Testes de Software

Profº Rafael Maltempe

INFORMAÇÕES SOBRE O AUTOR

Rafael Maltempe da Vanso

- Pós-Graduado em Docência no Ensino Superior: Tecnologias Educacionais e Inovação pela Unicesumar - PR.
- Pós-Graduado em Engenharia de Sistemas pela Escola Aberta do Brasil - ESAB - ES.
- Graduado em Ciência da Computação pela Fundação Faculdade de Filosofia, Ciências e Letras de Mandaguari - FAFIMAN - PR.

Sobre o Autor

Rafael Maltempe da Vanso é bacharel em Ciência da Computação pela Faculdade de Filosofia, Ciências e Letras de Mandaguari - FAFIMAN (2007), tem Especialização em Engenharia de Sistemas pela Escola Aberta do Brasil - ESAB (2010), formação complementar em Tecnologia em Telecomunicações e Gerência de Projetos, pela Escola Aberta do Brasil - ESAB (2010).

Foi professor formador na disciplina de Redes de Computadores no curso de Gestão em Tecnologia da Informação na UniCesumar - Maringá-PR (2016). Atualmente, é Professor Mediador na UniCesumar - Maringá-PR e Professor colaborador na UNIFAMMA - Maringá-PR.

Tem livros publicados de Fundamentos de Redes de Computadores, Tópicos Especiais e Banco de Dados Orientados a Objetos, pela UniCesumar; Administração e Segurança de Banco de Dados, pelo Grupo Ânima Educacional; Recursos Tecnológicos, Banco de Dados e Redes de Computadores e Segurança, pela UNIPAR - PR

INTRODUÇÃO DO LIVRO

Olá, caro(a) aluno(a)! Neste livro, iremos trabalhar conceitos relacionados ao teste de software, uma das principais áreas que vem crescendo no mercado de trabalho de construção de softwares.

A princípio, falaremos sobre os conceitos básicos de teste de software, pontuando algumas técnicas e critérios, características, fundamentos e a importância dos testes para o desenvolvimento de software. Falaremos também sobre o porque é necessário testar, mostrando alguns processos, planejamentos e verificar a monitoração e controle dos resultados de testes de software.

Mais adiante, iremos explanar um pouco mais sobre os testes de software, falando sobre testes unitários, de integração, de validação, de sistema e de aceitação. Claro, que dentro destes testes, iremos falar sobre algumas técnicas que auxiliam a realizar, da melhor forma, os testes.

Falaremos também sobre as técnicas de testes funcionais e estruturais, explanando também técnicas de teste de regressão, carga, estresse, usabilidade e segurança. Sempre pontuando cada característica para um melhor entendimento.

Na última unidade, iremos falar sobre tipos de frameworks de teste, e iremos explanar algumas práticas importantes, inspeções, sobre a automatização de testes e diversas outras características.

Ao final da leitura de nosso livro, você aluno(a) poderá ter o entendimento do que é um teste de software e o porque o mesmo teve o seu “boom” nos últimos tempos. A importância, de forma geral, poderá resultar numa melhor qualidade do software. E claro, também o entendimento da importância do papel de usuário, que faz parte da equipe de desenvolvimento e que realize o papel de testador, e por fim, também, a importância do usuário final para a realização de testes.

Vamos iniciar nossos estudos e adquirir esses conhecimentos importantes?

UNIDADE I

Fundamentos de Teste de Software

Rafael Maltempe da Vanso

Introdução

Entender o quanto é importante a etapa de teste é fundamental. Nesse sentido, nesta unidade apresentaremos os conceitos básicos relacionados aos testes de software. Iremos descrever as principais fases, apresentar técnicas, critérios, características e limitações direcionados ao teste de sistemas.

No decorrer da unidade, iremos descrever fundamentos essenciais relacionados aos testes, explicando a necessidade de serem realizados e definindo conceitos de teste de software.

Sobre processos fundamentais, iremos entender a importância do planejamento, desenho de testes, execução, monitoramento e controle e avaliação dos resultados. Encerrando a unidade, conceituaremos, ainda, as definições de verificação e validação em testes de software.

Fonte: Bestdesign36 / 123RF.

1. A importância de testes para o desenvolvimento de software

Em se tratando de desenvolvimento de software, já temos em mente conceitos e características importantes de como desenvolver software adequados a determinados problemas. Dentro desses conceitos e características podemos citar a análise e obtenção dos requisitos e diagramas, essenciais e cuja boa análise é extremamente importante. Porém, não são somente esses conceitos e características que são importantes. As metodologias existentes para o desenvolvimento de software não dão a total garantia de uma ótima qualidade para o software

Dessa forma, temos também o quesito teste de software. Mas qual a finalidade de se realizar um teste de software em um software desenvolvido? Para responder a essa pergunta, podemos realizar uma analogia a diversos produtos que são criados pela indústria, os quais são analisados e testados no decorrer da fabricação para que, quando disponibilizados ao usuário final, não tragam problemas.

Por essa razão, para a obtenção de uma boa qualidade do software produzido, temos como etapa fundamental o procedimento de testes em diversas vertentes do código fonte, já que essa é uma das últimas etapas do projeto de codificação (PRESSMAN; MAXIM, 2016).

Esses testes, realizados de uma forma criteriosa, têm assumido uma grande importância, pois, além de evitarem problemas na realização de suas funções, vêm cada vez mais impactando em custo, podendo chegar de 3 a 5 vezes o valor gasto em relação a outras atividades do desenvolvimento (SOMMERVILLE, 2011).

Imagine você se, em determinados tipos de software, como sistemas de controle de vôo, reatores nucleares e outros (que são sistemas críticos dos quais vidas humanas dependem), as atividades de testes de software não fossem realizadas adequadamente. Sabemos que isso poderia causar diversos problemas, não é?

Por esse motivo, todo e qualquer tipo de software deve passar por tipos de testes que devem ser realizados adequadamente, para que, assim, o desenvolvimento do software possa ser concluído sem problemas maiores, dependendo da sua finalidade.

Mas, se os testes são essenciais para a finalização dos software, por que, em muitas vezes, não são realizados adequadamente? A resposta para essa pergunta não é tão difícil. Sabemos que cada vez mais os prazos são tratados como importantes na fabricação de softwares, e de fato são. Porém, esses prazos estão interferindo na entrega de um bom produto. Muitas vezes, a pressão que se instala na entrega de um software interfere na contemplação da realização dos testes de forma correta e adequada.

Devemos entender que o desenvolvimento de um software está sujeito a diversas influências. Sendo assim, os testes tornam-se fundamentais. Outro ponto fundamental é que softwares são desenvolvidos por humanos que, querendo ou não, podem cometer erros de programação. Erros assim sempre estarão presentes, e os testes devem ser realizados de forma sistemática.

No final das contas, o grande desafio das empresas e, consequentemente, dos desenvolvedores é produzir qualquer software com qualidade. E mais: agregar a qualidade a um baixo custo e realizar o desenvolvimento em curto espaço de tempo. Porém, isso ainda está um pouco longe de acontecer.

Para concluir nosso entendimento sobre a importância da realização dos testes, vimos que a finalidade é agregar qualidade ao software final que será entregue. Essa qualidade ainda pode ser medida de acordo com os possíveis defeitos encontrados. Um software com boa qualidade é, no final, um software mais confiável, que passou por diversos testes e que teve suas falhas corrigidas.

1.1 Conceitos básicos

Falamos até agora sobre a importância dos testes de software, mas o que são eles? Os testes foram criados para revelarem erros que foram cometidos ao se construir ou planejar um projeto de software.

Mas como tratar desses erros? Para começarmos a entender, devemos aplicar estratégias de teste de software, que devem ser desenvolvidas por gerentes de projeto, engenheiros ou especialistas em testes.

O teste é um trabalho moroso e, muitas, vezes ocasiona mais trabalho do que a codificação do software. Devemos iniciar os testes por pequenos componentes, aplicando-os para descobrir erros em relação aos dados e à lógica programada. Conforme vão sendo realizados, passa-se a testar componentes maiores. À medida que os erros são descobertos, os mesmos devem ser corrigidos (PRESSMAN; MAXIM, 2016).

Devemos entender que testar um software não é somente executá-lo e começar a realizar operações de passos a fim de encontrar determinados erros. Há toda uma programação por trás dessa etapa, como planejar as escolhas de condições de testes a serem executados, modelar os casos de testes, checar todos os resultados encontrados, gerar relatórios, avaliar os testes realizados etc (RIOS; MOREIRA, 2013).

Dessa forma, o processo de teste de software visa averiguar se o programa desenvolvido realiza as atribuições a que foi proposto e se realiza de forma confiável. Sendo assim, ao inserirmos uma entrada de dados no programa, sua saída deve ser conforme esperado no momento de sua codificação.

Outro ponto importante é que devemos nos ater a alguns conceitos fundamentais que nos ajudarão no processo tanto de planejamento como de execução de testes. Para começar, podemos entender como erro algum procedimento incorreto cometido por um programador. Um defeito é a consequência de um erro, uma falha é um efeito de um ou mais defeitos e um *bug* é um erro na lógica de programação (MOLINARI, 2013).

Após entendermos conceitos e fundamentos dos testes de software, podemos citar alguns princípios desses testes, como a Lei de Pareto, que indica que 80% dos erros normalmente podem ser localizados em 20% do projeto e, comumente, em módulos principais do projeto (PRESSMAN; MAXIM, 2016).

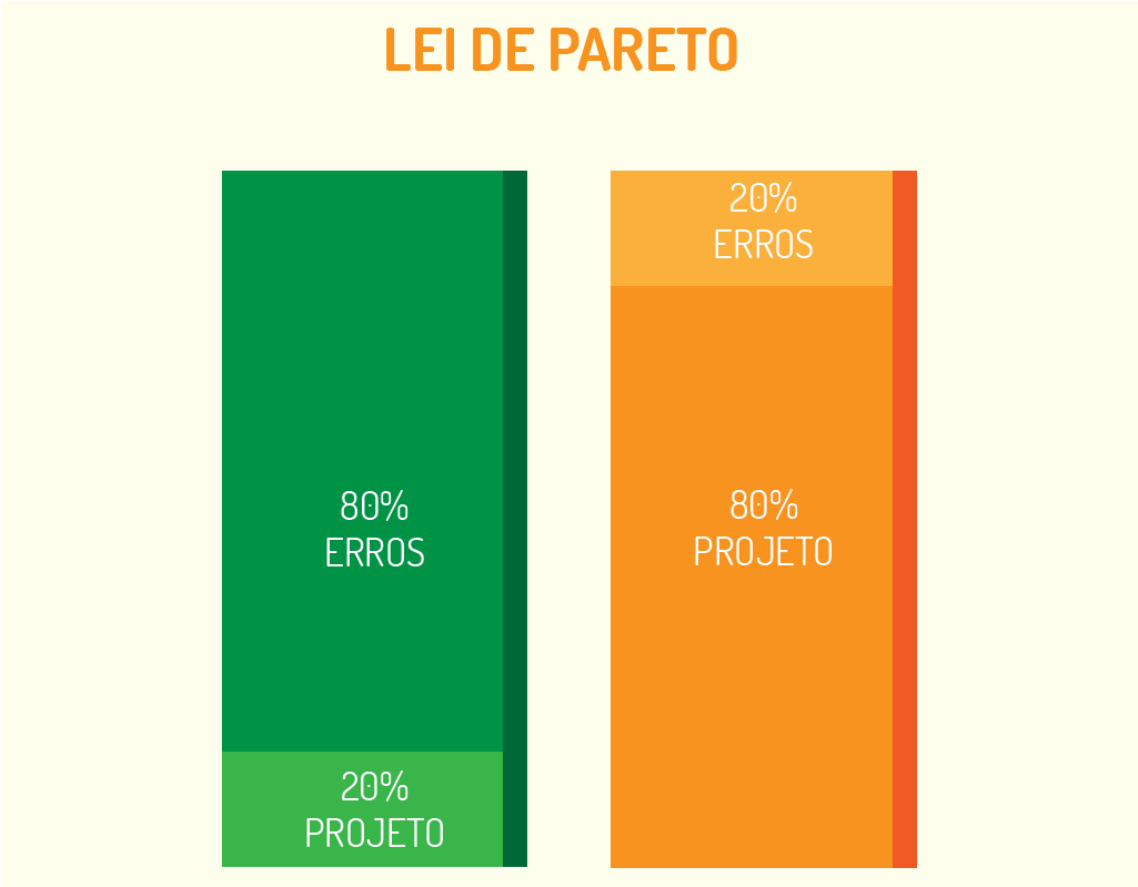


Figura 1.1 - Princípio de Pareto

Fonte: Araya Pacharabandit / 123RF.

Outro princípio importante é que bons casos de testes são aqueles que encontram erros que ainda não foram descobertos. Esses casos são projetados sempre analisando o que foi colocado como requisito do projeto. Assim, concluímos a importância de um bom levantamento de requisito.

FIQUE POR DENTRO

Não é novidade a qualidade relacionada a testes de software. Todos os anos, podemos verificar que as indústrias cada vez mais vão dando uma atenção diferenciada a essa atividade. E o futuro está aí! Sabemos que diversas tecnologias estão surgindo, como a Inteligência Artificial e a tão esperada Internet das Coisas. Nesse sentido, o que esperar do futuro do teste de software? Podemos verificar um pouco mais sobre esse assunto no *link* a seguir: <http://www.tecnisys.com.br/noticias/2018/o-futuro-do-teste-de-software>.

1.2. Fases da atividade de teste

Vimos que o objetivo da atividade de teste é identificar/verificar a presença de erros, sejam lógicos ou não e, no final, realizar a sua correção. A partir disso, podemos começar a entender um pouco mais sobre esse processo importante, como verificar as fases presentes na atividade de teste de software.

A atividade de teste é dividida em fases, cada uma com objetivos distintos. Podemos iniciar falando sobre o teste de unidade, que é a fase em que as unidades individuais de um sistema são testadas individualmente. Cada uma das unidades é validada e verifica-se se estão funcionando conforme o planejado.

Outra fase é o teste de integração, que tem por objetivo testar cada unidade de forma combinada, ou seja, as unidades são testadas como um grupo, de forma integrada, expondo-se, assim, possíveis falhas.

Temos, também, a fase de testes de sistema, na qual o sistema é testado completamente. Todas as funcionalidades são integradas e a conformidade do sistema com os requisitos especificados é avaliada.

Por fim, temos a fase de testes de aceitação, onde são testadas todas as funcionalidades que o cliente requereu durante todo o projeto do software. Essa fase, claramente, deve ser realizada pelo usuário que utilizará o sistema final.

Devemos ter em mente que a execução dessas fases da atividade de teste nas etapas do desenvolvimento irá detectar possíveis falhas em tempo de desenvolvimento e, consequentemente, produzirá maior qualidade, diminuindo o custo do software.

Nos próximos tópicos iremos nos aprofundar um pouco mais nessas fases da atividade de testes.

1.3. Técnicas e critérios de teste

Há vários tipos de testes existentes para serem aplicados no desenvolvimento de software, o que pode tornar o processo demorado e exaustivo. Dessa forma, existem alguns critérios e técnicas para se executar os tipos de testes existentes.

O objetivo dessas técnicas e critérios é realizar testes de forma clara e objetiva, possibilitando, assim, determinar em quais situações o software irá falhar.

Figura 1.2 - Técnicas e critérios de testes

Fonte: Egor Kotenko / 123RF.

Podemos citar brevemente algumas técnicas, como a técnica funcional. Essa, também conhecida como caixa preta, baseia-se nos requisitos funcionais e se preocupa com a saída de dados gerada após a entrada das informações. Normalmente, essa técnica apresenta algumas dificuldades em testar todas as entradas de dados possíveis. No entanto, ela pode ser aplicada por todo o desenvolvimento do software (RIOS; MOREIRA, 2013).

Um critério importante dessa técnica é o particionamento em classes de equivalência, que agrupa e otimiza casos de testes a fim de realizar uma melhor cobertura do sistema. Seu funcionamento se dá propondo a separação de possíveis entradas de dados em categorias diferenciadas. Como critérios para as divisões das partições, temos que a divisão deve ser bem clara e definida. Assim, um elemento deve ser igual ao outro da mesma partição e não pode existir ora em uma partição e ora em outra. Se, quando particionada, a divisão não possuir membros, ela simplesmente não existe e todo valor inválido também deve ser levado em contato para a divisão (RIOS; MOREIRA, 2013).

Dentro da técnica funcional, temos diversos testes que auxiliam na prevenção de erros. Podemos citar o teste de requisitos, o teste de regressão, o teste de interconexão e outros, que serão abordados adiante com mais detalhes.

Outra técnica que podemos mencionar é a técnica estrutural, ou caixa branca, que realiza testes relacionados à lógica (estrutura) interna do software, verificando a precisão do funcionamento do sistema codificado. Essa técnica é muito utilizada em fase de teste de unidade e integração (MOLINARI, 2013).

Uma característica importante da técnica estrutural engloba critérios baseados em fluxo de dados que irão descobrir caminhos para se testar o código, selecionar as definições de testes e, por fim, o uso das variáveis relacionadas ao código em questão. Esse critério utiliza de conceitos de grafos para a localização de caminhos simples. A partir disso, o critério de fluxo de dados verifica o comportamento das variáveis, até, por algum motivo, encontrar determinados defeitos que passaram despercebidos (PRESSMAN; MAXIM, 2016).

Na técnica estrutural, estão presentes os seguintes testes: o teste de execução, teste de estresse, teste de recuperação, teste de segurança e outros. Alguns desses testes serão aprofundados mais adiante, no decorrer de nosso livro.

Outra técnica presente é a técnica baseada em erros, que visa verificar se o sistema está livre dos erros típicos que desenvolvedores cometem ao implementarem um sistema. Como critérios, temos a análise de mutantes e semeadura de erros. Na análise de mutantes, os programas são criados baseados no software que deverá ser testado. Seu objetivo é encontrar testes que revelam a existência de comportamentos diferentes entre o software original e o software mutado. Ao final, como resultado, temos os erros e os números de erros esperado. Já no critério de semeadura de erros, como o próprio nome diz, o programa é semeado (inserido) artificialmente com alguns erros. Após isso, erros encontrados são analisados e é identificado o percentual de prováveis, descartando-se ao final os erros artificiais (MOLINARI, 2013).

REFLITA

Procurar uma nova validação a cada linha do código produzido, identificar possíveis brechas e garantir o tratamento de erros encontrados. Mas como podemos realizar essas atividades? Pensar, pensar e pensar é fundamental, e a resposta para isso é ser extremamente detalhista.

1.4. Características e limitações

Uma das principais características de um teste de software é atribuir uma maior qualidade ao sistema desenvolvido pelos programadores. Mas o que seria um software com qualidade?

De modo geral, a qualidade de um software está muito ligada ao que um *stakeholder*/cliente acredita ser o melhor, ou seja, suas expectativas. Dessa forma, quanto mais um produto atenta para o que o cliente necessita, maior será sua satisfação.

Basicamente, a qualidade aqui relacionada como característica proposta pelo teste de software deseja minimizar as incertezas, erros, e, ainda, sistematizar os critérios de aceitação do software por parte do usuário.

Realizar testes em determinado software não significa dizer que o mesmo terá necessariamente uma boa qualidade. Para isso, precisamos aplicar determinadas características aos testes como, por exemplo, realizar um planejamento adequado de um ciclo para testes, composto pelo desenvolvimento, sua execução e, claro, a manutenção. Esse planejamento deve ser executado com antecedência e, obviamente, executado sistematicamente. Por essa razão, um template, ou seja, um modelo com um conjunto de etapas com as técnicas específicas para o teste do projeto e seus métodos de execução, é realizado e proposto (SOMMERVILLE, 2011).

Devemos entender que a qualidade deve estar relacionada desde a criação até a manutenção de um software, ou seja, o ciclo de vida de nosso sistema.

Os testes normalmente ajudam a responder determinados questionamentos, como os relacionados às expectativas do cliente. O software realmente funciona como o cliente quer? Devemos imaginar essa característica como se fôssemos o cliente e, para isso, é importante averiguar as especificações dos requisitos e até mesmo consultar novamente os *stakeholders*. Isso lhe trará, podemos dizer, um amadurecimento final de todo o projeto (PRESSMAN; MAXIM, 2016).

Outra característica importante (e podemos dizer que é sempre desejada por todos) é “pegar” o possível defeito antes do cliente. Sabemos que é difícil testar todas as possibilidades em um software, porém devemos sempre nos perguntar: o software funciona como o desenvolvedor espera? Muitas vezes o desenvolvedor acredita ter realizado adequadamente o que foi proposto, porém a prática pode não demonstra isso.

Detectar problemas de design da aplicação e manter o sistema estável também são características essenciais. Possuir uma estratégia prévia, ou seja, planejada, do que deve ser realizado, torna a verificação dos erros por meio dos testes bem mais estruturada, o que propicia a garantia do funcionamento adequado.

Mas é fácil manter e testar um software? É uma pergunta pertinente e que nos faz pensar muito sobre o que estamos realizando em nosso software. De uma forma geral, o teste ajuda os desenvolvedores a realizarem o certo e de forma correta. Mas como? É fácil ter essa resposta! Se você, como testador, realiza a refatoração de seu sistema, elimina erros e procura as melhores práticas para serem aplicadas, temos uma grande porcentagem de seu software ter uma boa manutenibilidade. Podemos concluir, assim, que o grau de manutenibilidade do seu software é reflexo de sua testabilidade (RIOS; MOREIRA, 2013).

Concluímos, dessa forma, que das características dos testes podemos obter qualidades em sistemas. Porém, nem só do bom vivem as definições. Esbarramos em diversas limitações quando nos deparamos com os testes.

Determinados testes são exaustivos, tornando-os impraticáveis. Imagine você testar todas as possibilidades e todos os elementos possíveis de entrada de dados no software. Isso geraria um custo alto e um tempo de trabalho que não se encaixa no disponível pelas empresas.

Os procedimentos de testes existentes não provam a corretude de um sistema, mas, mesmo não sendo possível, é por meio desses testes que provamos que determinado programa está correto ou sem a presença da maioria dos erros. Realizar essa provação contribui para, de uma forma ou de outra, aumentar a confiança de um software em realizar as funções para as quais foi programado.

Outra limitação que podemos citar diz respeito à incompletude dos testes, ou seja, pode ser que os testes aplicados não encontrem erros, o que é um problema na construção do software.

Muitas limitações também esbarram em conhecimentos específicos, o que pode ser adquirido somente por *experts* em determinados problemas. Provar determinadas propriedades pode ser um pouco mais difícil. Assim, alguns testes podem ser limitados a propriedades de verificação simples e programas simples (MOLINARI, 2013).

FIQUE POR DENTRO

A indústria de software cada vez mais vem investindo em testes de software e verificamos, com isso, mudanças no desenvolvimento. Assim, temos uma tendência clara na performance e segurança, o que gera uma determinada qualidade no sistema desenvolvido. Falando em tendência, podemos verificar no *link* a seguir algumas inclinações para o teste de software esperadas para este ano. Vamos ler? Disponível em: <https://www.olisipolearning.pt/farol/10-tendencias-testes-de-software/>.

