

Software Testing Foundations

Internal presentation

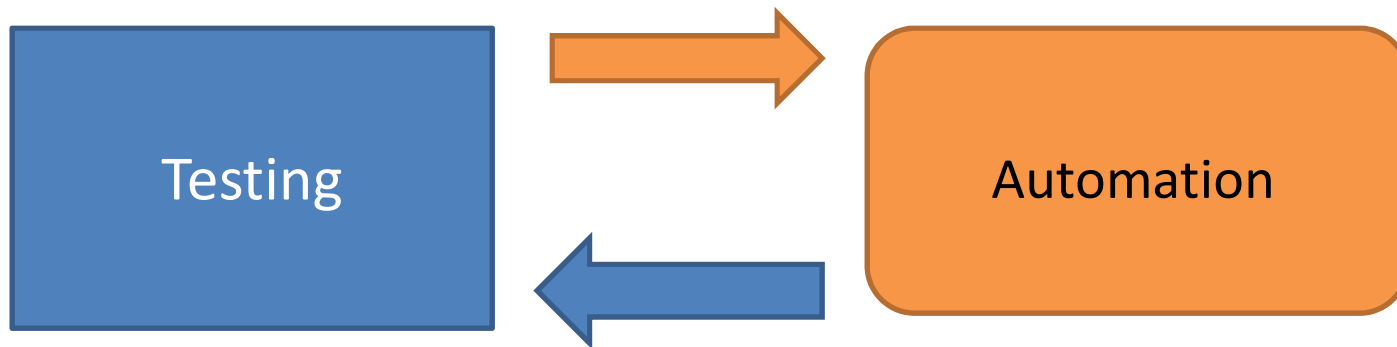
February 2019

Kostis Kapelonis

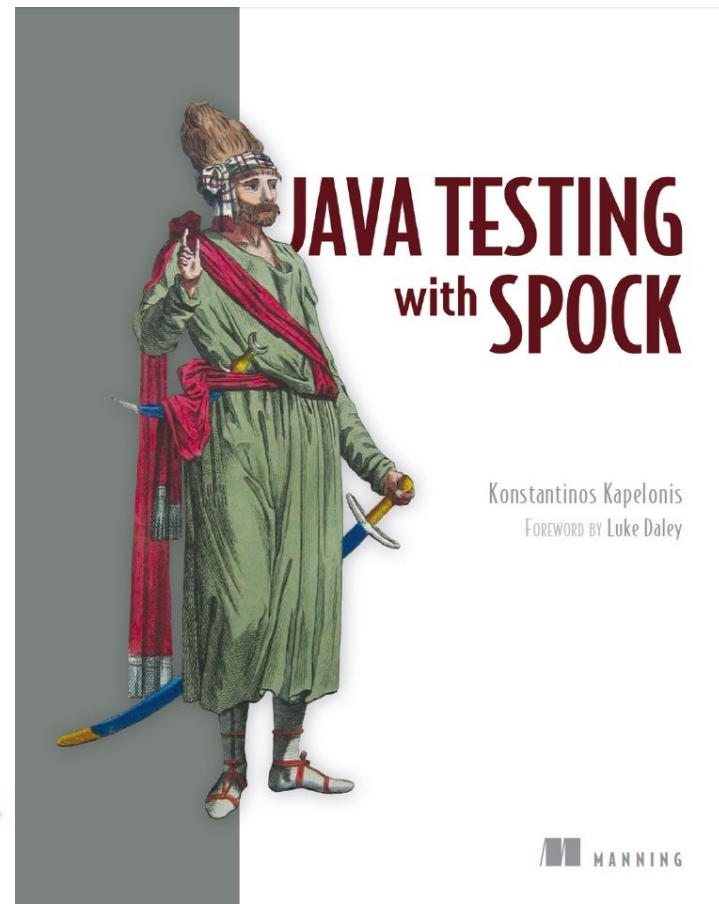
Strong foundations?



Things I love



Things I write



 Roger

★★★★★ **Changed how we do software testing**

July 26, 2017

Format: Paperback | **Verified Purchase**

This book was great in teaching how and why to use Spock for testing. We have since built our testing methodologies around Spock based on techniques learned from this book. Our non-technical staff finds Spock tests much easier to understand than straight JUnit. This book was very readable and had very good examples.

Things I did



BlizzardCS @BlizzardCS · Oct 15

These issues have been resolved. Thanks for hanging in there, folks!

BlizzardCS @BlizzardCS

[#Bnet] Update: Some features have been restored but our teams are still investigating issues affecting chat and Blizzard forums. Thanks for hanging in there everyone :) [twitter.com/BlizzardCS/sta...](https://twitter.com/BlizzardCS/status/1386111111)

5 7 83



BlizzardCS @BlizzardCS · Oct 15

[#Bnet] Update: Some features have been restored but our teams are still investigating issues affecting chat and Blizzard forums. Thanks for hanging in there everyone :)

BlizzardCS @BlizzardCS

[#Bnet] We are currently investigating reports of issues connecting to and using various social features. Thank you for your patience!

9 10 69

Things I blog

Software Testing Anti-Pattern List

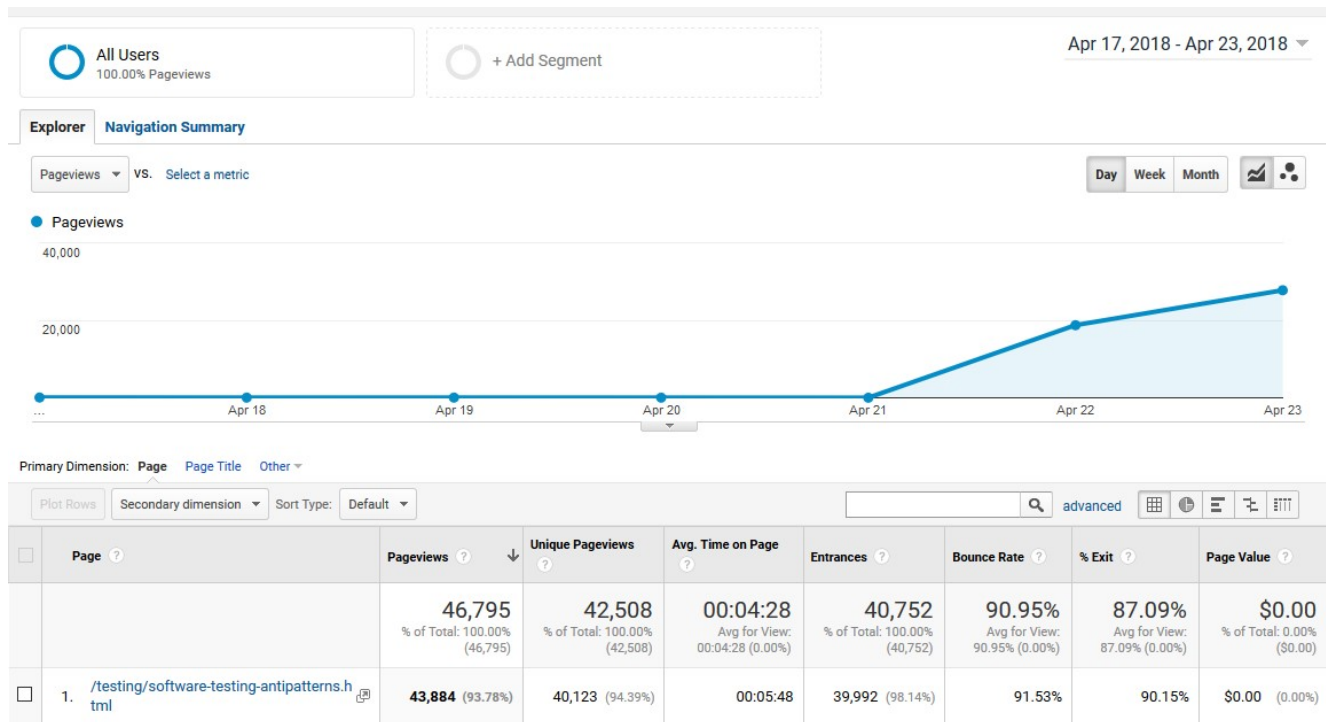
1. Having unit tests without integration tests
2. Having integration tests without unit tests
3. Having the wrong kind of tests
4. Testing the wrong functionality
5. Testing internal implementation
6. Paying excessive attention to test coverage
7. Having flaky or slow tests
8. Running tests manually
9. Treating test code as a second class citizen
10. Not converting production bugs to tests
11. Treating TDD as a religion
12. Writing tests without reading documentation first
13. Giving testing a bad reputation out of ignorance

<http://blog.codepipes.com/testing/software-testing-antipatterns.html>

Things I blog

Y Hacker News new | threads | comments | show | ask | jobs | submit

* Software Testing Anti-patterns (codepipes.com)
465 points by kkapelon 6 months ago | hide | past | web | favorite | 166 comments



<https://news.ycombinator.com/item?id=16894927>

Current Work



Docker based CI/CD
solution for Helm/
Kubernetes
deployments

Current Work

codefresh

Pipeline Name
Release a new update to prod. Must be apdafadsf asdsd...

Documentation Support [TRIGGER PIPELINE](#)

COMPLETED STEPS 12 [VIEW YAML](#) START TIME 1/8/2018 22:22 DURATION 10m TRIGGER COMMIT on Idan's Gitlab - codefresh-io/sf-secrets by Idan Arbel [DOWNLOAD LOG](#)

Initialization 2.43s

BUILD →

- GIT CLONE Clonning main repository 2.43s
- GIT CLONE Clonning main repository 2.43s
- GIT CLONE Clonning main repository 2.43s
- GIT CLONE Clonning main repository 2.43s

BUILD →

- GIT CLONE Clonning main repository 2.43s
- GIT CLONE Clonning main repository 2.43s
- GIT CLONE Clonning main repository 2.43s
- GIT CLONE Clonning main repository 2.43s

UNIT →

- GIT CLONE Clonning main repository
- GIT CLONE Clonning main repository

[Restart from Step](#)

Select pipeline step to view details



Current Work



Docker Tutorial | June 20, 2018

Using Docker from Maven and
Maven from Docker



Kostis Kapelonis



<https://codefresh.io/blog/>

<https://codefresh.io/features/>

Part 1 – Airport Management



Airport Management 101

Metrics for effective airport management?

