Desenvolvimento de Sistemas

Estratégias e níveis de teste

Para o bom andamento de um projeto, é essencial que exista planejamento na realização de algumas funções, e com a etapa de teste isso não é diferente. Para o sucesso do projeto, é necessária a existência de um roteiro para a realização das tarefas, definindo questões como tempo, esforço a ser realizado, recursos necessários e passos a serem executados.

Estratégias de teste: preventiva e reativa

A estratégia deve nortear alguns requisitos que são necessários para o acompanhamento, portanto deve ter os seguintes itens: indicar os objetivos do teste que está sendo realizado; indicar quais serão os papéis e as pessoas envolvidas no plano de teste; e determinar qual é o padrão que as documentações seguirão, por exemplo, como será desenvolvido o documento padrão de relatório de defeitos. Além de determinar quais serão os tipos de testes que cada nível terá, é preciso especificar quais serão as técnicas de testes aplicadas, assim como determinar a utilização de testes automáticos/manuais.

Requisitos de cobertura e métrica também devem estar especificados no documento de estratégias, portanto deve conter a quantidade de *bugs* esperados e definir a quantidade de *bugs* por linha e a quantidade de vezes que uma tarefa retorna para o desenvolvimento. Também deve especificar quais serão as ferramentas utilizadas para o acompanhamento dos defeitos, ou seja, o uso de ferramentas como Jira e Mantis, e definir como deve ser realizada a descrição dos problemas.

Ao criar a estratégia, devem-se considerar também os ambientes de testes que serão utilizados (desenvolvimento, homologação e produção) e a definição de quais relatórios serão realizados e quais devem ser entregues.

Esses itens são os sugeridos pela norma do IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos) que especifica o padrão a ser seguido, porém algumas empresas realizam adaptações conforme a sua realidade, considerando técnicas e metodologias de desenvolvimento aplicadas.

Geralmente a estratégia de teste fica em um *wiki*, em uma página *web* interna, em razão da fácil manutenção e do fácil acesso a todos da instituição, pois manter outras formas de documento gerará entraves para acesso, como a necessidade de licenças para realizar edições ou até mesmo a necessidade de *download* de arquivo toda vez que a estratégia passar por alterações.

Alguns fatores são essenciais para a criação da estratégia. O primeiro passo é compreender a realidade de desenvolvimento da instituição, ou seja, ambientar-se às ferramentas que a empresa usa, compreendendo como acontecem os processos e seus fluxos e inteirando-se das tecnologias que são utilizadas (linguagem de programação e teste com suas métricas de cobertura). O segundo fator de grande importância é conhecer a quantidade de ambientes de testes que estarão disponíveis e se serão utilizados ambientes gerais ou ambientes específicos para cada tipo de teste, por exemplo, o teste de carga.

Outro fator preponderante é conhecer quais são as áreas mais importantes do sistema que está sendo desenvolvido. Desse modo, deve-se compreender quais pontos, caso sejam liberados com falhas, podem ser fatais ao projeto, significando uma mancha irrecuperável para a imagem do cliente.

Um exemplo para uma situação dessas seria projetar um *software* para um sistema hospitalar que auxiliará médicos em diagnósticos de exames de alto grau de risco. Nesse caso, uma falha poderia trazer sérias consequências, pois geraria um diagnóstico que poderia colocar um paciente em grave risco de vida. No caso de ocorrência de uma fatalidade, isso afetaria a integridade dessa instituição junto à opinião pública, acarretando uma marca negativa tanto para o cliente quanto para a empresa de desenvolvimento. Além disso, possíveis problemas jurídicos poderiam ser desencadeados.

Outro fator essencial é conhecer a pirâmide de teste automatizado, para iniciar pelas bases da pirâmide, verificando se já existe alguma métrica de cobertura ou, no caso de estar sendo realizada a primeira bateria de testes em um *software* novo, criando esses parâmetros. Após essas definições serem realizadas, o próximo passo é subir para os degraus mais altos da estrutura e, por fim, verificar quais são os testes não funcionais que devem ser realizados.

