



Testing, Code Coverage, and Continuous Integration

ATPESC 2018

Jared O'Neal
Mathematics and Computer Science Division
Argonne National Laboratory

Q Center, St. Charles, IL (USA)
July 29 – August 10, 2018

License, citation, and acknowledgments



License and Citation

- This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/) (CC BY 4.0).
- Requested citation: Alicia Klinvex and Jared O'Neal, Testing, Code Coverage, and Continuous Integration, tutorial, in Argonne Training Program on Extreme-Scale Computing (ATPESC) 2018. DOI: 10.6084/m9.figshare.6972749.

Acknowledgements

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
- This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357
- Alicia Klinvex

Why is testing important?

Therac-25 computer-controlled radiation therapy machine

- Minimal software testing
- System administers radiation in improper setup
- Unlucky patients were struck with approximately 100 times the intended dose
- Recalled after six accidents resulting in serious injury
- Race condition in the code went undetected

Leveson & Turner, “An Investigation of the Therac-25 Accidents”. IEEE (1993).

Why is testing important?

Ariane 5 Launch Vehicle

- Launched to lift heavy payloads into low Earth orbit
- Initial rocket driven off course, started to disintegrate, and destroyed with 40 seconds of launch
- SW shutdown when 64bit floating-point to 16bit int overflowed
- Used directly SW from Ariane 4 - overflow physically impossible
- Conversion never tested

Bashar Nuseibah, “Ariane 5: who dunnit?”. IEEE Software (1997).

<http://iansommerville.com/software-engineering-book/case-studies/ariane5/>

Why is testing important?

Protein structures in scientific software

- Inherited data analysis code interchanged two data columns
 - inverted electron-density map
 - incorrect protein structure
- Retracted 5 publications
 - One was cited 364 times
- Many papers and grant applications conflicting with flawed results were rejected
- Others based their research on incorrect results

Greg Miller, “A Scientist’s Nightmare: Software Problem Leads to Five Retractions”. Science (2006).

How common are bugs?

Programs do not acquire bugs as people acquire germs,
by hanging around other buggy programs.
Programmers must insert them.
- Harlan Mills

Industry average for delivered software

- 1-25 errors per 1000 lines of code

Microsoft Applications Division

- 10-20 defects per 1000 lines of code during in-house testing
- 1 defect per 2000 lines of code in released product

Steven McConnell, "Code Complete". Second Edition (2004).

Avoid debugging

Debugging is twice as hard as writing the code in the first place.
Therefore, if you write the code as cleverly as possible, you are,
by definition, not smart enough to debug it.

- Brian Kernighan

- Upfront effort can lead to clean code
- Integrated tests encode result of time and effort
- Integrated tests can ease debugging

Benefits of testing

- Promotes high-quality software that delivers correct results and improves confidence
- Increases quality and speed of development, reducing development and maintenance costs
- Maintains portability to a variety of systems and compilers
- Automated testing helps create code that isn't brittle

Definitions

Classes of Tests by Granularity / Hierarchy

- Unit tests
 - Test individual functions or classes
- Component or Integration tests
 - Test linkage of functions and classes into higher-functioning unit
- System-level tests
 - At the user interaction level

Definitions

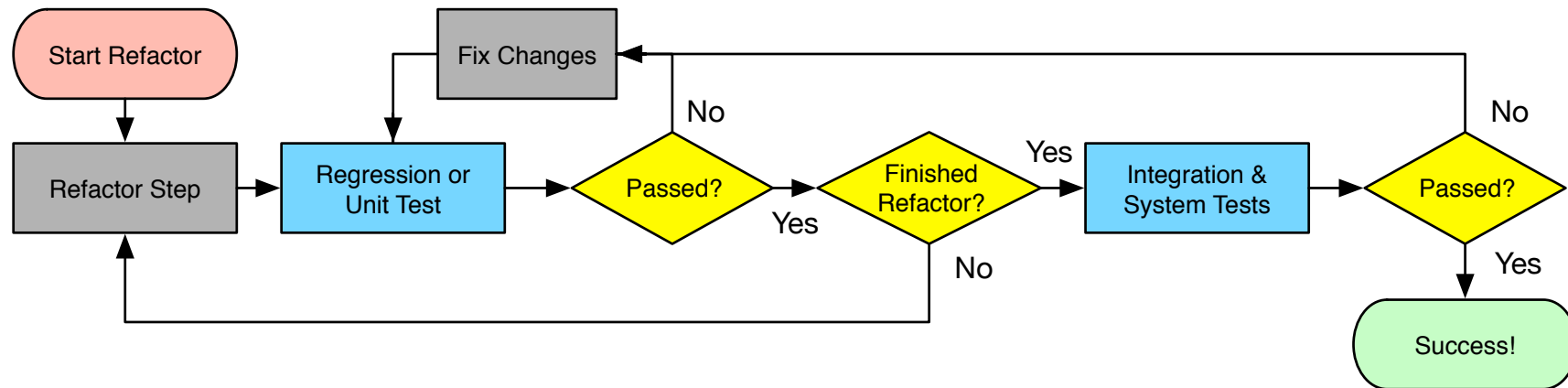
Classes of Tests by Intent

- General verification
 - Compare against analytic results, independently-derived result, *etc.*
- Regression tests
 - Compare current observable output to a gold standard
- Performance tests
 - Focus on the runtime and resource utilization
- Restart tests
 - Code starts transparently from a checkpoint