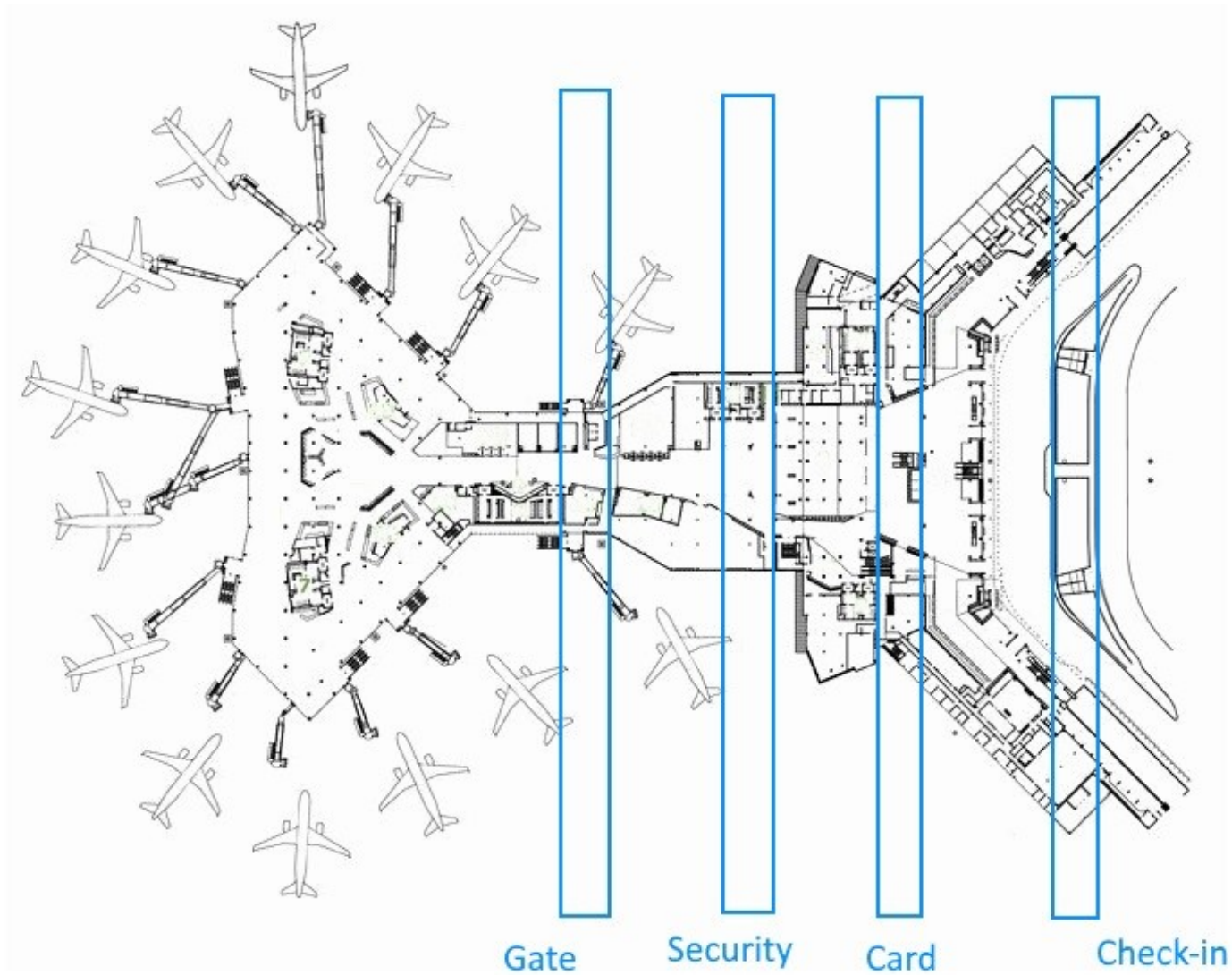


Airport Management 101

- Flights on time (100%)
- Bombs/threats/security incidents (0%)



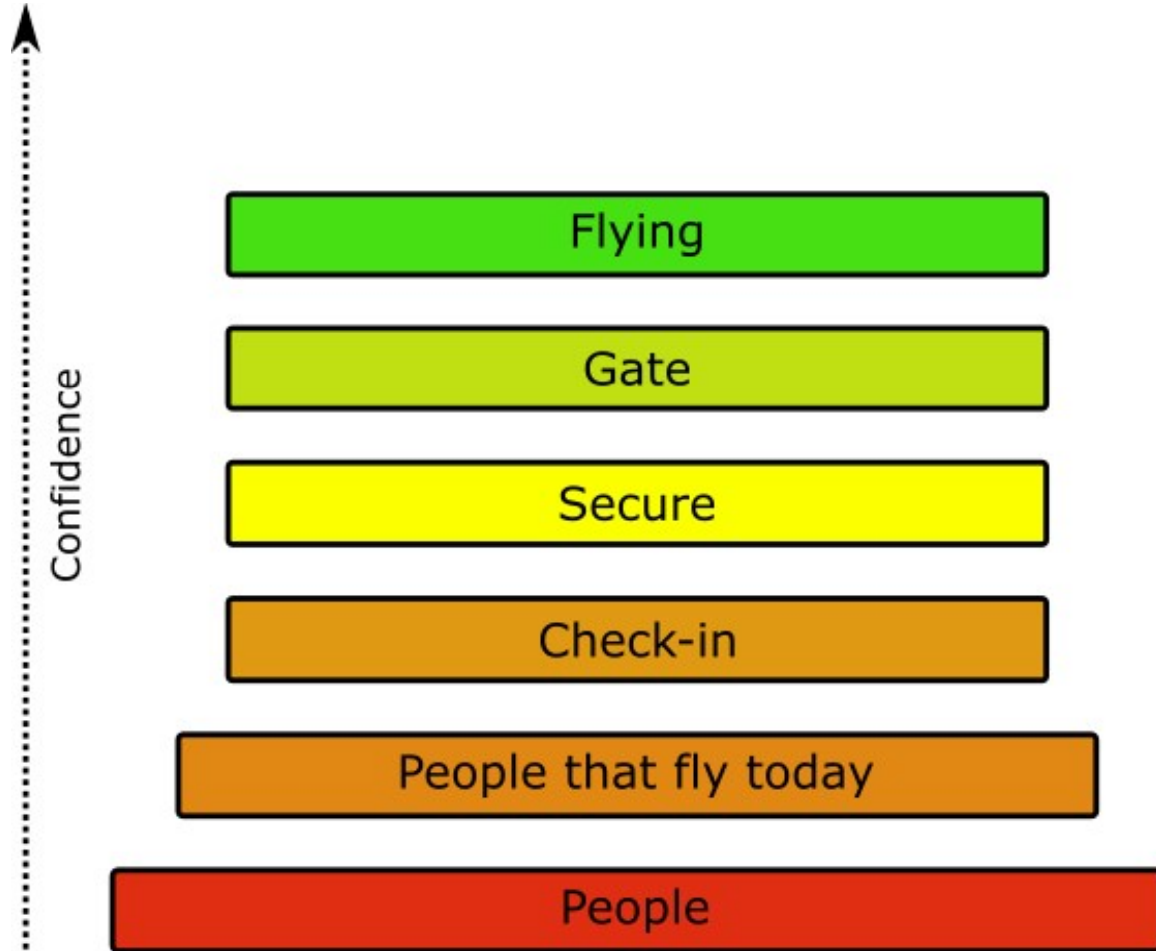
Airport structure



Categories

- People in Airport
- People in Airport that actually travel
- People in Airport that travel today
- People that have checked-in
- People that travel domestic/international
- People that have passed security
- People that are at the gate

Gradual confidence



Airport – Key Points

People pass checks to go to
“higher” categories



Airport – Key Points

The person at the gate just verifies your identify

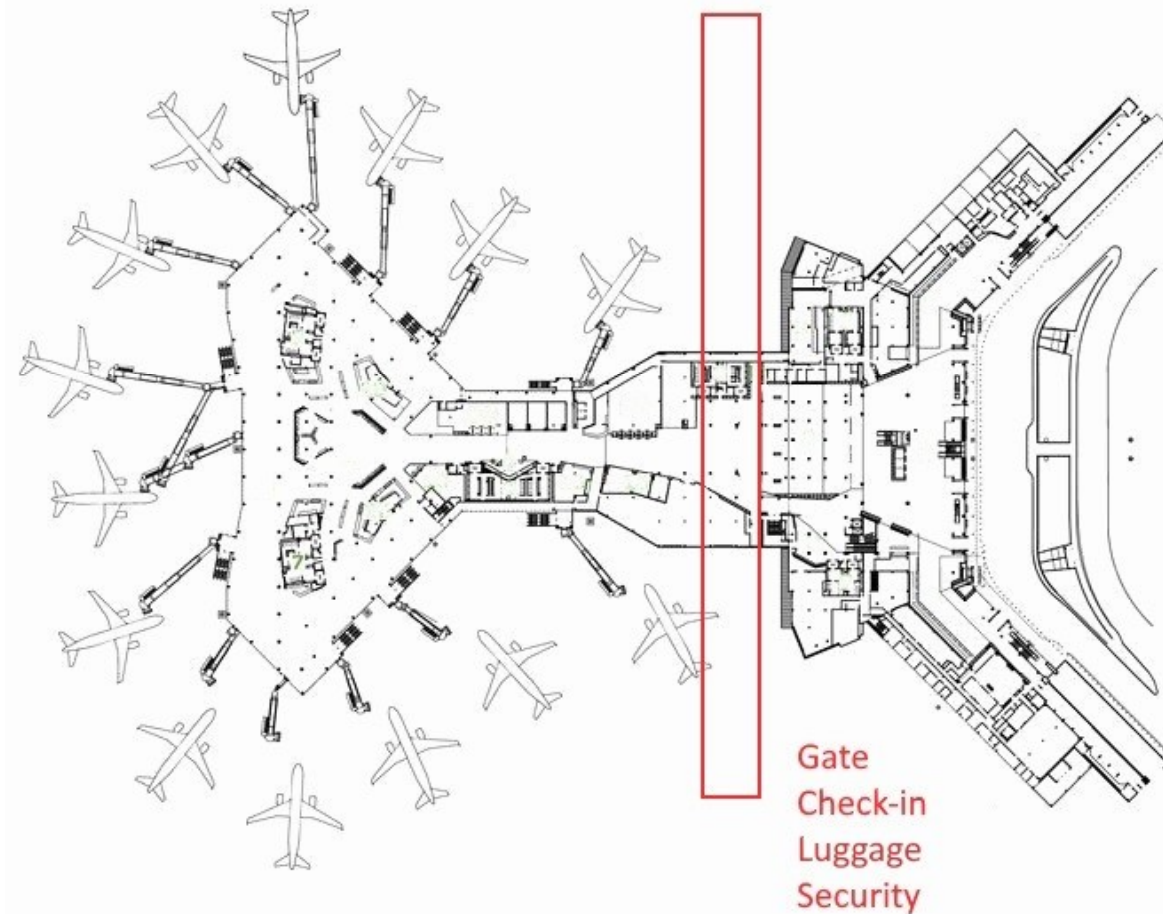


Airport – Antipattern 1

Single border for all checks



Airport – Antipattern 1



Airport – Antipattern 1



Airport – Antipattern 2

One flight per week, therefore
all security is “easy”!

