Odd-e

Learning Objectives

By the end of the course you will be able to:

- Feel the difference between a traditional and an agile team
- Understand the responsibilities of the team in Scrum
- Participate in Scrum Events such as Sprint Planning, Daily Scrum, Retrospectives
- Apply team practices such working agreements, decision making, pair programming
- Apply technical practices such as test-driven development, refactoring, continuous integration
- Apply agile design practices such as emergent design, SOLID principles and patterns

Certified Scrum Developer

- Requirements for Certified Scrum Developer:
  - **At least three days must be technical training through a three-day CSD-track technical skills course.**
  - At least one of the five days must be dedicated to the study of Scrum through a CSD-track Introduction to Scrum course or CSM/ CSPO course.
  - **The fifth day is an elective. You can choose from among the CSD-track courses to develop the skills you feel would benefit you the most in your role**. The two-day CSM or CSPO course fulfills both Intro to Scrum and the elective requirements.

More info at:
http://www.scrumalliance.org/pages/certified_scrum_developer

Agenda (1)

Monday

- Requirement workshop / A-TDD
- SCM, Build Automation and other tools
- Sprint Planning
- Pair Programming

Tuesday

- Continuous Integration and CI Systems
- Using your IDE
- Test-Driven Development
- Working in teams

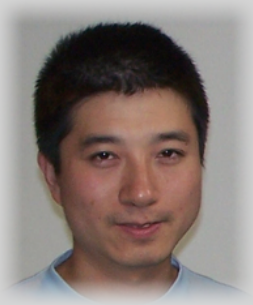
Agenda (2)

Wednesday
- Collective Code Ownership
- Mocking
- Code Smells & Refactoring

Thursday
- Good unit tests
- Thinking about Design
- Working with Legacy Code

Friday
- Craftsmanship
- Retrospective

Course Contributors



Lv Yi

Steven Mak

Stanly Lau

Lasse Koskela

Ebacky

Bas Vodde

The product

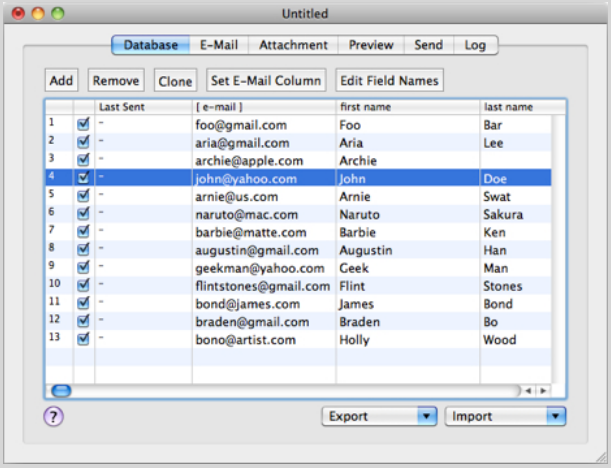
Product Background

Inspector Gadget-a gadget salesman-has a list of contacts to be informed of the latest, greatest and hottest gadgets.

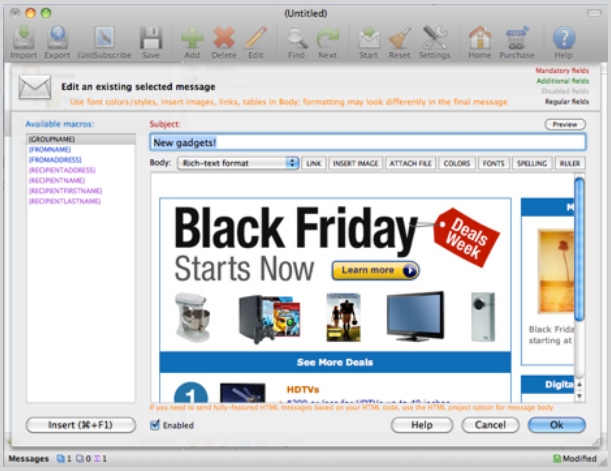
He wants to:

- send the “Gadget & Widget Newsletter”
- send personalized mail based on certain preferences of his contacts
- import/export his contacts from/to different sources
- use different mail servers such as gmail and his own provider

Products he looked at



http://www.falkon-ware.com/SerialMailer



http://www.emailarms.com/products/bulk_mac_mail.html

After careful inspection

None of the existing products fulfills the needs.

Some important missing functionality:

- Doing test-runs to avoid sending to the wrong people
- Being able to cc Penny
- Resending on failures
- Integration with Google
- Friendly to use, after all... he is... inspector gadget

So, he hired you (the team) to create a Personalized Mass Mailer

Project

Build a personalized mass mailer for Inspector Gadget

You:

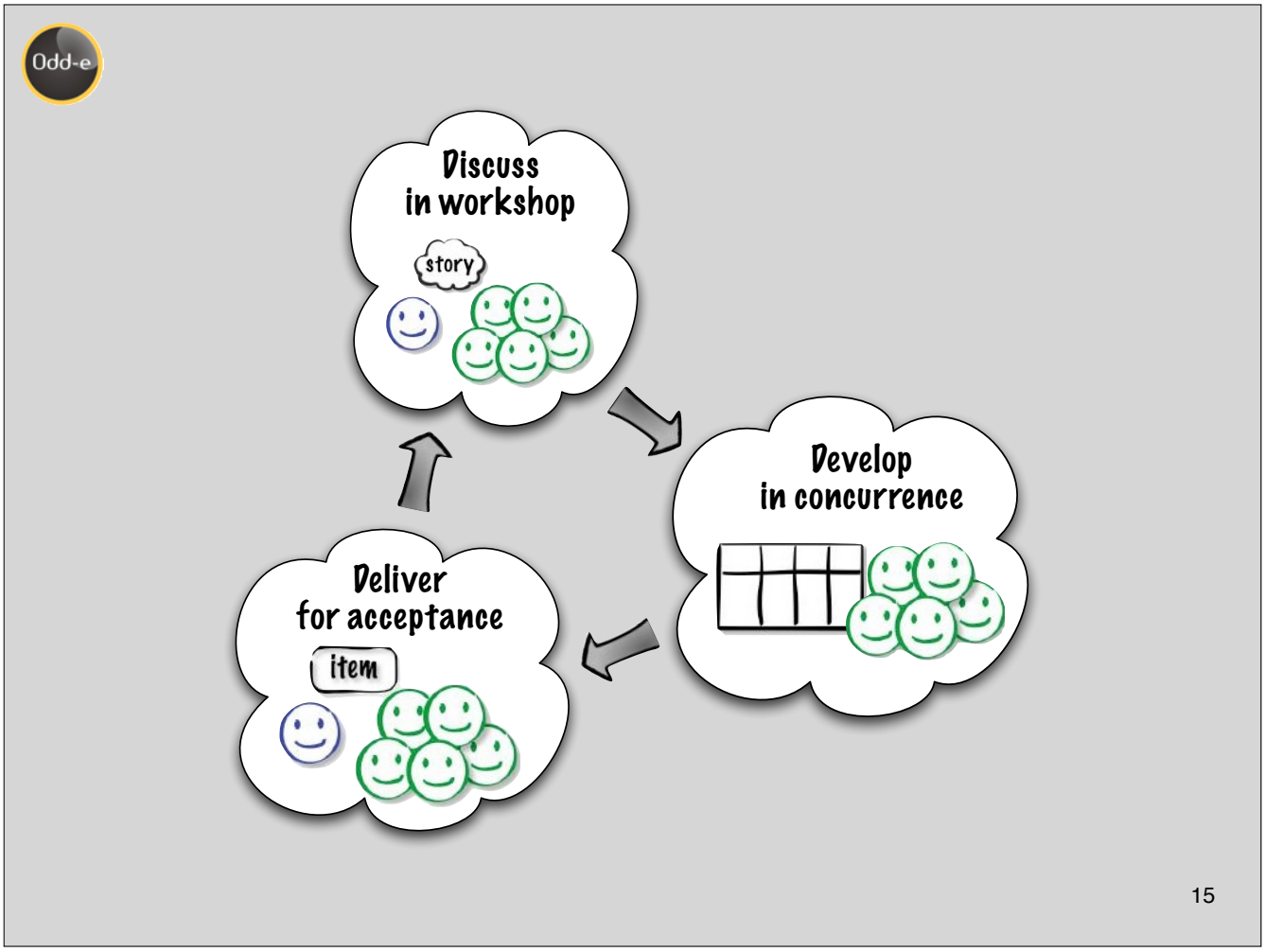
- work as a team
- are guided through the planning & requirement events
- are provided a Product Backlog
- will be introduced to technical practices before applying them
- have one week
- shall have fun!

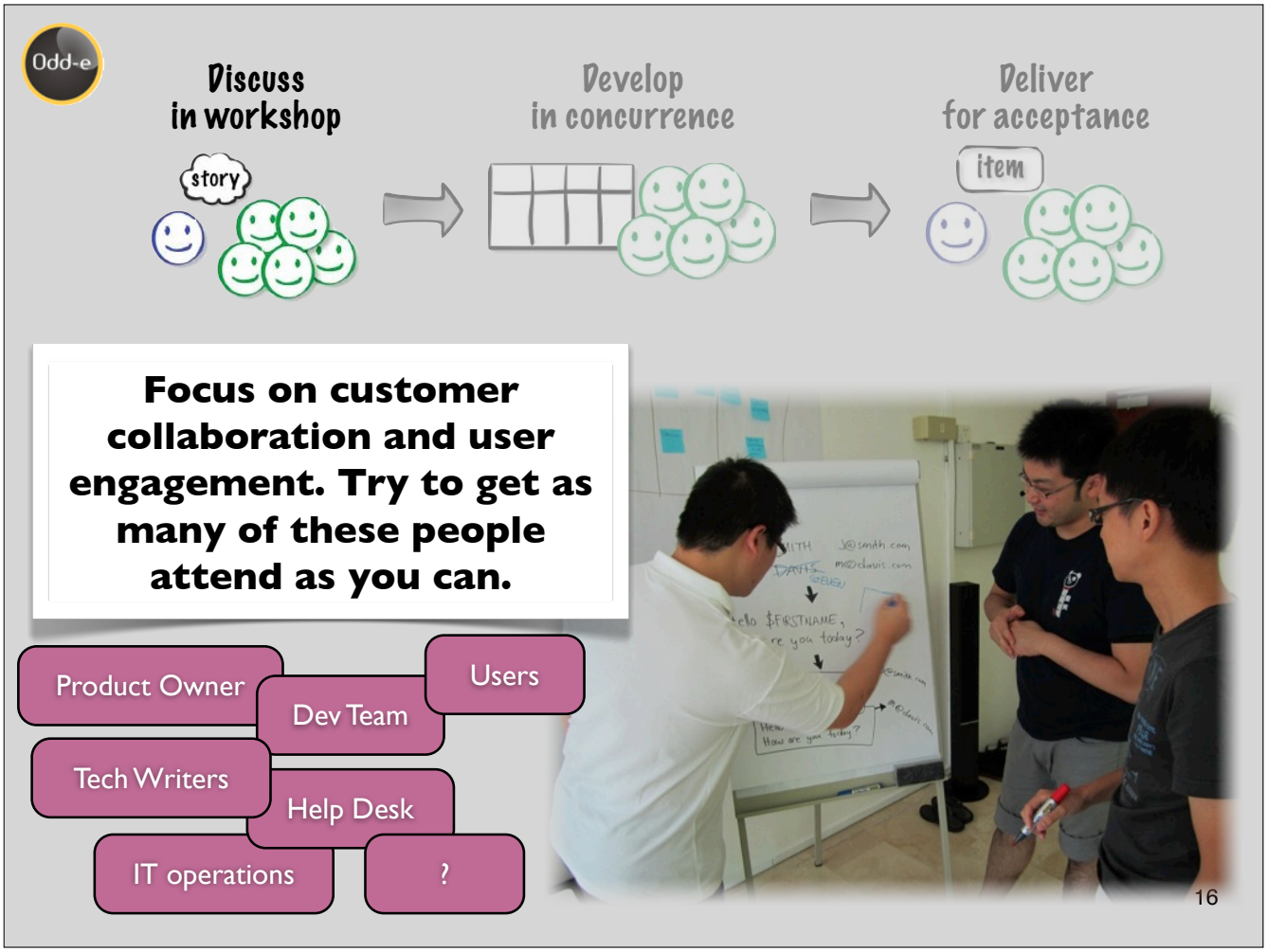
Have you ever...

- built the wrong thing due to misunderstanding
- had PO ask why this and that isn't supported
- had to wait for clarifications mid-sprint
- uncovered false assumption in sprint review
- built too fancy things for what was needed

Collaboration is key

- team gets feedback earlier
- team understands what they're implementing
- shared language and vocabulary is built
- scope of work is clear and understood by all
- team collaborates closely with product owner


Odd-e
Discuss
in workshop
story
Develop
in concurrence
Deliver
for acceptance
item

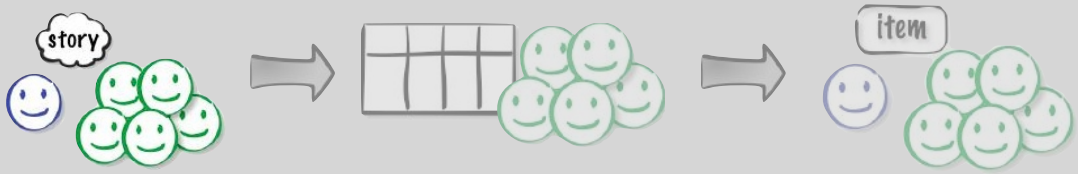

Odd-e
Discuss
in workshop
story
Develop
in concurrence
Deliver
for acceptance
item
Focus on customer collaboration and user engagement. Try to get as many of these people attend as you can.
Product Owner
Dev Team
Users
Tech Writers
Help Desk
IT operations
?

Odd-e

Discuss in workshop

Develop in concurrence

Deliver for acceptance



I want to show scores for the current pair.

OK. Can you give me an example of these scores?

So if the three judges gave scores 7, 11 and 2 then the displayed score should be 20.

OK. And would those individual judges' scores be shown? Or just the total?

Yes, individual scores. And they should be shown all the time, even before all judges have given their scores.

Like if first two judges give a 7 and an 11 then we'll show 18 as an intermediate score?

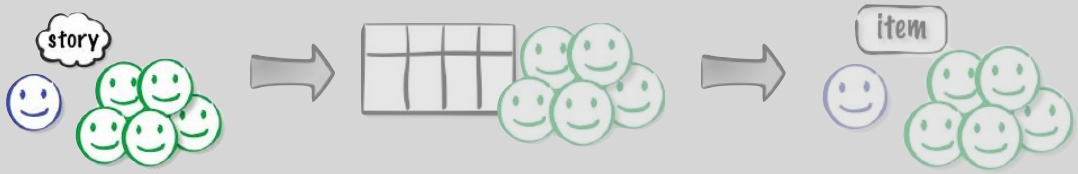
Exactly.

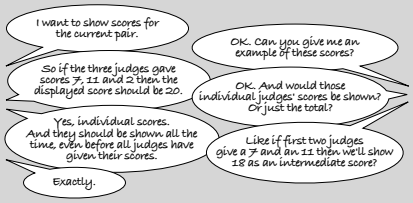
Odd-e

Discuss in workshop

Develop in concurrence

Deliver for acceptance





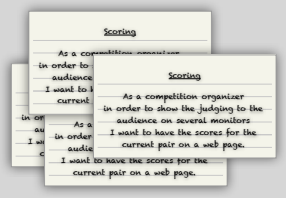
With 3 judges giving scores 4, 20, and 18, the displayed score should be 42.

When the first 2 judges have given their scores, e.g. 10 and 5, the intermediate score of 15 should be displayed already.

No scores displayed as a dash (–), not zero.

Maximum score from a judge is 20 points!

Judge A	B	C	Score
			—
10	5		15
4	20	18	42
21	0	-5	ERROR



Odd-e

Can you give me an example?

What if ... ?

How frequently does this happen?

What other variations of this would be common?

What other types of users would typically do this?

How could this step go wrong?

Is there anything else that would happen?

What would be different after this has happened?

Tell me how this would work?

Where would that data be used?

What would you expect then?

And then...?

What would that look like?

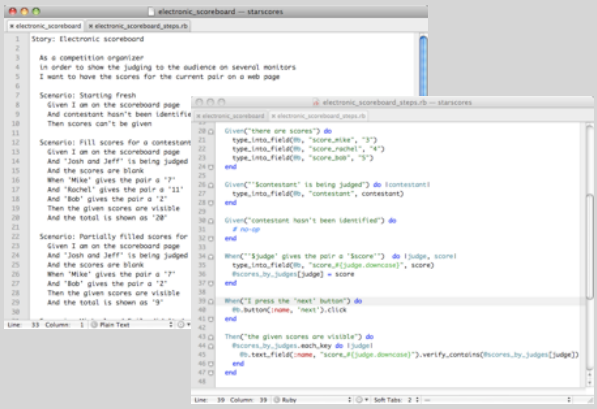
Can you draw that for me?

