**Técnicas de Caixa Preta (Funcional)**

Na maioria de projetos de teste, o tempo para a realização dos mesmos sempre é curto e os números de testes a serem realizados nas aplicações são inúmeros. Isso sem falar dos testes “Não-Funcionais”, como por exemplo, os de performance, de usabilidade, de stress, de escalabilidade, etc. É necessário se ter um planejamento bem elaborado para que todo o projeto de testes, consiga entregar um produto com no mínimo qualidade esperada pelo cliente e no prazo que foi acordado.

No intuito de se realizar esse processo com certa eficiência, é necessário utilizar das técnicas de teste de software que é o tema desse artigo. Para isso, respondemos a pergunta de como fazer o processo. Entretanto, para responder a pergunta, o que e o que priorizar primeiro, temos que levar em consideração os riscos associados ao projeto de teste que devem ser levantados tão logo se inicia o projeto de todo o software. Levantados e analisados os riscos e, fazendo uso correto das técnicas de teste, é muito provável que todo o projeto de teste consiga alcançar o mínimo de qualidade esperada.

O nome caixa preta vem do sentido de que nesse tipo de teste, não é necessário saber a estrutura interna de como o código foi implementado ou a tecnologia que foi utilizada. Nesse tipo de análise, essas questões são transparentes para os analistas de teste. É claro que para se ter um projeto de teste bem sucedido, é necessário usar a técnica de caixa preta e mais a de caixa branca, isto é, analisar, também, a estrutura interna da aplicação.

Na nomenclatura também se utiliza o termo em inglês “Black Box” ou “Testes Funcionais”.

O testador não está interessado em como as entradas são processadas, o testador só acompanha as saídas produzidas pelo sistema. Ou seja, só são observadas se as saídas são coerentes para as entradas dadas.

Em boa parte das empresas, uma equipe projeta o software enquanto outra o testa. A grande desvantagem dessa técnica é que nem todos os tipos de entrada são testadas e não existem "guias" para as entradas.

Alguns métodos de Caixa preta:

**Particionamento de equivalência**

Divide o domínio de entrada de um programa em classe de dados a partir das quais os casos de teste podem ser derivados. O particionamento de equivalência procura definir um caso de teste que descubra classes de erros, assim reduzindo o número total de casos de teste que devem ser resolvidos.

O projeto de casos de teste para o particionamento de equivalência baseia-se numa avaliação de classes de equivalência para uma condição de entrada. Uma classe de equivalência representa um conjunto de estados válidos ou inválidos para condições de entrada. As classes de equivalência podem ser definidas de acordo com as seguintes diretrizes:

* Se uma condição de entrada especificar um intervalo, uma classe de equivalência válida e duas classes de equivalência inválidas são definidas.
* Se uma condição de entrada exigir um valor específico, uma classe de equivalência válida e duas classes de equivalência inválidas são definidas.
* Se uma condição de entrada especificar um membro de um conjunto, uma classe de equivalência válida e uma classe de equivalência inválida são definidas.
* Se uma condição de entrada for booleana, uma classe válida e uma inválida são definidas.

Aplicando-se as diretrizes para derivação de classes de equivalências, os casos de testes para cada item de dados do domínio de entrada poderiam ser desenvolvidos e executados. Os casos de teste são selecionados de forma que o maior número de atributos de uma classe de equivalência seja exercitado de uma só vez.

**Análise de valor limite**

Um número maior de erros tende a ocorrer nas fronteiras do domínio de entrada do que no "centro". Por isso é que a análise do valor limite foi desenvolvida como uma técnica de teste. Os testes põem a prova os valores fronteiriços. Essa técnica complementa o particionamento de equivalência, e sob muitos aspectos as diretrizes se assemelham àquelas apresentadas para o particionamento de equivalência:

Se uma condição de entrada especificar um intervalo, delimitado pelos valores a e b, os casos de teste devem ser projetados com valores a e b logo acima e logo abaixo de a e b, respectivamente.

Se uma condição de entrada especificar uma série de valores os casos de teste que ponham à prova números máximos e mínimos devem ser desenvolvidos. Valores logo acima e logo abaixo do mínimo e do máximo também são testados.

Aplique as diretrizes 1 e 2 às condições de saída. Por exemplo, suponhamos que uma tabela de temperatura versus pressão seja exigida como saída de um programa de análise de engenharia. Os casos de teste devem ser projetados para criar um relatório de saída que produza um número máximo (e mínimo) permissível de entradas na tabela.

Se estruturas internas de dados do programa tiverem prescrito fronteiras, certifique-se de projetar um caso de teste para exercitar a estrutura de dados em sua fronteira.

Técnicas de grafo de causa-efeito

O grafo de causa-efeito, é uma técnica de projetos de caso de teste que oferece uma representação concisa das condições lógicas e das ações correspondentes. A técnica segue quatro passos:

1. Causas (condições de entrada) e efeitos (ações) são relacionados para um módulo e um identificador é atribuído para cada um.
2. Um grafo de causa efeito é desenvolvido.
3. O grafo é convertido numa tabela de decisão.
4. As regras da tabela de decisão são convertidas em casos de teste.

**Testes de comparação**

Usa-se quando a confiabilidade do software é absolutamente crítica. Alguns pesquisadores tem sugerido que versões independentes de software sejam desenvolvidas para aplicações críticas, mesmo quando uma única versão for usada no sistema computadorizado entregue. Essas versões independentes formam a base de uma técnica de teste de caixa preta denominada teste de comparação ou teste back-to-back.

O método de comparação não é infalível. Se a especificação a partir da qual todas as versões foram desenvolvidas estiver errada, provavelmente todas as versões refletirão o erro. Além disso, se cada uma das versões independentes produzir resultados idênticos, mas incorretos, os testes de condições deixarão de detectar o erro.

**Testes de sistemas de tempo real**

Métodos de projetos de caso de teste abrangente para sistemas de tempo real ainda precisam ser desenvolvidos. Porém uma estratégia global de qutro passos pode ser proposta:

**Teste de tarefas.**

O primeiro passo da atividade de testes de softwares de tempo real consiste em testar cada tarefa independentemente. Ou seja, testes de caixa preta e de caixa branca são projetados e executados para cada tarefa. Cada tarefa é executada independentemente durante esses testes. O teste de tarefas revela erros de lógica e de função, mas não revelará erros comportamentais ou de *timing*.

**Teste comportamental.**

Usando modelos de sistemas criados com as ferramentas CASE, é possível simular o comportamento de um sistema de tempo real e examinar seu comportamento como uma consequência de eventos externos. Essas atividades de análise podem servir como a base para o projeto de casos de teste que será realizado quando o software de tempo real for construído.

**Teste intertarefas.**

Assim que os erros em tarefas individuais e no comportamento de sistema tiverem sido isolados, a atividade de teste desvia-se para erros relacionados ao tempo. As tarefas assíncronas, que sabidamente comunicam-se entre sí, são testadas com diferentes taxas de dados e cargas de processamento para determinar se ocorrerão erros de sincronização inter tarefas, além disso, as tarefas que se comunicam por intermédio de uma fila de mensagens são testadas para descobrir erros na determinação do tamanho dessas áreas de armazenagem de dados.

**Teste do sistema.**

O software e o hardware são integrados e uma variedade completa de testes de sistema é levada a efeito, numa tentativa de descobrir erros na interface software/hardware.