ATIVIDADE

1) O teste de software vem crescendo imensamente nos últimos tempos e vem se tornando uma etapa fundamental até a entrega do software ao usuário final. Sendo assim, apresentamos duas técnicas relacionadas ao teste de software que auxiliam num resultado considerável. Sobre as definições anteriores, quais foram as técnicas abordadas?

   a) Técnica Tradicional e Técnica Funcional.

   b) Técnica Funcional e Técnica Estrutural.

   c) Técnica Organizacional e Técnica de Verificação.

   d) Técnica Estrutural e Técnica Organizacional.

e) Técnica Funcional e Técnica de Verificação.

2. Fundamentos de Teste

Neste tópico, iremos dar uma introdução ao teste de software, conceituando um pouco mais sobre esse assunto e indicando o porquê de ser necessário realizar testes em softwares computacionais.

2.1. Introdução ao teste de software

Alguns objetivos do teste de software foram introduzidos no tópico anterior, porém vale a pena relembrar. Eles podem ser expressos por meio de atividades de teste que descobrem erros, sendo que um bom caso de teste revela erros ainda não descobertos anteriormente. Dizemos que a etapa de teste de software deve encontrar o maior número de erros e com o menor tempo gasto possível, não havendo, assim, esforço desnecessário.

2.1.1 Por que é necessário testar?

Você sabia que tudo que consumimos é testado antes de chegar às prateleiras e, consequentemente, à sua casa? Pois bem, isso é um fato! Televisores, roupas e tudo que se possa imaginar é testado para que não haja problemas.

Assim, se tudo é testado, programas de computadores também são testados. Porém, softwares não são palpáveis, ou seja, não conseguimos pegar para analisar e verificar possíveis problemas. Dessa forma, como são realizados esses testes? Isso iremos aprender no decorrer de nossa disciplina.

Mas antes, vamos entender a razão de testarmos nossos softwares. Sabemos que, atualmente, falhas podem causar sérios prejuízos a empresas, uma vez que, se softwares possuem brechas de segurança ou outros tipos de erros, podem causar desde a perda de dados até problemas financeiros.

Como dito, hoje existem softwares presentes em praticamente tudo que conhecemos. Agora, imagine você: o que ocorreria se um software destinado a ser controlador de voo desse pane devido a erros de execução? Isso seria um caos, não acha? Aviões começariam a pousar e decolar ao mesmo tempo e, consequentemente, teríamos sérios problemas de segurança.

Com dito antes, independe de sua qualificação, humanos podem cometer erros em especificações, codificação e outros. Um erro acontece por diversos motivos, como falhas de comunicação, complexidades, pressão por prazos etc. Os testes irão trazer uma confiabilidade maior, pois sabe-se que passaram por todo um processo de verificação (MOLINARI, 2013).

Essa confiabilidade indica que um software não causará determinadas falhas por um tempo, já que um software pode conter defeitos não previstos ou encontrados e mesmo assim ser confiável.

Concluímos que um software sempre terá defeitos, pois há partes de um software que são menos testadas que outras, e problemas podem passar despercebidos, muitas vezes, pelo fato do prazo de entrega. A confiabilidade de um software tende a aumentar de acordo com o tempo, dependendo de como são realizados os testes.

Testes também precisam ter níveis de satisfação ou, como podemos dizer, níveis de independência. Um nível mais baixo diz respeito a testes realizados pelo próprio programador, o que não é muito aconselhável. Um nível médio diz respeito a testes realizados por programadores que fazem parte da equipe, porém diferente daquela que fez a codificação. Já no nível alto, os testes são realizados por uma pessoa que é independente da equipe de programação. E, por fim, um nível ainda mais alto é aquele no qual testes são realizados por ferramentas de testes. Este último é mais difícil de ser alcançado. Dessa forma, o aconselhável é realizar os testes por meio de pessoas independentes da equipe de programação (PRESSMAN; MAXIM, 2016).

Outro ponto importante relacionado ao teste de software é o que já comentamos anteriormente sobre teste e qualidade. De forma geral, a qualidade de um software é o grau ao qual determinado software atende sobre o que foi proposto às necessidades do usuário, como segurança, acessibilidade, custo pequeno e outros.

Mas será que testes em software podem aumentar a sua qualidade? Como resposta, obtemos um sonoro não. O teste, na verdade, ajuda a medir em qual nível está a qualidade do sistema e, consequentemente, a reduzir o risco da existência de problemas, aumentando, dessa forma, a sua confiabilidade.

Concluímos, então, após tudo que vimos, que a execução de testes é essencial para o sistema desenvolvido. No entanto, não é uma atividade que qualquer pessoa possa realizar. Concluímos, também, que testar não garante que não haja mais erros no sistema desenvolvido.

2.1.2 O que é teste de software

Nos tópicos anteriores, debatemos sobre assuntos relacionados a testes. São diversos conceitos essenciais para o entendimento de como realizar um teste em um sistema. Devemos compreender que o teste é composto por algumas atividades: planejamento e controle; análise e modelagem; implementação e execução; avaliação de critérios relacionados à saída de dados e relatórios; e, por fim, atividades relacionadas ao encerramento dos testes realizados (PRESSMAN; MAXIM, 2016).

De forma geral, os testes podem ser definidos como testes dinâmicos ou estáticos. No teste dinâmico, o sistema precisa ser executado para sua verificação, sendo composto pelas atividades de planejamento de testes, execução e controle de testes. Já o teste estático é composto pelas atividades de revisão, inspeção e análise estática do código (SOMMERVILLE, 2011).

Com o teste, de uma forma ou de outra, podemos verificar se um sistema executa o que foi programado e, para isso, aplicamos testes com dados fictícios, para que, assim, possam ser verificados possíveis erros.

Quando falamos sobre testes de software, podemos dizer que eles demonstram, tanto ao desenvolvedor quanto ao cliente, que o sistema em questão atende ao que foi proposto e, também, encontram situações que são equivocadas, como panes e processamentos dos dados de forma incorreta (MOLINARI, 2013).

Mas devemos entender, também, que os testes não demonstram se o sistema está livre de problemas, ou ainda se ele executará o especificado, dependendo de uma determinada situação aplicada.

O processo de teste normalmente realiza um mistura de testes manuais e automatizados. No manual, um testador irá executar o sistema com diversos tipos de dados e verifica se os resultados estão de acordo com sua expectativa. Já no teste automático, os tipos de testes são adicionados em um programa que, quando executado, realiza os testes no sistema (RIOS; MOREIRA, 2013).

Porém, apesar de seu aumento nos últimos anos, os testes automáticos não podem estar totalmente presentes em um sistema, pois o mesmo só realiza o que foi proposto no software de teste.

Podemos concluir que o teste de software faz parte de todo o processo de desenvolvimento do sistema e, conforme visto anteriormente, o principal objetivo é revelar possíveis erros para que seja possível a sua correção até atingir uma boa qualidade para a entrega deste produto ao usuário final.

Hoje, podemos entender que o profissional que trabalha com a realização dos testes é um dos mais importantes profissionais, possuindo, normalmente, um vasto ramo de atividades, como avaliação da especificação de requisitos e projeto técnico, verificação de documentos, testes de performance e capacidade, avaliação de interface, avaliação de módulos e diversas outras que são fundamentais (MOLINARI, 2013).

É importante lembrar que a realização de testes deve ser feita desde as fases iniciais da programação do software, o que pode gerar uma economia em seu custo final. Uma ideia é que os testes já sejam projetados desde o planejamento inicial do sistema. Dessa forma, para todas as suas funcionalidades, conforme vão sendo planejadas, vão sendo projetados possíveis testes que irão auxiliar na verificação de erros.

Temos diversos testes e técnicas que auxiliam e que podem ser feitos no decorrer da programação do software. Podemos verificar, no Quadro 1.1, um breve resumo de alguns tipos de testes existentes.

Tipo de Teste	Descrição
Teste de Unidade	Realiza o teste em módulos do código-fonte, ou seja, em cada componente ou classe.
Teste de Integração	Realiza o teste combinando/integrando os módulos do código-fonte a fim de verificar como é o funcionamento em conjunto desses testes.
Teste Operacional	Realiza o teste que verifica se o software pode trabalhar muito tempo sem cometer falhas.
Teste de Regressão	Realiza testes na aplicação toda vez que for alterado algo no software.
Teste de Caixa-preta	Realiza testes em todas as entradas e saídas desejadas no software. Neste caso, cada saída indesejada que é verificada no software é vista como um erro.
Teste Caixa-branca	Realiza o teste no código-fonte, já que podem existir trechos que nunca foram testados.
Teste Funcional	Realiza testes nas funcionalidades, requerimentos, regras de negócio, a fim de validar as funcionalidades descritas na documentação.
Teste de Interface	Realiza testes a fim de verificar se a navegabilidade e os objetivos da tela estão em correta funcionalidade.

Teste de Performance	Realiza testes em relação ao tempo de resposta e se o mesmo é o desejado para a aplicação.
Teste de Aceitação do Usuário	Neste tipo de teste, será verificado se a solução será bem vista pelo usuário, ou seja, se o software será bem recebido.
Teste de Estresse	Realiza teste na aplicação sem situações inesperadas.
Teste de Configuração	Este tipo de teste verifica se a aplicação funciona em diferentes ambientes e diferentes configurações.
Teste de Segurança	Realiza testes em relação à segurança da aplicação, a partir de permissões e perfis.

Quadro 1.1 - Tipos de Testes de Software

Fonte: Adaptado de Sommerville (2011).

Com o exposto anteriormente, começamos a entender um pouco mais sobre os motivos e importância de se realizar testes em softwares, e ter o entendimento é essencial para o prosseguimento de nossos estudos.

FIQUE POR DENTRO

Agora que começamos a entender um pouco mais sobre os testes de software, podemos nos perguntar: por que escolher a área de teste de software como carreira? Não precisamos de muito para responder a essa pergunta. Vimos em nosso texto que é algo crescente mundialmente. Para melhor entendermos, separamos o *link* a seguir, que demonstra um pouco mais sobre a área de testes de software. Vamos ver? Disponível em: <https://blog.db1.com.br/area-de-teste-de-software/>.

ATIVIDADES

2) O teste de software faz parte de todo o processo de desenvolvimento e tem como objetivo revelar erros para possível correção. Ao realizar testes em software, o testador trabalha com algumas atividades. Sobre essas atividades, a alternativa que indica corretamente duas delas é:

a) Avaliação da equipe e avaliação de módulos.

b) Avaliação da especificação de requisitos e avaliação de interface.

c) Testes de performance e capacidade e avaliação de programadores.

d) Avaliação de projeto técnico e avaliação de irregularidades.

e) Avaliação de interface e teste dimensional.

3. Processo Fundamental de Teste

Ao tratarmos sobre os testes, entendemos que, para que os mesmos funcionem de forma correta, devem ser tratados como um processo fundamental.

Esse processo fundamental é composto por algumas características, como planejamento, desenho dos testes, execução, monitoramento e controle e, por fim, avaliação dos resultados.

A partir de agora, iremos entrar nas definições dessas características, como poderemos verificar adiante.

3.1. Planejamento

É uma atividade essencial no desenvolvimento do software. O planejamento auxilia o programador ou testador no conhecimento de seus objetivos. Perceber qual meta você deseja alcançar é de suma importância.

O planejamento na fase dos testes não é muito diferente de qualquer outra fase de planejamento em todos os projetos de software. Nesse caso, garantir o sucesso e a qualidade do sistema é essencial nesta fase, que engloba alguns documentos como plano de teste, caso de teste e roteiro de teste.

Podemos dizer que o plano de teste é a base de qualquer teste de software e é um dos documentos que auxilia na aplicação em um projeto. Aqui, são apresentadas atividades relacionadas a teste, critérios de validação e outros (MOLINARI, 2013).

Um plano de teste deve realizar a integração entre as atividades existentes e deve ser o mecanismo que realiza a comunicação com as partes interessadas no sistema. Nesse sentido, pode conter definições a respeito do escopo e objetivos, o conjunto de requisitos a serem testados, os tipos de testes que devem ser realizados e possíveis ferramentas que podem ser utilizadas, cronogramas de atividades e possíveis recursos que podem ser utilizados nos testes (RIOS; MOREIRA, 2013).

Dessa forma, percebemos que o planejamento pode antecipar problemas que podem ocorrer com o nosso software a ser desenvolvido. Assim, podemos listar alguns itens que precisam ser considerados em um plano de teste, trazidos a seguir (MOLINARI, 2013):

- Deve-se realizar a descrição do projeto, uma identificação do que deverá ser realizado, e informar como se deve evoluir o plano de testes.
- Deve ser descrito o conjunto de requisitos a serem testados, como desempenho, segurança, funcionalidades e outros.
- Deve-se apresentar os tipos de testes a serem utilizados, bem como as técnicas e critérios para a realização dos testes selecionados. Além, claro, de possíveis ferramentas a serem utilizadas.

- Devemos, também, atribuir nesse plano a descrição da equipe e infraestrutura. Equipamentos, software de apoio e tudo que for utilizado para a elaboração do software deve ser exposto para garantir a estrutura adequada.
- Deve haver a descrição de todas as atividades, como se fosse uma espécie de cronograma do que se deve realizar no decorrer do projeto.
- E, por fim, apresentar uma documentação complementar, como manuais e outros, pertinentes ao projeto final.

Um documento pertinente ao planejamento é o caso de testes, que descreve determinadas condições a serem testadas, como se fosse uma criação de um caso de testes que irá encontrar falhas e problemas não tratados como requisitos. Outro item importante refere-se os roteiros de testes, que descrevem como realizar a execução dos casos de testes citados anteriormente (RIOS; MOREIRA, 2013).

Ao final, o processo de teste torna-se mais produtivo.

3.2. Desenho dos Testes

O desenho de casos de testes é utilizado para testar o sistema, sempre criando testes eficazes na validação e na identificação de defeitos. Entre as técnicas de desenho de defeitos, podemos citar os testes de caixa-preta, caixa-branca, dedução de erros, exploratório e outros, como vimos no Quadro 1.1.

Os testes devem ter a responsabilidade de conseguir indicar aos nossos clientes o que foi e o que não foi testado. Assim, quando o software desenvolvido funciona ou falha, dizemos que ele se comporta corretamente ou não para alguns valores inseridos, e isso é designado por divisão de equivalência.

A divisão de equivalência demonstra alguns métodos que auxiliam na compreensão do porquê de o sistema agir corretamente ou não para alguns valores. Essa ideia de equivalência nos mostra que o programa irá lidar com valores que são da mesma forma, ou seja, equivalentes. Assim, deveremos realizar um conjunto de valores possíveis e outro de valores equivalentes (MOLINARI, 2013).

Para essa técnica podemos testar a lógica, tanto para valores verdadeiros ou falsos como para entradas conhecidas ou desconhecidas, dados corretos ou incorretos e diversos outros que irão auxiliar nessa fase.

Outro método que podemos citar é o que realiza testes com valores corretos ou incorretos. Por exemplo, para testes com entradas numéricas, comece testando os valores típicos, costumeiros de serem digitados, e para valores incorretos, teste incluindo números negativos, zero, ou ainda símbolos e letras, que não são valores usuais (RIOS; MOREIRA, 2013).

Ainda temos o método de teste de combinações lógicas que, como o próprio nome diz, irá testar as combinações lógicas dos sistemas, visando encontrar possíveis buracos na lógica programada. Porém, são combinações perigosas, dependendo do tamanho dos problemas propostos no sistema.

3.3. Execução

Nesta etapa iremos implementar e executar nossos testes. Para isso, serão criados roteiros de testes. Nesses roteiros, criaremos dados de testes e, de acordo com esses dados, executaremos os casos de testes até serem finalizados.

Como retorno, analisaremos os dados e verificaremos possíveis atualizações, realizando a rastreabilidade entre a base de testes e os casos de testes, comparando os resultados obtidos com os esperados para que o sistema seja entregue de forma adequada (SOMMERVILLE, 2011).

3.4. Monitoração e Controle

A monitoração e o controle dizem respeito a monitorar situações que podem ocorrer relacionadas a defeitos. Dessa forma, retornam indicadores da situação de nosso projeto, como situações de defeitos em nível de reincidência, quais tipos de defeitos apareceram, níveis de rejeição e outros.

A monitoração, de uma forma ou de outra, visa garantir que o nosso teste no software ocorra conforme planejamos lá no início do projeto. Caso ocorram alguns problemas, os gerentes de testes podem aplicar determinadas características que coloquem as atividades de teste “na linha” novamente, para que retornem para as características desejadas (MOLINARI, 2013).

Existem alguns indicadores que auxiliam no monitoramento e controle dos processos, como: a verificação de números de defeitos encontrados no desenvolvimento; número de defeitos encontrados na atividade de teste; em relação aos requisitos, a verificação do número de casos de teste por cada requisito; monitoramento do tempo previsto do projeto, como horas gastas para a realização e conclusão do projeto; monitoramento dos defeitos por casos de testes; e possíveis reincidências de defeitos pelos testes realizados (SOMMERVILLE, 2011).

3.5. Avaliação dos Resultados

Sabemos que, ao final do desenvolvimento de um sistema, programadores, empresas e até clientes querem um software com qualidade. Vimos que, para que isso aconteça, as atividades de testes são fundamentais na construção de um sistema mais sólido, assim, com maior confiabilidade e qualidade.

A avaliação dos resultados tem por objetivo verificar o desempenho dos processos de teste. Para isso, deve-se compreender o problema e o entendimento desse processo, estabelecer métricas, coleta e análise dos dados, mapeamento de processos etc.

A compreensão do problema é fundamental, pois corresponde aos estudos do processo de testes que devem ser analisados, como a identificação de funções para o teste, identificação de artefatos e outros, para, assim, ser possível a verificação de pontos de melhoria no sistema referente aos processos que foram adotados.

Em relação às métricas, elas são estabelecidas para a verificação dos resultados obtidos. São definidas métricas de interesse, como por exemplo a verificação da satisfação em relação aos métodos utilizados.

Outro tipo é a análise e coleta de dados, que compreende a definição e seleção de informações a serem analisadas. Essa fase corresponde à verificação de possíveis inconsistências nos dados e aplicação de melhorias a partir desses dados.

Assim, podemos verificar a importância da avaliação dos resultados, e que a partir de dados coletados observa-se possíveis melhorias no sistemas.

FIQUE POR DENTRO

Você já ouviu falar em metodologias ágeis? Se ainda não, é importante que leia um pouco mais sobre esse assunto. Adicionar metodologias ágeis ao andamento do teste pode melhorar o processo fundamentalmente. Dessa forma, separamos pra você um material sobre testes ágeis que lhe ajudará a entender um pouco mais sobre o assunto, que vem crescendo de forma a melhorar os processos existentes. Vamos ver um pouco mais sobre isso? Acesse: <https://imasters.com.br/agile/agile-testing-dicas-para-testes-de-software-em-times-ageis>.

ATIVIDADES

3) Testes de software possuem determinados processos que são fundamentais para que um software possa ser entregue conforme o solicitado pelas empresas/usuários. Esses processos possuem algumas características que podemos encontrar em qual das alternativas a seguir?

a) Estratégias, Execução de Testes Automáticos, Controle e Avaliação dos Resultados.

b) Planejamento, Desenho dos Testes, Execução, Monitoração e Controle e Avaliação dos Resultados.

c) Execução de Testes Simples, Monitoração dos Resultados e Controle dos Erros Encontrados.

d) Execução de Análise de Requisitos, Planejamento Estrutural e Controle e Avaliação dos Resultados.

e) Desenho dos Testes, Controle e Monitoramento de Usuários, e Avaliação dos Erros Encontrados.

4. Conceitos Fundamentais

Dentro de testes de software existem alguns conceitos fundamentais que devem ser compreendidos e que asseguram que o software a ser desenvolvido seja construído de forma adequada. Sendo assim, os conceitos são de validação e verificação, os quais iremos compreender um pouco mais a seguir.

4.1. Verificação e Validação

O teste de software contempla conhecimentos essenciais sobre verificação e validação (V&V). A verificação refere-se, simplesmente, ao conjunto de tarefas que irão implementar corretamente uma determinada função do sistemas. Já a validação engloba um conjunto de tarefas no qual verifica-se se o software que foi criado atende às especificações desejadas pelo cliente.

Temos outras definições relacionadas. Por exemplo, na verificação pergunta-se: “Fizemos o software corretamente?”; e na validação: “Estamos criando o produto certo?” (PRESSMAN; MAXIM, 2016).

As atividades de verificação se resumem em avaliar se o que foi planejado foi, devidamente, realizado, incluindo os requisitos, funcionalidades, documentação etc. As atividades da validação avaliam o que foi entregue. Assim, analisa-se se os requisitos atendem ao que foi proposto. Ao final, o cliente realiza sua validação.

A verificação e a validação incluem atividades relacionada à garantia de qualidade de software, ou *software quality assurance* (SQA), a exemplo das revisões técnicas, monitoramento de desempenho, simulações, estudo de viabilidade, análise de algoritmo, teste de usabilidade e alguns outros (PRESSMAN; MAXIM, 2016).

Os testes têm um papel muito importante na verificação e validação, pois os erros podem ser descobertos. Porém, os testes não podem ser tomados como aquilo que irá gerar a segurança. A qualidade gerada pelos testes é incorporada ao software por meio de processos de engenharia de software, e as aplicações corretas desses métodos conduzem a uma qualidade (SOMMERVILLE, 2011).

Pressman e Maxim (2016), em seu livro “Engenharia de Software: Uma Abordagem Profissional”, citam Miller (1977), relacionando o teste de software com a garantia de qualidade, dizendo que:

> [...] a motivação que está por trás do teste de programas é a confirmação da qualidade de software com métodos que podem ser economicamente e efetivamente aplicados a todos os sistemas, de grande e pequena escala (MILLER, 1977, p. 1-3, apud PRESSMAN; MAXIM, 2016, p. 468).

Temos como padrão de verificação o padrão IEEE 1012, e seus processos são utilizados para ver se produtos de desenvolvimento de uma determinada atividade estão em conformidade e se o produto satisfaz ao que foi proposto, conforme a necessidade do usuário.

Os processos da Verificação e Validação possuem atividades como análise, avaliação, revisão, inspeção e teste de produtos, que têm como objetivos estabelecer uma estrutura de processos, atividades e tarefas e definir um plano de verificação e validação (PRESSMAN; MAXIM, 2016).

Para uma melhor compreensão de verificação e validação, podemos exemplificar com tampas de garrafas. Cada garrafa fabricada por um empresa tem um tamanho específico, dependendo do tipo de garrafa. Uma empresa, por exemplo, com o molde de sua garrafa, produz a tampa que deve encaixar e fechar a mesma. Porém, o protótipo dessa tampa não garante que as tampas irão fechar as garrafas fabricadas. Se o fabricante tentar fechar suas garrafas com as tampas produzidas, significa que o mesmo está validando suas tampas, podendo, assim, verificar se elas funcionam e atendem às suas necessidades.

