Definitions tests

EndToEnd	is a **methodology** used to test whether the **flow** of an application is performing as designed from **start to finish**. The purpose of carrying out end-to-end tests is to identify system dependencies and to ensure that the right information is passed between various system components and systems.
Unit	is a methodology to test the smallest unit of **functionality**, typically a method/function (e.g. given a class with a particular state, calling x method on the class should cause y to happen). Unit tests should be focussed on one particular feature.

Confusion of tongues

- Acceptance test = end-to-end test = scenario test = system tests = black box test
- Unit test = white box test = ...

Different structure of test (pseudocode)

	EndToEnd tests	Unit
Keywords	describing **flow**, requirement, scenario, happy scenario	describing **functionality**, edge cases, boundaries, exceptions
Tells you	the code is failing	where the code is failing
Tooling	Fitnesse or jasmine E2E	Jasmine (QUnit)

Essential differences in testcode

EndToEnd tests:

```
HorizonEndToEndTest {
  public void userChangesHorizon(){

    userLogin();
    userSelectsHorizon();
    verifyHorizonPage();
    userChangesHorizon();
    verifyHorizonChanged();
    userClosesApplication();
  }

// All subfunctions executes
// jquery calls ...
}
```

Unit:

```
describe(“A horzioncontroller”) {

  it(“generates dates”) {
    var contr = new Controller();
    expect(contr).toBe(..)
  }

  it(“generates periods”) {
    ..
  }
}
```

The Importance of End-to-End Testing: A Horror Story

Nat was once brought onto a project that had been using TDD since its inception. The team had been writing tests to capture requirements and show progress to their customer representatives. They had been writing unit tests for the classes of the system, and the internals were clean and easy to change. They had been making great progress, and the customer representatives had signed off all the implemented features on the basis of the passing acceptance tests.

But the acceptance tests did not run end-to-end—they instantiated the system’s internal objects and directly invoked their methods. The application actually did nothing at all. Its entry point contained only a single comment:

```
// TODO implement this
```

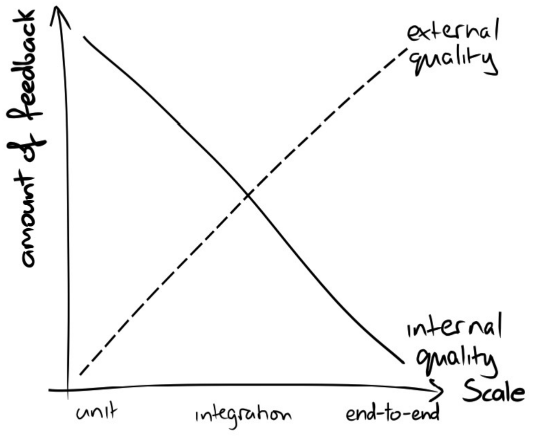
The Importance of Unit Testing: A Horror Story

For example, say you have three layers, each with five possible paths. To test all the paths through the entire application would require 53 end-to-end tests, but you can test all the paths through each unit with only 5·3 unit tests. If you only do end-to-end testing, a lot of paths end up getting neglected, mostly in error handling and boundary conditions.

External vs internal quality

There’s another way of looking at what the tests can tell us about a system. We can make a distinction between external and internal quality: External quality is how well the system meets the needs of its customers and users (is it functional, reliable, available, responsive, etc.), and internal quality is how well it meets the needs of its developers and administrators (is it easy to understand, easy to change, etc.).

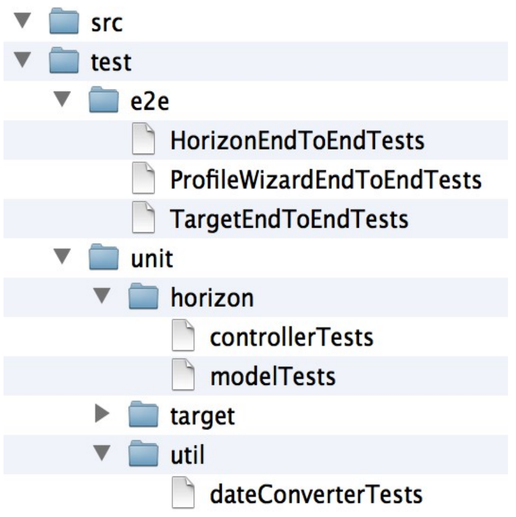
Running end-to-end tests tells us about the external quality of our system, and writing them tells us something about how well we (the whole team) understand the domain, but end-to-end tests don’t tell us how well we’ve written the code. Writing unit tests gives us a lot of feedback about the quality of our code, and running them tells us that we haven’t broken any classes—but, again, unit tests don’t give us enough confidence that the system as a whole works.



Living next to each other

End2End and unit tests

- compliments each other and should exist next to each other
- are n:n related to each other
- a project should have few end2end against many unit tests (the pyramid philosophy)



src
test
e2e
HorizonEndToEndTests
ProfileWizardEndToEndTests
TargetEndToEndTests
unit
horizon
controllerTests
modelTests
target
util
dateConverterTests