Better Software Development Through Automated Tooling

Summer Bursary 2017

Vaughan Kitchen

April 28, 2017

Structure

- Background
- Survey of Tools
- Live Demonstration
- Project Examples
- Related Topics

Background

What

Plug as many automated (free) tools as possible into the development of JASSv2 (A search enging being developed by Andrew Trotman)

When

A summer studentship at the start of 2017

How

The approach was to first figure out what tools we might want. Then what tools were available to us. Which of the available ones were worth integrating. And then get them integrated

Why

A story

You're a developer on a small team for a new project. You don't have the capital to hire quality assurance team so all testing is done internally. Up until now you've build the project and tested it at the end, but you've noticed it's becoming more and more difficult to track down bugs. You wonder if there's a way to catch them earlier in the life cycle, which would also allow you to not develop around them. So you begin writing unit tests. Some of the other developers think this is a great idea so begin writing tests as well. But they're not always the best at running them every time they commit. It's crunch time and you have to get the product out to your customers. One of the other developers writes a last minute fix and you ship. Oh no, she reintroduced a bug that was fixed weeks ago and has a test for. But because the tests weren't run it managed to make its way back into the code. You wonder if there's a better way

Why

A worse story

You don't unit test

A common story

You want to test. The rest of your team doesn't. How do you go about slowly introducing it and convincing your team of the benefit

An Overview of Tools

- Unit Tests
- Integration Tests
- System Tests
- Regression Tests
- Code Coverage
- Static Analysis
- Memory & Leak Checks
- Performance Profiling
- Code Hygiene
- Code Documentation

