

Test-Driven Development Foundations

Yes, you're probably in the right place.

Things you can or should do while waiting on class start:

- Install **mob** tool. See <https://github.com/remotemobxprogramming/mob> (see “how to install”)
- git clone <https://github.com/jlangr/tdd-javascript-jest-2020>
 - Do an **npm install**
 - Import as a project
 - Make sure you can run the tests (see the README.md, package.json)
- Either install Kahoot! on your phone, or navigate to kahoot.it



ICON
AGILITY SERVICES

**Test-Driven Development
Foundations v1.3**

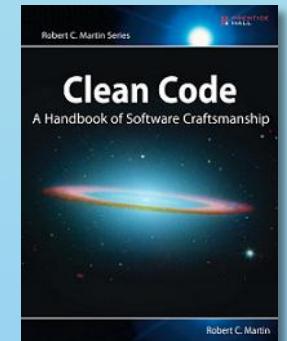
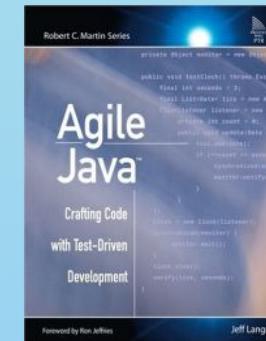
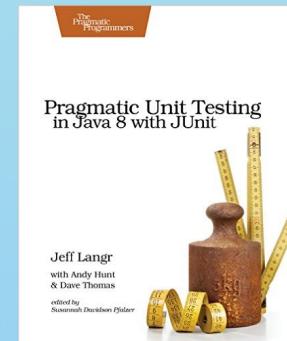
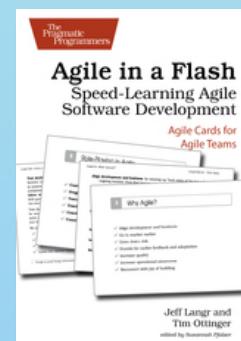
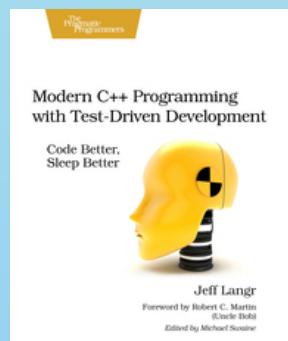
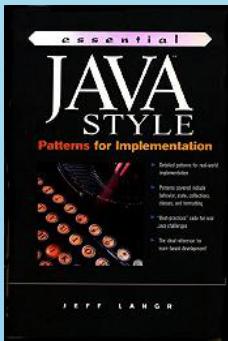
Jeff Langr

jeff@langrsoft.com
@JLangr



LANGR | SOFTWARE SOLUTIONS

<http://langrsoft.com>



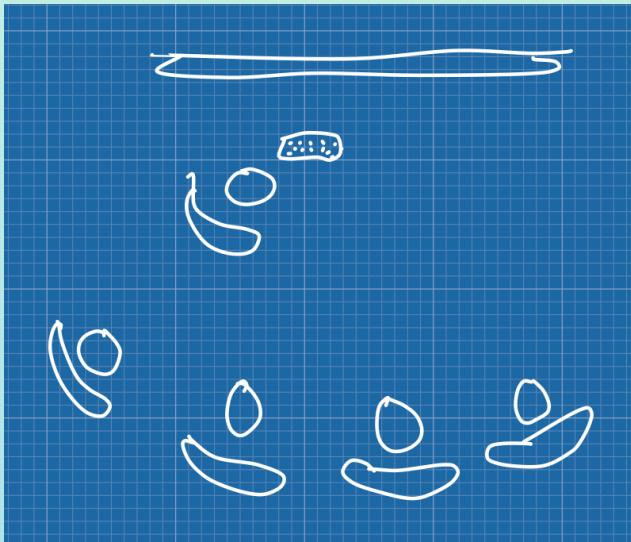


Exercise: Simple Outcomes



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

A Tale of Two Mobbing Rules



Once upon a time, there was a quaint notion of an “office,” a physical place where people could undistance and work together...

- Strong-style pairing
- Short, timed rotations

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()).toBeWithinRange(950, 1100)
  })
})
```

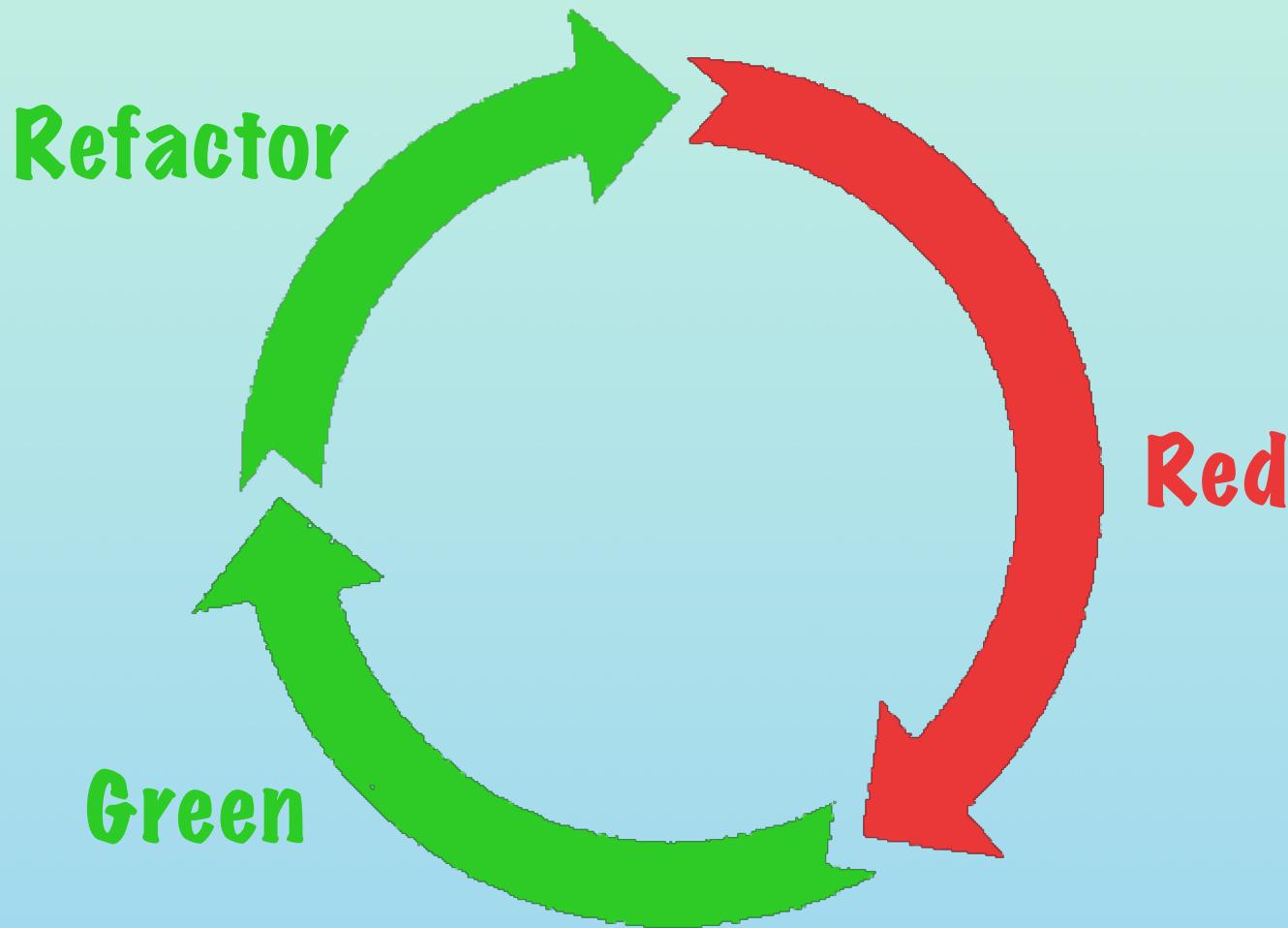
What's an Assertion Look Like?

```
expect(someCondition).toBe(true)
expect(idleSpeed).toBe(1000)
expect(alphabetizedName).toEqual('Schmoo, Kelly Loo')
expect(someArray).toEqual(['O', 'T', 'F', 'F', 'S', 'S'])
expect(alphabetizedName).toMatch(/^S.*,/)

expect(idleSpeed).not.toEqual(42)
expect(alphabetizedName).not.toEqual("Smelt"))
expect(someVariable).toBeUndefined
expect(someArray).toHaveLength(3)
expect(someArray).toContain(42)
expect(someArray).toContainEqual({ name: 'Amit' })
```

See <https://jest-bot.github.io/jest/docs/expect.html>

Test-Driven Development



An incremental design technique

Exercise: (Test)-Code-Refactor

TDD Paint by Numbers



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

`npm run test:watch`

Why???

A small, rectangular image showing a portion of a computer screen. The screen displays binary code in green text on a black background. The visible text includes '101001 01101', '00101 0010000', '000 01110100', '0 01110011 01', '01101110 0016', '1100010 011016', '01110 01110100', and '0011 01101000'.

```
101001 01101  
00101 0010000  
000 01110100  
0 01110011 01  
01101110 0016  
1100010 011016  
01110 01110100  
0011 01101000
```

Keep yer code clean
Document choices
Gate defects

Core TDD Themes



- Drive *behavior*
- Specification by example
- Small increments
- Stick to the cycle
 - Always see red
 - Always refactor

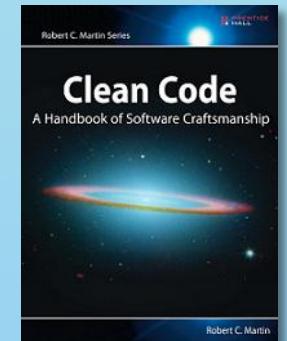
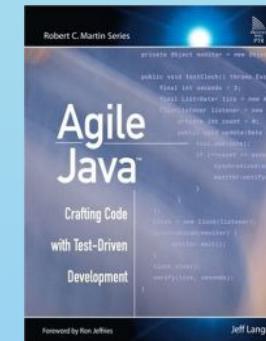
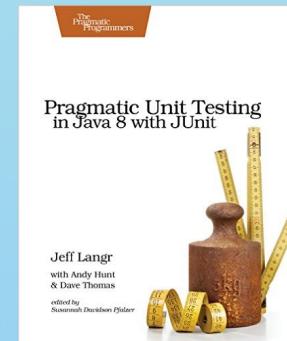
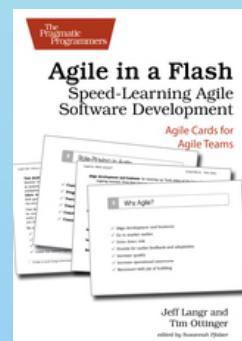
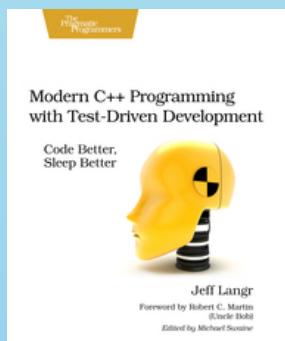
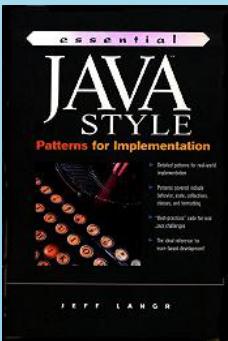
It's just code!



Stepping Back: Writing Tests



LANGR | SOFTWARE
SOLUTIONS



Recent File List

Manage a list of recently-opened files (max: 5) for use in a GUI application.



What tests do we need?