Airport – Antipattern 2



Part 2 – Software development



Software development

Metrics for effective software development



Software development

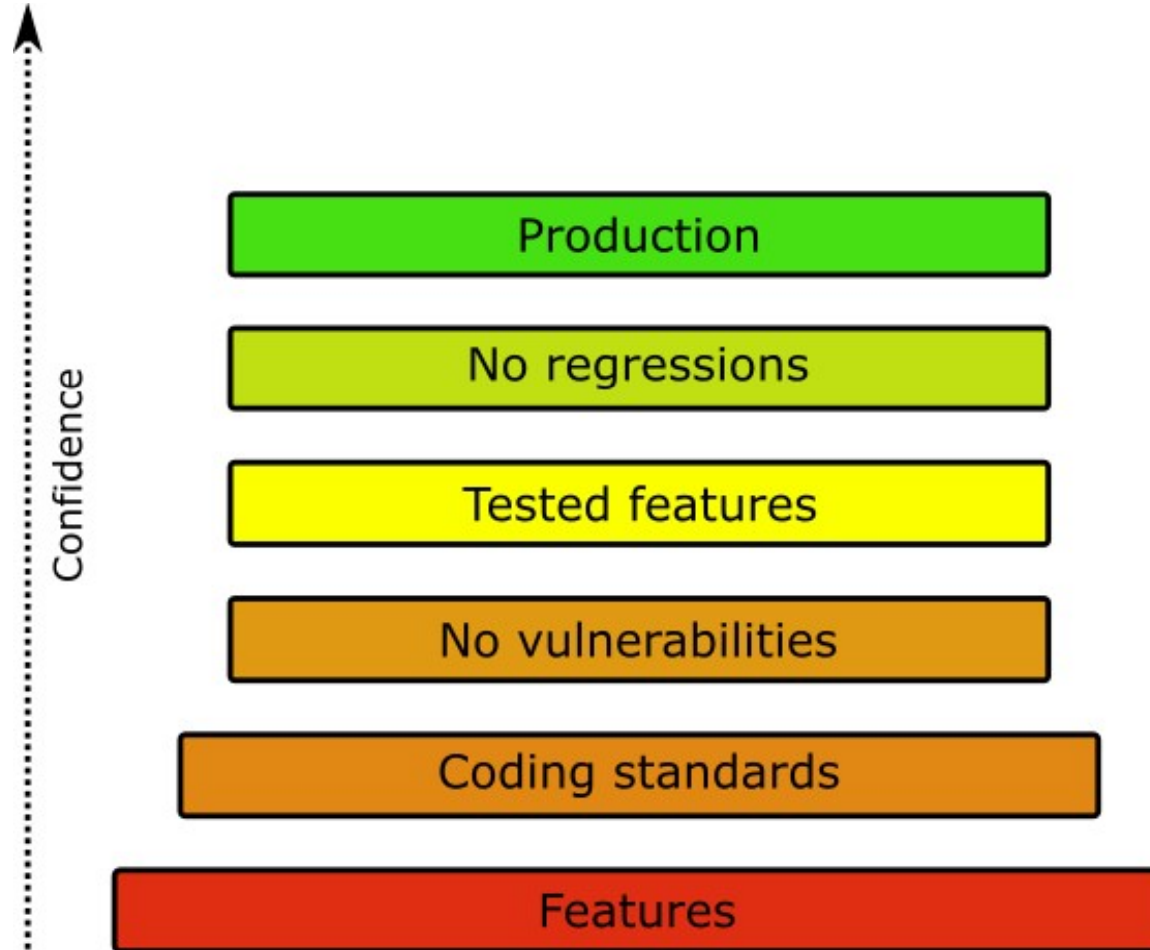
- Features that reach production(100%)
- Failed deployments (0%)



Categories

- Features that work on my workstation
- Features that work on any workstation
- Features that only work on their own
- Features that work with other features
- Features that have memory leaks
- Features that have bugs
- Features with security vulnerabilities

Software Quality



Quality – Key Points

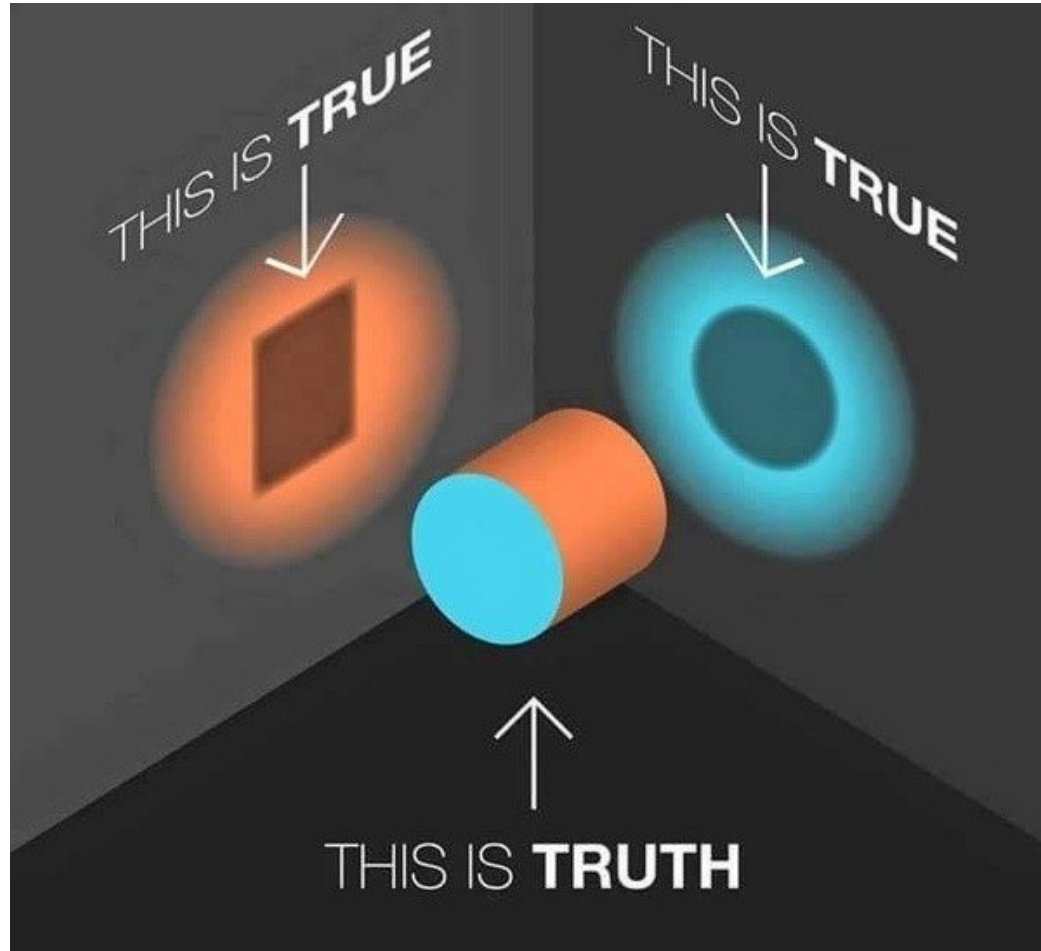
Features are passing multiple checks (CI/CD pipeline)



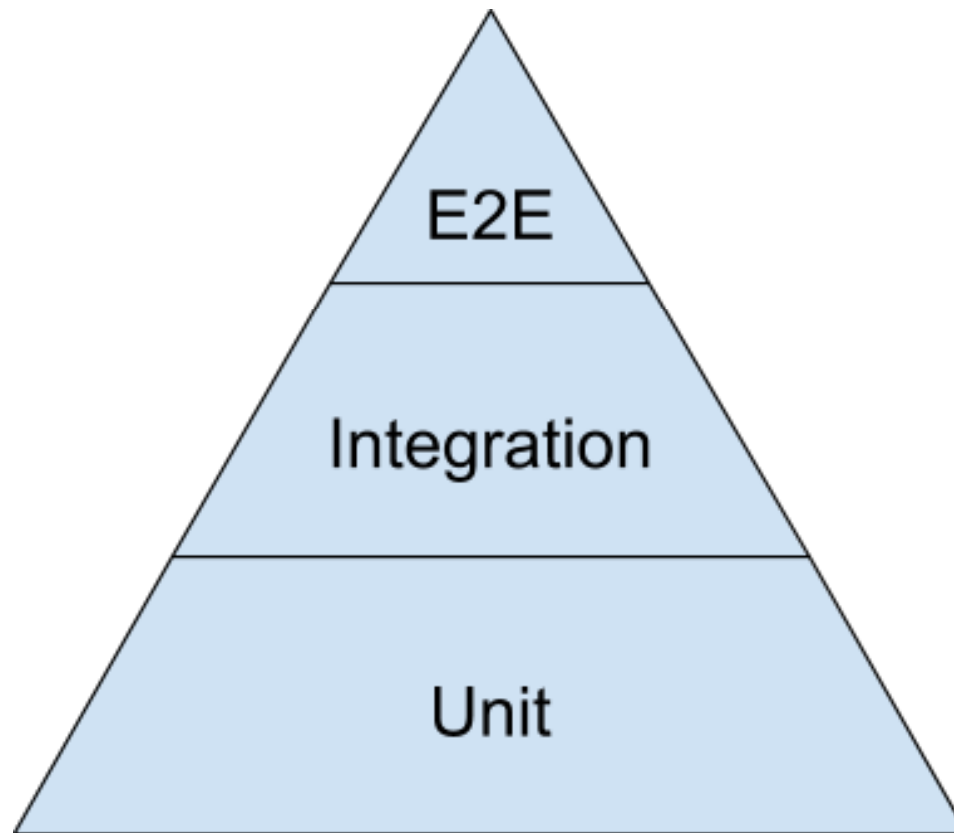
Anti-patterns

- Only one test suite (you need at least three)
- Manual tests
- Deployments happen once per week/month

Some definitions



Testing pyramid



Unit tests

- Require ONLY source code
- Everything that is external is mocked
- Mainly involve business logic testing
- Focus is on a single method/class
- Run with xUnit or similar framework
- Easy to setup and run
- Fast (20- 500ms)

Unit test example

```
Basket basket = new Basket()
```

```
basket.add("Samsung 4k TV", 600)
```

```
basket.add("Sony PS4", 300)
```

```
basket.getValue() == 900
```

Integration/Service/Component test

- Uses a database
- Uses the network to call another component
- Uses a queue/webservice
- Reads/writes files, performs I/O
- Needs the application to be deployed (even partially)
- Can be complex to setup and run
- Slow (seconds or even minutes)

Integration test example

```
Basket basket = new Basket(...)
```

```
Customer customer = new Customer(...)
```

```
customer.checkout(basket, cc, inventory)
```

Assert invoices, cc charge, inventory subtraction
etc.

