

UNIT TESTING FOR MERE MORTALS

Philip Japikse (@skimedic)

skimedic@outlook.com

www.skimedic.com/blog

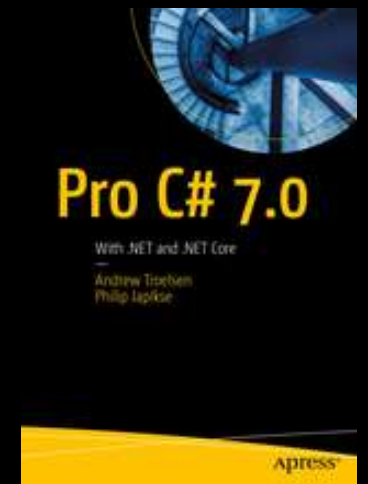
Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, PSM II, PSD

Consultant, Teacher, Writer



Phil>About()

- Director of Consulting/Chief Architect
- Author: Apress.com (<http://bit.ly/apressbooks>)
- Speaker: <http://www.skimedic.com/blog/page/Abstracts.aspx>
- Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, PSM II, PSD
- Founder, Agile Conferences, Inc.
 - <http://www.cincydeliver.org>
- President, Cincinnati .NET User's Group



UNIT TESTING

“The main thing that distinguishes legacy code from non-legacy code is tests, or rather a lack of tests.”

– Michael Feathers

UNIT TESTING MOTIVATION

➤ Cost of correcting a defect (by phase)¹:

➤ Requirements = \$1,000

➤ Design = \$455

➤ Coding = \$977

➤ Unit testing targets coming -> deployment and maintenance

➤ System Testing = \$1,136

➤ Maintenance = \$14,102

¹B.Boehm and V. Basili, "Software Defect Reduction Top 10 List", IEEE Computer, January 2001

WHY REALLY?

- The Team
 - Confidence
 - Courage
 - Cadence

IT'S *NOT* ABOUT TESTING

- Tests are used to drive design of the API
 - Leading to smaller, cleaner code base
 - Confirm success of the API with a rapid feedback loop
- Less Code
 - Only develop enough to meet the requirements
- Cleaner Design
 - Code is written in small increments in direct response to need

T/BDD BENEFITS

- Higher Code Coverage

- Once code is developed, often there isn't time in the schedule to go back and improve coverage

- Measurable Impact of Future Changes

- How many tests break with a change?

DEFINITIONS

UNIT OF WORK/UNIT TESTS

- Unit of Work
 - Smallest testable part of an application.
- Unit Test
 - Code used to validate units of work

UNIT TEST “CODE” OF HONOR

- Be Independent of all other unit tests
- Return the System Under Test to it's original state
- Note: Integration tests should follow these rules as well

TYPES OF UNIT TESTS

- State Testing
 - Easiest tests to write and execute
 - Asserted with value based semantics
- Behavior-based Testing
 - Verification of Behavior of SUT

COVERAGE

- Code Coverage
 - Measures lines of code executed by the Unit Tests
- Use Case Coverage
 - Test edge cases, exception handling

FAKES, STUBS, AND MOCKS

- Fakes have working implementations
 - Hardcoded, not suitable for production
- Stubs provide canned answers
- Mocks pre-programmed with expectations
 - Create a specification
 - Record behavior

WHY MOCK OBJECTS?

- Test Isolation
- Conditions that are difficult to reproduce
- Objects that are
 - Slow to execute or setup
 - Not yet coded
 - Expensive to call (e.g. external services)
- Objects that introduce noise that could occlude test results

TDD VS BDD

- Test Driven Development
 - Test blocks of code
- Behavior Driven Development
 - Test Behaviors

SUPPORTING PROCESSES

SOURCE CODE CONTROL

- Commit early and often
 - EVERY TIME you are green!
 - Update after every check in
- Run tests on each commit (CI)

AUTOMATED BUILD PROCESS

- Run all unit tests with every build
 - Builds should be run at least twice a day (striving towards Continuous Integration)
 - Failed Test = Failed Build = Doughnuts
- Integration tests should be run at least once a day