Esses são os fatores que devem ser considerados na hora da criação de estratégias, aplicando-os também para atualizações futuras.

Existem algumas abordagens que podem ser aplicadas, como no caso da abordagem preventiva ou reativa, que considera o período em que os testes devem iniciar, e da abordagem analítica ou heurística, que considera a base de dados conhecida.

Estratégia preventiva

A abordagem preventiva considera a participação da equipe de teste desde o momento em que o projeto é iniciado, minimizando, dessa forma, os custos, pois, ao especificar casos de teste e aplicá-los, possíveis falhas são encontradas com antecedência no processo. Assim, evita-se que ocorra retrabalho pela equipe de desenvolvimento, otimizando o tempo para o desenvolvimento de tarefas essenciais.

Um exemplo de projeto em que podem ser utilizadas estratégias preventivas é o desenvolvimento de uma aplicação Java para a organização de tarefas pessoais, em que, durante o processo de desenvolvimento, testes unitários foram produzidos para serem executados com periodicidade, garantindo a funcionalidade das classes e dos métodos da aplicação. Também foram produzidos testes de integração para cobrir a interação correta entre as classes e os métodos, validando entradas de dados e inclusão de operações de adição, edição e remoção de tarefas.

Estratégia reativa

Em uma abordagem de teste reativa, a equipe de teste participará do projeto já durante o desenvolvimento, ou seja, quando funcionalidades já foram desenvolvidas e muitas delas integradas, o que, após a realização dos testes e com o encontro de falhas, pode significar em retrabalho e onerar custos.

Um exemplo de projeto em que uma estratégia de teste reativa foi aplicada é o desenvolvimento de um aplicativo de gerenciamento de finanças pessoais. Durante o processo de desenvolvimento, foram realizados testes reativos para garantir a funcionalidade e a correção de erros no aplicativo. Isso incluiu testes de unidade para verificar o comportamento de cada componente individualmente, testes de integração para verificar a interação entre componentes e testes de sistema para verificar a funcionalidade geral do aplicativo.

Abordagem analítica e heurística

Na abordagem analítica, são utilizados dados para realizar o planejamento dos testes. É adicionada uma métrica para finalizar os testes, por exemplo: os testes são interrompidos quando chegarem a uma cobertura de 95%. Dessa forma, os testes são realizados considerando parâmetros como custo e tempo para definir a quantidade e a intensidade dos testes.

A técnica de analise heurística consiste na avaliação da usabilidade, para identificar problemas de usabilidade em sistemas interativos, como *softwares* e aplicações. Nessa abordagem heurística, os testes são realizados considerando o conhecimento e o domínio da equipe de teste, conforme os requisitos especificados no projeto. Após a realização dos testes, é fornecido um *feedback* sobre os problemas com a interface e a experiência do usuário.

Níveis de teste: unitário, sistema, integração e aceitação

A realização de testes é uma etapa importante para assegurar a qualidade de um *software*, portanto é necessária a realização de gerenciamento e organização dos testes, o que abrange a definição de níveis de testes a serem executados.

Sendo o conjunto de tarefas relacionadas ao gerenciamento e à organização dos testes, os níveis de testes consistem em: componentes, integração, sistema e aceite. Cada tarefa é realizada em um determinado momento do desenvolvimento do projeto, gerando uma esfera de teste, podendo ser realizado em unidades individuais ou componentes até englobar sistemas totalmente concluídos, tendo relação com diferentes tarefas nas etapas do desenvolvimento de *software*.

Cada nível de teste deve ter um ambiente condizente com a sua realidade. É possível citar como exemplo o teste de aceitação, em que o ideal é a realização com aspectos semelhantes ao ambiente de produção em que o sistema será executado. Já um teste de componente geralmente é realizado no ambiente de desenvolvimento.

Um nível de teste deve ter as seguintes propriedades: objetivos específicos; bases de teste diferentes para serem utilizadas para criar casos de testes; objeto de teste, que é aquilo que será testado; defeitos e falhas típicas e abordagens; e responsabilidades específicas.

