COLLEGE OF ENGINEERING
NUCLEAR ENGINEERING & RADIOLOGICAL SCIENCES
UNIVERSITY OF MICHIGAN

Lecture 23
Unit Testing

Prof. Brendan Kochunas
11/25/2019

NERS 590-004

MICHIGAN ENGINEERING
UNIVERSITY OF MICHIGAN

Outline

- Motivation
- Review of Testing
- Definitions
- Code Verification
- Solution Verification
- Validation & Prediction

If unit testing is new to you it might seem abstract?

- What do I test?
- How do I test it?
- How do I define a unit test?
- What are some of the challenges of writing a unit test?

Today's Learning Objectives

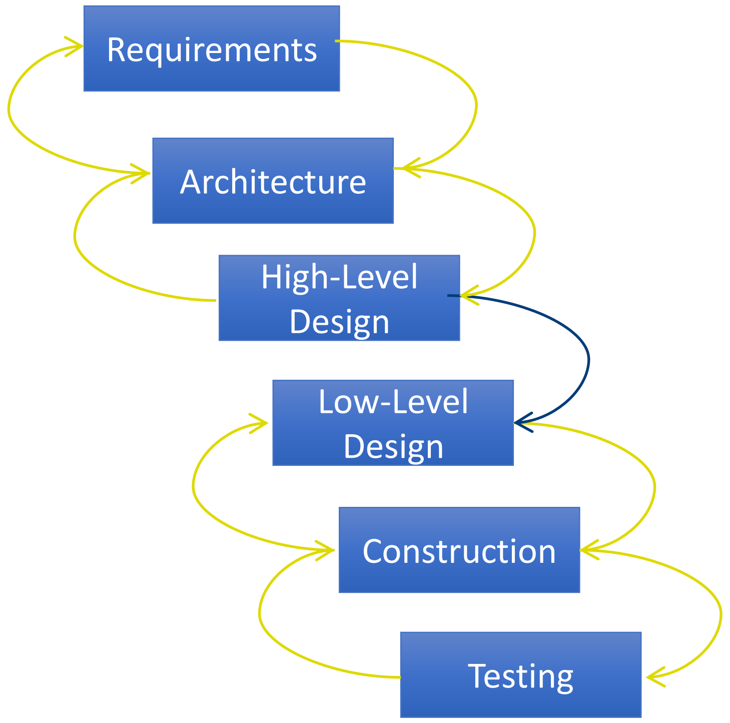
- Understand how to approach writing a unit test
- Work through a concrete example
- Become aware of unit testing frameworks

Further Reading

- McConnell, Steve. *Code Complete*. Second edition., Microsoft Press, 2004.

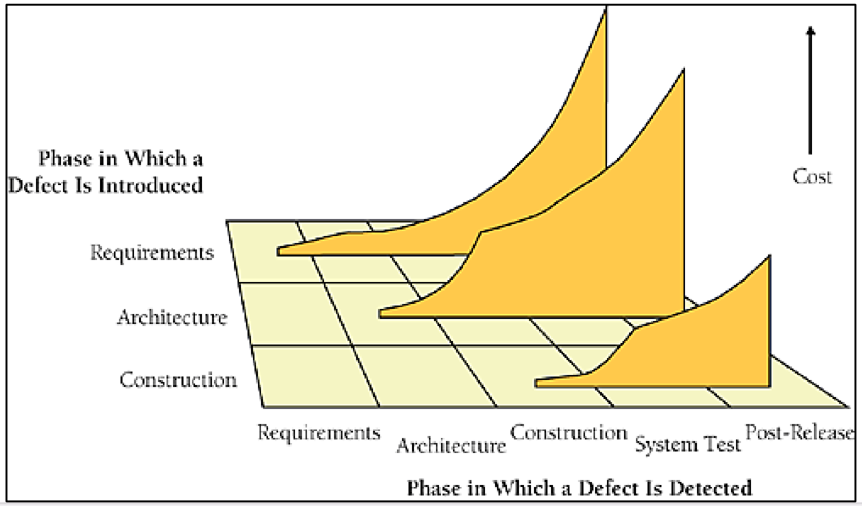
- https://search.lib.umich.edu/catalog/record/012121444
- Specifically, chapter: 22

Preliminaries



Cost of Fixing Defects

Time Introduced	Requirements	Architecture	Construction	System Test	Post-Release
Requirements	1x	3x	5-10x	10x	10-100x
Architecture	--	1x	10x	15x	25-100x
Construction	--	--	**1x**	**10x**	**10-25x**



