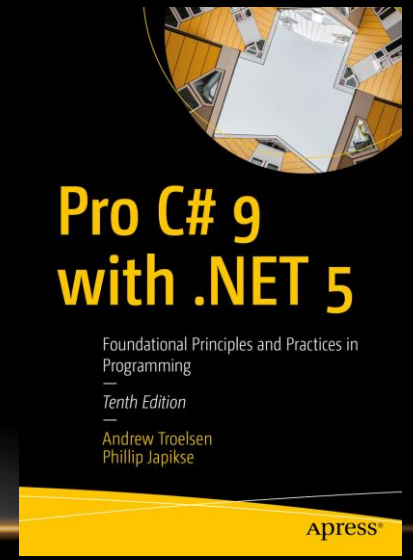
UNIT TESTING FOR MERE MORTALS

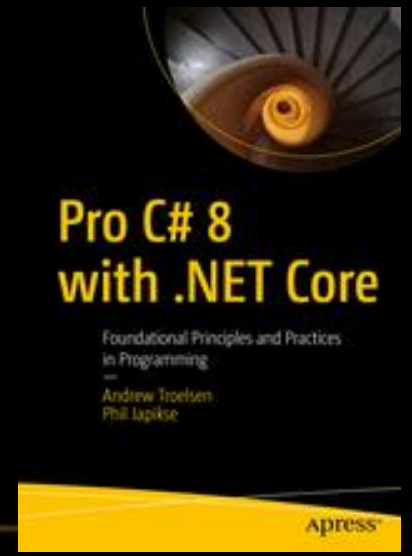
Philip Japikse (@skimedic)
skimedic@outlook.com
CTO, Author, Teacher
Microsoft MVP, ASPInsider, PST, PSM II, PSD

All slides copyright Philip Japikse http://www.skimedic.com

Phil.About()

- CTO/Chief Architect, Pintas & Mullins
- Author: Apress.com (http://bit.ly/apressbooks)
- Professional Scrum Trainer (PSF, PSD)
- Speaker: http://www.skimedic.com/blog/page/Abstracts.aspx
- Microsoft MVP, ASPInsider, PST, PSM II, PSD
- Founder, Agile Conferences, Inc.
  - http://www.cincydeliver.org
- President, Cincinnati .NET User's Group





All slides copyright Philip Japikse http://www.skimedic.com

UNIT TESTING

All slides copyright Philip Japikse http://www.skimedic.com

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

– Michael Feathers

All slides copyright Philip Japikse http://www.skimedic.com

UNIT TESTING MOTIVATION

- Cost of correcting a defect (by phase)[1]:
  - *Requirements* = $1
  - *Design* = $455
  - *Coding* = $977
    - Unit testing targets coding -> deployment and maintenance
  - *System Testing* = $3,136
  - *Maintenance* = $14,102

[1]B.Boehm and V. Basili, “Software Defect Reduction Top 10 List”, IEEE Computer, January 2001

All slides copyright Philip Japikse http://www.skimedic.com

WHY REALLY?

- The Team
  - Confidence
  - Courage
  - Cadence

All slides copyright Philip Japikse http://www.skimedic.com

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

All slides copyright Philip Japikse http://www.skimedic.com

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?