```
const recents = openFile(existingRecents, filepath)
```

ZOMBIES!

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



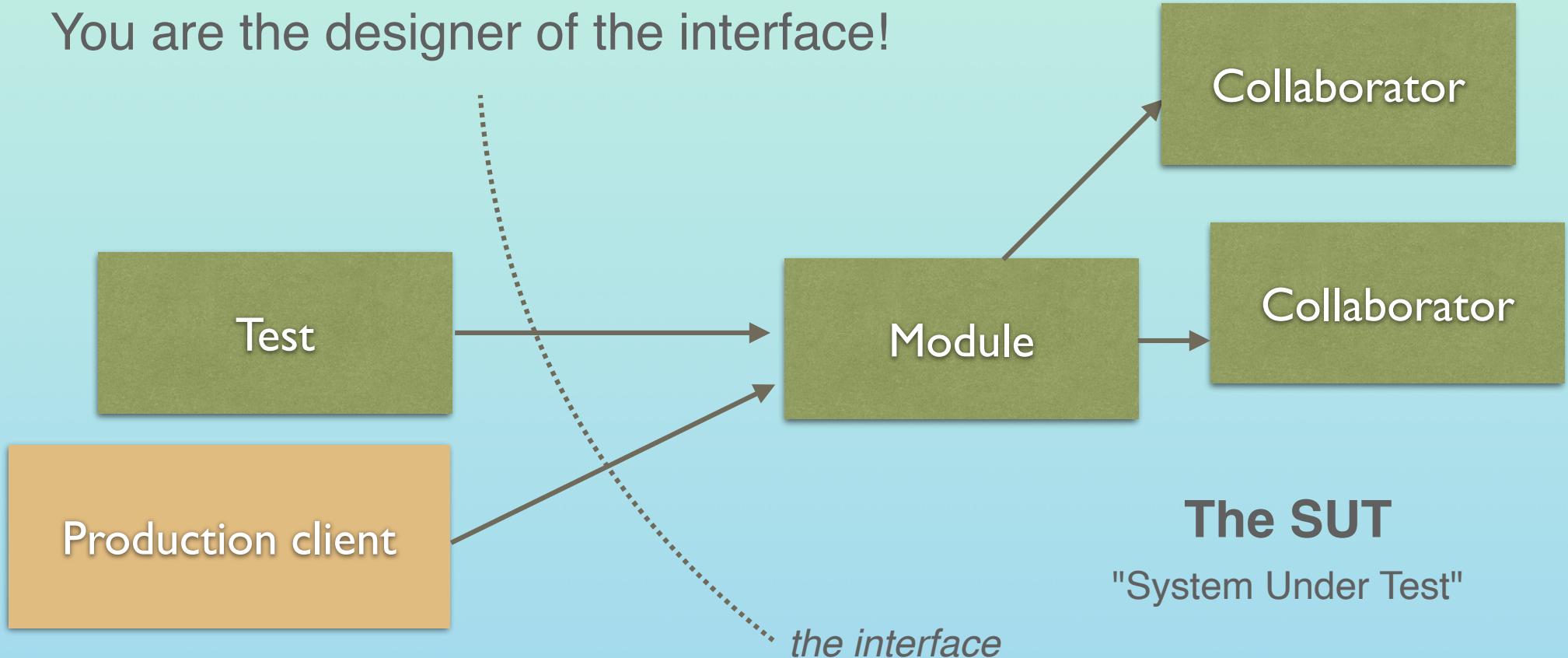
<http://blog.wingman-sw.com/archives/677>

Test Lists

- **Z:** No files opened
- **O:** One file opened
- **M:** Multiple files opened
- **B:** One more than capacity opened
- **E:** File opened twice

Your Test Is the First Client

You are the designer of the interface!



Structuring the Test

"sifts duplicate filename to top of recently-used list"

■	Arrange	Add file with name opened-A.txt Add file with name opened-B.txt Add file with name opened-C.txt
■	Act	Add file with name opened-A.txt
■	Assert	Expect recently-used list to equal: opened-A.txt opened-C.txt opened-B.txt

Assert-First

Consider working from the outcome.

"shifts duplicate filename to top of recently-used list"

```
expect(recentlyUsedList).toEqual([  
  duplicateOfAddedFirst, '3rd', '2nd' ])
```

Assert-First

Consider working from the outcome.

"shifts duplicate filename to top of recently-used list"

```
add(recentlyUsedList, duplicateOfAddedFirst)
```

```
expect(recentlyUsedList)
  .toEqual([duplicateOfAddedFirst, '3rd', '2nd'])
```

Assert-First

Consider working from the outcome.

"shifts duplicate filename to top of recently-used list"

```
add(recentlyUsedList, '1st')
add(recentlyUsedList, '2nd')
add(recentlyUsedList, '3rd')
const duplicateOfAddedFirst = '1st'
```

```
add(recentlyUsedList, duplicateOfAddedFirst)
```

```
expect(recentlyUsedList)
  .toEqual([duplicateOfAddedFirst, '3rd', '2nd'])
```

Exercise: Stock Portfolio Module

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?
- throw a RangeError when buying non-positive shares



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

*** Remember! ***

Red: Ensure test fails

Green: Implement no more code than necessary

Refactor: Clarify and eliminate all duplication

	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	
医薬品		+0.22	銀行	64

A Functional Implementation

Store portfolio info in a state object:

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

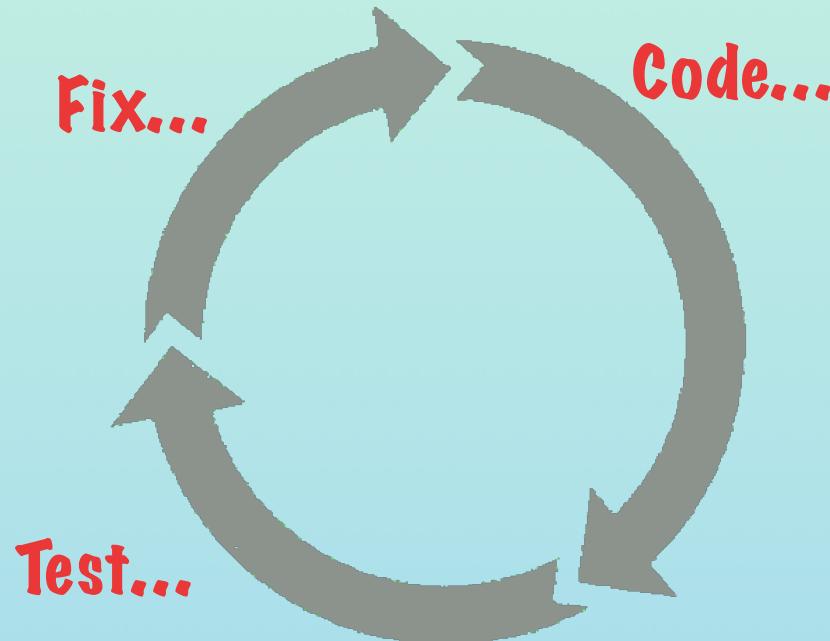
Pass state to each function, return updated state:

```
const create = () => ({ isEmpty: true })  
  
