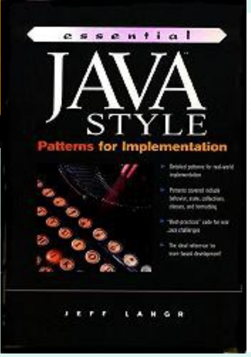
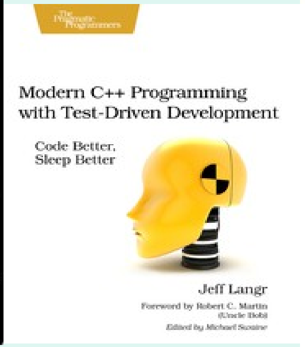
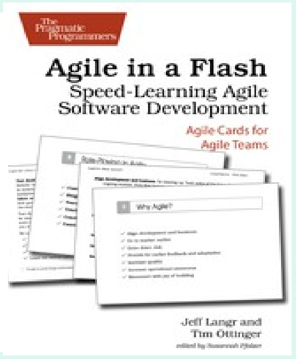
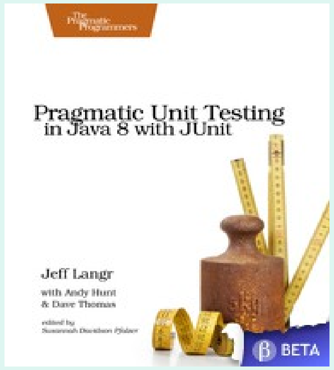
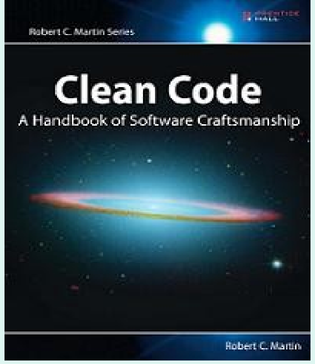
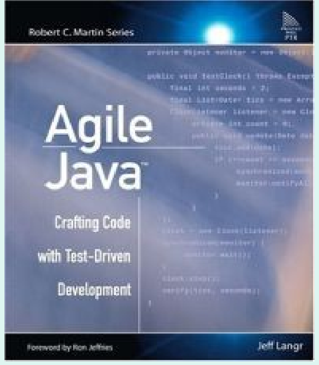



The Pragmatic Programmers
Modern C++ Programming with Test-Driven Development
Code Better, Sleep Better
Jeff Langr
Foreword by Robert C. Martin (Uncle Bob)
Edited by Michael Swaine

The Pragmatic Programmers
Agile in a Flash
Speed-Learning Agile Software Development
Agile Cards for Agile Teams
Jeff Langr and Tim Ottinger
edited by Susannah Pfalzer

The Pragmatic Programmers
Pragmatic Unit Testing in Java 8 with JUnit
Jeff Langr
with Andy Hunt & Dave Thomas
edited by Susannah Davidson Pfalzer
BETA




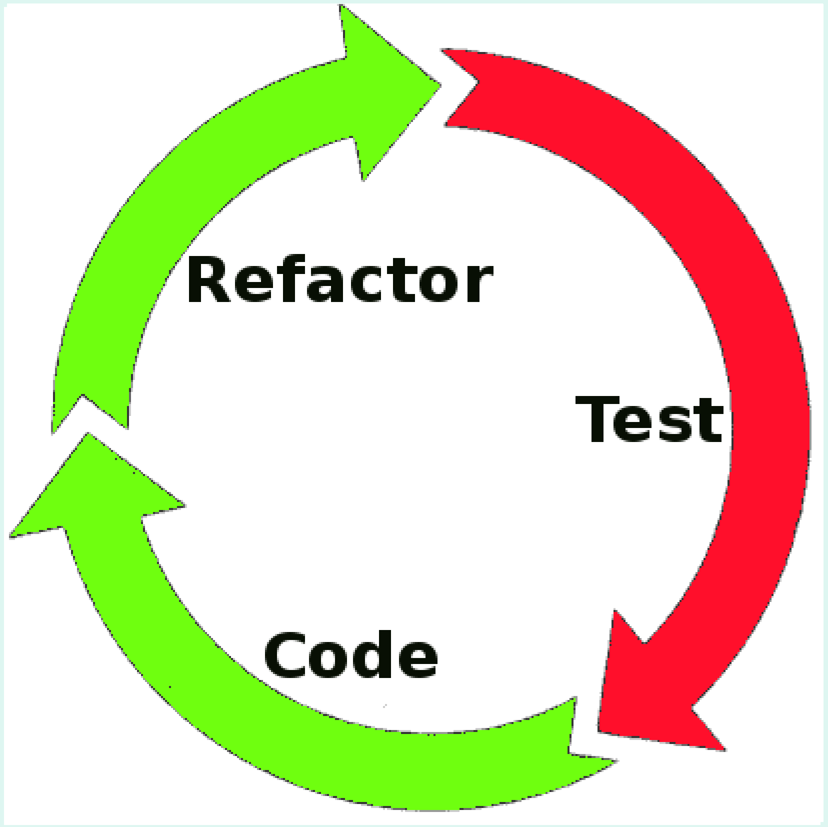
Test-Driven Development and Refactoring in Java