Continuous Integration

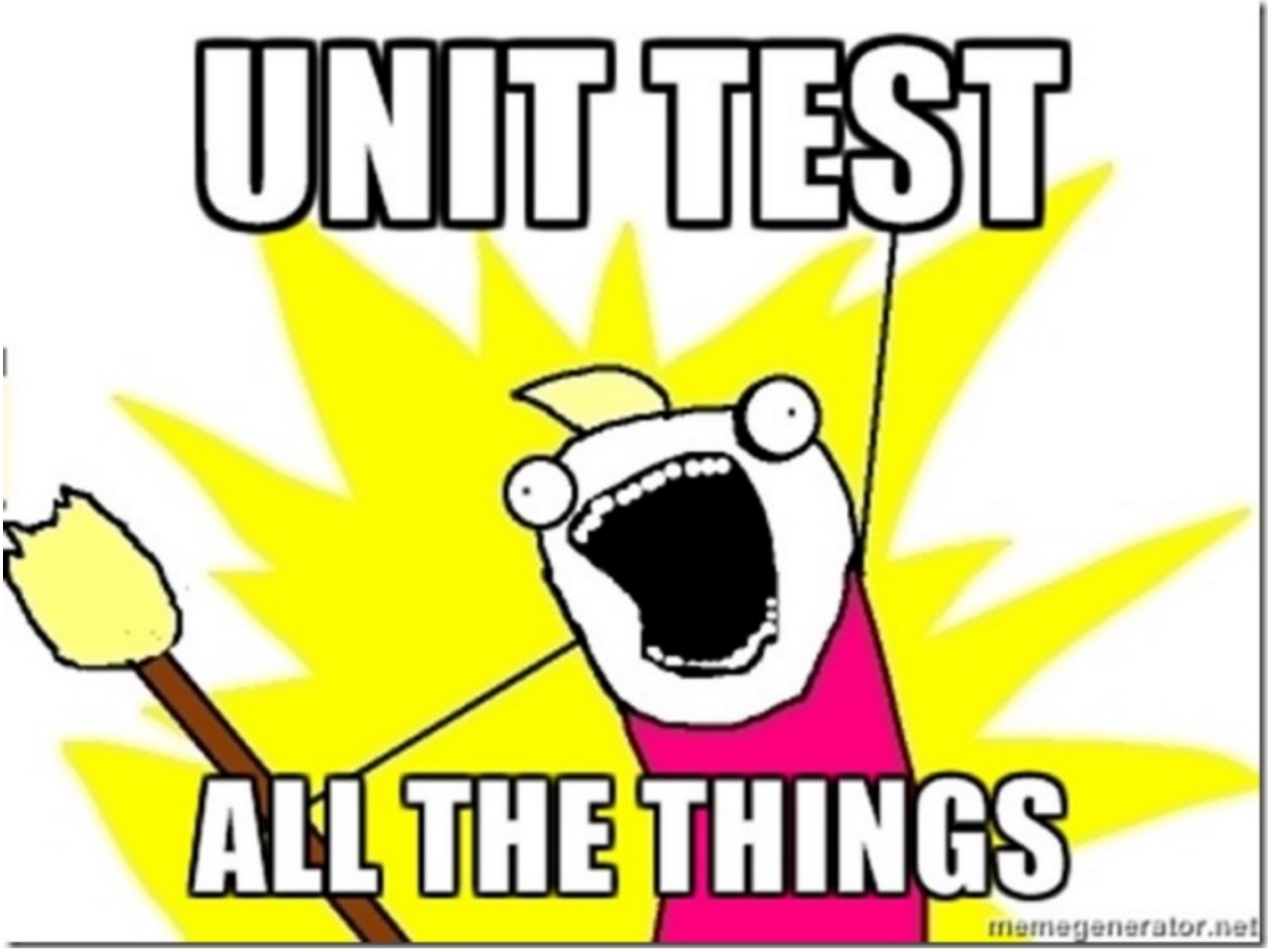
Travis CI

- Automates Build
- Integrates with Github (or BitBucket for some CI systems)
- Runs on every push
- An hour of (free) compute time per VM instance
- Do you develop on the same environment as your deploy?
- What if you deploy to multiple environments?
- What else can it do besides build?

Continuous Integration

- Merge all working copies to mainline several times a day
- Prevents "integration hell"
- Feature toggles for partially complete code
- Continuous Delivery (mainline is always in a deployable state)
- Continuous Deployment (automatic deployment to production)
- Notify the developer that broke the code (default) or specified manager/team

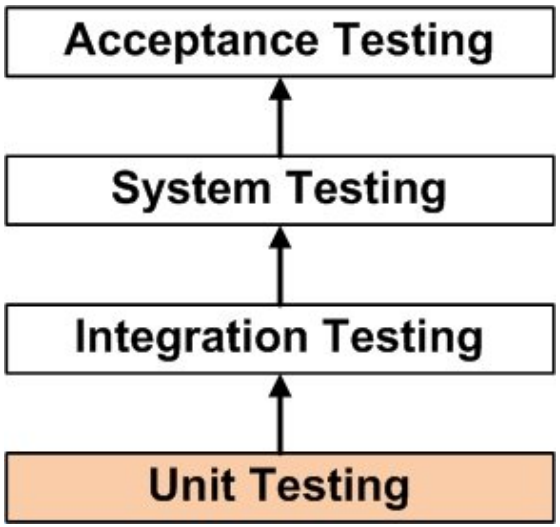
Unit Testing



Unit Testing

```
1  #include "unity.h"
2  #include "string2.h"
3
4  void  test_string_append (void)
5        {
6        /* test the normal use case */
7        struct  string  *s = string_new_c("cat");
8        string_append_c(s, "dog");
9        TEST_ASSERT_EQUAL_STRING(s->str, "catdog");
10       TEST_ASSERT_EQUAL_UINT(s->bytes, 6);
11
12        string_free (s);
13       }
```

