



**ICON**  
AGILITY SERVICES

**Test-Driven Development  
Foundations v1.1**

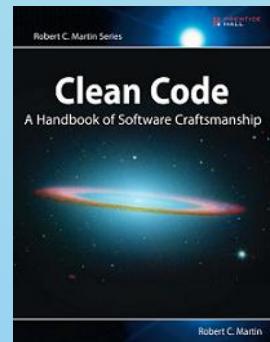
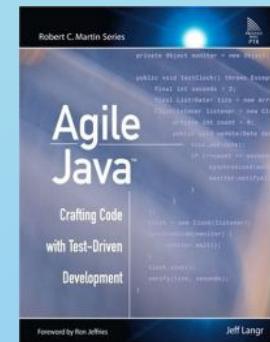
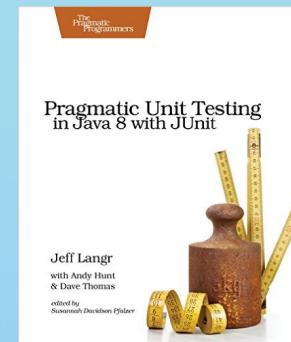
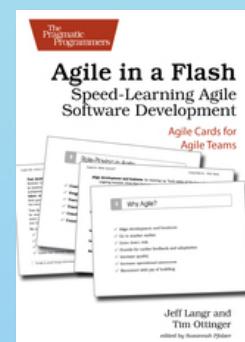
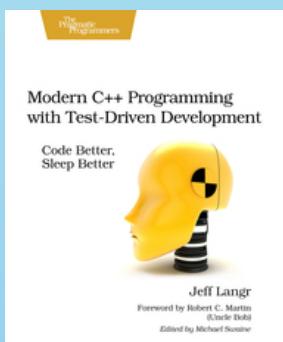
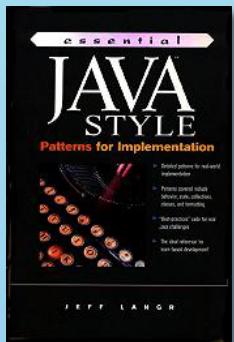
# Jeff Langr

[jeff@langrsoft.com](mailto:jeff@langrsoft.com)  
@JLangr



## Langr Software Solutions

SOFTWARE TRAINING/CONSULTING



# What's a Unit Test Look Like?

```
describe('something', () => {
  it('demonstrates some behavior', () => {
    // Arrange
    // Act
    // Assert
  })
})
```

The **AAA** mnemonic is courtesy of Bill Wake.

# A Sample Unit Test

```
describe('an auto', () =>
  const auto = new Auto()

  it('idles engine when started', () => {
    auto.depressBrake()

    auto.pressStartButton()

    expect(auto.RPM()).to.be.within(950, 1100)
  })
})
```

# What's an Assertion Look Like?

```
expect(someCondition).to.be.true
expect(idleSpeed).to.equal(1000)
expect(alphabetizedName).to.equal("Schmoo, Kelly Loo")
expect(alphabetizedName).to.match(/^S.*,/)
expect(idleSpeed).to.be.within(950, 1100)
expect(idleSpeed).to.not.equal(42)
expect(alphabetizedName).to.not.equal("Smelt"))
expect(someVariable).to.be.undefined
expect(someArray).to.have.lengthOf(3)
expect(someArray).to.be.empty
expect(someArray).to.include(42)
expect(someArray).to.deep.equal(['O', 'T', 'F', 'F', 'F', 'S'])
```

See <http://www.chaijs.com/api/bdd/>,  
<https://github.com/chaijs/deep-eql>

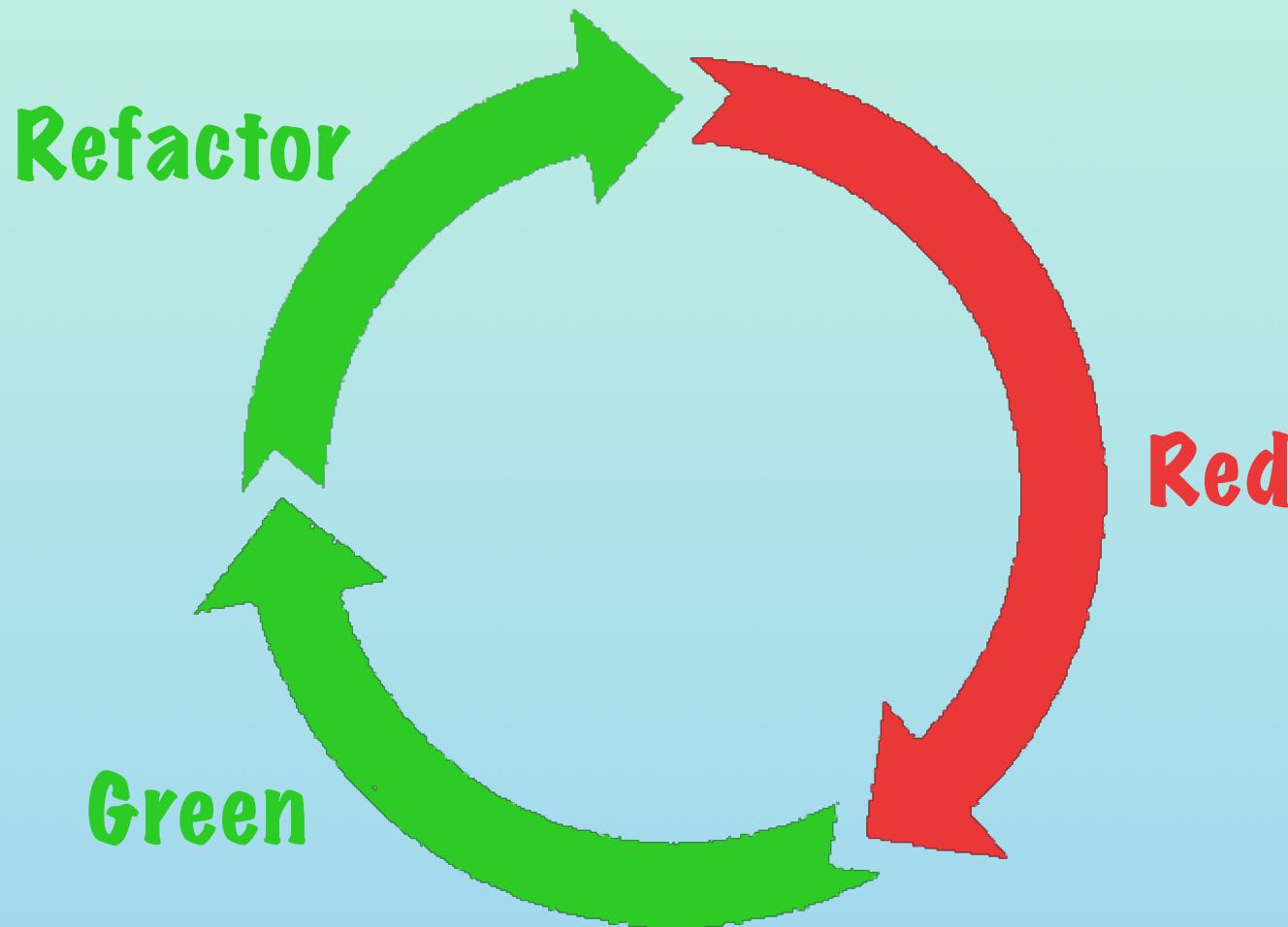
# Exercise: Simple Outcomes

---



Flesh out the tests in `./src/misc/basics.test.js`

# Test-Driven Development



*An incremental design technique*

# Exercise: (Test)-Code-Refactor

## TDD Paint by Numbers

---



See `./src/misc/name-normalizer.test.js`

# Core TDD Themes



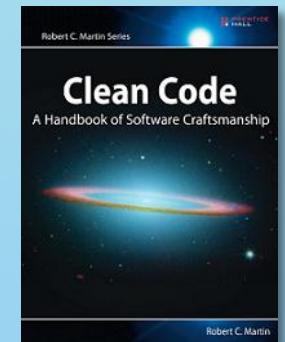
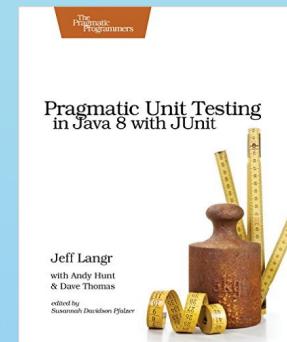
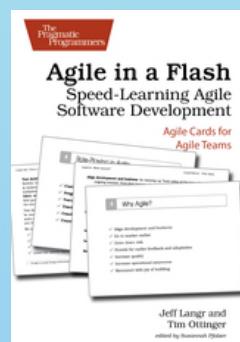
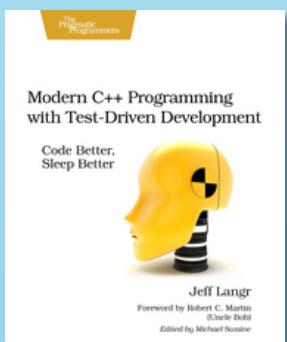
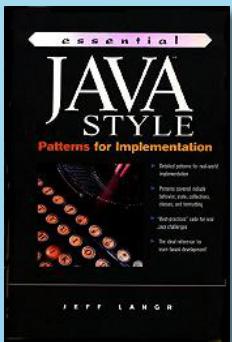
- Test-drive behavior
- Small increments
- Stick to the cycle
  - Always see red
  - Always refactor
- Specification by example

*It's just code!*

# Stepping Back: Writing Tests

**Langr**  
*Software Solutions*

SOFTWARE TRAINING/CONSULTING



# Area Calculator

Given an object with height & width,  
test-drive a function that calculates its  
area.



# ZOMBIES!

- **Zero**
- **One**
- **Many**
- **Boundaries**
- **Interface definition**
- **Exceptional behavior**
- **Simple Solutions & Scenarios**



<http://blog.wingman-sw.com/archives/677>

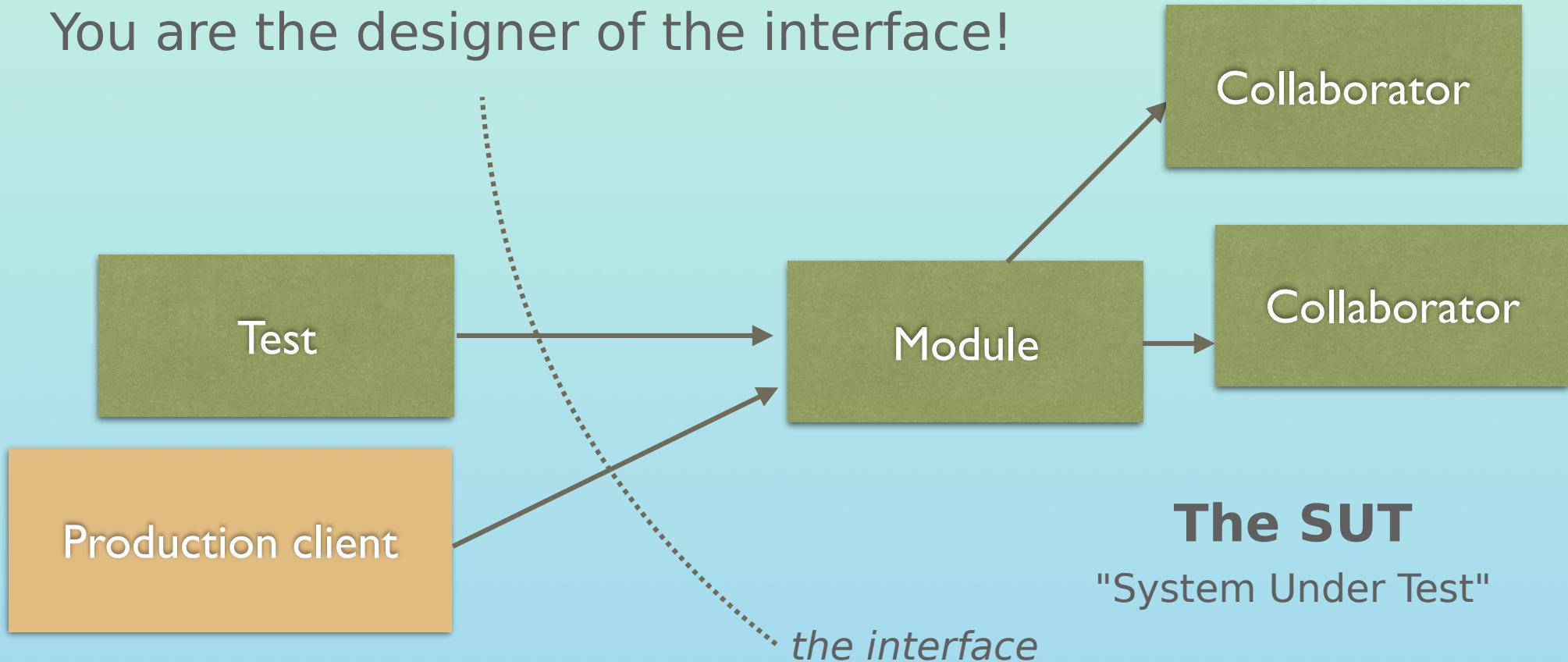
# Test Lists

---

- **Z:** Empty rectangle
- **O:** 1 x 1
- **M:** Larger sizes
- **B:** Maximum integer size
- **E:** Negative input

# Your Test Is the First Client

You are the designer of the interface!



- How does the client want to interact with the system?
- What sequence of events gets you into the right state?

# Structuring the Test

- **Arrange** Create a data structure & populate it
- **Act** Pass it to the area function & store the return
- **Assert** Compare against the return value

# Assert-First

---

Consider working from the outcome:

```
expect(areaOfEmptyRectangle)  
  .toEqual(0)
```

# Assert-First: Backing Up

---

```
const areaOfEmptyRectangle =  
  area(emptyRectangle)
```

```
expect(areaOfEmptyRectangle)  
  .toEqual(0)
```

# Assert-First: Backing Up

---

```
const emptyRectangle =  
{ height: 0, width: 0  
  
const areaOfEmptyRectangle =  
area(emptyRectangle)  
  
expect(areaOfEmptyRectangle)  
.toEqual(0)
```

# Weather Machine

The weather module is an OO class.

It accepts and stores new cities.

It can return the temperature for any one city (by interfacing with an external REST service).

It can return the average temperature for all cities.



Create a test list.

Write the average temperature test, assertion first.