Odd-e

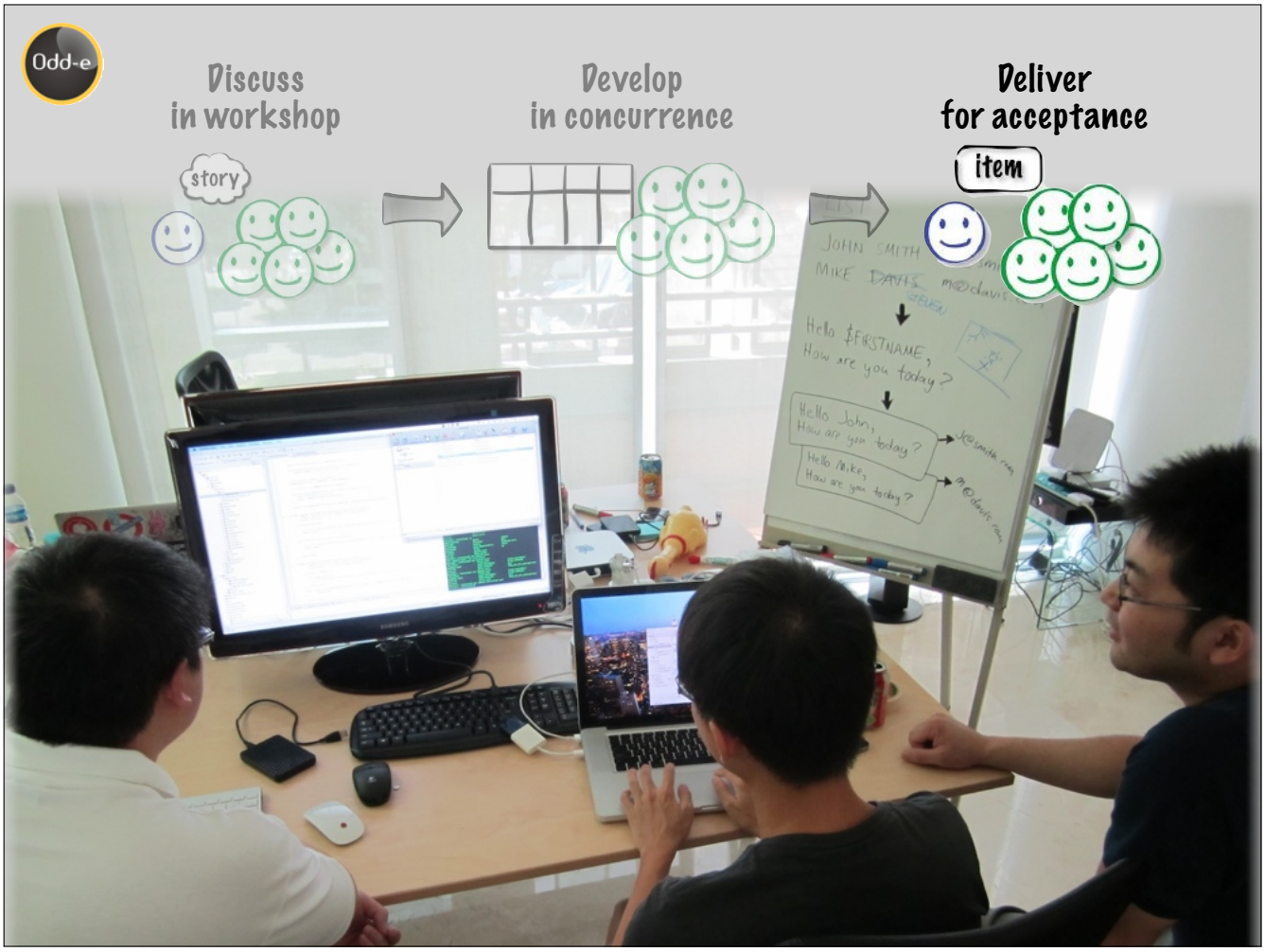
Discuss in workshop

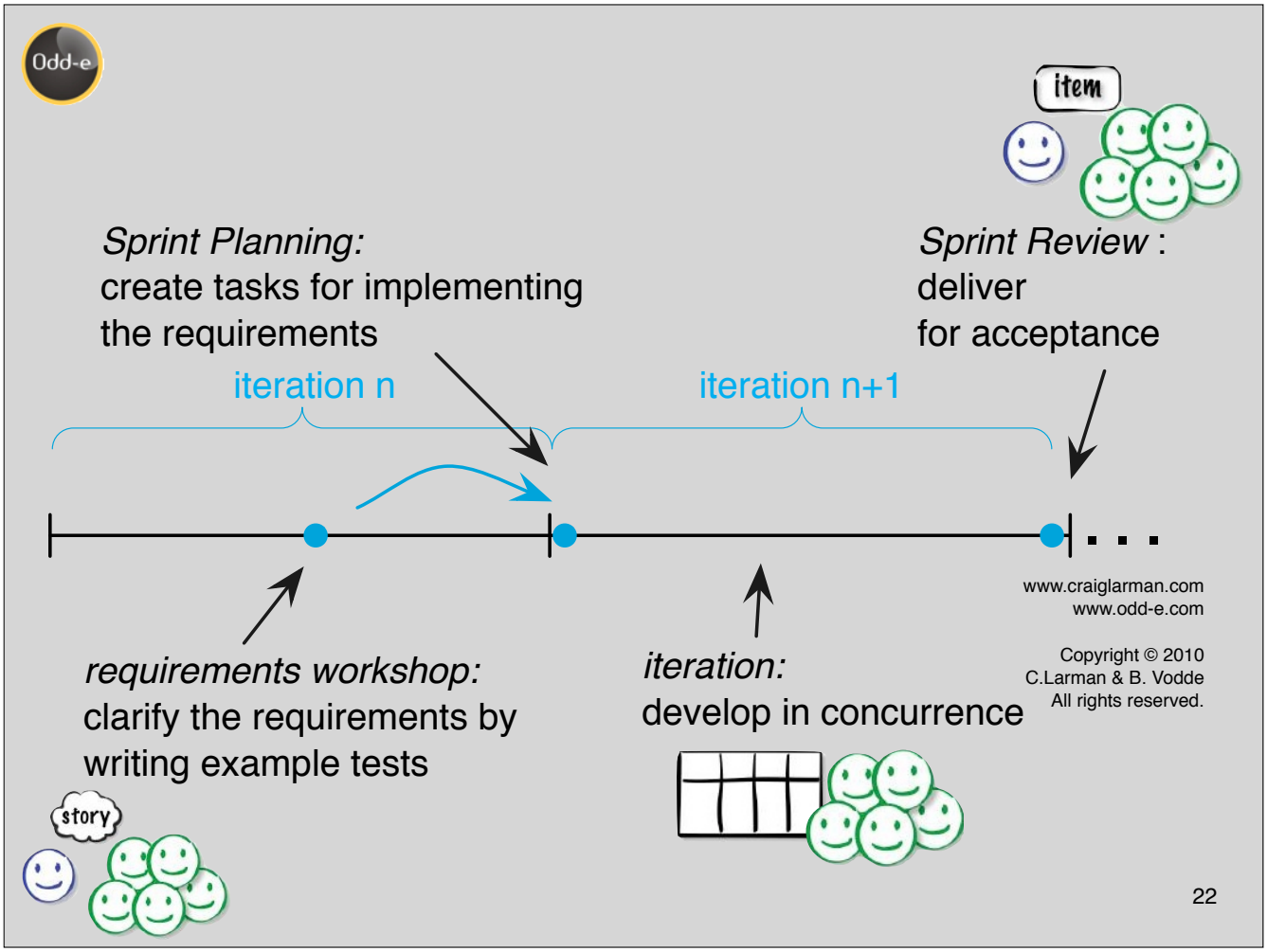
Develop in concurrence

Deliver for acceptance



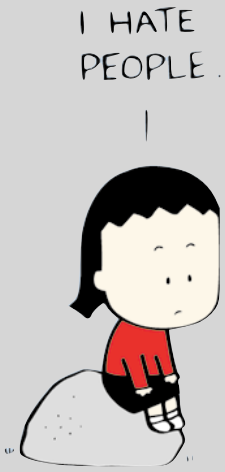
The examples are translated into executable tests, allowing a computer to run them against the product.


Odd-e
Discuss
in workshop
Develop
in concurrence
Deliver
for acceptance
story
item


Odd-e
item
Sprint Planning:
create tasks for implementing
the requirements
Sprint Review :
deliver
for acceptance
iteration n
iteration n+1
requirements workshop:
clarify the requirements by
writing example tests
iteration:
develop in concurrence
story
www.craiglarman.com
www.odd-e.com
Copyright © 2010
C.Larman & B. Vodde
All rights reserved.
22

Requirements workshops are great opportunities to interact with all kinds of stakeholder groups:

- end users
- help desk
- technical support
- marketing
- sales



ATDD in summary

collaborative requirements discovery

tests as requirements, requirements as tests

workshops for clarifying requirements

concurrent engineering

prevention rather than detection

Sample of robotframework

this is the name of a *test case*

these *keywords* form the test case

Test Case	Action	Argument	Argument
User can create an account and log in	Create Valid User	fred	P4ssw0rd
	Attempt to Login with Credentials	fred	P4ssw0rd
	Status Should Be	Logged In	
User cannot log in with bad password	Create Valid User	betty	P4ssw0rd
	Attempt to Login with Credentials	betty	wrong
	Status Should Be	Access Denied	

keywords receive *arguments*

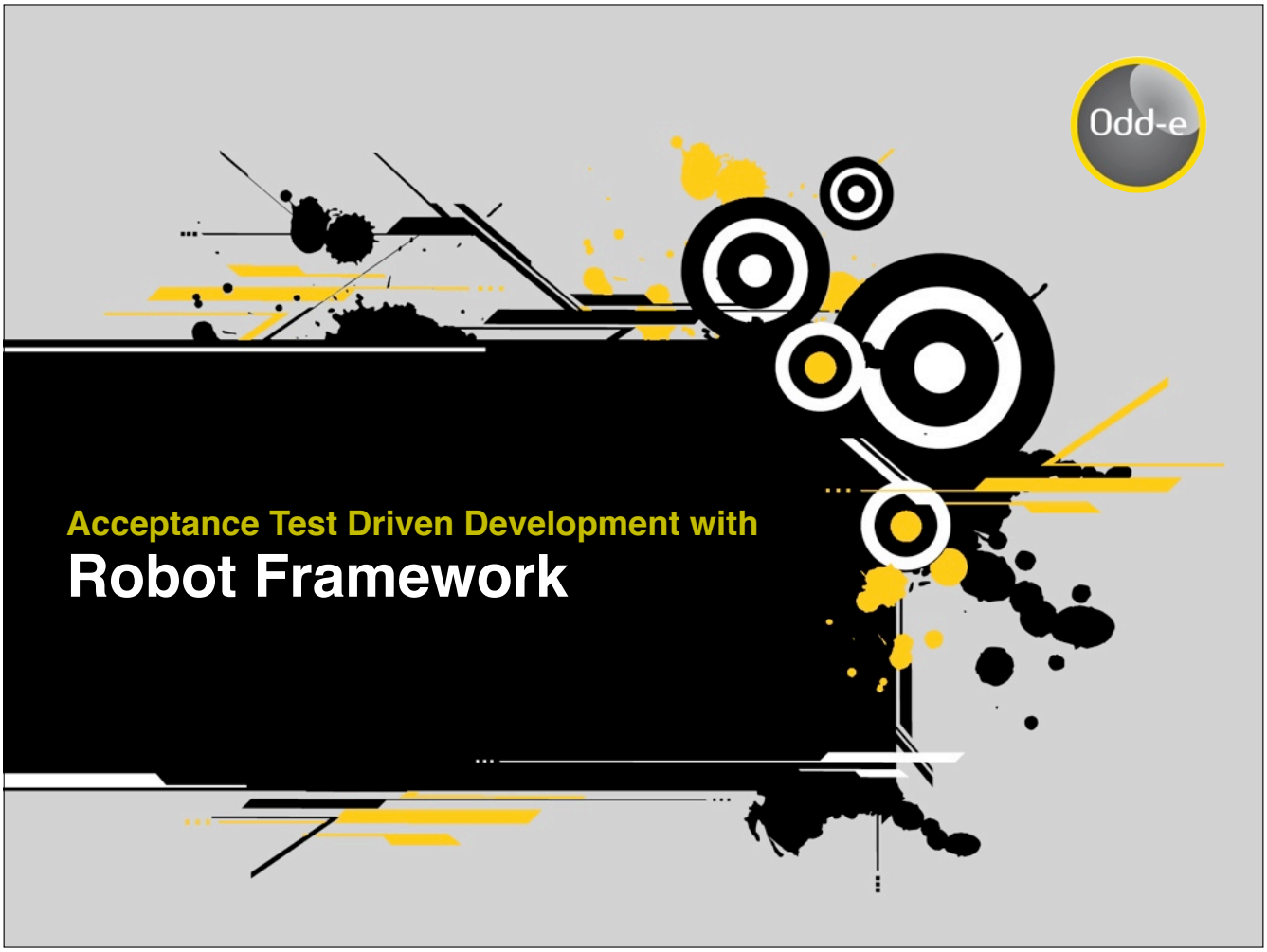
Data-driven test cases

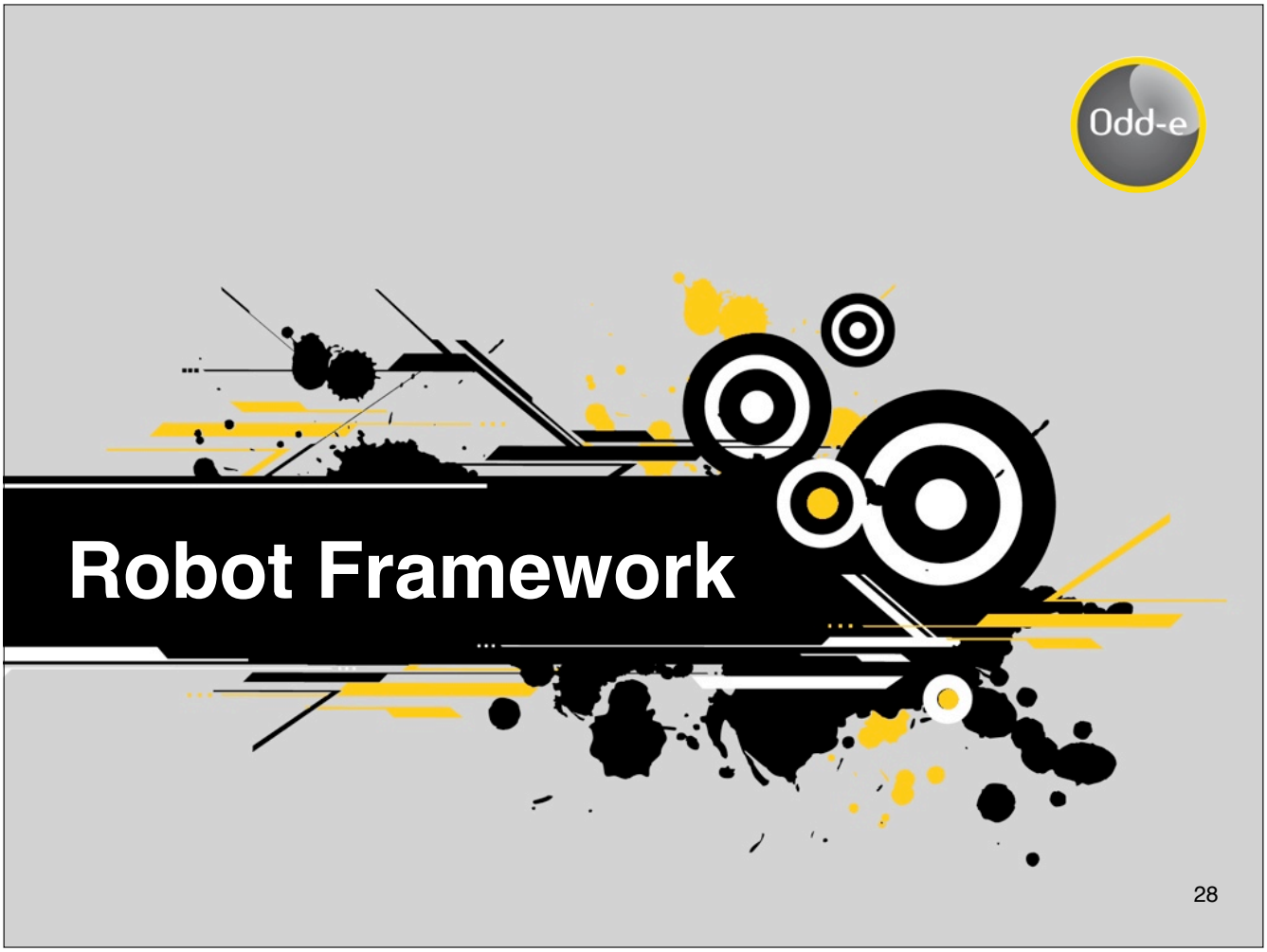
this is the name of a *test case*

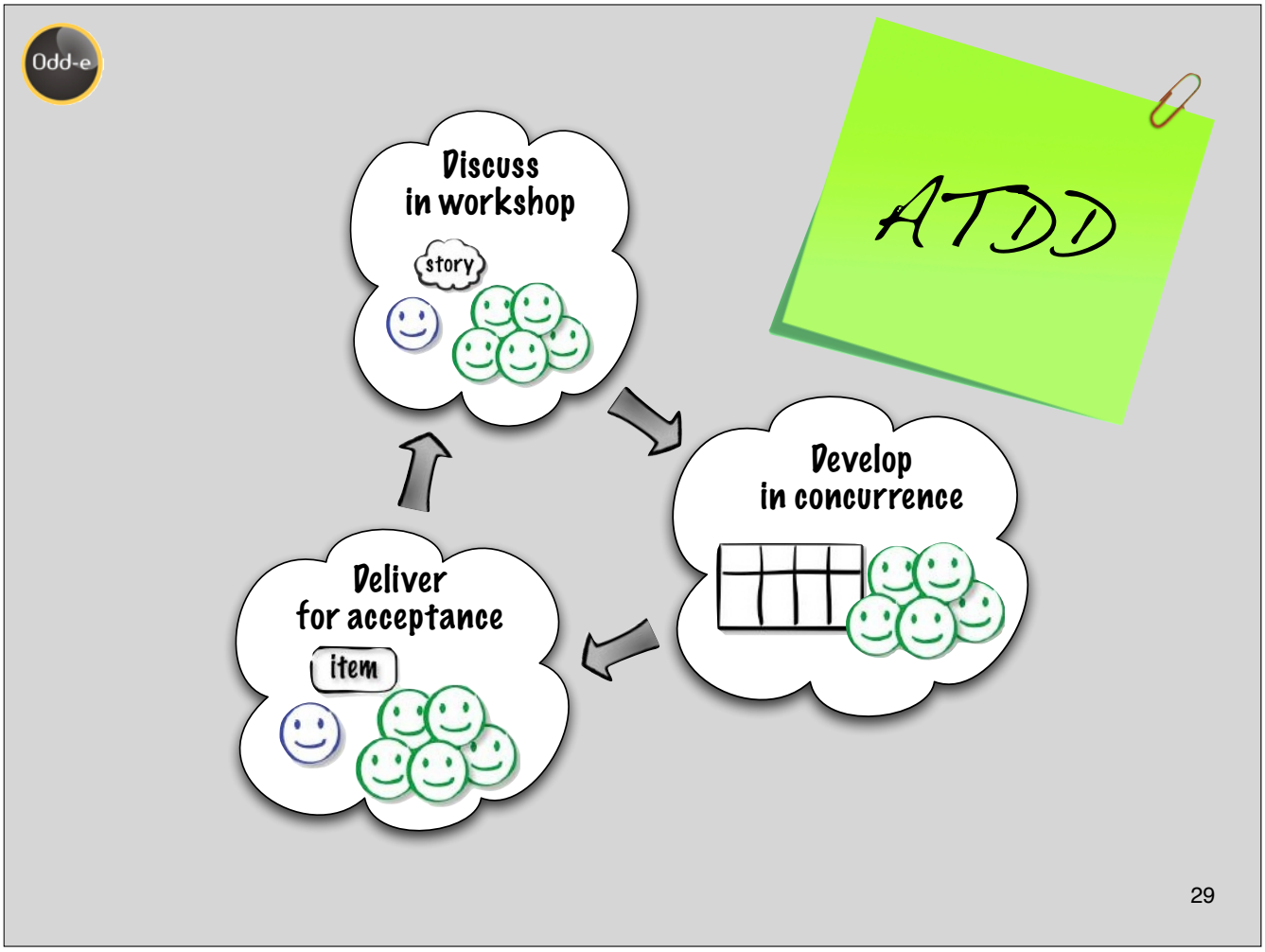
these *keywords* form the test case

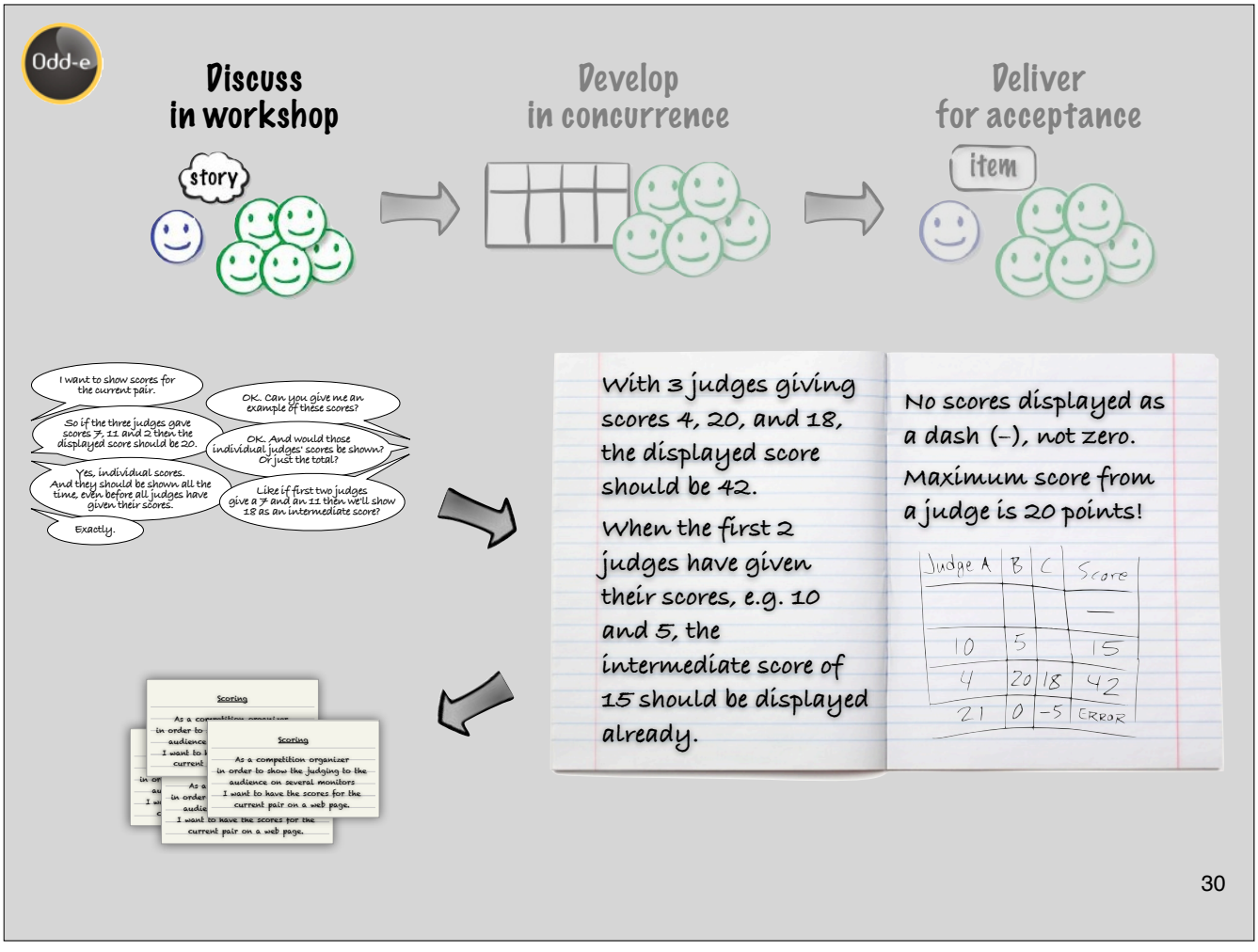
Test Case		Password	Expected error message
Too short password	Creating user with invalid password should fail	abCD5	${PWD INVALID LENGTH}
Too long password	Creating user with invalid password should fail	abCD567890123	${PWD INVALID LENGTH}
Password without lowercase letters	Creating user with invalid password should fail	123DEFG	${PWD INVALID CONTENT}
Password without capital letters	Creating user with invalid password should fail	abcd56789	${PWD INVALID CONTENT}
Password without numbers	Creating user with invalid password should fail	AbCdEfGh	${PWD INVALID CONTENT}
Password with special characters	Creating user with invalid password should fail	abCD56+	${PWD INVALID CONTENT}

keywords receive *arguments*


Odd-e
Acceptance Test Driven Development with
Robot Framework


Odd-e
Robot Framework

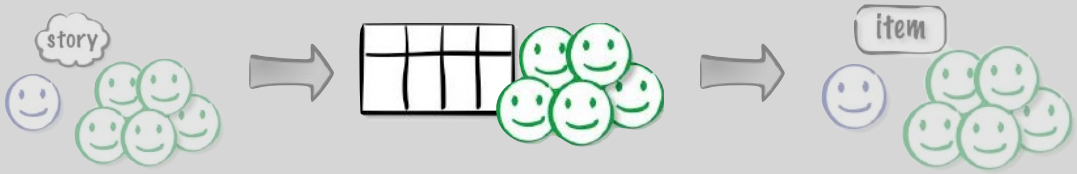

Odd-e
Discuss
in workshop
story
Develop
in concurrence
Deliver
for acceptance
item
ATDD


Odd-e
Discuss
in workshop
story
Develop
in concurrence
Deliver
for acceptance
item
I want to show scores for the current pair.
OK. Can you give me an example of these scores?
So if the three judges gave scores 7, 11 and 2 then the displayed score should be 20.
OK. And would those individual judges' scores be shown? Or just the total?
Yes, individual scores. And they should be shown all the time, even before all judges have given their scores.
Like if first two judges give a 7 and an 11 then we'll show 18 as an intermediate score?
Exactly.
With 3 judges giving scores 4, 20, and 18, the displayed score should be 42.
When the first 2 judges have given their scores, e.g. 10 and 5, the intermediate score of 15 should be displayed already.
No scores displayed as a dash (–), not zero.
Maximum score from a judge is 20 points!
Judge A | B | C | Score
—
10 | 5 | 15
4 | 20 | 18 | 42
21 | 0 | -5 | ERROR
Scoring
As a competition organizer
in order to show the judging to the audience on several monitors
I want to have the scores for the current pair on a web page.