Unit Testing

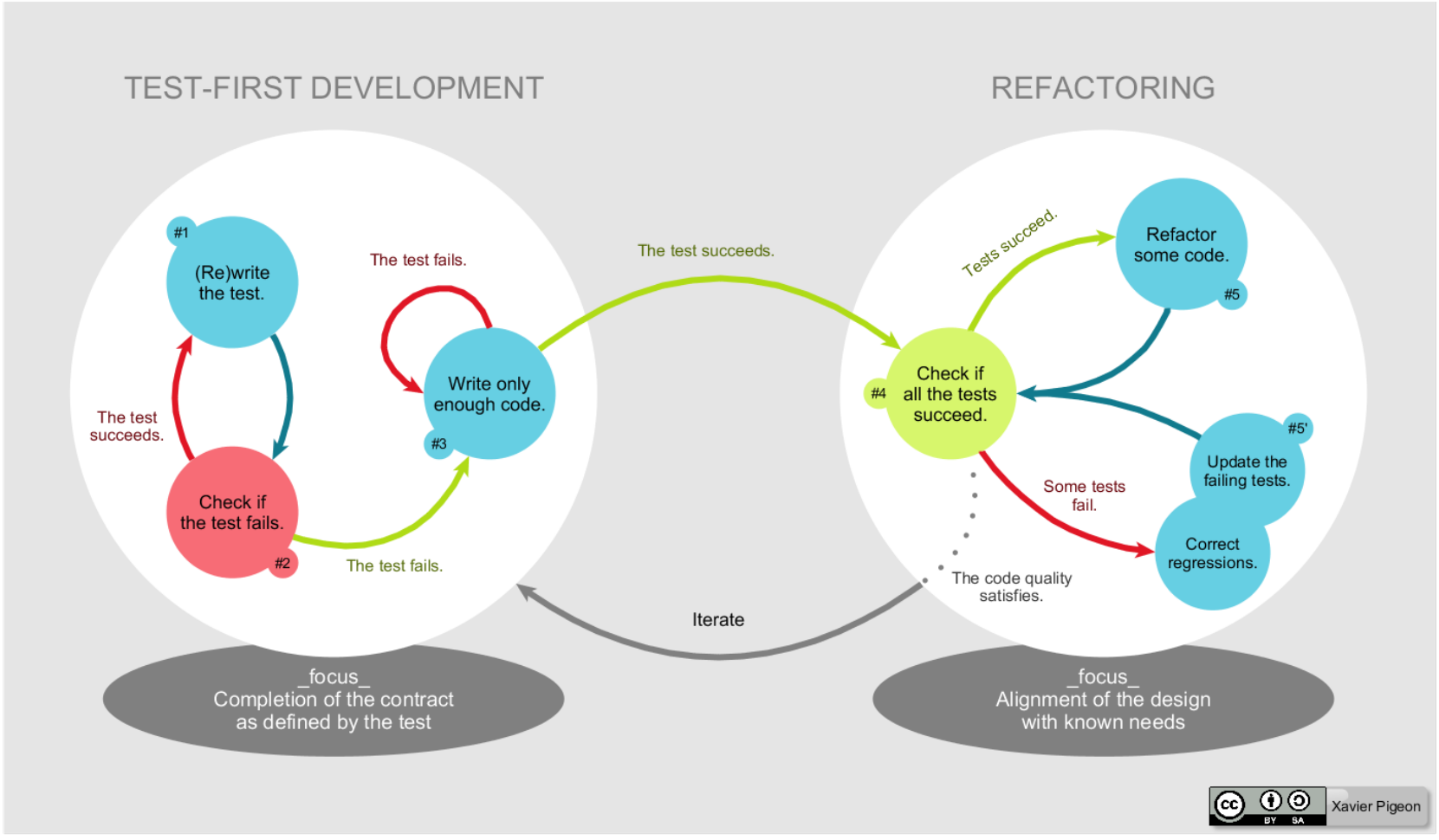


- If it isn't tested, it doesn't work
- Part of your test suite
- Runs from your Continuous Integration system
- Assures the quality of individual units
- Smallest testable unit (Function, or Class)
- Simplifies integration
- Confidence in refactoring
- Provides documentation
- JUnit, CUnit, Unity, tinytest, Jasmine...