# Exercise: Stock Portfolio Class

In-memory only

- is it empty or not?
- what is the count of unique symbols?
- given a symbol and # of shares, make a purchase
- how many shares exist for a given symbol?
- given a symbol and # of shares, sell the shares
- throw a RangeError when selling too many shares



"Tokyo Stock Exchange," courtesy <https://www.flickr.com/photos/31029865@N06/>  
License: <https://creativecommons.org/licenses/by/2.0/>

	09:00	11:30	12:30	15:00
T D K	3430	+85	トヨタ自	2522
キーエンス	18480	-80	ホンダ	2362
デンソー	2115	+38	スズキ	1613
ファナック	11790	+340	ニコン	1726
ローム	3565	+45	HOYA	1639
京セラ	6260	+120	キヤノン	3465
村田製	3960	+15	リコー	671
日東電	2819	+9	凸版印	575
三菱重	328	+2	大日印	756
日産自	696	+4	任天堂	10560
食品		+0.83	電力・ガス	56.43
エネルギー資源	113.35	+1.53	運輸・物流	
建設・資材		+0.62	商社・卸売	
素材・化学	97.74	+0.62	小売	
医療・		+0.22	銀行	64

\*\*\* Remember! \*\*\*

**Red:** Ensure test fails

**Green:** Implement no more code than necessary

**Refactor:** Clarify and eliminate all duplication

# A Functional Implementation

Store portfolio info in a state object:

```
{  
  holdings:  
  {  
    IBM: 20,  
    BAYN: 10  
  }  
}
```

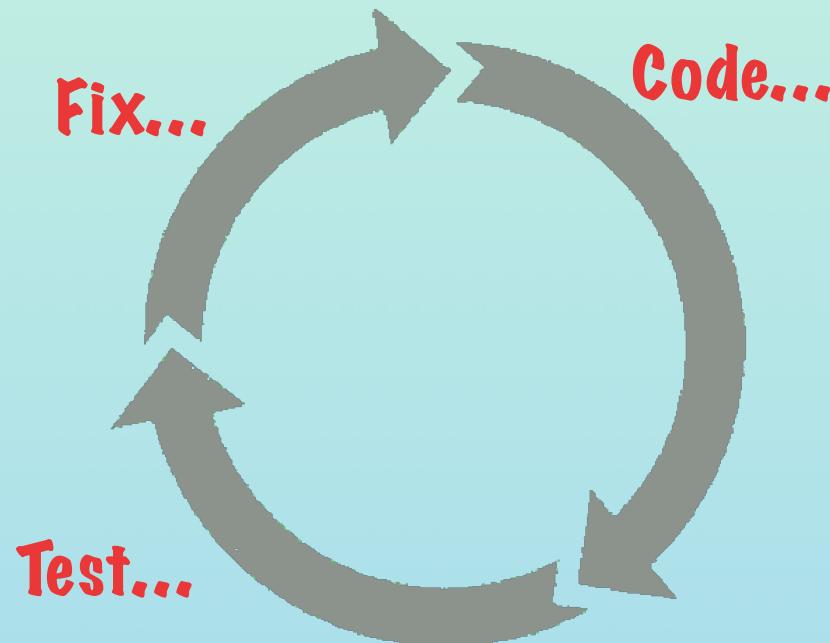
Pass state to each function, return updated state:

```
const portfolio = { holdings: [] }  
const newPortfolio = purchase(portfolio, "BAYN", 42)
```

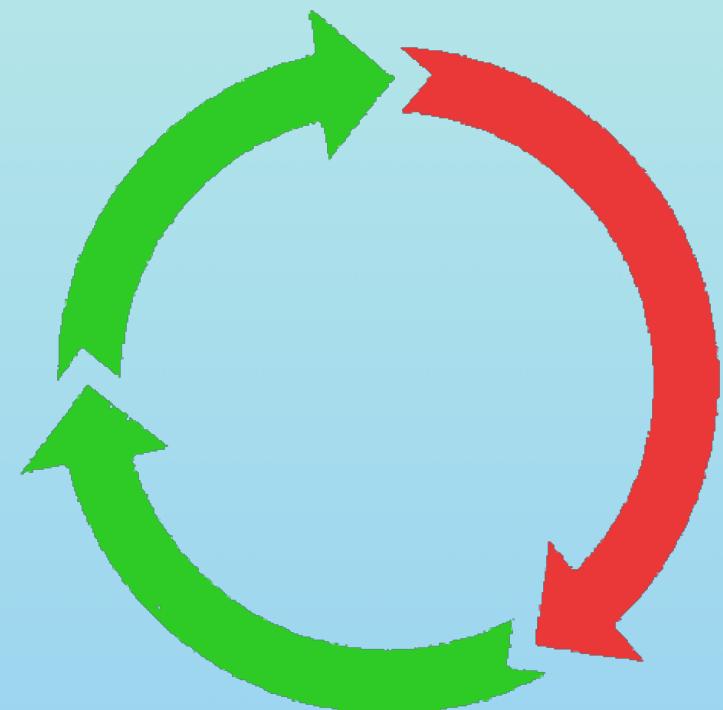
*Ensure your function creates no side-effects!*

i.e., **portfolio** should not change

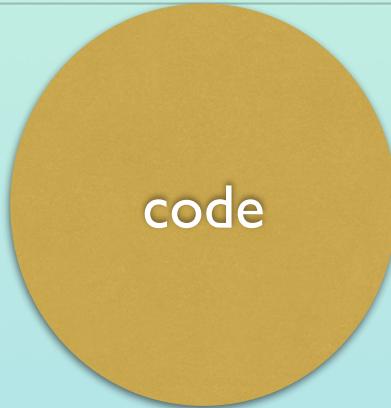
# TAD\*\* and TDD: What's the Difference?



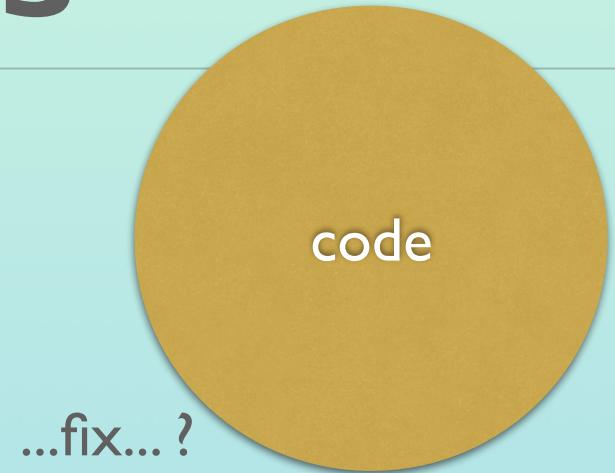
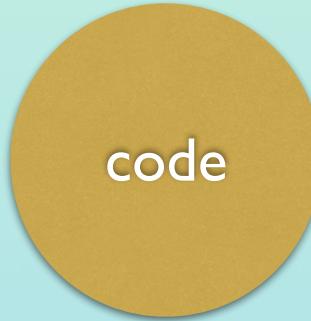
"Test-After Development"



# Small Increments



...fix... ?

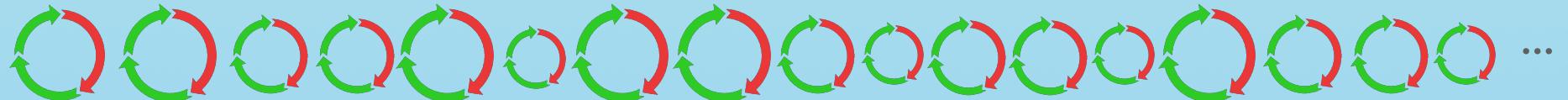


...fix...

---

TAD

VS.

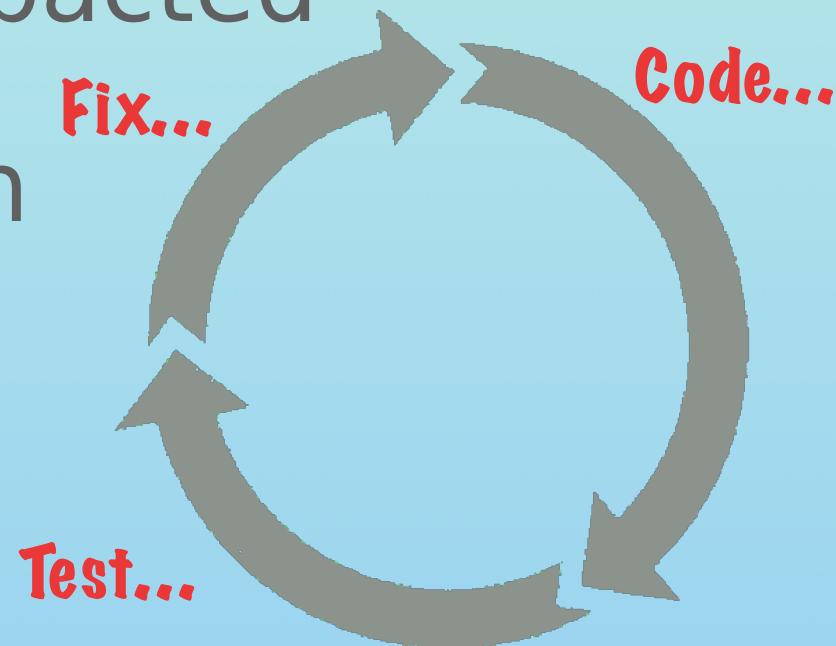


---

TDD

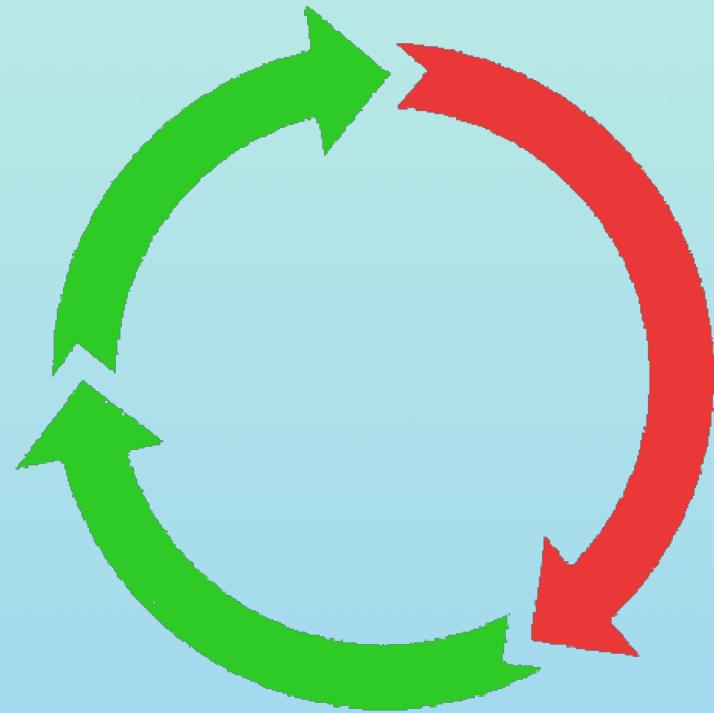
# Test-After Development (TAD)

- Some refactoring accommodated
- Coverage: ~70%
- Design not usually impacted
- Some defect reduction
- Separate task?

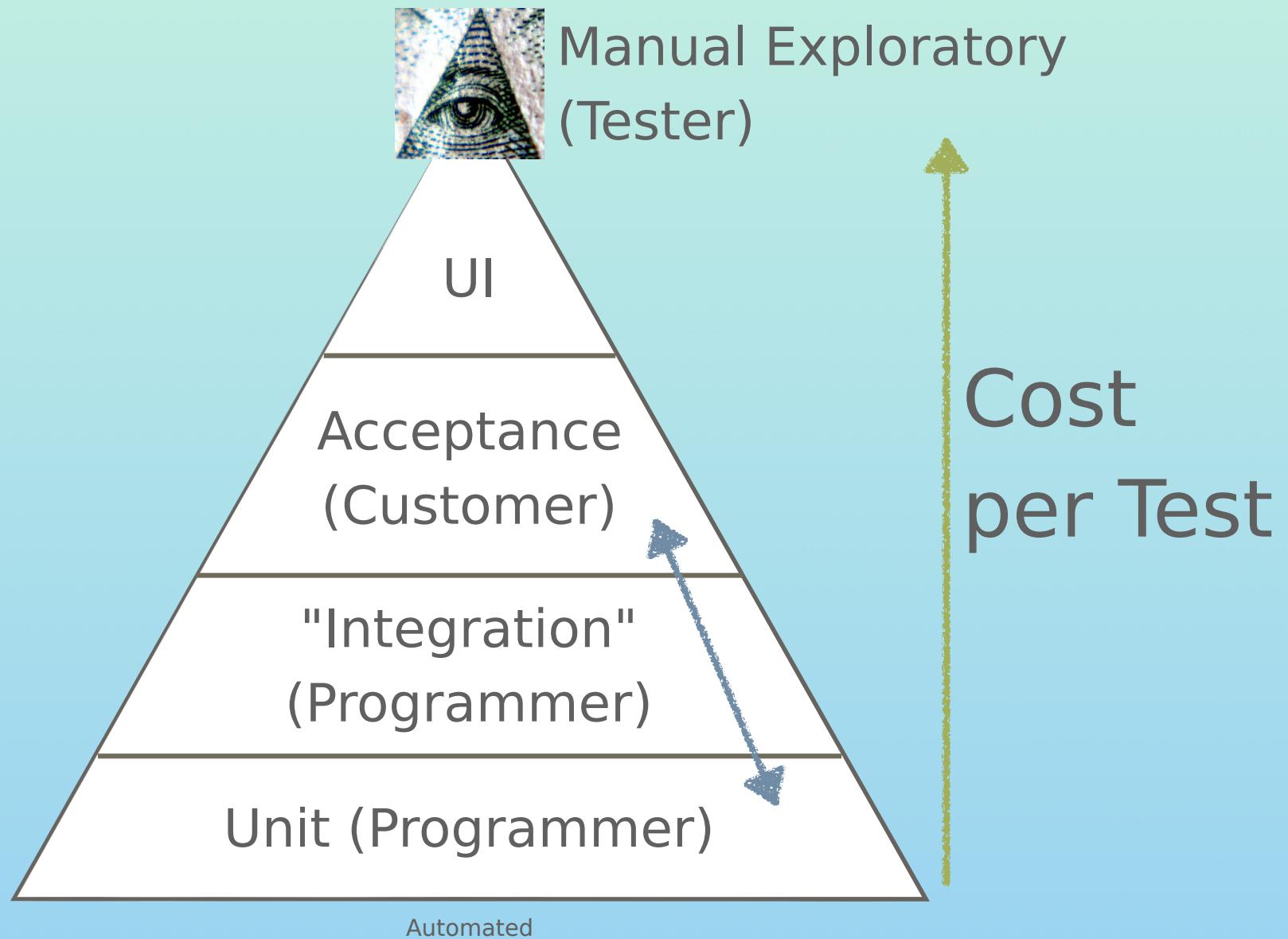


# Test-Driven Development (TDD)

- Continual refactoring
- Coverage for all intended features
- Incremental design shaping
- Significant defect reduction
- Minimized debugging
- Integral part of coding process
- Clarify / document needs / choices
- Continual forward progress
- Consistent pacing
- Continual feedback / learning
- Sustainable