All slides copyright Philip Japikse http://www.skimedic.com

DEFINITIONS

All slides copyright Philip Japikse http://www.skimedic.com

UNIT OF WORK/UNIT TESTS

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

All slides copyright Philip Japikse http://www.skimedic.com

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

All slides copyright Philip Japikse http://www.skimedic.com

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

All slides copyright Philip Japikse http://www.skimedic.com

COVERAGE

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

All slides copyright Philip Japikse http://www.skimedic.com

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

All slides copyright Philip Japikse http://www.skimedic.com

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

All slides copyright Philip Japikse http://www.skimedic.com

TDD VS BDD

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

All slides copyright Philip Japikse http://www.skimedic.com

SUPPORTING PROCESSES

All slides copyright Philip Japikse http://www.skimedic.com

SOURCE CODE CONTROL

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

All slides copyright Philip Japikse http://www.skimedic.com

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

All slides copyright Philip Japikse http://www.skimedic.com

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

All slides copyright Philip Japikse http://www.skimedic.com

T/BDD

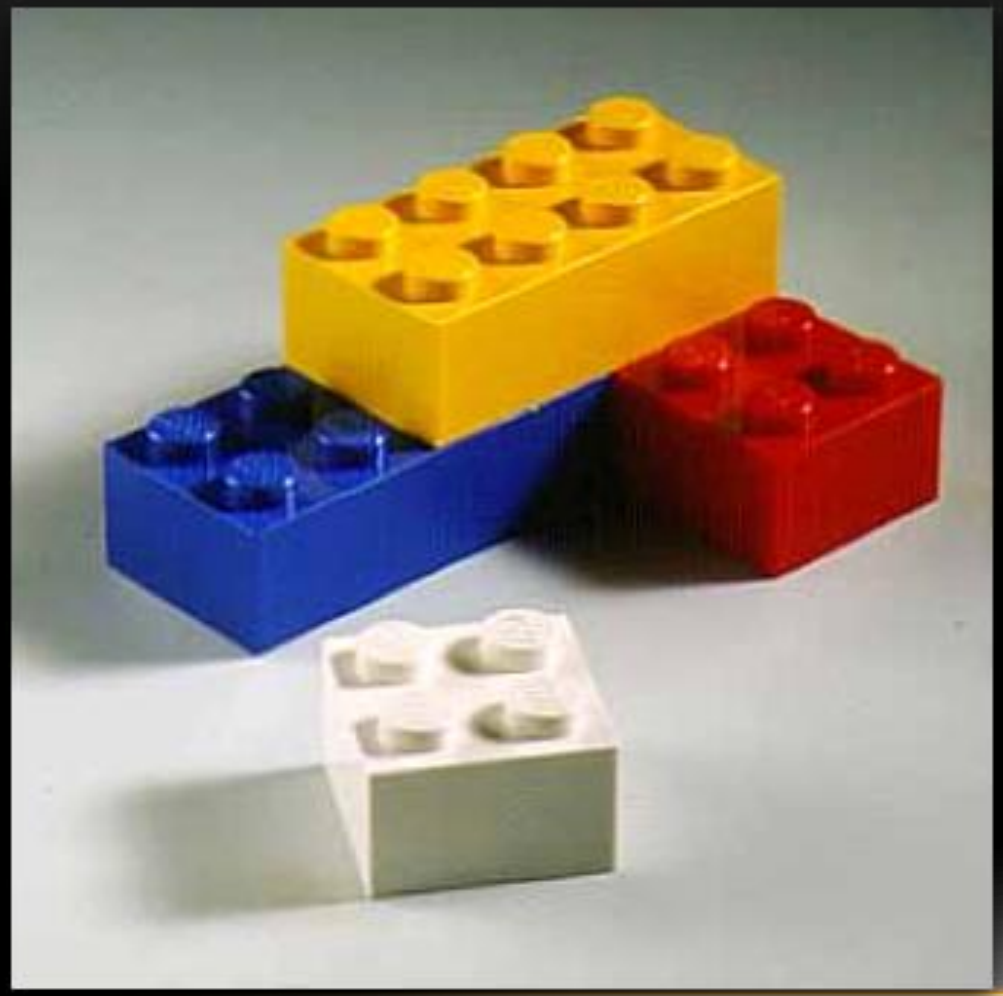
All slides copyright Philip Japikse http://www.skimedic.com