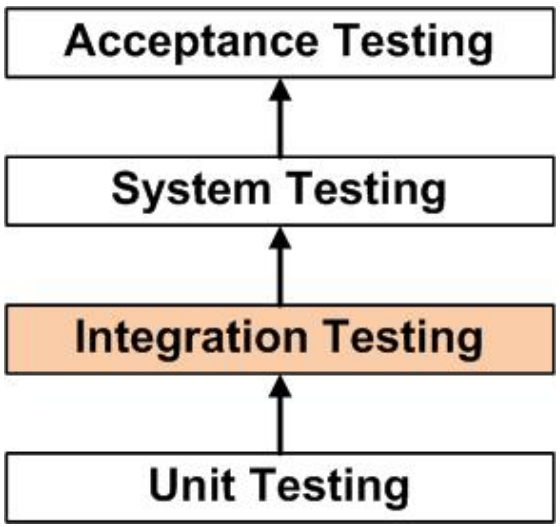
Test Driven Development

- Performed after the design is finished
- Used to build your units
- How do you know when you're finished?
- Do you know what you're building?
- How far through building it are you?
- Refactor mercilessly
- Type systems can't always save you
- Red Green Refactor

Test Driven Development

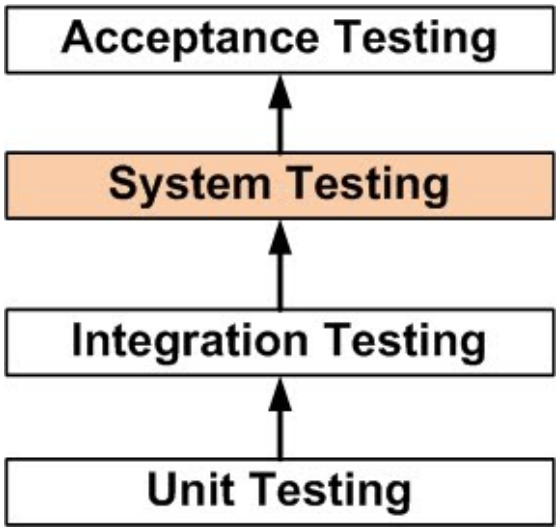


Integration Testing



- Part of your test suite
- Fine line between unit tests and integrations tests
- Runs from your Continous Integration system
- Next step up from unit tests
- Tests whether groups of modules work together correctly
- Doesn't test the correctness of the system as a whole

System Testing



- Does the application do what it should?
- Hard to automate entirely
- Can automate some code paths (CI has full power of Linux)
- bats (Bash Automated Testing System)

Regression Testing

- Does my program still do the same?
- Computing power is now cheap (free)
- Run the entire test suite for each push
- Continuous Integration system will notify on breakage
- Regression tests are now free

Regression Testing

Commits on Apr 22, 2017


fix resizing of deeply nested frames
vkitchen committed 6 days ago ✓ `bd41e31`


fix split and rearrange from conflicting against each other
vkitchen committed 6 days ago ✓ `b37b1c3`

Commits on Apr 21, 2017


split frame on drag
vkitchen committed 7 days ago ✗ `154189f`


shadow frame from hovering
vkitchen committed 7 days ago ✓ `dd3ce44`


pass test rearranging tabs with dead frames
vkitchen committed 7 days ago ✓ `8601ec3`

Commits on Apr 20, 2017


partial implementation of rearrange
vkitchen committed 8 days ago ✗ `4dde6f3`

Code Coverage

- How much of the source code does the test suite execute?
- Do tests cover succeeding and failing paths?
- Function coverage
- Statement coverage
- Branch coverage
- Condition coverage
- Codecov, Coveralls

