Configuration and Hardening

Building a Fortress in a greenfield

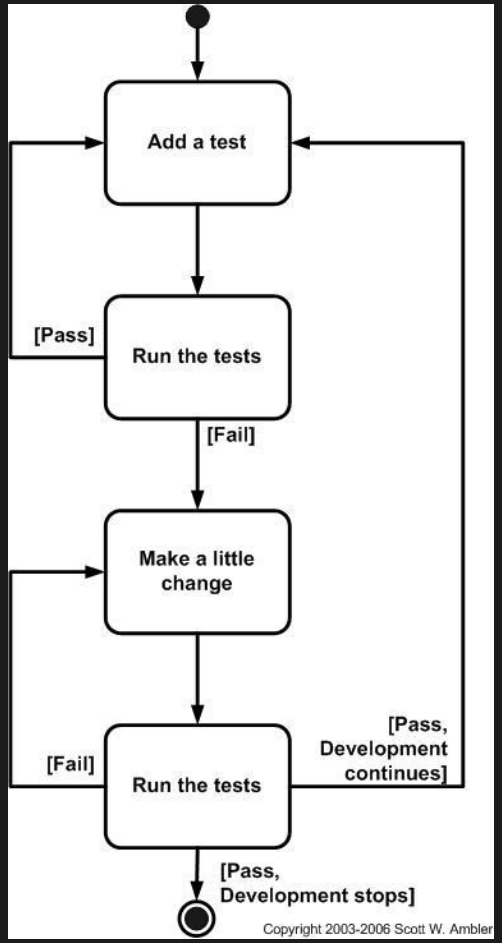
Dr. Hale
University of Nebraska at Omaha
Secure Web Application Development – Lecture 6

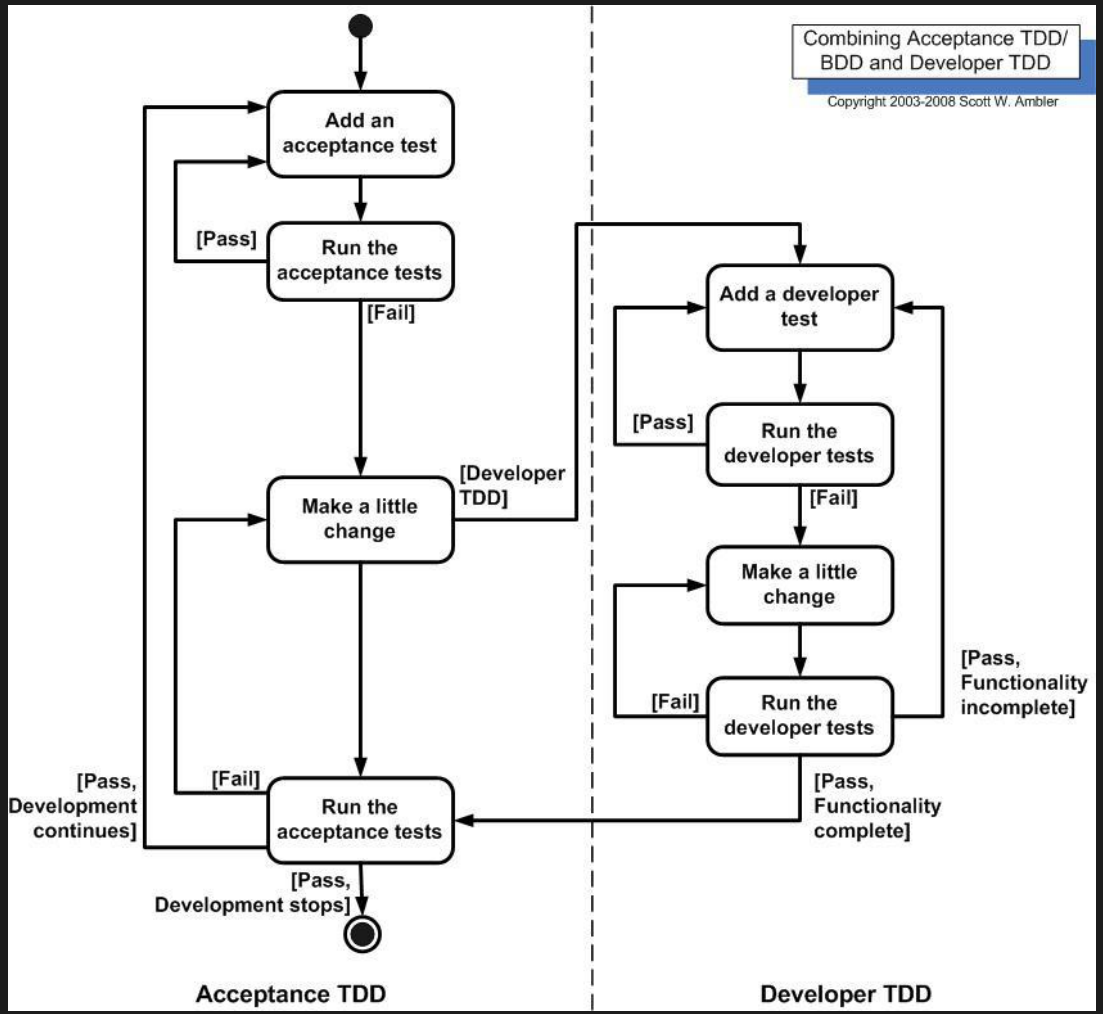
Today's topics:

