Better Software Development Through Automated Tooling

Vaughan Kitchen

April 28, 2017

Structure

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

Background

What

Plug as many automatic 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

An Overview of Tools

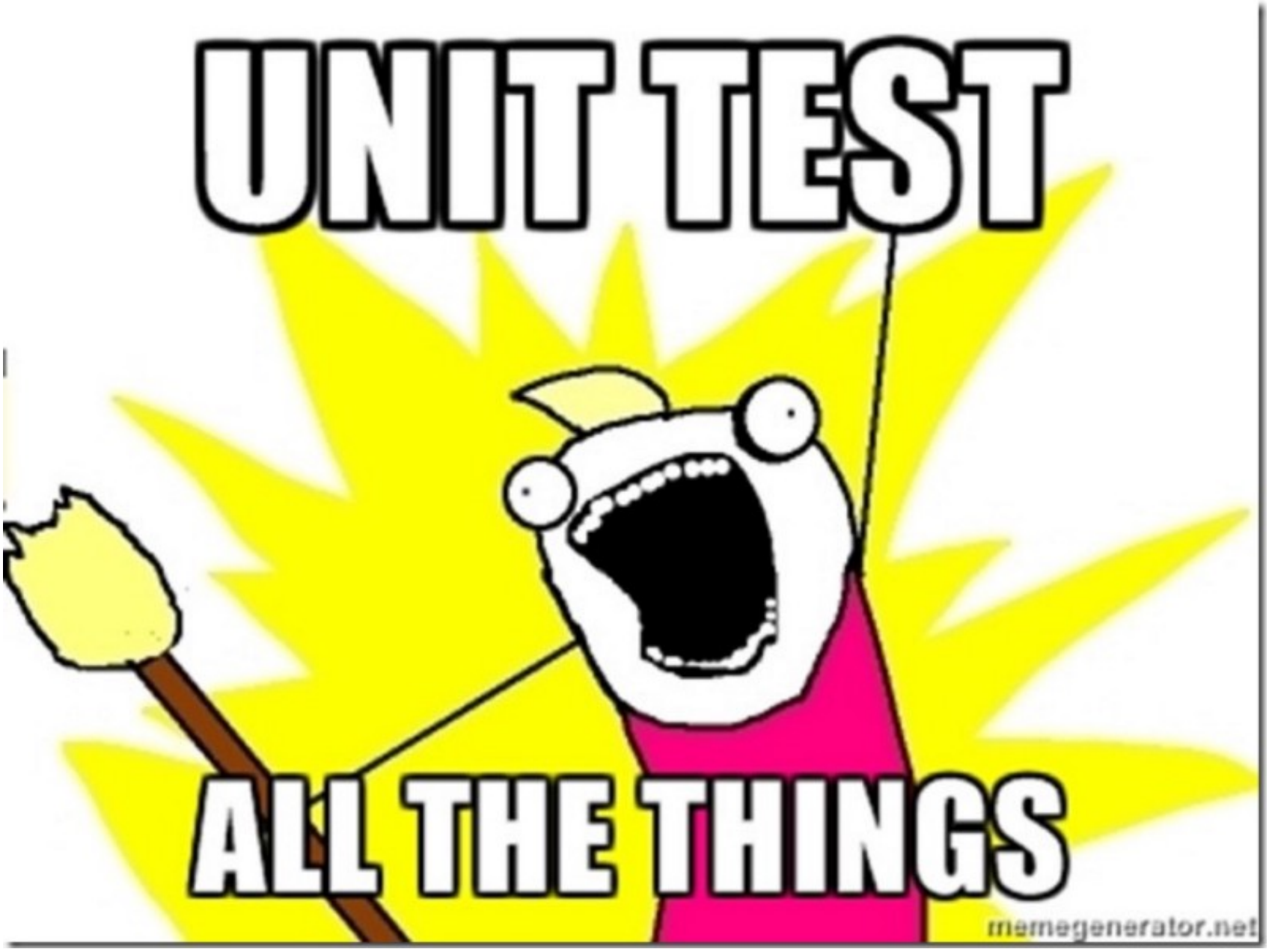
- Unit Tests
- Integration Tests
- System Tests
- Regression Tests
- Code Coverage
- Static Analysis
- Memory & Leak Checks
- Performance Profiling
- Code Hygiene
- Code Documentation

Continuous Integration pt.1

Travis CI

- Automates Build
- Integrates with Github (or BitBucket for some CI systems)
- Runs on every push
- An hour of compute time per VM instance
- What else can it do besides build?