Refactoring

A technique for improving the design or implementation of existing code without changing the behavior of the code.

Toy workflow with testing



Bonus Questions: How does this fit in with DVCS workflow?
When do you commit and merge?

Automated Test Frameworks

Software packages that

- provide routines that reduce tedium of writing test conditions,
- Simplify maintaining and building test suite,
- automates execution of a collection of client test routines, and
- report test results.

Examples:

- C++
 - Boost.Test, Catch, GoogleTest
- Fortran
 - Fruit, PUnit
- python - nose, pytest, unittest
- Java - JUnit
- MATLAB - built in
- Custom - FlashTest

Test Servers

Servers that

- automate the execution of a test suite or a subset of a test suite,
- allow for running tests on different environments,
- host an interface for viewing results, and
- allows for configuring when the tests are run.

Examples

- CTest/CDash
- Jenkins
- Travis CI and GitLab CI

Testing Policies

- Testing regime is only useful if it is maintained and monitored
 - Coevolve code and tests
- When to test and what to test? How to combine with DVCS workflow?
- When and how to update baseline data and tolerances?
- How to proceed when a bug is detected?

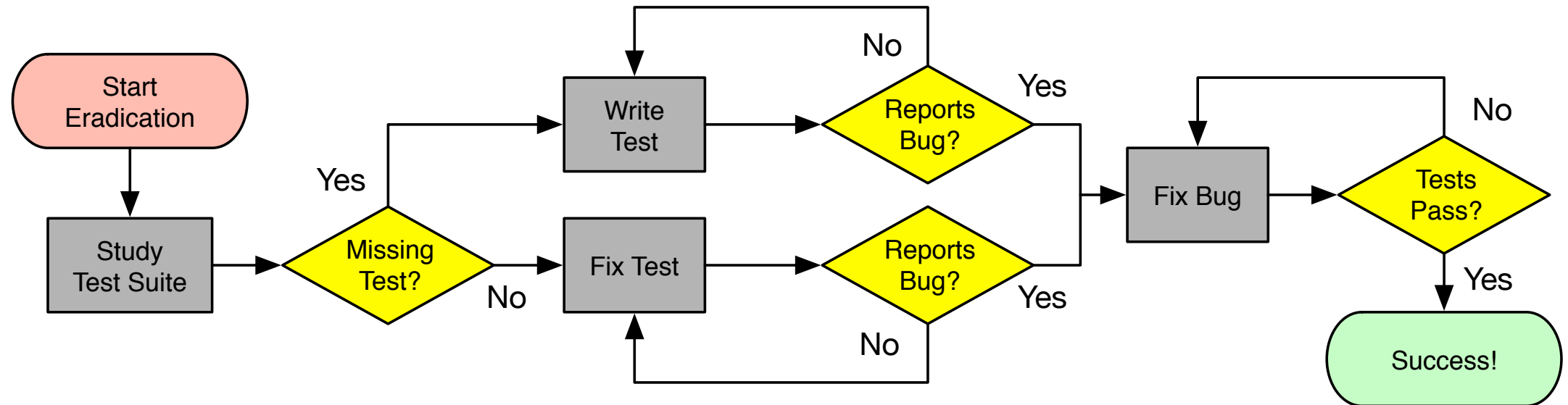
Baseline Policies

When and how to update baseline data and tolerances

- Determining tolerances can be hard
 - Too small triggers study of “failures”
 - Too large can lead to undetected failures
- Bit-by-bit matching
 - Not always possible with parallel programs
 - Requires study of deviations and maintaining baseline history

Bug Eradication Policies

How to proceed when a bug is detected?



How do we determine what other tests are needed?

