Testing for micro service

Peyman Ekhtiar

Developer for Decsoft S.A

Warsaw

Testing for micro Services

- ✓ Unit Testing
- ✓ The different types of tests
- ✓ Test Pyramid

Unit Testing

- **What is it ?**
  - A unit test focuses on a single “unit of code” that exercises a specific portion of your codebase in a particular context. Typically, each unit test sends a specific input to a method and verifies that the method returns the expected value, or takes the expected action. Unit tests prove that the code you are testing does in fact do what you expect it to do.

Unit Testing

- **Why does we use it ?**
  - It give you confidence : your code works as you expect it to work.
  - Save your time : It helps prevent regressions from being introduced and released. When you have a set of unit tests verifying your code works, you can safely change the code and trust that other parts of your program will not break.
  - Provide excellent implicits documentation : It shows exactly how the code is designed to be used.

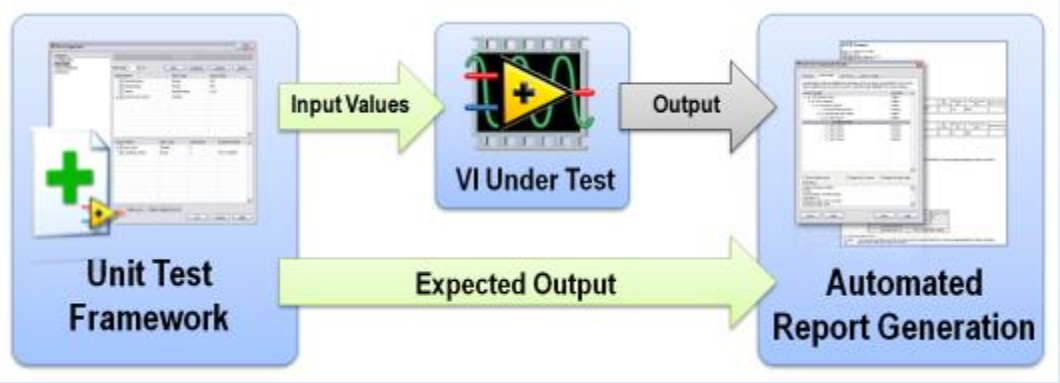
Unit Testing

- **What's the difference between Unit Testing, TDD and BDD?**
  - *Unit Testing* : A unit test focuses on a single "unit of code".
  - *TDD - Test-Driven Development*: It is a process for when you write and run your tests. Following it makes it possible to have a very high test-coverage. Test-coverage refers to the percentage of your code that is tested automatically, so a higher number is better. TDD also reduces the likelihood of having bugs in your tests, which can otherwise be difficult to track down.
  - *BDD - Behavior-Driven Development*: It is perhaps the biggest source of confusion. When applied to automated testing, BDD is a set of best practices for writing great tests. BDD can, and should be, used together with TDD and unit testing methods.

Unit Testing

- **What is Unit Test Structure?**
  - Set up all conditions for testing.
  - Call the method (or Trigger) being tested.
  - Verify that the results are correct.
  - Clean up modified records.

Unit Testing



- **What is a test framework?**
  - A test framework is simply a formalization or statement of the method and processes followed when testing within any given organization, based on the complete test approach. Expressed as a collection of assumptions, concepts and tools this can be implemented by automated software testing.
  - Test frameworks are popular because they are low maintenance, and because changes can be made to test case files without having to update things like Driver and Startup scripts. This keeps things clean and simple and reduces the risk of mistakes. Choosing the right framework technique also helps in maintaining lower costs, as the costs associated with test frameworks are due to development and maintenance efforts.

The different types of tests

- Unit tests
- Integration tests
- Functional tests
- End-to-end tests
- Acceptance testing
- Performance testing
- Smoke testing

The different types of tests

- Unit tests :
  - Unit tests are very low level, close to the source of your application. They consist in testing individual methods and functions of the classes, components or modules used by your software. Unit tests are in general quite cheap to automate and can be run very quickly by a continuous integration server.

The different types of tests

- Integration tests:
  - Integration tests verify that different modules or services used by your application work well together. For example, it can be testing the interaction with the database or making sure that microservices work together as expected. These types of tests are more expensive to run as they require multiple parts of the application to be up and running.