Maven lifecycle

<code>compile</code>	compile the source code of the project.
<code>process-classes</code>	post-process the generated files from compilation, for example to do bytecode enhancement on Java classes.
<code>generate-test-sources</code>	generate any test source code for inclusion in compilation.
<code>process-test-sources</code>	process the test source code, for example to filter any values.
<code>generate-test-resources</code>	create resources for testing.
<code>process-test-resources</code>	copy and process the resources into the test destination directory.
<code>test-compile</code>	compile the test source code into the test destination directory
<code>process-test-classes</code>	post-process the generated files from test compilation, for example to do bytecode enhancement on Java classes. Fo
<code>test</code>	run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
<code>prepare-package</code>	perform any operations necessary to prepare a package before the actual packaging. This often results in an unpack (2.1 and above)
<code>package</code>	take the compiled code and package it in its distributable format, such as a JAR.
<code>pre-integration-test</code>	perform actions required before integration tests are executed. This may involve things such as setting up the require
<code>integration-test</code>	process and deploy the package if necessary into an environment where integration tests can be run.
<code>post-integration-test</code>	perform actions required after integration tests have been executed. This may including cleaning up the environment.
<code>verify</code>	run any checks to verify the package is valid and meets quality criteria.

UI test

Anything that works with the DOM of the Web page

- Geb
- Protractor
- Selenium
- Cypress
- Karma

Following the pyramid

- We need unit and integration tests and UI tests
- Having only one type is an anti-pattern
- Each test suites run in different CI/CD phase

Other test categories

- Load testing
- Security testing
- Smoke testing (in production)
- Regression testing
- External service provider testing

Part 3 – Software pipeline



Quiz:

How many steps do you need to setup and run your whole test suite?

Wrong answers

1. Prepare database
2. Edit settings file
3. Prepare test environment
4. Run tests
5. Cleanup environment

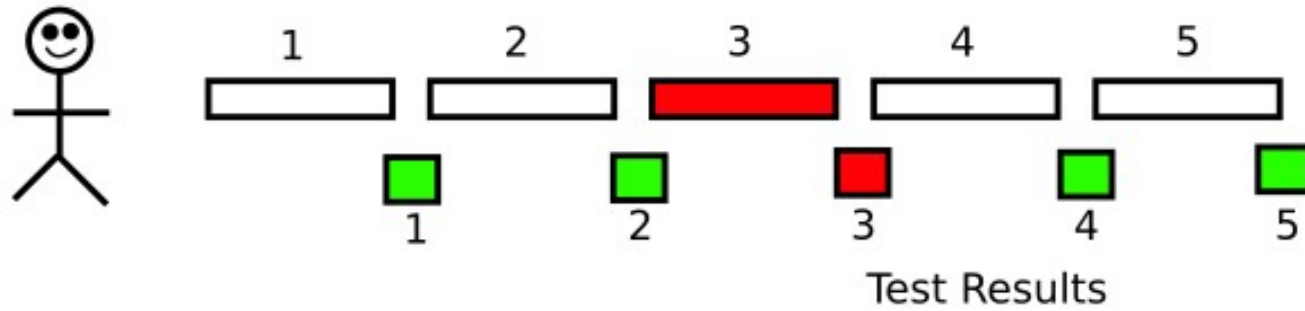
Correct answer

- Before commit: single command to run tests
- After commit: Tests run automatically, with no human intervention
- Only CI/CD server is running tests post-commit

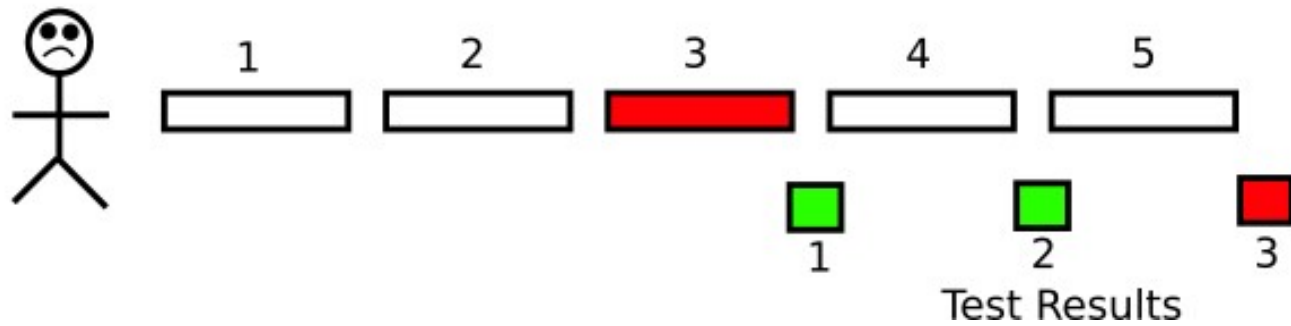
Correct answer

Dev

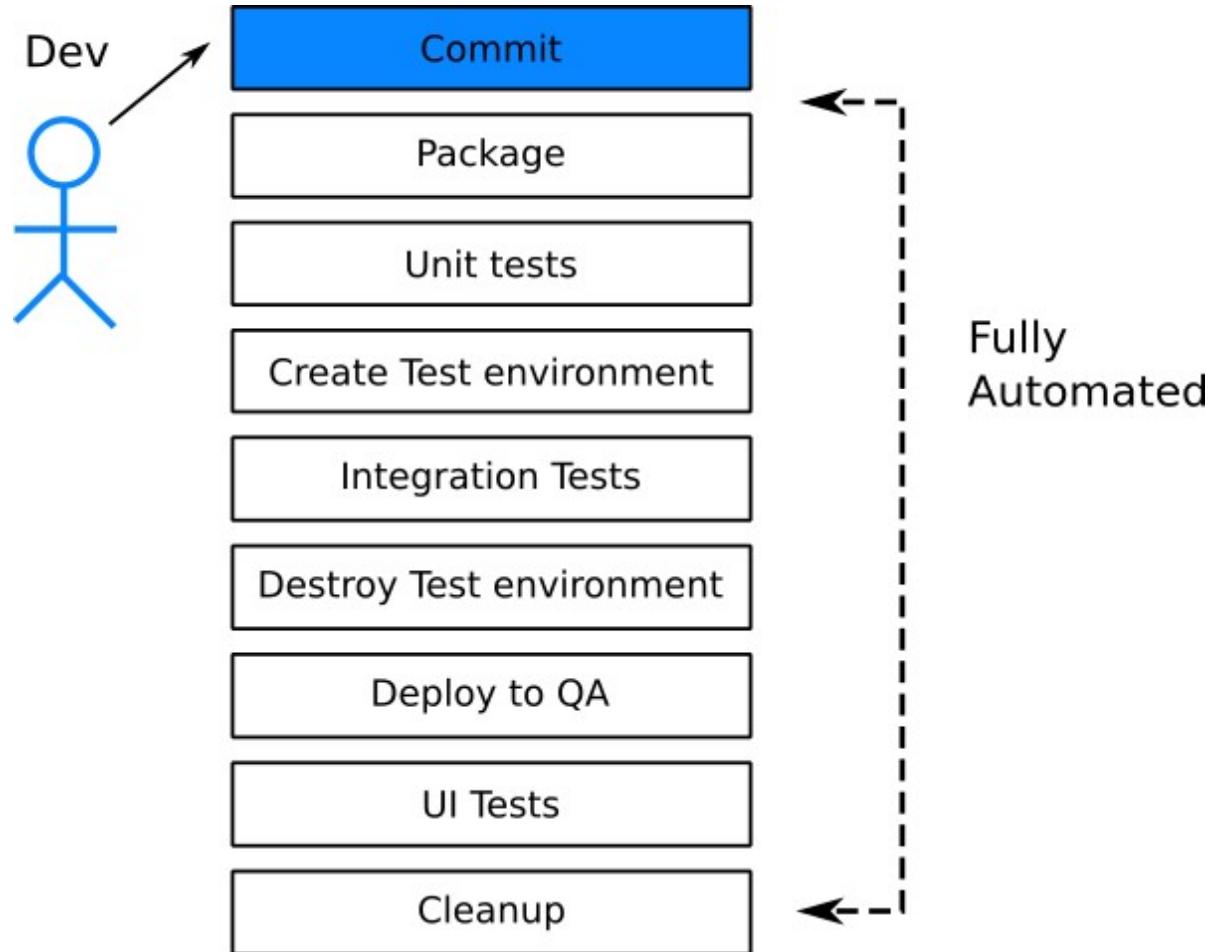
Features



Time



Testing strategy



Testing strategy

- Avoid manual steps/checklists
- Make local testing easy for developers
- CI server should run test for each feature branch in a transparent manner
- You should also have smoke/acceptance/production tests
- The CI/CD server runs tests post-commit, NOT humans

End goal

Developers insert their
features between testing
phases

Work like this – pipeline is always on



Avoid this – manual processes



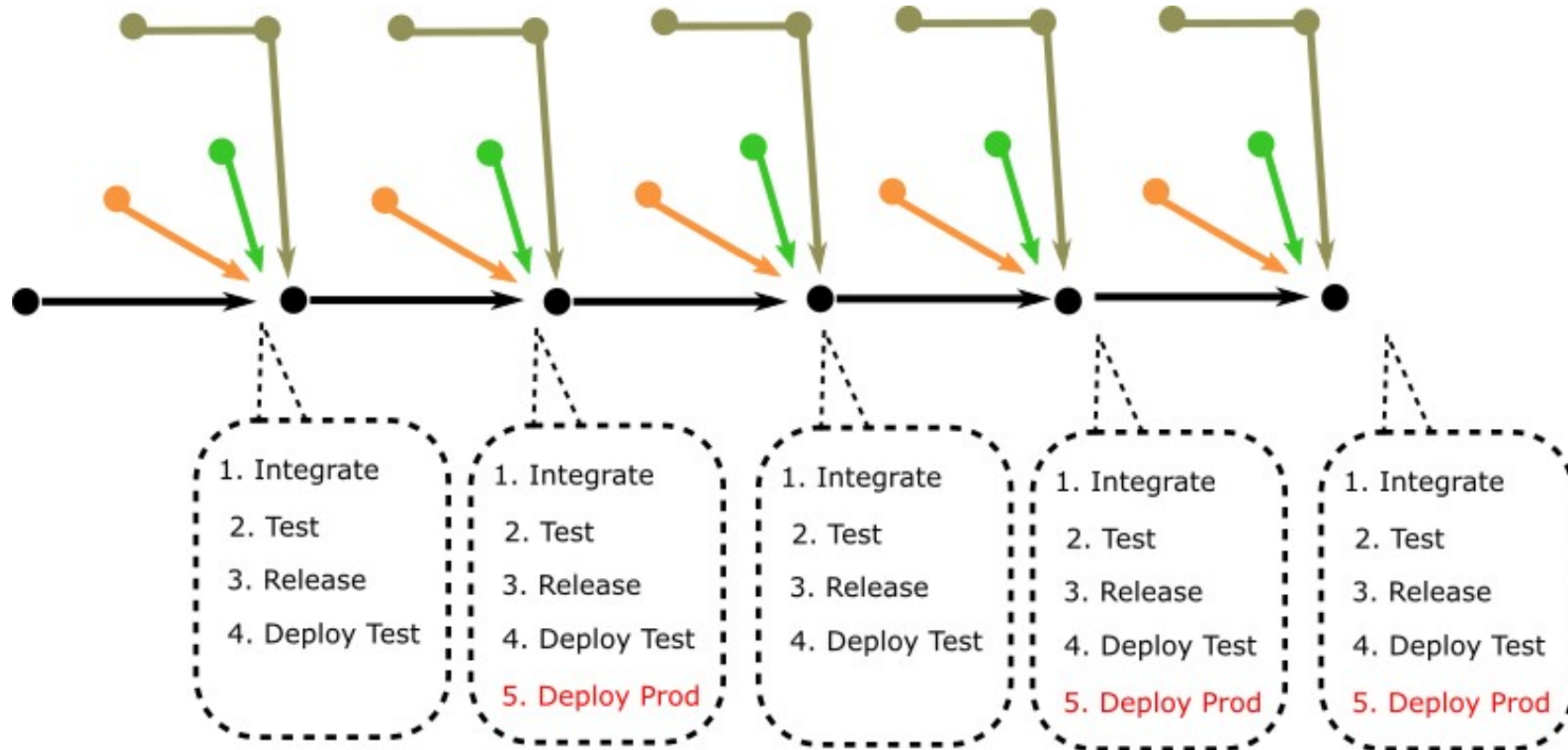
Deployment frequency

- Low (once per week or per month)
- Medium (multiple times per week)
- High (multiple times per day)
- Ultra High (fully automated deployments)