const portfolio = create()  
const newPortfolio = purchase(portfolio, 'BAYN', 42)
```

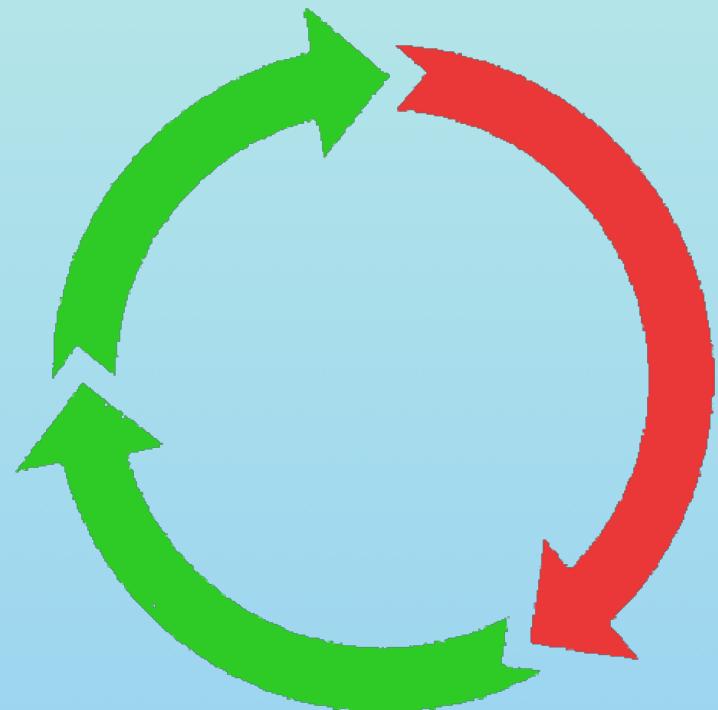
Ensure your function creates no side-effects!

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

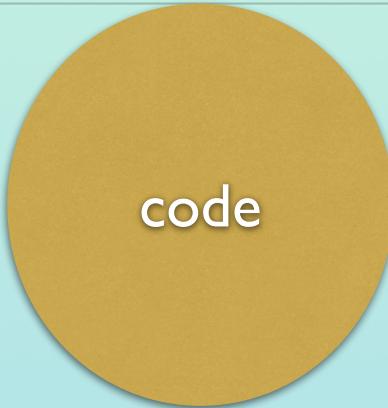
TAD** and TDD: What's the Difference?



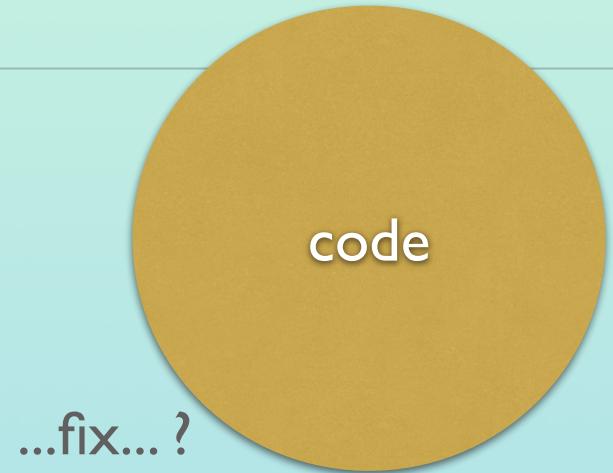
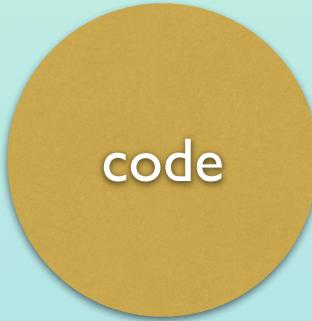
"Test-After Development"



Small Increments



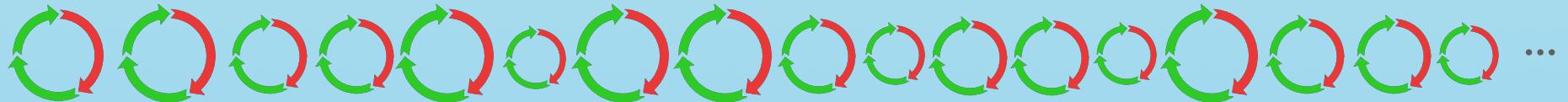
...fix... ?



...fix...

TAD

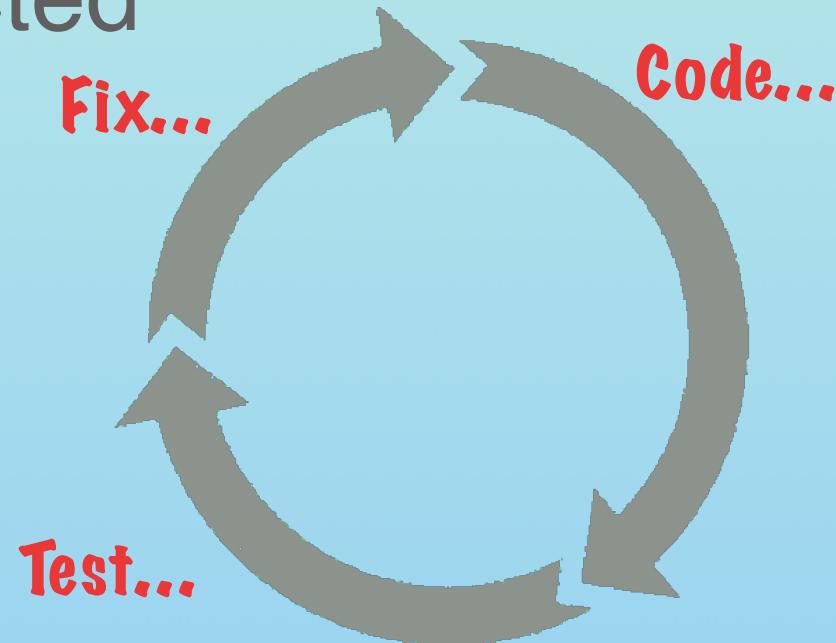
VS.



TDD

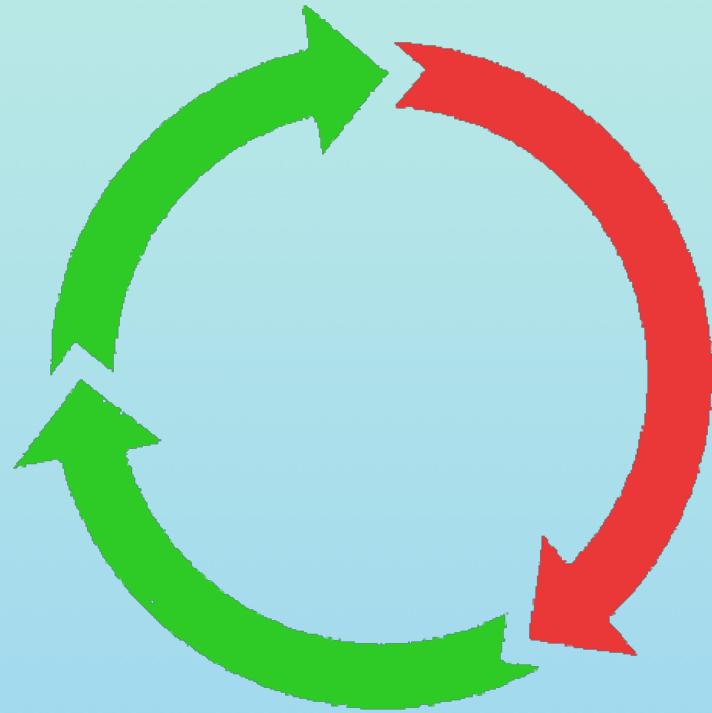
Test-After Development (TAD)

- 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





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>

<https://projecteuler.net/archives>



Roman Numeral Converter

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



Quick Feedback

How do you feel?

5 Great

4 Good

3 Whatever

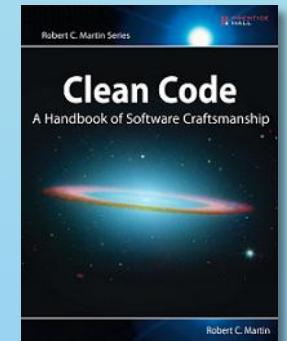
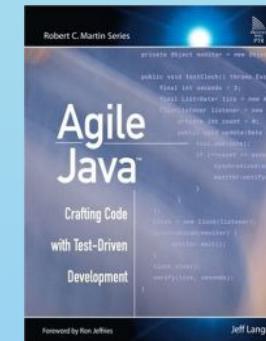
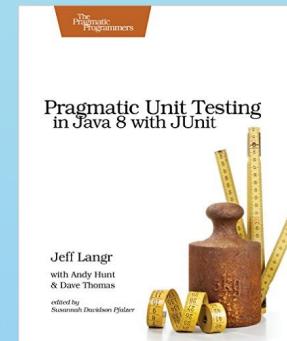
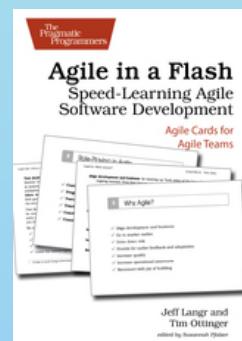
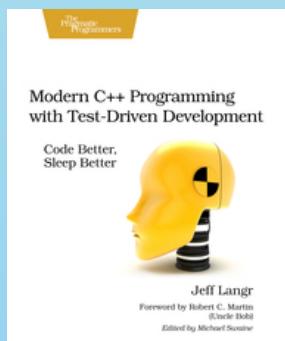
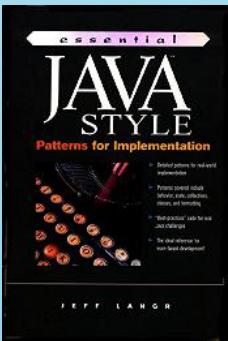
2 Eh

1 WTF

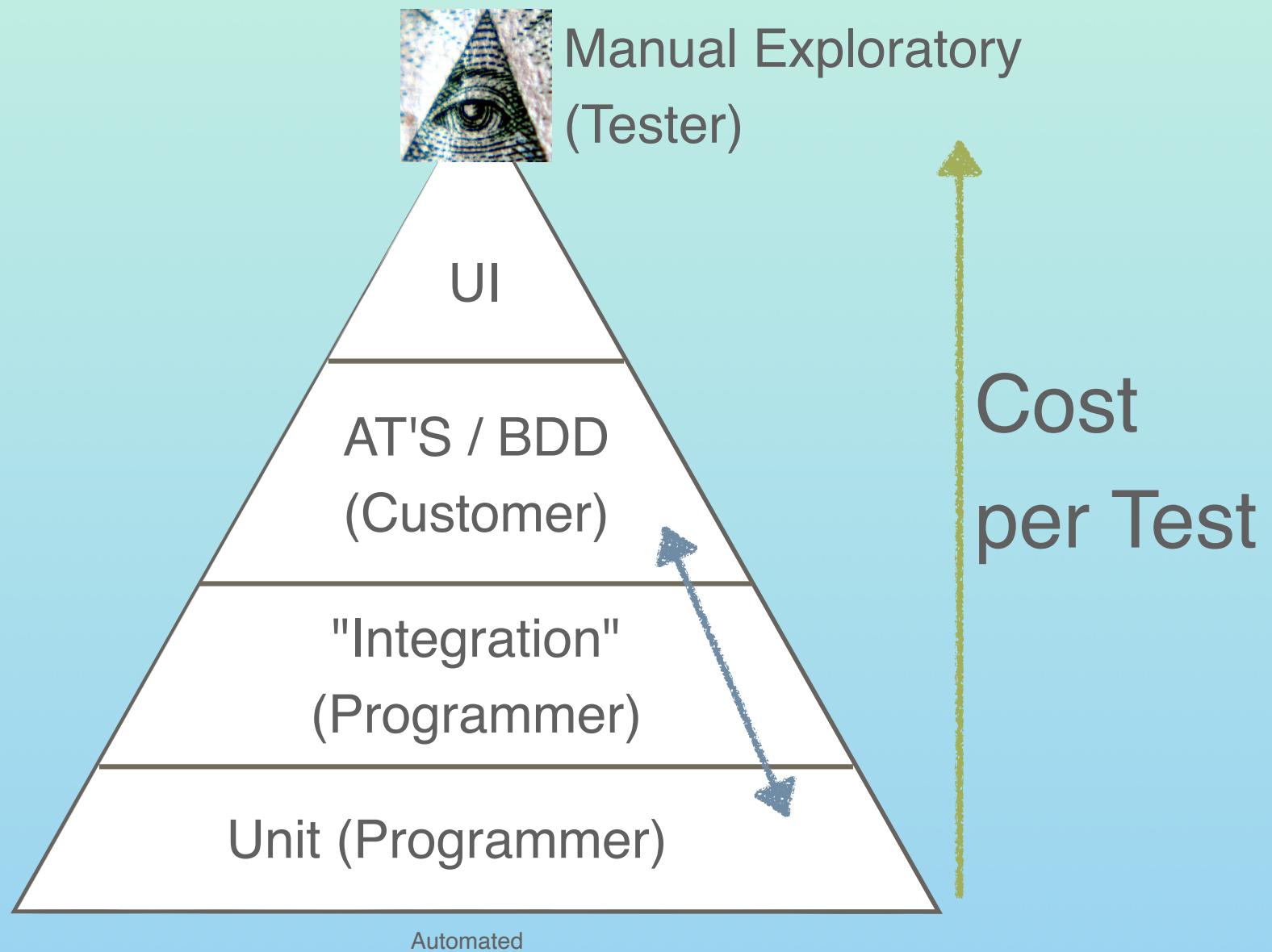
Where Does TDD Fit?



LANGR | SOFTWARE SOLUTIONS



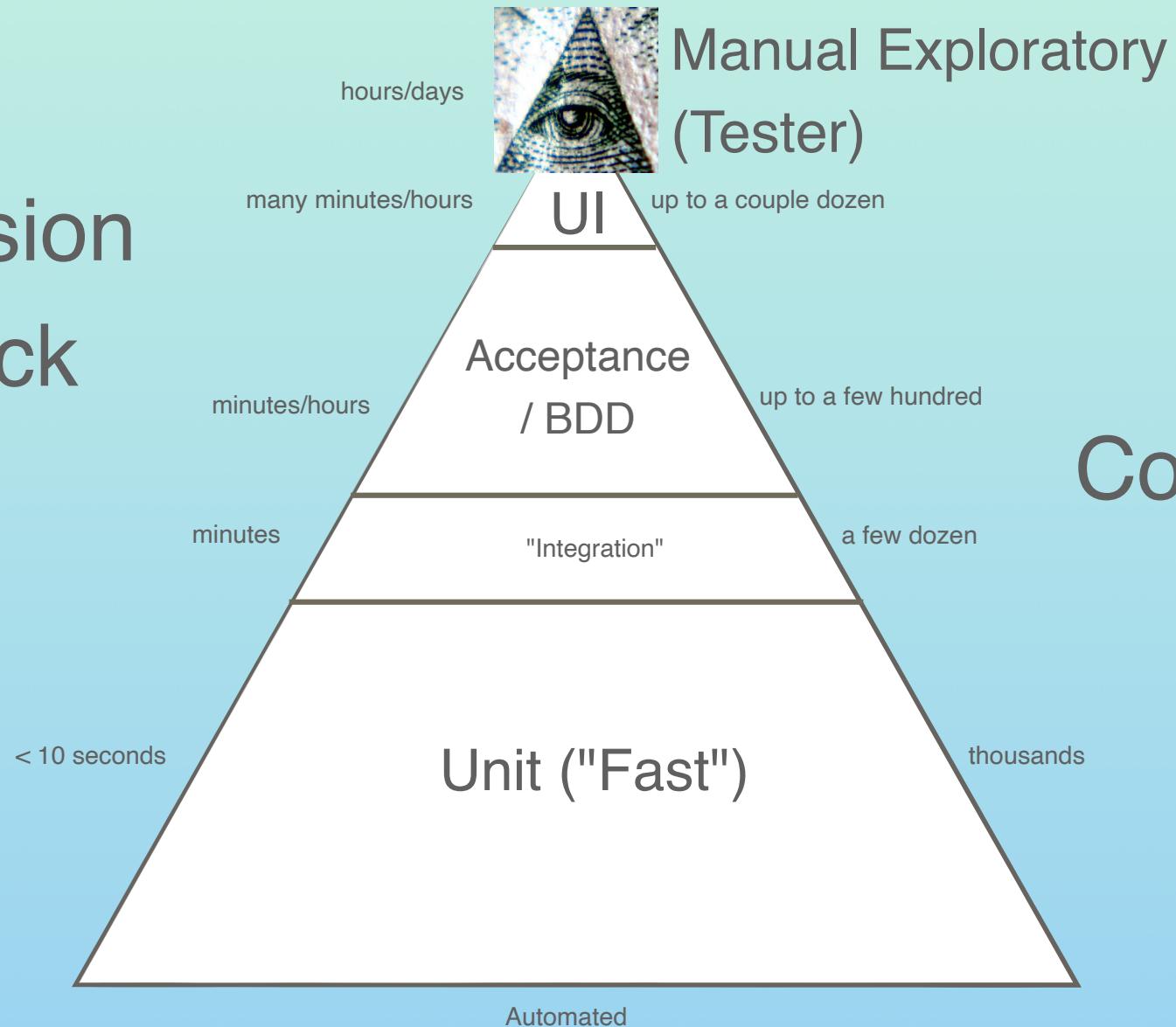
Unit Testing: Insufficient!



A Re-leveled Pyramid

Regression
Feedback

Count



Feedback Q's

■ :-)

What have I learned of value?

■ %-l

What, if anything, am I unclear about regarding the discussion / exercises?

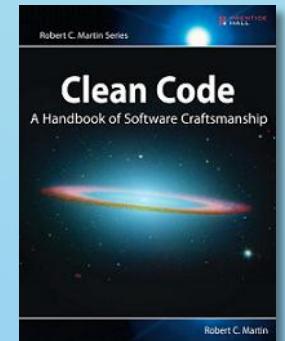
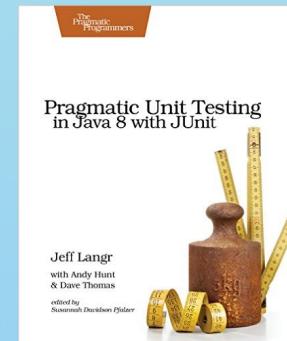
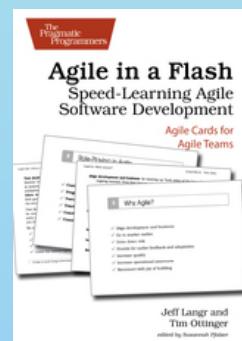
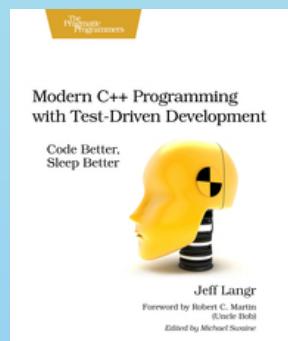
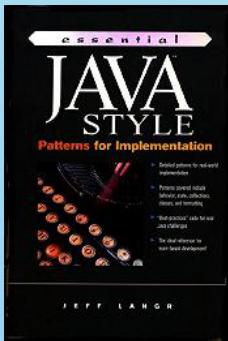
■ ?:-/

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

Documentation



LANGR | SOFTWARE SOLUTIONS



Tests as Documents

what we're describing