The Role of Testing

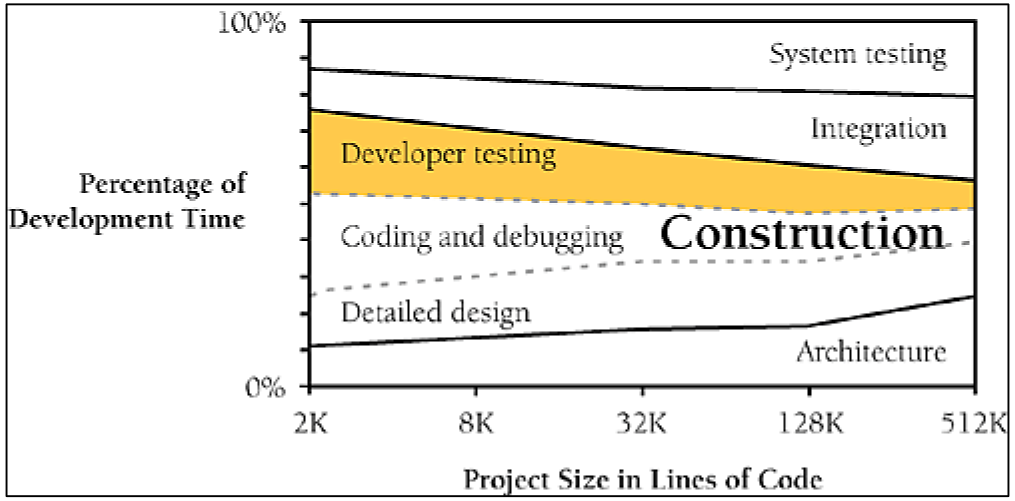
- The goal of testing is counter to normal development activities
  - Trying to "break" the code—not add features
  - Have to assume you'll find errors in your code (otherwise you won't)
- ***Unit Testing*** is the execution of a complete class, routine, or a small program that has been written by a single programmer or team of programmers, that is ***tested in isolation*** from the more complete system.
- Goal of unit testing: make sure code is doing what it is supposed to
- Testing can never completely prove the absence of errors.
- Testing by itself does not improve software quality

Approaches to Testing

Time in relation testing

How much time developing tests?

- "Rule of thumb" is ~50%
  - But this is misleading as it often includes debugging
- Real number is around ~25%



When to develop tests?

- As early as possible!
- Writing test now or later takes the same amount of time
- Having test cases first allows you to detect defects earlier, and more easily
- Writing tests first, gets you thinking more deeply about requirements and design
  - Identifies problematic/incomplete requirements
- During construction and before integration
  - If you develop 3 routines and integrate them, and there's a problem—where is it?

Recommended Approaches

- Plan tests at requirements stage—or as early as possible
- Test for each relevant requirement
- Test for design concerns (then plan at design stage)
- Use “basis testing” and “data-flow” testing

- Keep a checklist of the kinds of errors that are made on the project
  - For next level projects

Limitations

- Developer tests tend to be “clean tests”
  - Recommendation is 5 “dirty tests” for every clean test
- Tend to have an optimistic view of our testing
  - Your code is not as well tested as you think
- Tend to skip more complex testing
  - “Corner” cases are hard to think, but can be equally problematic
- Testing is not a magic bullet. It takes time.

Basis Testing

- Make sure your tests cover all execution paths with the minimal set of cases
  - Test cases are designed around program control flow

Case	**Test Description**	**Test Data**
1	Nominal case	All boolean conditions are true
2	The initial *for* condition is false	*numEmployees < 1*
3	The first *if* is false	*m_employee[id].governmentRetirementWith-held >=MAX_GOVT_RETIREMENT*
4	The second *if* is false because the first part of the *and* is false	*not m_employee[id].WantsRetirement*
5	The second *if* is false because the second part of the *and* is false	*not EligibleForRetirement(m_employee[id])*
6	The third *if* is false	*not EligibleForPersonalRetirement(m_employee[id])*

Data Flow Testing

- Idea based on assumption that data usage is at least as error prone as control flow
- Three states
  - Defined – Data has been initialized, but not yet used
  - Used – data has been used as an argument or right of assignment operator
  - Killed – In an undefined state
- Combine with control flow enter and exited
- Develop test cases around data state combinations e.g.
  - Defined-defined, defined-exited, killed-used, etc.

Unit Testing Frameworks

Evaluating a Unit Test Framework

- Minimal amount of work needed to add new tests
- Easy to modify existing tests
- Supports setup & teardown (e.g. Fixtures)
- Handles exceptions and crashes well
- Good assert functionality
- Timing information (catch performance errors, rather than correctness)
- Output and integration with other tools

Fortran

- FUnit
  - https://github.com/Goddard-Fortran-Ecosystem/pFUnit

- Futiliy
  - https://github.com/CASL/Futility/blob/master/src/Futility_DBC.h
  - https://github.com/CASL/Futility/blob/master/src/Futility_DBC.f90
  - https://github.com/CASL/Futility/blob/master/src/UnitTest.h
  - https://github.com/CASL/Futility/blob/master/src/UnitTest.f90

C++

- CppUnit
  - C++ port of JUnit
- Boost.Test
  - https://www.boost.org/doc/libs/1_45_0/libs/test/doc/html/utf.html
- CxxTest
- Catch
- Google Test
  - https://github.com/google/googletest

Python

- Unittest Library
  - Python Standard library
  - https://docs.python.org/3/library/unittest.html
- Full taxonomy
  - https://wiki.python.org/moin/PythonTestingToolsTaxonomy

Java

- JUnit
  - There’s just one

Examples