HANDLING SQUIRRELS

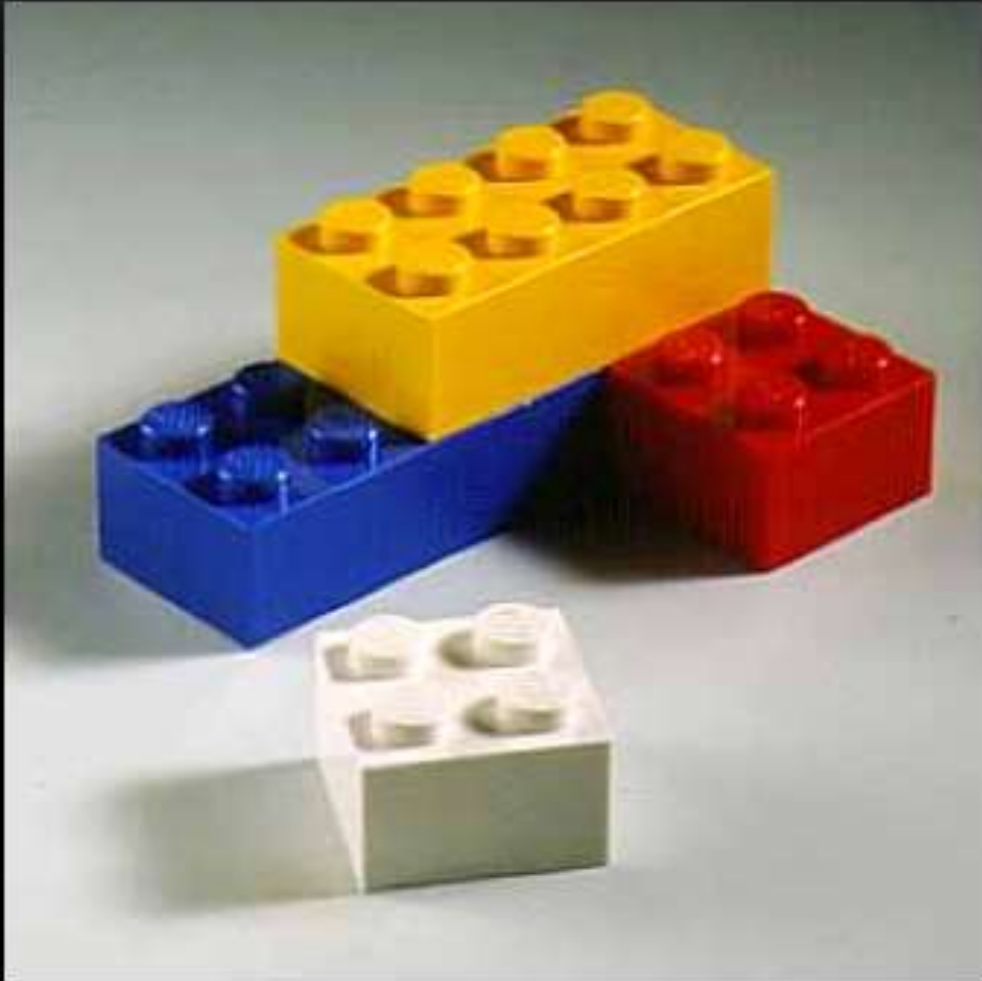
- Computers can multi-thread, people cannot
- Keep list of To-Dos
 - As ideas come up, write them down
 - Tackle them in order of confidence
- Finish what you start!
 - Don't context switch