Discuss in workshop

Develop in concurrence

Deliver for acceptance



With 3 judges giving scores 4, 20, and 18, the displayed score should be 42.

When the first 2 judges have given their scores, e.g. 10 and 5, the intermediate score of 15 should be displayed already.

Robot tests are written in tables so that computers can read them

Setting	Value	Value
Force Tags	regression	
Suite Setup	Open Login Page	
Suite Teardown	Close Browser	
Test Teardown	Go To Login Page	
Resource	resource.html	

Variable	Value	Value

Test Case	Action	Argument	A
Invalid Username	Login With Invalid Credentials Should Fail	invalid	mode
Invalid Password	Login With Invalid Credentials Should Fail	demo	invalid
Invalid Username And Password	Login With Invalid Credentials Should Fail	invalid	invalid
Empty Username	Login With Invalid Credentials Should Fail		invalid
Empty Password	Login With Invalid Credentials Should Fail	demo	\
Empty Username And Password	Login With Invalid Credentials Should Fail		\

Setting	Value	Value	Value
Force Tags	regression	smoke	
Test Setup	Open Login Page		
Test Teardown	Close Browser		
Resource	resource.html		

Variable	Value	Value	Value

Test Case	Action	Argument	Argume
Higher Level Valid Login	Input Username	demo	
	Input Password	mode	
	Click Login Button		
	Welcome Page Should Be Open		
Even Higher Level Valid Login	Login With Valid Credentials	demo	mode
Highest Level Login	Login As A Valid User		

Keyword	Action	Argument	Argume

It's all in the tables

Test Case	Step
Valid Login	Given login page is open
	When valid username and password are inserted
	and credentials are submitted
	Then welcome page should be open

Robot Architecture

Test Data (Tables)

Robot Framework

Test Library API

Test Libraries

Robot comes with a number of built-in test libraries and you can (should!) add your own.

Test Tools

Test libraries can use any test tool necessary to interact with the system under test.

application interfaces

System Under Test

Test Cases are composed of keyword-driven actions

this is the name of a *test case*

these *keywords* form the test case

Test Case	Action	Argument	Argument
User can create an account and log in	Create Valid User	fred	P4ssw0rd
	Attempt to Login with Credentials	fred	P4ssw0rd
	Status Should Be	Logged In	
User cannot log in with bad password	Create Valid User	betty	P4ssw0rd
	Attempt to Login with Credentials	betty	wrong
	Status Should Be	Access Denied	

keywords receive *arguments*

2 types of keywords

we can import keyword libraries for a *test case*

...and libraries may be configured, too.

Setting	Value	Value	Value
Library	Telnet	prompt=$	

This keyword comes from the imported library.

Test Case	Action	Argument	Argument	Argument
Example	Open connection	10.0.0.42	port=${25}	
	List files	options=-lh		
	List files	path=/tmp	options=-l	

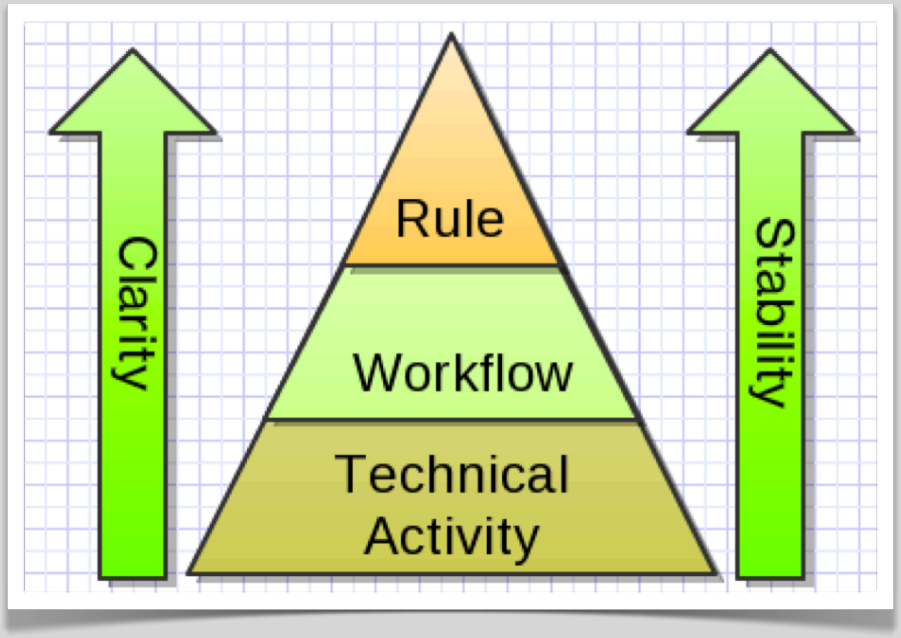
Keyword	Action	Argument	Argument	Argument
List files	[Arguments]	${path}=.	${options}=	
	Execute command	ls ${options} ${path}		

This is a *user keyword*, implemented in table format.
(Think macros composed of other macros.)

Variables

Variable	Value
${USERNAME}	janedoe
${PASSWORD}	J4n3D0e
${NEW PASSWORD}	e0D3n4J
${DATABASE FILE}	${TEMPDIR}${/}robotframework-quickstart-db.txt
${PWD INVALID LENGTH}	Password must be 7-12 characters long
${PWD INVALID CONTENT}	Password must be a combination of lowercase and uppercase letters and numbers

3 levels of automation



Principle of Symmetric Change

The best software is the one where the technical software model is aligned with the relevant business domain model. This ensures that one small change in the business domain (new requirements or changes to existing features) results in one small change in software. The same is true for the other artifacts produced for software — tests and documentation. When the tests are aligned with the business domain model, one small change in business will result in one small change in tests, making the tests easy to maintain.

Ref: The Secret Ninja Cucumber Scrolls

Odd-e

Syntax Alternatives Galore

robot_test_case.html

```
<table class="example docutils" border="1">
    <thead valign="bot
        <tr>
            <th class=
                Settir
            </th>
            <th class=
```

robot_test_case.psv

```
| *Setting*  |     *Value*     |
| Library    | OperatingSystem |

| *Variable* |     *Value*     |
| ${MESSAGE} | Hello, world!   |

| *Test Case*  | *Action*         | *Argument*   |
| My Test      | [Documentation]  | Example test |
|              | Log              | ${MESSAGE}   |
|              | My Keyword       | /tmp         |
| Another Test | Should Be Equal  | ${MESSAGE}   | Hello,
world!

| *Keyword*  |
| My Keyword | [Arguments] | ${path}
|            | Directory Should Exist | ${path}
```

robot_test_

```
***Settings***
Library     OperatingSystem

***Variables***
${MESSAGE}  Hell

***Test Cases***
My Test  [Docume
    Log
    My Keyword

Another Test
    Should Be Equ

***Keywords***
My Keyword  [Arg
    Directory Sh
```

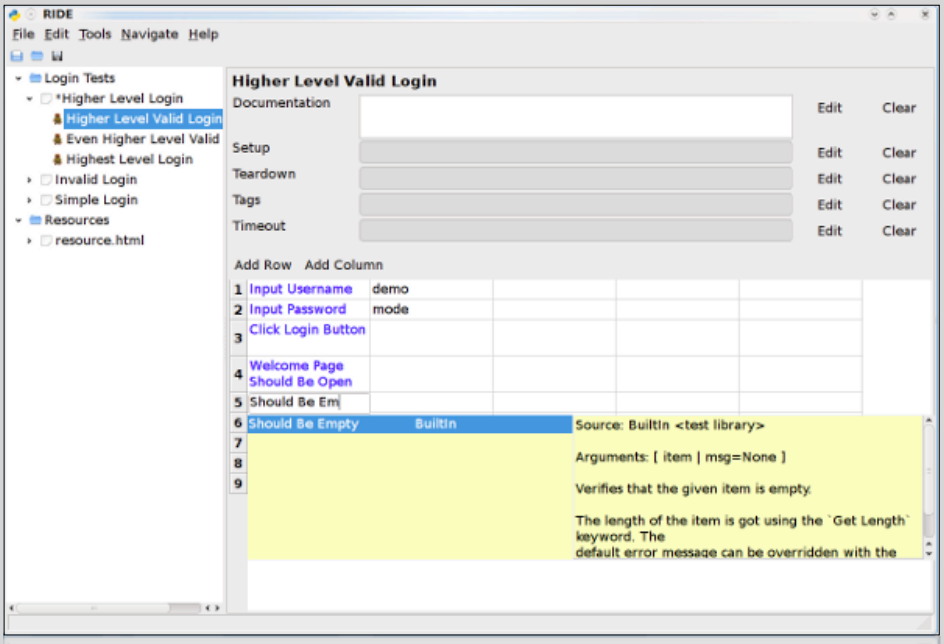
```
*Setting*
Library Oper

*Variable*
${MESSAGE}

*Test Case*
My Test [Doc
    Log ${ME
    My Keywo

Another Test

*Keyword*
My Keyword
    Directory Should Exist  ${pat
```

```
 Variable
==========
${MESSAGE}
==========


==========
 Test Case
==========
My Test
\
\
\
Another Test
==========
```

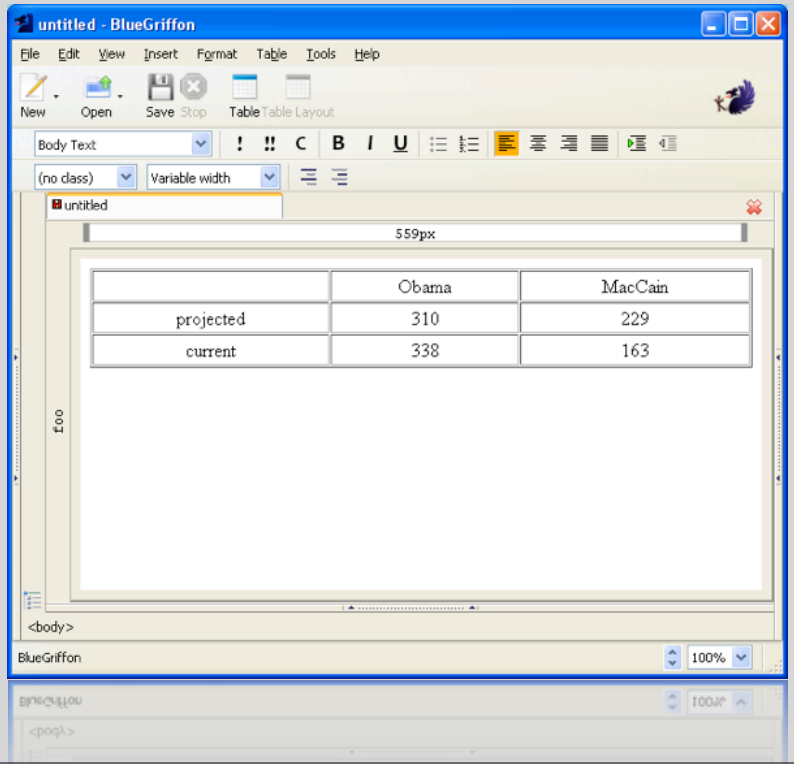
```
==========  ====================  ==========  ==========
 Keyword          Action           Argument    Argument
==========  ====================  ==========  ==========
My Keyword  [Arguments]           ${path}
\           Directory Should Exist ${path}
==========  ====================  ==========  ==========
```

Tools

RIDE


RIDE
File Edit Tools Navigate Help
Login Tests
*Higher Level Login
Higher Level Valid Login
Even Higher Level Valid
Highest Level Login
Invalid Login
Simple Login
Resources
resource.html
Higher Level Valid Login
Documentation
Edit
Clear
Setup
Teardown
Tags
Timeout
Add Row
Add Column
1 Input Username demo
2 Input Password mode
3 Click Login Button
4 Welcome Page Should Be Open
5 Should Be Em
6 Should Be Empty BuiltIn
Source: BuiltIn <test library>
Arguments: [item | msg=None]
Verifies that the given item is empty.
The length of the item is got using the `Get Length` keyword. The
default error message can be overridden with the

BlueGriffon



Built by:


```
<target name="robot" depends="init">
    <exec executable="jybot">
        <env key="CLASSPATH" value="${robot.test.classpath}"/>
        <arg line="-C off"/>
        <arg value="-d"/>
        <arg path="target/robot"/>
        <arg value="-v"/>
        <arg value="jetty.port:${jetty.port}"/>
        <arg path="src/test/acceptance/my_test_case.html"/>
    </exec>
</target>
```

Built by:

maven robotframework-maven-plugin

```
<plugin>
    <groupId>com.googlecode.robotframework-maven-plugin</groupId>
    <artifactId>robotframework-maven-plugin</artifactId>
    <version>1.1.1</version>
    <executions>
        <execution>
            <goals>
                <goal>run</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

http://code.google.com/p/robotframework-maven-plugin/

Robot Architecture

Test Data (Tables)

Robot Framework

Test Library API

Test Libraries

Robot comes with a number of built-in test libraries and you can (should!) add your own.

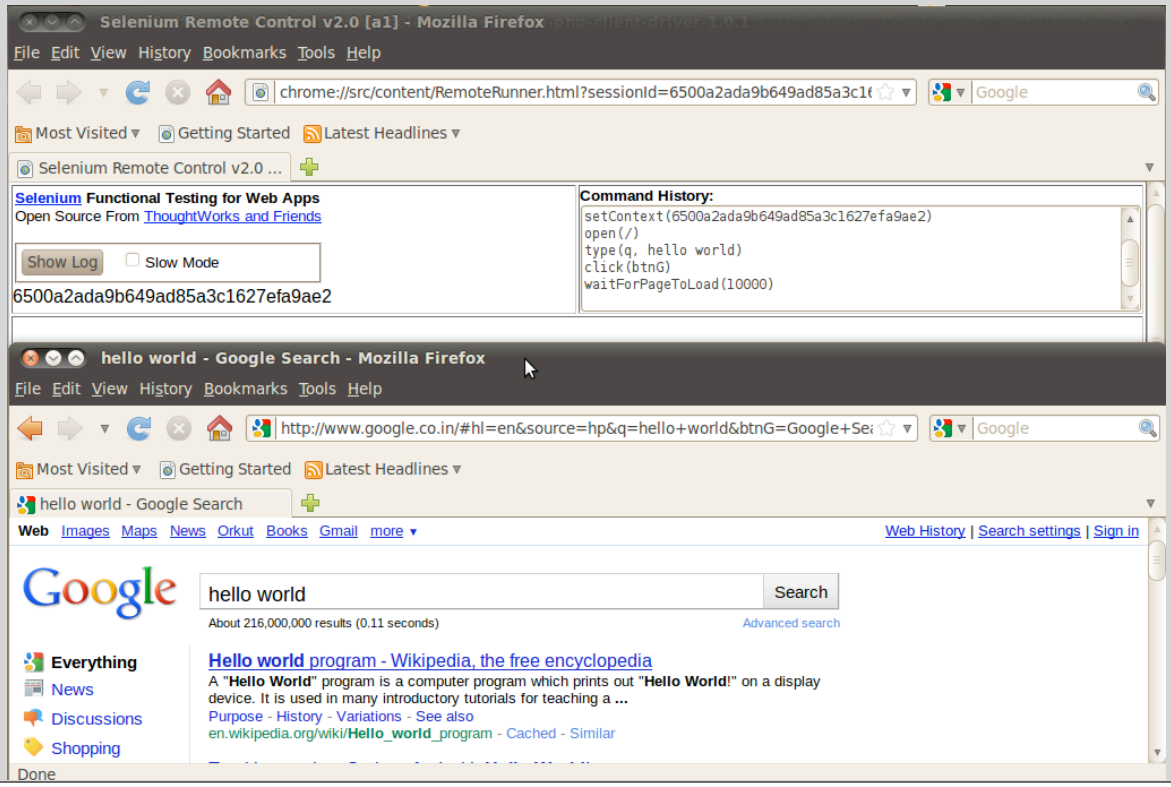
Test Tools

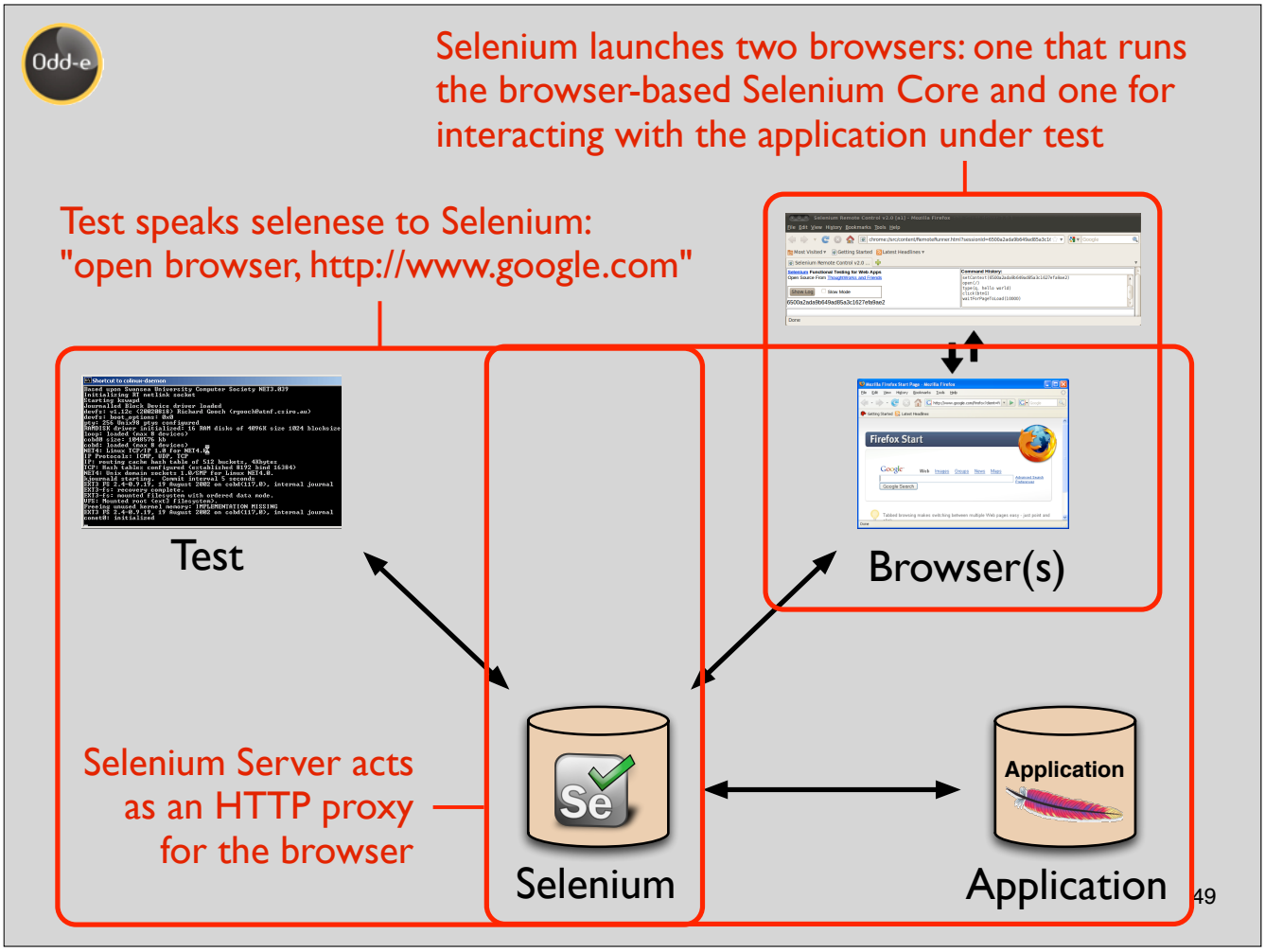
Test libraries can use any test tool necessary to interact with the system under test.

application interfaces

System Under Test

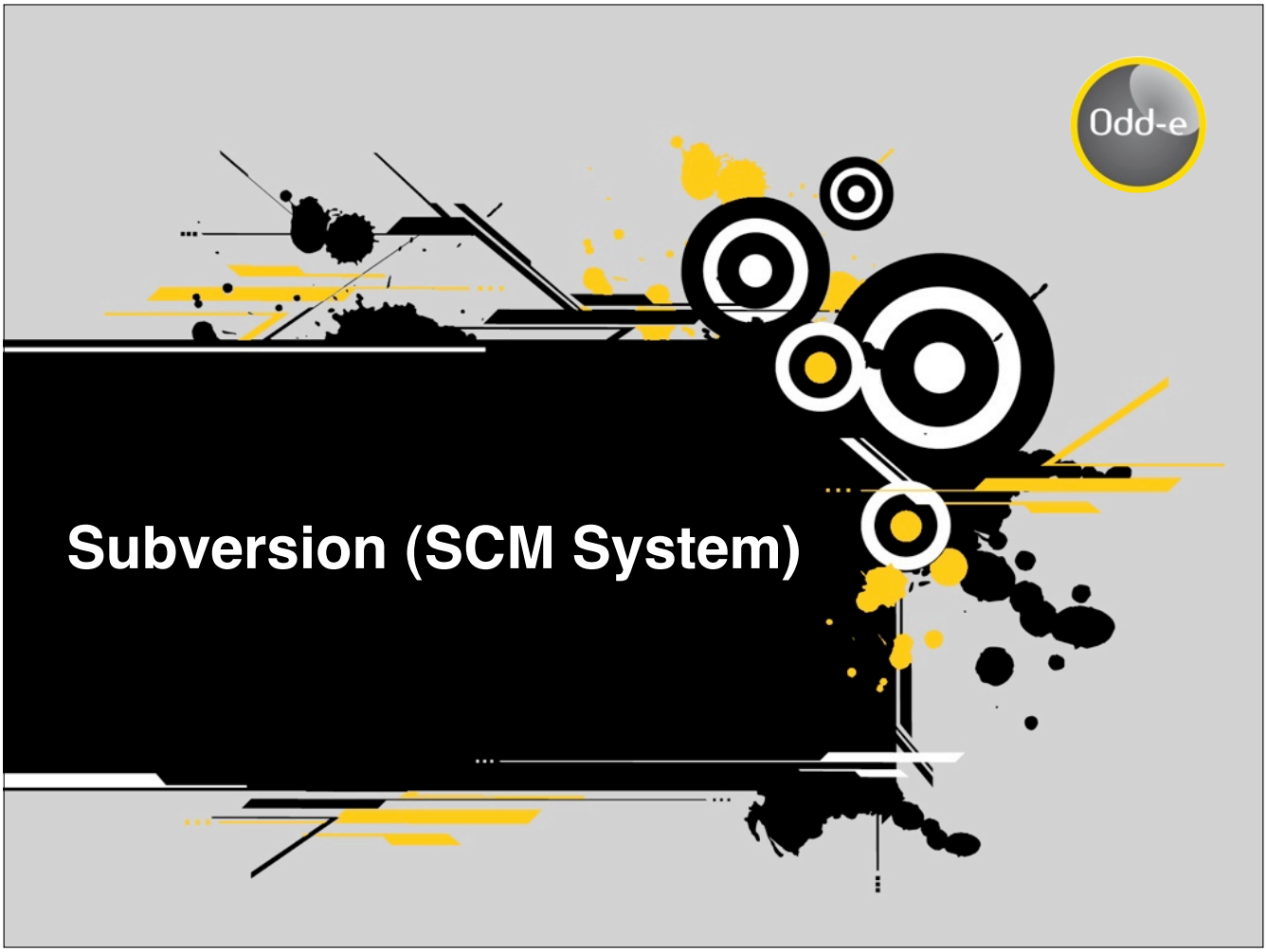
Selenium





Close All Browsers · Close Browser · Close Window · Confirm Action · Current Frame
Current Frame Should Contain · Delete All Cookies · Delete Cookie · Drag And Drop ·
Element Should Be Visible · Element Should Contain · Element Should Not Be Visible · E
Execute Javascript · Focus · Frame Should Contain · Frame Should Contain Text · Get
Get All Links · Get Cookie Value · Get Cookies · Get Element Attribute · Get Horizonta
Get List Items · Get Location · Get Matching Xpath Count · Get Source · Get Table Cell
· Get Value · Get Vertical Position · Get Window Identifiers · Get Window Names · Get
Go Back · Go To · Input Password · Input Text · List Selection Should Be · List Should
Location Should Be · Location Should Contain · Log Source · Maximize Browser Windo
Mouse Down On Image · ouse Over · Mouse Up
Open Context Menu · Button · Page Should
Page Should Contain E Should Contain Link
Page Should Contain L age Should Contain Tex
Page Should Not Conta Should Not Contain C
Page Should Not Contain Element · Page Should Not Contain Image · Page Should Not Co
Page Should Not Contain List · Page Should Not Contain Radio Button · Page Should Not
Press Key · Press Key Native · Radio Button Should Be Set To · Radio Button Should Not
Register Keyword To Run On Failure · Select All From List · Select Checkbox · Select Fr
· Select Radio Button · Select Window · Set Selenium Speed · Set Selenium Timeout · Si
Start Selenium Server · Stop Selenium Server · Submit Form · Switch Browser · Table C
Table Column Should Contain · Table Footer Should Contain · Table Header Should Cont
Table Row Should Contain · Table Should Contain · Textfield Should Contain · Textfield
Title Should Be · Unselect Checkbox · Unselect Frame · Unselect From List · Wait For C
Wait Until Page Contains · Wait Until Page Contains Element · Wait Until Page Loaded

Robot Framework hides (most of) this infrastructure behind simple keywords

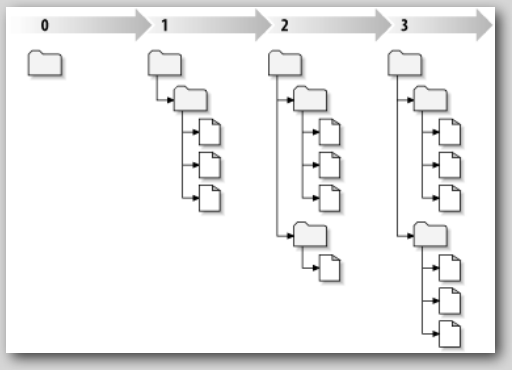


Odd-e

Software Configuration Management Systems

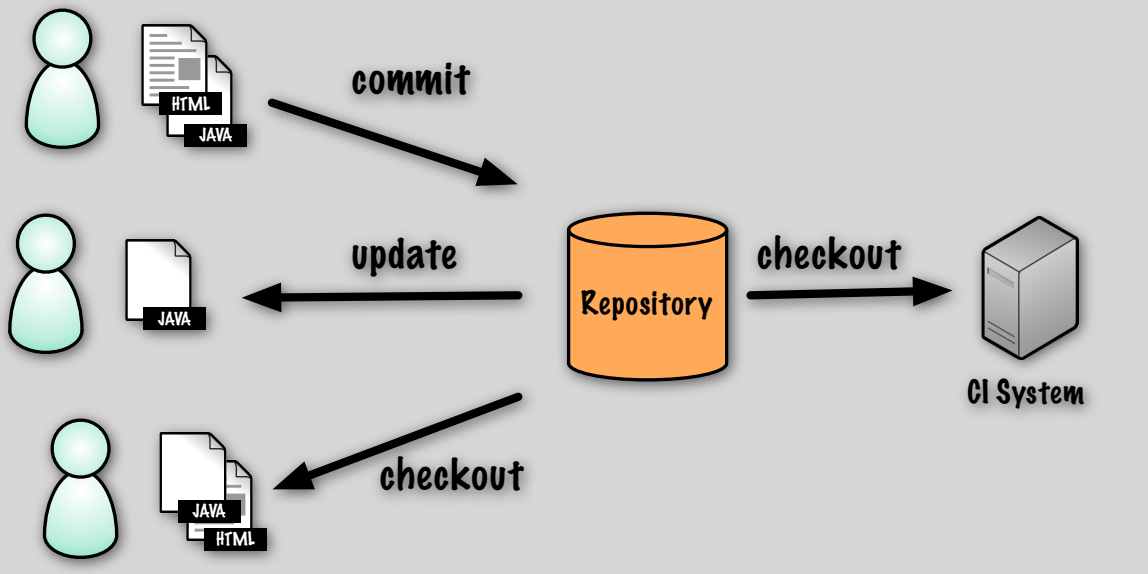
Concept

- Records the evolution of changes in your project
- Govern how your team collaborates



Benefits

- Reproduce your project's state at any time in the past
- Easy for all team members to share source code
- Know who and when the change is made



SVN Quick Guide

Update working copy `svn update`

Commit changes `svn commit -m "Handled parsing error"`

Display commit messages `svn log Foo.java`

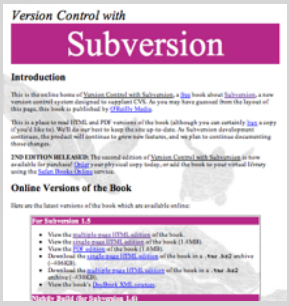
Checkout project into local dir. `svn co http://company/repos/mailer`

Undo local edits `svn revert Foo.java`

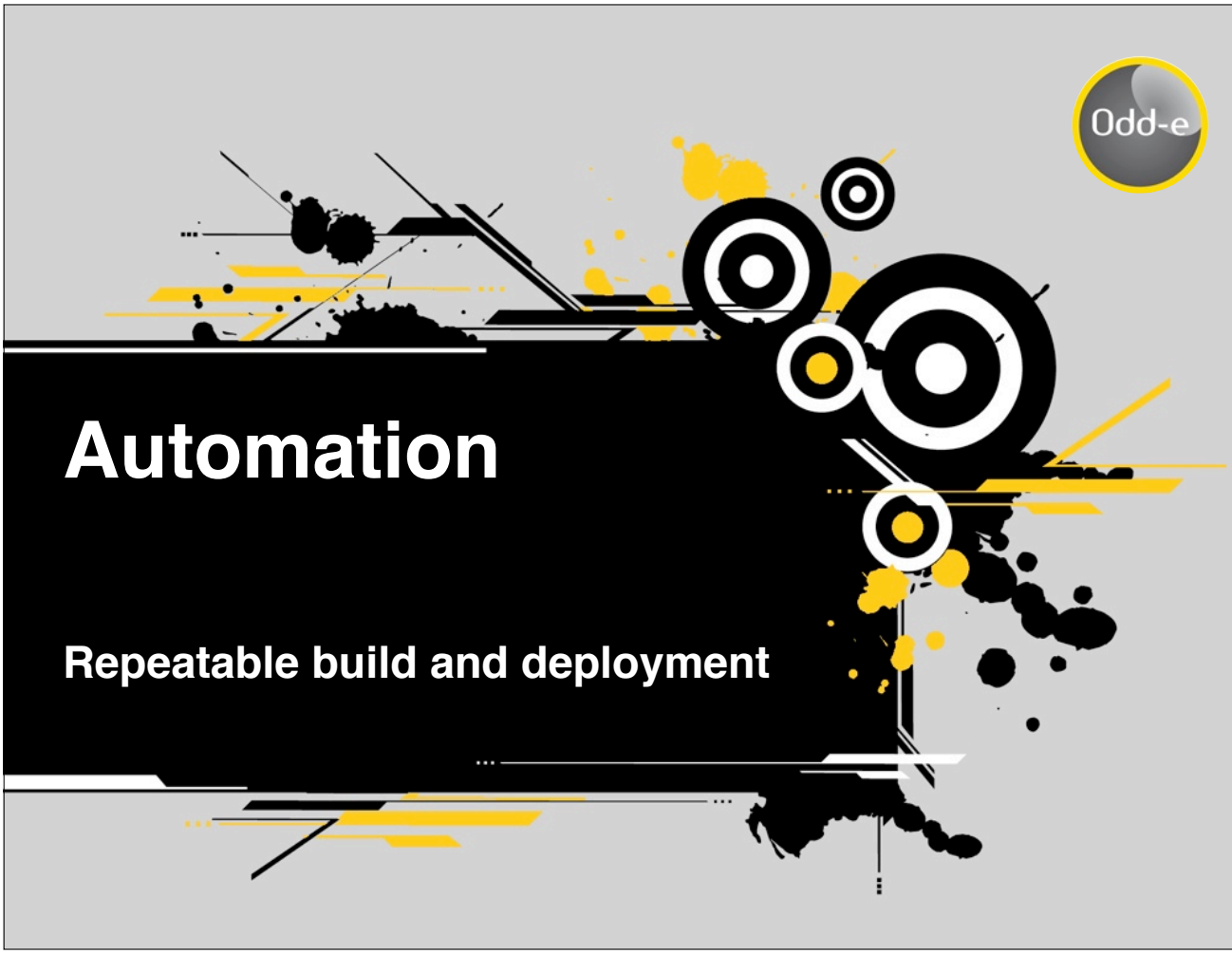
Create repository `svnadmin create /var/svn/repos`

Some Steps You Might Need

- Create a Subversion repository for the team
- Import your project skeleton into it
- Check out a working copy for everybody


Version Control with
Subversion
Introduction
Online Versions of the Book

For more details, http://svnbook.red-bean.com



Automation

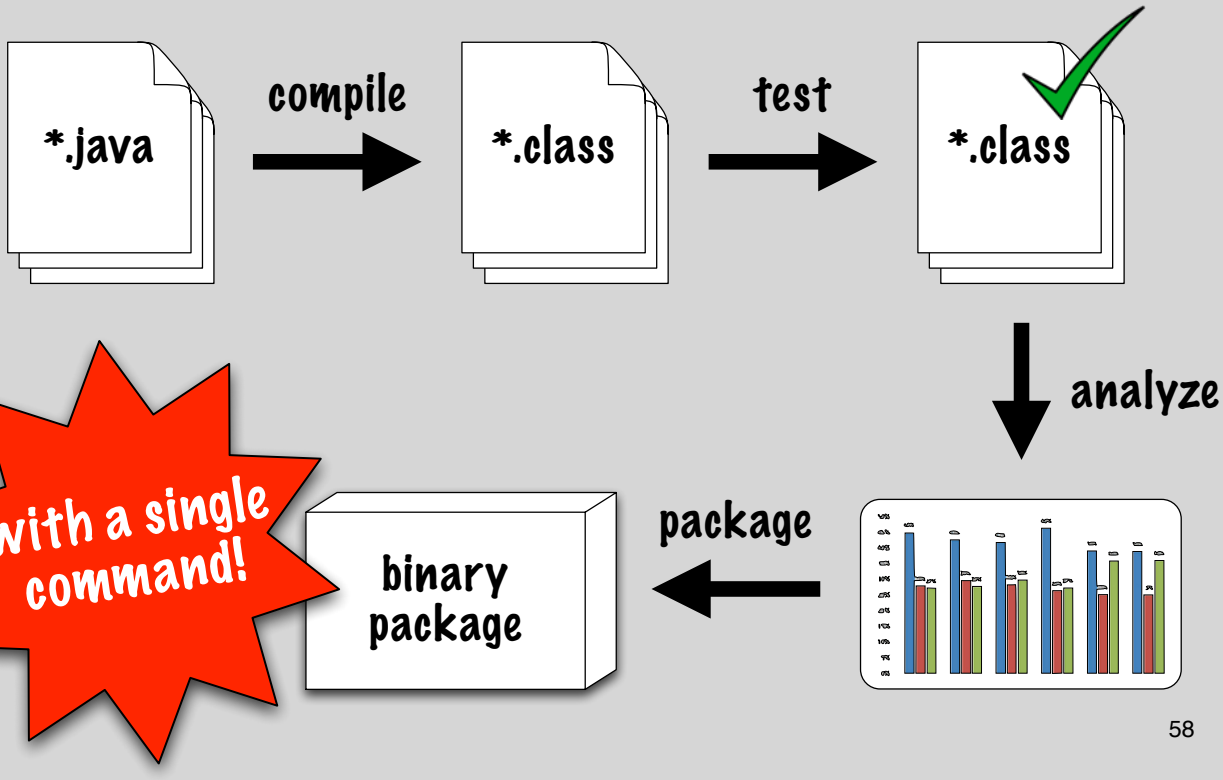
Repeatable build and deployment

Odd-e

Automated Build

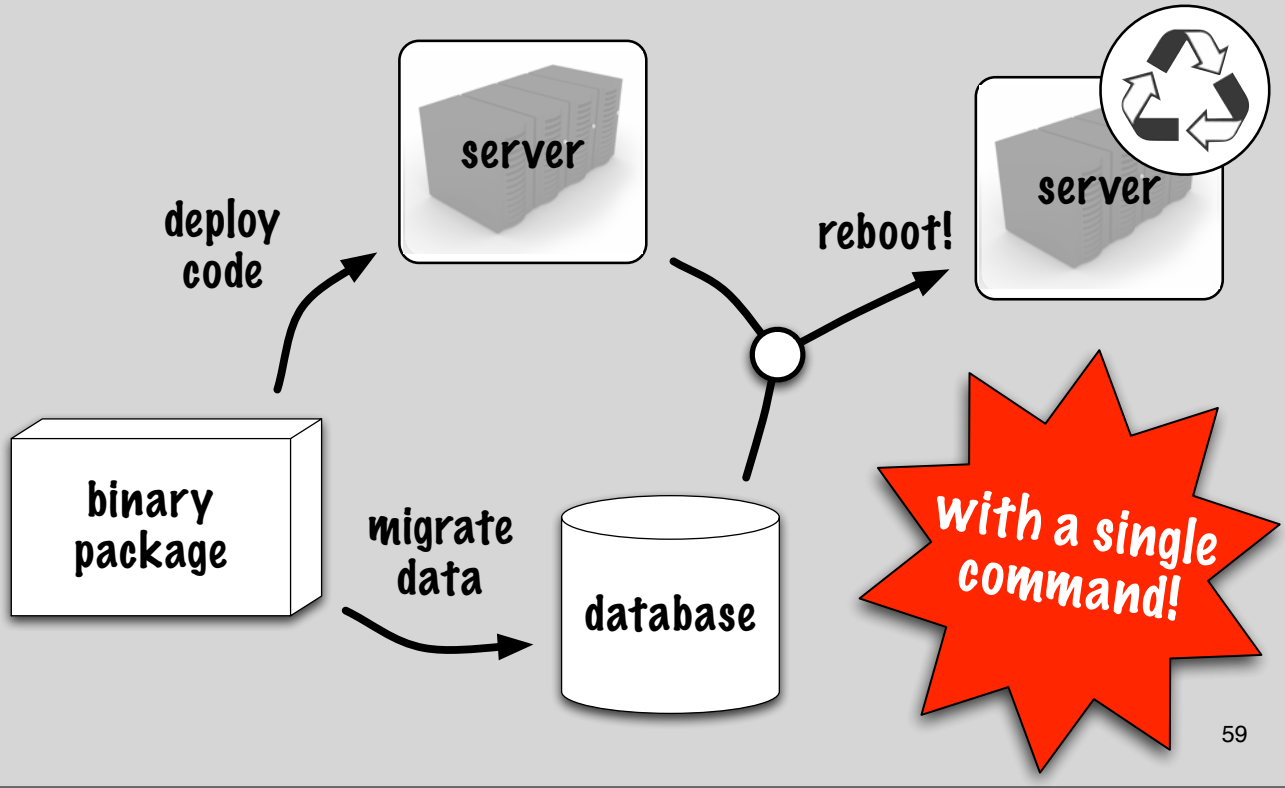
Odd-e

Automated Build



Odd-e

Automated Deployment



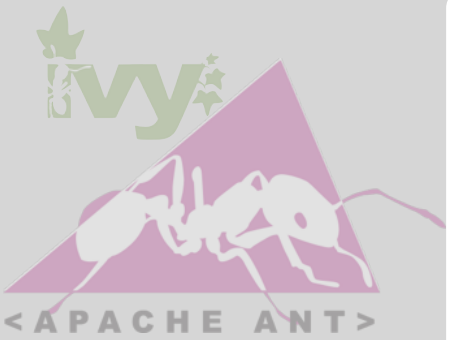
Odd-e

Tools

maven

buildr

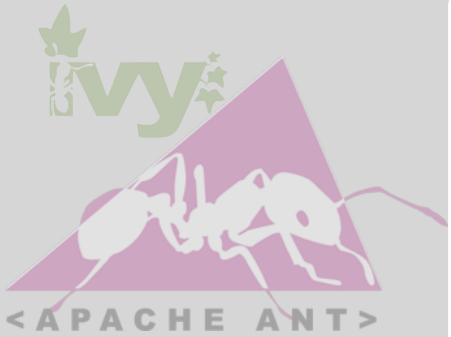
- First widely adopted build tool specially made for Java projects
- Build file written in XML
- Imperative programming
- Simple and verbose
  - You get *nothing* by default
- Dependency management through Apache Ivy
- Lots of 3rd party plugins



maven

buildr

- First build tool to incorporate built-in support for dependency management and a standard artifact repositories
- Build file written in XML
- Declarative programming
- Convention over configuration
- Lots of 3rd party plugins



maven

buildr

- First no-XML build tool for Java to gain a meaningful user base
- Build file written in Ruby
- Imperative programming
- Exploits the Maven ecosystem:
  - Convention over configuration
  - Dependency management
- Not so many 3rd party plugins

Samples

build.xml