FIQUE POR DENTRO

Testar um software antes que ele seja entregue ao cliente é fundamental, já que queremos entregar nosso software sem problemas ou falhas. Essa fase é crucial para verificar se o que foi feito cumpre com o que foi solicitado. E, claro, a verificação e a validação auxiliam nessa etapa. Para que você não fique com dúvidas quanto às definições e diferenças entre essas atividades, separamos um *link* para leitura. Acesse: <http://logicalminds.com.br/saiba-a-diferenca-entre-teste-validacao-e-verificacao-de-software/>.

ATIVIDADES

4) O teste de software possui alguns conceitos fundamentais que devem estar compreendidos pelos testadores de software. Sobre os conceitos fundamentais, a alternativa que apresenta corretamente o conceito de Verificação é:

   a) A verificação refere-se a um conjunto de tarefas que verificam se o software que foi criado atende às especificações que o cliente deseja.

   b) A verificação refere-se, simplesmente, ao conjunto de tarefas que irão implementar corretamente uma determinada função do sistemas.

   c) A verificação é utilizada para testar o sistema, sempre criando testes eficazes na validação e na identificação de defeitos.

   d) É na verificação que iremos implementar e executar nossos testes. Para isso, serão criados roteiros de testes.

   e) A verificação auxilia o programador ou testador a conhecer seus objetivos.

INDICAÇÕES DE LEITURA

Nome do livro: Engenharia de Software

Editora: Pearson

Autor: Ian Sommerville

ISBN: 978-8579361081

Comentário: O livro “Engenharia de Software”, de Ian Sommerville, é um dos livros que devemos colocar na cabeceira da cama. É um dos principais livros sobre o assunto e tem o intuito de atender às necessidades de alunos e professores da área de tecnologia da informação. A obra conta agora com capítulos que focalizam o desenvolvimento do software desde seu início, com seus princípios básicos, até chegar ao desenvolvimento ágil de software e os sistemas embarcados.

INDICAÇÕES DE FILME

Nome do filme: O Jogo da Imitação

Gênero: Drama

Ano: 2015

Elenco Principal: Benedict Cumberbatch, Keira Knightley, Matthew Goode, Charles Dance, Rory Kinnear e Mark Strong.

Comentário: O filme conta a história de vida de Alan Turing, que foi adaptada para o cinema. O intuito principal foi mostrar a construção de uma máquina que interpretava códigos nazistas, incluindo o "Enigma", o qual criptógrafos acreditavam ser inquebrável na época. Após desvendar as codificações, Turing torna-se herói, pois a máquina consegue o seu objetivo, alterando, consequentemente, um pouco o destino da guerra naquele período. Porém, em 1952, autoridades revelam sua homossexualidade, transformando a vida de Turing em um pesadelo.

UNIDADE II

Tipos de Teste

Rafael Maltempe da Vanso

Introdução

Neste capítulo, será tratado o conteúdo de teste um pouco mais aprofundado, com a apresentação de conceitos e técnicas que auxiliam na prevenção e combate de erros em sistemas projetos por desenvolvedores.

As principais técnicas de testes tratados aqui serão os testes unitários, de que irão tratar as unidades que compõem o software produzido. Em seguida, as técnicas: serão apresentados os testes de integração, que, como o próprio nome diz, trata de verificar a integração dos módulos, classes, e outros do sistema, para que assim possam ser verificados possíveis erros.

Em complemento, será ainda apresentado a técnica de teste de validação, que trabalha em cima da validação do software, a fim de verificar-se ele faz o que foi programado para fazer. Além disso, o teste de sistema e teste de aceitação serão também apresentados.

Fonte: Marina Putilova / 123RF.

1. Tipos de testes

O processo de teste de software possui dois objetivos distintos: um deles é demonstrar ao desenvolvedor e ao cliente que o software atende a todos os requisitos propostos; e, também, descobrir situações em que o sistema se comporta de maneira indesejável (SOMMERVILLE, 2011).

Ao se tratar de testes de software, devemos ter em mente que podemos utilizar diversas estratégias para a ação de testar. De um lado, podemos esperar até que o sistema esteja totalmente construído e, aí, sim, começar a executar testes à procura de erros, o que não é muito bom, pois resultará em um software defeituoso, podendo deixar alguns erros despercebidos. Em contrapartida, podemos aplicar os testes de forma modular, diariamente, sempre à procura de erros. Embora mais trabalhosa do que a primeira forma, pode ser muito mais eficaz. Muitas vezes, utiliza-se de uma visão incremental para a realização dos testes, começando com simples testes de unidade e evoluindo até determinados outros testes que levam a finalizar o processo do software.

1.1 Teste de Unidade

O Teste Unitário tem como objetivo verificar testes na menor unidade do projeto do sistema, ou seja, realizar testes em componentes, classes ou módulos que pertencem ao projeto de software (PRESSMAN; MAXIM, 2016).

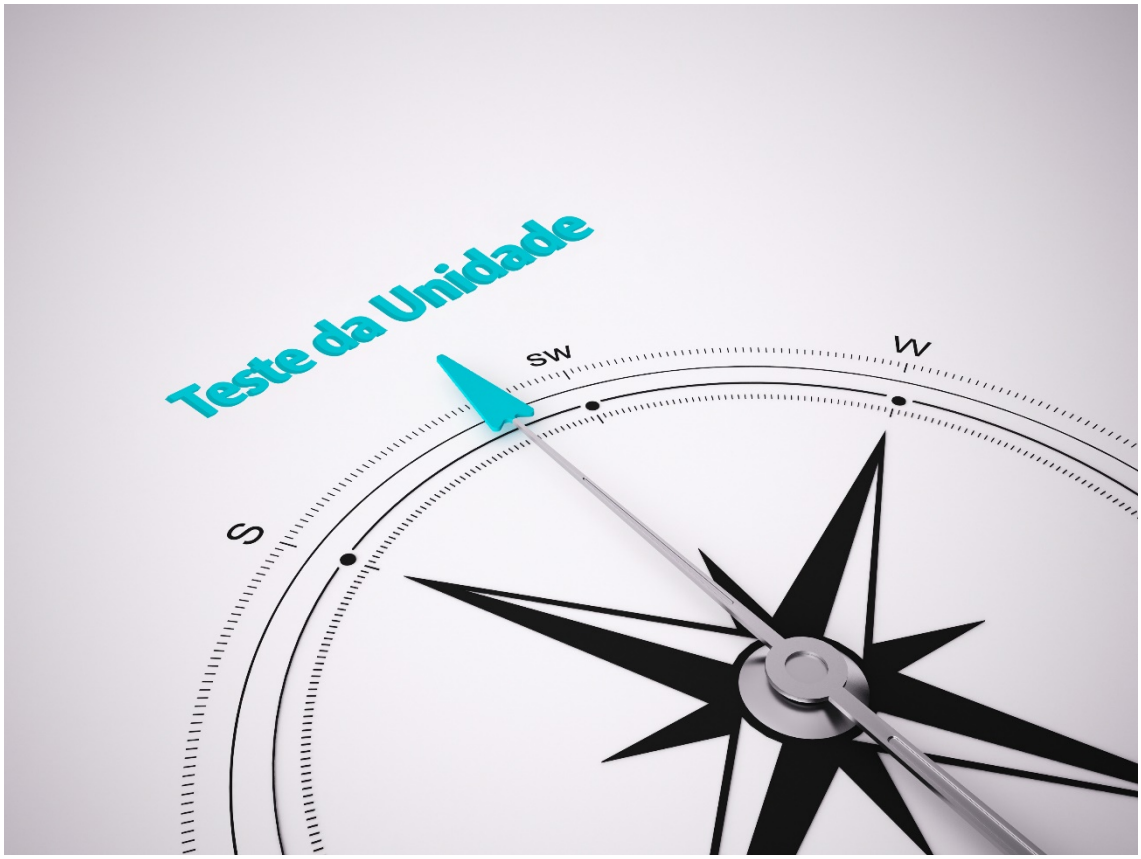


Figura 2.1 - Teste Unitário
Fonte: Nebuto / 123RF.

Testes de entrada e saída de uma unidade são realizados para descobrir erros que estão dentro do limite deste módulo unitário, ou seja, os erros aqui encontrados são limitados ao trecho do código exposto.

Nesse tipo de teste, você deve projetar testes, fornecendo uma cobertura para todas as características da unidade. Por exemplo, para uma classe desenvolvida, você deve testar operações e métodos presentes, definir valores para os atributos e manipular o objeto em todas as formas possíveis, o que significa que devemos simular ações que realizam mudanças no estado dessa classe (SOMMERVILLE, 2011).

REFLITA

Testes de unidade são os tipos de testes mais comuns. Inclusive, a maior parte dos testes escritos e recomendados por pesquisadores são testes unitários.

De uma forma geral, quando escrevemos testes unitários para uma determinada classe, esta bateria de testes testará o funcionamento desta classe sem interações com outras classes. Isso também acontece em relação aos testes unitários de componentes, de módulos, etc. Esquematicamente falando, Pressman e Maxim (2016) esquematizam os testes de unidade na figura 2.2:

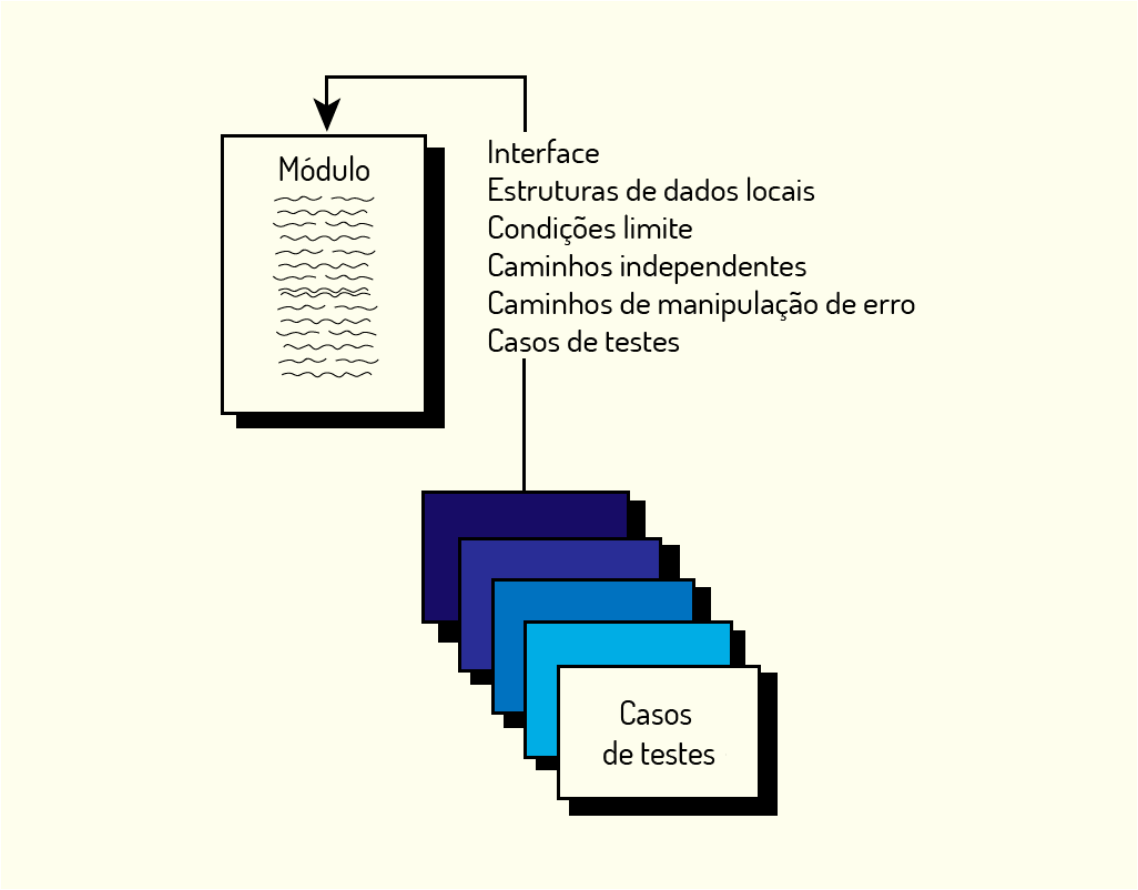


Figura 2.2 - Esquema Teste de Unidade

Fonte: Pressman e Maxim (2016, p.407)

No esquema proposto, a interface assegura que os testes flutuam de forma correta tanto para dentro quanto para fora da unidade testada. Já a estrutura dos dados são examinadas e devem garantir a integridade dos dados durante a execução do software. As condições de limite, como o próprio nome diz, agem no limite do módulo testado, limitando ou restringindo o processamento dentro do módulo. E, por fim, são testados todos os caminhos de manipulação de erros.

No Teste de Unidade, deve se dar atenção ao fluxo de dados, e este fluxo deve ser testado antes de qualquer outro teste. Se os dados não entram e saem corretamente, os outros testes se tornam discutíveis (PRESSMAN; MAXIM, 2016).

Casos de teste se utilizam de estruturas de dados, fluxo de controles e dados diversos, que podem conter diversos erros. Estes casos de testes deverão ser projetados cujo intuito principal é o de descobrir problemas. Um dos principais teste de unidade é o teste de fronteira, já que, de uma forma ou de outra, os softwares produzem falhas em suas fronteiras.

A partir das definições acima, entendemos assim que o teste de unidade é considerado como um assistente na etapa de implementação do código-fonte, podendo ser realizado até antes de se começar a codificação, ou após a sua geração pelo programador.

Devemos ter em mente, também, que o teste é custoso e que é importante escolher casos efetivos de testes, o que significa que os casos devem mostrar o que determinado módulo/componente/classe deve realizar; e, se houver defeitos, estes devem ser revelados por testes (SOMMERVILLE, 2011).

O que auxilia, também, é a seleção de diretrizes que podem ser utilizadas para a geração destes testes, como a escolha de entradas que possibilitam o sistema gerar todas as possíveis mensagens de erros que causem estouro de *buffers* de entrada, que obrigue a geração de saídas inválidas e cálculos que possuem valores grandes ou pequenos, e outros.

Aplicando essas características, a unidade é bem testada, os erros em sua maioria tratados; e de uma forma ou de outra, o testador ganha experiência na aplicação das diretrizes e testes.

FIQUE POR DENTRO

Vimos que os testes unitários são testes que analisam os erros da unidade do sistema que está sendo produzido, isolando cada parte do sistema para garantir o seu funcionamento adequado. Para o auxílio na realização destes testes, existem plataformas de testes para diversas linguagens, algumas até com interface gráfica. É importante estarmos cientes dessas ferramentas e ampliarmos conhecimentos. Vamos ler? Acesse o link:: <https://medium.com/assertqualityassurance/teste-unit%C3%A1rio-e-qualidade-de-software-acce7b9c537>.

ATIVIDADES

1) Um software é um programa que normalmente é desenvolvido para suprir e auxiliar o usuário a determinadas tarefas e necessidades do dia a dia. Para obter um software "redondo" é necessário aplicar testes de software. A alternativa que contempla corretamente o intuito de um teste de software é:
   a) Demonstrar ao usuário que o software foi construído em tempo hábil.
   b) Demonstrar se o software está de acordo e descobrir problemas de execução relacionados ao software.
   c) Demonstrar que o software precisa ser totalmente refeito após as descobertas de erros.
   d) Demonstrar que o usuário sempre tem a razão quando começa a utilizar o software e encontra erros.
   e) Demonstrar que o sistema não precisa ter a aplicação de testes muito coesos.

2. Teste de Integração

Figura 2.3 - Nuvem de Palavras para Unidade

Fonte: Radiantskies / 123RF.

Quando chegamos a essa etapa do teste, as vezes é inevitável nos perguntarmos: Se os módulos individuais, que foram testados, já funcionam individualmente, porque quando os juntamos devemos duvidar que eles podem não funcionar integrados? Bom, é uma resposta não tão difícil de se obter, não acham? Dados podem se perder quando módulos forem integrados, ou mesmo podem ter efeitos inesperados. E isso deve ser testado e consertado antes de prosseguir adiante.

Segundo Rios e Moreira (2013), o teste de integração é uma técnica utilizada para se construir a arquitetura de um software, e ao mesmo tempo, aplicar testes para a descoberta de erros quando integrados, e claro, com as suas interfaces.

Em muitos casos, programadores já tentam realizar estes testes quando terminam alguns módulos, integrando-os; porém podem resultar em problemas não previstos, e virar um caos ao final da construção do software, sendo encontrados diversos erros e conseqüentemente tendo a sua correção dificultada.

Temos alguns tipos de integração que auxiliam os nossos testes, como a integração incremental, na qual o sistema é construído e testado em pequenos incrementos e os seus erros são mais fáceis de serem isolados e corrigidos (RIOS; MOREIRA, 2013).

FIQUE POR DENTRO

Um ponto importante em testes de software é decidir quando devemos testar e como começar a testar. Por isso devemos entender os principais conceitos e o funcionamento de cada tipo de teste de software. Nesse sentido separamos um link importante sobre o teste de integração, que lhe ajudará a entender um pouco mais sobre esse tipo de teste. Vamos conferir pelo link: <http://blog.caelum.com.br/unidade-integracao-ou-sistema-qual-teste-fazer/>.

Dentro da integração incremental, temos estratégias como o Teste Big-Bang, Teste Bottom-up, Teste de Fluxo de Dados, Teste Funcional e Teste Top-down. Vamos estudá-los nos próximos tópicos:

2.1 Bottom-up

O teste Bottom-up, ou também conhecido como Teste de integração ascendente, começa a ser produzido e testado em classes/módulo que possuem um nível mais baixo na estrutura do programa construído.

Podemos aplicar uma estratégia de integração, seguindo os seguintes passos (RIOS; MOREIRA, 2013):

1. Componentes de baixo nível são combinados e executam uma subfunção específica de software.
2. Um programa de controle para teste pode ser utilizado e é escrito para coordenar as entradas e saídas dos testes.
3. O módulo agregado é testado.

4. Os programas controladores são removidos, e os módulos agregados são combinados, de forma a serem movidos para cima na estrutura do software produzido.

2.2 Top-Down

Este teste também é chamado de teste de integração descendente, e como o próprio nome diz, os módulos são integrados movendo-os para baixo a partir da arquitetura do sistema produzido, começando pelo módulo de controle principal (RIOS; MOREIRA, 2013).

Neste tipo de teste, os módulos são incorporados a estrutura: primeiro-em-profundidade (*depth-first*) ou primeiro-em-largura (*breadth-first*), e o processo de integração é realizado através dos seguintes passos (RIOS; MOREIRA, 2013):

1. O módulo de controle principal é utilizado como um testador e todos os componentes que são subordinados a ele são substituídos por programas controladores de testes.
2. Dependendo da forma de integração, os programas controladores de teste subordinados são substituídos, um de cada vez, pelos componentes reais.
3. Os testes são realizados quando cada componente é integrado.
4. Ao fim de cada conjunto de testes, outro controlador de testes substitui o componente real.
5. Um teste de regressão pode ser executado para garantir que não tenham sido introduzidos novos erros.

Este tipo de teste verifica os principais pontos de controle ou decisão no processo de teste do software de forma antecipada, o que torna um gerador de confiança para as partes interessadas no programa.

2.3 Fluxo de dados

Na técnica de Fluxo de dados, a integração dos componentes, programas, módulos ou subsistemas, é feita pelo desenho de fluxo de dados (RIOS; MOREIRA, 2013).

Desta forma, podemos entender que um teste de fluxo de dados é um método que seleciona caminhos de testes de um programa de acordo com a localização das definições. Assim, os caminhos são integrados e verificados possíveis problemas de comunicação, dependendo de como o fluxo dos dados são realizados; e posteriormente corrigidos.

REFLITA

Ter um sistema que comprometa algo em uma empresa pode causar diversos transtornos. Isso afeta a imagem ou o negócio da organização. De que forma afeta a organização socialmente? E economicamente? De quais outras formas a imagem da organização é afetada por isso?

2.4 Funcional

Na técnica funcional, a integração é feita baseada na junção de componentes, programas, módulos ou subsistemas que produzam um resultado funcional significativo para os usuários (RIOS; MOREIRA, 2013).

Em outras palavras, podemos dizer que essa técnica verifica se os requisitos funcionais, depois de serem integrados nos módulos, programas, classes e outros, continuam tendo a sua funcionalidade, conforme especificado no início do sistema, e que ainda tenha significado para o usuário final quando utilizado por eles.

2.5 Big-Bang

Segundo Rios e Moreira (2013), a abordagem “big-bang” combinam componentes, programas, módulos ou subsistemas de uma só vez; após serem testados isoladamente, como podemos verificar na imagem a seguir.

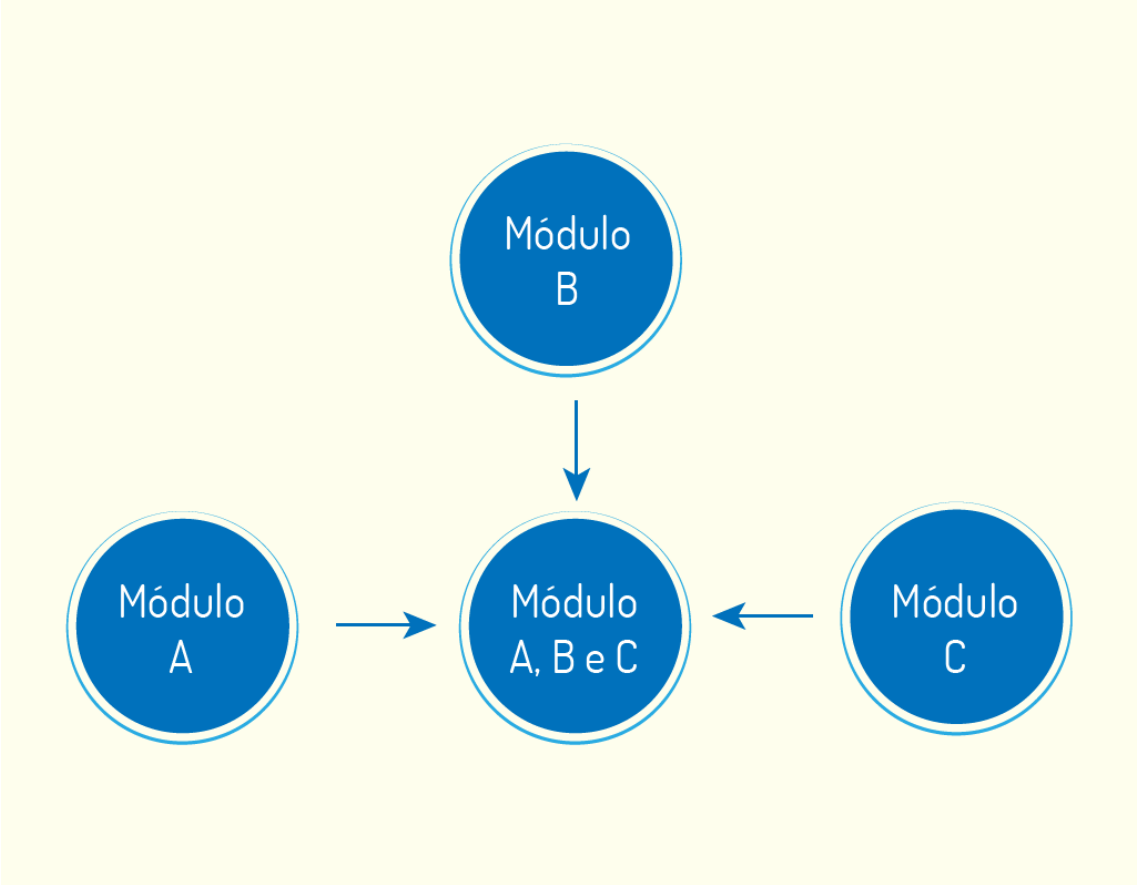


Figura 2.4 - Integração de módulos via técnica Big-Bang
Fonte: Elaborada pelo autor.