```
describe('an unstarted auto', () => {
```

```
  let auto
```

```
  beforeEach(() => { auto = new Auto(); })
```

the generalized behavior it supports

```
  it('idles engine when started', () => {
```

```
    auto.depressBrake()
```

```
    auto.pressStartButton()
```

```
    expect(auto.RPM()).
```

```
      toBeWithinRange(950, 1100)
```

```
  })
```

```
})
```

an example of that behavior

Seek to first understand through the tests.

Behavior-Driven 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('x', ...)
```

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

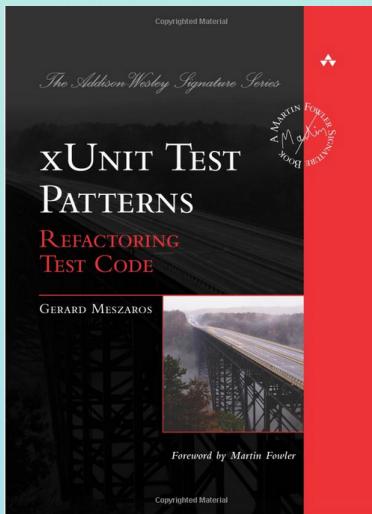
```
it('verifies engine start idle speed', ...)
```

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

- Review often and holistically!

Sustainable Tests

*The long
answer:*



A quicker route:

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

Exercise: Test Smells

Find and fix test smells.

Paraphrase cleaned tests
to your pair.



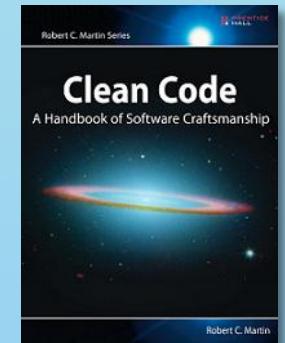
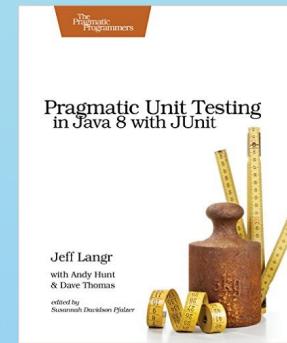
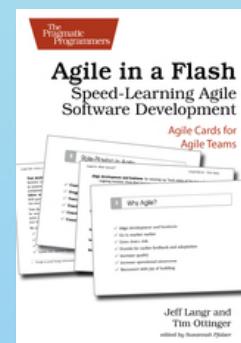
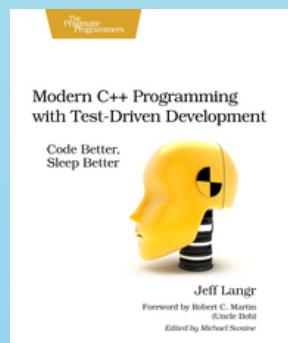
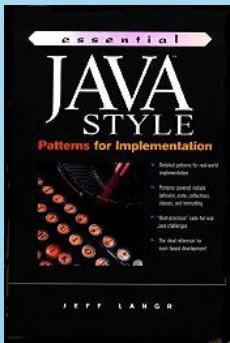
Further instructions: `./src/pos/routes/checkout.test.js`



Continual Design



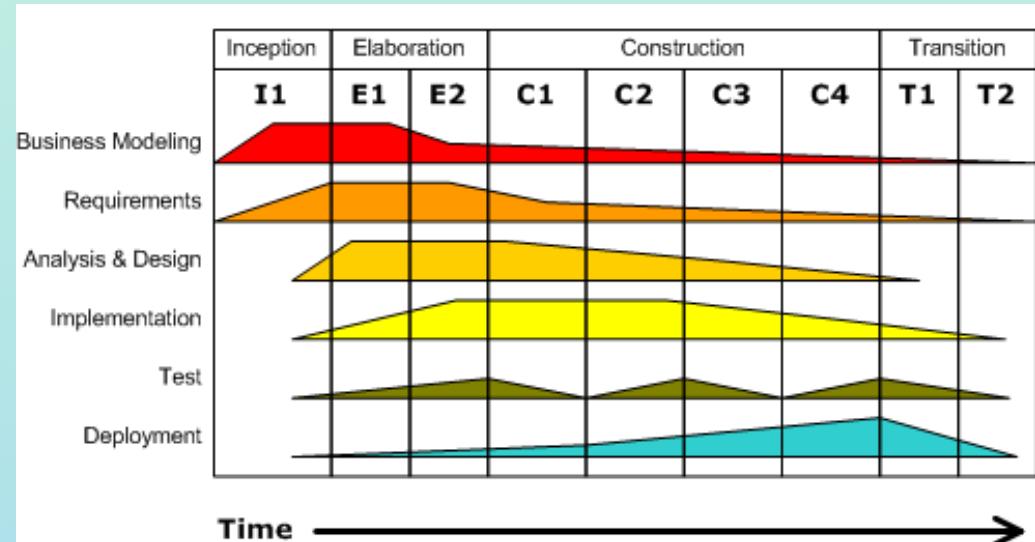
LANGR | SOFTWARE SOLUTIONS



Software Development

■ Activities:

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

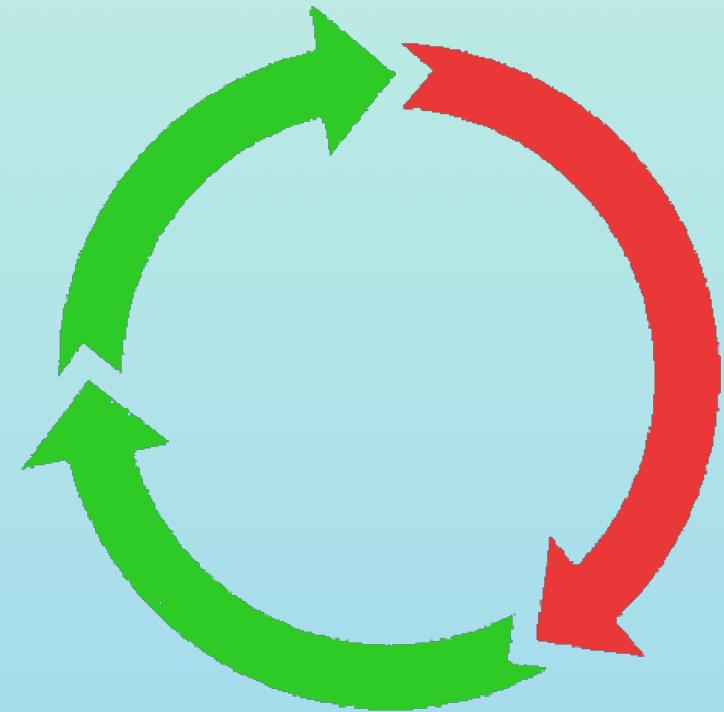


■ Agile:

- *All activities all the time*

TDD: Continual ...

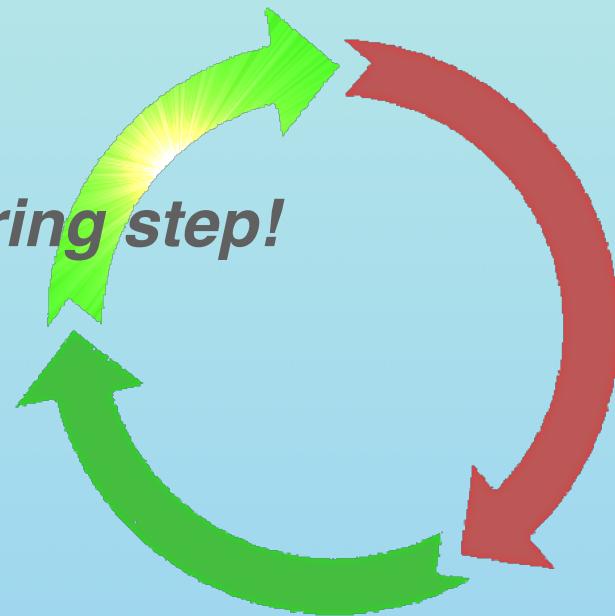
- Testing
- Coding
- Design
- Documentation
- Review



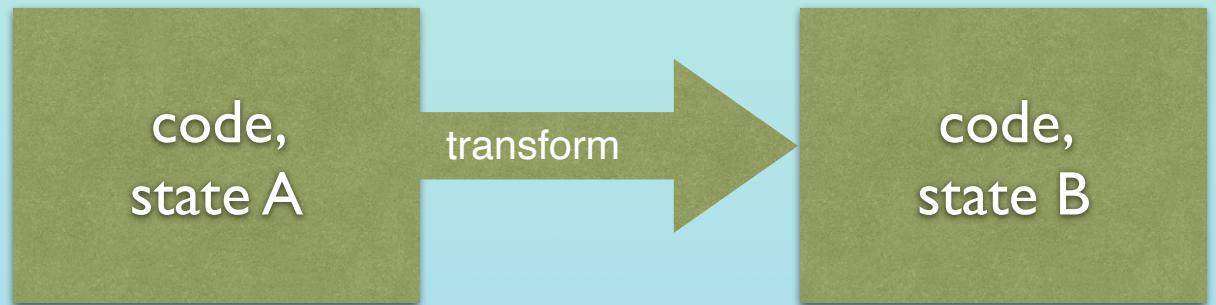
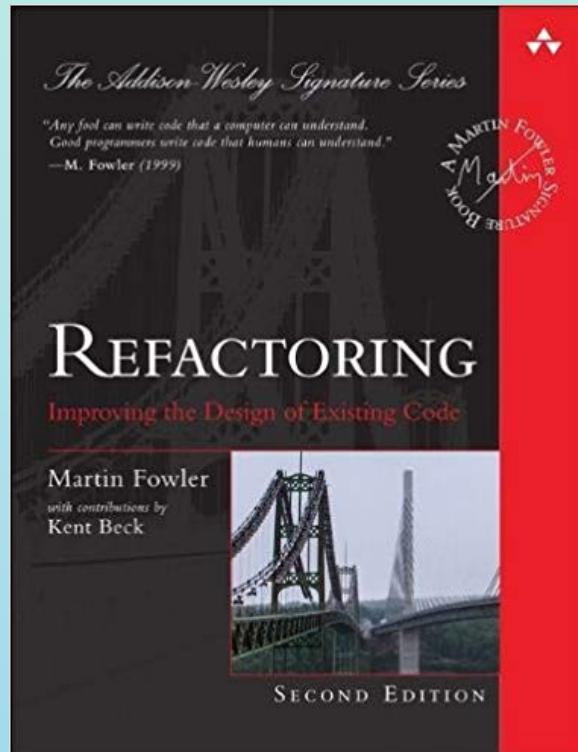
Continual Design

- Always retain an optimal design
- Entropy is unavoidable

Never skimp on the refactoring step!