Langr Software Solutions

Consulting and Training for Software Teams

Test-Driven Development?

- An incremental design technique



Doing TDD Well

It's just code!

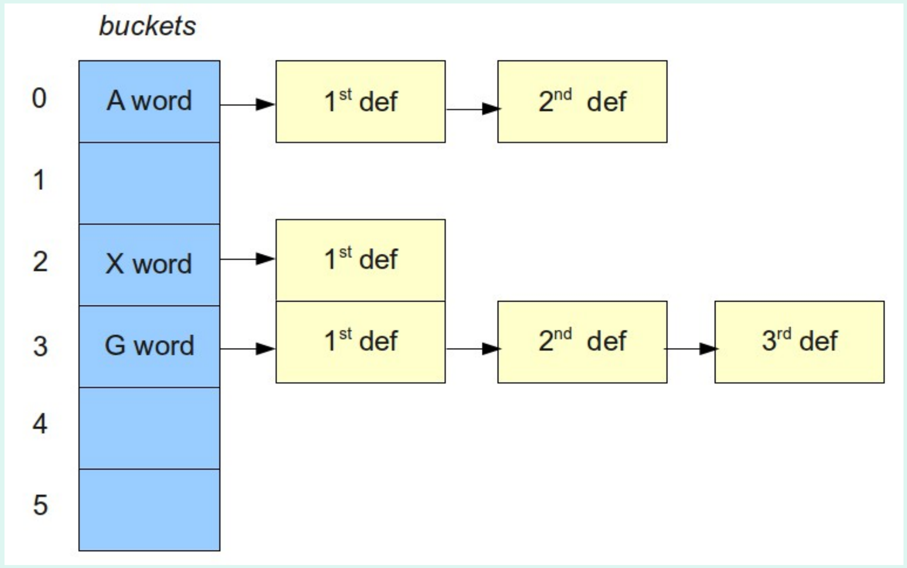
- Test behavior, not methods
- Incrementalism—tiny steps!
  - Hardcoding → more coverage
- Stick to the cycle
  - Always see red
  - Always refactor
- Specification by example

TDD Exercise: MultiMap

- Associates multiple values with a key
- Interface
  - void map.put(K, **V**)

    Single value at a time!
  - **List<V>** get(K)

    Returns a list!
  - int size()
  - boolean isEmpty()
  - int countOfAllValues()
- Throws an exception on a null key



Test-After Development (TAD)

- Allows some refactoring
- Coverage levels up to ~75%
- No direct design impact
- Can reduce defects
- Can be treated as separate task

Test-Driven Development (TDD)

- Enables continual refactoring
- Coverage approaching 100%
- Helps incrementally flesh out a design
- Significantly reduces defects/debug cycles
- Part of the coding process
- Clarifies, documents requirements
- Continual progress, consistent pacing
- Continual feedback and learning
- Sustainable

Unit Testing

- $$
- Only a piece of the puzzle

Where Do I Start?