O teste de componentes também é conhecido como teste de unidade. O teste de integração tem duas partes, sendo divido em: integração entre componentes e integração entre sistemas. Já o teste de aceite ou aceitação pode ser: aceite de usuário, aceite operacional, aceite contratual e alfa/beta teste.

Teste de componente (unidade)

Será um teste realizado no menor fragmento do sistema, podendo ser realizado em uma função, uma classe ou um componente do *software*. Portanto, é necessário ter acesso ao código-fonte, sendo realizado pelo próprio desenvolvedor, de forma isolada de outras partes do sistema, como ao testar uma função que calculará o valor a ser pago em um período dentro de um estacionamento. A função deve executar outra função ao final do pagamento, a qual emite a nota fiscal. O teste deve ser executado apenas na função de cálculo do pagamento, simulando a sequência da execução da outra função. Isso é uma característica essencial do teste de unidade. Outra característica marcante é a rapidez com a qual o teste pode ser executado. Em metodologias ágeis, ele é realizado antes do desenvolvimento do código, conhecido como TDD, que significa *test driven development* (desenvolvimento guiado por teste, em português). Outra propriedade desse tipo de teste é a automatização dele.

Esse teste tem como objetivos reduzir risco, considerar aspectos de requisitos funcionais e não funcionais dos componentes esperados e especificados, gerar uma relação de confiança na qualidade do componente, buscar defeitos no componente e evitar que os problemas se expandam para níveis mais altos de teste.

Para a base de teste é utilizado o projeto detalhado, o código e as especificações dos componentes. O objeto de teste é a própria estrutura de código, considerando classes e módulos de bancos de dados.

São consideradas falhas típicas funções que não têm lógica e/ou estrutura de desenvolvimento de códigos corretos e que destoam do que foi informado na especificação do projeto. A velocidade de correção do teste se dá em razão de a realização do teste ser feita pelo próprio desenvolvedor, que, ao perceber a falha, já realiza a correção, sem a necessidade de abrir um chamado que parará em uma lista de eventuais ajustes.

Considere o seguinte código:

```
public class Calculadora {
public int soma(int a, int b) {
    return a + b;
}

public int subtrai(int a, int b) {
    return a - b;
}

public int multiplica(int a, int b) {
    return a * b;
}

public int divide(int a, int b) {
    if (b == 0) {
        throw new IllegalArgumentException("Não é possível dividir por zero.");
    }
    return a / b;
}
}
```

Agora, pense no seguinte exemplo. Ao criar uma classe para realizar as quatro operações básicas da matemática, serão realizados testes para verificar se a soma, a subtração, a multiplicação e a divisão estão retornando o valor correto do cálculo e se na divisão, ao informar um valor 0, uma exceção é lançada.

Teste de integração

No teste de integração, assim como no teste de unidade, é necessário acesso ao código-fonte, de modo a realizar testes baseados na integração entre componentes ou em sistemas, se for o caso. Desenvolvedores geralmente realizam testes nos componentes desenvolvidos, enquanto os testadores realizarão os testes caso exista a integração entre um ou mais sistemas.

O teste de integração de *software* é um tipo de teste que verifica se as diferentes partes de um sistema de *software* estão funcionando corretamente juntas, isto é, se há uma integração adequada entre as partes. Ele é importante porque permite identificar problemas que não podem ser detectados por meio de testes unitários isolados, por exemplo, problemas de comunicação entre componentes, problemas de consistência de dados entre módulos, entre outros.

O objetivo do teste de integração é assegurar que o sistema como um todo funciona corretamente, antes do lançamento em produção. Ele é realizado após o término dos testes unitários e geralmente envolve a combinação de vários módulos ou componentes para formar o sistema completo.

É importante ressaltar que o teste de integração é subdivido em duas partes. A primeira é o teste de integração de componentes, que é realizado após o teste de componentes, tendo o objetivo de testar as interações e interfaces nos componentes integrados, de forma automatizada na maioria das situações. Em um desenvolvimento interativo e incremental, esses testes ocorrem na execução de integração contínua.

