Assertion Roulette

Ocorre quando um método de teste tem várias asserções não documentadas. Várias declarações de asserção em um método de teste sem uma mensagem descritiva afetam a legibilidade/compreensão/manutenção, pois não é possível entender o motivo da falha do teste.

Detecção: Um método de teste contém mais de uma instrução de asserção sem nenhuma explicação/mensagem (parâmetro no método de asserção).

Conditional Test Logic

Os métodos de teste precisam ser simples e executar todas as instruções no método de produção. As condições dentro do método de teste alterarão o comportamento do teste e sua saída esperada e levariam a situações em que o teste falha em detectar defeitos no método de produção, pois as instruções de teste não foram executadas porque uma condição não foi atendida. Além disso, o código condicional dentro de um método de teste afeta negativamente a facilidade de compreensão pelos desenvolvedores.

Detecção: um método de teste que contém uma ou mais instruções de controle (ou seja, if, switch, expressão condicional, for, foreach e while).

Constructor Initialization

Idealmente, o conjunto de testes não deve ter um construtor. A inicialização dos campos deve estar no método setUp(). Os desenvolvedores que desconhecem o propósito do método setUp() dariam origem a esse test smell definindo um construtor para o conjunto de testes.

Detecção: uma classe de teste que contém uma declaração de construtor.

Default Test

Por padrão, o Android Studio cria classes de teste padrão quando um projeto é criado. Essas classes devem servir como um exemplo para desenvolvedores ao escrever testes de unidade e devem ser removidas ou renomeadas. Ter tais arquivos no projeto fará com que os desenvolvedores comecem a adicionar métodos de teste a esses arquivos, tornando a classe de teste padrão um contêiner de todos os casos de teste. Isso também pode causar problemas quando as classes precisarem ser renomeadas no futuro.

Detecção: uma classe de teste é denominada `ExampleUnitTest' ou `ExampleInstrumentedTest'.

Duplicate Assert

Esse test smell ocorre quando um método de teste testa a mesma condição várias vezes no mesmo método de teste. Se o método de teste precisar testar a mesma condição usando valores diferentes, um novo método de teste deve ser utilizado; o nome do método de teste deve ser uma indicação do teste que está sendo realizado. Possíveis situações que dariam origem a esse test smell incluem: (1) desenvolvedores agrupando várias condições para testar um único método; (2) desenvolvedores realizando atividades de depuração; e (3) uma cópia e colagem acidental de código.

Detecção: um método de teste que contém mais de uma instrução de asserção com os mesmos parâmetros.

Eager Test

Ocorre quando um método de teste invoca vários métodos do objeto de produção. Esse test smell resulta em dificuldades na compreensão e manutenção do teste.
Detecção: um método de teste contém várias chamadas para vários métodos de produção.

Empty Test

Ocorre quando um método de teste não contém instruções executáveis. Esses métodos são possivelmente criados para fins de depuração e, em seguida, esquecidos ou contêm código comentado. Um teste vazio pode ser considerado problemático e mais perigoso do que não ter um caso de teste, pois o JUnit indicará que o teste foi aprovado mesmo se não houver instruções executáveis presentes no corpo do método. Como tal, os desenvolvedores que introduzem alterações de quebra de comportamento na classe de produção não serão notificados sobre os resultados alternativos, pois o JUnit relatará o teste como aprovado.
Detecção: Um método de teste que não contém uma única instrução executável.

Exception Handling

Esse test smell ocorre quando um método de teste explicitamente passa ou falha em um método de teste e depende do método de produção lançando uma exceção. Os desenvolvedores devem utilizar o tratamento de exceção do JUnit para passar/reprovar automaticamente no teste, em vez de escrever um código de tratamento de exceção personalizado ou lançar uma exceção.
Detecção: Um método de teste que contém uma instrução throw ou uma cláusula catch.

General Fixture

Ocorre quando uma fixture de caso de teste é muito genérica e os métodos de teste acessam apenas parte dele. Um método de setup/fixture de teste que inicializa campos que não são acessados por métodos de teste indica que a fixture é muito genérica. Uma desvantagem de ser muito genérica é que um trabalho desnecessário está sendo feito quando um método de teste é executado.
Detecção: Nem todos os campos instanciados no método setUp de uma classe de teste são utilizados por todos os métodos de teste na mesma classe de teste.