- Object creation isn't a bad idea
  - What next?
- Group exercise:
  - HashSet
  - The Bowling Game
  - SQL generator

Kata: Roman Numeral Converter

- http://codekata.pragprog.com
- Given positive int from 1 to 4000
  - Answer Roman numeral equivalent

Behavior-Driven Naming

- Consider:
  - *somethingHoldsTrueWhenCondition*
    or
  - *whenConditionThenSomethingHoldsTrue*
- Review test names holistically
- Rename continually!

```
@Test public void something()
@Test public void create()
@Test public void defaultCreate()
@Test public void isEmptyOnDefaultCreation()
```

Improving Tests

- Single behavior tests
  - → Behavior-driven naming
- Test abstraction
- AAA: Arrange, Act, Assert
  - – Bill Wake
- Correlate results with context

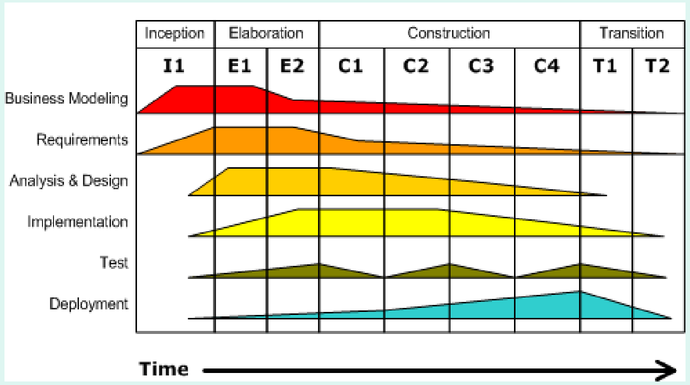
Exercise: find & fix smells

Exercise: Improving Tests

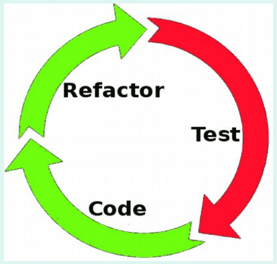
- Find and fix test smells in the codebase
- Paraphrase cleaned tests to your pair

Software Development

- Activities:
  - Analysis, design, coding, testing, review, documentation, planning, deployment, ...

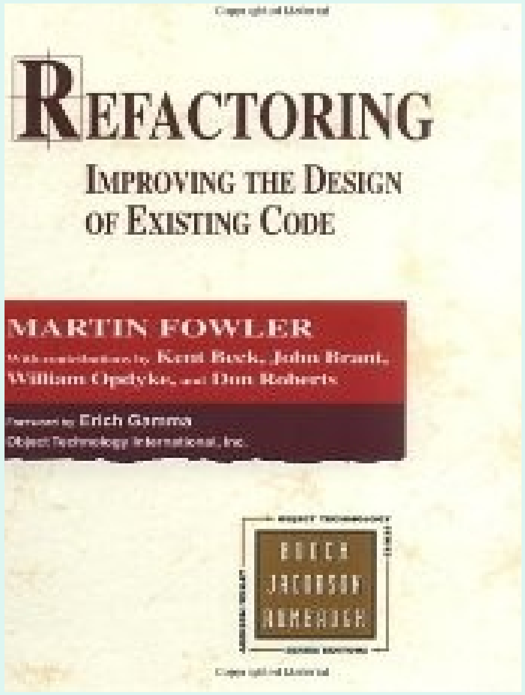


- Agile:
  - *All activities all the time*
- TDD:
  - Continual testing
  - Continual design

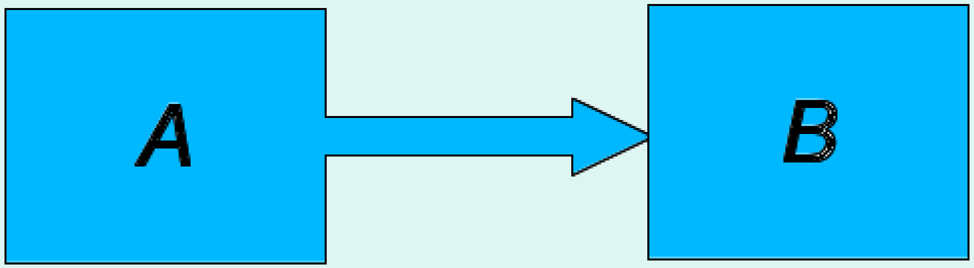


- Up-front design?

