A pirâmide do teste prático

A pirâmide de testes de Mike Cohn consiste em 3 camadas que um conjunto de testes deve conter:

1. Testes unitários
2. Testes de serviço
3. Testes de interface do usuário

#### Ferramentas e bibliotecas que veremos

* [JUnit](http://junit.org/) : nosso executor de testes
* [Mockito](http://site.mockito.org/) : para simular dependências
* [Wiremock](http://wiremock.org/) : para eliminar serviços externos
* [Pacto](https://docs.pact.io/) : para escrever testes CDC
* [Selenium](http://docs.seleniumhq.org/) : para escrever testes de ponta a ponta baseados em IU
* [REST-assured](https://github.com/rest-assured/rest-assured) : para escrever testes ponta a ponta baseados em API REST

#### Testes doubles:

1. Mocks
2. Stubs

#### Estrutura do teste

1. Configurar os dados de teste
2. Chame seu método em teste
3. Afirme que os resultados esperados são retornados

#### Testes Unitários

A base do seu conjunto de testes será composta por testes unitários. Seus testes unitários garantem que uma determinada unidade (o *objeto em teste* ) da sua base de código funcione conforme o esperado. Os testes unitários têm o escopo mais restrito de todos os testes em seu conjunto de testes. O número de testes unitários em seu conjunto de testes superará em muito qualquer outro tipo de teste.

#### Testes de integração

Todas as aplicações não triviais se integrarão com outras partes (bancos de dados, sistemas de arquivos, chamadas de rede para outras aplicações). Eles testam a integração da sua aplicação com todas as partes externas à aplicação.

* Integração de Banco de Dados

### Integração com serviços separados

#### Testes de contratos

Como você frequentemente distribui os serviços de consumo e fornecimento entre diferentes equipes, acaba se deparando com a necessidade de especificar claramente a interface entre esses serviços (o chamado **contrato** ).

#### Teste de consumidor (nossa equipe)

Nosso microsserviço consome a API de clima. Portanto, é nossa responsabilidade escrever um **teste de consumidor** que defina nossas expectativas para o contrato (a API) entre nosso microsserviço e o serviço de clima.

#### Teste de provedor (a outra equipe)

O teste do provedor precisa ser implementado pelas pessoas que fornecem a API do clima. Estamos consumindo uma API pública fornecida pela darksky.net. Em teoria, a equipe da darksky implementaria o teste do provedor para verificar se não estão quebrando o contrato entre o aplicativo deles e o nosso serviço.

#### Teste de provedor (nossa equipe)

Vimos como testar o contrato entre nosso serviço e o provedor de clima. Com essa interface, nosso serviço atua como consumidor, e o serviço de clima atua como provedor. Pensando um pouco mais adiante, veremos que nosso serviço também atua como provedor para outros: fornecemos uma API REST que oferece alguns endpoints prontos para serem consumidos por outros.

#### Testes de IU

*Os testes de IU* verificam se a interface do usuário do seu aplicativo funciona corretamente. A entrada do usuário deve desencadear as ações corretas, os dados devem ser apresentados ao usuário e o estado da IU deve mudar conforme o esperado.

#### Testes de ponta a ponta

Testes ponta a ponta (também chamados de [Testes de Pilha Ampla](https://martinfowler.com/bliki/BroadStackTest.html) ) oferecem a maior confiança quando você precisa decidir se seu software está funcionando ou não. [O Selenium](http://docs.seleniumhq.org/) e o [Protocolo WebDriver](https://www.w3.org/TR/webdriver/) permitem automatizar seus testes, conduzindo automaticamente um navegador (headless) contra seus serviços implantados, realizando cliques, inserindo dados e verificando o estado da sua interface de usuário. Você pode usar o Selenium diretamente ou usar ferramentas desenvolvidas sobre ele, como [o Nightwatch .](http://nightwatchjs.org/)

* Teste de ponta a ponta da interface do usuário

### Teste de ponta a ponta da API REST

#### Testes de aceitação

#### A questão é a seguinte: em algum momento, você deve se certificar de testar se o seu software funciona corretamente da perspectiva *do usuário* , não apenas da perspectiva técnica.

#### Teste exploratório

Trata-se de uma abordagem de teste manual que enfatiza a liberdade e a criatividade do testador para identificar problemas de qualidade em um sistema em execução. Basta reservar um tempo regularmente, arregaçar as mangas e tentar quebrar sua aplicação.