliados para testar a interação entre eles.
Teste de qualificação	*Qualification testing*	Teste realizado para determinar se um sistema ou componente é adequado para uso operacional.
Teste de regresso	*Regression testing*	Teste seletivamente repetido de um sistema ou componente para verificar se alterações não causaram efeitos indesejáveis e se o sistema ou componente mantém a conformidade com seus requisitos especificados.
Teste de sistema	*System testing*	Teste conduzido em um sistema completo integrado, para avaliar sua conformidade com os requisitos especificados.
Teste de unidade	*Unit testing*	Teste individual de unidades ou grupos de unidades.
Teste funcional	*Functional testing*	Teste realizado para avaliar a conformidade de um sistema ou componente com os requisitos funcionais especificados.
Teste operacional	*Operational testing*	Teste realizado para avaliar um sistema ou componente em seu ambiente operacional.

Fonte: Adaptado de Paula Filho (2009, p. 352).