Refactoring?



REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International, Inc.
BOOCH JACOBSON RUMBAUGH



- Code transform
- Same behavior

Extract Method

Turn related code fragment into its own method, using a name that explains the method's purpose

- Core catalog pattern
  - Very explicit steps
- What's the value?

Composed Methods

- Comprehension time
- Minimize / decipher defects
- Reuse / code reduction
- Potential algorithm improvement
- Identify performance issues

Demo / Exercise

- Look for refactoring opportunities
  - Extract method
  - Replace temp with query
  - Renames (almost free!)
- Follow detailed Fowler steps
- Run tests as often as possible
- Do at least two method extractions

Feature Envy

Method sends messages to other objects more than itself.

- Why a problem?
- Apply **Move Method**
  - [Fowler1999], p142
- May require Replace Temp With Query

Backing Into Tests

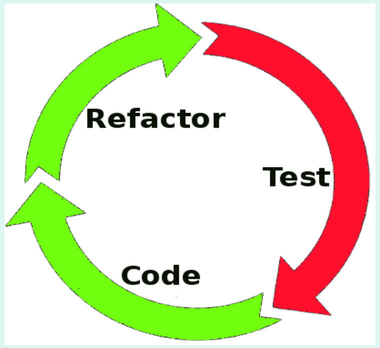
- Moved code not directly tested!
  - Cover with tests to document
  - Difficult? Reconsider.

Exercise: Feature Envy

- Locate and fix feature envy
- Add characterization tests as needed

Guidelines for Refactoring

- Single-goal
- Run *all* tests
- Never skip!
  - Avoid having to ask



Design Drivers

- Code smells (Fowler)
- Simple design rules (Beck)
- Classic design principles (Meyer, Martin)
- Design patterns (Gamma et al)

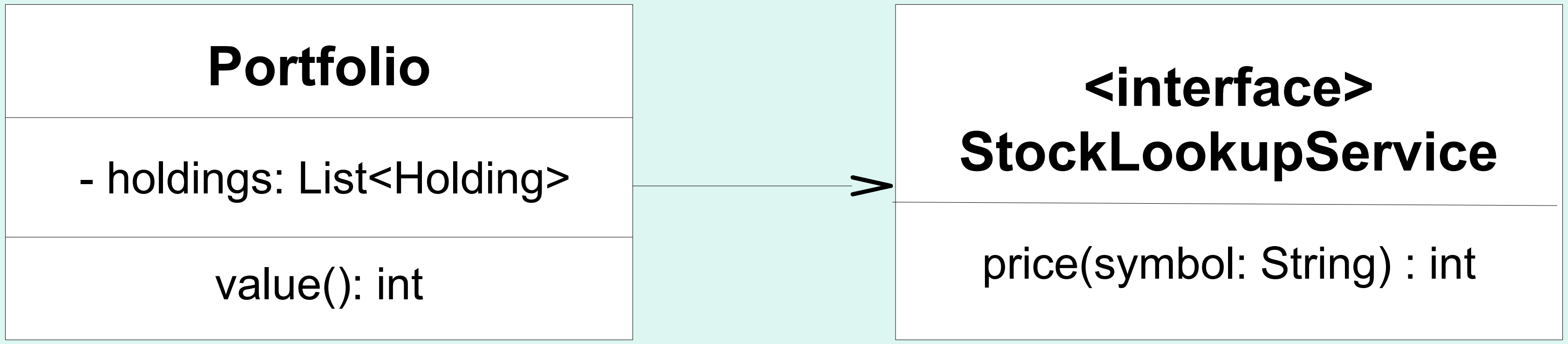
- All point to same place?