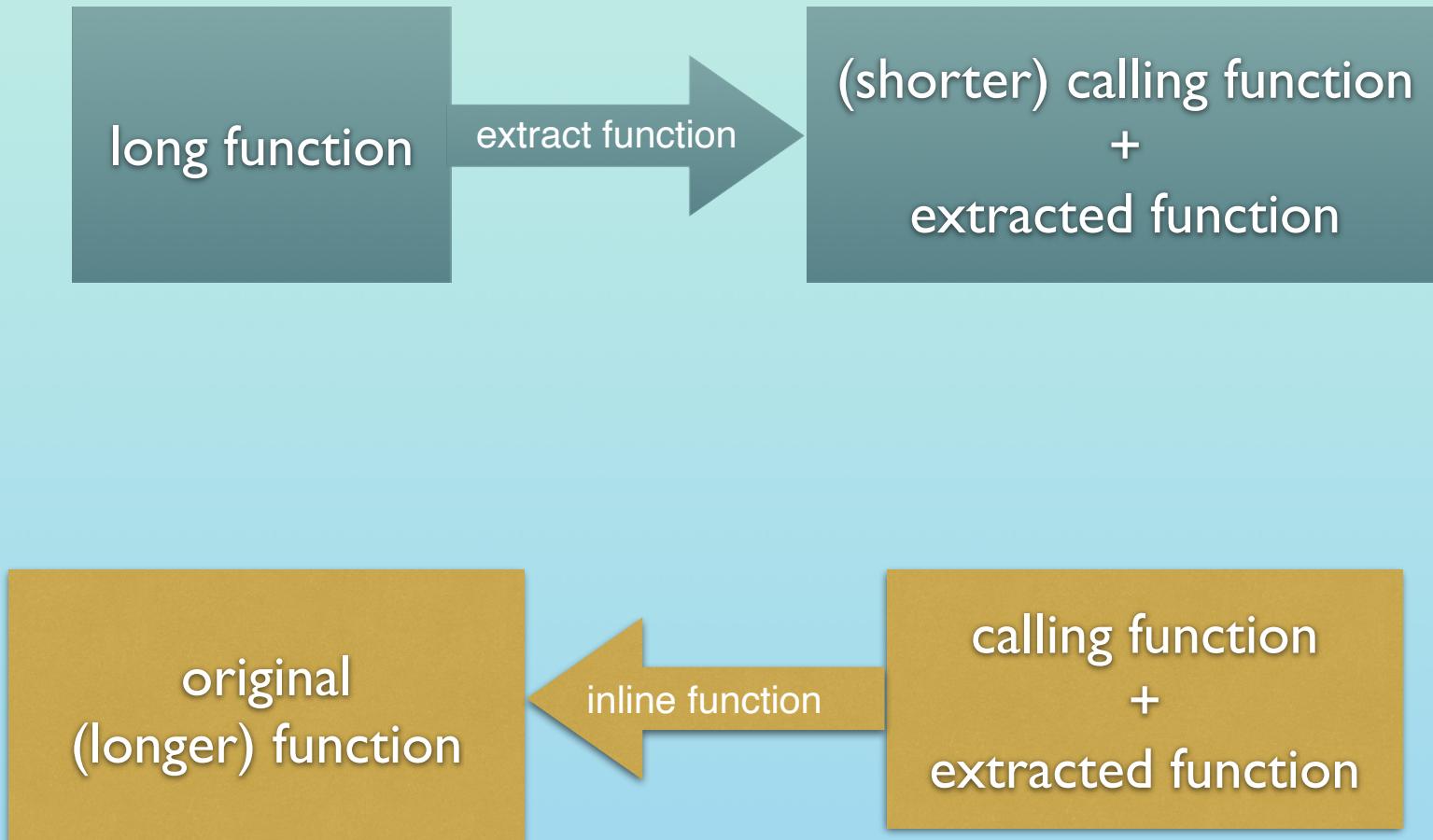


Refactoring



Same "*externally recognized*" behavior

For (*almost*) Every Transform...



Code Smells

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

An aroma?



...or a stink?



"A complete code smells reference:"

<https://github.com/lee-dohm/code-smells>

Exercise: Code Smells

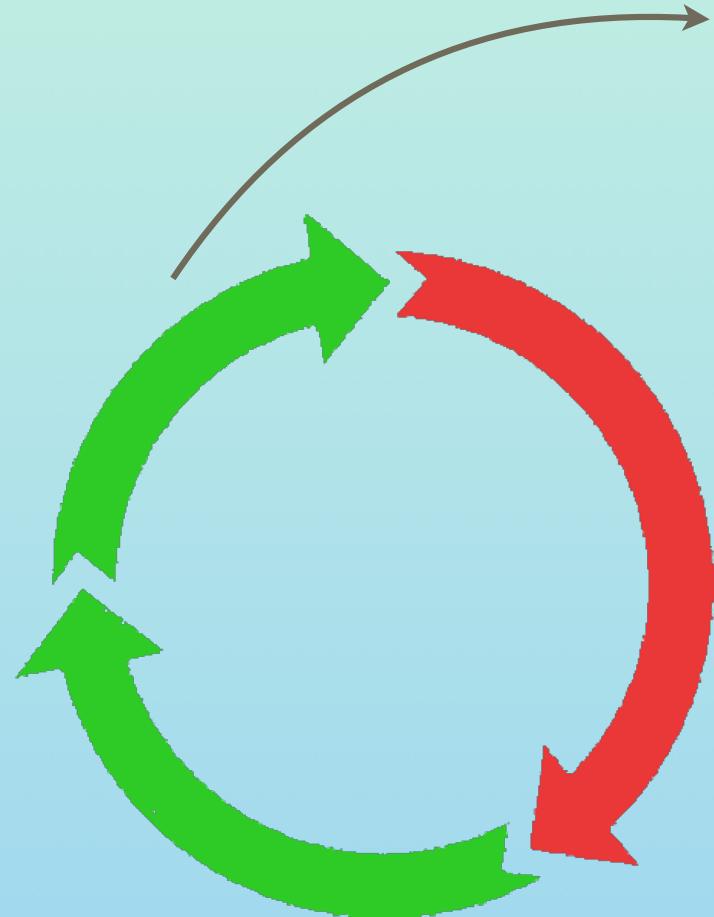
Identify test smells.



See `./src/pos/routes/checkout.test.js`



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 uglyFunction = () => {
  // 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);

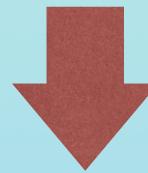
    total += discountedPrice;

    // 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

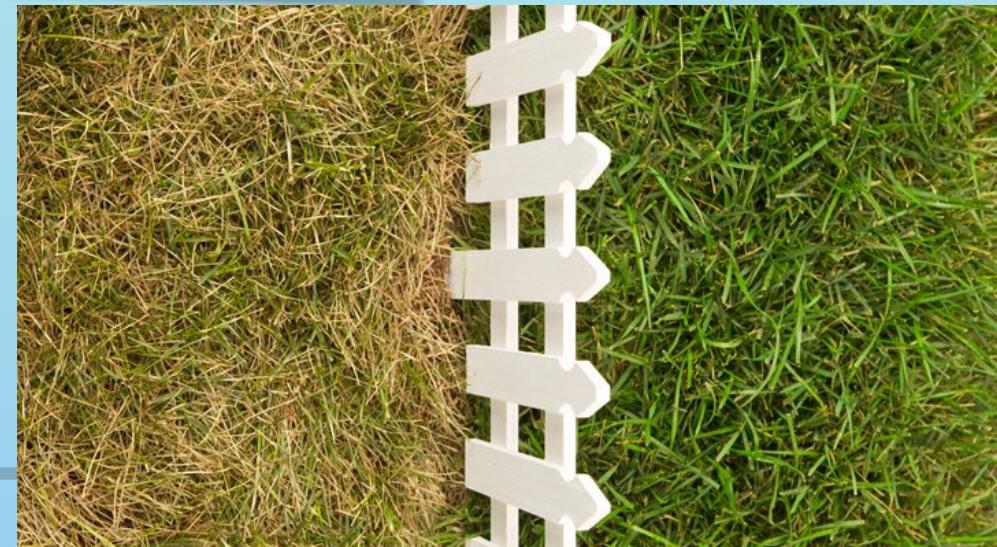
```
import * as GreenerGrass from './otherside'

const thisSideFunction = () => {
  const x = helperFunction(someValue)
  ...
}

const helperFunction = someValue => {
  const result = GreenerGrass.doStuff(someValue)
  const newResult = GreenerGrass.doMoreStuff(result)
  return GreenerGrass.answer(newResult)
}
```

thisside.js

What's the problem?



Refactoring: Move Function

```
import * as GreenerGrass from './otherside'

const thisSideFunction = () => {
  const x = GreenerGrass.helperFunction(someValue)
  ...
}
```

thisside.js

```
export const homebodyMethod = someValue => {
  const result = doStuff(someValue)
  const newResult = doMoreStuff(result)
  return answer(newResult)
}
```

otherside.js

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)!



Planning: Up-Front Design

■ Not just once!

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

■ Minimal: sketches & conversations

- Not detailed specs



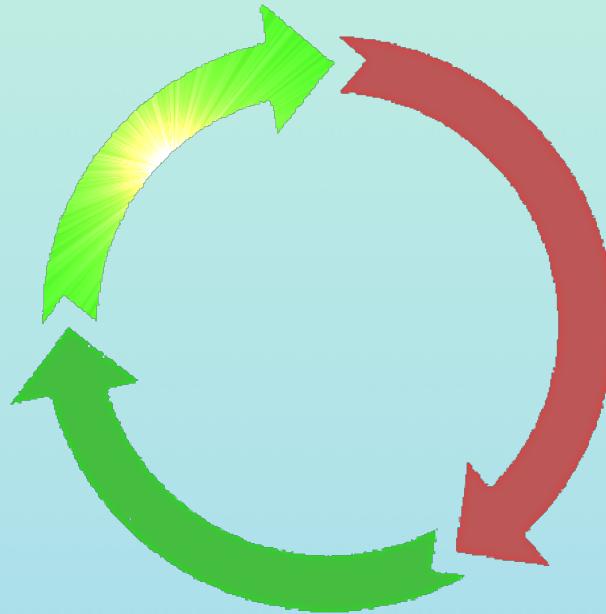
Premature Generalization

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



Refactoring Guidelines

Single-goal



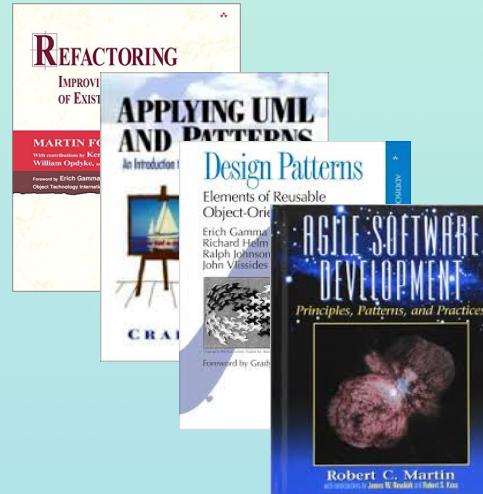
Never skip!

Run *all* tests

It's your responsibility

Design Drivers / Guidelines

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

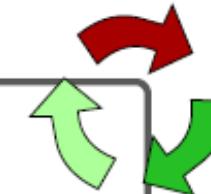


It's all good!

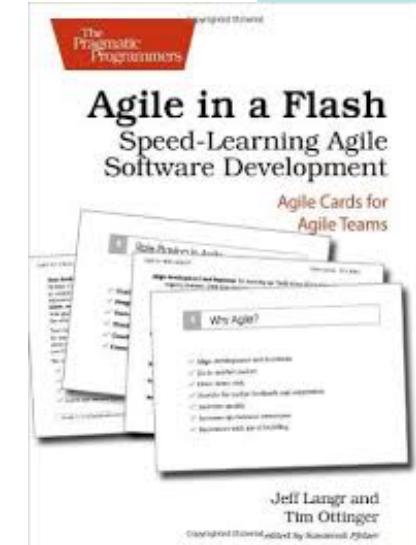
Does everything point to the same place?

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?

