
Wouter Lagerweij
@wouterla
16.5M Bikes
~1 Bike/person
public void testMultiplication() {
    Dollar five = new Dollar(5);
    five.times(2);
    assertEquals(10, five.amount);
}

class Dollar {
    private int amount;

    Dollar(int amount) {
        this.amount = amount;
    }

    void times(int multiplier) {
        return amount * multiplier;
    }
}
public void testThatItDoesSomething() {
    Eh...
}
● Slow, unreliable unit tests
● Slower Selenium tests
● Messy, duplicated code
● Manual deployments
● Production issues
- TDD
- BDD
- coding katas
- pairing
- training w/ Chet
- Improve test run time
- 100% coverage of new code
- refactor in the small (boy scout rule)
- refactor in the large (planned improvements)
Unit test coverage:

- Feb: 1.7%
- May: 2.1%

Total: 25%
Rebuild?
So...

- Deliver value from *day one*
- Don’t rebuild the same mess
- Don’t rebuild using the same process
- Don’t rebuild the same functionality
In Practice

- Architecture: Strangler Pattern
- Process: Continuous Delivery
- The First Sprints
Architecture: before

request for website A
request for website B
request for website C
request for website D
request for website E

Legacy system

DB
Incoming request for new page

website A

New Page

Service

Legacy system

Incoming request
Process

- 100% unit-test coverage
- TDD all the way
- All functionality defined by BDD scenarios
- Fully automated deployment

- Every push goes to production
Incoming request for other pages

Incoming request for job detail page

website A

Job Detail Page

Job Service

Legacy

DB
Sprint 1
Sprint 1: Proxy

Goal:

Get a new front-end component up that proxies all traffic to the old system but serves ‘hello world’ if we ask nicely.
Sprint 1: Proxy

- Create a new environment, in this case on Amazon’s AWS
- Create Ansible scripts to provision a basic infrastructure:
  - jenkins
  - sonar
  - Docker nodes for test, acceptance and production
- Set up a VPN to the existing (hosted somewhere else) environment
- Create a new codebase for a (php, symfony) component
- Create build scripts for the new component
- Create a hello world page for the URL that pointed to the job detail page
- Create a proxy so that all calls would be delegated to the old system (Apache vhost.conf)
- Create a feature toggle so we could override the proxy based on a cookie (Apache vhost.conf)
- Package our new component as a Docker image, and figure out how to deploy it
- Use a Docker registry to store and version out deployment packages
- Create Jenkins Job Builder configuration to set-up a delivery pipeline for our new web front end
- Create a basic smoke test for the component to check successful deployment in the pipeline
- Set-up the AWS loadbalancers for all the environments
- Make sure New Relic could be configured on all our nice new Docker containers
- Route all traffic for our first website through our new infrastructure
Sprint 1: Proxy

- Create a hello world page for the url that pointed to the job detail page
Sprint 2: Detail Page

Goal:

Get a fully functional Job Detail Page replacement working behind the feature toggle
Sprint 2: Detail Page

- Create Job Service component, including full pipeline
Incoming request for other pages

Incoming request for job detail page

website A

Job Detail Page

Job Service

Legacy

DB
Sprint 2: Detail Page

● Create Job Service component, including full pipeline
● Database access to old system’s DB
● Unit testing build steps for php
● Sonar (static analysis) deployment and build steps
● Contract test for Job Service
● Client library for Job Service
● Acceptance scenarios for job detail functionality
● Unit testing build steps for javascript
● Sonar setup for javascript
● Implement Just Enough to Read a Job
● Create a nice looking front-end...
Done?

1,1 Sec
In one month

- legacy system
- few tests
- manual deploys
- 1 release per week
- timid team

- decoupled services
- 100% coverage
- full automation
- 30 releases per day
- courage
Problems
Thank You!

@wouterla
Wouter Lagerweij