- ***Learn it all!***

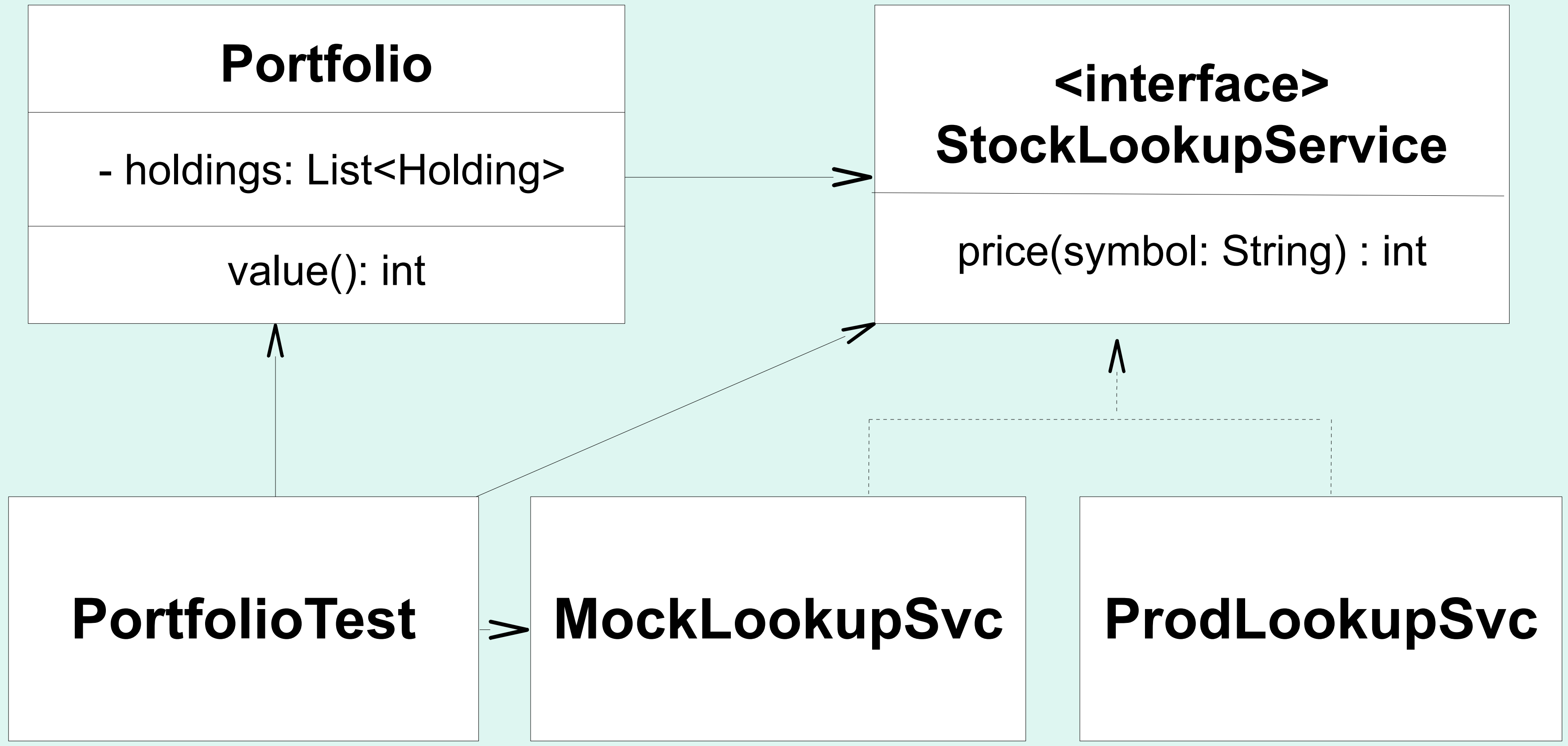
41 Build Superior Systems with Simple Design

- All tests must pass
- No code is duplicated
- Code is self-explanatory
- No superfluous parts exist