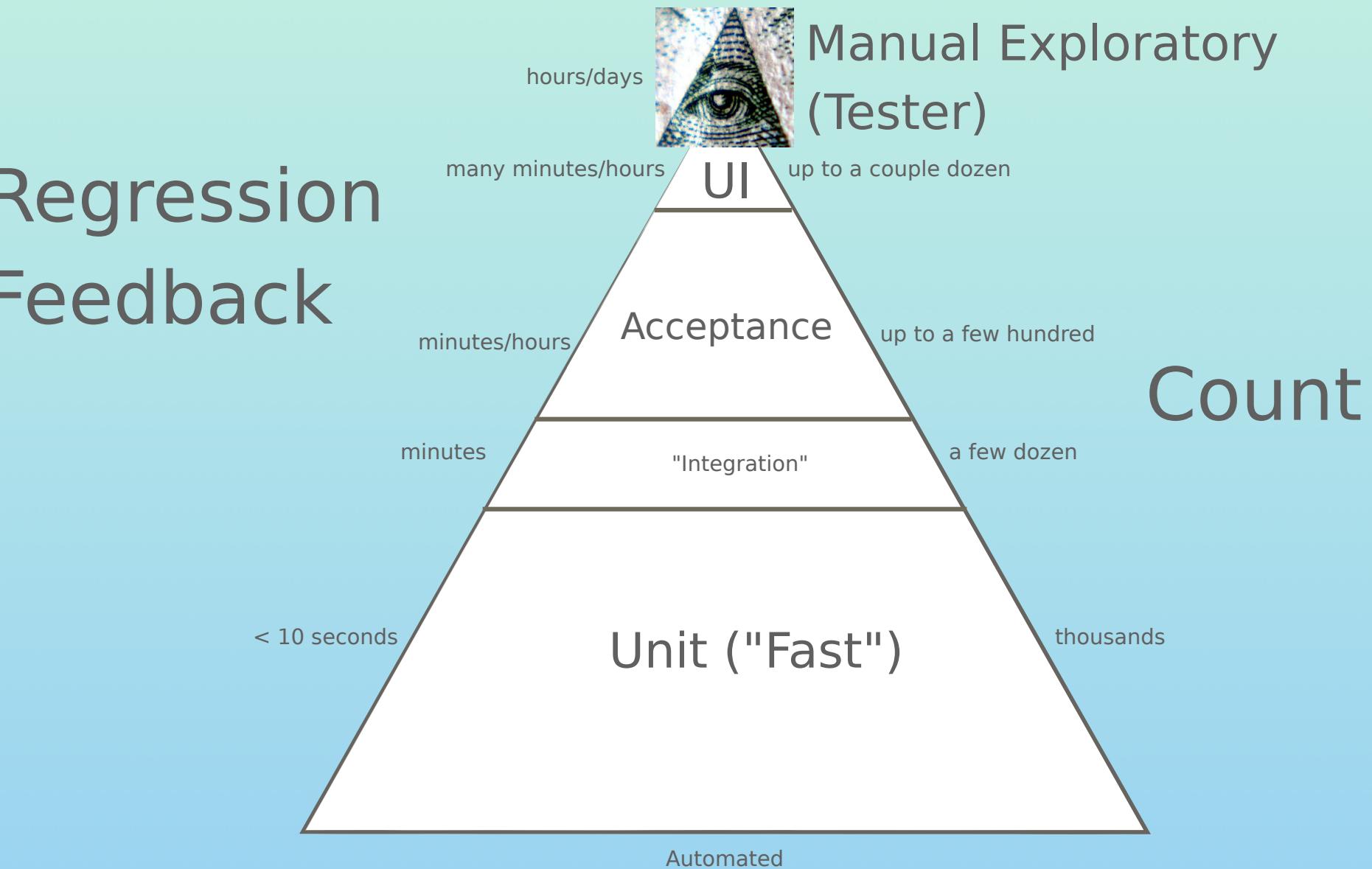
# Unit Testing: Insufficient!



# A Re-leveled Pyramid

Regression  
Feedback

Count



# Katas

<http://codekata.com>

## Other sites:

<https://exercism.io>

<http://cyber-dojo.org>

<https://sites.google.com/site/tddproblems/>

<https://www.codewars.com>

<https://github.com/jlangr/name-normalizer>

<https://github.com/emilybache>



# Roman Numeral Converter

Given a positive integer  
from 1 up to 4000,  
answer its Roman equivalent



# Feedback Q's

■ :-)

What have I learned of value?

■ %-|

What am I unclear about regarding the discussion / exercises?

■ ?:-/

What, if anything, did I find disturbing / disconcerting?

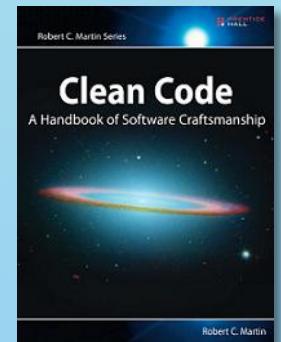
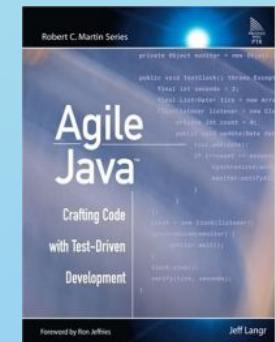
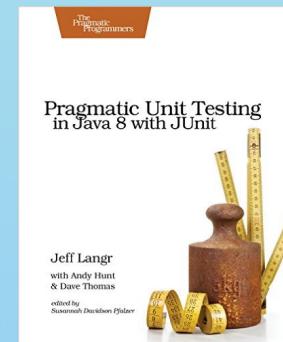
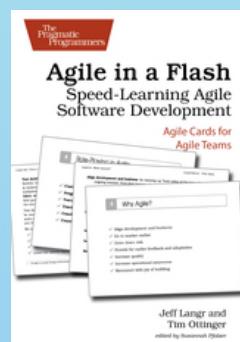
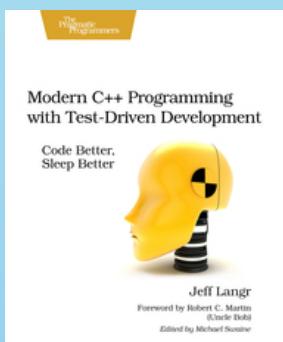
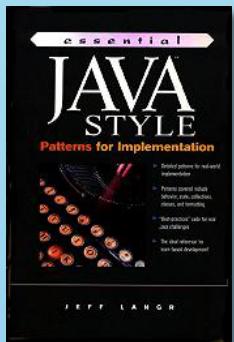
# Documentation



ICON  
AGILITY SERVICES

Langr  
*Software Solutions*

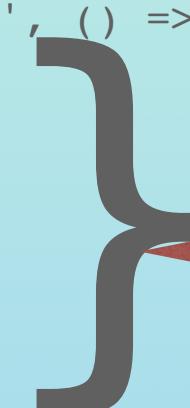
SOFTWARE TRAINING/CONSULTING



# Tests as Documents

what we're describing

```
describe('an unstarted auto', () => {  
  let auto  
  beforeEach(() => { auto = new Auto() })  
  
  it('idles engine when started', () => {  
    auto.depressBrake()  
  
    auto.pressStartButton()  
  
    expect(auto.RPM()).  
      to.be.within(950, 1100)  
  })  
})
```



the generalized behavior it supports

an example of that behavior

***Seek to first understand through the tests.***

# BDD Structuring

```
describe('given some context', () => {
  describe('when some event occurs', () => {
    it('then can be verified', () => {
      }
    }
  }
})
```

# BDD Structuring Example

```
describe('given a checked-out material', () => {
  let dueDate

  beforeEach(() => {
    dueDate = borrow(AgileJava, PatronId)
  })

  describe('when returned late', () => {
    let daysLate
    beforeEach(() => {
      daysLate = returnMaterial(AgileJava)
    })

    it('is marked as available', () => {})
    it('generates a late fine', () => {})
    it('notifies patrons with hold', () => {})
  })
})
```

# Iterative Naming

---

- Re-consider name continually

```
it('does something', ...)
```

```
it('verifies engine start', ...)
```

```
it('verifies engine start idle speed', ...)
```

```
it('has low idle on engine start', ...)
```

- Review often and holistically!

# Sustainable Tests

---

- Single behavior tests
- AAA
- Correlate result with context
- Test abstraction

# Exercise: Test Smells

Find and fix test smells.  
Paraphrase cleaned tests  
to your pair.



Further instructions: `./src/pos/routes/checkout.test.js`

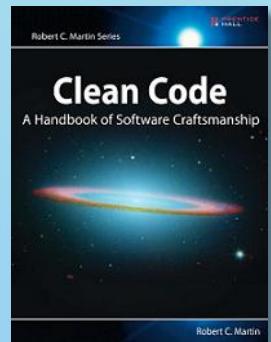
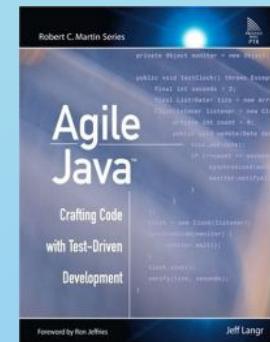
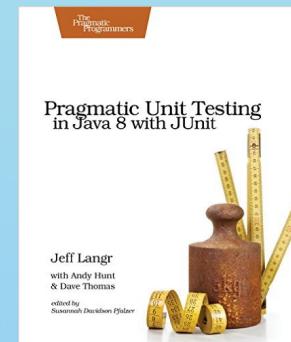
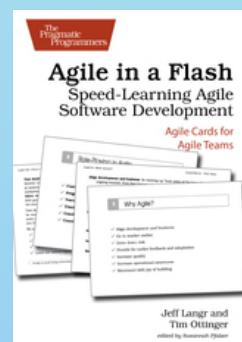
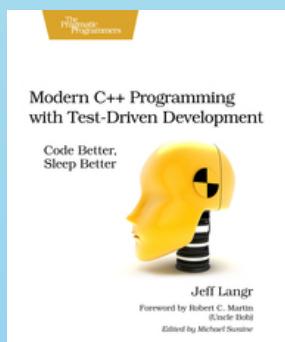
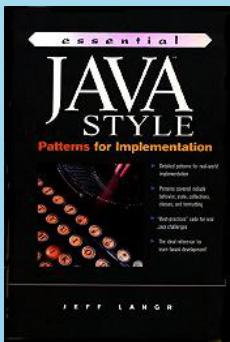
# Continual Design



iICON  
AGILITY SERVICES

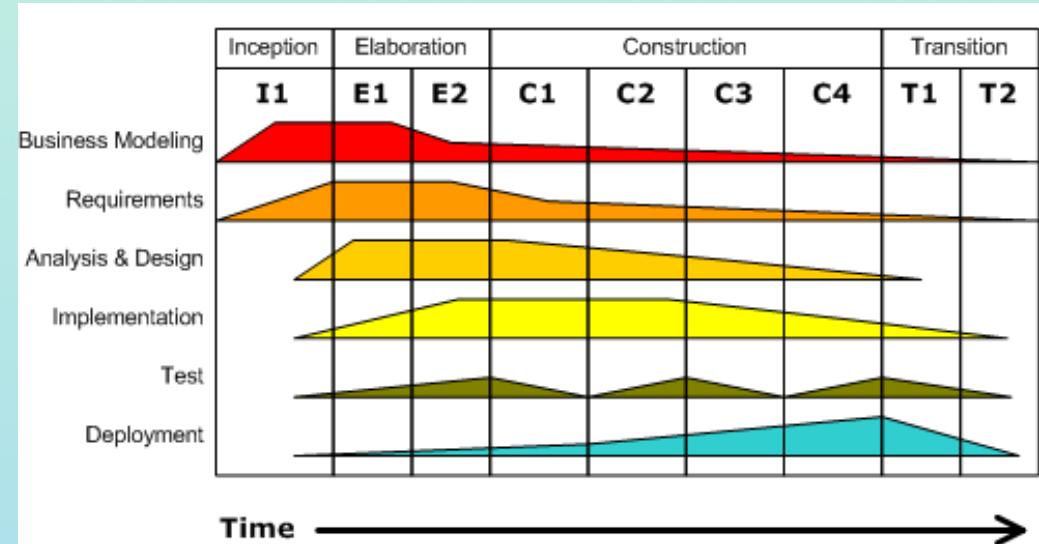
Langr  
Software Solutions

SOFTWARE TRAINING/CONSULTING



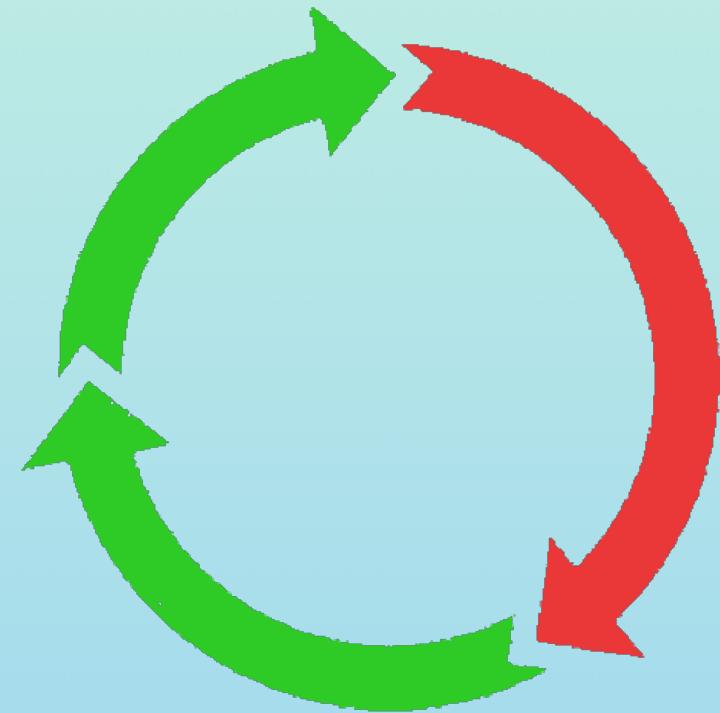
# Software Development

- Activities:
  - Analysis, design, coding, testing, review, documentation, planning, deployment, ...
- Agile:
  - *All activities all the time*



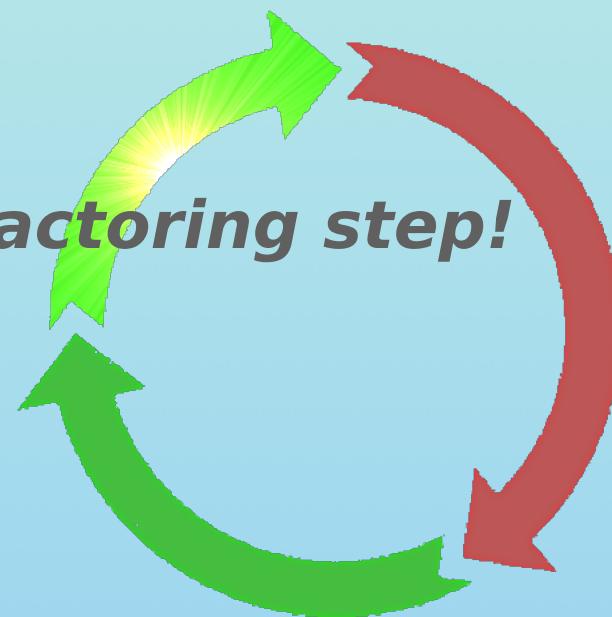
# TDD: Continual ...