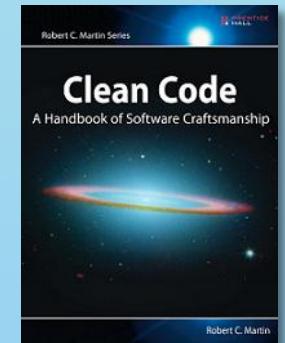
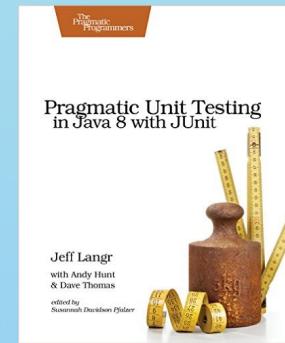
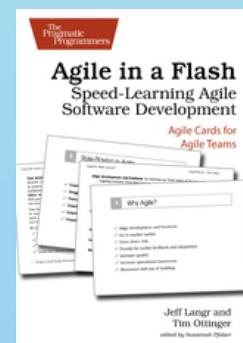
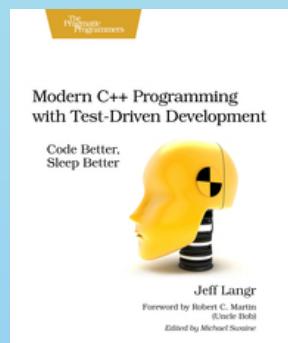
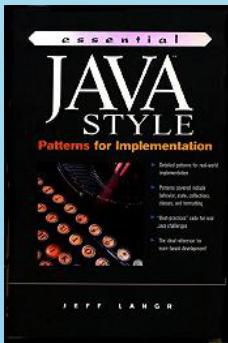




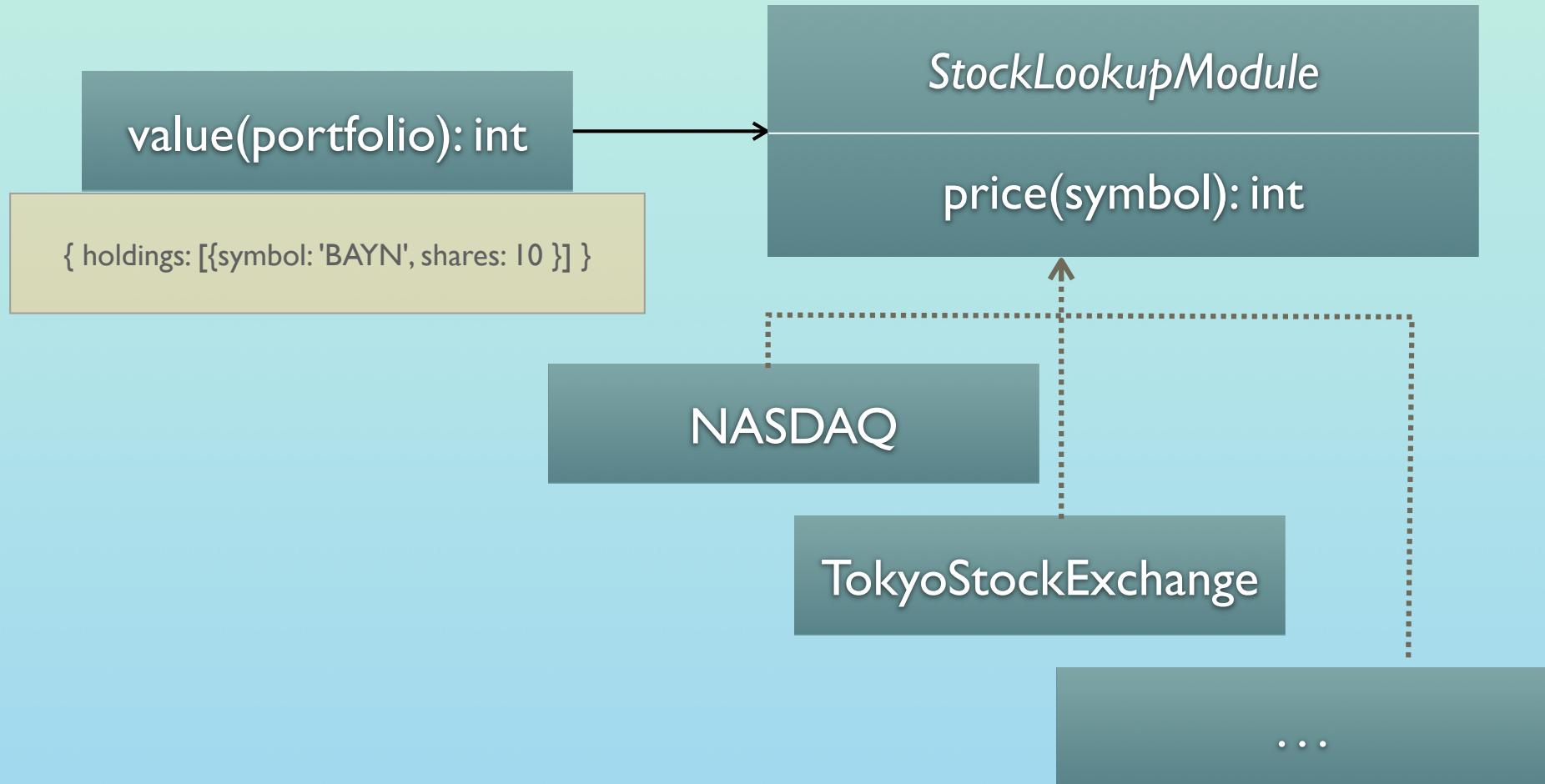
Test Doubles



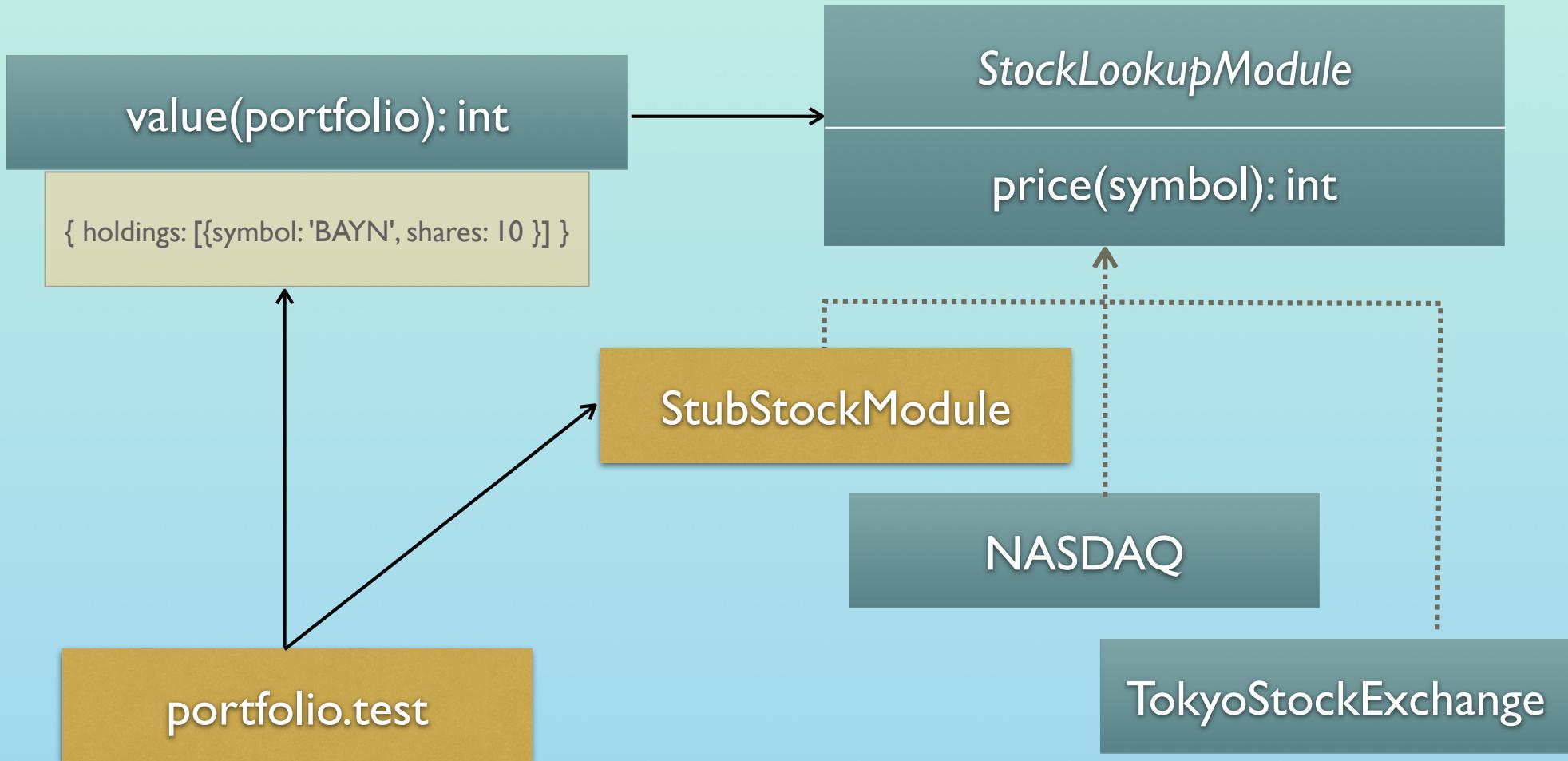
LANGR | SOFTWARE SOLUTIONS



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?
  })
})
```

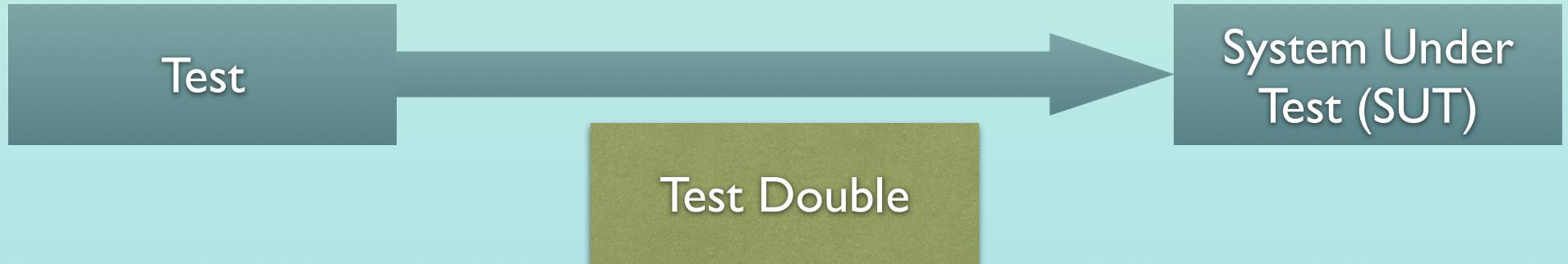
Some notion of an implementation in portfolio.js:

```
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 */ }
```

Test Double Injection Techniques



Functional

Function argument
import * Function Override
exports injection

OO

Constructor / setter
Prototype
Method override

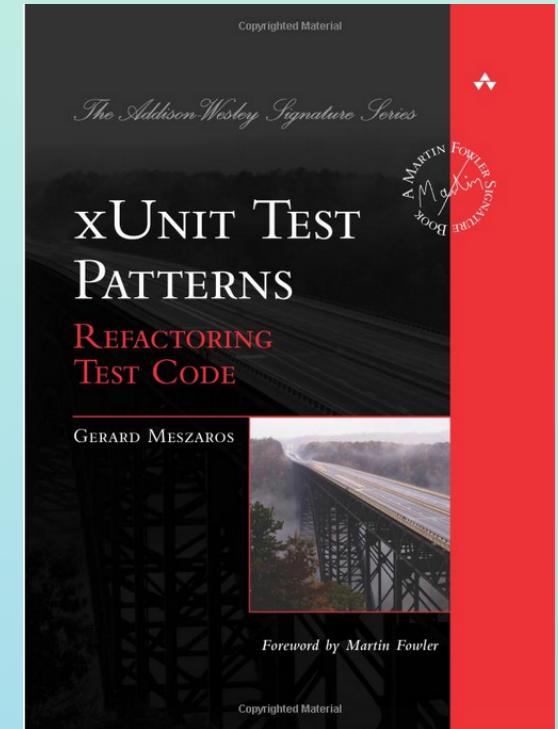
Test Double Terms

Stub: dumb emulation

Spy: captures values to verify

Mock: self-verifies

Fake: whole collaborator emulation



Mock Tools



History suggests more & changes coming!

Jest: A Simple Stub

<https://jestjs.io>

What if the cities are ordered differently?