Testing Challenge: Portfolio

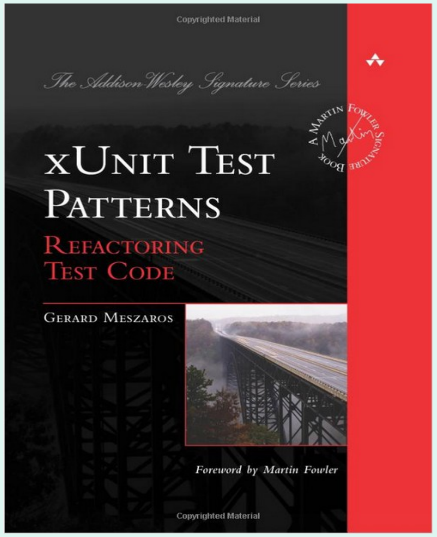


A Test Double



Test Double Terms

- Stub: dumb emulation
- Mock: self-verifies [Feathers2005]
- Spy: captures values
- Fake: whole collaborator emulation



Source: XUnit Patterns, Gerard Meszaros

Exercise: Test Doubles

(Hand-Crafted!)

- Only add patrons passing credit check:

```
interface CreditVerifier {
   boolean verify(String cardNumber);
}
```

- Extra credit:
  – Write audit record when credit is bad

Using Test Doubles

- First define:
  - Nested / top-level class
  - Self shunt
  - Anonymous inner class stub / lambda
- Then inject:
  - Constructor / setter
  - Factory
  - Getter override
  - Tool (Dagger, Spring DI, Guice)

Mock Tools

- jMock, EasyMock, Mockito
  - PowerMock: static support for EasyMock & Mockito
  - JMockit: static and final support
- History suggests more changes coming
  - Lesson: Avoid too much dependency on tool/idiom!

Mockito

http://code.google.com/p/mockito

- Stub before execution:

```
StockLookupSvc s =
  org.mockito.Mockito.mock(StockLookupSvc.class);
when(s.price("IBM")).thenReturn(50);
```

- Verify afterward:

```
Inventory inventory = mock(Inventory.class);
Scanner scanner = new Scanner(inventory);
scanner.scan("123")
verify(inventory).add("123");
```

Exercise: Mock Tools

- Use Mockito to create a CreditVerifier mock

Mockito Miscellany

- Shorthand:

```
@org.mockito.Mock private Inventory inv;
@Before public void init() {
   org.mockito.MockitoAnnotations.initMocks(this); }
```

- Argument matchers allow wildcard matching:

```
verify(inv).add(org.mockito.Matchers.anyString());
```

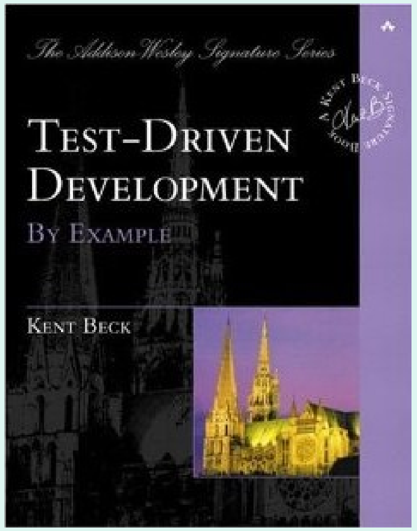
- Exceptions:

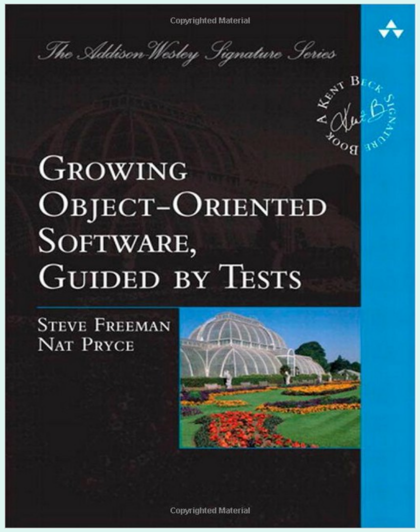
```
when(s.price("")).thenThrow(new Exception());
doThrow(new Exception()).when(inv).add("");
```

- Can use `spy` to allow partial mocks
  – Should only need rarely—legacy/constrained design