The different types of tests

- Functional tests :
  - Functional tests focus on the business requirements of an application. They only verify the output of an action and do not check the intermediate states of the system when performing that action.

    - There is sometimes a confusion between integration tests and functional tests as they both require multiple components to interact with each other. The difference is that an integration test may simply verify that you can query the database while a functional test would expect to get a specific value from the database as defined by the product requirements.

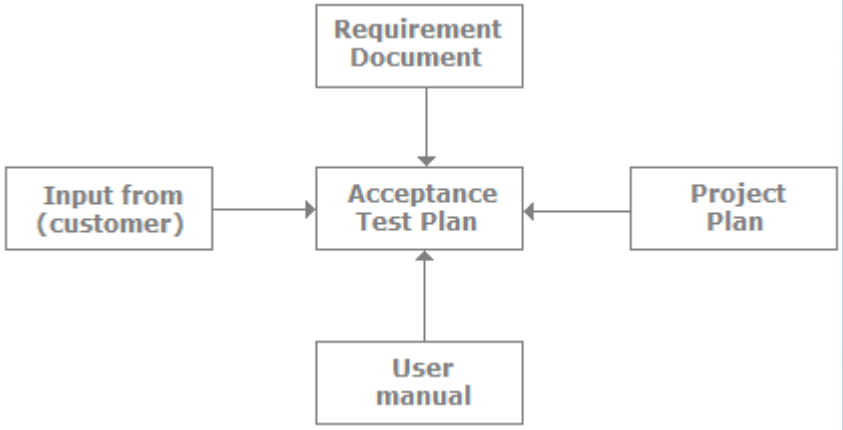
The different types of tests

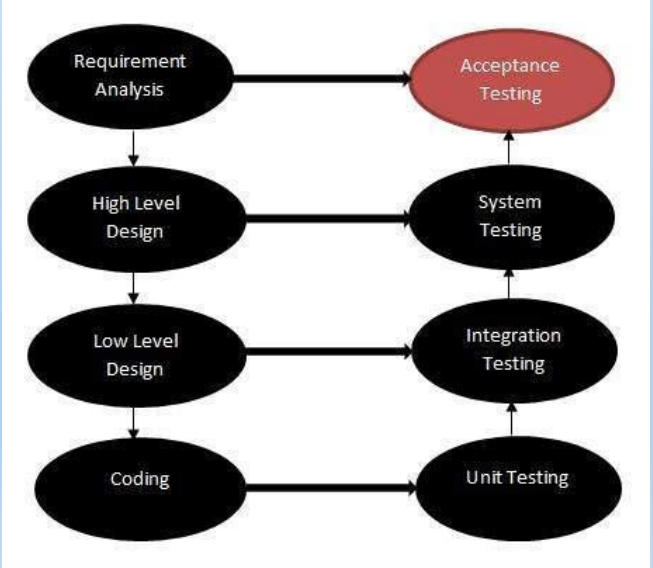
- End-to-end tests
  - End-to-end testing replicates a user behavior with the software in a complete application environment. It verifies that various user flows work as expected and can be as simple as loading a web page or logging in or much more complex scenarios verifying email notifications, online payments, etc...

    - End-to-end tests are very useful, but they're expensive to perform and can be hard to maintain when they're automated. It is recommended to have a few key end-to-end tests and rely more on lower level types of testing (unit and integration tests) to be able to quickly identify breaking changes.

The different types of tests

- Acceptance testing :
  - Acceptance tests are formal tests executed to verify if a system satisfies its business requirements. They require the entire application to be up and running and focus on replicating user behaviors. But they can also go further and measure the performance of the system and reject changes if certain goals are not met.