```
const temperatureServerStub = jest.fn()
temperatureServerStub.mockReturnValueOnce(96)
temperatureServerStub.mockReturnValueOnce(88)

const result = averageTemp(['Miami', 'St. Louis'], temperatureServerStub)
expect(result).toEqual(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
}
```

jest-when

<https://github.com/timkindberg/jest-when>

```
import { when } from 'jest-when'
// ...
const temperatureServerStub = jest.fn()
when(temperatureServerStub).calledWith('Miami').mockReturnValue(96)
when(temperatureServerStub).calledWith('St. Louis').mockReturnValue(88)

const result = averageTemp(['Miami', 'St. Louis'], temperatureServerStub)

expect(result).toEqual(92)
```

npm install --save-dev jest-when

Mocks Are Trackers

<https://jestjs.io/docs/en/mock-function-api.html>

`mockFn.mock.calls`: list of arguments for all calls to `mockFn`

`mockFn.mock.results`: list of results for all calls to `mockFn`

jest.fn() shorthand

```
jest.fn(() => doSomething())
```

is equivalent to:

```
jest.fn().mockImplementation(() => doSomething())
```

Exercise: Jest Stubs



Test-drive the `value()` function for your Portfolio solution, using Jest for stubs.

Jest: Exceptions

```
it('answers 0 when price retrieval throws', () => {
  const portfolio = {}
  purchase(portfolio, "BAYN", 1)

  const priceRetrieveStub =
    jest.fn(() => { throw new Error() })
  const result = value(portfolio, priceRetrieveStub)

  expect(result).toEqual(0)
})
```

Exercise: Jest Exceptions



When calculating the portfolio value,
use zero dollars for the current stock price
if the symbol lookup throws an error.

Verifying a "Tell" (aka Spying)

How to test-drive?

```
const sell = (portfolio, symbol, shares, auditor) {  
  portfolio[symbol] -= shares  
  auditor('sell made')  
}
```

```
const portfolio = {}  
purchase(portfolio, "BAYN", 10)  
const auditorSpy = jest.fn()  
  
sell(portfolio, "BAYN", 3, auditorSpy)  
  
expect(auditorSpy).toHaveBeenCalled()
```

Jest: Was It Called?

```
expect(mock).toHaveBeenCalled()
```

```
expect(mock).toHaveBeenCalledTimes(2)
```

```
expect(mock).toHaveBeenCalledWith(arg1, arg2, ...)
```

```
expect(mock).toHaveBeenCalledWith(1, arg1, arg2, ...)
```

```
expect(mock).toHaveBeenCalledWith(2, arg1, arg2, ...)
```

```
expect(mock).toHaveReturned() // i.e. did not throw error
```

```
expect(mock).toHaveReturnedWith('some value')
```

```
expect(mock).toHaveLastReturnedWith('....')
```

...

Jest Spying: Verifying Arguments

```
expect(auditorStub)
  .toHaveBeenCalledWith('sold shares of BAYN', 3)
```

```
const sell = (portfolio, symbol, shares, auditor) => {
  portfolio[symbol] -= shares
  auditor(`sold shares of ${symbol}`, shares)
}
```

Jest Argument Matching

How to verify the date?

```
auditor(`sold shares of ${symbol}`, shares, Date.now())
```

Use `expect.any(constructor)`:

```
expect(auditorStub)
  .toHaveBeenCalledWith(
    'sold shares of BAYN',
    3,
    expect.any(Number))
```

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.

Jest: Replace a Module Function

```
import * as TemperatureServer from './temperature-server'

TemperatureServer.currentTemperature = jest.fn()

it('calculates average temperature', () => {
  TemperatureServer.currentTemperature.mockReturnValueOnce(83)
  TemperatureServer.currentTemperature.mockReturnValueOnce(52)

  const result = averageTemperature(['Miami', 'St. Louis'])

  expect(result).toEqual(67)
})
```

Not as common as mocking a module (next!)

Mocking a Module

```
import { currentTemperature } from './temperature-server'

jest.mock('./temperature-server.js')

it('calculates average temperature', () => {
  currentTemperature.mockReturnValueOnce(82)
  currentTemperature.mockReturnValueOnce(52)

  const result = averageTemperature(['Miami', 'St. Louis'])

  expect(result).toEqual(67)
})
```

All exports in a module are initialized to jest.fn()

Jest: Is That All?

No.

- Manual mocks
- Alternate ways to create an ES6 class mock
- Partial module mocks ("bypassing module mocks")
- Fake timers (`setTimeout`, `setInterval`, etc.)

Schools of Mock

Classic ("Detroit")

algorithmic approach

verification of state

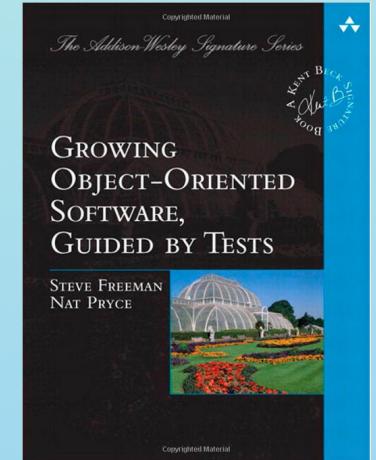
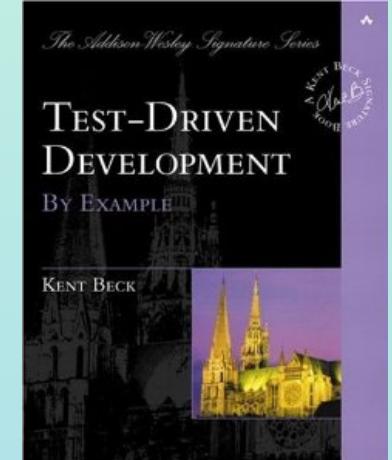
tests drive code from specific to general

London

roles, responsibilities, and interactions

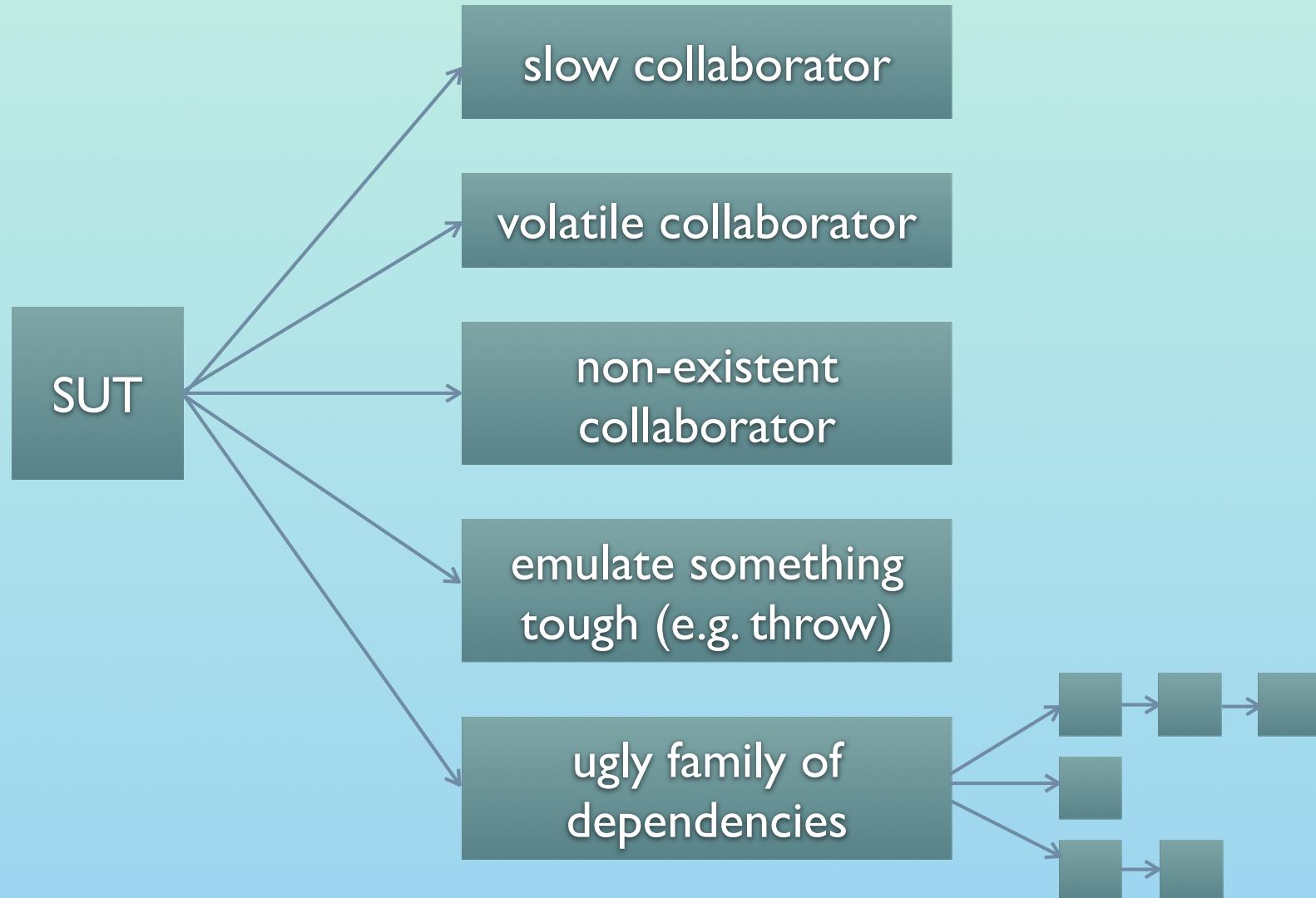
"Fake it until you make it"

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 Mockster



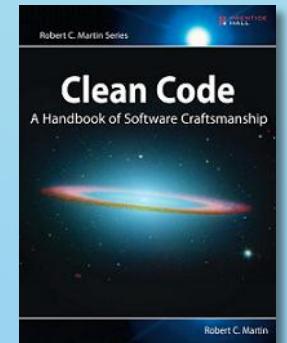
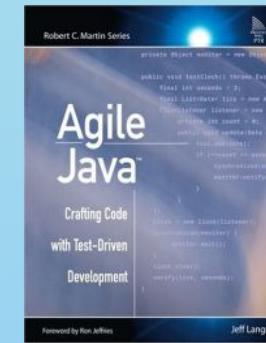
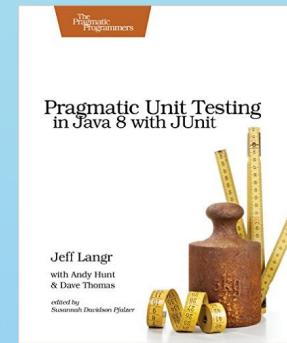
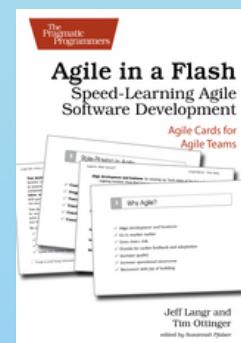
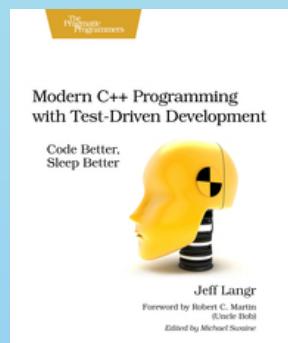
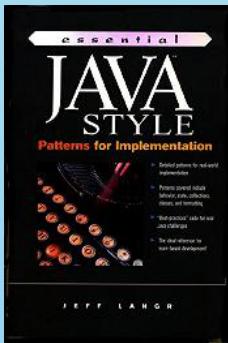
- Writes integration tests where mocks are used
- Watches coupling between tests & implementation
- Seeks to mock only direct collaborators
- Avoids mocking what they don't own
- Avoids fakes (and if necessary, test-drives 'em!)
- Isolates, minimizes the use of test doubles
 - But prefers testability



Growing & Sustaining TDD



LANGR | SOFTWARE
SOLUTIONS



Next Week



- Establish team policies & standards around TDD
- Ensure your build pipeline supports it
- Reviews must include the tests
- Consider pairing / mobbing

Growing TDD

- Sharing sessions
- Challenges / contests
- Coaches & champions
- "Stop the line" mentality
- Influential, dynamic metrics
- More mobbing



Sustaining TDD

Everything under "Growing TDD," plus:

- TDD-specific retrospectives
- Daily stand-down
- "Why" reminders
- Review standards regularly
- Keep the tests fast
- Always use tests as an entry point
- Refactor, refactor, refactor
 - Tests too!



Mastering TDD

TDD is a skill.

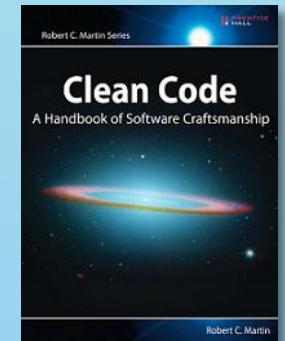
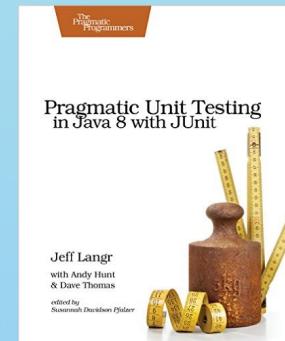
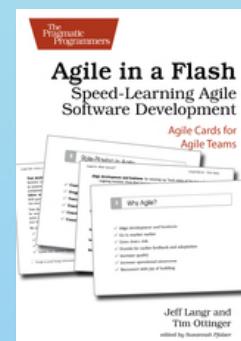
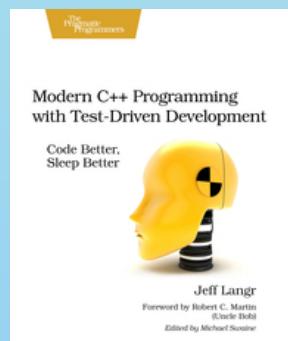
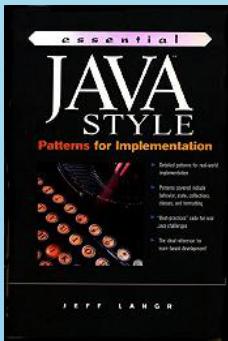


Practice, practice, practice.

Miscellaneous Topics



LANGR | SOFTWARE SOLUTIONS



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);

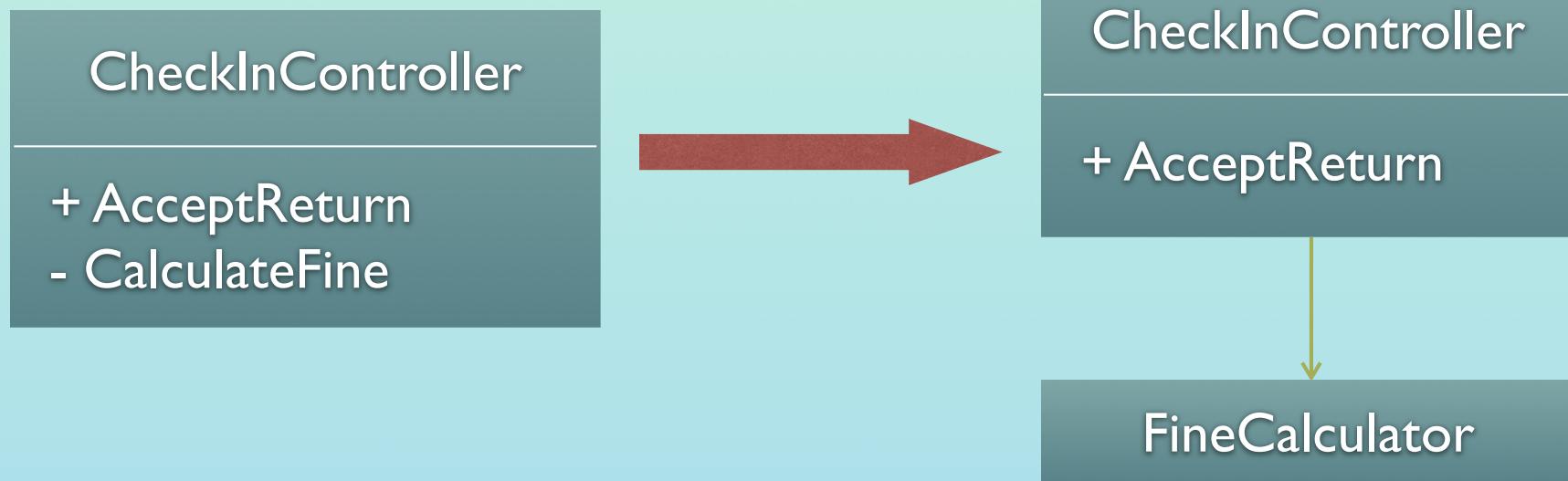
  portfolio.value()
    .then(result => {
      expect(result).to.equal(IBM);
      done();
    });
});
```