Ignored Test

O JUnit 4 fornece aos desenvolvedores a capacidade de suprimir a execução de métodos de teste. No entanto, esses métodos de teste ignorados resultam em sobrecarga, pois adicionam sobrecarga desnecessária em relação ao tempo de compilação e aumentam a complexidade e a compreensão do código.
Detecção: Um método ou classe de teste que contém a anotação @Ignore.

Lazy Test

Ocorre quando vários métodos de teste invocam o mesmo método do objeto de produção.
Detecção: vários métodos de teste chamando o mesmo método de produção.

Magic Number Test

Ocorre quando declarações assert em um método de teste contêm literais numéricos (isto é, números mágicos) como parâmetros. Números mágicos não indicam o significado/propósito do número. Portanto, eles devem ser substituídos por constantes ou variáveis, fornecendo assim

um nome descritivo para a entrada.
Detecção: Um método de asserção que contém um literal numérico como argumento.

Mystery Guest

Ocorre quando um método de teste utiliza recursos externos (por exemplo, arquivos, banco de dados, etc.). O uso de recursos externos em métodos de teste resultará em problemas de estabilidade e desempenho. Os desenvolvedores devem usar objetos fictícios no lugar de recursos externos.
Detecção: Um método de teste contendo instâncias de objetos de arquivos e classes de bancos de dados.

Redundant Print

As instruções de impressão em testes de unidade são redundantes, pois os testes de unidade são executados como parte de um processo automatizado com pouca ou nenhuma intervenção humana. As instruções de impressão são possivelmente usadas pelos desenvolvedores para fins de rastreabilidade e depuração e depois esquecidas.
Detecção: Um método de teste que invoca o método print ou println ou printf ou write da classe System.

Redundant Assertion

Esse test smell ocorre quando os métodos de teste contêm declarações de asserção que são sempre verdadeiras ou sempre falsas. Esse cheiro é introduzido pelos desenvolvedores para fins de depuração e depois esquecido. **Detecção:** Um método de teste que contém uma declaração de asserção na qual os parâmetros esperados e reais são os mesmos.

Resource Optimism

Esse test smell ocorre quando um método de teste faz uma suposição otimista de que o recurso externo (por exemplo, File), utilizado pelo método de teste, existe.
Detecção: Um método de teste utiliza uma instância de uma classe File sem chamar os métodos exists(), isFile() ou notExists() do objeto.

Sensitive Equality

Ocorre quando o método toString é usado em um método de teste. Os métodos de teste verificam os objetos invocando o método padrão toString() do objeto e comparando a saída com uma string específica. Alterações na implementação de toString() podem resultar em falha. A abordagem correta é implementar um método personalizado dentro do objeto para realizar essa comparação.
Detecção: um método de teste invoca o método toString() de um objeto.

Sleepy Test

Colocar explicitamente uma thread para dormir pode levar a resultados inesperados, pois o tempo de processamento de uma tarefa pode diferir em diferentes dispositivos. Os desenvolvedores introduzem esse test smell quando precisam pausar a execução de instruções em um método de teste por um determinado período (ou seja, simular um evento externo) e, em seguida, continuar com a execução.
Detecção: Um método de teste que invoca o método Thread.sleep().

Unknown Test

Uma declaração de asserção é usada para declarar uma condição booleana esperada para um método de teste. Ao examinar a declaração de afirmação, é possível entender o propósito do método de teste. No entanto, é possível que um método de teste seja escrito sem uma instrução de asserção; nesse caso, o JUnit mostrará o método de teste como aprovado se as instruções dentro do método de teste não resultarem em uma exceção, quando executadas. Os novos desenvolvedores do projeto acharão difícil entender o propósito de tais métodos de teste (ainda mais se o nome do método de teste não for suficientemente descritivo).
Detecção: Um método de teste que não contém uma única instrução de asserção e o parâmetro de anotação @Test(esperado).