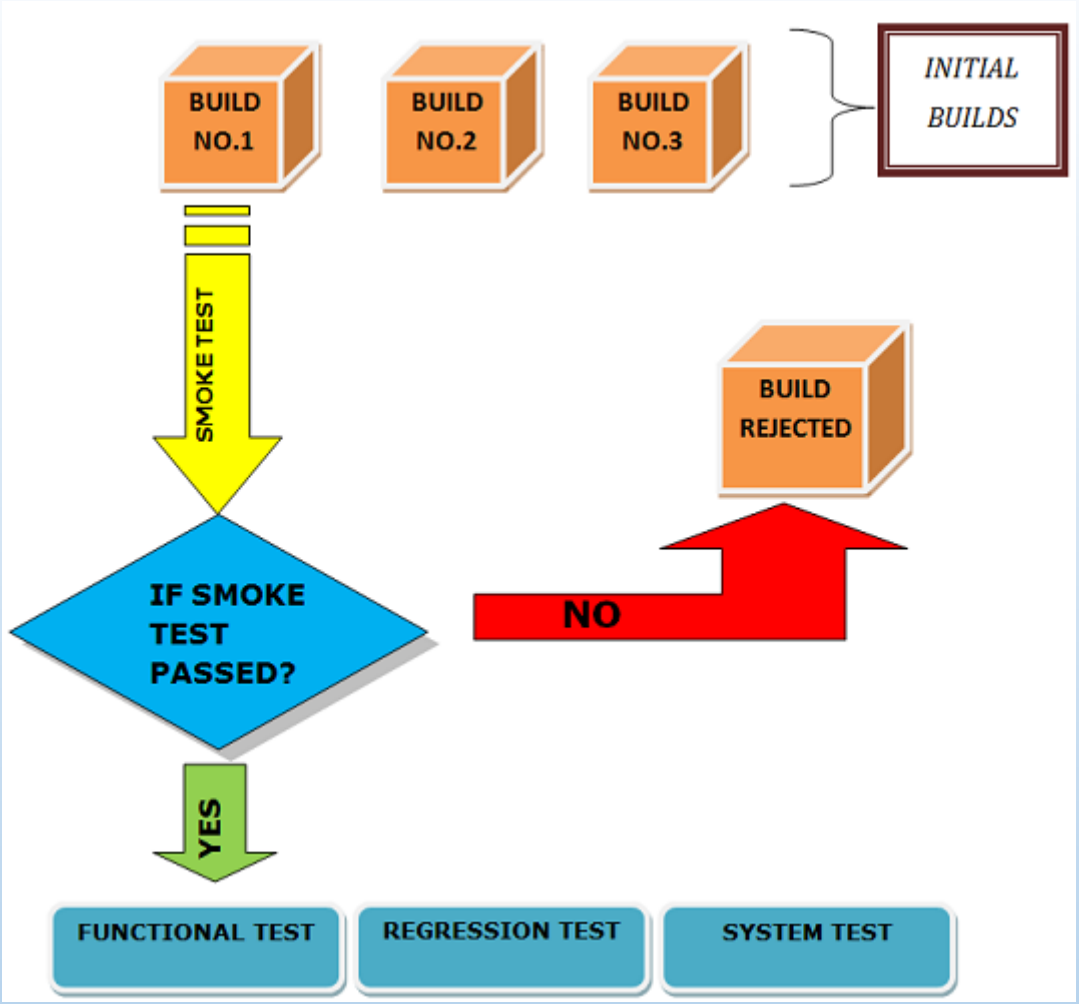
The different types of tests

- Performance testing :
  - Performance tests check the behaviors of the system when it is under significant load. These tests are non-functional and can have the various form to understand the reliability, stability, and availability of the platform. For instance, it can be observing response times when executing a high number of requests, or seeing how the system behaves with a significant of data.

- Performance tests are by their nature quite costly to implement and run, but they can help you understand if new changes are going to degrade your system.