- Testing
- Coding
- Design
- Documentation

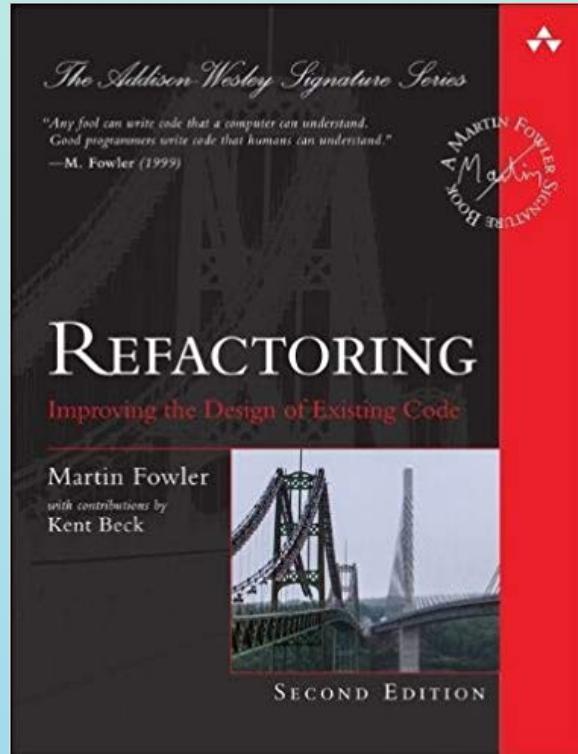


# Continual Design

- Always retain an optimal design
- Entropy is unavoidable

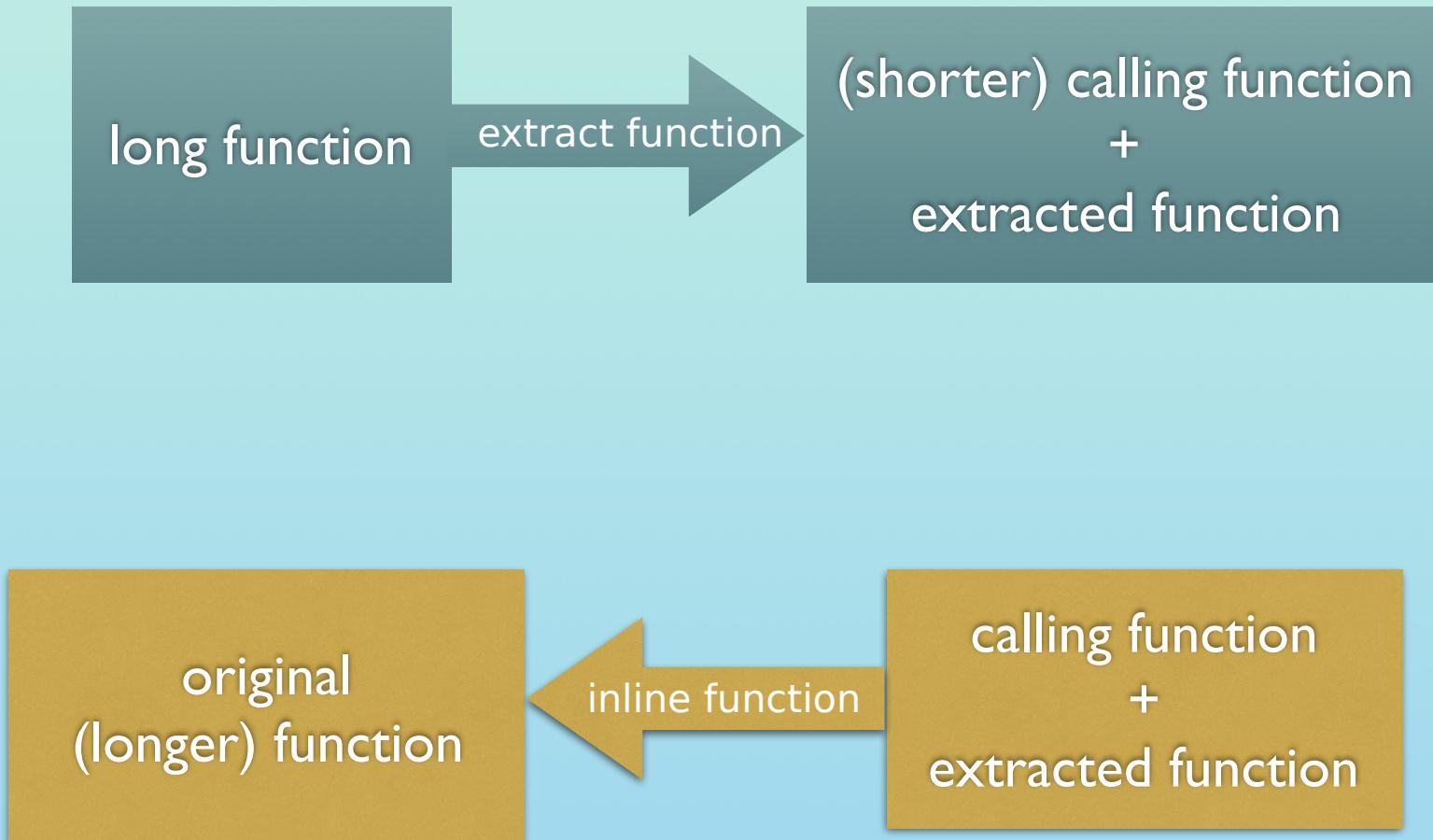


# Refactoring



Same "*externally recognized*" behavior

# For (*almost*) Every Transform...



# Code Smells

- Comments
- Duplicate Code
- Feature Envy
- Large Class
- Long Method
- Long Parameter List
- Primitive Obsession
- Shotgun Surgery
- Speculative Generality
- Switch Statements
- . . . and many more



"a complete code smells reference:"  
<https://github.com/lee-dohm/code-smell>

# The Nature of the Smell

Aroma?



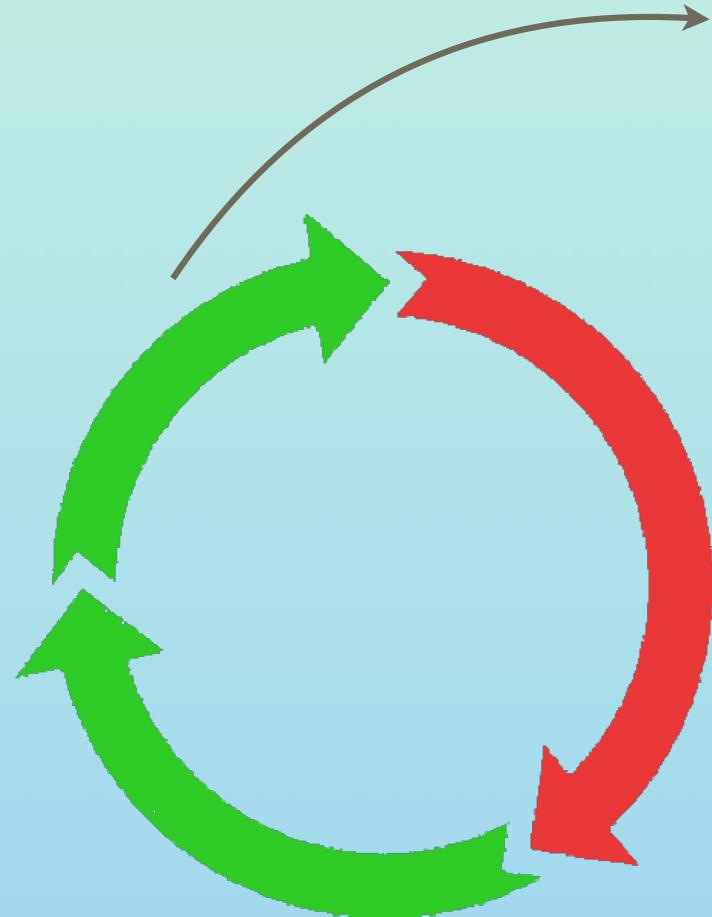
Stench?



"beagle smelling roses," courtesy Grangernite, <https://www.flickr.com/photos/lidoty/2573672102>  
<https://creativecommons.org/licenses/by/2.0/>

adapted from "Yum, that smells good!", courtesy Tankboy,  
<https://www.flickr.com/photos/tankboy/284958387>; <https://creativecommons.org/licenses/by-sa/2.0/>

# Refactoring Opportunities



small cleanup  
verify  
small cleanup  
verify  
...

**One thing at a time!**

Break things once in a while.

# Refactoring: Extract Function

```
const uglyFunction = () => {
  // stuff a
  ...
  // stuff b
  ...
  // some smaller behavior
  doSomething()
  doSomethingElse()
  // ...
  // stuff c
  // ...
}
```



```
const lessUglyFunction = () => {
  // stuff a  ...
  // stuff b
  ...
  someSmallerBehavior()
  // ...
  // stuff c
  // ...
}

const someSmallerBehavior = () => {
  doSomething()
  doSomethingElse()
}
```

## Why?

# Single-Line Extract?

```
catch(err) {  
  const { sev, system, module, msg } = err  
  const errMsg = `${new Date(): ${system}-${module} ${sev} ${trunc(msg, 80)}`  
  log(errMsg)  
  throw new Error(errMsg, err)  
}
```



```
catch(err) {  
  const { sev, system, module, msg } = err  
  const errMsg = formatErrorMessage(sev, system, module, msg)  
  log(errMsg)  
  throw new Error(errMsg, err)  
}  
  
const formatErrorMessage = (sev, system, module, msg) =>  
  `${new Date(): ${system}-${module} ${sev} ${trunc(msg, 80)}`
```

# Exercise: Extract Function



- In ./src/pos/routes/checkout.js
  - Do a single-line function extract on postCheckoutTotal
  - Otherwise focus on renaming and eliminating lies

# Refactoring-Inhibiting Temp

```
checkout.items.forEach(item => {
  let price = item.price;
  const isExempt = item.exempt;
  if (!isExempt && discount > 0) {
    const discountAmount = discount * price;
    const discountedPrice = price * (1.0 - discount);

    // add into total
    totalOfDiscountedItems += discountedPrice;

    let text = item.description;
    // format percent
    const amount = parseFloat(Math.round(price * 100) / 100).toFixed(2)
    const amountWidth = amount.length;

    let textWidth = LineWidth - amountWidth;
    messages.push(pad(text, textWidth) + amount);

    total += discountedPrice;

    // discount line
    const discountFormatted = '-' + parseFloat(Math.round(discountAmount * 100) / 100).toFixed(2)
    textWidth = LineWidth - discountFormatted.length;
    text = ` ${discount * 100}% mbr disc`;
    messages.push(` ${pad(text, textWidth)} ${discountFormatted}`);

    totalSaved += discountAmount;
}
```

# Refactoring: Replace Temp With Query

```
const discountAmount = discount * price;  
// . . .  
  
// discount line  
const discountFormatted = '-' + parseFloat(Math.round(discountAmount * 100) / 100).toFixed(2)  
textWidth = LineWidth - discountFormatted.length;  
text = ` ${discount * 100}% mbr disc`;  
messages.push(`${pad(text, textWidth)}${discountFormatted}`);  
  
totalSaved += discountAmount;
```



```
// discount line  
const discountFormatted =  
  '-' + parseFloat(Math.round(discountAmount(discount, price) * 100) / 100).toFixed(2)  
textWidth = LineWidth - discountFormatted.length;  
text = ` ${discount * 100}% mbr disc`;  
messages.push(`${pad(text, textWidth)}${discountFormatted}`);  
  
totalSaved += discountAmount(discount, price);  
// ...  
});  
  
const discountAmount = (discount, price) => discount * price;  
// . . . . .
```

# Replace Temp With Query

---

Steps:

- Make temp const
- Extract Function on the rhs
- Replace temp references with the query
- Remove temp

Why?

Concerns?

Can we go the other way?