Code coverage tools

- Expose parts of the code that aren't being tested
- gcov
 - standard utility with the GNU compiler collection suite
 - Compile/link with `-coverage` & turn off optimization
 - counts the number of times each statement is executed
- lcov
 - a graphical front-end for gcov
 - available at <http://ltp.sourceforge.net/coverage/lcov.php>
- Hosted servers (e.g. coveralls, codecov)
 - graphical visualization of results
 - push results to server through continuous integration server

Code coverage output

Overall Analysis

SOURCE FILES ON BUILD 45					
LIST 2	CHANGED 0	SOURCE CHANGED 0	COVERAGE CHANGED 0		
▲ COVERAGE	Δ	FILE	LINES	RELEVANT	COVERED
— 74.39		src/functions/linear_fcn_class.f90	301	82	61
— 100.0		src/general/modulo_mod.f90	52	3	3

Detailed Analysis

```
265      ! Error distribution same for all x values
266      delta = S*Sxx - Sx*Sx
267      if (delta == 0.0_wp) then
268          ERRORMSG("Cannot do linear least-sqrs. Divide by zero.")
269          stop
270      end if
271      delta_inv = 1.0_wp / delta
```

Online tutorial - <https://github.com/amklinv/morpheus>

Other example - <https://github.com/jrdoneal/infrastructure>

Code coverage is popular

- gcov also works for C and Fortran
- Other tools exist for other languages
 - Jcov for Java
 - Coverage.py for python
 - Devel::Cover for perl
 - profile for MATLAB
 - *etc.*

The Short & Sweet of Continuous Integration

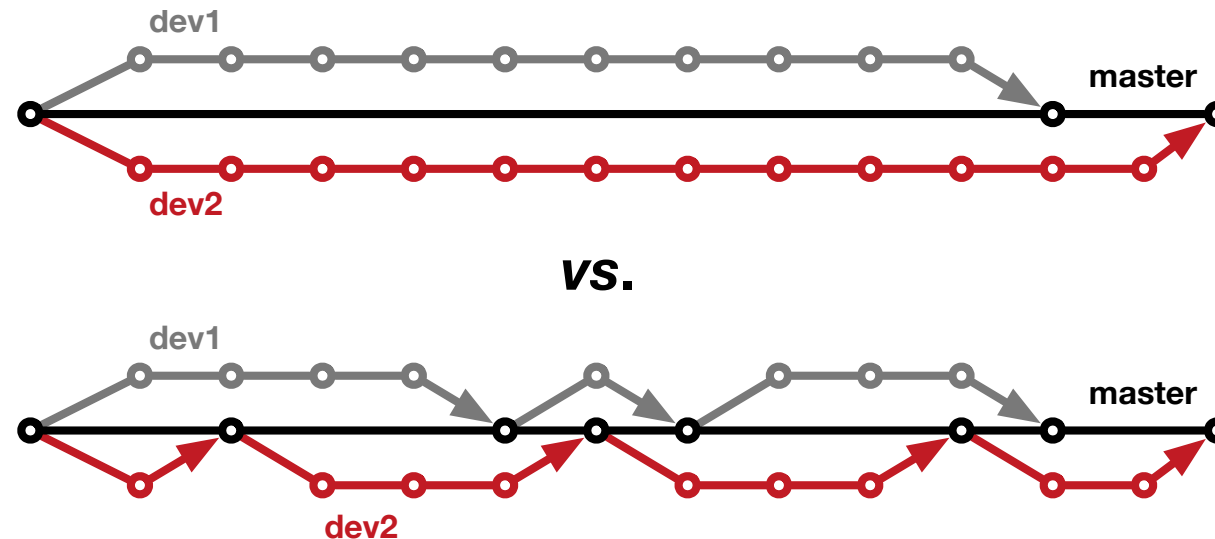
A master branch that always works

- DVCS workflow isolate master from integration environment
- Extend workflow to address difficulties of integrating
 - Minimize likelihood of merge conflict
 - Detect bugs immediately
 - Make debugging process quick and easy

Work Decomposition

Commit and integrate often

- Limit divergence between feature and master branches
- Decreased probability of conflict
- Conflict resolution is simpler and less risky



Error detection

Test at integration to identify failures immediately

- Control quality of code
- Isolate failure to few commits
- No context switching for programmer

We want a system that

- triggers automated builds/tests on target environments when code changes and
- ideally tests on proposed merge product without finalizing merge.

Continuous integration (CI)

- Has existed for some time and interest is growing
- ECP working to adapt CI for HPC machines
- Setup, maintenance, and monitoring required
- Prerequisites
 - A reasonably automated build system
 - An automated test system with significant test coverage & useful feedback
 - Builds/tests must finish in reasonable amount of time
 - Ability to bundle subset of tests

Cloud-based CI

- Linked to VCS hosts
 - GitHub & Travis CI
 - GitLab CI
 - BitBucket Pipelines
- Automated builds/tests triggered *via* pushes and pull requests
- Builds/tests can be run on cloud systems
- Test results are reported in repository's web interface
- Can trigger code coverage analysis & documentation build
- Run tests on different environments