A segunda parte é a integração de sistema, voltada para a interação entre interfaces do sistema, considerando inclusive interfaces que possam ser fornecidas por sistemas de terceiros, por exemplo, um sistema que realizará vendas e consumirá uma API (*application programming interface*, ou, em português, interface de programação de aplicação) de transações bancárias para realizar a funcionalidade de pagamento.

Como base de teste pode ser utilizado o próprio *software*, o diagrama de sequência, os protocolos de comunicação e as especificações de interfaces, os casos de uso e a arquitetura em nível de componente ou sistema. Como objetos de testes, tem-se: subsistemas, bancos de dados, infraestrutura, interfaces e APIs.

Os defeitos típicos que podem ser encontrados no teste de integração entre componentes são: ausência de dados, dados errôneos e desenvolvimento de código incorreto. O uso de bibliotecas incompatíveis pode gerar falhas por falta de comunicação na troca de informações entre os componentes, assim como a falta de tratamento para essas falhas. Para os testes de integração de sistemas, há, além dos já citados, os seguintes defeitos típicos: mensagens incoerentes entre os sistemas e não seguimento de especificações obrigatórias de segurança.

Considere o seguinte código:

```
public class SistemaPedidos {
private ProdutoDAO produtoDAO;
private PedidoDAO pedidoDAO;
private EstoqueService estoqueService;

public void realizaPedido(Pedido pedido) {
    List<Produto> produtos = produtoDAO.buscaProdutos(pedido.getIdsProdutos());
    if (!estoqueService.temEstoque(produtos)) {
        throw new IllegalArgumentException("Produto sem estoque");
```

```
    }
    estoqueService.diminuiEstoque(produtos);
    pedidoDAO.salvaPedido(pedido);
}
}
```

Agora, analise a seguinte situação. A classe **SistemaPedidos** representa um sistema de gerenciamento de pedidos. Ela depende de outras três classes: **ProdutoDAO**, **PedidoDAO** e **EstoqueService**.

Nesse caso, o teste de integração desse sistema verificaria se essas três estruturas estão sendo utilizadas de forma correta pela classe **SistemaPedidos**. Dessa forma, seriam realizados testes para verificar se a busca dos produtos pelo ID está funcionando corretamente. Ao acessar a classe **ProdutoDAO**, verifica se o estoque está funcionando corretamente; ao acessar a classe **EstoqueService**, verifica se está diminuindo corretamente do estoque também presente na classe **EstoqueService**; por fim, ao acessar a classe **PedidoDAO**, verifica se a persistência do pedido está funcionando de forma correta. Garante-se, assim, que todas essas partes do sistema estejam funcionando corretamente em conjunto.

Teste de sistema

É realizado no momento em que o sistema está mais próximo para a liberação de uso ao usuário final. Com base nos dados gerados nesse teste, é decidido ou não se haverá a liberação para a publicação do sistema, pois, como os testes são realizados utilizando um ambiente próximo ao contexto que o usuário utilizará, falhas impactantes podem mudar o rumo de uma finalização do projeto.

Praticamente todas as funções do sistema são testadas, incluindo requisitos legais e regulamentares do *software*. Por exemplo, em um *software* da área contábil, esse nível de teste verificará se os cálculos de impostos estão adequados ao que está estipulado por lei.

Os objetivos assemelham-se aos níveis anteriores, buscando verificar se o sistema segue os itens projetados e especificados, assim como o ponto de oclusão em que está, encontrando defeitos e evitando que falhas possam chegar ao nível de uso do usuário, ou seja, liberar um produto que tem sérios *bugs* e falhas.

A base de teste são os casos de uso, as *users stories*, os diagramas de estado, os manuais de usuário, o relatório de análise de riscos e as especificações de requisitos do sistema. Com isso, são testadas as aplicações em si, os sistemas operacionais, os sistemas de *hardware* e *software* e as configurações do sistema.