Esse tipo de teste normalmente é utilizado devido às pressões que ocorrem diariamente e desta forma, demonstram como o sistema está sendo operado.

O maior problema do uso desta técnica é caso ocorra problemas relacionados na interface de um módulo com outro, sendo difícil assim encontrar a causa de sua falha. Normalmente é aconselhável utilizar esta técnica para sistemas menores, estáveis e bem estruturados cujos componentes já passaram por testes de unidade e em implementações orientadas a objetos.

Dificilmente esta técnica trará benefícios para outros tipos de sistemas que não sejam esses citados, podendo sim significar em ganho de diversos problemas do que a resolução.

ATIVIDADES

2) Existem várias técnicas que contemplam os teste de software, e cada uma pode ser aplicado em um determinado momento, ou em qualquer momento do desenvolvimento do software. A técnica do teste de integração é definida corretamente como:
   a) Teste que realiza a descoberta de erros em partes unitárias do software produzido.
   b) Teste para a descoberta de erros quando módulos, classes e outros são integrados com as interfaces.
   c) Alternativa: Teste que verifica somente a integração com a interface do banco de dados utilizado.
   d) Teste que verifica a integração do software com o usuários final, ou seja, se o usuário interage corretamente com o software produzido.
   e) Teste que trabalha com problemas relacionados a integração entre os desenvolvedores do software.

3. Teste de Validação

O Teste de Validação tem por objetivo autenticar algo. Ele se inicia ao término do teste de integração. Assim sendo, são executados quando os módulos de cada unidade já foram testados e o software está completamente integrado e seus erros de interface já foram devidamente corrigidos (PRESSMAN; MAXIM, 2016).

Figura 2.5 - Teste de Validação

Fonte: Convisum / 123RF.

Esse tipo de teste está ligado a ações perceptíveis ao usuário e as saídas reconhecidas pelo próprio usuário. Assim, a validação é bem sucedida quando o sistema está funcionando da maneira esperada pelo cliente.

REFLITA

Entendemos que a validação é um sucesso quando o software executa o que o cliente espera, ou seja, cumpre com suas expectativas. Tendo isso em mente, pense sobre o seguinte: o quão importante é ouvir e compreender o que o cliente deseja?

Temos que a validade do software é realizada a partir de casos testes que averiguam, e por conseqüente, demonstram a conformidade dos requisitos, ou seja, procedimentos de testes definem características específicas, que conduzidas adequadamente, garantem que os requisitos funcionais sejam satisfeitos, que todas as características relacionadas ao comportamento do sistema sejam obtidas, que a documentação esteja correta, e por fim, que qualquer outra característica relacionada ao requisitos sejam cumpridos (PRESSMAN; MAXIM, 2016).

Nesta etapa, após a validação ocorrer, podemos perceber que as características propostas e o desempenho estão de acordo, e assim são aceitas; ou ainda, pode-se descobrir determinados problemas que geram deficiências. Estes desvios raramente são corrigidos antes do prazo de entrega do sistema ao cliente, desta forma, deverá haver um consenso sobre um novo prazo para resolver estas pendências (RIOS; MOREIRA, 2013).

3.1 Revisão de configuração

Figura 2.6 - Revisão de Configuração
Fonte: Etiamos / 123RF.

Quando falamos em processo de software, necessitamos entender alguns conceitos. Temos que o seu resultado são programas de computadores, sejam eles o código-fonte ou o executável; dados ou conteúdos que estão contidos nos programas ou externos a ele; e por fim, os produtos que descrevem os programas de computador. Estes itens entram como informações do processo de criação do software e que são chamados de configuração do software (PRESSMAN; MAXIM, 2016).

Assim, dizemos que a gestão de configuração de software é um conjunto de atividades desenvolvidas com a finalidade de gerenciar possíveis alterações a partir do ciclo de vida do software construído.

Segundo Pressman e Maxim (2016), a revisão da configuração é uma etapa importante no processo de validação, já que tem a finalidade de revisar e garantir que todos os elementos relacionados a configuração do sistema tenham sido tratados, desenvolvidos e documentados; para desta forma, auxiliar nas atividades relacionadas ao suporte do software.

3.2 Teste Alfa e Beta

Instruções de uso podem ser mal interpretadas, podem acontecer combinações estranhas de dados, resultados que são claros para o pesquisador podem ser confusos para um usuário ao executar o sistema, e assim por diante. É complicado, mas possível. E quando tratamos de um único cliente que utilizará o sistema, fica mais fácil validar os requisitos. Porém, quando se trata de diversos usuários, o problema fica maior e mais difícil.

Nesse sentido, muitos construtores de software usam processos chamados de teste alfa e beta para que, assim, seja possível descobrir erros que apenas o usuário final pode encontrar.

Desta forma, Pressman e Maxim (2016) citam o teste alfa, que é realizado na instalação pelo desenvolvedor do sistema e executado por um grupo de usuários finais, ou seja, o desenvolvedor do sistema analisa como os usuários do sistema utilizam-no em seu cenário natural, o da empresa.

Assim, o desenvolvedor consegue registrar todos os erros e problemas causados por uma possível utilização errônea por parte do usuário; conduzindo os testes em um ambiente controlado.

Já o teste beta é conduzido nas instalações de um ou mais usuários finais. Nesse tipo de teste, o desenvolvedor geralmente não fica presente, e, conseqüentemente, não pode ter o controle pelas ações realizadas pelo cliente. Assim, é essencial que o cliente faça o registro dos problemas encontrados, e de posse destes problemas, o engenheiro de software realiza possíveis modificações (PRESSMAN; MAXIM, 2016).

Teste Beta também recebe o nome de teste aceitação do cliente, que executa um determinado número de testes específicos com a finalidade de encontrar erros antes de o software desenvolvido ser entregue totalmente.

FIQUE POR DENTRO

Como vimos, o teste beta também é conhecido como teste aceitação que aproxima o contato entre a equipe de desenvolvimento e o cliente, analisando se há uma aceitação ou não por parte do cliente das funções desenvolvidas pelo desenvolvedor. Desta forma, separamos um link interessante com conceitos e boa práticas a respeito do teste de aceitação. Acesse: <http://blog.caelum.com.br/testes-de-aceitacao-e-suas-boas-praticas/ >.

ATIVIDADES

3) Esse tipo de teste está ligado a ações perceptíveis ao usuário. Desta forma, é verificada a funcionalidade do sistema como se o cliente estivesse de acordo com o que está sendo executado. Essa descrição trata da definição do teste de:
   a) Unitário.
   b) Validação.
   c) Integração.
   d) Sistema.
   e) Conexão.

4. Teste de Sistema

Realizar um teste de sistema envolve a integração de componentes para que assim seja criada uma outra versão do sistema e, em seguida, o teste integrado do sistema. Assim, o teste de sistema é, na realidade, uma série de diferentes testes que verificam se todos os componentes são compatíveis quando são integrados corretamente nesta versão do sistema (SOMMERVILLE, 2011).

Figura 2.7 - Teste e otimização

Fonte: Etiamos / 123RF.

Pressman e Maxim (2016) dizem que o teste do sistema deve centrar-se em testar todas as interações entre objetos e componentes que compõem um sistema. Essa interação deve procurar bugs que são revelados quando um componente é utilizado por outros componentes na junção final do sistema. O teste de interação ajuda a verificar possíveis equívocos realizados pelos desenvolvedores de módulos sobre outros módulos do software.

4.1 Teste de Recuperação

Você já deve ter passado, durante uma implementação de sistemas, com problemas relacionados a falhas em sistemas, e que, de uma forma ou de outra, retornam falhas que interferem no processamento do sistema, podendo até causar paradas inesperadas.

Na maioria das vezes, os sistemas devem tolerar estas falhas para que não gerem essa paralisação do sistema. Testes de recuperação são testes que têm o objetivo de forçar as falhas de várias maneiras, para que, no final, se verifique se a recuperação após as falhas está sendo executada corretamente (PRESSMAN; MAXIM, 2016).

Neste caso, se a recuperação for executada de forma automática pelo sistema, os mecanismos de verificação, reinicialização e recuperação de dados são avaliados quanto à uma possível correção. Se esta recuperação requer intervenção humana, deve ser avaliado o tempo médio de reparo para determinar se está dentro dos limites aceitáveis (SOMMERVILLE, 2011).

4.2 Teste de Segurança

O teste de segurança deve verificar se mecanismos de proteção do sistema vão proteger de fato contra possíveis acessos indevidos. Sabemos que qualquer software que trabalha com informações importantes é sempre alvo de pessoas que desejam adquirir essas informações que não lhes pertence.

Podemos chamar esses invasores de *Hackers*, funcionários que possuem problemas com a empresa e querem estas informações para prejudicar a empresa e diversos outros. Assim, a segurança deve ser testada para que essa vulnerabilidade seja tratada.

Sommerville (2011) diz que, durante o teste de segurança, os pesquisadores devem fazer o papel da pessoa que está invadindo o sistema, devem pensar e analisar as diversas formas papéis que o mesmo utiliza para esta ação. Uma destas atividades é tentar obter senhas por meios externos ao sistema, atacar o software, projetando o rompimento das defesas implementadas, sobrecarregar o sistema ou outros meios que possam causar erros.

Um bom teste de segurança, em seu final, conseguirá cumprir o seu papel, que é invadir o sistema; porém isso leva tempo e recurso. Desta forma, o papel principal do desenvolvedor é tornar o custo para a invasão do seu sistema maior do que as informações que o próprio sistema armazena.

Sabemos que a segurança é fundamental, seja qual sistema for. Um sistema coerente e que possui uma integridade em seus dados, podemos afirmar que possui implementado um módulo de segurança.

4.3 Testes de esforço

O teste de esforço nada mais é do que colocar o sistema em situações anormais de execução. Normalmente, testes produzem uma ou duas interrupções, o que são normais; os testes de esforço visa aumentar esse número de interrupções para que assim o sistema possa ser verificado mais adequadamente para cada ação. São executados diversos casos de testes que requerem o máximo de memória ou outros recursos; teste que podem causar excessiva procura por dados residentes em disco; e outros; podemos dizer assim que o testados procura “quebrar” o programa (PRESSMAN; MAXIM, 2016).

Como variação deste teste, temos o teste de sensibilidade, que tem como objetivo descobrir combinações de dados dentro de classes/módulos de entradas válidas que talvez causem instabilidade ou processamento inadequado do software.

REFLITA

"Se você está tentando encontrar verdadeiros defeitos no sistema e ainda não submeteu o seu software a um verdadeiro teste de esforço, então é hora de começar a fazê-lo." Boris Beizer (apud PRESSMAN; MAXIM, 2016, p. 487).

4.4 Teste de desempenho

Segundo Sommerville (2011) e Pressman e Maxim (2016), o teste de desempenho é projetado para testar, como o próprio nome diz, o desempenho em tempo de execução do sistema em relação a um sistema integrado. O teste de desempenho pode ser realizado em todas as etapas de teste, como por exemplo, em nível de unidade; no entanto, o seu verdadeiro desempenho só poderá ser válido, de fato, quando ele for verificado a partir da integração de todos os elementos do sistema.

Usualmente os testes de desempenho são acoplados ao teste de esforço, que trabalham com partes de hardware e software, ou seja, medindo a utilização dos recursos, monitorando assim o sistema com os instrumentos presentes e suas possíveis falhas.

Como em outros tipos de testes, o teste de desempenho também irá mostrar se o sistema atende aos requisitos e também auxiliará na descoberta de defeitos e problemas. Vejamos um exemplo prático disso:

> Por exemplo, digamos que você está testando um sistema de processamento de transações que é projetado para processar até 300 transações por segundo. Você começa a testar esse sistema com menos de 300 transações por segundo; então, aumenta gradualmente a carga no sistema para além de 300 transações por segundo, até que esteja bem além da carga máxima de projeto do sistema e o sistema falhe. (SOMMERVILLE, 2011, p.159).

Esse tipo de teste é também conhecido como teste de estresse e tem duas funções: primeiro, a de testar o comportamento de falha do sistema e, também, a de estressar o sistema e trazer à tona defeitos ainda não descobertos.

4.5 Testes de Disponibilização

O intuito do teste de disponibilização é verificar se o software implementado funciona adequadamente no ambiente operacional que o mesmo deve operar. Se o mesmo foi implementado para operar em diversas plataformas, o mesmo deve ser testado para a verificação de possíveis problemas.

Pressman e Maxim (2016) dizem que o teste de disponibilização examina todos os procedimentos de instalação do sistema que serão utilizados pelos clientes e toda a documentação gerada ao final do sistema que será usada para fornecer o software para os usuários finais.

Para finalizar, integrado aos testes de disponibilização, devemos também realizar os testes de segurança citados anteriormente, já que a questão da segurança é um problema importante que deve ser tratado.

FIQUE POR DENTRO

Temos diversas técnicas que auxiliam a melhorar o desempenho através da detecção de erros em possíveis gargalos, e claro, aplicando diferentes tipos de circunstâncias. Falamos sobre testes de desempenho, carga e estresse. Vamos dar uma conferida em alguns outros? Para isso, disponibilizamos o link a seguir. Acesse: <https://www.testar.me/teste-de-software>.

ATIVIDADES

4) No teste de sistema, temos técnicas como teste de recuperação, teste de segurança, esforço, desempenho e outros. Podemos dizer que esses tipos de testes se completam. A definição correta de teste de sistema está descrito na alternativa:
   a) Teste de sistema é o teste que verifica as unidades separadas do sistema, a fim de encontrar erros.

b) Teste do sistema deve centrar-se em testar todas as interações entre objetos e componentes que compõem um sistema.
c) Teste de sistema são teste que verificam a aceitação do sistema por parte do usuário final.
d) Teste de sistemas visam realizar a integração dos módulos/classes do sistemas a fim de verificar possíveis erros.
e) Teste de sistemas visam verificar se usuários e desenvolvedores estão integrados para a construção do software.

5. Teste de aceitação

Este teste pode revelar erros e omissões na definição dos requisitos do sistema e também pode revelar problemas de requisitos em que os recursos do sistema não atendam às necessidades do usuário ou o desempenho do sistema seja inaceitável.

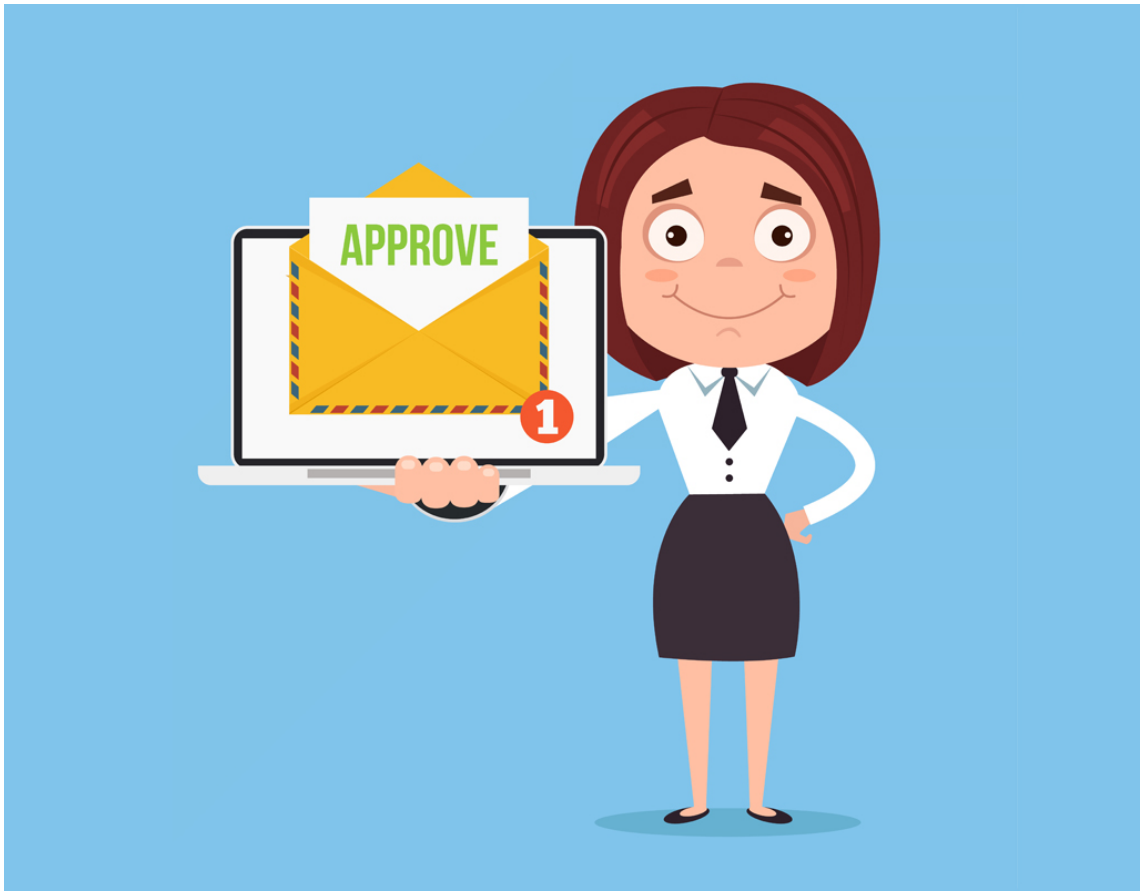


Figura 2.8 - Teste de aceitação
Fonte: Irina Griskova / 123RF.

Um teste de aceitação também é utilizado para realizar a aproximação com o cliente final em relação ao resultado que se espera do sistema, além claro, do próprio usuário poder auxiliar desenvolvedores no que eles esperam, e assim garantir o mais próximo possível com a conformidade esperada (RIOS; MOREIRA, 2013).

Podemos completar ainda, afirmando que o teste de aceitação é a última ação antes da implantação do software, e é realizado para avaliar a sua qualidade externa e também quanto à qualidade de uso; desta forma, é um teste que exprime a relação com o cliente, que deve ser colocado a par neste teste de software, já que o intuito principal é verificar se este software produzido está pronto para uso dos próprios usuários (SOMMERVILLE, 2011).

Dentro do teste de aceitação temos os testes denominados de “alfa”, “beta” e ainda “paralelo”. Assim podemos explanar que o teste alfa verifica a qualidade externa, o teste beta a qualidade de uso, e o teste de paralelo quando um sistema é desenvolvido para substituir outro que já está em funcionamento (RIOS; MOREIRA, 2013).

Agora que nos aprofundando um pouco mais sobre as estratégias de testes comuns no teste de aceitação, iremos conceituar um pouco mais sobre três estratégias propostas, sendo a aceitação formal, a aceitação informal (teste alfa) e o teste beta.

5.1 Teste de Aceitação Formal

Este tipo de teste é uma extensão ao teste de sistema e deve ser projetado como tudo o que é feito no sistema, com muito cuidado e muito detalhado. Sendo assim, podemos dizer que estes testes são subconjuntos dos testes de sistema.

Este teste pode ser executado pelo grupo de teste de uma organização que representam o usuário final, ou ainda, executado inteiramente pelos usuários finais.

Como benefício, temos que as funções e os recursos que devem ser testados são conhecidos por inteiro, e podem ser também medidos; o seu progresso pode ser monitorado e critérios de aceitabilidade do sistema também conhecidos; e por fim, este tipo de teste pode ser automatizado (RIOS; MOREIRA, 2013).

Testes, em sua maioria, geram benefícios, porém testes também podem não revelar determinados defeitos que o software contém, já que, em sua maioria, os testes que são executados são para a procura de defeitos preestabelecidos, ou seja, esperados.

5.2 Teste de Aceitação Informal

Neste tipo de teste, as características e métodos para a execução não são realizados com o mesmo rigor apresentado no teste de aceitação formal. Neste teste, o testador individual determina o que e como fazer. Desta forma o teste se torna mais subjetivo do que o teste apresentado anteriormente.

O teste de aceitação informal normalmente é realizado pela organização do usuário que irá utilizar o sistema e possui alguns benefícios em sua aplicação, como o conhecimento das funções e recursos testados, critérios de aceitabilidade e monitoramento; e aqui serão revelados defeitos mais subjetivos (RIOS; MOREIRA, 2013).

Neste tipo de teste, também possuímos alguns problemas, pois como os teste são realizados pela organização do usuário final, não há um controle sobre os casos testes que são utilizados, e ainda pode acontecer uma comparação do novo sistema a ser implementado com o sistema antigo, o que prejudica a procura de erros no sistema (RIOS; MOREIRA, 2013).

5.3 Teste Beta

O teste beta é um dos testes menos controlados das opções apresentadas para o teste de aceitação. As características e a abordagem adotada aqui é de inteira responsabilidade do testador individual, ou seja, cada testador pode criar seu próprio ambiente de testes com funções, recursos e tarefas; desta forma cada um cria critérios para a aceitação ou não do sistema (RIOS; MOREIRA, 2013).

Este teste é o mais subjetivo entre os apresentados e por ser implementado por usuários finais, não possui nenhum gerenciamento por parte dos desenvolvedores do sistema.

Este tipo de teste aumenta a satisfação do cliente em relação aqueles que participam. Porém, com ele é difícil medir o progresso de como está o teste, pois usuários podem se adaptar ao sistema e não conseguir encontrar mais defeitos; além da possível comparação com o sistema antigo (RIOS; MOREIRA, 2013).

FIQUE POR DENTRO

Todo tipo de teste tem sua parcela de importância na construção de um software, e o teste de aceitação não é diferente disso. Desta forma, separamos uma matéria que apresenta boas práticas em relação a aplicabilidade do teste de aceitação. Acesse:

<http://shipit.resultadosdigitais.com.br/blog/5-boas-praticas-para-se-aplicar-em-testes-de-aceitacao/>

ATIVIDADES

5) O teste de aceitação visa revelar erros e omissões na definição dos requisitos do sistemas, e que desta forma, não atendem às necessidades dos usuários. Essa técnica possui três estratégias, que estão representadas na alternativa:
   a) Teste de Fluxo de Dados e Teste Funcional.
   b) Teste de Aceitação Formal, Teste de Aceitação Informal e Teste Beta
   c) Teste de Recuperação e Teste de Segurança.
   d) Teste Bottom-up e Teste Big-Bang.
   e) Teste de Disponibilização e Teste de Desempenho.