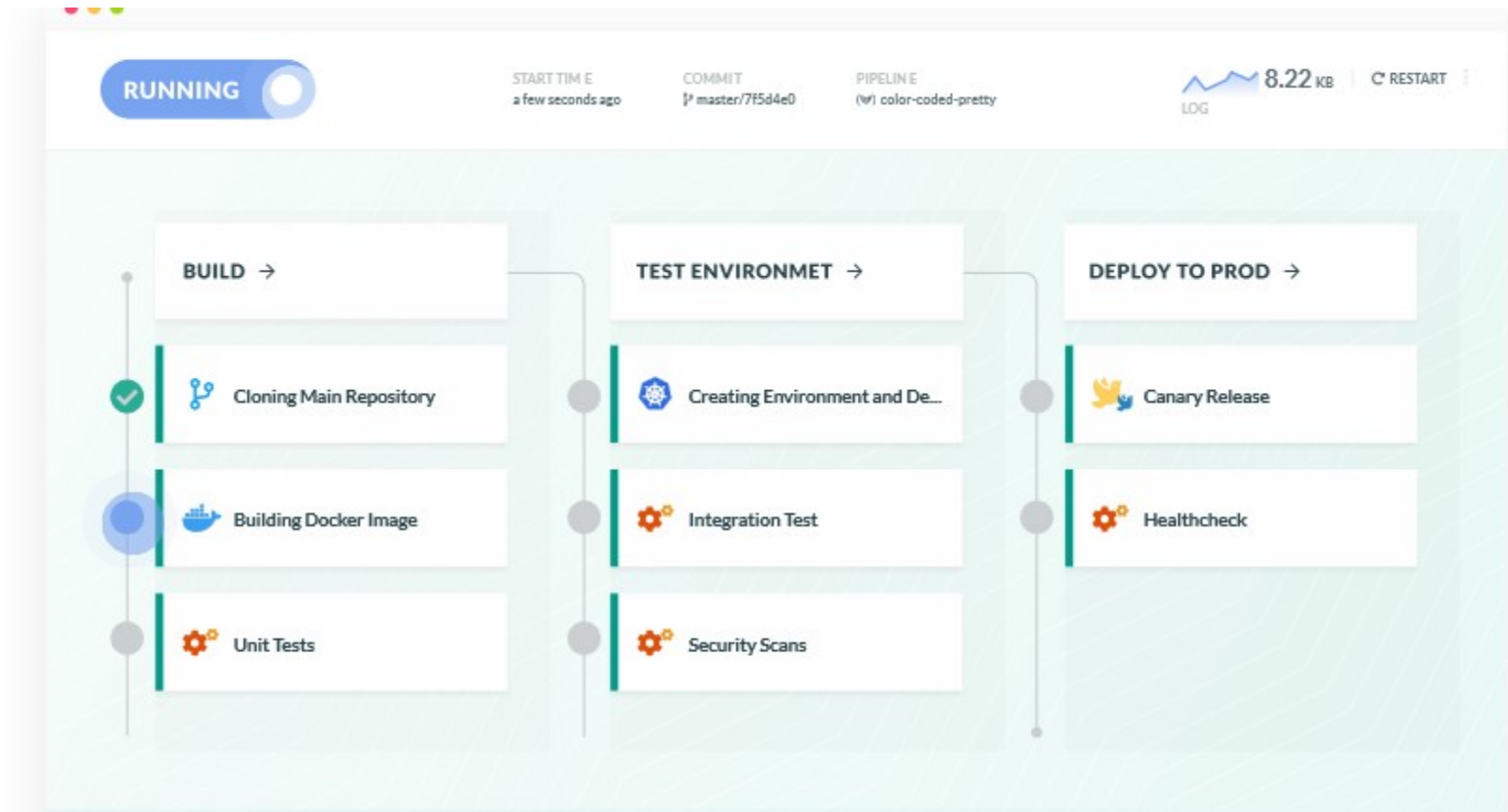
Deployment speeds

- Flickr deploys 10 times per day
- Etsy deploys 50 times per day
- AAA gaming company deploys every 15 min
- Amazon deploys every 11.6 seconds

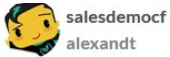
Continuous Delivery



Pipeline to production



Preview environments

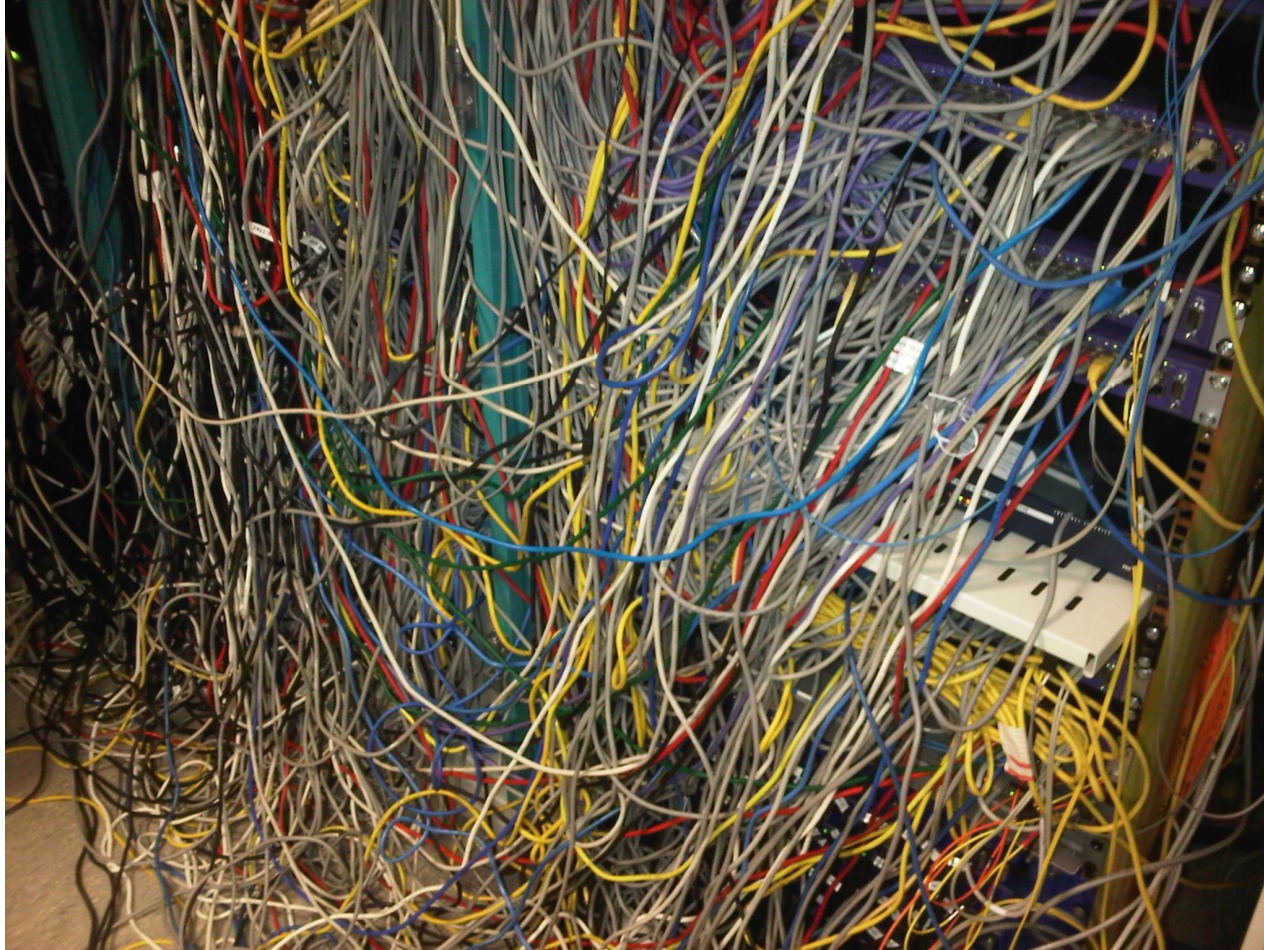
Builds Help UPGRADE ADD REPOSITORY 

COMPLETED STEPS 19/19 START TIME 20 hours ago DURATION 5 min 4 s REPOSITORY salesdemocf/example-voting-... COMMIT new-vote-for-vcs/856fcfd PIPELINE example-voting-app LOG 1.11 MB RESTART

Initializing Process 6 s

Stage	Step Name	Step Type	Duration
DEFAULT →	Cloning main repository...	git-clone	3 s
	GenerateReportsIndexH1	freestyle	2 s
	Upload Clair Reports	freestyle	5 s
	Set Pull Request Environ.	freestyle	3 s
	Set Pull Request Environ.	freestyle	2 s
BUILD →	Building Result Image	build	16 s
	Building Vote Image	build	16 s
	Building Worker Image	build	14 s
	Building Test Image	build	17 s
SECURITY SCANS →	ScanResultImage	freestyle	49 s
	ScanVoteImage	freestyle	45 s
	ScanWorkerImage	freestyle	1 min 44 s
PULL REQUEST →	Create Ephemeral Helm F	freestyle	7 s
	RunPRDVTs	freestyle	2 min 30 s

Part 4 – what to test



Code != file folders

apps	ID3	account
auth	IXR	activities
channels	Requests	admin
mail	SimplePie	attachments
permissions	Text	auth_sources
route	certificates	auto_completes
rss	css	boards
settings	customize	calendars
themes	fonts	common
url	images	context_menus
labs.js	js	custom_field_enumerations
slack.js	pomo	custom_fields
webhooks.js	random_compat	documents
xmlrpc.js	rest-api	email_addresses
	theme-compat	enumerations