```
<?xml version="1.0"?>
<project xmlns:ivy="antlib:org.apache.ivy.ant" name="KillerApp" default="package" basedir=".">
    <path id="build.classpath">
        <fileset dir="lib/build" includes="**/*.jar"/>
    </path>
    <target name="init" depends="ivy-dependencies">
        <mkdir dir="target/classes/production"/>
        <mkdir dir="target/classes/test"/>
    </target>
    <target name="ivy-dependencies">
        <taskdef resource="org/apache/ivy/ant/antlib.xml" uri="antlib:
        <ivy:settings file="ivysettings.xml"/>
        <ivy:retrieve/>
        <ivy:cachepath pathid="src.classpath" conf="production"/>
        <ivy:cachepath pathid="test.classpath" conf="development"/>
    </target>
    <target name="compile" depends="init" description="compile the sour
        <javac srcdir="src" destdir="target/classes/production" classp
        <javac srcdir="test" destdir="target/classes/test" classpathre
    </target>
    <target name="test" depends="compile" description="run unit tests">
        <mkdir dir="target/reports"/>
        <junit printsummary="yes" haltonfailure="yes" showoutput="yes">
            <classpath>
                <path refid="test.classpath"/>
                <pathelement location="target/classes/test"/>
            </classpath>
            <batchtest fork="yes" todir="target/reports">
                <formatter type="xml"/>
                <fileset dir="test" includes="**/*Test.java,**/Test*.j
            </batchtest>
        </junit>
    </target>
    <target name="package" depends="compile, test" description="generat
        <mkdir dir="target/package"/>
        <jar jarfile="target/package/killerapp.jar" basedir="target/cl
    </target>
    <target name="clean" description="clean up">
        <delete dir="target"/>
    </target>
</project>
```

```
$ ant clean package
Buildfile: /Users/lkoskela/Desktop/CSD/Automation/ant/build.xml

clean:
   [delete] Deleting directory /Users/lkoskela/Desktop/CSD/Automation/ant/target

ivy-dependencies:
[ivy:retrieve] :: Ivy 2.2.0 - 20100923230623 :: http://ant.apache.org/ivy/ ::
[ivy:retrieve] :: loading settings :: file = /Users/lkoskela/Desktop/CSD/Automation/ant/
ivysettings.xml
[ivy:retrieve] :: resolving dependencies :: com.odde#killerapp;working@1162.local
[ivy:retrieve]      confs: [production, development]
[ivy:retrieve]      found junit#junit;4.8.2 in ibiblio
[ivy:retrieve] :: resolution report :: resolve 187ms :: artifacts dl 4ms
          ---------------------------------------------------------------------
          |                  |            modules            ||   artifacts   |
          |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
          ---------------------------------------------------------------------
          |  production      |   0   |   0   |   0   |   0   ||   0   |   0   |
          |  development     |   1   |   0   |   0   |   0   ||   1   |   0   |
          ---------------------------------------------------------------------
[ivy:retrieve] :: retrieving :: com.odde#killerapp
[ivy:retrieve]      confs: [production, development]
[ivy:retrieve]      0 artifacts copied, 1 already retrieved (0kB/7ms)

init:
   [mkdir] Created dir: /Users/lkoskela/Desktop/CSD/Automation/ant/target/classes/production
   [mkdir] Created dir: /Users/lkoskela/Desktop/CSD/Automation/ant/target/classes/test

compile:
   [javac] Compiling 1 source file to /Users/lkoskela/Desktop/CSD/Automation/ant/target/classes/
production
   [javac] Compiling 1 source file to /Users/lkoskela/Desktop/CSD/Automation/ant/target/classes/
test

test:
   [mkdir] Created dir: /Users/lkoskela/Desktop/CSD/Automation/ant/target/reports
   [junit] Running com.odde.build.SampleTest
   [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.084 sec

package:
   [mkdir] Created dir: /Users/lkoskela/Desktop/CSD/Automation/ant/target/package
     [jar] Building jar: /Users/lkoskela/Desktop/CSD/Automation/ant/target/package/killerapp.jar

BUILD SUCCESSFUL
Total time: 4 seconds
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.odde.build</groupId>
 <artifactId>webapp</artifactId>
 <packaging>war</packaging>
 <version>1.2</version>
 <name>Killer App</name>
 <url>http://www.killerapp.com</url>

 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.8.2</version>
   <scope>test</scope>
  </dependency>
  <dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>servlet-api</artifactId>
   <version>2.4</version>
  </dependency>
 </dependencies>

 <build>
  <finalName>killer</finalName>
 </build>
</project>
```

```
$ mvn clean package
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building Killer App
[INFO]   task-segment: [clean, package]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] ...
[INFO] [compiler:compile {execution: default-compile}]
[INFO] ...
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] ...
[INFO] [surefire:test {execution: default-test}]
[INFO] ...
[INFO] [war:war {execution: default-war}]
[INFO] ...
maven2/src/main/webapp]
[INFO] Webapp assembled in[82 msecs]
[INFO] Building war: ./target/killer.war
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3 seconds
[INFO] Finished at: Wed Nov 24 11:30:26 SGT 2010
[INFO] Final Memory: 16M/81M
[INFO] ------------------------------------------------------------------------
$
```

Buildfile

```
VERSION_NUMBER = '1.2.0'

repositories.remote << 'http://repo1.maven.org/maven2/'

SERVLET_API = 'javax.servlet:servlet-api:jar:2.4'
LOG4J = 'log4j:log4j:jar:1.2.9'
JUNIT = transitive('junit:junit:jar:4.8.2')

define 'killerapp' do
 project.version = VERSION_NUMBER
 compile.with SERVLET_API, LOG4J
 test.with JUNIT, SERVLET_API, LOG4J
 package :jar

 task :run => :compile do
  system 'java -cp target/classes com.odde.build.Sample'
 end

 task :package => :compile do
  package(:war).libs += artifacts(LOG4J)
  package(:tgz).include _('src/main/migrations')
 end

 integration.setup do
  puts "start server"
  puts "deploy application"
 end

 integration.teardown do
  puts "stop server"
 end
end
```

```
$ buildr clean build
(in /Users/lkoskela/Desktop/CSD/Automation/buildr, development)
Cleaning killerapp
Building killerapp
Compiling killerapp into /Users/lkoskela/Desktop/CSD/Automation/buildr/target/
classes
Compiling killerapp:test into /Users/lkoskela/Desktop/CSD/Automation/buildr/
target/test/classes
Running tests in killerapp
Trying to override old definition of datatype junit
    [junit] Testsuite: com.odde.build.SampleTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.299 sec
    [junit]
    [junit] Testcase: test took 0.007 sec
Completed in 5.588s
$
```

More about Maven

Before Maven

build-main.bat

different

turbine

turbine-build.sh

Before Maven

what does the project need to build?

what libraries do I download?

where do I put them?

what should I execute on the build?

Consistency

1. checkout source
2. `mvn package`

dependency management and reuse of build logic

Maven Lifecycle

1. validate
2. compile
3. test
4. package
5. integration-test
6. verify
7. install
8. deploy

`mvn package`

will invoke validate, compile **and** test

Conventions

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.odde.build</groupId>
 <artifactId>webapp</artifactId>
 <packaging>war</packaging>
 <version>1.2</version>
 <name>Killer App</name>
 <url>http://www.killerapp.com</url>

 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.8.2</version>
   <scope>test</scope>
  </dependency>
  <dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>servlet-api</artifactId>
   <version>2.4</version>
  </dependency>
 </dependencies>

 <build>
  <finalName>killer</finalName>
 </build>
</project>
```

assumes

`src/test/java`

`src/main/java`

`src/main/webapp/WEB-INF/web.xml`

`target/killer.war`

Configuration & Repository

~/.m2/settings.xml

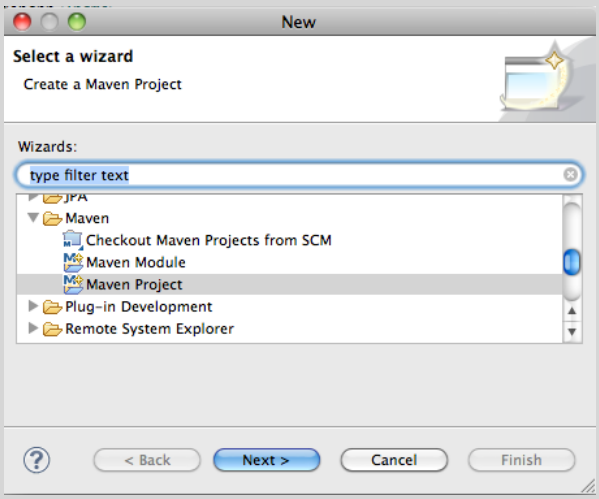
user-specific configuration for authentication, repositories and other Maven customization

~/.m2/repository/

your local Maven repository. Downloaded dependencies are stored here.

How to start?

Eclipse plugin, m2e



Command line

1. create pom.xml
2. `mvn eclipse:eclipse`
3. mvnrepository.com for dependencies

http://maven.apache.org/guides/mini/guide-ide-eclipse.html

Resources

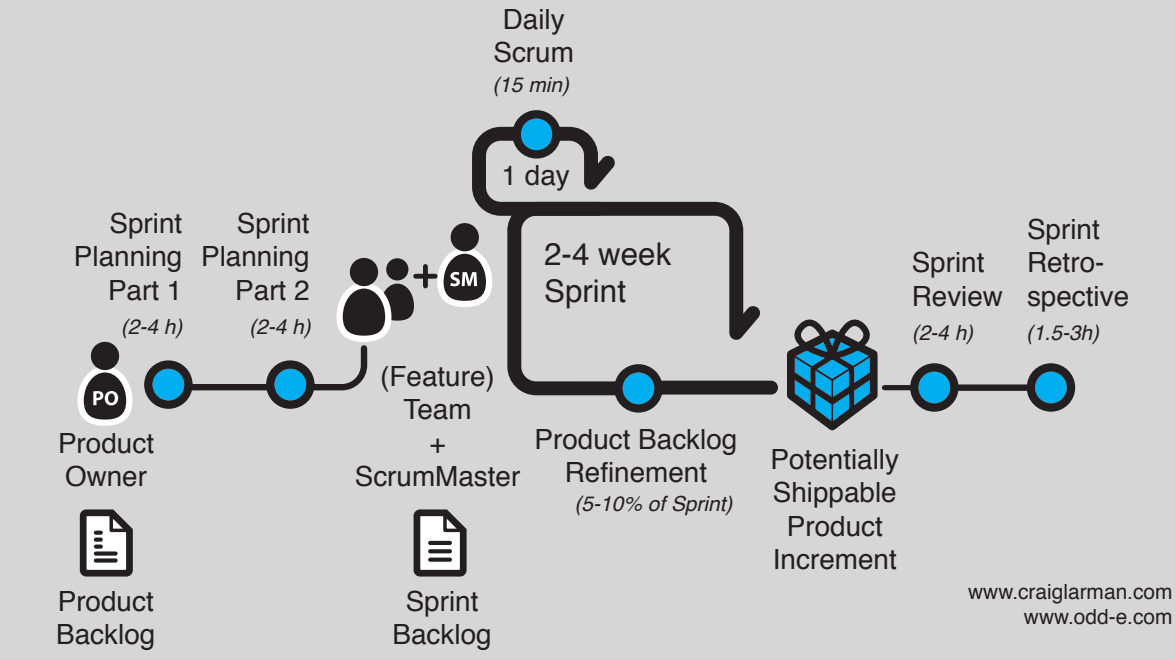
Archetypes

Multi-module

http://www.sonatype.com/books/mvnref-book


Odd-e
Sprint Planning


Odd-e
Sprint Planning Overview



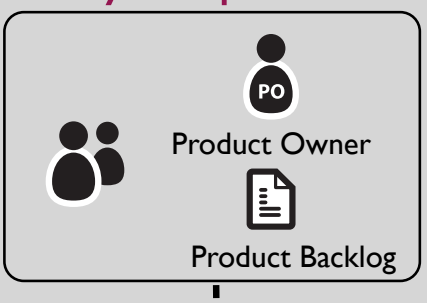
www.craiglarman.com
www.odd-e.com

Copyright © C.Larman &
B. Vodde 2009.
All rights reserved.

Sprint Planning

Clarify Requirements

Sprint Planning Part 1



2-4h timebox

input

- Team Capabilities
- Business Conditions
- Technology Stability
- Previous Sprint Product Increment

Selected Product Backlog Items

Initial Design and Plan

Sprint Planning Part 2

2-4h timebox

output

Updated Product Backlog

- Sprint Backlog
- Sprint Goal

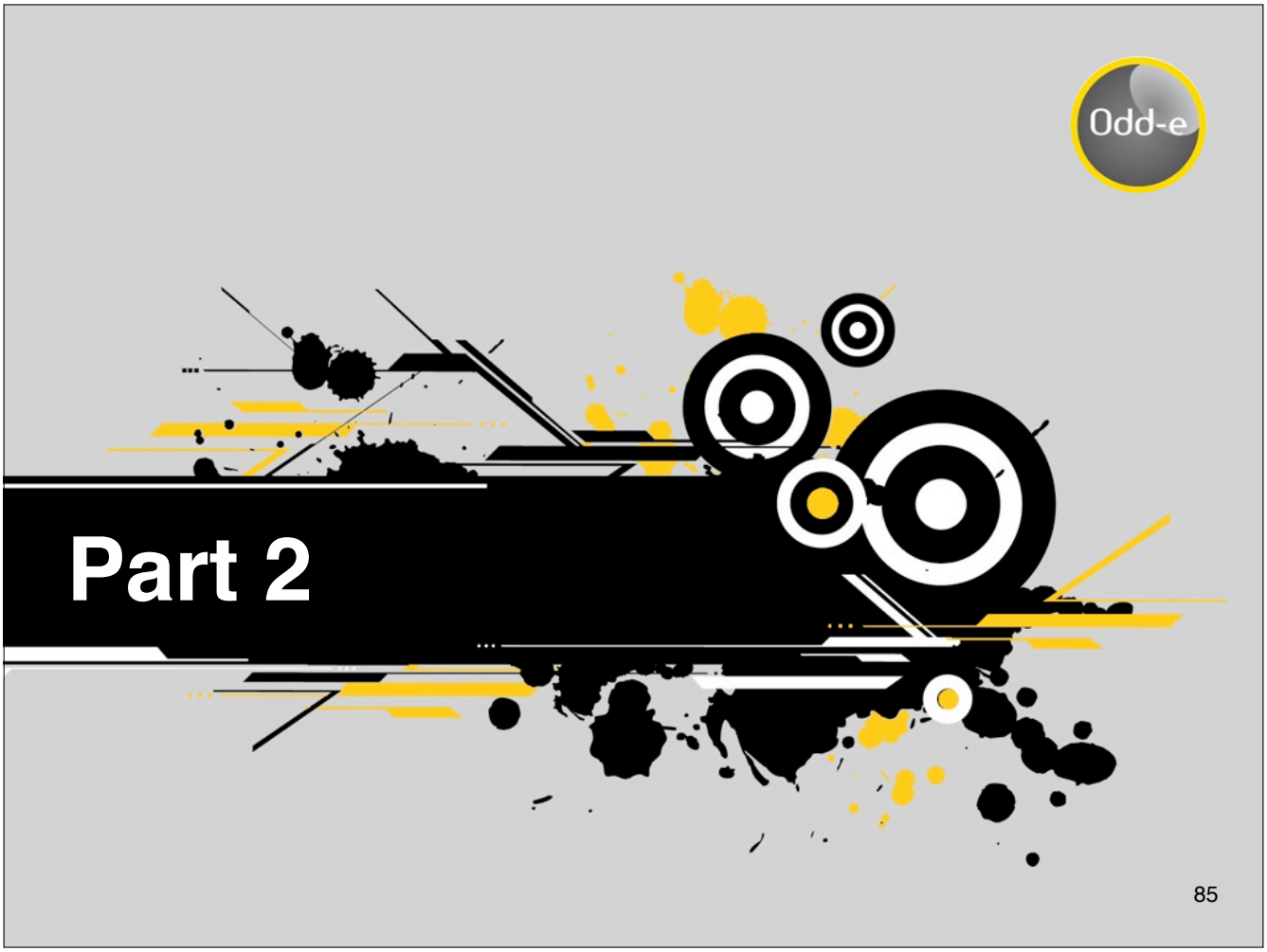
Sprint Commitment

- Made after Sprint Planning part 2
- Should be taken serious by the team
  - Creates trust between PO and team
  - If not taken seriously -> problems with the team
- Software development is new product development, which means that stuff comes up!
  - Even good teams might deliver on their sprint commitment perhaps 80% of the time.

Typical Part 1

- Agenda:
  - Introduce Product Backlog
  - Tentatively select backlog items
  - Clarify unclear items
- Tips:
  - Can use projector, but printed or wall backlogs work well
  - Should be short due to Backlog Refinement

Exercise:
Do Sprint Planning Part 1



Odd-e

Typical Agenda for Part 2

- Create initial design through a short Agile Modeling session
- Create the team's sprint calendar
- Estimate team's hour/capacity budget

- Until capacity is reached:
  - Pick an item from product backlog
  - Generate implementation tasks
  - Estimate the tasks

- Make an explicit sprint commitment – this is what we will deliver!