INDICAÇÕES DE LEITURA

Nome do livro: Engenharia de Software. Uma Abordagem Profissional

Editora: AMGH

Autor: Roger S. Pressman

ISBN: 978-8580555332

Comentário: O livro de Pressman segue a linha dos livros mais importantes da área de Engenharia de Software. Essa última edição está amplamente atualizada para incluir os novos tópicos da “engenharia do século 21”. Há novos capítulos que abordam sobre um dos quesitos mais importantes atualmente, que é a segurança de software e os desafios específicos ao desenvolvimento para aplicativos móveis.

INDICAÇÕES DE FILME

Nome do filme: Piratas do Vale do Silício

Gênero: Drama

Ano: 1999

Elenco principal: Paul Freiberger, Michael Swaine

Comentário: Um filme que conta um pouco da história da tecnologia e o surgimento de Steve Jobs e Bill Gates. A história é narrada quando, os dois, ainda estudantes, lideraram uma revolução que integrou os computadores ao nosso dia a dia. Se deu assim o surgimento das maiores empresas de tecnologia mundial.

UNIDADE III

Técnicas de Testes

Rafael Maltempe da Vanso

Introdução

Testes devem ser entendidos e tratados com grande consciência por todos os envolvidos no desenvolvimento de um software. Desta forma, este capítulo visa atribuir um pouco mais de conhecimento sobre as técnicas de testes de software.

Iremos trabalhar com conceitos de teste de caixa-preta, ou ainda teste funcional, que visa procurar e identificar falhas existentes na programação. Outro teste, é o teste de caixa-branca, ou ainda teste estrutural, que é um teste que usa a estrutura de controle do projeto para derivar casos de testes.

Em sequência, falaremos também sobre testes de regressão, teste de carga, teste de estresse, teste de usabilidade e teste de segurança, que também são testes importantes e que devem ser conceituados e entendidos por todos os profissionais de tecnologia. Vamos começar nossos estudos?

Fonte: Egor Kotenko / 123RF.

1. Técnicas De Teste

Entendemos que o objetivo de testar é encontrar erros, e que um bom teste é aquele que tem alta probabilidade de encontrar um erro no sistema. Desta forma um engenheiro, um testador, ou ainda o profissional que irá trabalhar com testes, deve implementar ou projetar testes, que de uma forma ou de outra, possibilitam que o objetivo principal seja atingido; ou seja, atingir o mínimo de esforço para encontrar e resolver possíveis problemas (SOMMERVILLE, 2011).

Sendo assim, as técnicas de teste de software devem apresentar determinadas características importantes, como testabilidade, operabilidade, observabilidade, controlabilidade, decomponibilidade, simplicidade, estabilidade e compreensibilidade.

Quando falamos em testabilidade de software, estamos falando da simplicidade e a facilidade com que um programa de computador pode ser testado; já a operabilidade diz respeito quanto melhor funcionar, mais eficientemente pode ser testado. A observabilidade indica que o que você vê é o que você testa e controlabilidade diz respeito a quanto melhor pudermos controlar o software, mais o teste pode ser automatizado e otimizado. Em seqüência, a decomponibilidade, na qual controlando o escopo do teste, podemos isolar problemas mais rapidamente e executar um reteste mais racionalmente. No que diz respeito a simplicidade, quanto menos tiver que testar, mais rapidamente podemos testá-lo; já a estabilidade, quanto menos alterações, menos interrupções no teste; e por fim, a compreensibilidade que diz que quanto mais informações tivermos, mais inteligente será o teste (PRESSMAN; MAXIM, 2016).

1.1 Teste Funcional (Caixa-preta)

O teste de caixa-preta, ou teste funcional ou comportamental, é baseado nos requisitos funcionais do software, com foco nos requisitos da aplicação, nas ações que ela deve desempenhar. Esta técnica não está preocupada em como o sistema age internamente ou com o conteúdo do código, mas com o que é gerado como saída e entrada de dados.

Comparando com outros tipos de testes, podemos dizer que este é relativamente mais simples. Assim, um exemplo simples é que podemos verificar a consistência de dados que serão inseridos e que fazem relacionamento com a interface, ou ainda, inserir possíveis entradas incorretas dos dados e verificar como o sistema se comporta (PRESSMAN; MAXIM, 2016).

No entanto, essa técnica apresenta algumas dificuldades, como por exemplo, testar todas as entradas possíveis. Ela é essencial no desenvolvimento de um sistema, porém, não é suficiente para identificar alguns problemas que ocorrem em projetos de sistema (SOMMERVILLE, 2011).

O teste de caixa-preta é útil para tentarmos encontrar determinados erros, como por exemplo funções incorretas, erros de interface, erros em estrutura de dados ou acesso a base de dados, erros de comportamento ou desempenho, erros de iniciação e término.

Desta forma, ao realizar os testes, devemos buscar simular todo tipo e espécie de erros que qualquer usuário que utilizar o sistema pode cometer, como por exemplo: usar como data de nascimento um valor incorreto, como datas atuais ou futuras; preencher campos com um número insuficientes ou errados; usar valores negativos para números que só aceitam valores positivos, botões que não executam as ações que foram programadas e diversos outros (MOLINARI, 2013).

Segundo Pressman e Maxim, os testes funcionais devem responder às seguintes questões:

- Como a validade funcional é testada?
- Como o comportamento e o desempenho do sistema é testado?
- Que classes de entrada farão bons casos de teste?
- O sistema é particularmente sensível a certos valores de entrada?
- Como as fronteiras de uma classe de dados é isolada?
- Que taxas e volumes de dados o sistema pode tolerar?

- Que efeito combinações específicas de dados terão sobre a operação do sistema? (PRESS; MAXIM, 2016, p. 440)

Esses questionamentos são importantes e facilitam o entendimento de como são os testes funcionais, e claro, que tipo de características negativas devem ser trabalhadas no sistemas.

REFLITA

Aplicar novas tecnologias pode auxiliar o desenvolvedor ou o pesquisador do software a executar determinados tipos de testes de uma melhor forma. Mas será que é melhor aplicar apenas métodos ágeis em testes de software?

1.2 Método de teste com base em grafo

Antes de iniciar o teste de caixa preta, devemos entender seus objetivos; após isso, deve ser definida uma série de testes. Em outras palavras, o teste de software começa criando um grafo de objetos importantes com suas respectivas relações, para que assim possam ser realizados os testes cabíveis nestes objetivos e relações, a procura de erros.

Pressman e Maxim dizem que:

> Para executar esses passos, você começa criando um *grafo* — uma coleção de *nós* que representam objetos, *ligações* que representam as relações entre objetos, *pesos de nó* que descrevem as propriedades de um nó (por exemplo, o valor específico de um dado ou comportamento de estado), e pesos de ligação (*link weights*) que descrevem alguma característica de uma ligação. (PRESSMAN; MAXIM, 2016, p. 439)

Podemos verificar a representação simbólica de um grafo na figura a seguir:

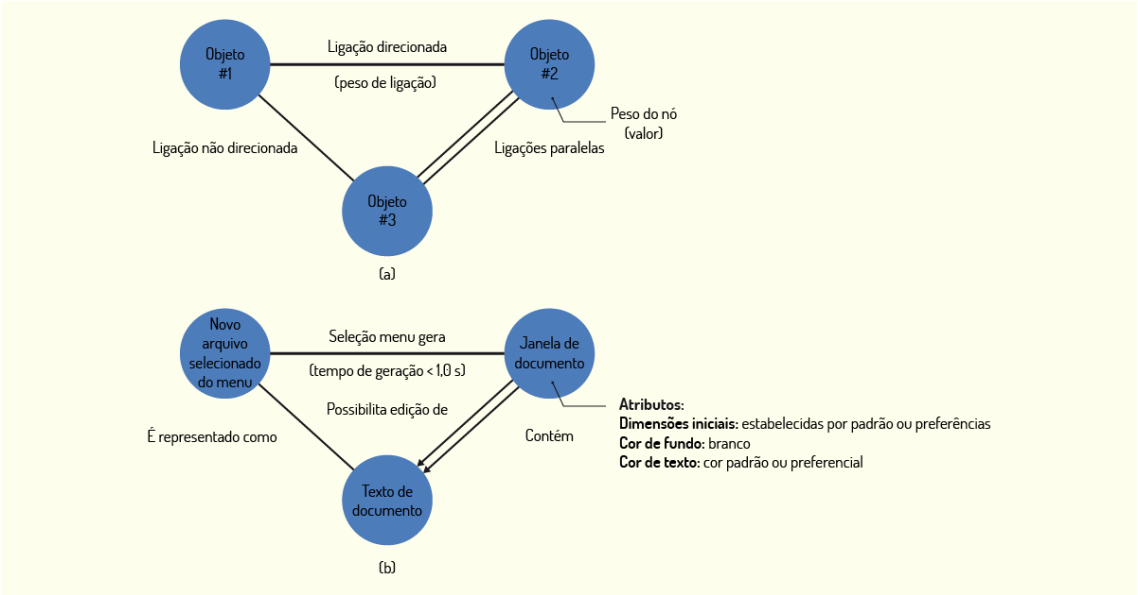


Figura 3.1 - Notação de Grafo (a) e Exemplo Simples (b)
Fonte: Pressman e Maxim (2016, 439).

Existem alguns métodos de teste comportamental que se utilizam de grafos, como o de modelagem de fluxo de transação, que é relacionado a conexão lógica e transações realizadas; a modelagem de estado finito, que está relacionado a estados observáveis por usuários; e diversos outros.

1.3 Particionamento de equivalência

Este é um método existente no teste de caixa-preta e tem o intuito de dividir o domínio de entrada em classe de dados, e que a partir disso podem ser criados casos de testes, que por sua vez, descobrem uma classe de erros (RIOS; MOREIRA, 2013). Desta forma, podemos dizer que o projeto de casos de teste para particionamento de equivalência possibilita que as classes de equivalência sejam analisadas e avaliadas da melhor forma, dependendo de como são inseridos possíveis entradas.

Podemos verificar na imagem abaixo, um pouco mais sobre o particionamento de equivalência:

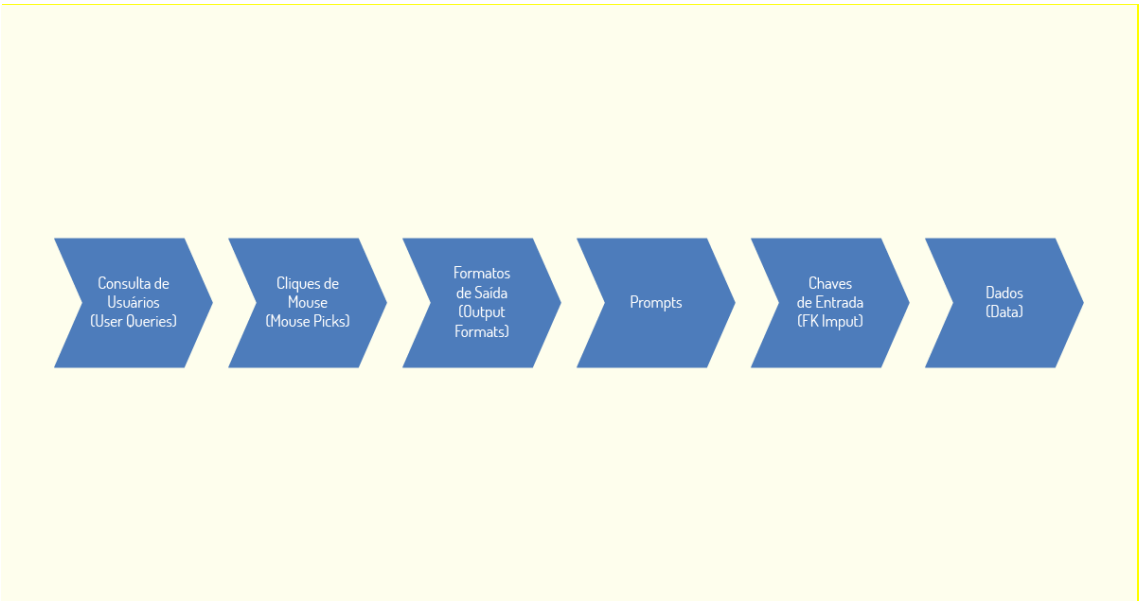


Figura 3.2 - Particionamento de Equivalência
Fonte: Molinari (2013, p.168).

Classes de equivalência são estabelecidas conforme algumas regras, que podemos verificar a seguir:

> 1. Se uma condição de entrada especifica um intervalo, são definidas uma classe de equivalência válida e duas classes de equivalência inválidas.
> 2. Se uma condição de entrada requer um valor específico, são definidas uma classe de equivalência válida e duas classes de equivalência inválidas.
> 3. Se uma condição de entrada especifica um membro de um conjunto, são definidas uma classe de equivalência válida e uma classe de equivalência inválida.
> 4. Se uma condição de entrada for booleana, são definidas uma classe válida e uma inválida. (PRESSMAN; MAXIM, 2016, p. 441).

Estabelecer regras auxiliam na tratativa de problemas de forma adequada, estabelecendo problemas que podem, talvez, ser esquecidos ou tratados de forma errada.

1.4 Análise de valor limite

A análise de valor limite é uma técnica de projeto de casos de teste na qual são selecionados possíveis casos de testes que empregam valores considerados limites e finaliza como particionamento de equivalência. Desta forma, dizemos que esta técnica destaca a seleção de casos de teste nas extremidades das classes; e em vez de dedicar-se a somente condições relacionadas a entrada, ela também trabalha com casos de testes relacionados a saída dos dados (MOLINARI, 2013).

A maioria dos engenheiros de software trabalham intuitivamente com a análise de valor limite, desta forma, ao aplicar essas diretrizes, o teste de fronteira será mais completo, podendo assim encontrar mais tipos de erros.

1.5 Teste Baseado em Modelos

Este tipo de teste é uma técnica que usa informações contidas no modelo de requisitos como base para geração de casos de testes. Também é conhecido como *Model-Based Testing* – MBT.

Pressman e Maxim citam os cinco passos desta técnica, como podemos verificar a seguir:

> 1. Analise um modelo comportamental existente para o software ou crie um.
> 2. Percorra o modelo comportamental e especifique as entradas que forçaram o software a fazer a transição de um estado para outro.
> 3. Reveja o modelo comportamental e observe as saídas esperadas à medida que o software faz a transição de um estado para outro.
> 4. Execute os casos de teste.
> 5. Compare os resultados reais e esperado e tome a ação corretiva necessária. (PRESSMAN; MAXIM, 2016, p. 444).

Podemos dizer que este teste ajuda a descobrir erros no comportamento do software e é extremamente útil ao testar aplicações acionadas por eventos.

FIQUE POR DENTRO

Ao se falar em testes de software, além de entender conceitos, sempre devemos tratar do planejamento correto para se realizar testes de software. Um planejamento pode atribuir tipos de testes em determinados métodos/classes/módulos que podem conter determinados tipos de erros, dependendo das entradas inseridas. Nesse sentido, separamos um link muito interessante que fala sobre como planejar testes de software. Vamos verificar mais esse link e obter mais conceitos sobre o assunto? Acesse em: <https://webinsider.com.br/um-roteiro-para-planejar-testes-de-software/>.

ATIVIDADES

1) Temos a existência de técnicas que trabalham com a parte funcional e a parte estrutural do sistema. Sobre a técnica funcional, ou ainda denominada teste de caixa-preta, a alternativa que melhor a define é:

a) Esta técnica baseia-se na análise dos laços de repetição existentes em todo software desenvolvido.
b) Esta técnica é baseado nos requisitos funcionais do software, com foco nos requisitos da aplicação, nas ações que ela deve desempenhar.
c) Esta técnica analisa a estrutura de controle, ou seja, como é o estrutura de execução do sistema.
d) Esta técnica analisa os fluxos de dados de acordo com a sua localização no sistema.
e) Esta técnica trabalha com o desenvolvimento de software sem as especificações corretas do sistema.

2. Teste de Caixa Branca

O teste de caixa branca também é conhecido como teste estrutural. Com ele, o engenheiro pode criar casos de testes que garantem que todos os caminhos independentes foram exercitados ao menos uma vez; exercitam todas as decisões lógicas; executam todos os ciclos de testes em seus limites e dentro de suas fronteiras operacionais, e exercitam estruturas de dados internas para assegurar a validade (PRESSMAN; MAXIM, 2013).

Neste teste, o testador tem total acesso à estrutura interna, e o fato de conhecer toda esta estrutura, faz com que os testes sejam mais precisos.

2.1 Teste de Caminho Básico

Segundo Molinari (2013, p. 166), o teste de caminho básico pode ser definido da seguinte forma:

> O teste de caminho básico permite ao projetista de casos de teste derivar uma medida da complexidade lógica de um projeto procedimental e usar essa medida como guia para definir um conjunto base de caminhos de execução. (MOLINARI, 2013, p. 166).

2.2 Teste de Estrutura de Controle

Este tipo de teste trabalha com características relacionadas ao controle da execução do programa, analisando todos os comandos e desvios que pertencem ao mesmo, para que assim possa determinar quais estruturas são necessárias (PRESSMAN; MAXIM, 2016).

Podemos citar alguns critérios para essa técnica, como a chamada “Todos-nós”, que exige que cada comando presente no programa seja executado ao menos uma vez; outra que podemos citar é “Todos-caminhos”, que requer que todas as possibilidades de caminhos de execução no software sejam executados.

Bom, é claro que, se pensarmos globalmente no software, podemos dizer que executar todos os caminhos existentes é uma combinação ideal para um programa, já que podemos executar todas as ações e assim verificar os erros que podem ocorrer.

2.3 Teste de condição

O teste de condição, como o próprio nome diz, analisa condições. Assim sendo, é um tipo de teste que põe à prova as condições lógicas do software desenvolvido (PRESSMAN; MAXIM, 2013).

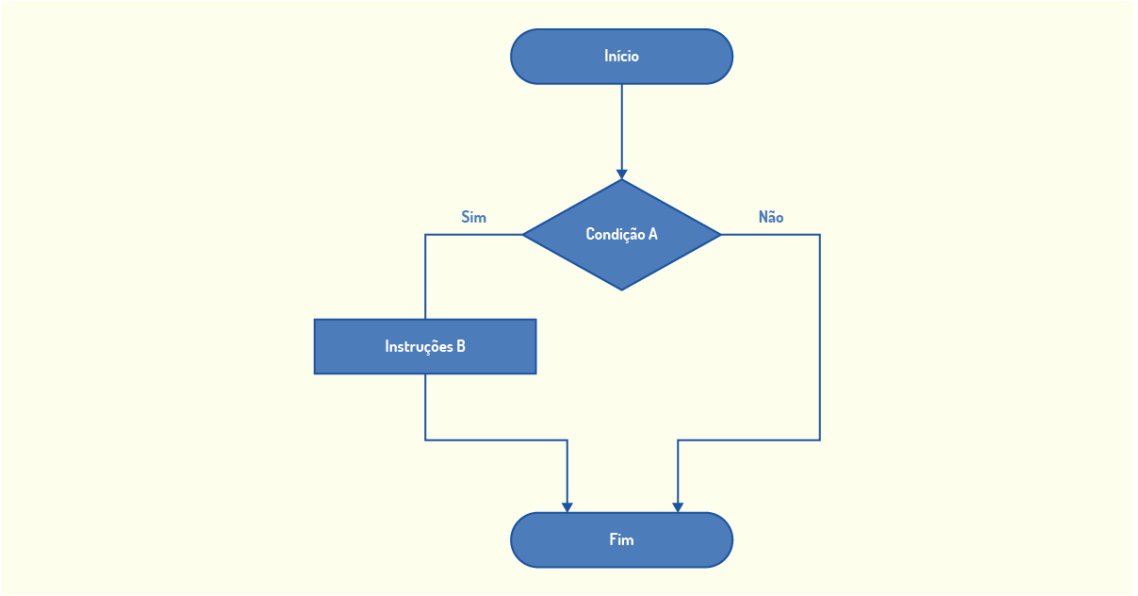


Figura 3.3 - Estrutura de condições
Fonte: Elaborada pelo autor.

Compreende-se que testar toda a condição do sistema é essencial. Esse tipo de teste realiza essa tratativa, além de encontrar outros tipos de erros no programa.

Como condição atrativa para determinadas ações, entende-se, assim, que se algo for verdadeiro, será executado um trecho de código; caso contrário, será executado outro. Assim, temos algumas condições existentes e que trabalham com expressões aritméticas, booleanas e relacionais. Dentro das condições podemos encontrar as simples, que tratam de análises com uma variável; já a composta, trabalha com uma ou mais condições.

Desta forma, podemos entender que os tipos de erros existentes nesse teste dizem respeito aos erros de operador booleano, de variáveis, parênteses, operadores relacionais e erros de expressões aritméticas (MOLINARI, 2013).

2.4 Teste de Fluxo de Dados

O teste de fluxo de dados elege possíveis caminhos de teste que um sistema pode tomar, porém isso depende de como estão localizados as definições e de como são utilizadas as variáveis no programa. As estratégias de fluxo de dados são úteis para selecionar caminhos de teste de um sistema que contenham instruções de laços e IF aninhados (MOLINARI, 2013).

Podemos dizer que uma motivação para a utilização deste teste é a constatação que os testes baseados no fluxo de controle não era suficientes; desta forma foram introduzidos critérios baseados em fluxo de dados, que eram mais adequados a determinados tipos de testes (PRESSMAN; MAXIM, 2016).

REFLITA

"É irreal supor que o teste de fluxo de dados será usado extensivamente ao se testar um grande sistema. No entanto, ele pode ser usado especificamente em áreas do software que sejam suspeitas". (PRESSMAN; MAXIM, 2016, p.437).

2.5 Teste de Laços

Sabemos que os laços existentes em todos os tipos de programas são fundamentais para a execução de determinados tipos de atividades. Os Testes de laços, ou ainda Teste de Ciclos (loop) tem a finalidade de validar as construções destes laços (PRESSMAN; MAXIM, 2016).

Molinari fala da existência de quatro diferentes tipos de laços (MOLINARI, 2013, p. 166):

- Laços simples;

- Laços Aninhados;
- Laços Concatenados;
- Laços não estruturados.