TDD Schools

- Classic *aka Detroit/Chicago*
  - TDD: more algorithmic approach
  - Tests drive code specific → general
- London
  - Roles, responsibilities, & interactions
  - *“Fake it until you make it”*
- Fowler, M. “Mocks Aren't Stubs”


The Addison-Wesley Signature Series
Test-Driven Development
By Example
Kent Beck
A Kent Beck Signature Book


Copyrighted Material
The Addison-Wesley Signature Series
A Kent Beck Signature Book
Growing Object-Oriented Software, Guided by Tests
Steve Freeman
Nat Pryce
Copyrighted Material

Classic School

Use test doubles when you:

- Can't control collaborators
  - e.g., emulate exception
- Have dependency on:

  volatile / slow / non-existent class
- Have many/hard-to-construct collaborators

London School

- Mock all collaborators
  - Fake it 'til you make it!
- “Build out” and explore design, outside-in

Pragmatic School

- Mock when you must
- Great at subsystem boundaries
- Verify protocols / sequencing
- Avoid re-testing details

Test Doubles

- Inherently introduce concessions
  - Tests tightly couple to impl; can inhibit refactoring
  - ...Or design becomes slightly more complex
  - Create “holes” in coverage
- Isolate / minimize use
  - But prefer testability

Wrap-Up Exercise

Test-Drive New Scan Mode

- "Inventory reconciliation"
  - Triggered by scanning branch ID in inventory state
  - Collects scanned holdings
  - Pressing "Complete" produces report
    - And returns to inventory state
    - List scanned items not in inventory + patron & vice versa
- Work from outside in
  - Stub all service calls outside scanner subsystem
  - Including calls to any new service methods needed

TDD is a skill.

Practice, practice, practice.

TDD

Miscellaneous Topics

Interesting Stuff

- Timer (10-minute rule)
- Infinitest
- Randori-style dojos
  - Pair demos on-screen TDD katas
  - See http://kaksles.org/2006/03
- TDD as if you meant it (Braithwaite)
  - tddAsIfYouMeantIt.txt in distro

Cucumber

- Ruby based; derived from RSpec
- **Feature**: Scanner

  In order to

  As a patron

  I want to check out a book

  Scenario: Patron scans book

  Given I have scanned my card with id 123

  When I scan book with id QA123:1

  Then my list of holdings contains QA123:1

- Implemented in Ruby *a la* JBehave

BDD

- Core principles:
  - Business & technology use common language
  - Deliver identified, verifiable business value
  - Big up-front P/A/D: diminishing returns
- Programming:
  - Mock focus: verify expectations about object interaction
  - Promotes use of narrow, role-defining IFs
  - Promotes multiple fixtures per target
  - JBehave, JDave, beanSpec, Instinct

JBehave

```
public class MultiMapBehaviorWhenEmpty { // ...
   @Given("an empty map")
   public void theMapIsCreated() {
      multiMap = new MultiMap<Object,Object>();
   }
   @When("I put a value $key->$value")
   public void iPutAValue(Object key, Object value) {
      multiMap.put(key, value);
   }
   @Then("it contains $key->list($value)")
   public void itContains(Object key, Object value) {
      assertThat(multiMap.get(key), hasItems(value));
   }

// run via script:
Given an empty map
When I put a value Word->Definition
Then it contains Word->list(Definition)
```

The Transformation Priority Premise

- Uncle Bob's theory—an heuristic in flux!
  - Next best step can be determined using the most “simple” transformation

http://cleancoder.posterous.com/the-transformation-priority-premise

http://thecleancoder.blogspot.com/2011/01/transformation-priority-and-sorting.html --> already presents a revised list!

- See tpp.txt in source base
- Choose items higher on priority list

- Now... redo the Roman numeral kata!