# Exercise: Replace Temp with Query



- In ./src/pos/routes/checkout.js
  - Apply Replace Temp with Query in postCheckoutTotal as appropriate
  - Extract as many additional functions as you can

# Code Smell: Feature Envy

```
class WrongPlace {
    client() {
        const x = enviousFunction()
        ...
    }

    enviousFunction() {
        const there = new GreenerGrass()
        const result = there.doStuff(this.someValue)
        there.doMoreStuff(result)
        return there.answer()
    }
}
```

*Why is this a problem?*

# Refactoring: Move Function

```
class WrongPlace {
    client() {
        const x = new GreenerGrass().homebodyMethod(this.someValue)
        ...
    }
}

class GreenerGrass {
    homebodyMethod(someValue) {
        var result = doStuff(someValue)
        doMoreStuff(result)
        return answer()
    }
    ...
}
```

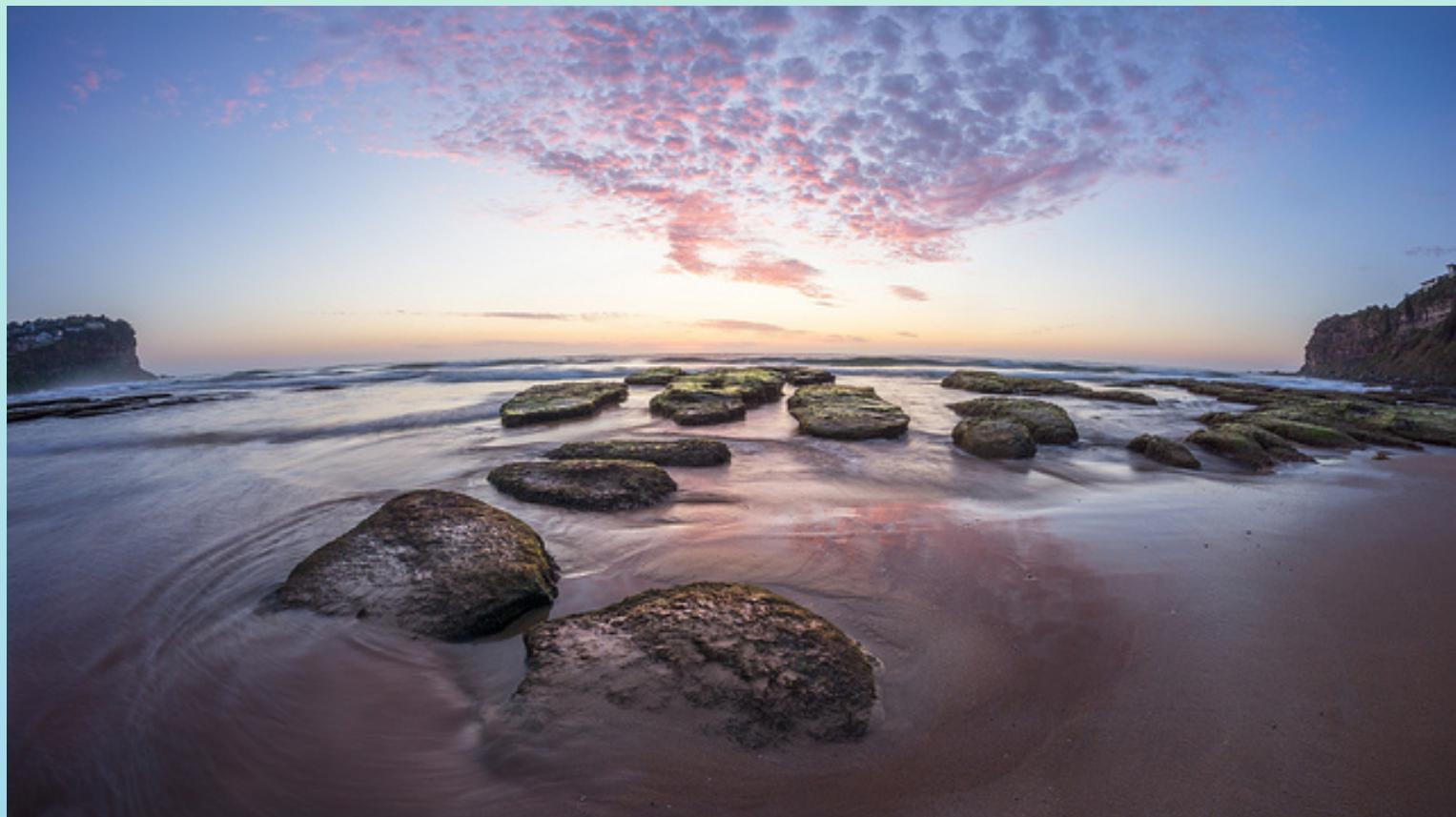
*What must you also do when you move a function?*

# Exercise: Move Function

- In ./src/pos/routes/checkout.js
- Move at least 2 envious functions to better (or new!) homes
- Ensure you've added documentation (i.e tests)!



# Macro Refactorings



Might need to backtrack!

# Smell: Switch Statements

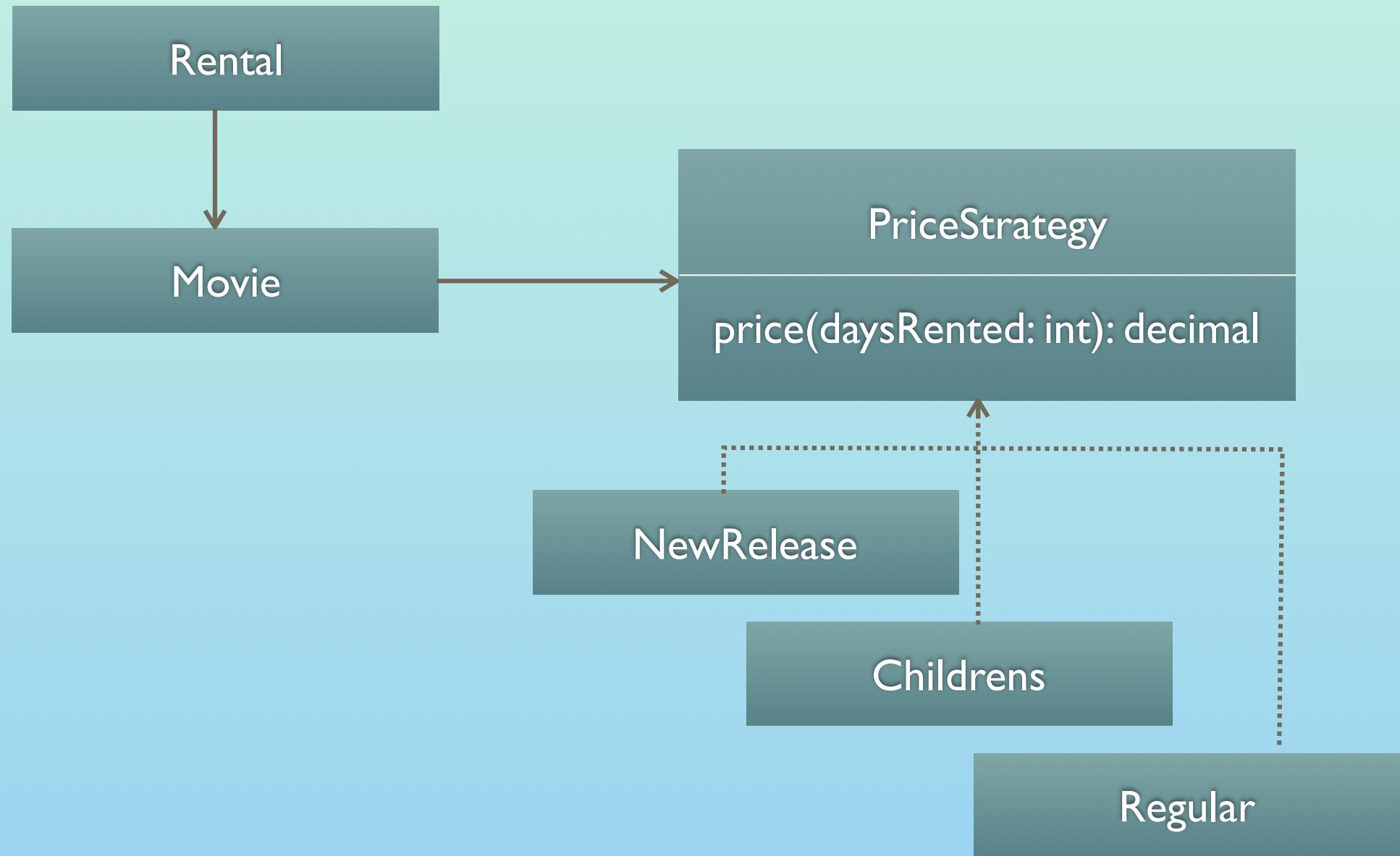
```
switch (movie.code) {  
    case 'regular':  
        thisAmount = 2  
        if (r.days > 2) {  
            thisAmount += (r.days - 2) * 1.5  
        }  
        break  
    case 'new':  
        thisAmount = r.days * 3  
        break  
    case 'childrens':  
        thisAmount = 1.5  
        if (r.days > 3) {  
            thisAmount += (r.days - 3) * 1.5  
        }  
        break  
}
```

# Some of the Transforms (OO)

---

- Self-encapsulate field
- Replace Conditional With Polymorphism
- Extract Function, Move Function
- Replace Type Code with Subclasses
  - Push Down Method / Field

# A Refactored Result (oo)



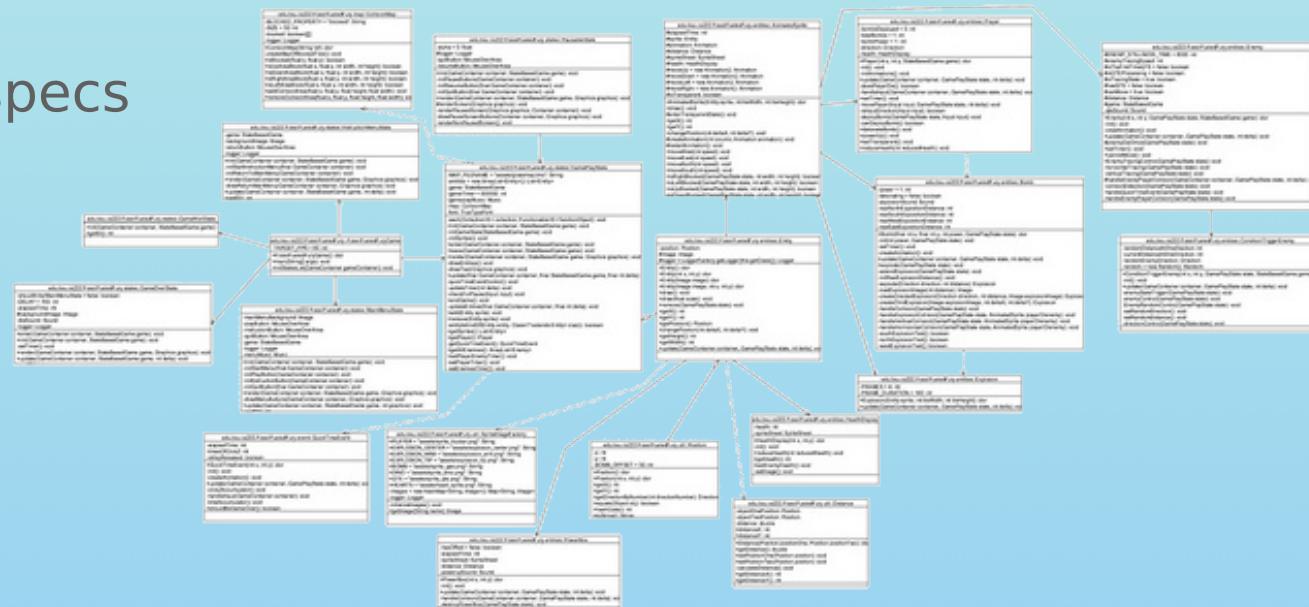
# Planning: Up-Front Design

## ■ Not just once!

- Planning: project, release, iteration, day, task, test
- Any estimation

## ■ Minimal: sketches & conversations

- Not detailed specs



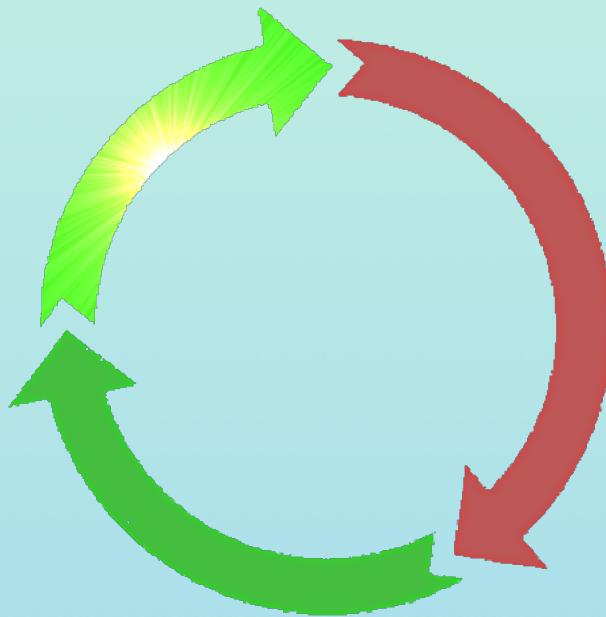
# Premature Generalization

- Need might never materialize
- Details might change
- Interim complexity \$



# Refactoring Guidelines

Single-goal



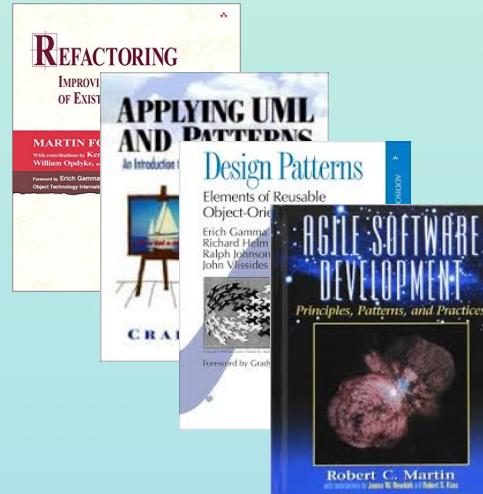
Never skip!

Run *all* tests

It's your responsibility

# Design Drivers / Guidelines

- Code smells
- SOLID
- Design patterns
- GRASP
- Simple design
- DRY
- ...