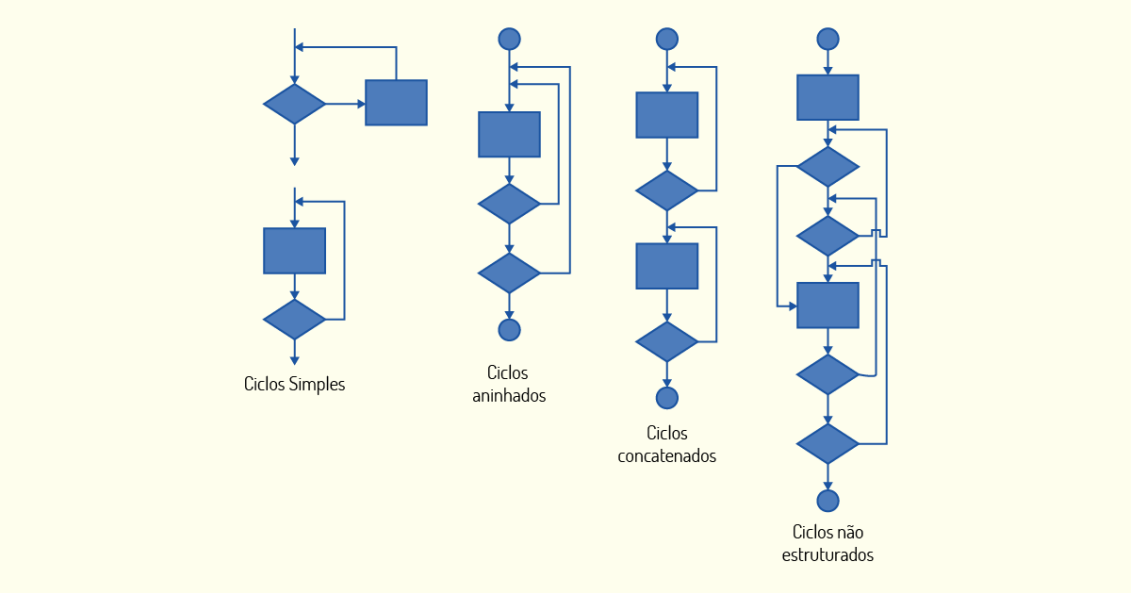


Figura 3.4 - Estrutura de Laços
Fonte: Pressman e Maxim (2016, p. 437).

2.6 Laços Simples

Os testes relacionados aos laços simples podem ser aplicados a estruturas simples de laços de repetição, onde *n* é o número máximo de vezes que o ciclo é executado (PRESSMAN; MAXIM, 2016).

Classes de ciclos:

1. Pular o ciclo inteiramente.
2. Somente uma passagem pelo ciclo.
3. Duas passagens pelo ciclo.
4. *m* passagens através do ciclo onde $m < n$.
5. $n - 1$, n, $n + 1$ passagens através do ciclo.

2.7 Laços aninhados

Se fôssemos aplicar os testes de ciclos simples para os aninhados, o número de testes possíveis cresceria, desta forma, uma abordagem que ajudará a reduzir o número de testes é:

> **1.** Comece pelo ciclo mais interno. Coloque todos os outros ciclos nos seus valores mínimos.
> **2.** Faça os testes de ciclo simples para o ciclo mais interno mantendo, ao mesmo tempo, os ciclos externos em seus parâmetros mínimos de iteração. Acrescente outros testes para valores fora do intervalo ou excluídos.
> **3.** Trabalhe para fora, fazendo testes para o próximo ciclo, mas mantendo todos os outros ciclos externos nos seus valores mínimos e outros ciclos aninhados com valores "típicos".
> **4.** Continue até que todos os ciclos tenham sido testados. (PRESSMAN; MAXIM, 2016, p. 438).

Os laços aninhados exigem um pouco mais de cuidado do que os laços simples, assim as abordagens citadas devem ser analisadas adequadamente e cuidadosamente

2.8 Laços concatenados

Testes concatenados podem ser definidos da seguinte forma:

> Estes podem ser testados usando a abordagem de ciclos simples, se cada um for independente do outro. No entanto, se dois ciclos forem concatenados e a contagem para o ciclo 1 for usada como valor individual para o ciclo 2, então os ciclos não são independentes. Quando os ciclos não são independentes, é recomendada a abordagem aplicada a ciclos aninhados. (PRESSMAN; MAXIM, 2016, p. 438).

FIQUE POR DENTRO

A preocupação de um pesquisador também é a de executar os testes de software, e, claro, encontrar o máximo de erros e problemas possíveis, para que o software produzido seja entrega com uma grande qualidade ao cliente final. Desta forma, é essencial no aprendizado ter uma visão técnica a respeito dos testes apresentados. Assim sendo, apresentamos o link abaixo que contém um artigo sobre uma visão técnica do teste de caixa branca. Vamos conferir? Acesse: <https://www.devmedia.com.br/uma-visao-da-tecnica-de-teste-de-caixa-branca/15610>.

ATIVIDADES

2) Neste teste, o testador tem total acesso à estrutura interna do sistema e assim possibilita que os testes sejam mais precisos. As características expostas se referem a uma técnica importante dos testes, e que está corretamente representada na alternativa:

a) Teste de Caixa Preta.
b) Teste de Caixa Branca.
c) Teste Funcional.
d) Teste Baseado em Grafos.
e) Teste de regressão.

3. Teste de regressão

O teste de regressão nada mais é do que uma técnica que consiste na aplicação de versões mais recentes do software, a fim de verificar que não surgiram novos defeitos em componentes que já foram analisados. Assim, novos componentes ou componentes que foram atualizados são verificados a procura de possíveis problemas com a integração a componentes antigos.

Desta forma, se juntarmos novos componentes ou suas atualizações com os módulos restantes ao nosso software e assim surgirem novos defeitos em partes que não foram alteradas do sistema, dizemos que o sistema regrediu (PRESSMAN; MAXIM, 2016).

De acordo com o ISTBQ (*International Software Testing Qualifications Board*), um teste de regressão é realizado sempre após outros testes, para assegurar que todos os problemas foram resolvidos. Bom, até aqui conseguimos entender o conceito de testes de regressão. Mas como é que surge o teste de regressão? Novos componentes ou atualizações podem fazer com que partes do sistema falhem, causando defeitos não previstos na construção do sistema. Resumidamente, dizemos que “quebramos” um teste anterior com a nova funcionalidade adicionada.

Nesse conceito, podemos entender então que cada módulo que acrescentamos em um software, como parte do teste de integração, faz com que o mesmo mude, gerando novas informações, novos fluxos de dados, novas entradas e por aí vai. E, claro, essas alterações podem causar determinados problemas. O teste de regressão é a reexecução de um mesmo subconjunto de testes que já foram trabalhados anteriormente no software e que, assim, asseguraram que as alterações realizadas não tenham adicionado novos erros ao sistema. (PRESSMAN; MAXIM, 2016).

O teste de regressão também confirma se todas as mudanças ocorridas devido ao teste, ou por qualquer outra razão, não adicionaram um comportamento possivelmente inesperado, indesejado ou possíveis erros que surgiram ao sistema.

Mas como sabemos quando o teste de regressão deve ser utilizado? Usualmente, o teste de regressão é utilizado durante alguns processos do desenvolvimento do software, como por exemplo durante o desenvolvimento iterativo, depois da realização de depuração, quando iniciaremos a integração de módulos, na manutenção de software, entre outros.

E em relação aos softwares construídos dentro do escopo da orientação a objetos? Bom, não deixa de ser diferente em relação ao exposto anteriormente, porém é aplicado, claro, ao escopo deste paradigma; assim, é utilizado quando uma subclasse é desenvolvida, quando uma superclasse é alterada, quando uma classe é estendida, quando a classe é reusada em outro contexto, quando uma correção de falha é realizada, e assim por diante.

O teste de regressão pode tanto ser executado manualmente ou a partir de ferramentas automatizadas. No teste manual será re-executado o conjunto de todos os casos de testes; e os automatizados; e ainda irão permitir que o responsável pelos testes de software capturem determinados casos de testes e possíveis segmentos para re-execução e comparação de informações (MOLINARI, 2013).

O conjunto de testes a ser executado no teste de regressão possui três vertentes diferentes de casos de testes:

> - Uma amostra representativa dos testes que usam todas as funções do software.
> - Testes adicionais que focalizam as funções de software que podem ser afetadas pela alteração.
> - Testes que focalizam os componentes do software que foram alterados. (PRESSMAN; MAXIM; 2016, p. 411).

Devemos compreender também que, a medida que os testes de integração progridem, a quantidade de números de testes de regressão crescem, e muitas vezes, em grande escala. Desta forma, o conjunto de testes que realizam a regressão deve ser implementado de forma a atribuir somente determinados testes que trabalham com uma ou mais classes de erros que podem estar presentes em cada uma das funções principais do sistema (MOLINARI, 2013).

FIQUE POR DENTRO

Teste de regressão são importantes para o desenvolvimento de um software. E claro, há diversos materiais e livros que explanam sobre esse tipo de teste. É importante compreender que todos levam a explicar o funcionamento e possíveis estratégias de aplicação deste tipo de teste. Desta forma, separamos um link interessante para leitura, que fala sobre estratégias e como otimizar este tipo de teste da melhor forma. Vamos conferir? Acesse: <https://www.infoq.com/br/articles/regression-testing-strategies>.

As definições parecem um pouco complicadas, mas, a seguir, iremos apresentar algumas dicas que irão ajudar na identificação do que devemos ou não planejar e testar nesta etapa tão importante.

Bom, uma das dicas é manter mapeada uma rastreabilidade do software, o que nos permitirá saber o que pode ser afetado se aplicados determinadas mudanças no software. Por exemplo, um software de recursos humanos, se alterarmos o cálculo do imposto de renda, podemos verificar a partir da rastreabilidade o que essa alteração irá impactar a geração e emissão dos holerites, devido a possíveis faixas salariais, e no caso, isso deve ser planejado ao realizar as alterações.

Em relação a rotinas críticas, deverão ser identificadas quais rotinas são mais utilizadas pelos clientes, e assim identificar quais são as mais críticas e que não podem apresentar problemas. Devemos tomar cuidado também com testes redundantes e testes que podem falhar em seu objetivo

REFLITA

“O teste de regressão é uma estratégia importante para reduzir “efeitos colaterais”. Execute testes de regressão toda vez que for feita uma alteração grande no software (incluindo a integração de novos componentes)” (PRESSMAN; MAXIM, 2016, p. 411).

ATIVIDADES

3) É uma técnica que consiste na aplicação de versões recentes do software, a fim de verificar a ocorrência de novos defeitos em módulos do software que já foram testados. Esta técnica se refere ao teste de:

a) Recuperação.
b) Regressão.
c) Desempenho.
d) Restauração.
e) Disponibilização.

4. Teste de Carga

A finalidade de um teste de carga é validar e avaliar os limites operacionais do sistema em relação a cargas de trabalho que ele aceita em um ambiente real, para que o mesmo permaneça constante. Em outras palavras, é verificar qual é o limite de dados processados pelo sistema, até que ele não consiga mais processar os dados (PRESSMAN; MAXIM, 2016).

E como esse tipo de teste pode auxiliar as empresas? Imagine você estar acessando um determinado sistema ou site e o mesmo parar de funcionar? É algo que não gostaríamos de passar, não é? Em empresas, pode gerar muito mais do que desgaste, usuário podem perder a produtividade imediatamente; além claro, de ocorrer custos financeiros ou jurídicos.

Desta forma, podemos verificar que os testes de carga auxiliam em prever investimentos desnecessários, realizando claro, planejamentos prévios. Quando bem implementado, o teste reduzirá significativamente o número de falhas (MOLINARI, 2013).

Podemos constatar que através do teste de carga é possível conseguir algumas características, como avaliar a atuação do software que recebe um fluxo de dados intenso, conseguimos avaliar a estabilidade de um sistema durante um período grande de recebimento de carga, e com isso estabelecer uma margem de operação de forma considerável (RIOS; MOREIRA, 2013).

Além disso, podemos verificar se alterações feitas no sistema afetam o seu desempenho e, com isso, podemos identificar pontos críticos. A partir disso, podemos monitorar o desempenho do sistema e assim fornecer métricas que retratam qual é a capacidade de operação do sistema (MOLINARI, 2013).

Temos alguns tipos de testes de carga que podem ser aplicáveis. Dentre eles, temos o constante, que é utilizado quando desejamos aplicar uma carga invariável por um período; o teste por etapa, que é realizado quando queremos aplicar progressivamente um volume de carga e após analisar o desempenho do software; e por fim, o teste baseado em meta, na qual é definido uma meta de desempenho para que o sistema esteja acessível (SOMMERVILLE, 2011).

Os testes de carga também podem ser chamados de teste de performance, e na imagem abaixo podemos verificar quando devemos aplicar, ou realizar estes possíveis testes. A imagem abrange de uma forma simples explicativa de como proceder.

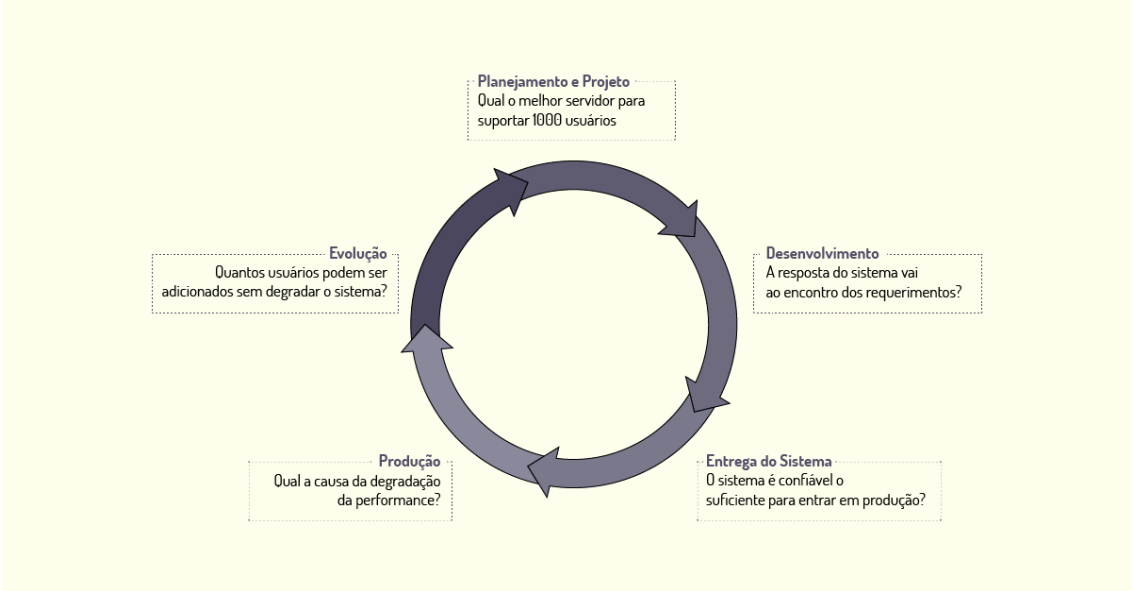


Figura 3.5 - Ciclo de testes relacionados a performance

Fonte: Molinari (2013, p. 184).

Podemos verificar que temos alguns processos a serem seguidos, e esse ciclo se repete até que seja solucionado os problemas encontrados.

4.1 Teste de Estresse

O teste de estresse é como se fosse uma continuação do teste de carga. Assim, também pode ser colocado como um teste de performance que verifica se um software possui robustez, disponibilidade e confiança diante de condições consideradas extremas (RIOS; MOREIRA, 2013).

São consideradas condições de estresse o alto acesso de usuários simultâneos, limitações de recursos computacionais e volume excessivo de dados, o que faz com que haja muita perda de recursos.

Esse tipo de teste é muito vantajoso para o sistema, pois fornece informações que permitem que toda a estrutura de um software seja construído de forma adequada, sendo possível sua evolução a partir da correção dos problemas.

Há alguns exemplos de problemas que ocorrem quando o sistema passa por algum problema relacionado ao estresse, como por exemplo dados perdidos/corrompidos; quando retornado a condições consideradas normais, determinados recursos continuam inaceitáveis e componentes continuam a falhar não atendo alguns comandos, até que o programa trava e para de funcionar (PRESSMAN; MAXIM, 2016).

Há algumas técnicas que devem ser aplicadas ao se trabalhar com testes de estresse, como podemos verificar abaixo:

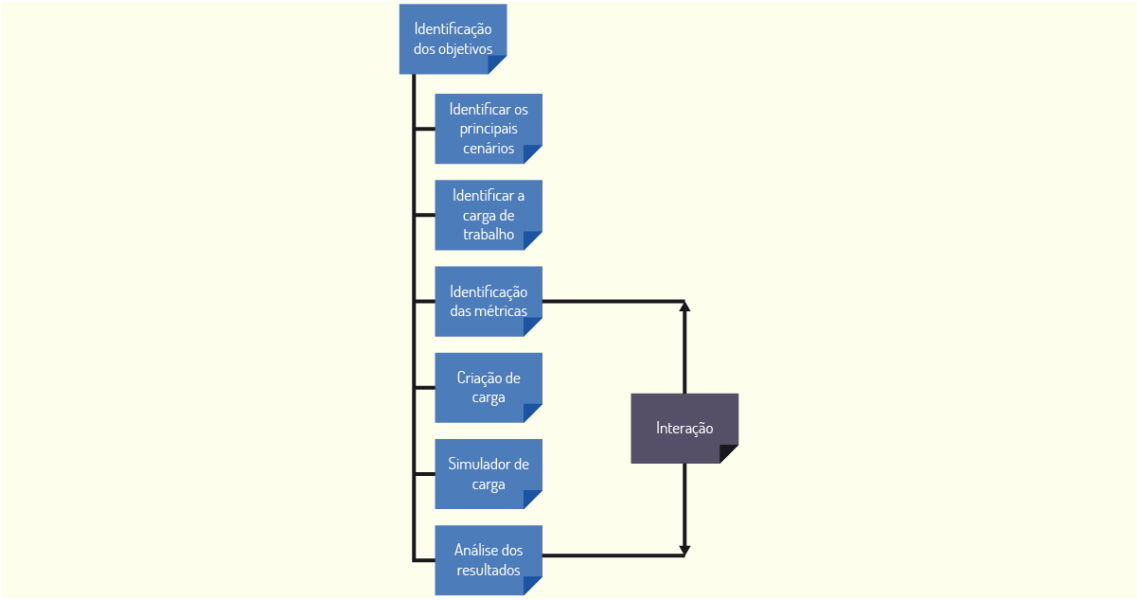


Figura 3.6 - Técnicas

Fonte: Elaborada pelo autor.

Ainda sobre o teste de esforço, temos que existem uma variação chamada de alternância de pico, que tem como objetivo elevar a carga até a capacidade máxima, depois diminuí-la para condições normais, e por fim, aumentá-la novamente. Com isso você poderá verificar se o sistema se restabelece de forma adequada com o sobe e desce de instruções.

Vejamos na figura abaixo uma curva de esforço típico de desenvolvimento de uma aplicação web:

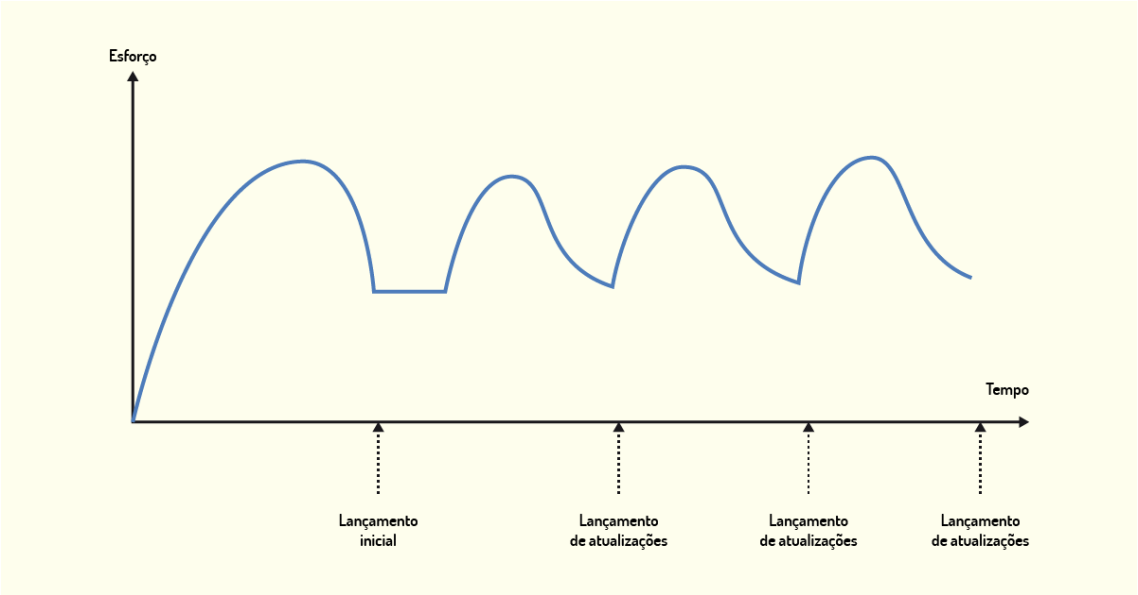


Figura 3.7 - Ciclo de esforço numa aplicação web
Fonte: Molinari (2013, p.178).

Vemos na imagem que dependendo do esforço, o sistema apresenta ciclos que podem causar mais problemas que outros, e que também, dependendo de algumas atualizações realizadas, podem ocorrer problemas de esforço.

4.2 Teste de Usabilidade

O teste de usabilidade tem por objetivo verificar como que o software desenvolvido está sendo compreendido e se está sendo manipulado com facilidade, assim sendo também é analisado o comportamento dos usuários ao se utilizar do software (MOLINARI, 2013).

É um dos testes considerados mais importantes e pode identificar problemas como sistema lento, que averigua quanto um sistema demora para executar uma ação; baixa legibilidade relacionada a parte gráfica do sistema; disposição pouco intuitiva de botões; existência de menus confusos e diversos outros. A existência destes problemas podem ocasionar ao abandono do software por parte do usuário (RIOS; MOREIRA, 2013).

Outra definição que podemos citar é da ISO, com a IEC 9126, que diz que a "usabilidade refere-se a capacidade de uma aplicação ser compreendida, aprendia, utilizada e atrativa para o usuário, em condições de utilização".

Partindo-se do exposto acima, podemos dizer que o teste de usabilidade também está ligado ao teste de aceitação, já que dependendo de como é a reação do usuário ao utilizar o sistema, o mesmo pode não ser aceito.

Bom, então resumidamente o objetivo deste teste é identificar possíveis problemas que prejudicam a usabilidade da ferramenta. Mas como esse teste é feito e quais os principais benefícios?

O teste pode ser realizado por usuários e a dinâmica é simples: deve-se estabelecer possíveis métricas e em suas execuções devem ser analisados o quanto foi gasto para a finalização, quais erros foram cometidos e qual a opinião dos envolvidos. Assim, após definir estas métricas, o teste é aplicado e os usuários devem realizar diversas tarefas (PRESSMAN; MAXIM, 2016).

Como benefício, podemos citar a minimização dos erros e a geração de um sistema com boa funcionalidade, o que pode gerar um produto atrativo em relação ao mercado de software.

E como sei que meu software é bom para o cliente? Bom, como vimos, a facilidade de uso, na memorização de comandos, segurança no uso e outros, fazem refletir na satisfação do usuário, que conseqüentemente se traduz na aceitação pelo usuário.