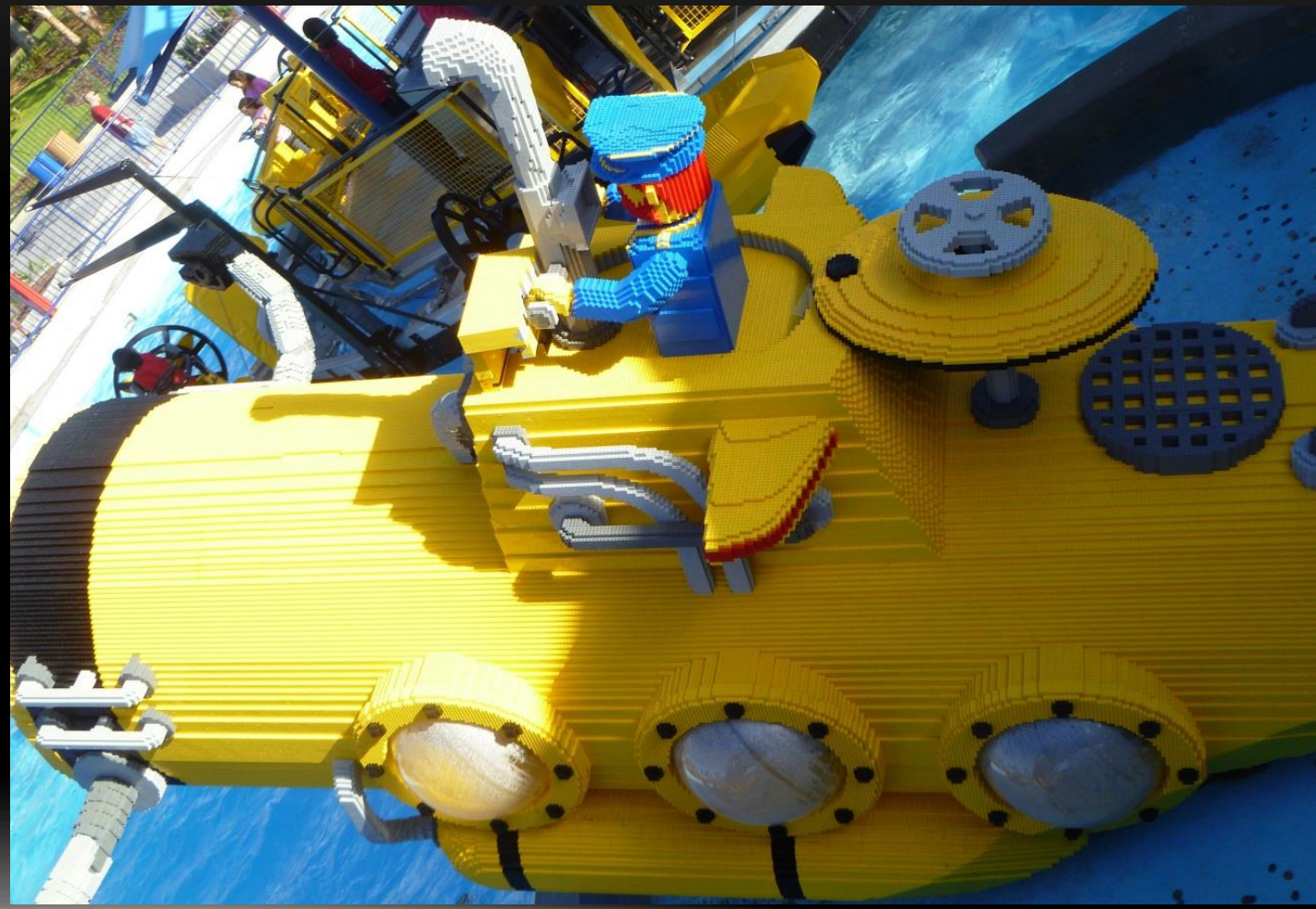
TEST STRUCTURE

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

All slides copyright Philip Japikse http://www.skimedic.com

TEST DRIVEN DEVELOPMENT/DESIGN (TDD)





All slides copyright Philip Japikse http://www.skimedic.com

BEHAVIOR DRIVEN DEVELOPMENT

All slides copyright Philip Japikse http://www.skimedic.com

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)

All slides copyright Philip Japikse http://www.skimedic.com

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

All slides copyright Philip Japikse http://www.skimedic.com

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

All slides copyright Philip Japikse http://www.skimedic.com

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

All slides copyright Philip Japikse http://www.skimedic.com

BARRIERS TO ENTRY

All slides copyright Philip Japikse http://www.skimedic.com

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?

All slides copyright Philip Japikse http://www.skimedic.com

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

All slides copyright Philip Japikse http://www.skimedic.com

DEMO

TDD/BDD/Mocking

All slides copyright Philip Japikse http://www.skimedic.com

Contact Me

skimedic@outlook.com
www.skimedic.com/blog
www.twitter.com/skimedic

http://bit.ly/apressbooks

Questions?

Thank You!

https://github.com/skimedic/presentations

All slides copyright Philip Japikse http://www.skimedic.com