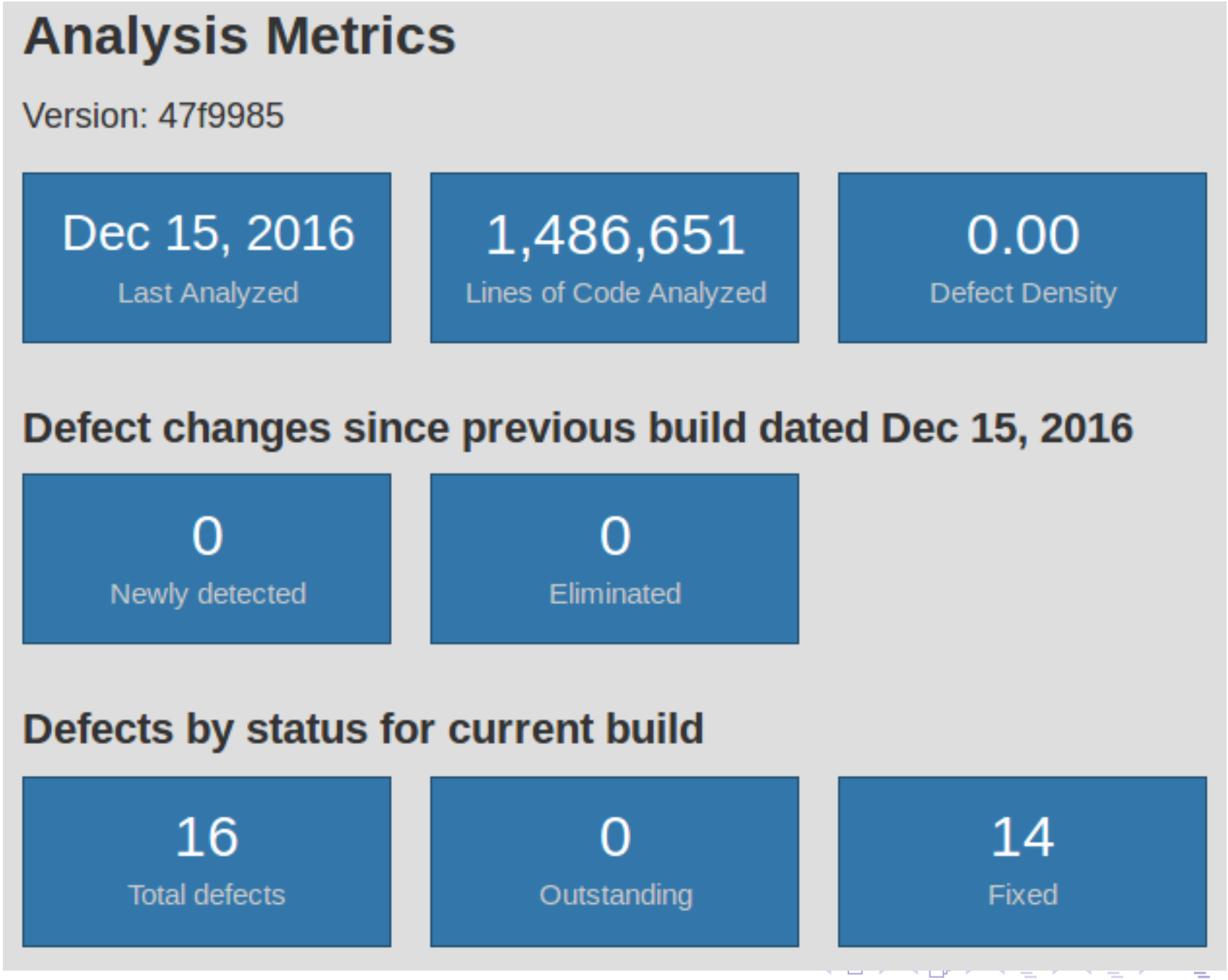
Code Coverage

```
size_t file::read_entire_file(const std::string &filename, std::string &into)
{
FILE *fp;                                                    // "C" pointer
struct stat details;                                  // file system's details of th
size_t file_length = 0;                       // length of the file in bytes

/*
        Fopen() the file then fstat() it.  The alternative is to stat() then f
*/
if ((fp = fopen(filename.c_str(), "rb")) != nullptr)
        {
        if (fstat(fileno(fp), &details) == 0)
                if ((file_length = details.st_size) != 0)
                        {
                        into.resize(file_length);
                        if (fread(&into[0], details.st_size, 1, fp) != 1)
                                into.resize(0);
                        }
        fclose(fp);
        }

return file_length;
}
```