The different types of tests

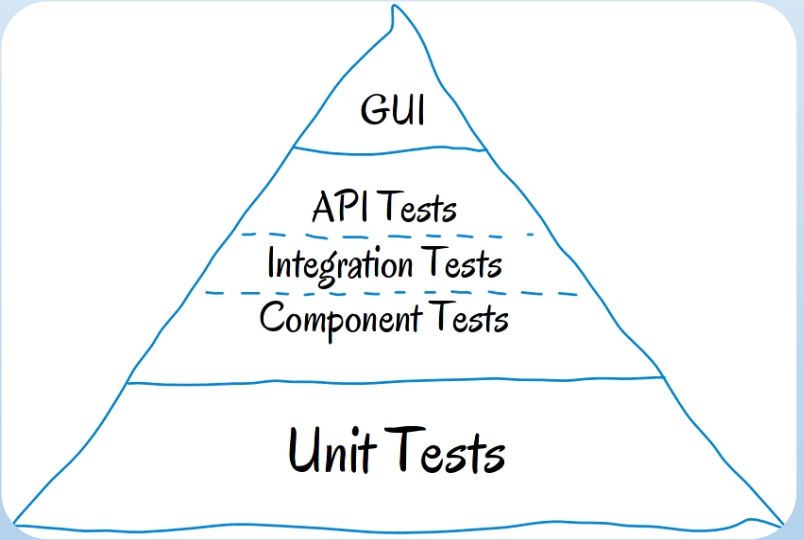
- Smoke testing :
  - Smoke tests are basic tests that check basic functionality of the application. They are meant to be quick to execute, and their goal is to give you the assurance that the major features of your system are working as expected.



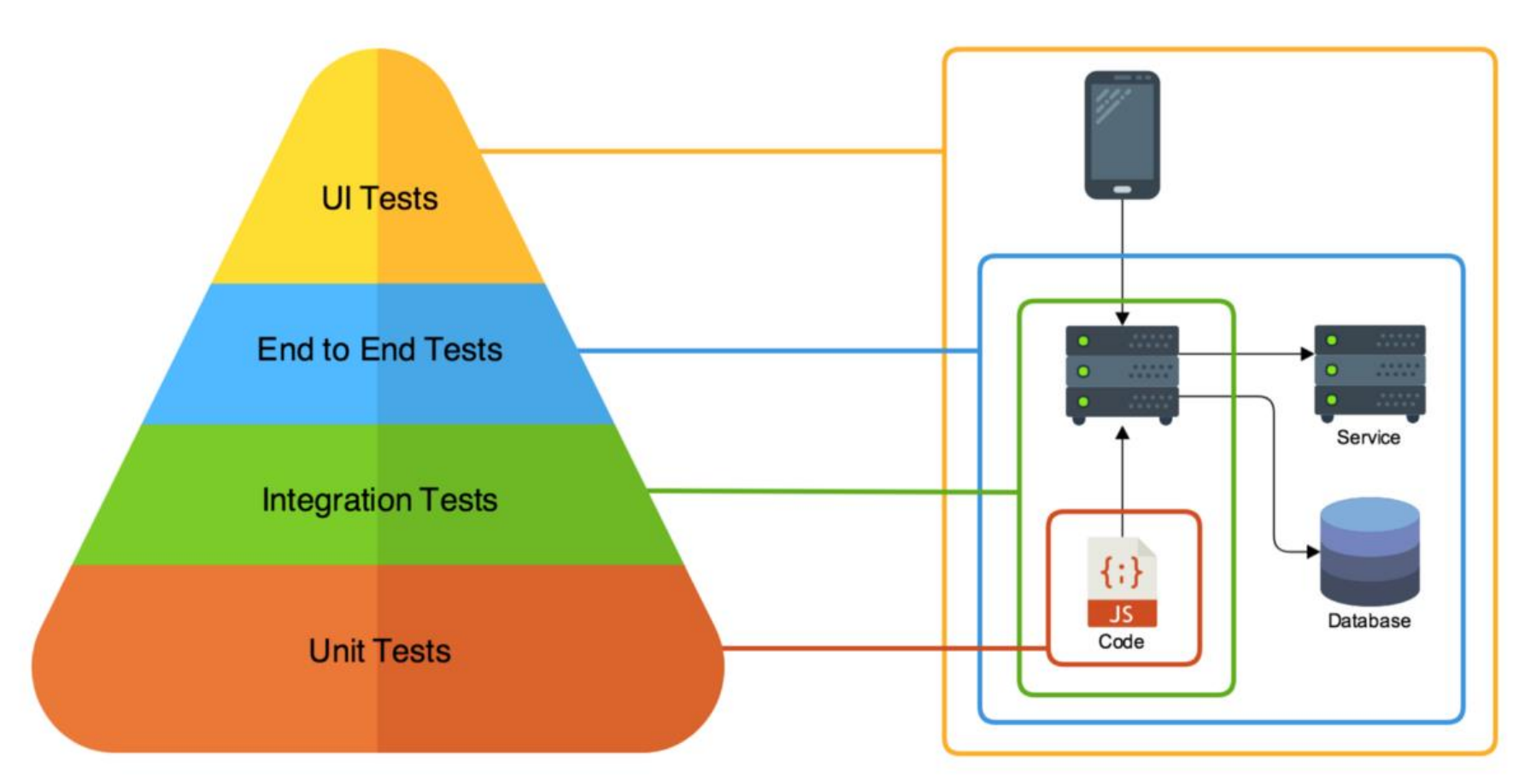
Smoke tests can be useful right after a new build is made to decide whether or not you can run more expensive tests, or right after a deployment to make sure that they application is running properly in the newly deployed environment.