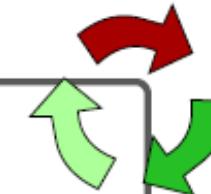


**It's all good!**

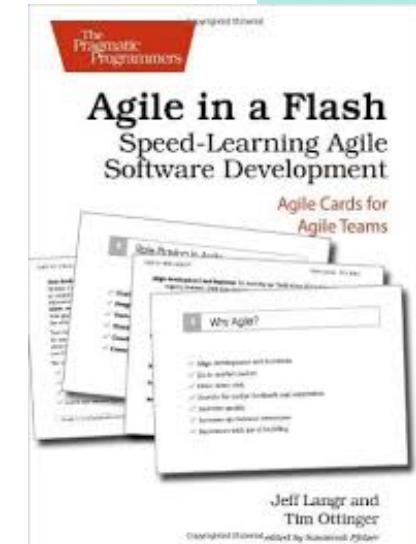
*Does everything point to the same place?*

41

## Build Superior Systems with Simple Design



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



Kent Beck's (ordered) rules for emergent design

# Exercise: Simple Design Rules

- In ./src/pos/routes/checkout.js
  - Stamp out duplication!
  - Strive for rapid readability
  - Have you gone too far?



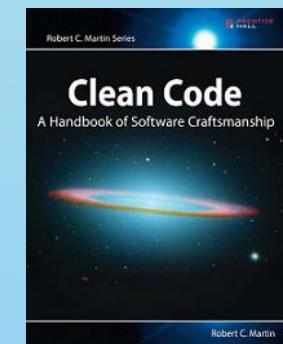
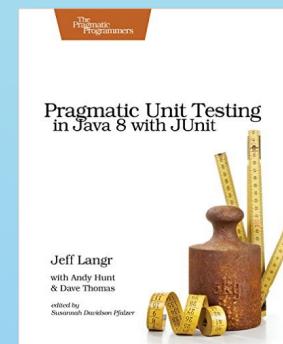
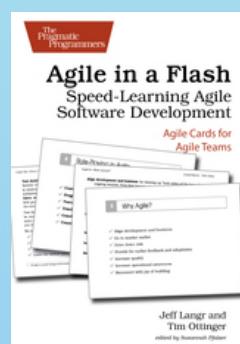
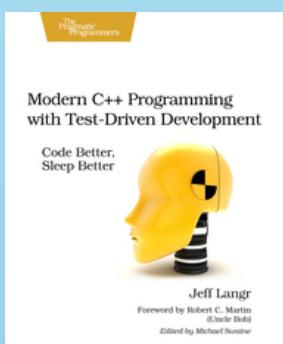
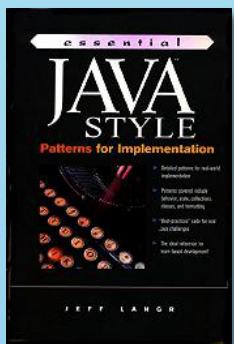
# Test Doubles



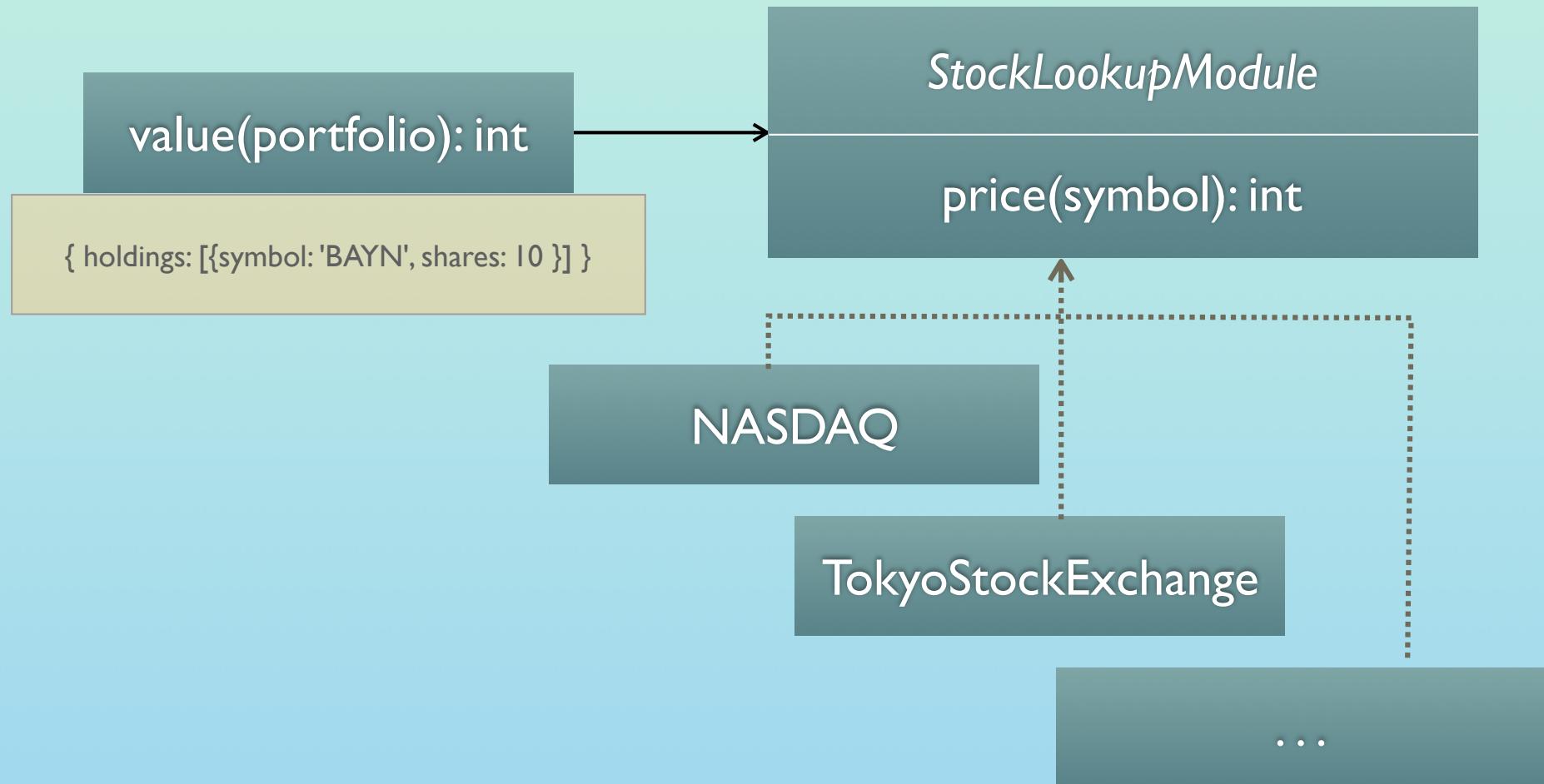
iICON  
AGILITY SERVICES

Langr  
*Software Solutions*

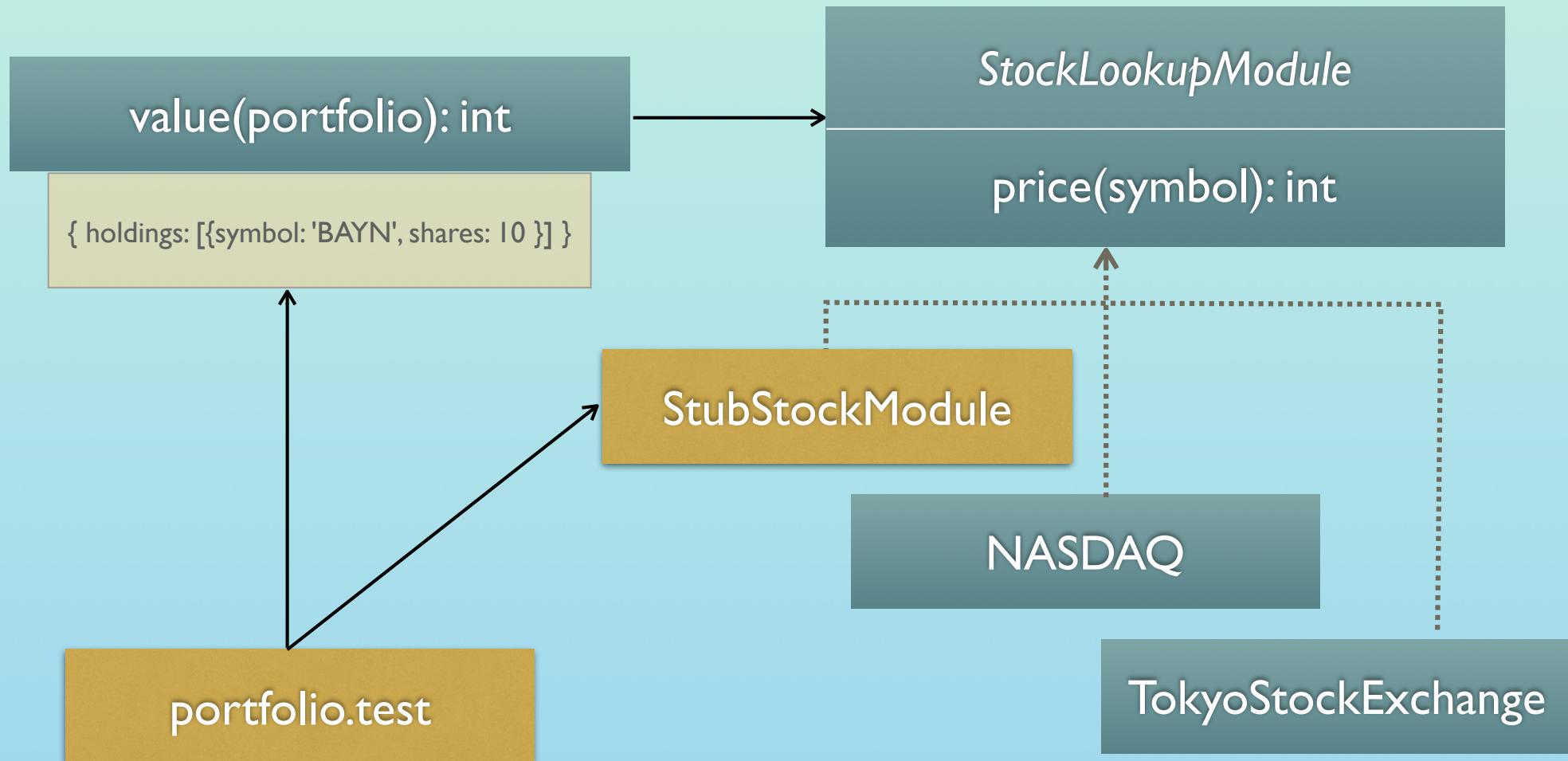
SOFTWARE TRAINING/CONSULTING



# Testing Challenge: Portfolio



# Using a Test Double



# Portfolio Code

Test-Driving from portfolio.test.js:

```
import * as StockLookupService from './stock-lookup-service'

describe('portfolio value with stub', () => {
  it('is symbol price for single-share purchase', () => {
    portfolio = purchase(portfolio, 'IBM', 1)
    expect(value(portfolio)).to.equal(42) // how to make this work?
  })
})
```

Some notion of an implementation in portfolio.js:

```
import { symbolLookup } from './stock-lookup-service'
export const value =
  portfolio => symbolLookup(Object.keys(portfolio.holdings)[0])
```

The troublesome dependency in stock-lookup-service.js:

```
export let symbolLookup = _symbol => { /* prod code */ }
```

# ES6: Function Override

(using `import * as`)

## portfolio.test.js

```
import * as StockLookupService from './stock-lookup-service'

describe('portfolio value with stub', () => {
  it('is symbol price for single-share purchase', () => {
    StockLookupService.symbolLookupStub(_ => 42)
    portfolio = purchase(portfolio, 'IBM', 1)
    expect(value(portfolio)).to.equal(42)
  })
})
```

## stock-lookup-service.js

```
export let symbolLookup = _symbol => { /* prod code */ }
export const symbolLookupStub = stub => symbolLookup = stub
```

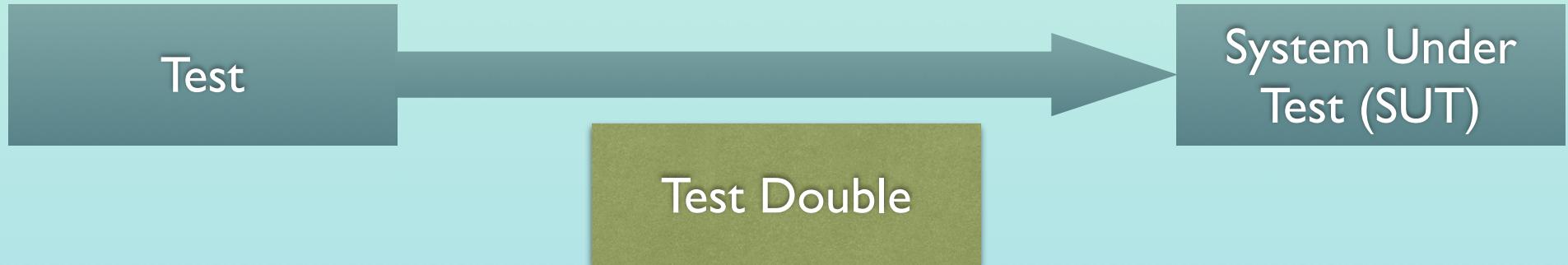
## portfolio.js

```
import { symbolLookup } from './stock-lookup-service'
export const value =
  portfolio => symbolLookup(Object.keys(portfolio.holdings)[0])
```

# **Exercise: Function Override**

using `import * as`

# Test Double Injection Techniques



## Functional

Function argument  
import \* Function Override  
exports injection

## OO

Constructor / setter  
Prototype  
Method override

# Prototype Injection

```
PortfolioObj.prototype.value = function() {
  if (this.size() === 0) return 0
  return this.lookupPrice('BAYN')
    .then(function({_symbol, price}) {
      return price
    })
}

PortfolioObj.prototype.lookupPrice = function() {
  return ax.get(`http://localhost:3001/api/quote?symbol=BAYN`)
    .then(function(response) {
      return { symbol: response.symbol, price: response.price }
    })
}

let realLookupPrice

beforeEach(() => {
  realLookupPrice = PortfolioObj.prototype.lookupPrice
  PortfolioObj.prototype.lookupPrice =
    function() {
      return Promise.resolve(
        { symbol: Monsanto, price: MonsantoValue })
    }
})

afterEach(() => {
  PortfolioObj.prototype.lookupPrice = realLookupPrice
})

it('...', async () => {
  portfolio.purchase(Monsanto, 1)
  const result = await portfolio.value()
  expect(result).to.equal(MonsantoValue)
})
```

# Exercise: Prototype Injection

---



Flesh out the portfolio value function, using TDD, starting with `./src/misc/portfolio-obj.test.js` and `./src/misc/portfolio/portfolio-obj.js`

*Start simple! Add in small increments!*

You should end up with at least three tests, perhaps four.