Meeting Tips for Part 2

- Sprint planning is time-boxed. Do not let it flow over.
- Avoid computers, projectors, screens – use inclusive media
  - A computer might be necessary for quick research or checking facts about the current code base
- Keep up the pace – try not to dwell on an individual item for too long
- Team owns the meeting – beware of a dominating Scrum Master
- Establish a collaborative environment
- The goal is a sufficient level of certainty for making a commitment
  - You will never uncover tasks *exhaustively*
  - Estimates will never be *accurate*
  - The design will never be *perfect*

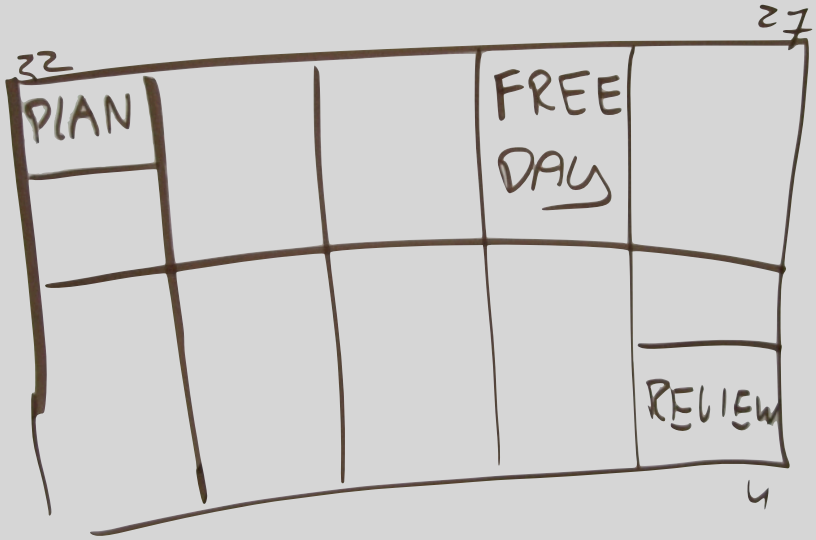
Design Tips for Part 2

- Time-box the design activity
- Involve everyone, avoid one person dominating
  - Team should reach a *common* understanding
  - Hand *everyone* a marker pen, stickies, etc.
- Model at whiteboards or flip charts
  - Focus on the essential, avoid extraneous detail
  - Try to be useful, not precise
  - Forget full-blown UML diagrams – Boxes, lines, arrows and ugly stick figures work best!

Steps in Part 2

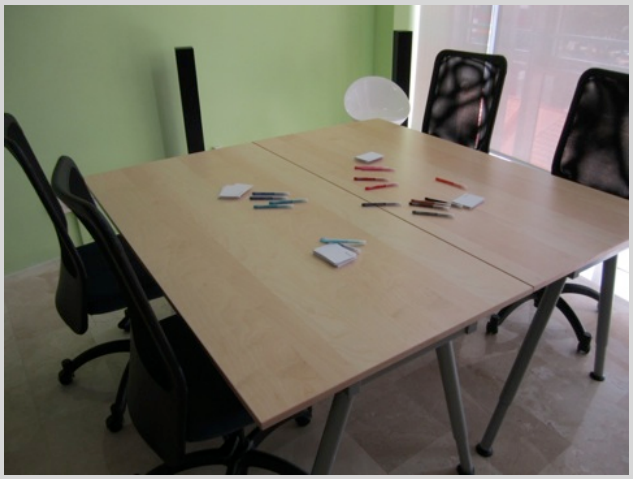
Modeling

Sprint calendar

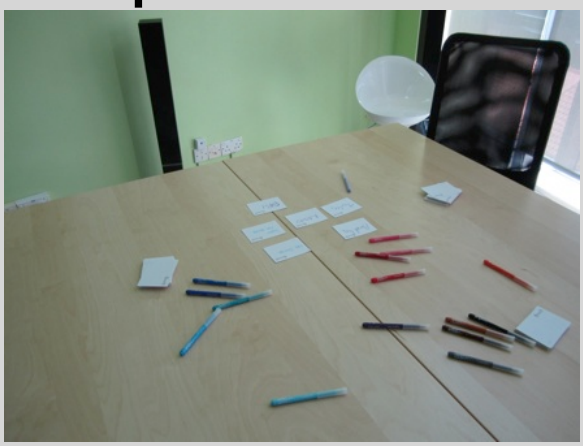


Good table (even though it is not round :( )

Use Index Cards



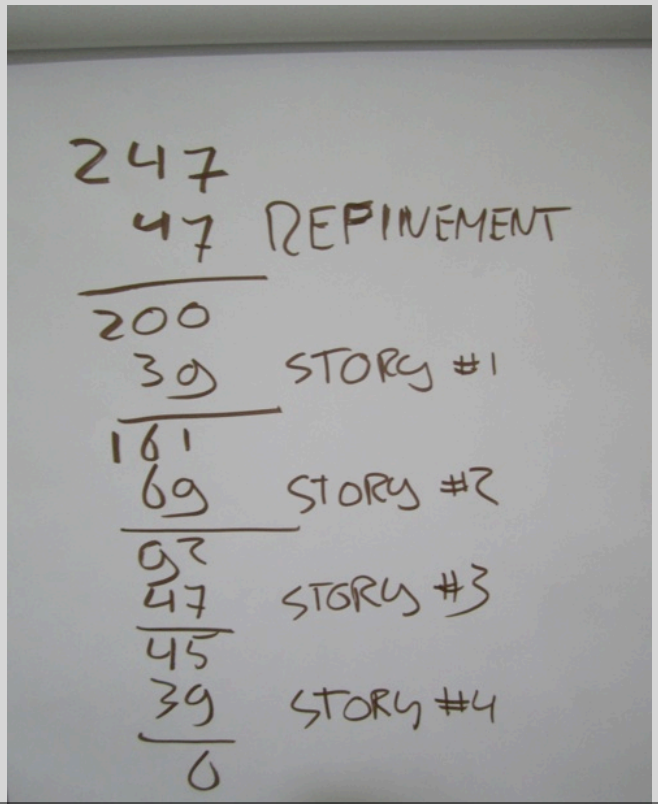
Use the Table Space



Planning Poker

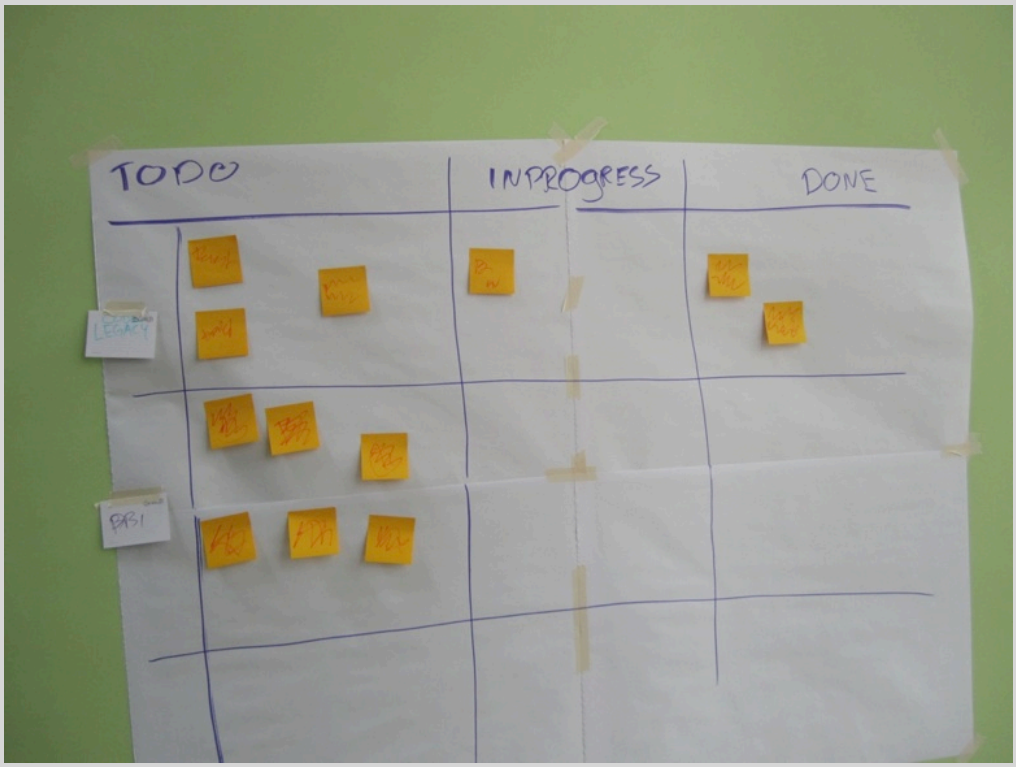
Poker in Action

Calculations

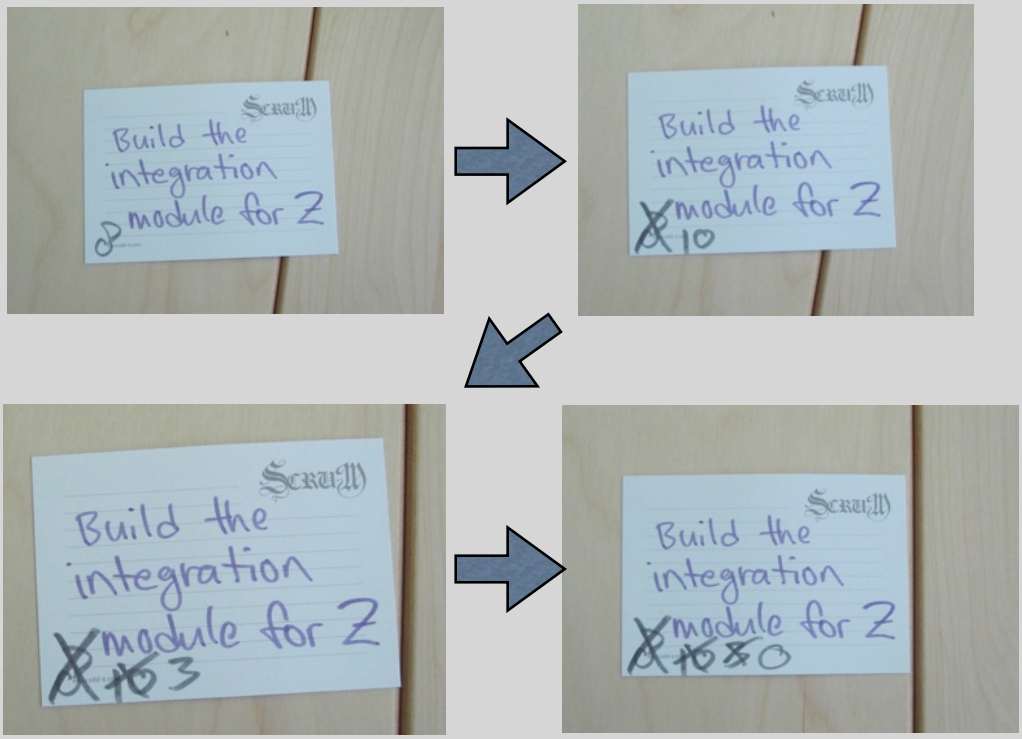


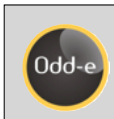
Task Board (Cork)

Task Board (Post-it)



Updating Estimates

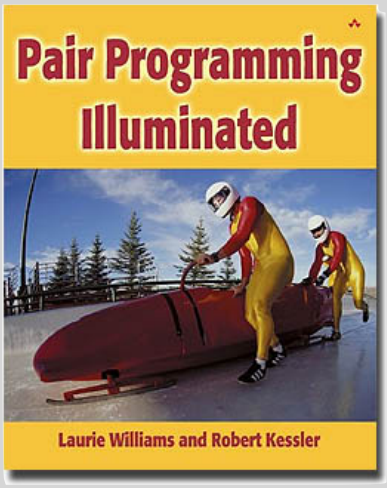




Exercise:
Do Sprint Planning Part 2

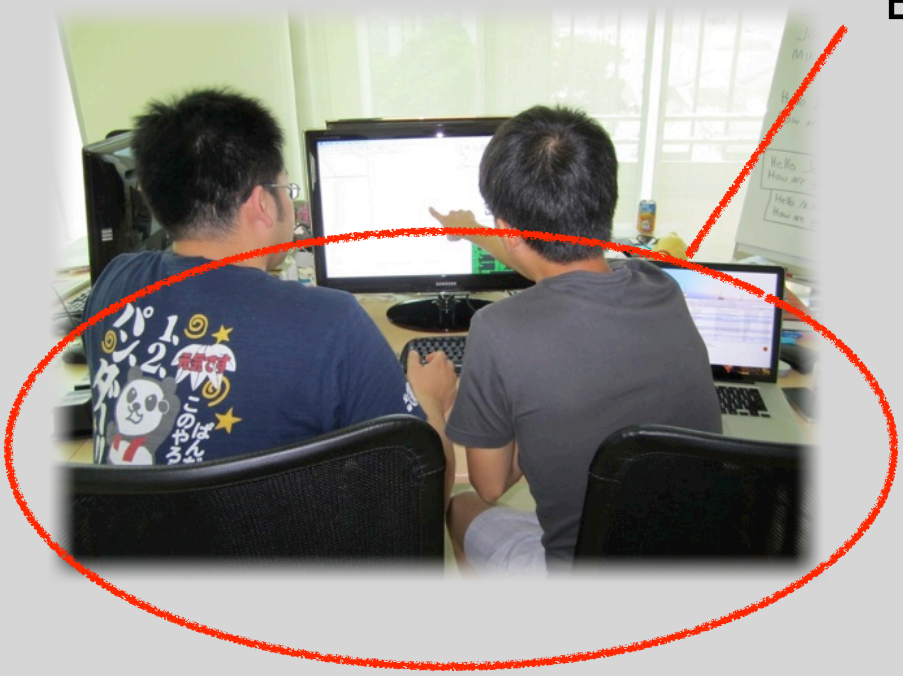
Pair Programming

- Two people working together on one computer
  - Not just for programming
- Benefits:
  - Shared understanding of what code does
  - Continuous review helps spot mistakes quicker
  - Healthy peer pressure discourages bad habits
  - Learning from others through observation and discussions about practices and techniques



How to pair program

How to pair program



Enough Space

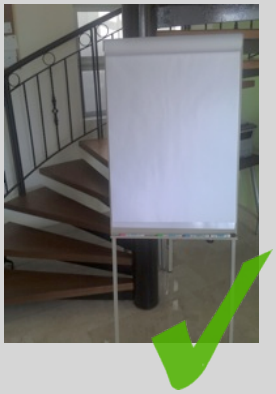
How to pair program

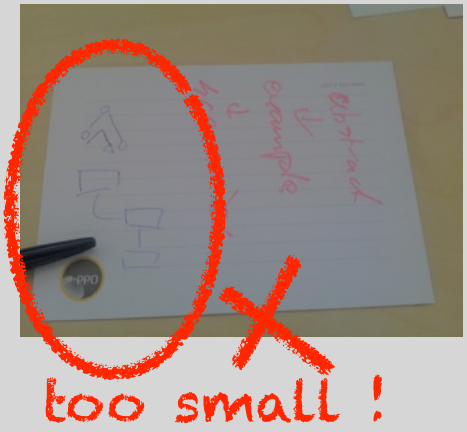
Think aloud

Ask questions

Tips

- Avoid dominating
- Don't point it out trivial mistakes too quickly, e.g. missing semicolon
- Design with flip chart or large piece of paper





Team doesn't like to pair?

- Discuss with the team how much pair programming they feel is good
  - When do they feel it is appropriate
  - What is it expected
  - When is it optional
  - Working agreement?
- Seek to improve their pairing technique
- Try:
  - Share videos of how others pair program
  - Ping-pong programming to give structure and make it interesting
  - Pomodoro Technique for maintaining focus

Ping-Pong Programming

1. **Jim** writes a failing test

2. **Bob** writes code to pass the test

3. Jim and Bob refactor the code **together**

4. Cycle starts again: **Bob** writes a new failing test and hands the keyboard back to Jim

Continuous Integration

What is Continuous Integration?

CI System

NOT the point!

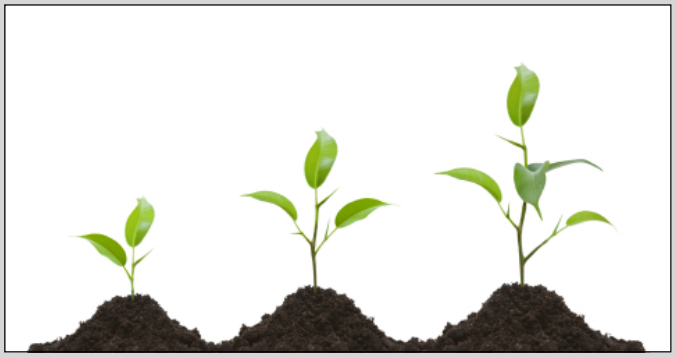
Automation

CI is a developer practice

Discipline to integrate frequently (hourly, at least daily)

CI is a developer practice

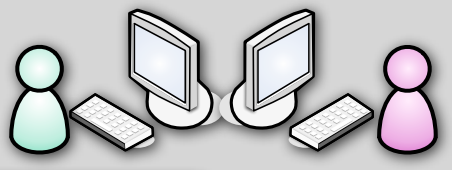
Strive to make small changes



CI is a developer practice

Strive for fast feedback

Scenario



Hey! I'm changing the method that conflicts with your changes!

Roger!

... done and committed!

Ok! I'll update and check the merged changes!
... Looks good!

Great!

CI is a practice...

it is about **what people do**, **not about what tools** they use.

As a project starts to scale, it is **easy to be deceived** into thinking that the **team is practicing continuous integration** just because **all of the tools are set up and running**.

If developers do not have the **discipline to integrate** their changes on a **regular basis** or to maintain the **integration environment in a good working order** they are **not practicing continuous integration**. Full stop.

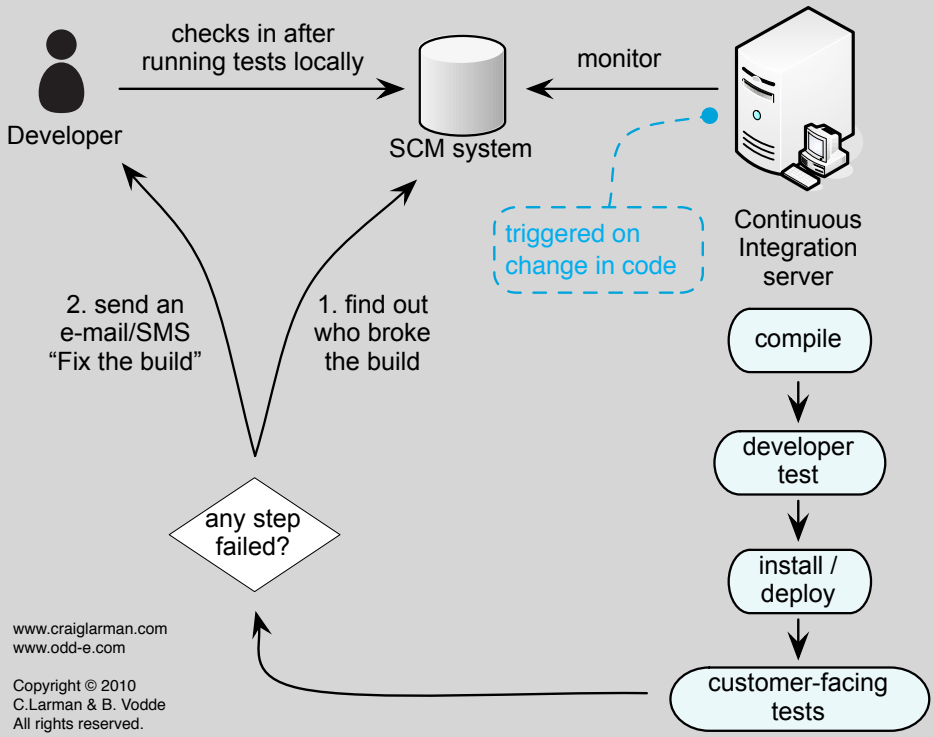
R. Owen Rogers

Adopting CI...

requires a change in **human behavior**!

Continuous Integration System

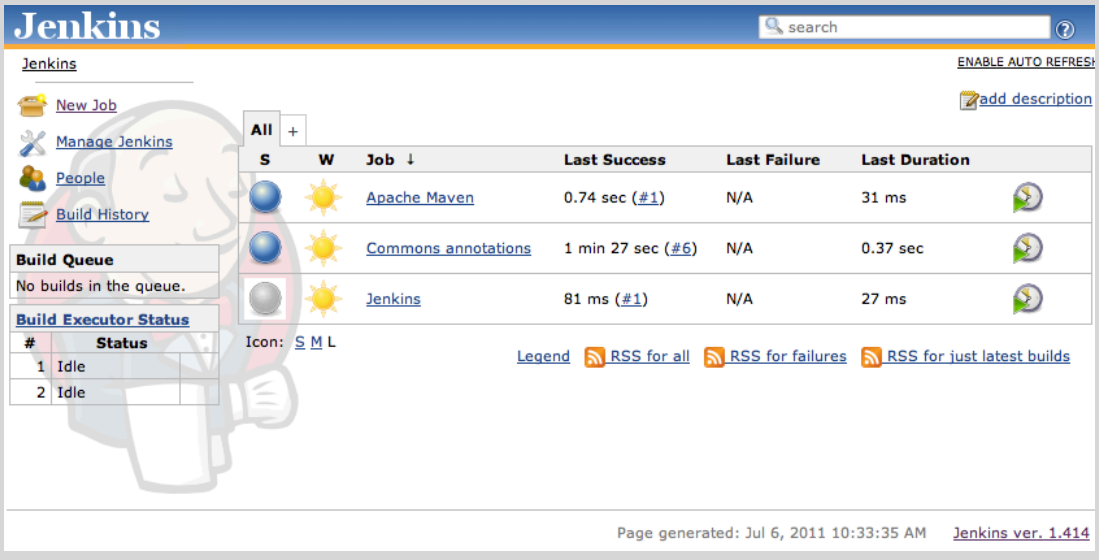
CI System



www.craiglarman.com
www.odd-e.com

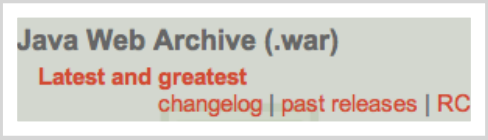
Copyright © 2010
C.Larman & B. Vodde
All rights reserved.