Static Analysis

- Verify behaviour
- Lint for best practices
- Prove correctness
- Sometimes freely available to Open Source projects
- Can be integrated into CI

Static Analysis



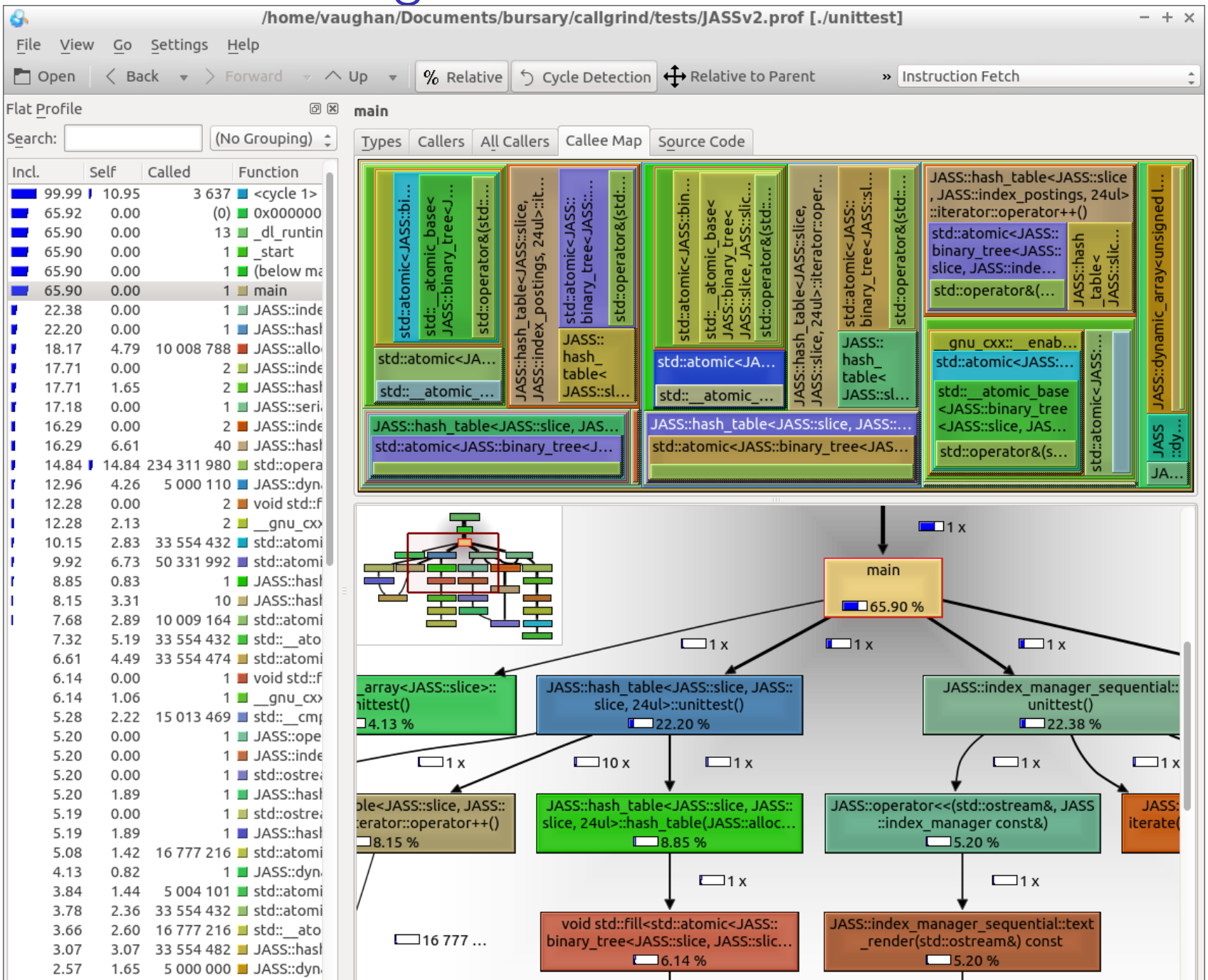
memcheck

- Dynamic analysis
- Memory leaks
- Corrupted memory
- Valgrind

Performance Profiling

- Algorithmic performance
- Web performance
- Latency vs Throughput
- Part of regression testing
- Instrumentation, Sampling
- Gprof, Callgrind

Performance Profiling



Code Hygiene

- Code smell
- Linting
- Best Practices
- Anti-patterns
- CodeClimate, CodeLingo

"lint was the name originally given to a particular program that flagged some suspicious and non-portable constructs (likely to be bugs) in C language source code. The term is now applied generically to tools that flag suspicious usage in software written in any computer language."