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 $< n \text{ ms}$

Fail on tests $> n \text{ ms}$

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

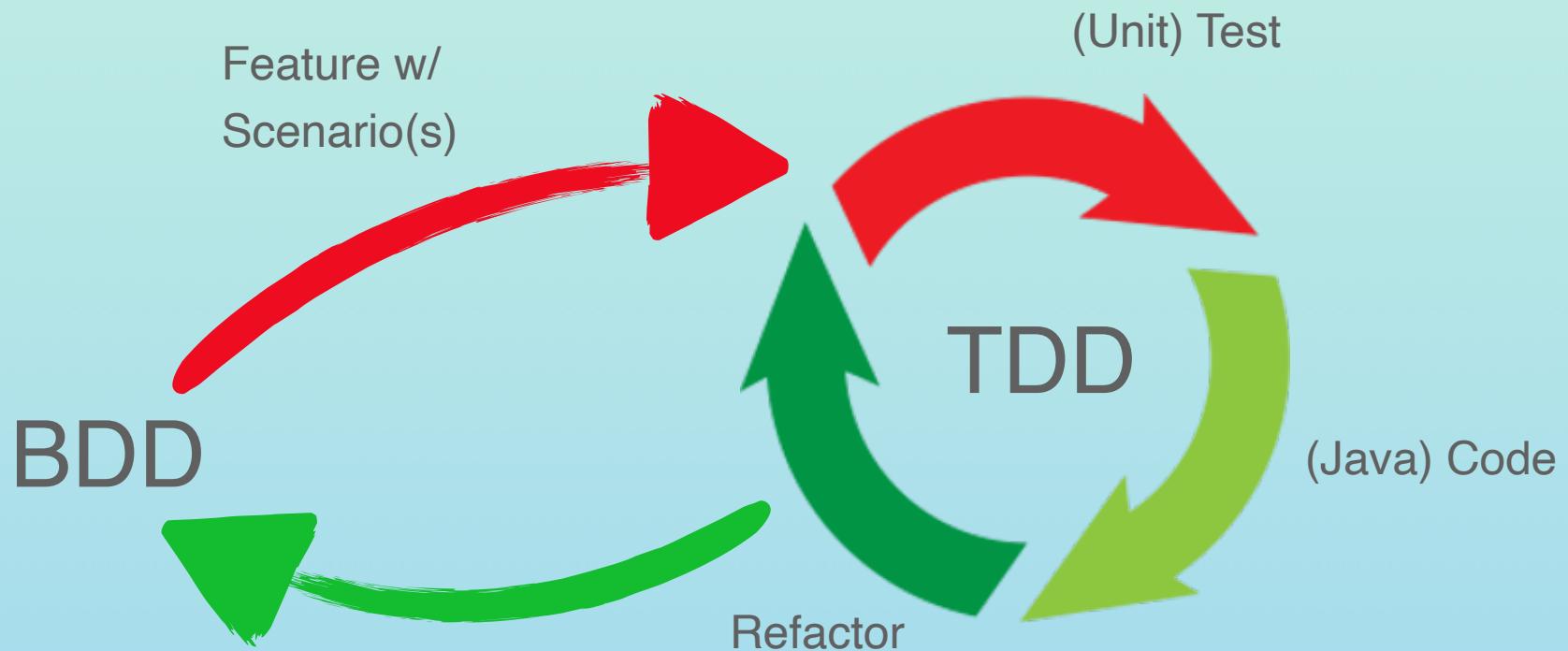
- ({}->nil) no code at all->code that employs nil
- (nil->constant)
- (constant->constant+) a simple constant to a more complex constant
- (constant->scalar) replacing a const. with a variable or an argument
- (statement->statements) adding more unconditional statements.
- (unconditional->if) splitting 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://thecleancoder.blogspot.com/2011/02/fib-t-p-premise.html>

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

Behavior-Driven Development (BDD)



Terminology

TDD = test-driven development

Usually: "**driving code units by describing examples**"

also ...

"driving a system guided by examples"

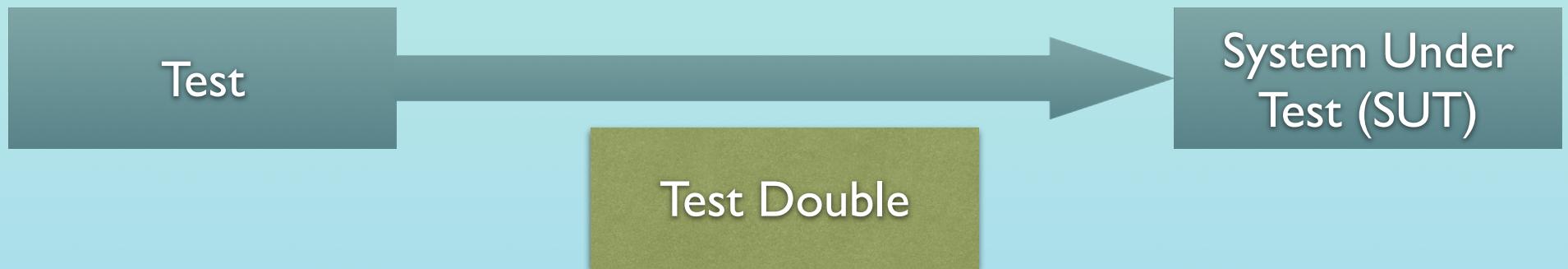
BDD = behavior-driven development

Usually: "**driving a system by describing examples**"

also ...

"driving code units guided by examples"

Alternate Test Double Injection Techniques



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`

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`)
    .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.

Jest: Method Dependency

<https://jestjs.io/docs/en/es6-class-mocks>

```
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
};
```

Jest Method Stub

```
import TemperatureService from './temperature-class'  
import { when } from 'jest-when'
```

```
const mockCurrentTemperature = jest.fn()  
jest.mock('./temperature-class', () =>  
  jest.fn().mockImplementation(() => ({ currentTemperature: mockCurrentTemperature }))  
)
```

```
TemperatureService.mockClear() // in beforeEach  
mockCurrentTemperature.mockClear() // in beforeEach
```

```
const temperatureService = new TemperatureService()  
when(mockCurrentTemperature).calledWith('Miami').mockReturnValueOnce(82)  
when(mockCurrentTemperature).calledWith('St. Louis').mockReturnValueOnce(52)
```

```
const result = averageTemperature(['Miami', 'St. Louis'], temperatureService)  
expect(result).toEqual(67)
```

```
class TemperatureService {  
  currentTemperature(_city) {  
    // ...  
  }  
}
```

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: Old School

portfolio.js

```
var value = function(symbolLookup, 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 = function(symbolLookup) {
  return {
    // ... other funcs ...
    value: value.bind(null, symbolLookup)
  }
}
```

Injecting Via Exports

portfolio.test.js

```
var stubSymbolLookup = jest.fn()
var Portfolio = require('./portfolio')(stubSymbolLookup)

describe('portfolio value', function() {
  it('is symbol price for single-share purchase', function() {
    stubSymbolLookup.mockReturnValueOnce(IBMPrice)
    var portfolio = createPortfolio()

    portfolio = purchase(portfolio, 'IBM', 1)

    expect(value(portfolio)).toEqual(IBMPrice)
  })
})
```

References / Reading List

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

[Beck2002] Beck, Kent. **Test-Driven Development: By Example**. Addison-Wesley, 2002.

[Ellnestam2014] Ellnestam, O. and Brolund, D. **The Mikado Method**. Manning, 2014.

[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!



LANGR SOFTWARE SOLUTIONS