Test Pyramid

- **What is test pyramid?**
  - The test pyramid is a concept developed by Mike Cohn. Its essential point is that you should have many more low-level unit tests than high level end-to-end tests running through a GUI. A test pyramid delivers a graphical representation of a best-case test scenario where you have a large number of low-level unit tests (around 70%) and comparatively few high level end-to-end system tests (about 10%), with an intermediate layer of integration tests sandwiched in between which adds up to around 20%.

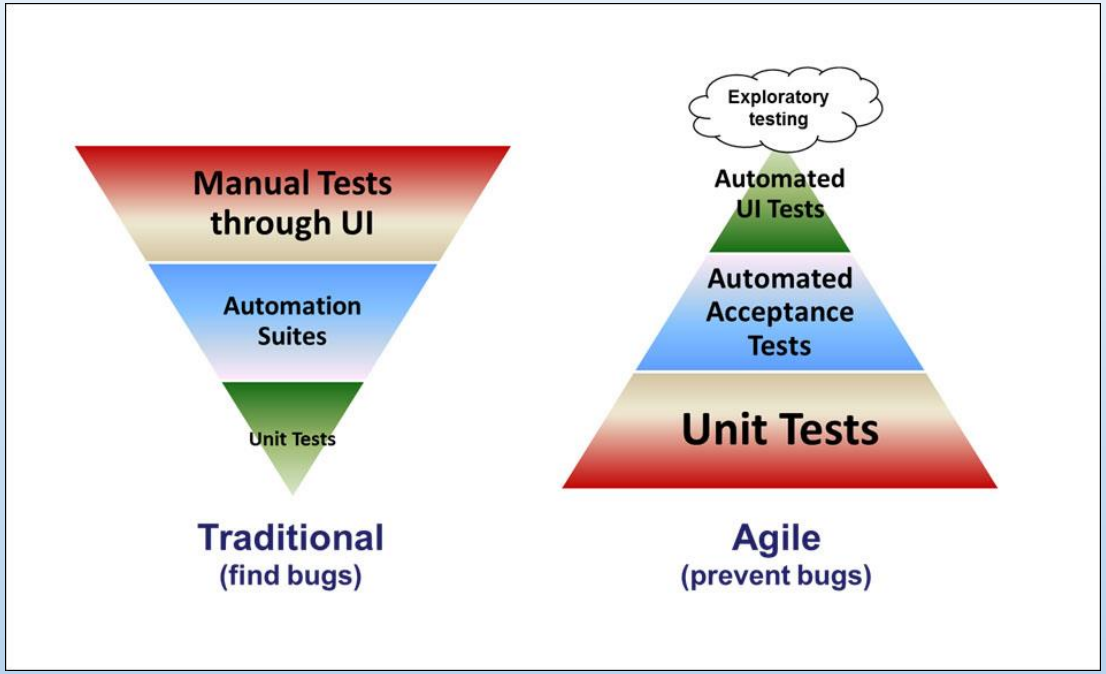


Test Pyramid



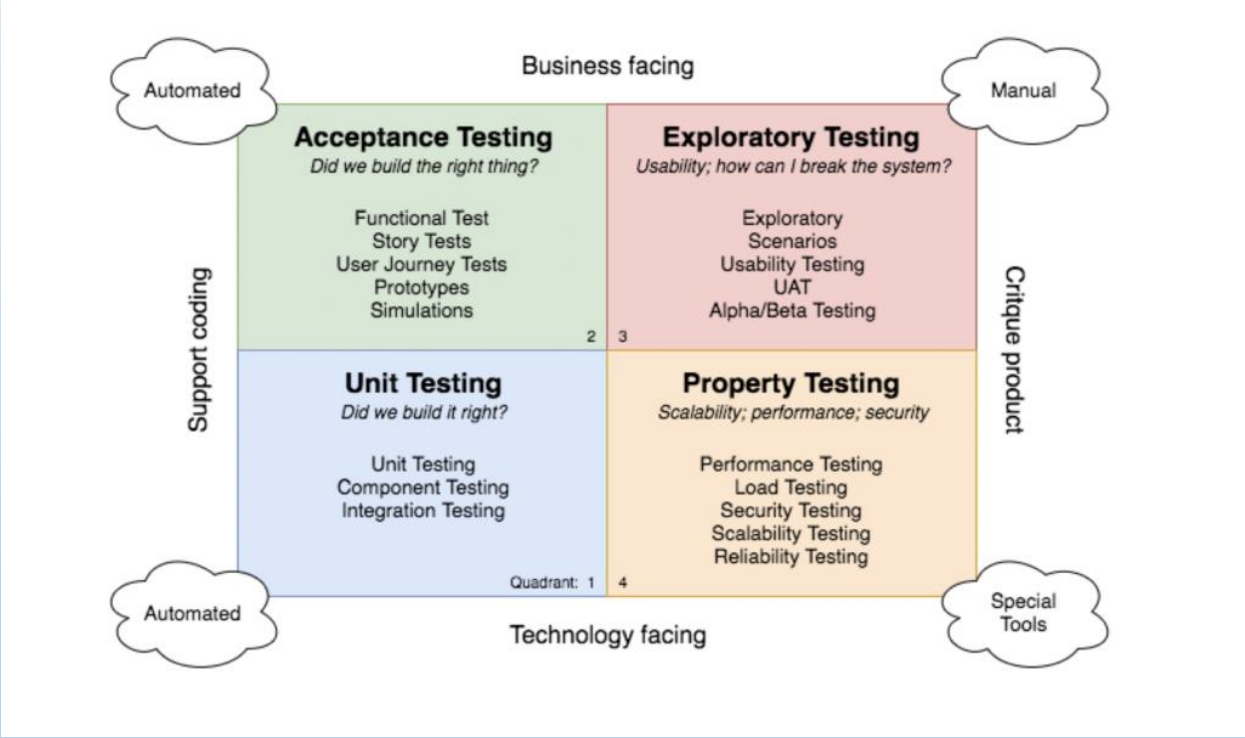
Test Pyramid

- In traditional test Pyramid:
  - Traditional software testing can be described using the pyramid on the left of Figure. The great majority of testing is done using test plans to manually exercise the system through the UI. There may be some automation in the middle tier to test services. There may be some unit testing by developers.



Test Pyramid

- Test Quadrant :
  - The quadrants include a lot of different types of testing, and each can be useful at certain times. What you choose will depend on what your system does, how it's architected and the characteristics of your system's users. In terms of automated testing for Microservices, we will exclude manual testing and focus primarily on the left-hand side quadrants.



Test Pyramid

- Automated Testing by Layer :

Layer	Proportion	Scope	Isolation	Speed	Cost	Quadrant	Orientation	Types
Unit	Foundational level/many, many unit tests	Least scope	Highest isolation	Fastest	Cheapest	1	Technology-facing	• Unit testing
Service	Moderate number of service tests	More than unit but less than whole system	Less isolation than unit level	Fast to moderate	Cheap to moderate	1,4	Technology-facing	• API testing • Integration testing • Consumer-driven contract testing
End-to-End	Very few ("very low double digits even for complex systems")	Highest scope/whole system	None	Slow	Expensive	2,3,4	Business- and Technology-facing	• Functional testing • User journey testing

Thank you for your attention