Test driven development

- Unit testing
- Acceptance Criteria
- Think-test-build-test-repeat

Core Philosophy




Combining Acceptance TDD/ BDD and Developer TDD
Copyright 2003-2008 Scott W. Ambler
Add an acceptance test
[Pass]
Run the acceptance tests
[Fail]
Make a little change
[Developer TDD]
[Pass, Development continues]
[Fail]
Run the acceptance tests
[Pass, Development stops]
Add a developer test
[Pass]
Run the developer tests
[Fail]
Make a little change
[Fail]
Run the developer tests
[Pass, Functionality incomplete]
[Pass, Functionality complete]
Acceptance TDD
Developer TDD

Terminology

Acceptance Test – A measure that ensures that a feature meets functional demands. Usually acceptance tests are tied to user stories or use cases.

Unit test – A smaller test that ensures isolated chunks of functionality (known as units) are functional and operating as expected.

Integration tests – Between unit tests and acceptance tests. Focuses on ensuring that different units function together (said to be integrable).

UNIT Testing

Can be done manually or programmatically – MUCH easier to do the latter – since your components may change and manually testing each time is onerous

Basically you boil down exactly what a feature or component should be doing and you logically state these criteria. Each time you modify the feature/component you run the unit tests to see if they pass. When they all pass you move on to integration tests.

Integration Testing

Can be done manually or programmatically – MUCH easier to do the latter – since your components may change and manually testing each time is onerous

Here you define how different components need to interact and state those constraints logically. When all of the integration tests work – it means you move on to acceptance tests and make sure the collected components satisfy the original goals in the user story or use cases.

Acceptance Testing

Can be done manually or programmatically – easier to do the latter

You basically define the set of all acceptance tests related to your user stories and use cases and – when you demonstrate the app passes all of the tests you are done!

Testing in Ember

Testing in Ember

Acceptance Test – ember g acceptance-test <user story name>
e.g. login
(in /tests/acceptance/login-test.js)

```
import { test } from 'qunit';
import moduleForAcceptance from 'people/tests/helpers/module-for-acceptance';

moduleForAcceptance('Acceptance | login');

test('visiting /login', function(assert) {
  visit('/login');
  fillIn('input.name', 'testuser');
  fillIn('input.password', 'test');
  click('button.submit');

  andThen(function() {
    assert.equal(currentURL(), '/home');
  });
});
```

This will go to /login, check to make sure the url loads, then enter username and password, click submit, and finally check to make sure the app transitions to /home (i.e. expected by our app if the user logs in)

Testing in Ember

UNIT Test – created for the component, model, controller etc whenever you generate a new feature in ember.
See https://guides.emberjs.com/v2.8.0/testing/unit-testing-basics/

Why should I reinvent the wheel when their example is good

Testing in Ember

Integration Test – created for the component, model, controller etc whenever you generate a new feature in ember.
Just set integration: true if you want to enable integration testing.
Otherwise it is similar to unit testing (i.e. it takes place in the same test file generated by ember for you, only in /integration instead of /unit.

Testing in Django

Testing in Django

You will mostly just want to use unit tests in Django to check different routes and models (especially validators). Python has a default unittest framework. Django includes this framework in the module django.test
It is pretty simple:

```
from django.test import TestCase
from myapp.models import Post

class PostTestCase(TestCase):
    def setUp(self):
        Post.objects.create(title=“test title", content=“some text“, likes=5)

    def test_post_likes_feature(self):
        """“Show likes for a post"""
        post = Post.objects.get(name=“test title")
        self.assertEqual(lion.likes(), 5)
```

Testing in Django

The other important feature in django is the Client class in django.test that lets you interact with your app by making requests.

See: https://docs.djangoproject.com/en/1.10/topics/testing/tools/

Questions?

Matt Hale, PhD

University of Nebraska at Omaha

Interdisciplinary Informatics

mlhale@unomaha.edu

Twitter: @mlhale_