Deployment time



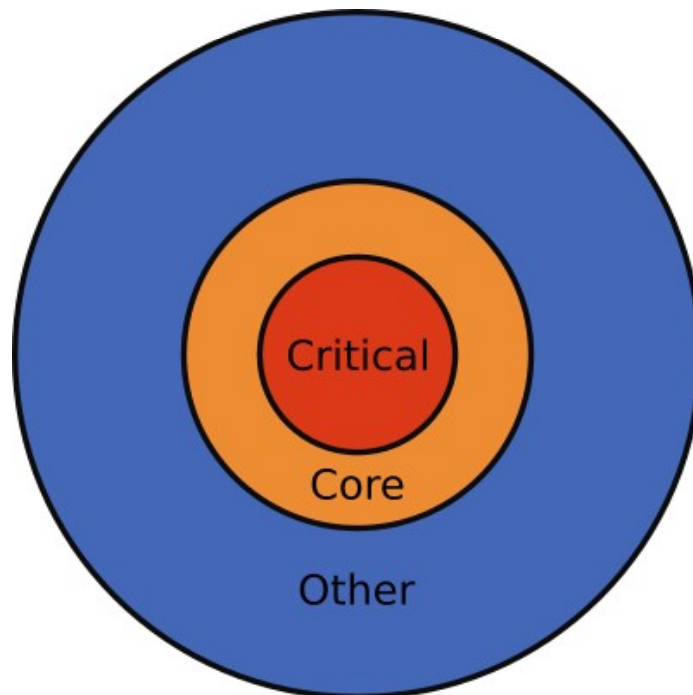
Two bugs after deployment

1. Customers cannot book an apartment
2. Customers get wrong recommendations when they browse apartments.

Obviously first one is critical, second one is not

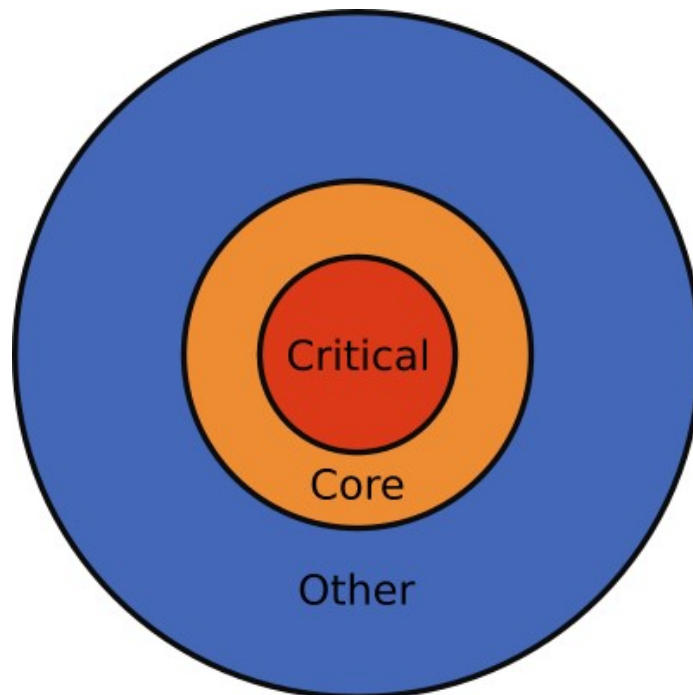
Code severity

Critical code - This is the code that breaks often, gets most of new features and has a big impact on application users



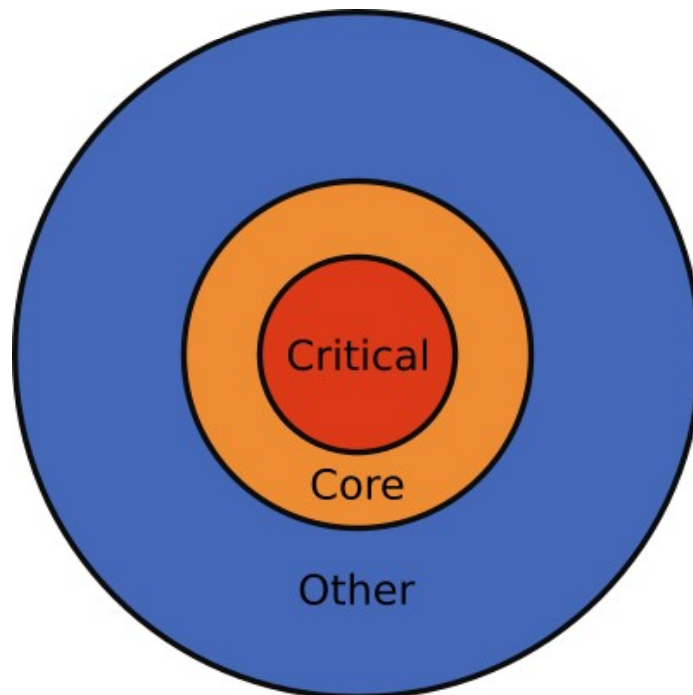
Code severity

Core code - This is the code that breaks sometimes, gets few new features and has medium impact on the application users



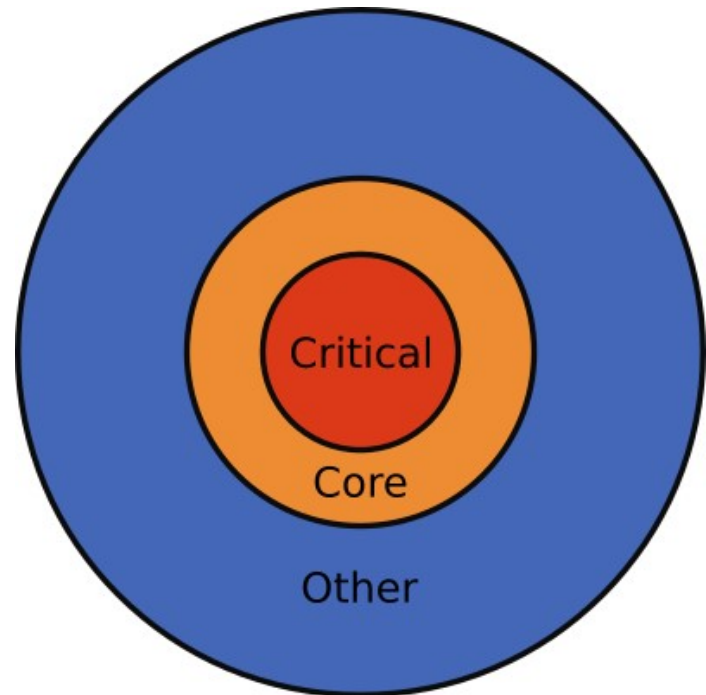
Code severity

Other code - This is code that rarely changes, rarely gets new features and has minimal impact on application users.