As falhas esperadas nesse nível são baseadas em critérios definidos pelos manuais do sistema, por exemplo, cálculos expressos e situações condicionais que devem ser seguidas conforme o fluxo esperado, além de encontrar falhas nos comportamentos definidos por meio dos requisitos funcionais e não funcionais do sistema.

Teste de aceite

O objetivo deste nível de teste é garantir que o sistema esteja operando conforme as definições, expressando a qualidade do *software* na sua totalidade, de modo que o usuário final não encontre problemas de falhas. Caso *bugs* sejam encontrados nesse ponto, significa que existem grandes riscos ao escopo do *software*, pois o objetivo desse teste é não encontrar falhas, tendo em vista que o nível anterior, o teste de sistema, deve coletar informações essenciais para o aceite ou não da publicação do projeto. Logo, disponibilizar um sistema com problemas pode causar transtornos à área de *marketing* da empresa desenvolvedora, que ficará marcada por permitir o lançamento de um projeto incompleto.

Neste nível de teste, pode ocorrer a integração de uma equipe de colaboradores, operadores e administradores da empresa que utilizará o sistema, de modo que as pessoas que utilizarão o sistema validem o projeto. Essa pratica é conhecida como teste de aceite de usuário.

Outra prática é o teste de aceite operacional, em que são executados testes que visam à confiabilidade do sistema em se manter operando de forma adequada para os usuários do ambiente de produção, realizando validações em *backups*, restaurações, *performances* e vulnerabilidades de segurança do sistema. Também é realizada a prática de teste de aceite contratual e regulatório, focando a garantia das definições contratuais que foram estabelecidas no momento em que o acordo foi oficializado via assinatura contratual. Os resultados são supervisionados por possíveis testemunhas ou agências regulamentadoras, durante a execução dos testes.

A última pratica deste nível de teste é a de alfa e beta, realizada por organizações que já vendem soluções prontas, ou seja, empresas que seguem um padrão de desenvolvimento e disponibilizam seus projetos de forma padronizada no mercado, conhecidas também como desenvolvedoras de *software* de prateleira.

Essa é uma prática que visa encontrar falhas relacionadas ao ambiente, pois algumas situações não são possíveis de serem validadas pelo time de desenvolvimento. O teste alfa é realizado no ambiente da empresa desenvolvedora, sendo realizado por testadores independentes, clientes ou operadores. Já o teste beta é realizado geralmente por potencial cliente em seu próprio local, utilizando versão do sistema comumente chamado de beta.

A base do nível de teste de aceite considera, de forma geral, basicamente itens da documentação do projeto, sendo estes: casos de uso, requisitos de sistema, documento de regulamentação e normas, contratos legais, procedimentos de instalação e relatório de análise de risco.

Já para aceite operacional, ou seja, quando o sistema está performando como se espera, é realizado tendo como base o pacote de dados, normas e regulamentos de segurança, as métricas de *performance*, os requisitos não funcionais, os procedimentos para recuperação de desastres, os procedimentos de *backup* e a restauração, assim como as documentações de operações e instruções de implantação e instalação.

Os objetos de testes são o próprio sistema, a configuração do sistema e dos dados de configuração, os processos de negócios para um sistema totalmente integrado, os relatórios, os dados de produção existentes e os processos operacionais e de manutenção.

As falhas típicas esperadas são em relação aos requisitos de negócios ou do usuário, à validação das regras de negócio, ao *software* não estar seguindo os requisitos contratuais ou regulatórios, assim como às vulnerabilidades de segurança, às falhas não funcionais e à eficácia da *performance*.

Encerramento

Neste material, você aprendeu quais fatores são essências para a elaboração de uma estratégia de teste, verificou conceitos sobre itens que devem ser considerados e compreendeu o uso de algumas estratégias de abordagem.

Além disso, você também verificou conceitos de níveis de teste, compreendendo que há quatro níveis e que cada nível de teste deve ter um ambiente adequado e as próprias propriedades.