T/BDD

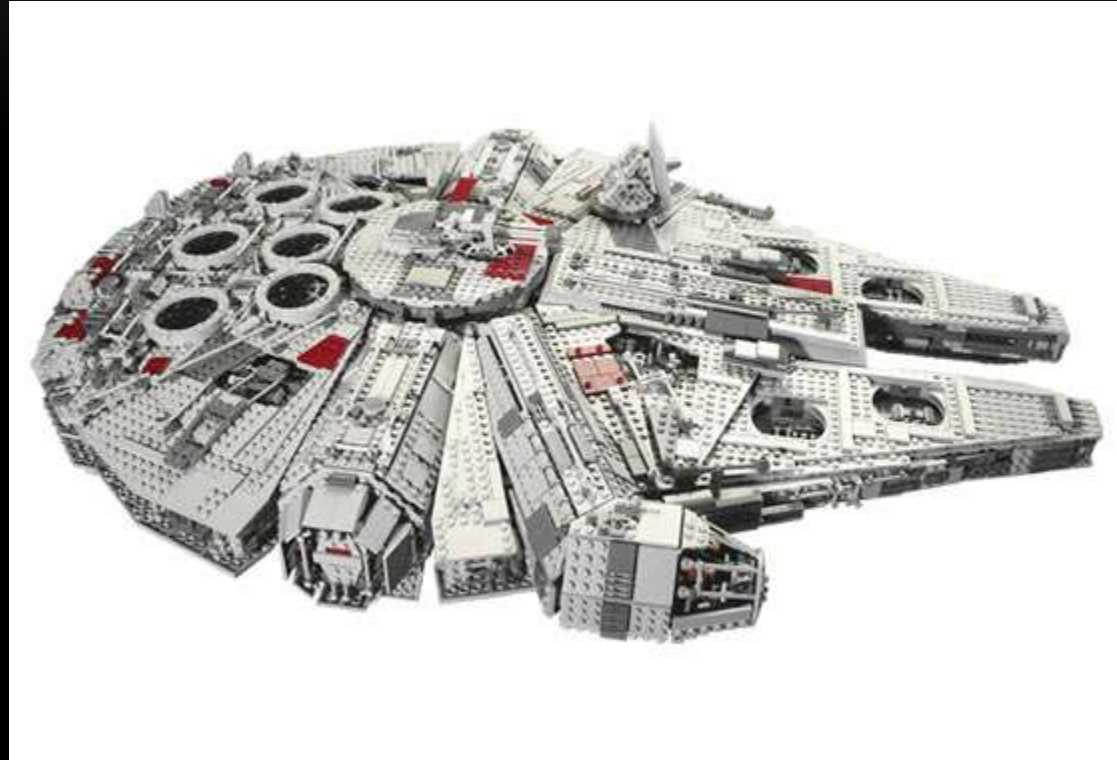
TEST STRUCTURE

- Arrange
 - Create all dependencies (mocks)
 - Instantiate the System Under Test
- Act
 - Execute the method to be tested
 - Highlander Principle
- Assert
 - Verify Results

TEST DRIVEN DEVELOPMENT/DESIGN (TDD)



BEHAVIOR DRIVEN DEVELOPMENT



THE T/BDD MANTRA

- **Red** – write a breaking test (failed build = broken test)
- **Green** – write just enough code to have the test pass
- **Refactor** – eliminate any duplicate code (or anything else that isn't self-documenting or is overly complex)

RED

- Write the test
 - Use BDD naming even if in TDD paradigm
 - “Should_Add_Two_Integers”
 - Add assertion(s)
 - Write just enough code to enable the build
 - Failed build is a failed test

GREEN

- Write just enough code to pass the test
- Expand test one Use Case at a time
 - Rerun all tests - if any of the Use Cases fail the test, continue to flush out the target code
 - Refactor target code AND test code along the way
- Continue until complete Use Case Coverage accomplished

REFACTOR

- Remove:
 - Hardcoded values
 - Duplicate Code (Keep DRY)
 - Any code that is not self documenting or unclear
- This also applies to the tests
 - Or does it?
 - Moist is ok

BARRIERS TO ENTRY

COMMON FRICTION POINTS

- It's Hard.
- <Fill in the blank> doesn't want me writing "twice as much code".
- I don't have time for it.
- My code doesn't have bugs.
- That's QA's job.
- Any others?

SUMMARY

- Where does T/BDD fit?
 - Anywhere you are writing code
- Where does TED fit?
 - Generated Code*
 - New/Updated Frameworks
- FTW:
 - Defect reduction (“elimination”)
 - QA team shifts to proactive mode
 - Increased Agility and faster Time To Market

DEMO DEWNO

TDD/BDD/Mocking

Contact Me

skimedic@outlook.com

www.skimedic.com/blog

www.twitter.com/skimedic

<http://bit.ly/skimediclyndacourses>

<http://bit.ly/apressbooks>

www.hallwayconversations.com

Questions?



Thank You!