Isso significa que o sistema torna o usuário mais produtivo, já que o usuário não precisa interromper seu trabalho devido a erros ou problemas como travamento do software.

Como vantagens relacionadas a aplicação do teste de usabilidade, é que o custo do desenvolvimento passa a ser menor; faz com que o tempo de teste diminua no ciclo do desenvolvimento; e claro, que um cliente satisfeito irá querer sempre utilizar um sistema (MOLINARI, 2013).

Um dos modos de problemas serem identificados, é aplicando teste de heurística, que nada mais é do que resolver possíveis problemas a partir de experiências práticas. Dessa forma, pode-se adequar certos pontos dependendo da necessidade de um projeto, como por exemplo, a compreensão por parte do usuário, feedbacks, mensagens de erros claras, avaliações relacionadas à experiência e outros (RIOS; MOREIRA, 2013).

Quando falamos a respeito deste teste, o mínimo que um usuário sempre espera é que todo sistema funcione sem imprevistos e sem complexidade. Todo processo de desenvolvimento do software deve explorar isso, a experiência com o usuário; já que o usuário não "liga" para como um sistema é codificado.

FIQUE POR DENTRO

Usabilidade faz parte do processo de desenvolvimento de um sistema. Desta forma, é essencial que os envolvidos neste processo entendam como é o funcionamento deste teste. Desta forma, planejar como deve ser realizado este tipo de teste se torna importante. Mas como planejar esse tipo de teste? É o que podemos verificar no link abaixo. Separamos um link bem bacana que nos trará um entendimento melhor sobre esse assunto. Vamos ler? Acesse: <http://blog.ti.lemaf.ufla.br/2017/05/11/teste-de-usabilidade-com-usuarios/>

4.3 Teste de Segurança

Este teste tem o intuito de garantir a segurança do software. Mas como é feito isso? A segurança diz respeito ao funcionamento da aplicação, que seja realizado como fora especificado. Desta forma, é verificado se o software continua funcionando mesmo passando por diversas tentativas ilegais de acesso. Desta forma, são testados todos os mecanismos que protegem o sistema (PRESSMAN; MAXIM, 2016).

Imagine você um site ou um software que manipule dados importantes e confidenciais, é de extrema importância que haja uma proteção recorrente. É mais comum do que imaginamos ocorrer situações como esta, ataques ocorrem a todo momento e isso deve ser trabalho do sistema.

Considerando os conceitos apresentados, o teste de segurança, ou ainda, security testing, visa verificar e determinar qual tipo de precaução deverá ser tomado para determinados tipos de ataques, que podem ser do tipo negação de serviço e diversos outros.

A validação de segurança de um sistema pode ser realizada localizando as falhas inseridas durante a fase de desenvolvimento do projeto, como por exemplo erros humanos no código. Esse tipo de análise pode ser realizada por inspeção de código. Outro tipo de validação, é a verificação da implementação durante a sua execução, na qual entradas de dados são fornecidas para verificar como reagem os mecanismos de segurança (RIOS; MOREIRA, 2013).

Existem diversos tipos de proteção que podem ser implementadas dependendo do tipo de aplicação desenvolvida. Podemos citar o firewall, que realiza filtros e bloqueios contra ataques; mecanismos de autenticação, que validam a identidade de clientes; criptografia, que codifica e protege dados; e diversos outros (PRESSMAN; MAXIM, 2016).

Justamente em cima destes mecanismos que os testes de segurança deverão ser projetados, assim, com o esforço aplicado poderão ser descobertos falhas. Para a aplicação de um projeto de teses, é necessário o conhecimento do funcionamento de cada elemento de segurança.

Teste de segurança ajudam a melhorar significativamente coisas como:

- Ponto de referência para as correções de segurança;
- Definição clara das atividades indicando falhas de segurança;
- Registro de status das atividades de segurança no que tange aos requerimentos para segurança da organização;
- Ajuda na análise de custo-benefício, ou seja, quanto custa a perda de algo? (MOLINARI, 2013, p.190).

Concluímos assim, que o teste de segurança visa aplicar penetrações no software, para que assim o sistema possa apresentar problemas, e posteriormente, serem corrigidos.

REFLITA

"O teste de regressão é uma estratégia importante para reduzir "efeitos colaterais". Execute testes de regressão toda vez que for feita uma alteração grande no software (incluindo a integração de novos componentes)" (PRESSMAN; MAXIM, 2016, p. 411).

ATIVIDADES

4) É um teste que verifica se o sistema possui robustez, disponibilidade e confiança diante de condições extremas. A alternativa que contempla corretamente tipo de teste definido é:

a) Regressão.

b) Estresse.

c) Carga.

d) Segurança.

e) Desempenho.

INDICAÇÕES DE LEITURA

Nome do livro: Teste de Software: Produzindo Sistemas Melhores e mais Confiáveis.

Editora: Érica

Autor: Leonardo Molinari

ISBN: 9788571949591

Comentário: Temos que a qualidade de software tem sido uma das principais agregações ao longo do tempo. Desta forma, como devemos desenvolver sistemas usando essa ciência chamada de testes de software? Para obter esta resposta, devemos ler o livro e verificar que sua divisão em três partes: visão macro da qualidade de software e seus principais elementos, testes de software em detalhes e principais ou grandes áreas de teste. Além disso, o livro apresenta algumas ferramentas importantes que auxiliam no caso de testes.

INDICAÇÕES DE FILME

Nome do filme: JOBS

Gênero: Drama

Ano: 2013

Elenco principal: Ashton Kutcher, Josh Gad, Dermot Mulroney, Matthew Modine, Lukas Haas

Comentário: O filme conta a história de Steve Jobs, que em 1976 abandonou a faculdade e junto com seu amigo Steve Wozniak, iniciaram uma revolução com a invenção do Apple 1, o primeiro computador pessoal. O filme conta que o computador foi construído na garagem dos pais de Jobs e que a formação da empresa mudou o mundo completamente.

UNIDADE IV

Frameworks de Testes

Rafael Maltempe da Vanso

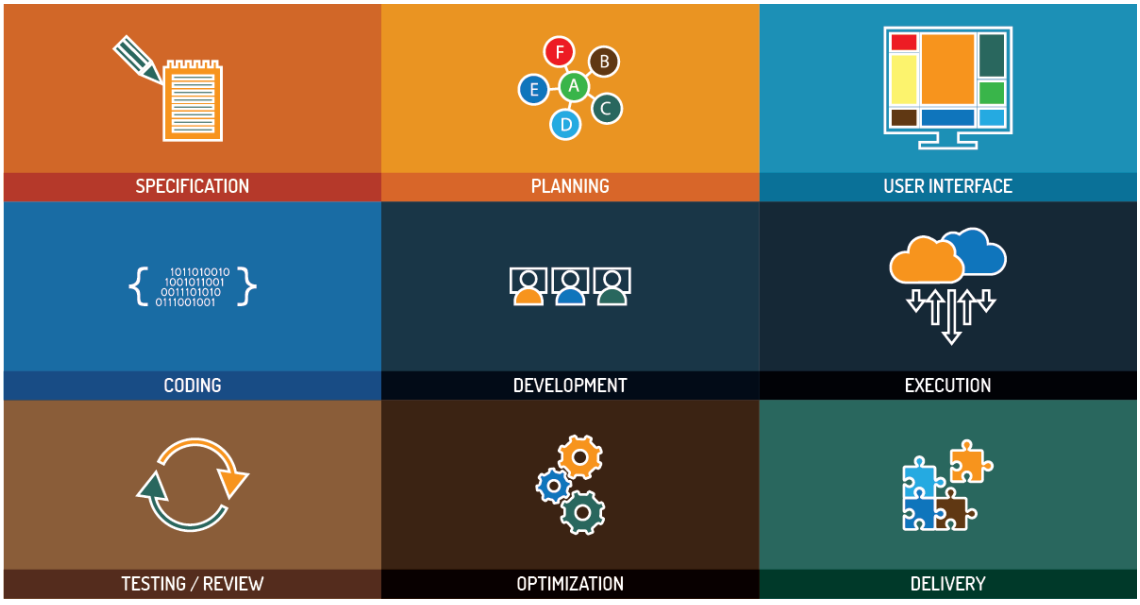
INTRODUÇÃO

Neste capítulo, iremos trabalhar e nos aprofundar mais em ferramentas que auxiliam os testes de software, chamados de Frameworks. Eles possuem características específicas, desta forma, iremos falar um pouco sobre práticas selecionadas, inspeções, walkthrough e review.

Iremos entender, também, os motivos pelos quais devemos automatizar, que critérios selecionar e quando automatizar. Mas claro, antes de tudo, iremos entender os conceitos fundamentais do que é uma automação de testes.

Como terceiro item, iremos falar sobre as ferramentas que possibilitam realizar essa automatização, assim, iremos entender quais os tipos de ferramentas existentes, o que são mocks e como podemos utilizá-los para os testes e alguns cuidados que devemos entender.

Por fim, mostraremos alguns exemplos de ferramentas que se utilizam de mocks, em diversas linguagens que são mais utilizadas, citando a ferramenta e o que devemos realizar para ter acesso a ela. É um capítulo importante e que deve ser entendido de forma adequada, para que, após ao término, não fique dúvidas futuras. Vamos lá?



Fonte: Mnsanthoshkumar / 123RF.

1. FRAMEWORK DE TESTES

Por diversos anos, os testes de software eram feitos por programadores e analistas que realizavam simulações de situações, a fim de buscar erros nos programas. Os dados e simulações eram gerados manualmente ou, ainda, com a utilização de softwares destinados a esse fim. Existiam poucas ferramentas que realizam esse tipo de serviço e que eram difíceis de manipular.

Figura 4.1 - Framework
Fonte: Dizanna / 123RF.

O processo de teste de software cada vez foi se tornando mais complexo de ser executado, assim o seu custo, o tempo requerido de testes e a piora na qualidade aumentaram (RIOS; MOREIRA, 2013).

Desta forma, o surgimento de novas ferramentas foi necessário para apoiar e melhorar o processo do teste de software. A automação existe para que seja reduzido ao máximo o envolvimento humano com a máquina, isto é, para que seja reduzido a interação em atividades repetitivas do testador ou desenvolvedor na procura de erros.

A automação de testes pode adicionar muita complexidade na realização dos testes; mas, de uma forma ou de outra, trás muitos benefícios e tornam-se indispensáveis em determinados ambientes de testes.

Temos que o principal objetivo dos testes automatizados é serem utilizados em testes de regressão, já que o testes são re-executados, a fim de verificar a correção ou adição de novas funcionalidades no sistema.

REFLITA

Não adianta automatizar um processo de testes desorganizado e sem uma base metodológica sólida. Em casos pontuais, ou em projetos específicos, tais como o teste de estresse de uma aplicação Web, é quase impossível realizar o teste sem uma ferramenta de apoio. Neste caso, a ferramenta de apoio é usada em uma parte de um projeto específico, mas não na automatização do processo como um todo
Fonte: Adaptado de Rios e Moreira (2013).

Desta forma, a partir de agora, iremos falar um pouco mais sobre frameworks e suas técnicas de revisões de software, vamos lá?

1.1 Review

Vimos que a melhor estratégia para atingir os melhores índices de qualidade no processo de teste é adotar algumas técnicas de verificação para localizar e corrigir defeitos em documentos, planos e requisitos, e especificações e códigos, como complemento aos próprios testes dos componentes codificados (RIOS; MOREIRA, 2013).

Podemos iniciar falando das técnicas de Walkthrough, Inspeção ou revisões técnicas e Pair Programming.

1.2 Walkthrough

Também conhecido como Travessia, essa técnica realiza um conjunto de revisões, que têm como finalidade sempre melhorar a qualidade no desenvolvimento de softwares.

Este tipo de técnica possui algumas características importantes, que podemos verificar a seguir (RIOS; MOREIRA, 2013):

- São realizadas avaliações informais para avaliação do produto.
- Pouca ou nenhuma preparação é usualmente requerida.
- Não existem restrições a discussões sobre a validade do defeito e/ou a solução para correção.

1.3 Revisões Técnicas (Inspeções)

São práticas em que um determinado grupo de pessoas se dedicam a descobrir defeitos em produtos, no nosso caso, defeitos em softwares. Assim, as revisões em software visam melhorar a qualidade dos produtos desenvolvidos para reduzir prazos e custos de desenvolvimento.

Figura 4.2 - Inspeções de software
Fonte: Wrightstudio / 123RF.

Realizar uma inspeção (revisão) em um software é de extrema importância e pode ser atribuído a qualquer software e seus artefatos, como podemos verificar na figura a seguir:

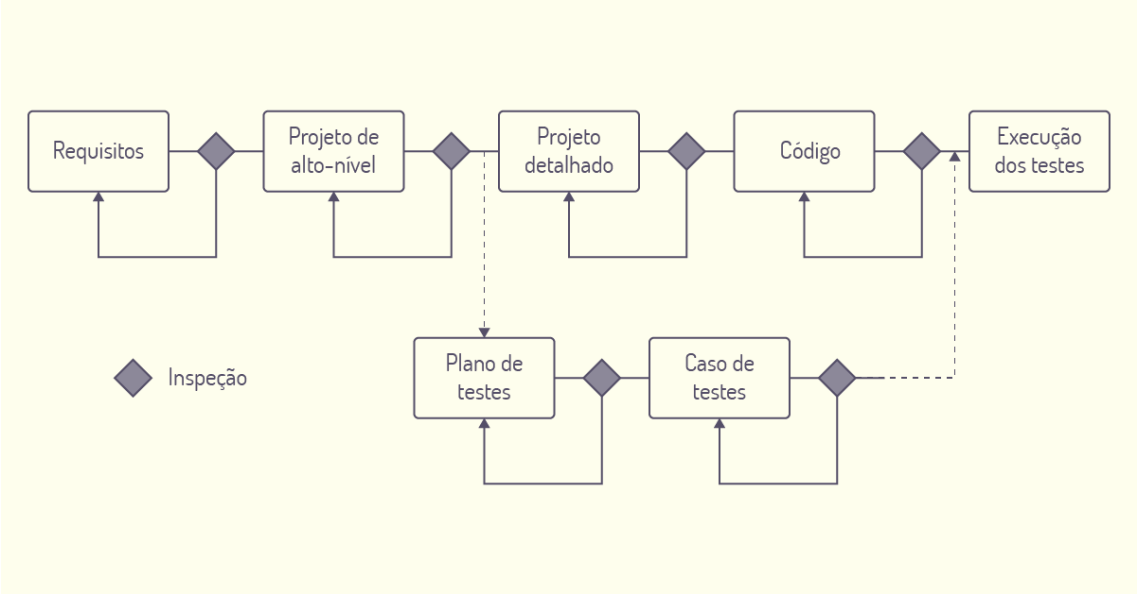


Figura 4.3 - Inspeções de Software nos Diferentes Artefatos
Fonte: Adaptado de Ackerman et al. (1989, p. 33).

Segundo Rios e Moreira (2013), o processo de revisão técnica possui algumas características, as quais podemos evidenciar a seguir:

- Existência de um coordenador para uma determinada revisão.
- Necessidade de “check-lists” para cada tipo de linguagem.
- Muitos defeitos são identificados com a revisão técnica.
- Pessoas que participam de uma revisão recebem documentos para revisão e seus check-lists correspondentes para realização do trabalho de revisão, que pode demorar entre 1 e 4 horas de preparação.
- Não devem ser revisados mais de 20 páginas ou 250 linhas de código num mesmo exercício.
- E outros.

A partir destas características, podemos descrever o fluxo do processo de revisão (inspeção), na qual, primeiro, temos o planejamento da revisão, em que nomeamos o coordenador, realizamos o agendamento dos trabalhos, formação da equipe de revisores e distribuição do produto e do “check-list”.

O próximo fluxo diz respeito ao nivelamento das informações, em que acontece a apresentação dos objetivos do produto e do “check-list” para os revisores. Dando sequência, temos a análise dos produtos a serem revisados, em que ocorre o estudo do produto, apontamento dos potenciais defeitos e não conformidades.

Continuando, temos a reunião de revisão, em que temos a identificação, classificação da severidade e registro dos defeitos. Após, ocorre a correção dos defeitos, em que há o planejamento e execução das correções dos defeitos. E para encerrar, o acompanhamento das medições, em que ocorre o acompanhamento das correções e coleta das medições de defeitos encontrados.

Figura 4.4 - O fluxo do processo de revisão (inspeção)

Fonte: Tarapong / 123RF.

São muitas as considerações que devem ser levadas em conta para a adoção do processo de revisão técnica. Podemos, ainda, citar a respeito das metodologias de desenvolvimento, que devem estabelecer etapas e atividades a serem cumpridas. Outra consideração é a respeito de padrões, que devem definir um formato e conteúdo para as revisões, com padrões para requisitos, programação, documentação e outros (RIOS; MOREIRA, 2013).

Continuando, deverão ser elaborados procedimentos para a realização das revisões, prevendo, assim, critérios para definição de produtos a serem revisados, medições de qualidade e desempenho etc.

Ainda temos outras considerações, como o apoio gerencial, que gerencia determinadas atividades, e que se não realizada, fica fadado ao fracasso. Para finalizar essas considerações, temos as estratégias de introdução, que devem ser críticas em suas análises; os check-lists, que devem ser tarefas para preparação de técnicas; e, por fim, medições, que são importantes para a gestão da qualidade e do processo.

1.4 Benefícios da Aplicação de Inspeções de Software

Alguns benefícios são produzidos ao se inspecionar um software, podemos citar a produtividade, tempo, custo e esforço. Em relação ao esforço e custo, podemos citar o quanto o retrabalho atrapalha na implantação de projetos, o que pode chegar numa proporção de variação entre 40% e 50% do retrabalho. A implantação da inspeção de software produz efeitos positivos nas fases iniciais do desenvolvimento, facilitando, assim, a detecção e correção de defeitos sem grandes esforços.

Em relação à produtividade e tempo, respectivamente, com a aplicação das inspeções, temos um aumento da produtividade e uma redução do tempo de desenvolvimento.

Para concluirmos sobre a aplicação de inspeções, percebemos que a intenção sempre é melhorar a qualidade por meio da análise dos artefatos, verificando, analisando e excluindo erros e defeitos encontrados

1.5 Pair Programming

Outra técnica que podemos falar é a técnica de programação em par, que tem como intuito o compartilhamento da codificação entre dois programadores, assim, os dois poderão trabalhar ao mesmo tempo, sendo que um trabalha codificando e o outro observando se os padrões do projeto estão sendo cumpridos. Podemos dizer que os programadores compartilham de seus conhecimentos para melhor implementar.

Figura 4.5 - Programação em Par
Fonte: Rasmus Ursem / 123RF.

Podemos citar como vantagem da programação em par, o rodízio de pessoas, código coletivo, a utilização de padrões etc. É observado que há um aumento na atenção e análise da codificação; há um maior rigor na análise dos objetivos, e a quantidade de erros diminui.

FIQUE POR DENTRO

Vimos que a inspeção é de extrema importância, mas temos que entender que há uma diferença entre inspecionar um software e testar um software. Entender essa diferença implica em conhecer como é o processo de inspeção. Nesse sentido, separamos um artigo interessante que demonstra a diferença desses dois conceitos. O link a seguir nos mostra um artigo. Vamos obter um pouco mais de conhecimento? Disponível em: <http://rederequisitos.com.br/qual-diferenca-entre-inspecao-e-teste-de-software-voce-conhece-o-processo-de-inspecao/>.

ATIVIDADES

1) Também conhecido como Travessia, esta é uma técnica que realiza um conjunto de revisões com a finalidade de melhorar a qualidade no desenvolvimento de softwares. Estamos nos referindo a que tipo de técnica:
   a) Review
   b) Walkthrough.
   c) Revisões técnicas.
   d) Mocks
   e) Frameworks

2. FERRAMENTAS E AUTOMAÇÃO DE TESTES

Antes de iniciarmos a explanação sobre as ferramentas de automação de testes, vamos entender um pouco mais sobre esse conceito.

Bom, a automação de testes nada mais é do que a utilização de softwares que controlam a execução de testes, aplicando, assim, possíveis estratégias.

A automação de testes, porém, pode tanto contribuir como também ter efeito reverso e, desta forma, causar determinados problemas no software. Conforme os anos vão passando, a automação vem crescendo e se tornando cada vez mais importante na área de testes, o que trás uma série de pontos positivos, como a diminuição de custos e diminuição do tempo em contato com o desenvolvedor, reduzindo assim o tempo de mão de obra; e, ainda, há um aumento na velocidade de se realizar os testes de softwares (MOLINARI, 2013).

No entanto, há de se ter um cuidado dobrado com a automação de testes, pois este deve estar bem estruturado antes de ser aplicado e, claro, ter uma equipe preparada, principalmente relacionada a conhecimentos em testes de software, treinamentos, conscientização e outros.

Devemos entender que com as características expostas, a automação de testes também agrega valores à marca e a todos os envolvidos no projeto. Vantagens como estas podem fazer uma grande diferença e influenciar o nível de uma empresa no mercado de software.

Mais o que devemos levar em conta para que haja a automatização nos testes de software? É o que iremos estudar um pouco mais adiante.

2.1 Quando Automatizar?

Uma das principais características da não automatização dos testes é o custo que se tem ao se implantar tais tipos de testes. Desta forma, muitos gerentes de projeto negligenciam esse processo, ocasionando em uma qualidade inferior do projeto ao ser finalizado.

Reconhecer quando aplicar a automatização de testes é fundamental; assim, quando o gerente perceber que o processo normal do teste está ineficiente ou mais caro, é hora de se pensar na automatização e evitar possíveis colapsos, o que pode prejudicar, e muito, o decorrer do projeto.

Percebemos, no decorrer de nossa explicação, que a automação de testes está, de uma forma ou de outra, relacionada à qualidade do software. Desta forma, para se automatizar, precisamos verificar diversas características e, principalmente, se acarretará em um ganho de tempo, custo e qualidade.

Verificado isso, devemos entender sobre o comportamento do sistema testado, reutilização de teste e quanto de tempo necessitamos para realizar a automação. A partir disso, poderemos começar a realizar a automação necessária.

Depois, devemos entender o que automatizar e o por quê automatizar. Temos diversas ferramentas que auxiliam nesse processo, desde a revisão de documentos até a execução de diversos testes.

Por exemplo, em se tratando de testes de regressão, sabemos, pela definição, que são testes já executados e, de uma forma ou de outra, são realizados novamente devido à alguma atualização ou inclusão de um nova funcionalidade, a fim de verificar possíveis erros a partir disso. Assim sendo, percebemos que testes de regressão são maçantes justamente pelo fato de ser repetido. Tudo isso torna esse tipo de teste um grande candidato a ser automatizado. Assim, estaremos evitando um excesso de trabalho dos profissionais da área de teste e conseguindo um feedback aceitável de forma mais frequente (MOLINARI, 2013).