Transformation Priority List

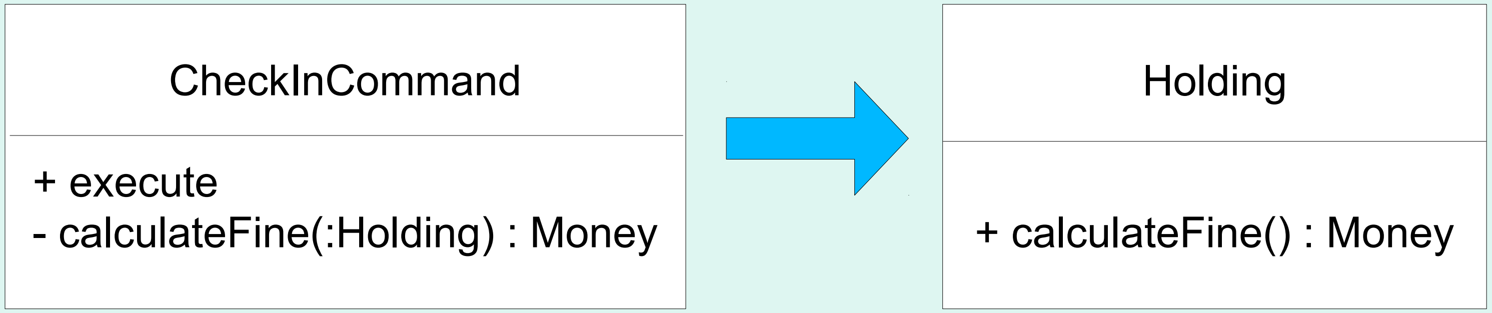
(Feb 2, 2011—per Uncle Bob)

```
({}->nil) no code at all->code that employs nil
(nil->constant)
(constant->constant+) a simple constant to a more complex constant
(constant->scalar) replacing a const. with a variable or an argument
(statement->statements) adding more unconditional statements.
(unconditional->if) splitting the execution path
(scalar->array)
(array->container)
(statement->tail-recursion)
(if->while)
(statement->recursion)
(expression->function) replacing expression w/ a function or algorithm
(variable->assignment) replacing the value of a variable.
(case) adding a case (or else) to an existing switch or if
```

Source: http://thecleancoder.blogspot.com/2011/02/fib-t-p-premise.html

Test Private Behavior?

- Often suggests need for redesign
- Other options:
  - Relax access specifier. No worries.
  - Might use reflection. Don't.



Accessing Private Data

- Verify may require inspecting “privates”

- Expose or relax access specifier
  - 'sok. Tests protect and document acceptable use
  - Don't “cheat” using reflection

Abstract Test Pattern

- Template method applied to tests
  - Runs each base test vs. all derivative impls
  - Base test declares abstract factory method:

    `abstract protected ScanStationState createSut();`

- *See:* ScanStationStateTest
  - `toStringSpecifiesStateName` run once for each of four state subclasses

Sustaining TDD

- Awareness of coverage, problem spots
  - Be careful!
- Education/sharing sessions
  - Brown bags
  - Randori dojos?
- Challenges/contests
- TDD-specific retrospectives
- Pair & paraphrase
  - *Three*+ sets of eyes always

Resources/References

- [Beck2002] Beck, Kent. **Test-Driven Development: By Example**. Addison-Wesley, 2002.
- [Feathers2005] Feathers, Michael. **Working Effectively With Legacy Code**. Prentice Hall, 2005.
- [Fowler1999] Fowler, Martin. **Refactoring: Improving the Design of Existing Code**. Addison-Wesley, 1999.
- [Freeman, 2009]. Freeman, S. and Pryce, N. **Growing Object-Oriented Software, Guided By Tests**. Addison-Wesley, 2009.
- [Langr2005] Langr, Jeff. **Agile Java: Crafting Code With Test-Driven Development**. Prentice Hall, 2005.
- [Langr2011] Langr, Jeff and Ottinger, Tim. **Agile in a Flash**. Pragmatic Programmers, 2011.
- [Langr2013] Langr, Jeff. **Modern C++ Programming With Test-Driven Development**, Pragmatic Programmers, 2013.
- [Langr2015] Langr, Jeff, et. al. **Pragmatic Unit Testing in Java 8 with Junit.** Pragmatic Programmers, 2015.
- [Martin2002] Martin, Robert C. **Agile Software Development: Principles, Patterns, and Practices**. Prentice Hall, 2002.