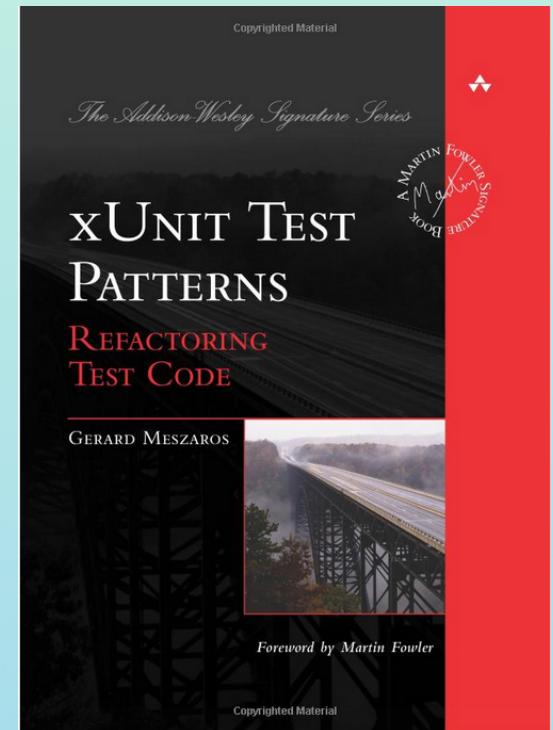
# Test Double Terms

**Stub:** dumb emulation

**Spy:** captures values to verify

**Mock:** self-verifies

**Fake:** whole collaborator emulation



# Mock Tools



*History suggests more & changes coming!*

# Sinon: Many tricks, many sleeves



# Sinon: A Simple Stub

<https://sinonjs.org>

```
const temperatureServerStub = sinon.stub()  
temperatureServerStub.withArgs('Miami').returns(96)  
temperatureServerStub.withArgs('St. Louis').returns(88)  
  
const result = averageTemp(['Miami', 'St. Louis'], temperatureServerStub)  
  
expect(result).to.equal(92)
```

```
export const currentTemperature = _city => {  
  throw Error('server down')  
}  
  
export const averageTemp = (cities, currentTemperature) => {  
  return cities  
    .map(city => currentTemperature(city))  
    .reduce((sum, temp) => sum + temp, 0) / cities.length  
}
```

# What's the Problem?

```
describe('avg temp', () => {
  let stub

  beforeEach(() => {
    stub = sinon.stub()
  })
  afterEach(() => sinon.restore())

  it('is same as temp for 1 city', () => {
    sinon.replace(TemperatureService.prototype, 'currentTemperature', stub)
    stub.withArgs('Phoenix').returns(115)

    const result = avgTemperature(['Phoenix'])

    expect(result).to.equal(115)
  })

  it('throws when currentTemperature not stubbed', () => {
    expect(() => {
      new TemperatureService().currentTemperature('Phoenix')
    }).to.throw()
  })
})
```

# Exercise: Sinon Stubs

---



Test-drive the `value()` function into your `Portfolio` class, using Sinon for method stubs.

# require() Dependency Challenge

## portfolio.js

```
var symbolLookup = require('./stock-lookup-service').symbolLookup

// ... other funcs ...

var value = function(portfolio) {
  var sumValues = function(totalValue, symbol) {
    return totalValue + symbolLookup(symbol) * sharesOf(portfolio, symbol)
  }
  return empty(portfolio) ? 0 :
    Object.keys(portfolio.holdings).reduce(sumValues, 0)
}

module.exports = { /* ... */, value: value }
```

## stock-lookup-service.js

```
module.exports = {
  symbolLookup: function(symbol) { /* ... */ }
}
```

## Exports Injection

### portfolio.js

```
module.exports = function(symbolLookup) {
  return {
    // ... other funcs ...
    value: function(portfolio) {
      var sumValues = function(totalValue, symbol) {
        return totalValue +
          symbolLookup(symbol) * this.sharesOf(portfolio, symbol)
      }.bind(this)
      return this.empty(portfolio) ? 0 :
        Object.keys(portfolio.holdings).reduce(sumValues, 0)
    } }
}
```

### portfolio.test.js

```
var stubSymbolLookup = sinon.stub()
var Portfolio = require('./portfolio')(stubSymbolLookup)

describe('portfolio value', function() {
  it('is symbol price for single-share purchase', function() {
    var portfolio = Portfolio.createPortfolio()
    stubSymbolLookup.withArgs('IBM').returns(IBMPrice)
    portfolio = Portfolio.purchase(portfolio, 'IBM', 1)
    expect(Portfolio.value(portfolio)).to.equal(IBMPrice)
  })
})
```

# Pushing Out the Dependency

Clients of portfolio must inject the production implementation

```
var Portfolio = require('./portfolio')(require('./stock-lookup-service'))  
  
describe('a portfolio in production', function() {  
  var portfolio = Portfolio.createPortfolio()  
  portfolio = Portfolio.purchase(portfolio, 'BAYN', 10)  
  var value = Portfolio.value(portfolio)  
  expect(value).to.be.greaterThan(0)  
})
```

# Clean Up With Default Args

If you are on ES2015+:

```
var defaultService = require('./stock-lookup-service')

module.exports = function(Service=defaultService) {
  return {
    // ...
    value: function(portfolio) {
      var sumValues = function(totalValue, symbol) {
        return totalValue +
          Service.symbolLookup(symbol) * this.sharesOf(portfolio, symbol)
      }.bind(this)
      return this.empty(portfolio) ? 0 :
        Object.keys(portfolio.holdings).reduce(sumValues, 0)
    }
}
```

# Sinon: Stubbing Exports Object

stock-lookup-service.js

```
module.exports = {
  symbolLookup: function(symbol) { /* ... */ }
}
```

the test:

```
var stockService = require('./stock-lookup-service')
...
var stub = sinon.stub(stockService, 'symbolLookup')
```

# Exceptions

```
it('answers 0 when price retrieval throws', async () => {
  sinon.stub(portfolio, 'retrievePrice').throws();
  portfolio.purchase(Monsanto, 2);

  const result = await portfolio.value();

  expect(result).to.equal(0);
});
```

## Variants:

```
stub.throws('exception name property');

stub.throws(exceptionObj);
```

# Verifying a "Tell"

How to test-drive?

```
purchase(symbol, shares) {  
    this.updateShares(symbol, shares);  
    this.auditor('purchase');  
}
```

Use a Sinon spy:

```
const auditSpy = sinon.spy();  
portfolio.useAuditor(auditSpy);  
  
portfolio.purchase('BAYN', 10);  
  
expect(auditSpy.called).to.be.true;
```

# Sinon Spy: Verifying Arguments

```
expect(auditSpy.calledWith(  
  'purchase 10 shares of BAYN')).to.be.true;
```

```
purchase(symbol, shares) {  
  this.updateShares(symbol, shares);  
  this.auditor(`purchase ${shares} shares of ${symbol}`);  
}
```

# Sinon JS Assertions for Chai

```
import chai, { expect } from 'chai';
import sinonChai from 'sinon-chai';
chai.use(sinonChai);

// ...

expect(auditSpy).to.have.been.calledWith(
  'purchase 10 shares of BAYN');
```

AssertionError: expected auditor to have been called with  
arguments purchase 10 shares of BAYN

**Purchase 10 shares of BAYN purchase 10 shares of BAYN**

<http://www.chaijs.com/plugins/sinon-chai/>

# Argument Matchers

What's the challenge?

```
this.auditor(  
  `purchase ${shares} shares of ${symbol}`,  
  new Date());
```

```
expect(auditSpy).to.have.been.calledWithMatch(  
  'purchase 10 shares of BAYN', sinon.match.date);
```

<https://sinonjs.org/releases/v6.1.5/matchers/>

# Spy Exercise

Verify that the portfolio sends an appropriate audit message on all buy & sell transactions.

The audit function should take on a description, the current timestamp, and the number of shares involved.



# Schools of Mock

Classic  
("Detroit")

algorithmic approach

verification of state

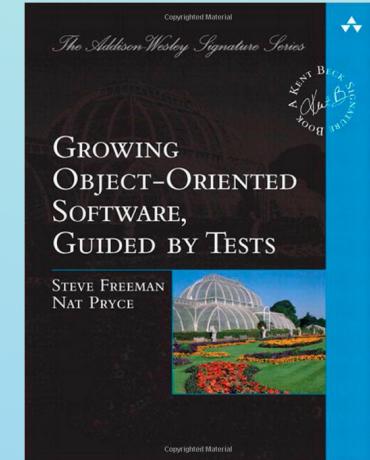
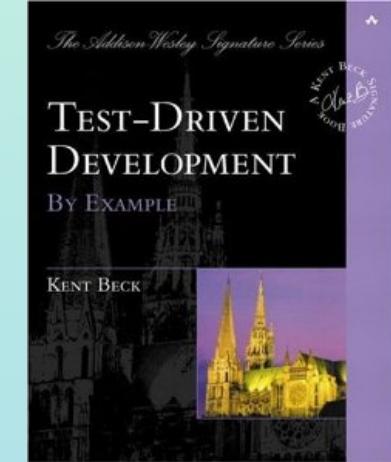
tests drive code from specific to general

London

roles, responsibilities, and interactions

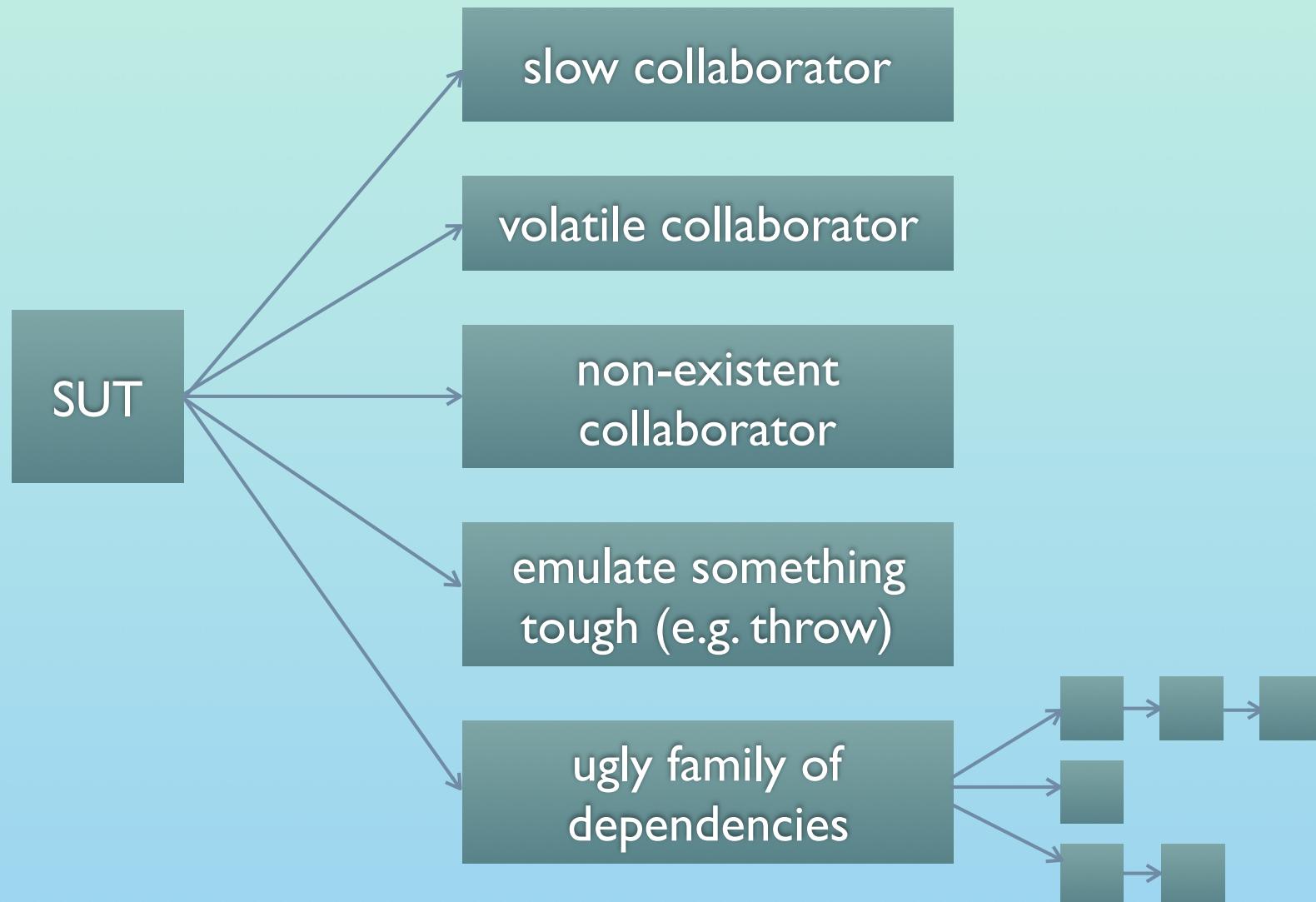
"Fake it until you make it"

drive incrementally from the outside in



**Read Martin Fowler's "Mocks Aren't Stubs"**

# Classic School: When to Mock



# When *Not* to Mock?

---

Can you inject the data or a generator instead?