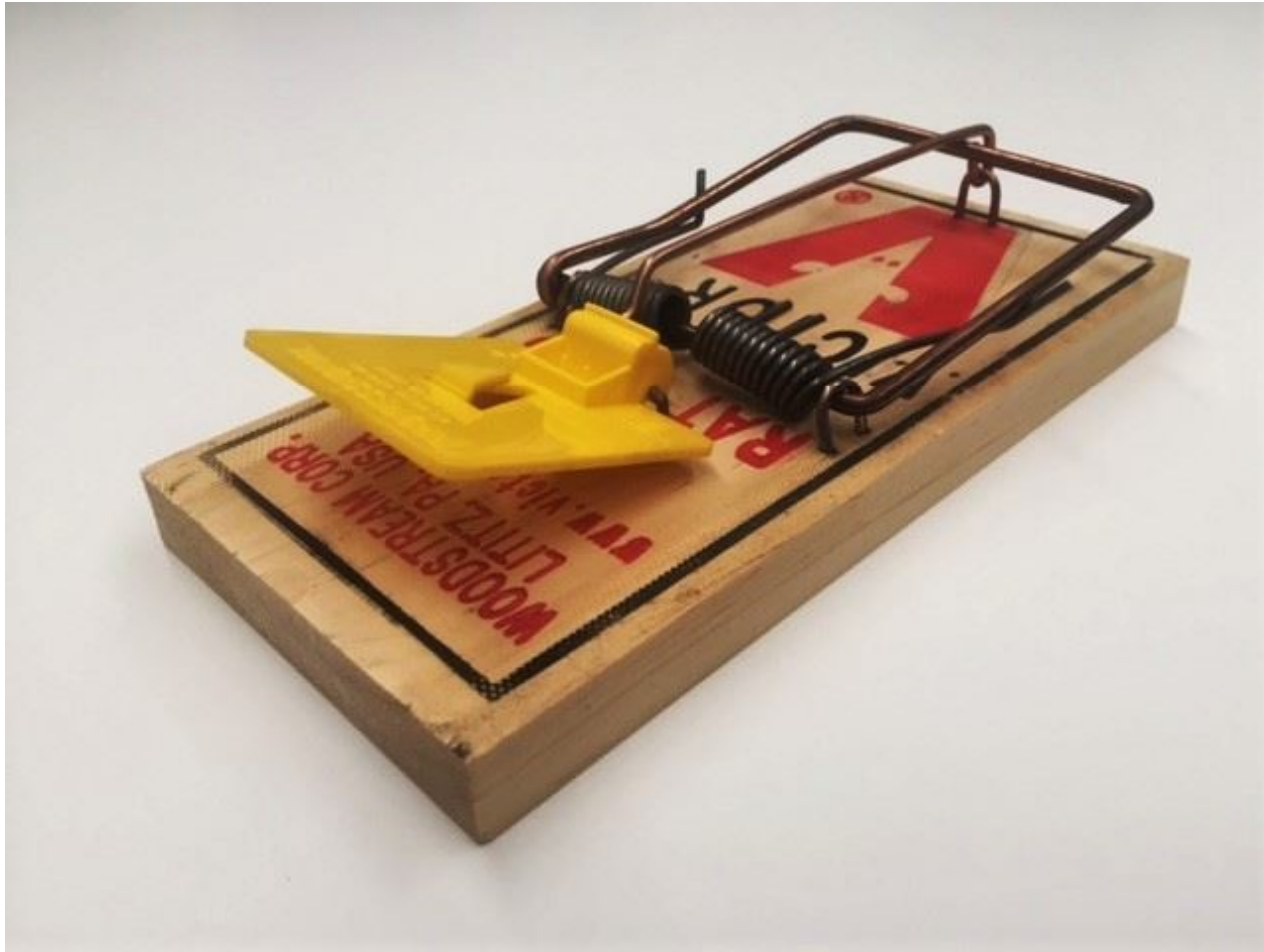


Write tests for code that

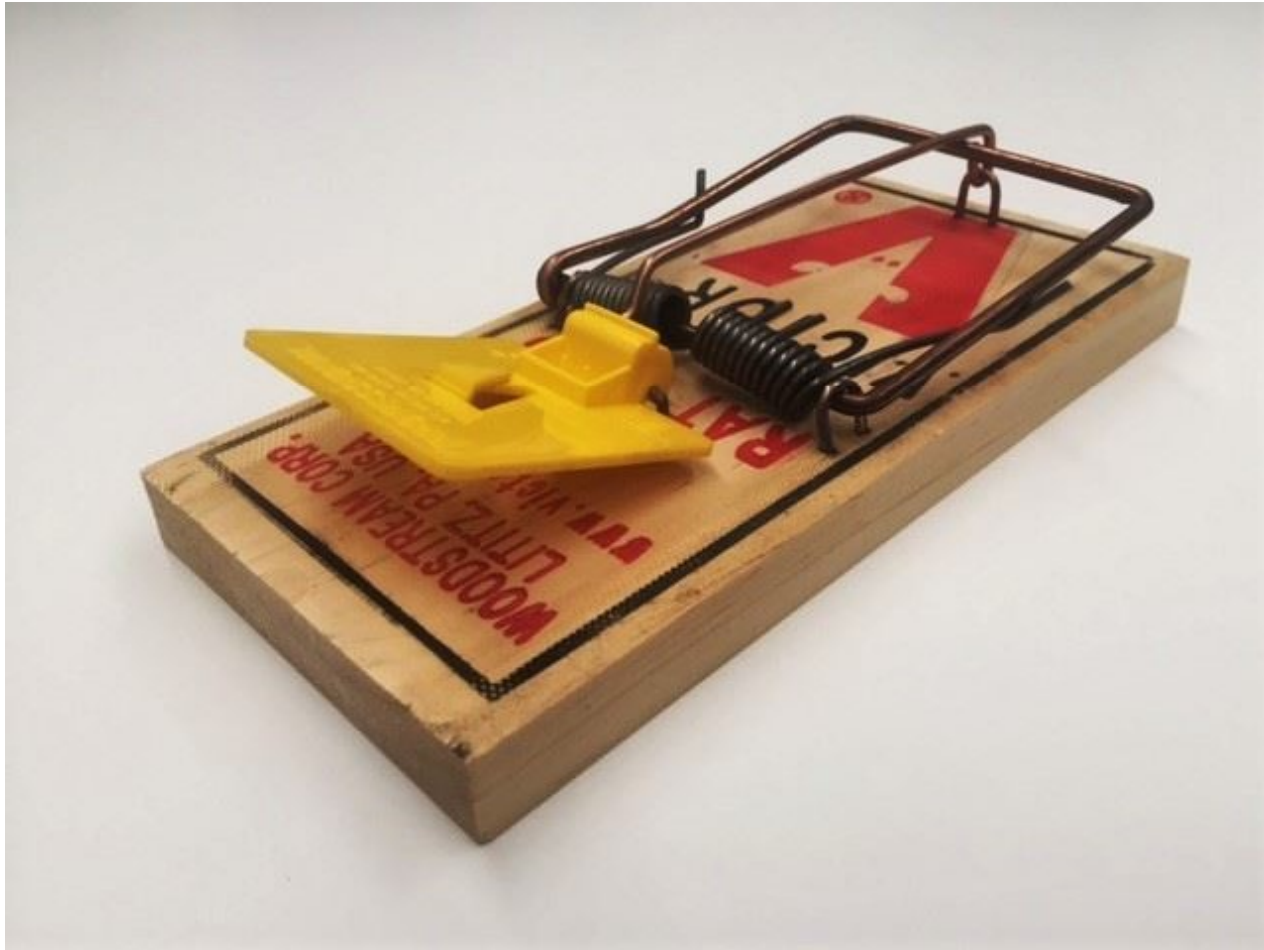
- Breaks often
- Changes often
- Is critical to the business



Code coverage is a trap



How much code coverage is enough?



Code coverage everywhere

- It is easy to understand
- It is easy to measure
- There are many tools for measuring it
- Also familiar to other project stakeholders
- Beloved by QA departments and managers

I will tell you a secret



I will tell you a secret

A project can be full of bugs and
still have 100% code coverage

I will tell you a secret

Do not try to achieve a specific
number (such as 100%)

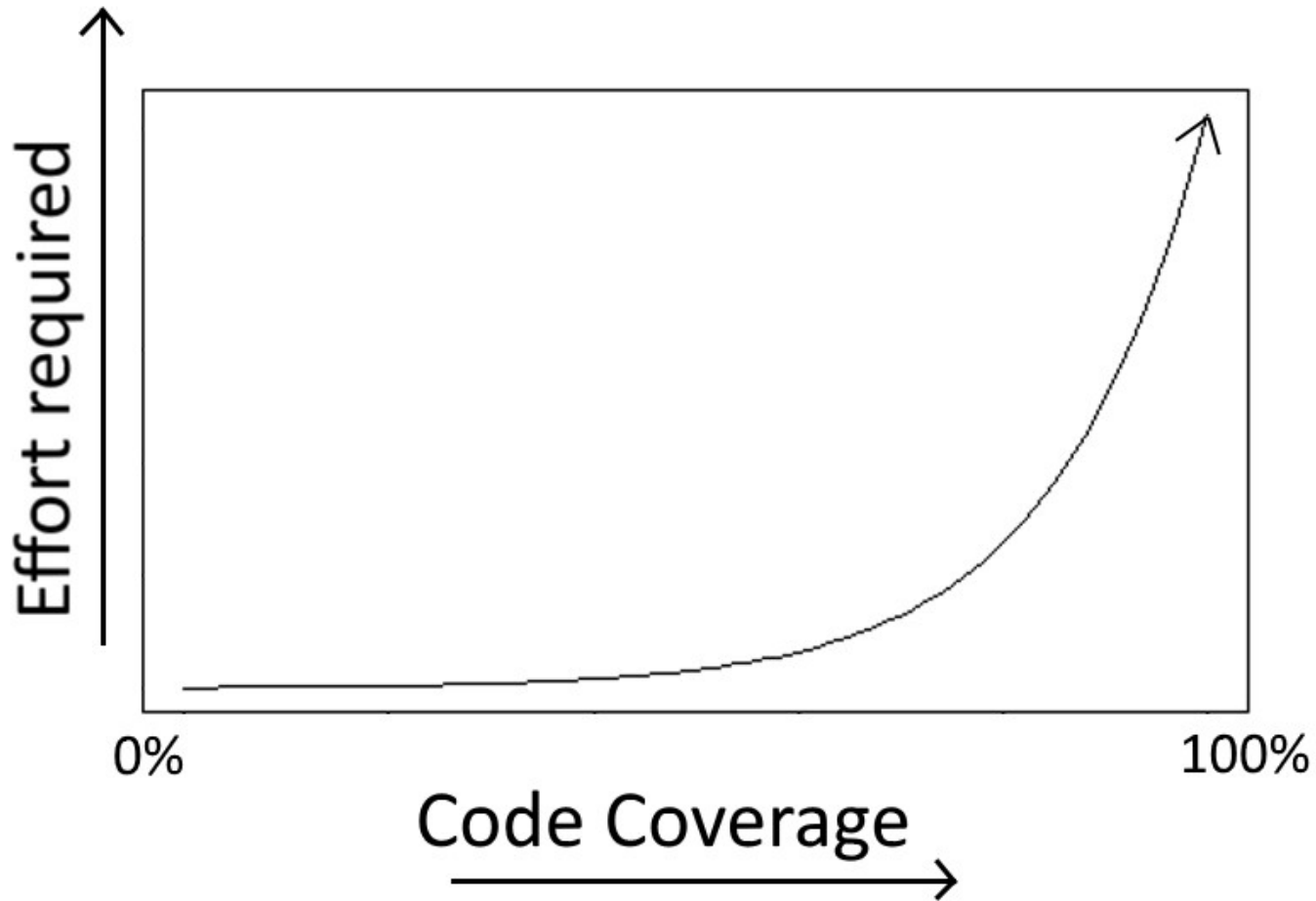
I will tell you a secret

Bigger numbers require more
effort (logarithmic?)

I will tell you a secret

Getting from 80% to 100% is
much more difficult than 0% to
20%

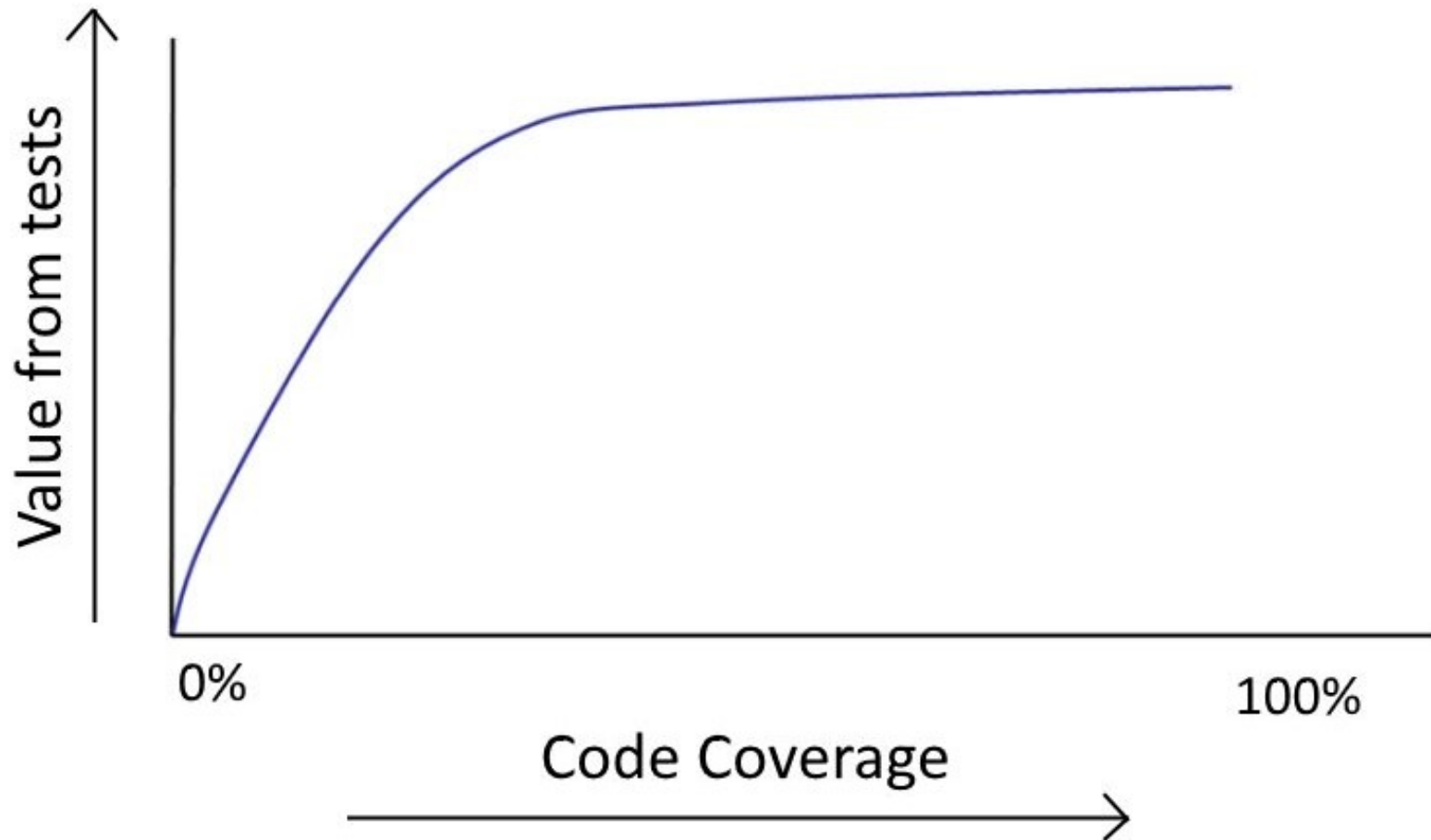
I will tell you a secret



I will tell you a secret

Increasing code coverage has
diminishing returns

I will tell you a secret



I will tell you a secret

High code coverage !=
high code quality

Software developers goals?