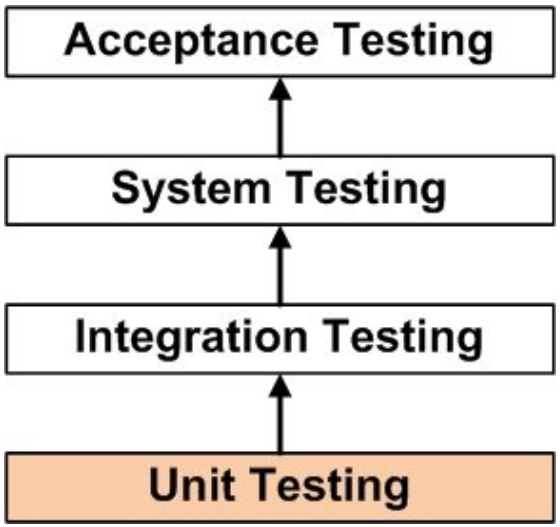
Unit Testing 1



Unit Testing 2

```
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 3

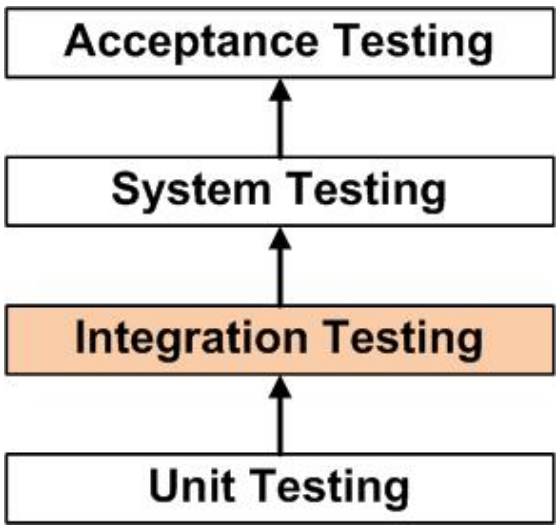


- If it isn't tested, it doesn't work
- Part of your test suite
- Runs from your Continuous Integration system
- Assures the quality of individual units
- Smallest testable unit (Function, or Class)
- Simplifies integration
- Confidence in refactoring
- Provides documentation
- JUnit, CUnit, Unity, tinytest,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

Test Driven Development

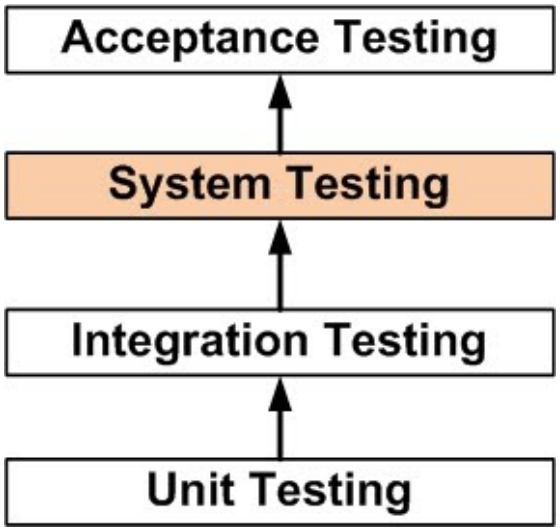
- How do you know when you're finished?
- Do you know what you're building?
- How far through building it are you?
- Refactor mercilessly
- Type systems can't always save you

Integration Testing



- Part of your test suite
- 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 the previously functional software still function correctly?
- Computing power is now cheap
- Run the entire test suite for each push
- Continuous Integration system will notify on breakage
- Regression tests are now free

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

Static Analysis

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

memcheck

- Dynamic analysis
- Memory leaks
- Corrupted memory
- Valgrind

Performance Profiling

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

Code Hygiene

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

Code Documentation

- Doxygen, Javadoc
- CodeDocs.xyz
- GitHub Pages

Continuous Integration pt.2

- 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

YAML 1

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

YAML 2

```
"object": {
    "key": "value",
    "array": [
        {
            "null_value": null
        },
        {
            "boolean": true
        },
        {
            "integer": 1
        },
        {
            "string": "rope"
        }
    ]
}
```

YAML 3

```yaml
object:
    key: value
  array:
    - null_value:
    - boolean: true
    - integer: 1
        - string: rope
```

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)

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 images, used under CC Attribution-ShareAlike