Jenkins



Go to

Download

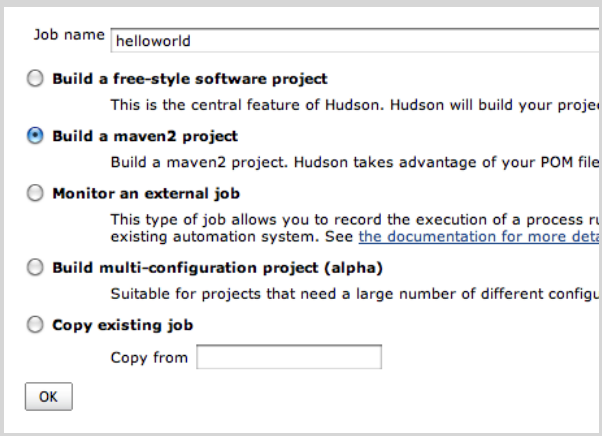


Start up

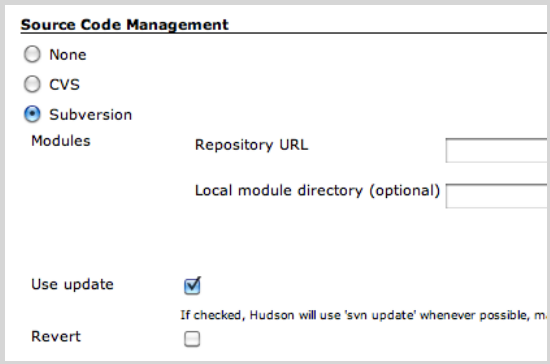
```
java -jar jenkins.war
```

For more details, http://wiki.jenkins-ci.org/display/JENKINS/Use+Jenkins

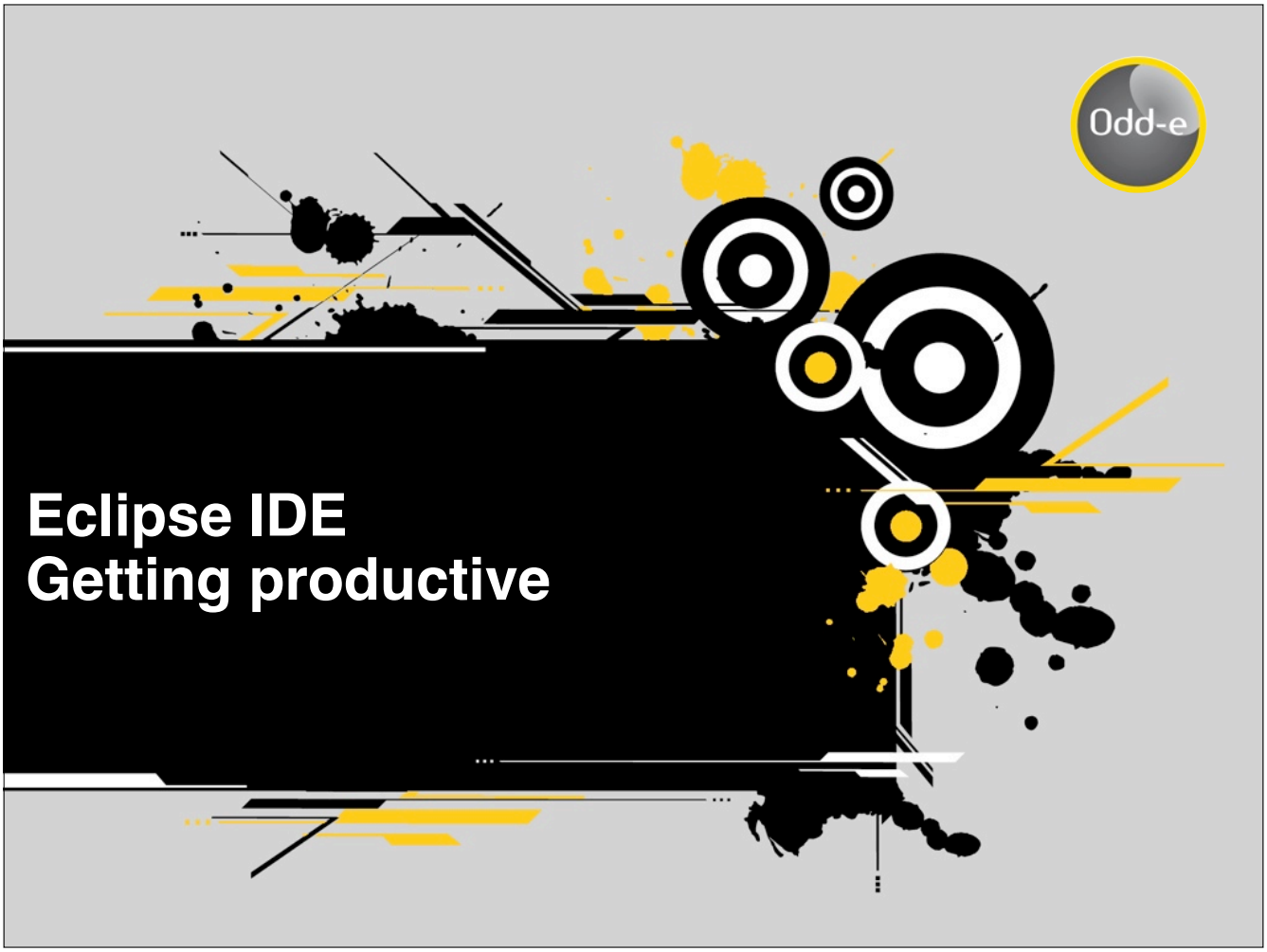
Job settings



Maven project



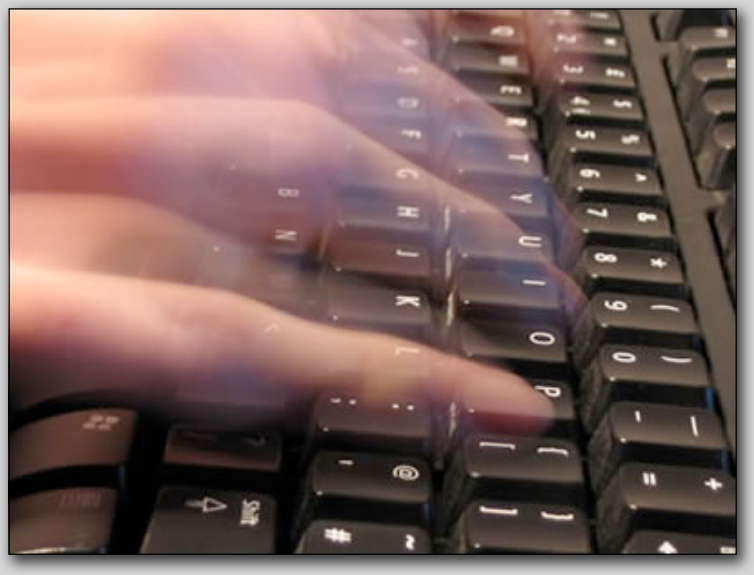
SCM
(Source Code Management)




```
        for (int i = 0
            copy[i] =
        }
        return copy;
}
```

Programming is a text based activity

Instead of precise mousing...



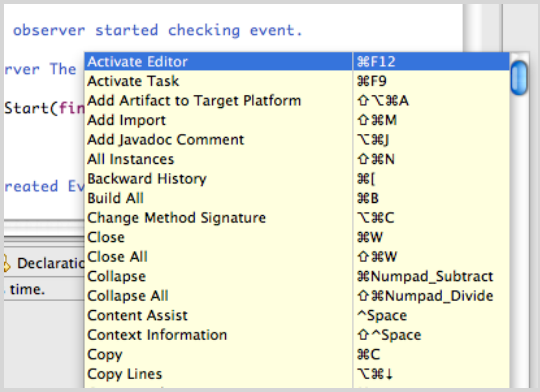
prefer the keyboard when coding

Learning methods

Do not memorize a long list

Learning methods

Short-cut keys with Shift+Ctrl+L



Key promoter tools like MouseFeed

Keyboard Shortcuts

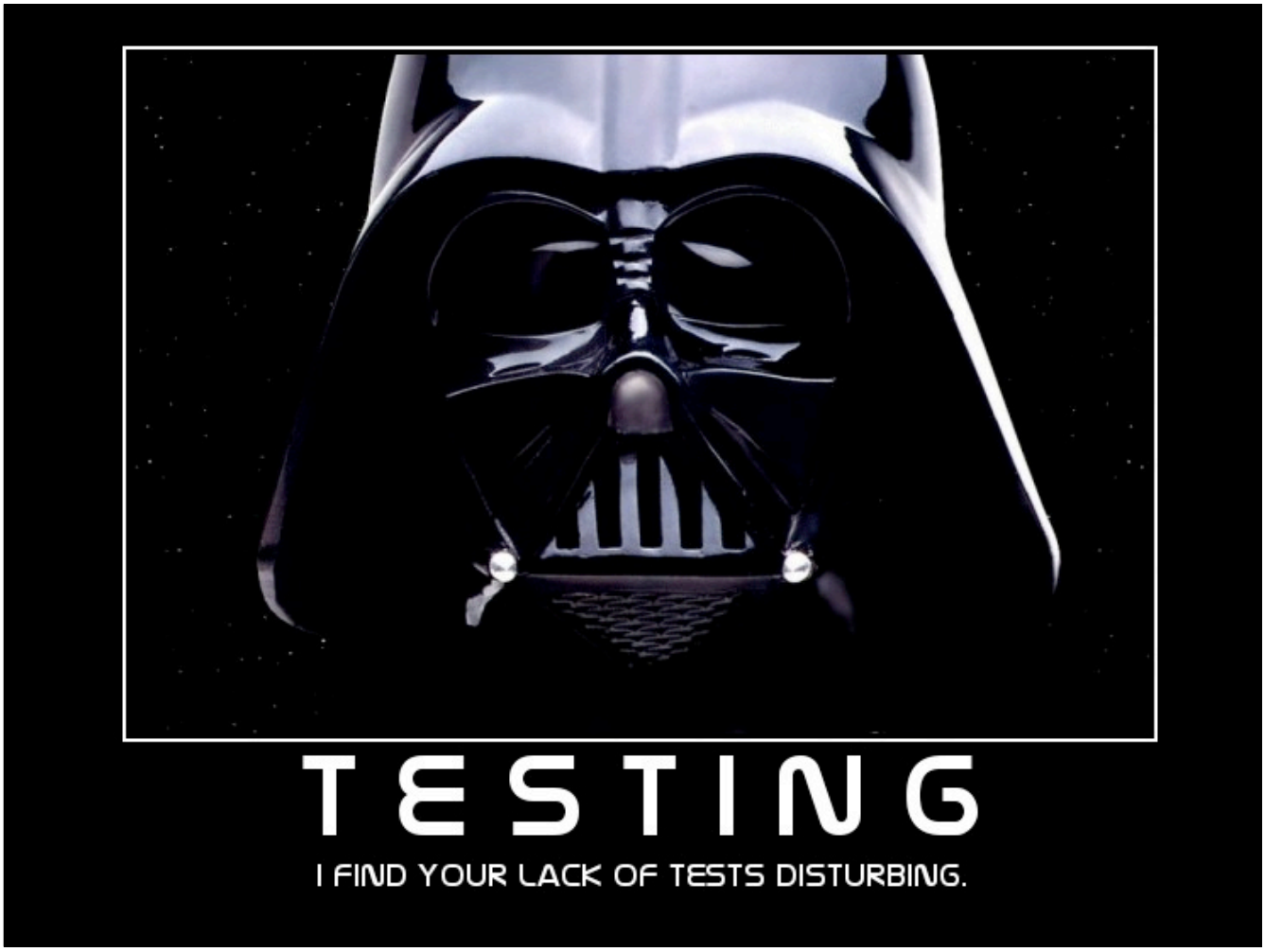
Quick Fix	Ctrl+1
General Short-cut	Ctrl+3
Delete a line	Ctrl+D
Search outline of the class	Ctrl+O
Content assist	Ctrl+Space
Extract Method (Refactor)	Shift+Alt+M
Inline (Refactor)	Shift+Alt+I
Rename (Refactor)	Shift+Alt+R
Search and open for Type	Shift+Ctrl+T
Run a new JUnit test	Shift+Alt+X T
Run last unit test	Ctrl+F11

Eclipse plugins

M2Eclipse a Maven client

Subclipse a SUBVERSION client

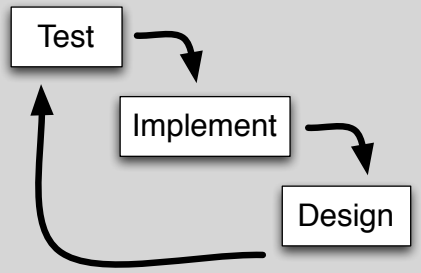
MouseFeed feeds you keyboard shortcuts.

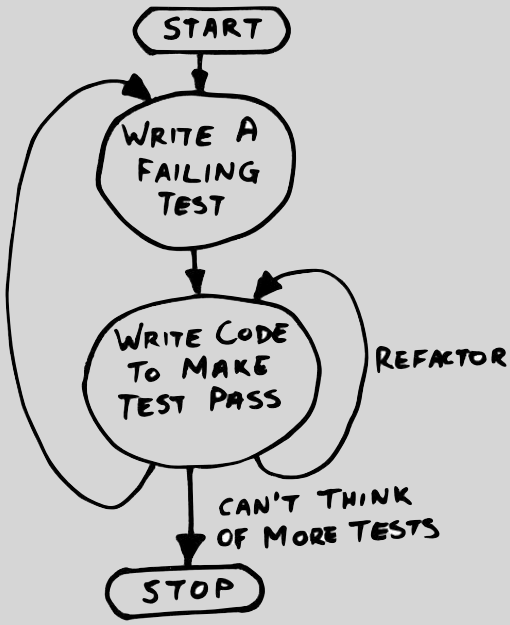


Odd-e

Quick Intro

TDD Cycle

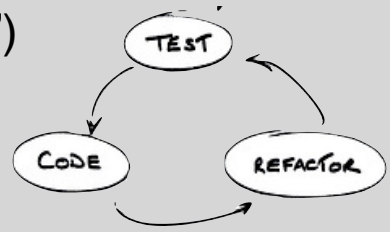




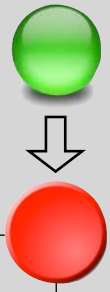
- Short
- Rhythmic
- Incremental
- Design-focused
- **Disciplined**

TDD Rules

- The one rule to rule them all
  - Only ever write code to fix a failing test
- 3 Steps:
  - Write a test (which fails “red”)
  - Code (to make test pass “green”)
  - Refactor (test still pass “green”)
  - Repeat forever



Write test

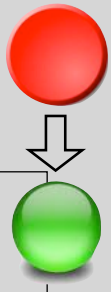


TemplateEngineTest.java

```
public class TemplateEngineTest {

    @Test
    public void templatesWithoutPlaceHoldersDoNotChange() throws Exception {
        Template template = new Template();
        TemplatePlaceholderValues replacementValues = new TemplatePlaceholderValues();
        template.set("Nothing here");
        assertEquals("Nothing here", template.replaceValues(replacementValues));
    }
}
```

This does not compile!
Template and TemplatePlaceholderValues do **not** exist.

Step 2 -> Code



Template.java

```
public class Template {

    private String template;

    public void set(String template) {
        this.template = template;
    }

    public String replaceValues(TemplatePlaceholderValues replacementValues) {
        return template;
    }

}
```

TemplatePlaceholderValues.java

```
public class
TemplatePlaceholderValues {

}
```

Run the test!

Runs: 1/1 Errors: 0 Failures: 0

Refactor

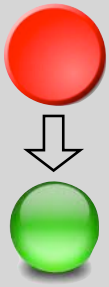
New Test

TemplateEngineTest.java

```
@Test
public void templateWithOnePlaceholderReplacesItToValue() throws Exception {
    Template template = new Template();
    TemplatePlaceholderValues replacementValues = new TemplatePlaceholderValues();
    template.set("Hi, $name");
    replacementValues.placeholder("$name").hasValue("John");
    assertEquals("Hi, John", template.replaceValues(replacementValues));
}
```

Again, won't compile!

... and implementation



Template.java

```
public String replaceValues(TemplatePlaceholderValues replacementValues) {
    return template.replace("$name", "John");
}
```

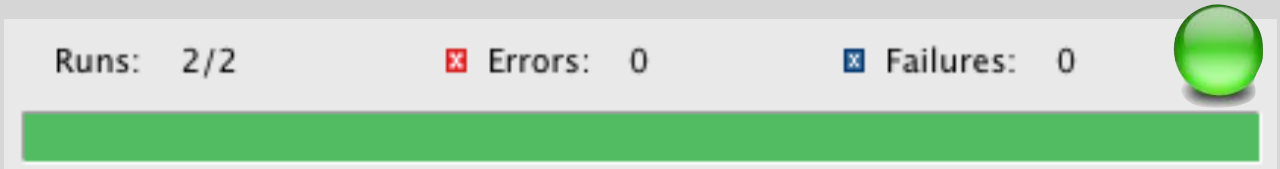
TemplatePlaceholderValues.java

```
public class TemplatePlaceholderValues {

    public Placeholder placeholder(String name) {
        return new Placeholder();
    }
}
```

Placeholder.java

```
public class Placeholder {

    public void hasValue(String value) {
    }
}
```

Run!



Refactor (test)

Pull out common objects

TemplateEngineTest.java

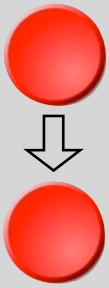
```
Template template = new Template();
TemplatePlaceholderValues replacementValues = new TemplatePlaceholderValues();

@Test
public void templateWithoutPlaceholderDoesNotChange() throws Exception {
    Template template = new Template();
    TemplatePlaceholderValues replacementValues = new TemplatePlaceholderValues();
    template.set("Nothing here");
    assertEquals("Nothing here", template.replaceValues(replacementValues));
}

@Test
public void templateWhereOnePlaceholderIsReplaced() throws Exception {
    template.set("Hi, $name");
    replacementValues.placeholder("$name").hasValue("John");
    assertEquals("Hi, John", template.replaceValues(replacementValues));
}
```

New Test -> Drive out logic

TemplateEngineTest.java

```
@Test
public void templateWhereOnePlaceholderIsReplacedAndOneIsNot() throws Exception {
    template.set("Hi, $name $last_name");
    replacementValues.placeholder("$last_name").hasValue("Doe");
    assertEquals("Hi, $name Doe", template.replaceValues(replacementValues));
}
```


More implementation



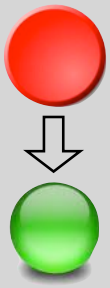
Template.java

```
public String replaceValues(TemplatePlaceholderValues replacementValues) {
    return template.replace(replacementValues.firstPlaceholder().name(),
           replacementValues.firstPlaceholder().value());
}
```

TemplatePlaceholderValues.java

```
private Placeholder placeholder = new Placeholder("");

public Placeholder placeholder(String name) {
    placeholder = new Placeholder(name);
    return placeholder;
}

public Placeholder firstPlaceholder() {
    return placeholder;
}
```

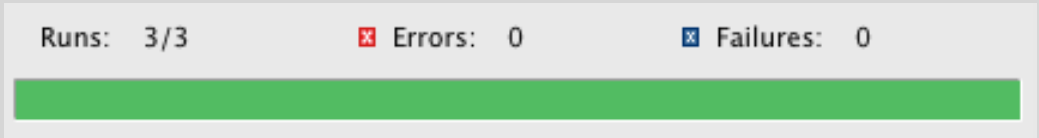
And more implementation



Placeholder.java

```
public class Placeholder {

    private String name;
    private String value = new String();

    public Placeholder(String name) {
        this.name  = name;
    }

    public void hasValue(String value) {
        this.value = value;
    }

    public String name() {
        return name;
    }

    public String value() {
        return value;
    }
}
```


Run!