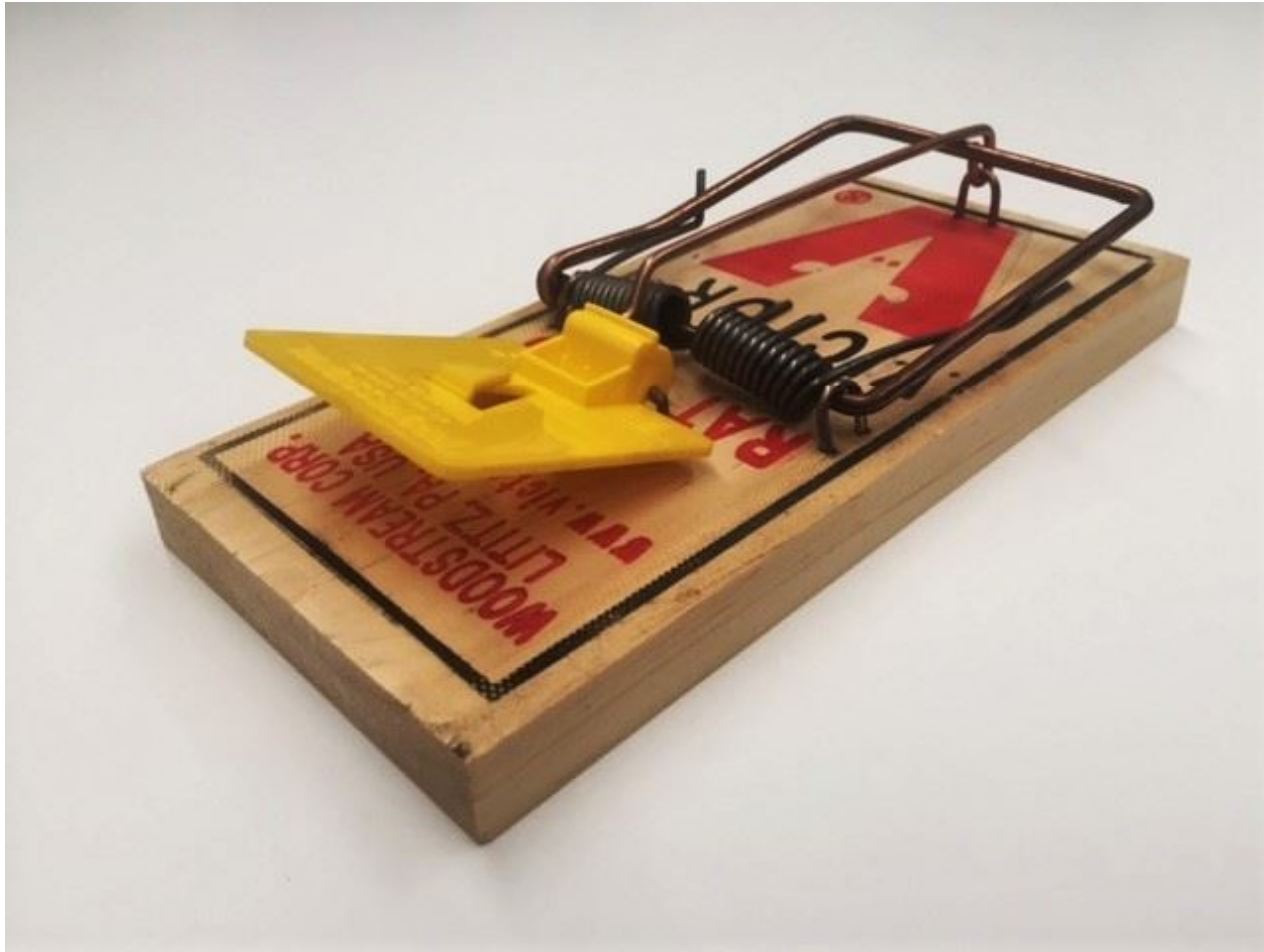
- Write code
- Write unit tests
- Write documentation
- Prepare architecture documents
- Resolve bugs
- Install software packages
- Take part in meetings
- Keep up with latest frameworks

The real goals

- Solve problems
- Offer value to customers



Give me a number!



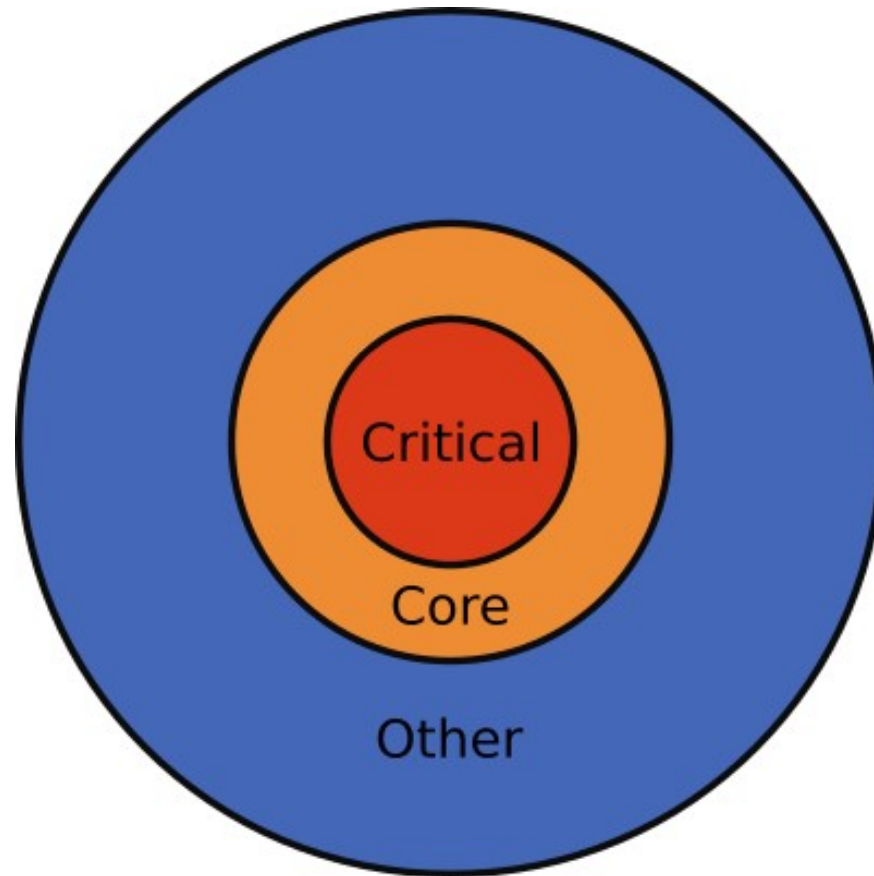
Best code coverage

20% is the magic
number

Pareto principle

20% of your code is
responsible for 80% of your
bugs

Pareto principle



Pareto principle

Try to achieve 100% coverage of your CRITICAL code, (which itself is probably 20% of total code)

Part 5 – Greenfield projects

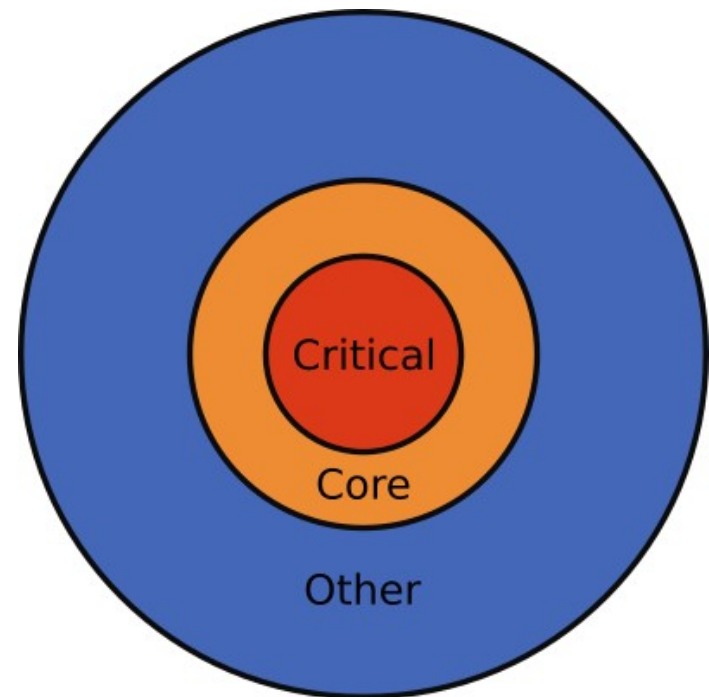


Quiz:

You start working on an unknown project with zero tests. Where do you start testing?

Write test for code that

- Breaks often
- Changes often
- Is critical to the business



How do you find critical code

See what bugs appear in
production

How do you find critical code

...and write unit/integration/ui
tests for them

Production bugs

- Have passed all QA gates (since they appeared in production already)
- Are great for regression testing

Production bugs

Should only happen once!

New project – zero tests

- Do NOT start testing code you understand
- Do NOT start testing code that requires easy tests
- Do NOT start testing the first folder in your file system
- Do NOT start testing what a colleague suggested

New project – zero tests

First test suite should be
production bugs

Part 6 – How to be a pro



A professional is..

...somebody who knows the
tools of the trade



Read test documentation

- Do not re-invent the wheel
- Do not write new test utilities
- Do not create “smart” test solutions
- Do not copy paste test code
- Do not write “helper” test methods
- Do not ignore off-the-self test libraries

Show respect

Treat test code like your production code



Research and learn

Your test framework and its capabilities



Learn about

- Parameterized tests
- Mocks and stubs (and spies)
- Test setup and tear down
- Test categorization
- Conditional running for tests
- Assertion grouping



Learn about

- Test data creators
- Http client libraries
- HTTP mock libraries
- Mutation/fuzzy testing
- Db cleanup/rollback
- Load testing
- Environment launching



Conclusion

- You need multiple test suites (pyramid)
- Features get promoted between phases
- Write tests that add value (critical bugs)
- Run tests automatically in a pipeline
- Create preview environments
- Learn your testing tools well
- Treat test code as production code