# The Pragmatic Mocker



- Writes integration tests where mocks are used
- Watches coupling between tests & implementation
- Seeks to mock only direct collaborators
- Avoids fakes (and if necessary, test-drives 'em!)
- Isolates, minimizes the use of test doubles
  - But prefers testability

# Wrap-Up Exercise



- Support voids on checkout



Work outside in:

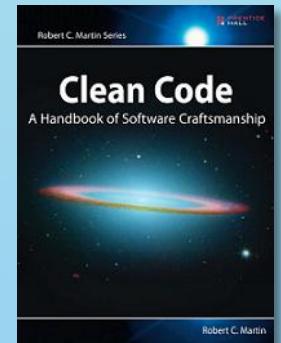
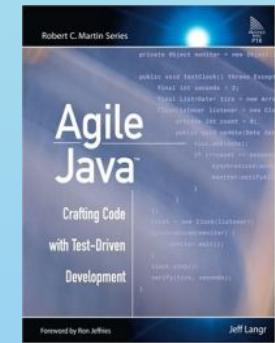
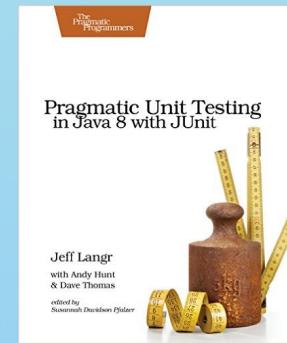
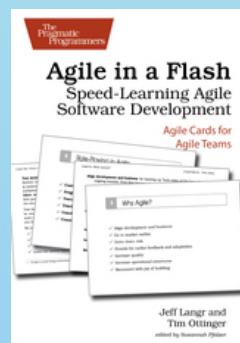
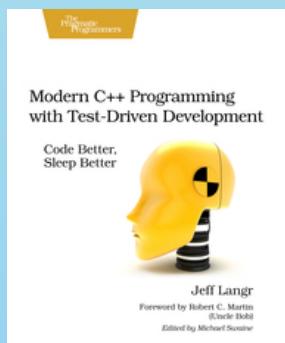
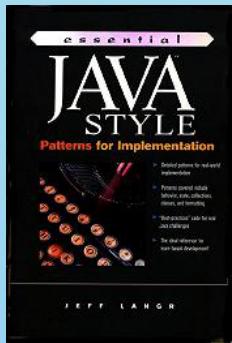
- Stub collaborators until ready to build them out

# Growing & Sustaining TDD

iICON  
AGILITY SERVICES

Langr  
Software Solutions

SOFTWARE TRAINING/CONSULTING



# Next Week



- Create team policies/standards around TDD
- Ensure your build pipeline supports it
- Reviews must include the tests
- Pairing / mobbing helps quite a bit!
- Focus on problem areas first
- Adopt on smaller scale if necessary

# Growing TDD

- Education / sharing sessions
- Challenges / contests / randoris
- Coach / champion
- "Stop the line" mentality
- Changing metrics for education
- Pair & paraphrase
  - *Three+* sets of eyes!
- Mobbing



# Sustaining TDD

Everything under "Growing TDD," plus:

- TDD-specific retrospectives
- Daily stand-down
- Regular reminders as to "why?"
- Revisit standards regularly
- Keep the tests fast
  - Segregate (and fix) slow tests
- Always use tests as an entry point
- Refactor, refactor, refactor
  - Tests too!



# Mastering TDD

TDD is a skill.



*Practice, practice, practice.*

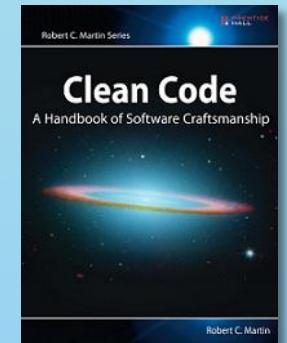
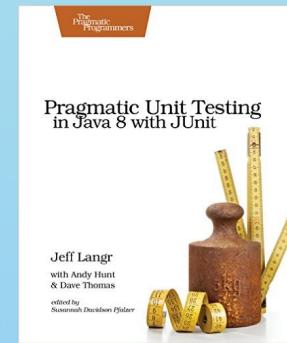
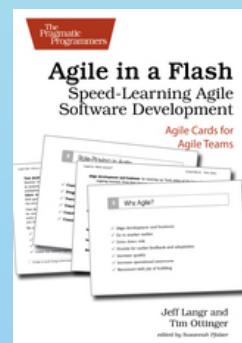
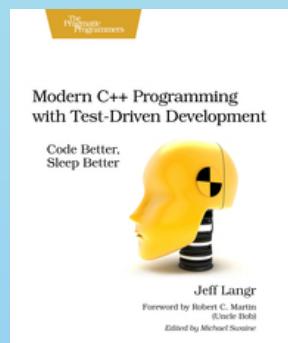
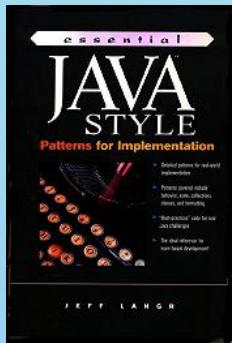
# Miscellaneous Topics



iICON  
AGILITY SERVICES

Langr  
*Software Solutions*

SOFTWARE TRAINING/CONSULTING



# Testing With Promises: ES6

```
const funcReturningPromise = () =>  
  new Promise((resolve, _reject) => setTimeout(resolve, 500, 42))
```

Nope:

```
describe('testing promises', () => { // doesn't work!  
  const x = funcReturningPromise()  
  expect(x).to.equal(41)  
})
```

ES6: await

```
describe('testing promises', async () => {  
  const x = await funcReturningPromise()  
  expect(x).to.equal(41)  
})
```

# Testing With Promises: Old School

```
it('is worth share price for single share purchase', done => {
  portfolio.retrievePrice = () => Promise.resolve({ symbol: IBM, price: 100 });
  portfolio.purchase(IBM, 1);

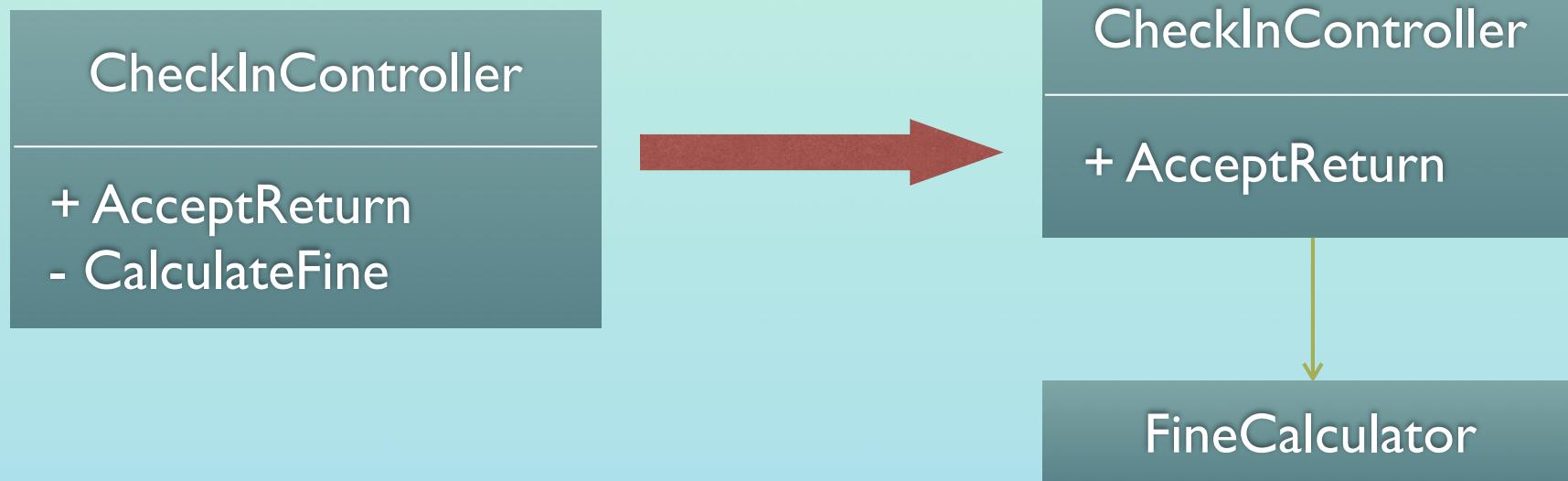
  portfolio.value()
    .then(result => {
      expect(result).to.equal(IBM);
      done();
    });
});
```

*If that doesn't work:*

```
it('is worth share price for single share purchase', done => {
  portfolio.retrievePrice = () => Promise.resolve({ symbol: IBM, price: 100 });
  portfolio.purchase(IBM, 1);

  portfolio.value()
    .then(result => { expect(result).to.equal(100); })
    .then(done, done);
});
```

# Testing Non-Public Behavior?



Or simply relax access.

# Rules of Ten



Green & clean in < 10 (minutes)

Delete and repeat if you're late!  
... taking smaller steps this time.

# Rules of Ten



Debate stops at 10 (minutes)

"Show me."

# Rules of Ten



3

All unit tests run in < 10 (seconds).

# Fixing a Slow Test Run



Warn on tests <  $n$  ms

Fail on tests >  $n$  ms

*Running a subset of tests increases risk.*

# Uncle Bob's TPP

Transformations have a preferred ordering, which if maintained by ordering of tests, prevents "long outages" in TDD.

The  
Priority  
List

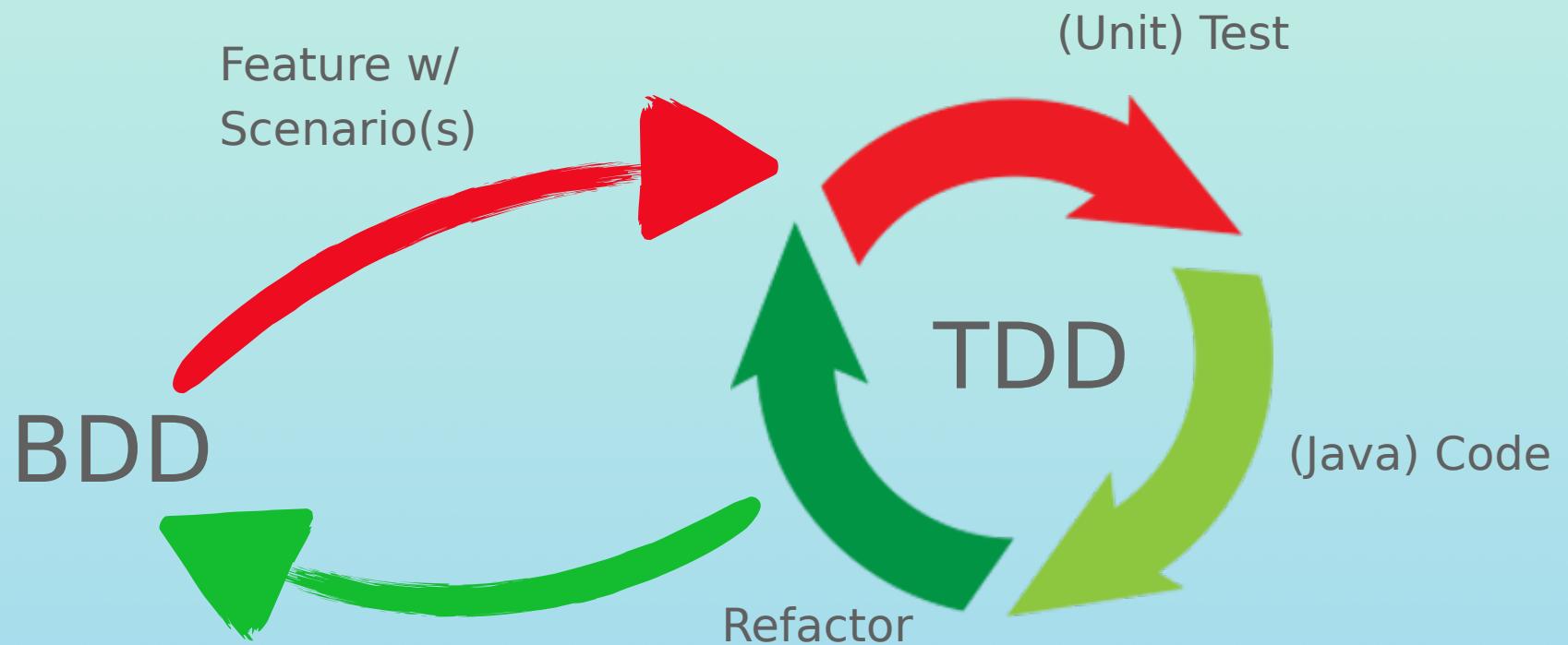
- ({}->**nil**) no code at all->code that employs nil
- (**nil->constant**)
- (**constant->constant+**) a simple constant to a more complex constant
- (**constant->scalar**) replacing a const. with a variable or an argument
- (**statement->statements**) adding more unconditional statements.
- (**unconditional->if**) 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

"As the tests get more specific, the code gets more generic."

<http://thecleanncoder.blogspot.com/2011/02/fib-t-p-premise.html>

<https://8thlight.com/blog/uncle-bob/2013/05/27/TheTransformationPriorityPremise.html>

# Behavior-Driven Development (BDD)



# Sinon: A Method Stub

```
const temperatureService = new TemperatureService()
const currentTemperatureStub = sinon.stub(temperatureService, 'currentTemperature')
currentTemperatureStub.withArgs('St. Louis').returns(88)
currentTemperatureStub.withArgs('Col Springs').returns(72)

const result = averageTemperature(['Col Springs', 'St. Louis'], temperatureService)

expect(result).to.equal(80)
```

```
class TemperatureService {
  currentTemperature(_city) {
    throw Error('server *really* down')
  }
}

const averageTemperature = (cities, temperatureService) => {
  return cities
    .map(city => temperatureService.currentTemperature(city))
    .reduce((sum, temp) => sum + temp, 0) / cities.length
};
```

# Sinon: Stubbing the Prototype Method

```
const stub = sinon.stub()  
sinon.replace(TemperatureService.prototype, 'currentTemperature', stub)  
stub.withArgs('Phoenix').returns(115)  
  
const result = avgTemperature(['Phoenix'])  
  
expect(result).to.equal(115)
```

```
const avgTemperature = cities => {  
  return cities  
    .map(city => new TemperatureService().currentTemperature(city))  
    .reduce((sum, temp) => sum + temp, 0) / cities.length  
}
```

# Sinon Mocks

*Intent:* "If you wouldn't add an assertion for some specific call don't mock it. Use a stub instead."

```
const mock = sinon.mock(portfolio);
mock.expects('lookupPrice').once()
  .returns(Promise.resolve({ symbol: Monsanto, price: MonsantoValue }));
portfolio.purchase(Monsanto, 1);
const result = await portfolio.valueViaLocalFunc();
expect(result).to.equal(MonsantoValue);
```

But:

```
// multiple 'withArgs' is not supported!
const mock = sinon.mock(portfolio);
mock.expects('lookupPrice')
  .withArgs(Monsanto)
  .returns(Promise.resolve({ symbol: Monsanto, price: MonsantoValue }));
  .withArgs(Ibm)
  .returns(Promise.resolve({ symbol: Ibm, price: 50 }));
```

# References / Reading List

---

"Chat" icon courtesy Gregor Cesnar, Noun Project (Creative Commons)

[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.

[Meszaros2007] Meszaros, Gerard. **xUnit Test Patterns: Refactoring Test Code**. Addison-Wesley, 2007.

# Thank you!



iICON  
AGILITY SERVICES



Langr  
Software Solutions

SOFTWARE TRAINING/CONSULTING