— wikipedia

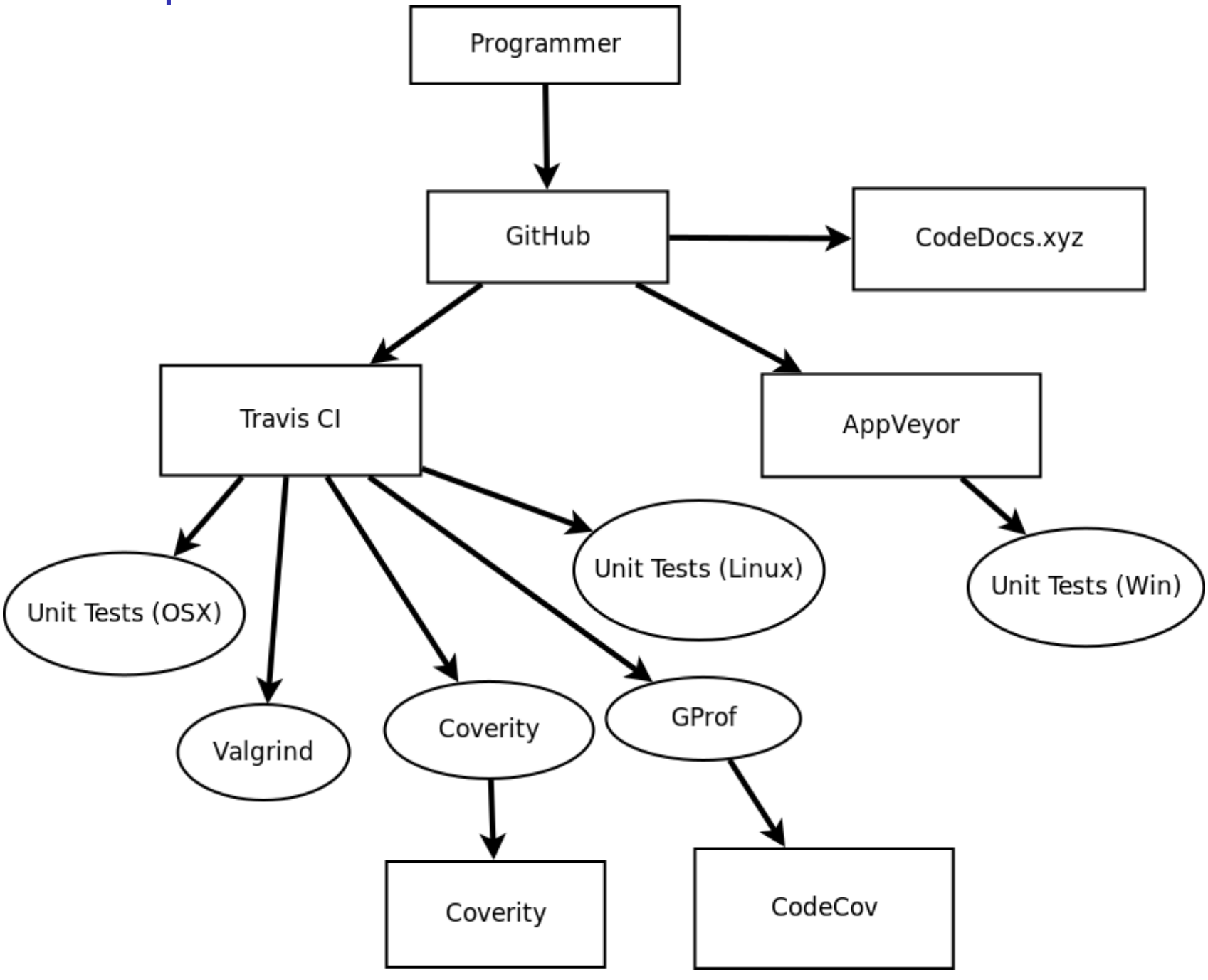
Code Documentation

- Doxygen, Javadoc
- CodeDocs.xyz
- GitHub Pages

Our Set Up

- Travis CI (OSX, and Linux)
- AppVeyor (Windows)
- Unit Testing in a custom framework
- Coverity (coverity_scan branch)
- Valgrind
- CodeCov (Gcov)
- CodeDocs.xyz (integrates directly with GitHub)

Our Set Up


Programmer
GitHub
CodeDocs.xyz
Travis CI
AppVeyor
Unit Tests (OSX)
Unit Tests (Linux)
Unit Tests (Win)
Valgrind
Coverity
GProf
Coverity
CodeCov

YAML 1

- Superset of JSON
- Supports comments
- Often used as a configuration format

YAML 2

```
1  "object": {
2      "key": "value",
3      "array": [
4          { "null_value": null },
5          { "boolean": true },
6          { "numeric": 1 },
7          { "string": "unquoted" },
8          { "quoted": "123" }
9      ]
10 }
```

YAML 3

```
1  object :
2        key :  value
3        array :
4              - null_value :
5              - boolean: true
6              - numeric: 1
7              - string :  unquoted
8              - quoted: "123"
```

Live Demonstration

Other Set Ups

- https://github.com/DandyHQ/mace-prototype
- https://github.com/rubinius/rubinius
- https://github.com/IronLanguages/main

Continuous Deployment

- Next step from Continuous Delivery
- Code automatically deployed to production
- Seen in SAAS or Web Applications
- Application monitoring and Dashboards
- Feature Toggles
- Graphs and anomoly detection

Property Based Testing

Fuzzing

- Runs the program against random input
- Used to check for security vulnerabilities
- American Fuzzy Lop (AFL) uses genetic algorithms and instrumentation to try and reach all code paths

Property Based Testing

- Unit Testing on roids
- Do properties of the output still hold when the program is run with random input
- Forces the consideration of edge cases
- Reduces failures to minimal counter examples
- QuickCheck, ScalaCheck, ClojureCheck, JavaQuickCheck, RapidCheck (C++)

Roll Your Own

Webhooks

- Webhooks form the foundation of integrations
- Subscribe to specific events
- push, fork, issues, release, watch...
- HTTP POST payload to specified URL
- Secure your webhook with a secret token, payloads will be signed

Other APIs

- There is also an integration API to give applications access to private repositories etc.
- OAuth application is authenticated as if it's the user

Attributions

- `http://softwaretestingfundamentals.com` Testing heirarchy images, used under CC Attribution-ShareAlike
- `https://upload.wikimedia.org/wikipedia/commons/0/0b/TDD_Global_Lifecycle.png` TDD, CC Attribution-SharAlike
- The various screenshots are of JASSv2 which is licensed under BSD

Questions?