Uma ferramenta de automação de testes possui várias funcionalidades, desde a execução repetida dos testes até a medição de performance de uma determinada aplicação. Por exemplo, se desejarmos validar a performance de uma parte do sistema, podemos aplicar um teste automatizado que realize determinada característica do sistema, como um cadastro de fornecedores. A atividade será executada diversas vezes, inundando o sistema com uma grande inserção de dados e verificando como é sua performance.

Devemos sempre considerar para quais casos cabe a automação dos testes, pois é um investimento que demora um pouco a ter retorno, dizemos que de médio a longo prazo. Também devemos considerar que mesmo se utilizando da automação dos testes, ainda devemos trabalhar com testes manuais para determinadas atividades.

Podemos, ainda, citar alguns testes que podem ser automatizados, como o já citado teste de regressão, para tarefas repetitivas, tarefas que exigem cálculos matemáticos, funcionalidades consideradas críticas em software; teste de performance; teste unitário; e teste de integração. Claro, podemos aplicar em diversos outros testes a automação, porém estes são os principais que podemos citar.

2.2 Projeto de Automação de Testes

Um projeto de implementação de automação de testes deve sempre ser cercado de diversos cuidados e suportar características, como podemos verificar na figura a seguir:

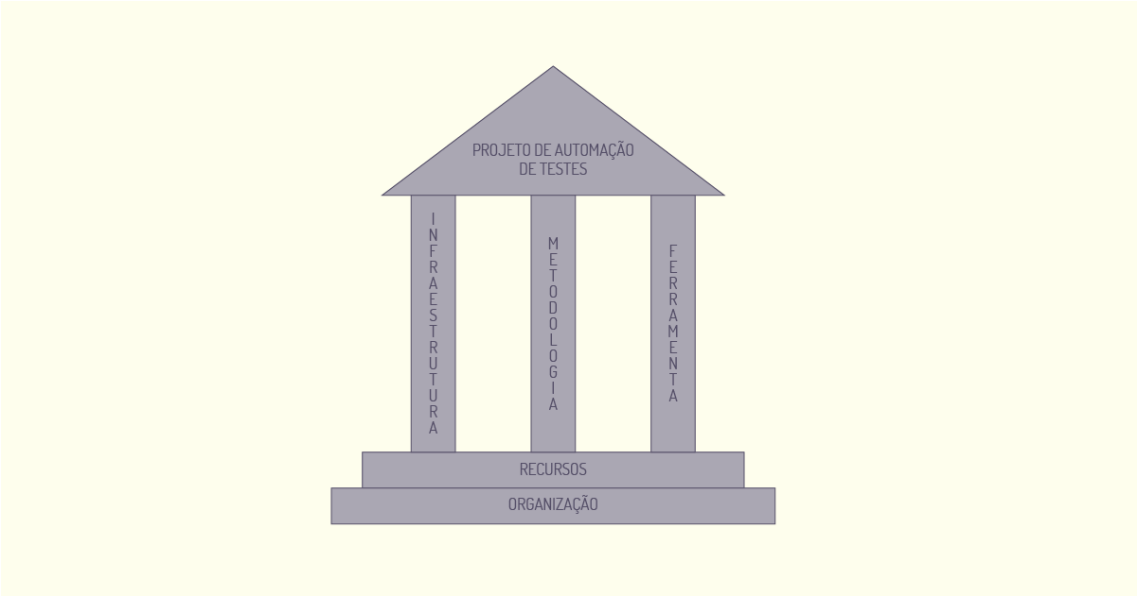


Figura 4.6 - Fundações e Pilares
Fonte: Rios e Moreira (2013, p. 156).

Em se tratando das fundações, o responsável pela implementação dos testes automatizados deve se organizar e ter em mãos os recursos necessários para isso.

Como Pilares, temos as ferramentas, metodologias e infraestrutura. Como ferramentas, o responsável pela implementação dos testes deve realizar a seleção da ferramenta certa, adequada à tecnologia utilizada e que possa se integrar com as metodologias de desenvolvimento de testes. Já sobre a metodologia, o responsável deve se utilizar de metodologias e testes consolidados, que possam se integrar com a ferramenta escolhida. E, por fim, infraestrutura, na qual deve ter disponível maquinas, recursos, outras funcionalidades dedicados ao projeto de automação de testes (RIOS; MOREIRA, 2013).

2.3 Considerações para Automação de Testes

Ainda, antes de iniciar ou considerar uma automação em testes, temos que verificar determinadas características. Dentre as considerações, iremos tratar das que compõem esse processo (RIOS; MOREIRA, 2013).

Figura 4.7 - Características da automação de testes
Fonte: Sergei Polivanov / 123RF.

A seguir, veremos as especificações de cada uma dessas características:

- **Escopo:** É necessário identificar quais tipos de testes devem realmente ser automatizados. Para isso, é necessário realizar um estudo para identificar quais casos requerem está automação. Podemos citar como critérios para identificação casos no quais a realização dos testes necessitam de uma grande quantidade de pessoas e grande esforço, e casos nos quais é praticamente inviável realizar os testes por meio dos esforços normais.

- **Prazo de preparação:** Considerar o tempo de preparação de scripts para serem executados os testes. Assim, temos que levar em conta o tempo de experiência dos testadores na preparação de scripts, deve-se treinar novos testadores antes de continuar o prazo de preparação nos primeiros sistemas; não escolher sistemas que mudam com grande frequência, e considerar, nos custos, a manutenção dos scripts.
- **Estratégias:** Deve-se criar estratégias que garantam o sucesso do projeto de automação de testes, como realizar uma prova de conceitos e ter controle sobre custos, prazos e benefícios; criar checklists para apoiar no acompanhamento das tarefas; desenvolver um estudo de caso baseado em todos os registros e observações feitas; não deve ser esquecido o custo de perda de negócio devido à má qualidade dos testes que possam resultar na indisponibilidade do sistema ou pela insatisfação dos usuários; e, claro, verificar estratégias quanto à organização do grupo do projeto.
- **Melhoria do Processo de Testes:** É recomendável que as organizações considerem a avaliação do seu processo de testes por meio do Modelo de Maturidade de Testes e com base em seus resultados. Com isso, a documentação dos sistemas deve ser melhorada; as abordagens dos testes deve ser explicitada; especificar os dados a serem usados para os testes ou os guias para disponibilizá-los; e, especificar o desenho dos testes e os resultados esperados.
- **Prova de Conceito:** Uma prova de conceito visa a validar o uso da ferramenta na realização de uma determinada tarefa, procurando focar nas suas principais características, qualidades e restrições.
- **Tratamento como projeto:** Deve ser tratado da mesma forma e com os mesmos cuidados de um projeto de desenvolvimento de sistemas. No caso do projeto de automação, temos etapas, como seleção da ferramenta, desenvolvimento de scripts, execução e conclusão.

- **Seleção da Ferramenta:** Um bom teste requer a implementação de uma ferramenta para apoiá-lo. Para isso, a empresa precisa ter um processo de seleção de uma ferramenta, que requer um grupo multidisciplinar composta por um coordenador e representantes das áreas de teste, suporte e outros; preparação de check-lists; definição das informações a serem prestadas pelos fornecedores; planejamento do trabalho; instalação do software e documentação de todo o processo de seleção.

REFLITA

"Ter sempre em mente que automatizar custa caro, sendo recomendado à preparação de um "business case" para apoiar a tomada de decisão sobre usar ou não e da estratégia para introdução".

Fonte: Rios e Moreira (2013, p. 160).

Antes de prosseguir com a automação dos testes, deve, ainda, verificar e fazer uma avaliação dos seus atuais processos. Se estes processos não incorporam melhorias práticas, provavelmente a automação de testes não será a solução possivelmente adequada para prover os seus usuários com os softwares com boa qualidade.

FIQUE POR DENTRO

Como estudado, a automação de testes possui grandes desafios e o principal deles, podemos dizer, é em cima de iniciantes. Como a automação de testes é uma área que vem crescendo muito, sempre ter pessoas qualificadas para que realizem trabalhos adequados é essencial. Desta forma, separamos um artigo que diz a respeito de dicas para iniciantes, e também desafios enfrentados na automação de testes. Vamos adquirir novos conhecimentos:

Disponíveis em: <https://blog.cedrotech.com/o-desafio-da-automacao-de-testes-dicas-para-iniciantes/>.

<https://www.monitoratec.com.br/blog/5-desafios-na-automacao-de-teste/>.

ATIVIDADES

2) A automação de testes é parte importante quando se trata de teste de software. Utilizar-se de todas as técnicas sempre trará benefícios ao produto final. Sobre automação de testes, a alternativa que indica corretamente a sua definição é:

a) É a execução de testes manuais em softwares a procura de determinados problemas.
b) É a utilização de softwares que controlam a execução de testes, aplicando, assim, possíveis estratégias.
c) São estratégias para testes unitários aplicados por testadores diferentes.
d) São testes que não podem ser realizados por qualquer pessoa integrante da equipe, mas somente por testadores.
e) São estratégias que são pensadas antes da construção do software e que não podem ser alteradas no decorrer do projeto.

3. FERRAMENTAS

Ferramentas são importantes para a automação de testes, e no mercado existem para diversos tipos de testes. Geralmente, essas ferramentas são utilizadas pelos profissionais a partir de algumas características importantes, como suas funções, especialização e como se comportam em testes caixa branca versus caixa preta.

Nesse sentido, iremos explanar um pouco mais sobre essas características a partir de agora. Sobre as funções, como o próprio nome diz, as ferramentas são selecionadas a partir de características a respeito das funções que executam.

Desta forma, temos ferramentas para determinadas características, por exemplo, ferramentas de aquisição de dados, que adquirem dados por meio de extração, conversão dos dados utilizados nas atividades de teste. Outro tipo são as ferramentas de medição estática que analisam informações existentes no código fonte, que possibilitará a análise do fluxo lógico e de dados. Esta ainda trabalha com as métricas relacionadas à qualidade.

Temos, também, ferramentas de medição dinâmica, que trabalham na execução do código, assim, analisam toda a estrutura em tempo de execução, detectando possíveis erros. Ainda temos ferramentas que chamamos de simuladores ou geradores, que são destinados à execução de atividades específicas; e, ainda, ferramentas de gerenciamento de testes que trabalham como planejamento das ações, como implementação, execução e outros.

Outro tipo de ferramentas existentes são aqueles que trabalham com a análise do funcionamento de testes de caixa branca versus caixa preta. As de caixa branca sempre se utilizam do conhecimento relacionado em cima do código fonte para a possibilidade de implementar e executar os testes. Já as de caixa preta, estão relacionados a parte funcional do sistema.

Mencionamos, também, sobre as ferramentas baseadas na especialização, ou seja, baseadas em atividades que detêm maior habilidade de verificação. Assim, temos as ferramentas de registro/produção, na qual combinam tanto a aquisição dos dados quanto a medição dinâmica; as ferramentas de medição de qualidade, que analisam características que no final sugerem uma possível confiabilidade, capacidade de manutenção e outros.

Temos, também, as ferramentas de monitoramento de cobertura, que está relacionado ao quanto o teste abrange em relação ao código, verificando, assim, toda a sua cobertura para o que foi especificado. Já as ferramentas geradores de casos de testes, como o próprio nome diz, automatizam a geração de dados de testes, a fim de introduzir diversos tipos de entradas para causar determinados erros no sistema.

Dando sequência, temos também as ferramentas de comparação, cuja finalidade é comparar diferenças existentes entre os resultados do teste e resultados de referência.

3.1 Utilização de Mocks para testes

Entendemos como Mock tudo aquilo utilizado como cópia de algo, de um objeto que tem o mesmo comportamento que o objeto original, relacionado à unidade em um teste.

Para melhor entendermos, vamos a um exemplo prático, imagine um banco de dados. Para realizarmos o teste neste banco de dados, devemos criar um objeto que seja cópia dele, assumindo todas as funcionalidades do original. Desta forma, podemos realizar testes neste banco de dados relacionado à conexão e outros tipos, tomando-se que o mesmo irá funcionar e retornar valores como o previsto em sua implementação.

REFLITA

Às vezes, o objeto que você está testando tem dependências em outros objetos que podem não ter sido escritos ou que atrasam o processo de teste quando são usados. Desta forma, existem Mock objects que podem ser usados para simular operações anormais ou eventos raros (SOMMERVILLE, 2011).

Podemos dizer, então, em linguagem popular, que Mocks são fakes por simularem algo como se fosse o real. No entanto, um Mock sempre irá decidir se um teste obteve sucesso ou falhou.

Mocks são muito utilizados em testes de interação, ou seja, testes que trabalham com a análise de uma determinada ação. Desta forma, o mock tem o intuito de verificar se na ação ocorreu tudo certo, se foi executada sem nenhum problema, em vez de verificar se o mesmo retornou algo.

Mocks também pode ser conhecidos como Objetos Mock ou também objetos simulados. Então, se você, caro(a) aluno(a), estiver lendo algum material a respeito, fique atento.

Mas para que se utilizar de objetos Mocks? Bom, existem algumas razões para isso. Ele não é somente utilizado em testes de interação, mas com parcelas significativas em outros testes também. Por exemplo, em testes unitários, na qual podem simular como objetos irão se comportar; desta forma, são muito úteis para quando objetos são difíceis de se criar.

O uso de objetos mock também podem causar determinadas limitações, como negar benefícios que testes unitários oferecem. Assim, o excesso do uso desses objetos pode resultar em um crescimento de possíveis alterações que, necessariamente, precisam ser feitas no processo de evolução do sistema.

Vimos, também, que o mock simula o comportamento de um objeto e, como limitação, seu comportamento pode ser afetado se o objeto foi realizado por outro desenvolvedor, ou caso não tenho sido realizado; e se esse comportamento não for modelado de forma correta, torna-se inútil o teste unitário.

Ainda sobre as limitações, muitos mocks executados podem acarretar no ocultamento de problemas entre os objetos; desta forma, os objetos que foram mockados podem dar determinado resultados diferentes quando estes estiverem em funcionamento real.

Dessa forma, podemos concluir que não precisamos criar mocks em todos os casos e, assim, torna-se importante analisar cada caso para procurar a melhor solução.

FIQUE POR DENTRO

Mockar ou não mockar? Eis a questão. Bom, agora que entendemos um pouco sobre esse novo vocabulário e sua funcionalidade, e ao decidir se devemos ou não mockar, também devemos conhecer algumas ferramentas que nos auxiliam. O link a seguir nos mostra um pouco mais sobre um framework chamado mokito. Vamos ver como é o funcionamento deste framework?

Disponível em: <http://blog.caelum.com.br/facilitando-seus-testes-de-unidade-no-java-um-pouco-de-mockito/>.

ATIVIDADES

3) Existem diversas ferramentas que auxiliam a automação de testes no mercado, e cada uma com características é destinadas a alguns testes específicos. Sobre isso, a alternativa que menciona, ao menos, três tipos existentes de ferramentas é:
   a) Aquisição de testes, medição estática e geradores.
   b) Aquisição de dados, medição dinâmica e simuladores.
   c) Medição de qualidade, monitoramento e medição anormal.
   d) Comparação, especialização e gradual.
   e) Aquisição de métodos qualitativos, monitoramento e gerenciamento.

4. EXEMPLOS DE FERRAMENTAS MOCK

Agora que entendemos um pouco mais sobre os mocks, podemos começar a falar mais sobre as ferramentas existentes no mercado. Relembrando um pouco, os mocks são utilizados para simular objetos reais para verificação de problemas mediante testes.

Existem diversas razões para que sejam utilizadas essas ferramentas, já que podem simular comportamento de objetos reais que podem ser muito complexos; um exemplo disso é uma chamada remota, que pode ser simulada adequadamente por meio de objetos mock.

Entendemos que testes com muito mocks ocultam problemas entre objetos e que necessitam da realização de testes de integração. Nesse sentido, iremos citar, no quadro a seguir, algumas ferramentas que auxiliam na criação de mocks.

Vamos, então, a leitura de algumas ferramentas

Ferramenta	Linguagem	Site
Mocha	Ruby	<https://rubygems.org/gems/mocha/versions/1.1.0>
RSpec	Ruby	<http://rspec.rubyforge.org/>
FlexMock	Ruby	<https://pypi.org/project/flexmock/>
jMock	Java	<http://www.jmock.org/>
EasyMock	Java	<http://www.easymock.org/>
rMock	Java	<http://rmock.sourceforge.net/>
SevenMock	Java	<http://seven-mock.sourceforge.net/>
Mockito	Java	<http://code.google.com/p/mockito/>
RhinoMocks	.NET	<http://www.ayende.com/projects/rhino-mocks.asp>
NMock	.NET	<http://nmock.org/>
Moq	.NET	<http://code.google.com/p/moq/>
MockPP	C++	<http://mockpp.sf.net/>
Amop	C++	<http://code.google.com/p/amop>

MockitNow	C++	<http://code.google.com/p/mockitnow>
SimpleTest	PHP	<http://simpletest.sourceforge.net/>

Quadro 4.1 - Ferramentas MOCK

Fonte: Elaborado pelo autor.

REFLITA

Podemos citar as ferramentas EasyMock, o jMock e o Mockito como as mais utilizadas no processo de testes. Ferramentas que também podem ser utilizadas para testes de unidade em processos ágeis. Disponível em: <http://www.desenvolvimentoagil.com.br/xp/praticas/tdd/mock_objects>.

Ao se trabalhar com essas ferramentas, é essencial ter o conceito do que são mocks e, claro, o conceito de testes de software. Trabalhar com todas as possibilidade de incidência de erros e formas de verificar isso acarretará em um software com maior confiabilidade e qualidade; lembremos, ainda, que se ter 100% de um software livre de erros é impossível, mas podemos trabalhar para esse percentual atingir o mais alto possível.

A título de curiosidade, podemos falar um pouco mais sobre a ferramenta Mockito, que auxilia na criação de mocks, vamos lá? Iniciaremos explicando um exemplo prático para melhor entendimento. Assim, vamos utilizar um exemplo que sempre aparece em referências bibliográficas, sites e outros, por ser algo fácil e prático de entender.

Iremos modelar um marceneiro. Mas como assim um marceneiro? Criaremos uma classe marceneiro como se fosse um método com suas características. Desta forma, se fossemos criar o código fonte, iríamos criar classes destinadas a criar um marceneiro e tudo o que ele pode fazer. No caso de verificar como que um marceneiro irá realizar determinadas ações, como cortar algo, e que tipo de ferramenta ele vai utilizar, podemos criar mocks que nos auxiliem; assim, você não precisa criar de verdade uma "serra" para verificar os testes em cima das classes marceneiro e outros, basta realizar a simulação, que é o intuito do mock e, assim, verificar se a função cortar com a serra funciona corretamente.

Para melhor entendimento, você pode acessar o link <http://www.vidageek.net/2013/08/27/mockito-mocks/>, onde temos trechos de códigos que ficará melhor para o nosso entendimento da ferramenta mockito.

A seguir, ainda descreveremos algumas definições retiradas do site da ferramenta (https://code.google.com/archive/p/mockito/) e que caso você trabalhe com isso, irá ser de muito auxílio, vamos ver? A seguir, citamos alguns comandos:

- **@Mock**: Anotação que você coloca para que o MockitoAnnotations.initMocks(this) inicialize os seus mocks.
- **Mockito.mock(classe)**: Caso não haja a geração automática de mocks, você pode criá-los manualmente com esse método.
- **Mockito.when(chamada_de_metodo)**: Você utiliza para ensinar o seu mock o que ele deve devolver quando chamado.
- **Mockito.doThrow(exception)**: Semelhante ao when, mas você usa quando quer que a invocação do método lance a exceção **exception** Object mock = Mockito.mock(Object.class); doThrow(new NullPointerException()) .when(mock).equals(anyObject());

FIQUE POR DENTRO

O mock auxilia muito em relação à facilidade em testes de unidade. Claro, ele possui diversos softwares, como já vimos, mas um dos frameworks mais utilizados é o mockito, que trabalha com a linguagem java. Nesse sentido, separamos um artigo bem bacana sobre mockito para a sua leitura, que é de extrema importância em relação a novos conhecimentos. Vamos lá?

Disponível em: <http://blog.caelum.com.br/facilitando-seus-testes-de-unidade-no-java-um-pouco-de-mockito/>.

ATIVIDADES

4) Mocks são ferramentas importantes e que simulam o comportamento de objetos reais. Das alternativas a seguir, quais ferramentas mock são exemplos de ferramentas que utilizam a linguagem java.
   a) Mock PP, Simple Teste e Mockito.
   b) Mock, jMock e Mockito.
   c) FlexMock, EasyMock e MockitNow.
   d) Moq, rMock e Amop.
   e) Mocha, jMock e SMock.

INDICAÇÕES DE LEITURA

Nome do livro: Teste de Software

Editora: Altabooks

Autor: Rios, Emerson; Moreira Filho,Trayahú R.

ISBN: 978-85-7608-775-5

Comentário: O livro trata de atividades de teste de software que devem ser inseridas no processo de desenvolvimento, por testadores ou por responsáveis da execução destas atividades. No livro, é explanado sobre a importância desses profissionais, que são técnicos altamente qualificados. Esse livro dá um enfoque aos principais aspectos relacionados ao teste de software, desta forma, além das métricas e metodologias, é apresentado como os testes devem ser organizados e executados.

INDICAÇÕES DE FILME

Nome do filme: Black Mirror

Gênero: Ação

Ano: 2011

Comentário: É uma série que retrata como a sociedade lida com tecnologia e de uma forma ou de outra realiza previsões sobre como novas ou possíveis tecnologias irão afetar o mundo num futuro bem próximo. Há diversas temporadas e tratam assuntos comoreality shows, realidade virtual e redes sociais.

CONCLUSÃO DO LIVRO

Olá, caro(a) aluno(a), a partir deste livro angariamos conhecimentos a respeito de testes de software e sua importância no decorrer de produção de um projeto de software. Testes estes que devem ser compreendidos de forma adequada para que no final nosso software possua um grau de qualidade suficiente para cumprir, de forma perfeita, o que o sistema solicitado pelo usuário solicitou.

Vimos na primeira unidade os conceitos principais de teste de software e suas principais características, expondo também o porque é necessário testar e como é importante realizar um bom gerenciamento de um teste de software, expondo o processo fundamental do teste.

Na unidade dois entramos um pouco mais a fundo em tipos de testes de software e suas principais características, falando sobre testes de unidade, testes de integração, testes de validação, testes de sistema e por fim, testes de aceitação.

Em nossa terceira unidade, nos aprofundamos um pouco mais em técnicas de testes de software, exposto as características de testes funcionais, estruturais, regressão, carga, segurança, estresse e usabilidade.

E por fim, em nossa última unidade, falamos um pouco sobre frameworks de testes de software e os conceitos a respeito de automação de teste. Expomos o porque também é importante automatizar alguns tipos de testes e apresentamos algumas ferramentas para determinadas linguagens.

Ao final de nossos estudos, temos a possibilidade de concluir a importância dos testes e destacar o quanto vem crescendo essa área, o que possibilita dar um passo à frente e estar preparado para o mercado de trabalho.