View of toy repository

<https://github.com/jrdoneal/infrastructure>

Repository Root

documentation	Included proper location of folders
src	Convert line_t into a class and impr
.gitignore	Travis CI should now run code cover
.travis.yml	Convert line_t into a class and impr
Makefile	Dont delete html or html/.git as this
README.md	Update README.md
README.md	

Sample .travis.yml

```
1 language: cpp
2
3 os:
4   - linux
5   - osx
6
7 compiler:
8   - g++
9   - clang
10
11 script: make && ./runtests.pl
```

Status of Personal codebase

build passing coverage 75%

Updated automatically

Results of CI Actions

Default branch

master

Updated 16 days ago by jrdoneal

✓

Your branches

gh-pages


Updated 16 days ago by jrdoneal

✓


All checks have passed

2 successful checks

✓

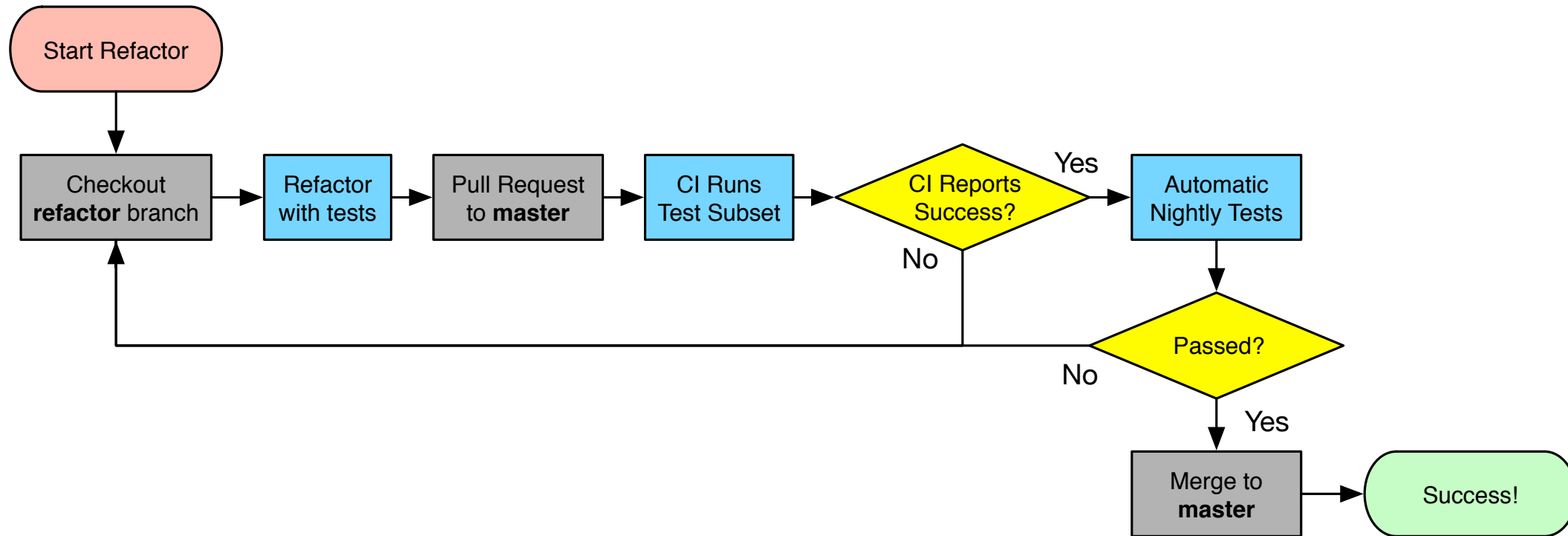
continuous-integration/travis-ci/push — The Travi...

✓

coverage/coveralls — Coverage remained the same...

Putting it all together

Toy CI Workflow



Other resources

Software testing levels and definitions:

http://www.tutorialspoint.com/software_testing/software_testing_levels.htm

Working Effectively with Legacy Code, Michael Feathers. The legacy software change algorithm described in this book is very straight-forward and powerful for anyone working on a code that has insufficient testing.

Code Complete, Steve McConnell. Includes testing advice.

Software Carpentry: <http://katyhuff.github.io/python-testing/>

Tutorial from Udacity: <https://www.udacity.com/course/software-testing--cs258>

Papers on testing:

<http://www.sciencedirect.com/science/article/pii/S0950584914001232>

https://www.researchgate.net/publication/264697060_Ongoing_verification_of_a_multiphysics_community_code_FLASH

Resources for Trilinos testing:

Trilinos testing policy: <https://github.com/trilinos/Trilinos/wiki/Trilinos-Testing-Policy>

Trilinos test harness: <https://github.com/trilinos/Trilinos/wiki/Policies--%7C-Testing>