Refactoring

TemplatePlaceholderValues.java

```
private Placeholder placeholder = new Placeholder();

public Placeholder placeholder(String name) {
    placeholder = new Placeholder(name);
    placeholder.setName(name);
    return placeholder;
}
```

Placeholder.java

```
private String name = new String();
private String value = new String();

public void setName(String name) {
    this.name = name;

}
```


New Test

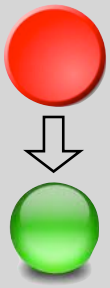
TemplateEngineTest.java

```
@Test
public void templateWherePlaceholdersAreReplaced() throws Exception {
    template.set("Hi, $first_name $last_name");
    replacementValues.placeholder("$first_name").hasValue("John");
    replacementValues.placeholder("$last_name").hasValue("Doe");
    assertEquals("Hi, John Doe", template.replaceValues(replacementValues));
}
```

Compiling passed,
but test fails

org.junit.ComparisonFailure: expected:<Hi, [John] Doe> but was:<Hi, [$first_name] Doe>

Backtrack for refactoring



```
// @Test
// public void templateWherePlaceholdersAreReplaced() throws Exception {
//     template.set("Hi, $first_name $last_name");
//     replacementValues.placeholder("$first_name").hasValue("John");
//     replacementValues.placeholder("$last_name").hasValue("Doe");
//     assertEquals("Hi, John Doe", template.replaceValues(replacementValues));
// }
```

Implementation would be too complicated.
Prepare the code for the implementation first!

Template.java

```
public String replaceValues(TemplatePlaceholderValues replacementValues) {
    return template.replace(replacementValues.firstPlaceholder().name(),
           replacementValues.firstPlaceholder().value());
}
```

Would require iteration over replacementValues.
And it already smells!

Refactor before...

Template.java

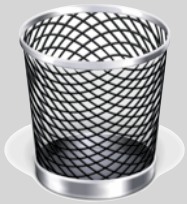
```
public String replaceValues(TemplatePlaceholderValues replacementValues) {
    return template.replace(replacementValues.firstPlaceholder().name(),
           replacementValues.firstPlaceholder().value());
    return replacementValues.replaceValuesIn(template);
}
```

TemplatePlaceholderValues.java

```
public String replaceValuesIn(String template) {
    return template.replace(placeholder.name(), placeholder.value());
}
```

TemplatePlaceholderValues.java

```
public Placeholder firstPlaceholder() {
    return placeholder;
}
```



And more...

TemplatePlaceholderValues.java

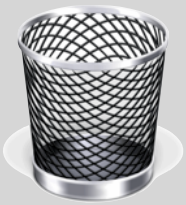
```
public String replaceValuesIn(String template) {
    return template.replace(placeholder.name(), placeholder.value());
    return placeholder.replaceValuesIn(template);
}
```

Placeholder.java

```
public String replaceValuesIn(String template) {
    return template.replace(name, value);
}
```

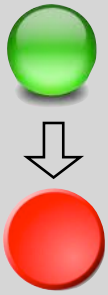
Placeholder.java

```
public String name() {
    return name;
}

public String value() {
    return value;
}
```



Still green!

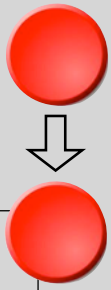
Try again!



TemplateEngineTest.java

```
@Test
public void templateWherePlaceholdersAreReplaced() throws Exception {
    template.set("Hi, $first_name $last_name");
    replacementValues.placeholder("$first_name").hasValue("John");
    replacementValues.placeholder("$last_name").hasValue("Doe");
    assertEquals("Hi, John Doe", template.replaceValues(replacementValues));
}
```

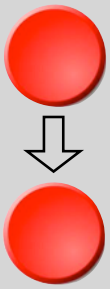

Implementation

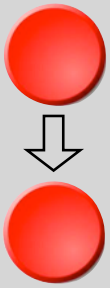


TemplatePlaceholderValues.java

```
private List<Placeholder> placeholders = new ArrayList<Placeholder>();

public Placeholder placeholder(String name) {
    Placeholder placeholder = new Placeholder();
    placeholders.add(placeholder);
    return placeholder;
}

public String replaceValuesIn(String template) {
    String replacedTemplate = template;
    for (Placeholder placeholder : placeholders) {
        replacedTemplate = placeholder.replaceValuesIn(replacedTemplate);
    }
    return replacedTemplate;
}
```

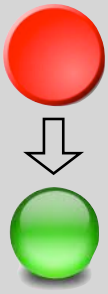
Run!





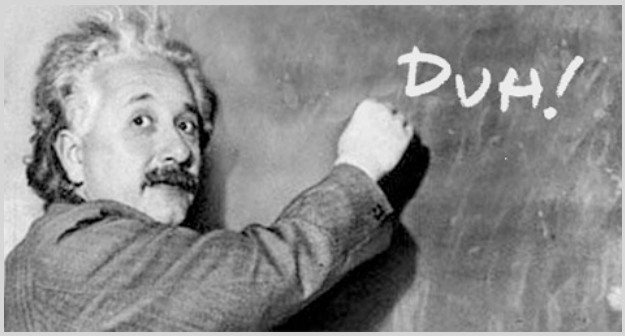
org.junit.ComparisonFailure:
expected:<[Hi,]John>
but was:<[JohnHJohniJohn,John John$JohnnJohnaJohnmJohne]John>

Mistake!



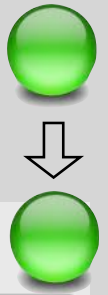
TemplatePlaceholderValues.java

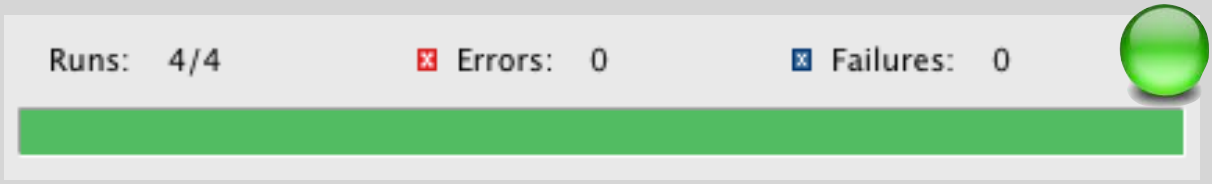
```
public Placeholder placeholder(String name) {
    Placeholder placeholder = new Placeholder();
    placeholder.setName(name);
    placeholders.add(placeholder);
    return placeholder;
}
```

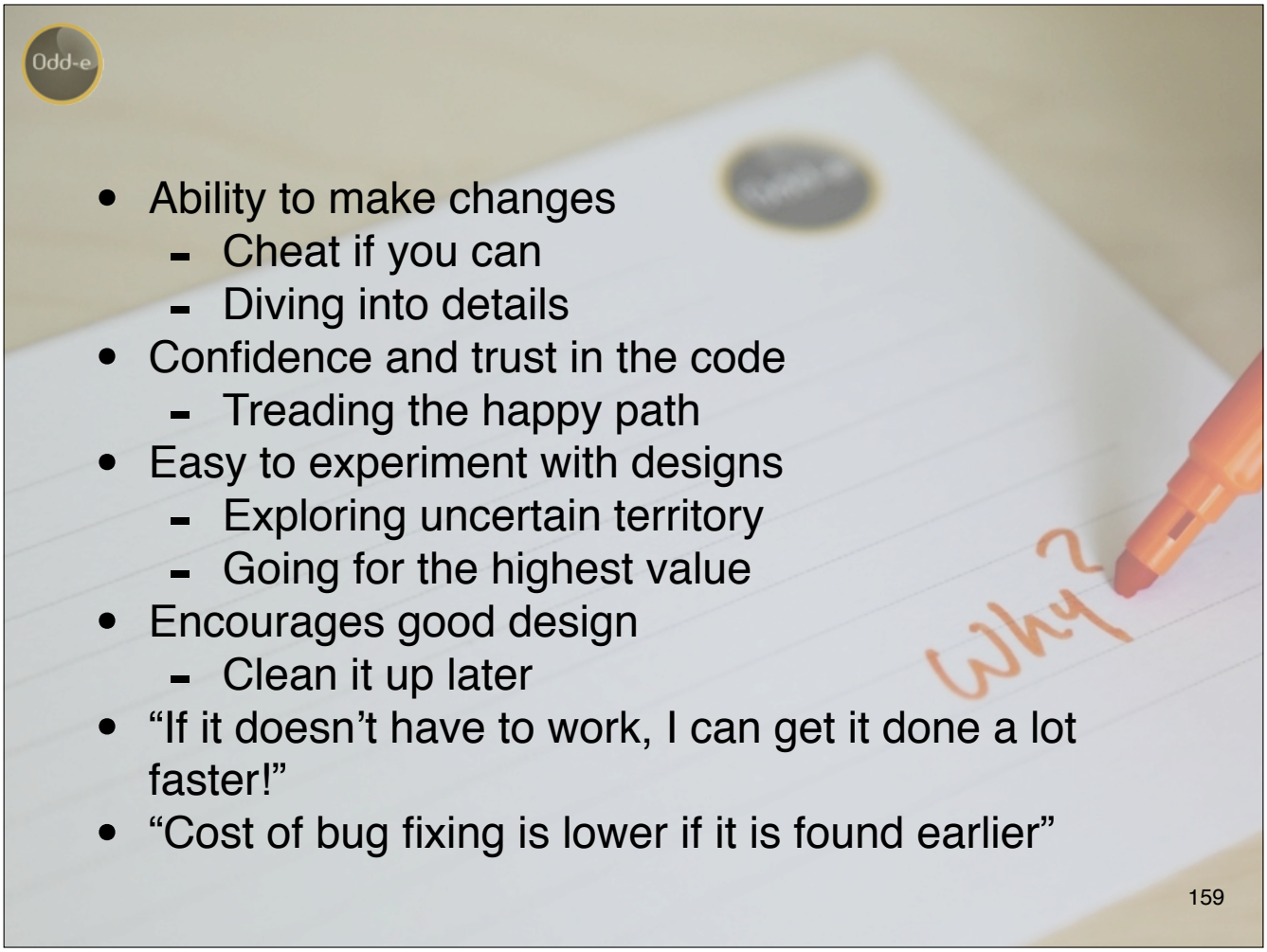


Forgot to set the placeholder name!

Run!







- Ability to make changes
  - Cheat if you can
  - Diving into details
- Confidence and trust in the code
  - Treading the happy path
- Easy to experiment with designs
  - Exploring uncertain territory
  - Going for the highest value
- Encourages good design
  - Clean it up later
- “If it doesn’t have to work, I can get it done a lot faster!”
- “Cost of bug fixing is lower if it is found earlier”

Refactoring

Structured code transformation to prepare the code for change

Key points:

- Doesn't adjust functionality
- Small and disciplined
- Well defined
- Keeps code healthy

Code Smells

It smells!

It stinks!

A sign there is probably something wrong with your code

Example: Duplicate code

```
public DataSet navigateFileUp()
{
    int fileCount = this.db.getFileCount();
    int x = 0;
    if(fileCount <= 0) return null;
    x = (this.db.getCurrentFileIndex() + 1) % fileCount;
    if(this.db.getFileSize(x) == 0)//file still empty
        return this.db.setNewCurrentSet(x);
    return this.db.setCurrentSet(x, this.db.getCurrentCorrectAnsweredIndex(), 0);
}

public DataSet navigateFileDown()
{
    int fileCount = this.db.getFileCount();
    int x = 0;
    if(fileCount <= 0) return null;
    x = (this.db.getCurrentFileIndex() - 1 + fileCount) % fileCount;
    if(this.db.getFileSize(x) == 0)//file still empty
        return this.db.setNewCurrentSet(x);
    return this.db.setCurrentSet(x, this.db.getCurrentCorrectAnsweredIndex(), 0);
}
```

Only difference!

Refactoring visualized

Cost of change increases rapidly!

Cost of change does not increase

Example refactoring - Original Code

```
if (andNode.getFirstNode().getNodeType().alwaysTrue()) {
    // compile first node as non-expr and then second node
    compile(andNode.getFirstNode(), context, false);
    compile(andNode.getSecondNode(), context, expr);
} else if (andNode.getFirstNode().getNodeType().alwaysFalse()) {
    // compile first node only
    compile(andNode.getFirstNode(), context, expr);
} else {
    compile(andNode.getFirstNode(), context, true);
    BranchCallback longCallback = new BranchCallback() {
                public void branch(BodyCompiler context) {
                    compile(andNode.getSecondNode(), context, true);
                }
            };

    context.performLogicalAnd(longCallback);
    if (!expr) context.consumeCurrentValue();
}
```


Refactoring Example Extract Local Variable

```
Node firstNode = andNode.getFirstNode();
if (firstNode.getNodeType().alwaysTrue()) {
     // compile first node as non-expr and then second node
     compile(firstNode, context, false);
     compile(andNode.getSecondNode(), context, expr);
 } else if (firstNode.getNodeType().alwaysFalse()) {
     // compile first node only
     compile(firstNode, context, expr);
 } else {
     compile(firstNode, context, true);
     BranchCallback longCallback = new BranchCallback() {
                 public void branch(BodyCompiler context) {
                     compile(andNode.getSecondNode(), context, true);
                 }
             };

     context.performLogicalAnd(longCallback);
     if (!expr) context.consumeCurrentValue();
 }
```

Alt-Shift-L

Refactoring Example Extract Method

```
public void compileAnd(Node node, BodyCompiler context, final boolean expr) {
    final AndNode andNode = (AndNode) node;

    ...
    } else {
        compile(andNode.getFirstNode(), context, true);
        BranchCallback longCallback = createBranchCallbackForNode(andNode);

        context.performLogicalAnd(longCallback);
        if (!expr) context.consumeCurrentValue();
    }
}

private BranchCallback createBranchCallbackForNode(final AndNode andNode) {
   return new BranchCallback() {
               public void branch(BodyCompiler context) {
                   compile(andNode.getSecondNode(), context, true);
               }
           };
}
```

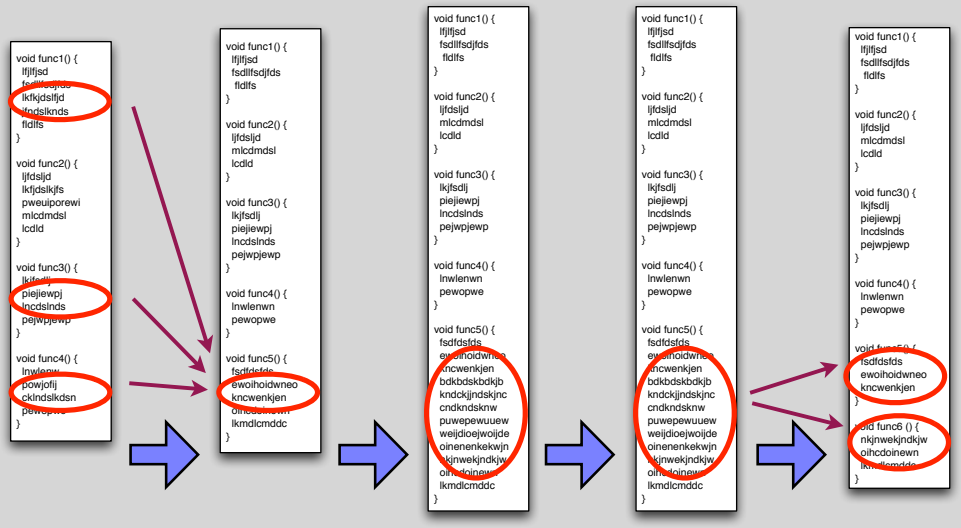
Alt-Shift-M

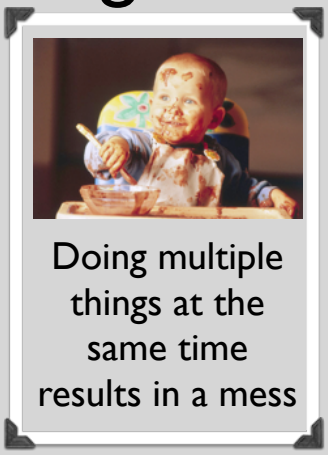
Refactoring Example Inline Variable

```
public void compileAnd(Node node, BodyCompiler context, final boolean expr) {
    final AndNode andNode = (AndNode) node;

    if (andNode.getFirstNode().getNodeType().alwaysTrue()) {
        // compile first node as non-expr and then second node
        compile(andNode.getFirstNode(), context, false);
        compile(andNode.getSecondNode(), context, expr);
    } else if (andNode.getFirstNode().getNodeType().alwaysFalse()) {
        // compile first node only
        compile(andNode.getFirstNode(), context, expr);
    } else {
        compile(andNode.getFirstNode(), context, true);
        context.performLogicalAnd(createBranchCallbackForNode(andNode));
        if (!expr) context.consumeCurrentValue();
    }
}
```

Alt-Shift-I

Refactor before, Refactor after, but do **not** refactor during





Doing multiple things at the same time results in a mess

1. Multiple change points
   Need to prepare for the change
2. Refactor before
   Change point now at one place
3. Make the change
   Do not refactor
4. Identify smells
   opportunities to clean up
5. Refactor after
   Clean up to keep the code in a good state

It is not a unit test when:

- It talks to the database
- It communicates across the network
- It touches the file system
- It can't run at the same time as any of your other unit tests
- You have to do special things to your environment (such as editing config files) to run it.

Michael Feathers

JUnit 4

Test class containing tests

Import JUnit package

Import assertion methods

```
import org.junit.Test;
import static org.junit.Assert.*;

public class PersonTest {

    @Test
    public void defaultNameOfThePersonIsJohn() throws Exception {
        Person john = new Person();
        assertEquals("John", john.name());
    }
}
```

Test method

Annotation for test methods

Assertion

Test Fixture

Data needed in each test.

Import Before/ After annotations

```
import org.junit.After;
import org.junit.Before;

public class PersonTest {

    Person john = new Person();

    @Before
     public void initializePerson() {
        john.initialize();
    }

    @After
    public void cleanUpPerson() {
        john.clean();
    }

    @Test
    public void defaultNameOfThePersonIsJohn() throws Exception {
        assertEquals("John", john.name());
    }
}
```

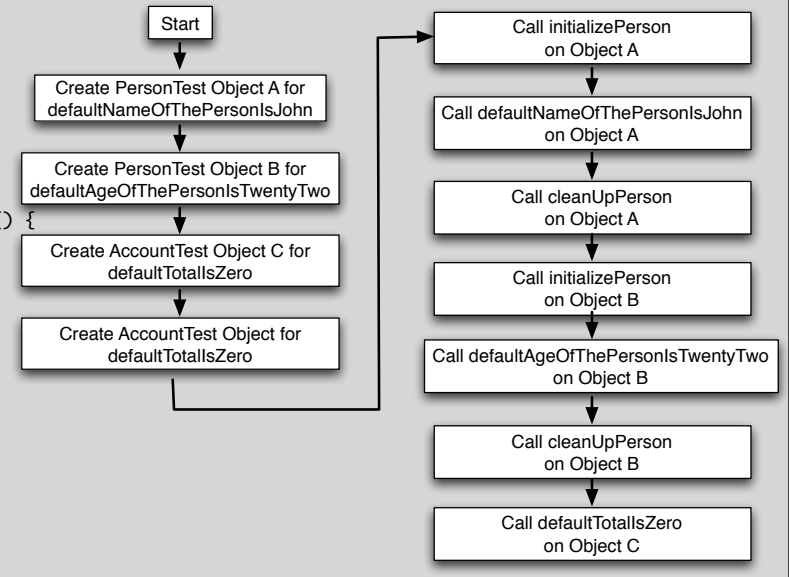
Run before each test

Run after each test

The data and setup / teardown (before, after) is also sometimes called: **test fixture**

JUnit Runtime Model

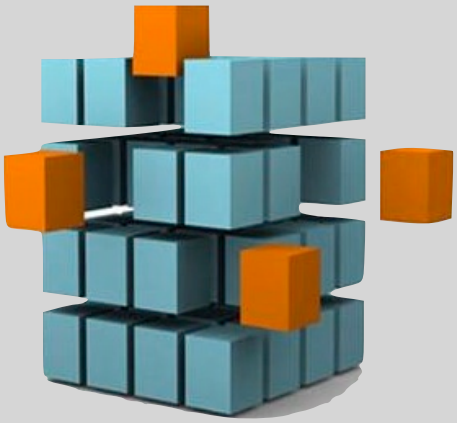
```
public class PersonTest {

    Person john = new Person();

    @Test
    public void defaultNameOfThePersonIsJohn() {
        assertEquals("John", john.name());
    }

    @Test
    public void defaultAgeOfThePersonIsTwentyTwo() {
        assertEquals(22, john.age());
    }

    @Before
     public void initializePerson() {
        john.initialize();
    }

    @After
    public void cleanUpPerson() {
        john.clean();
    }

public class AccountTest {

    @Test
    public void defaultTotalIsZero() {
        Account account = new Account();
        assertEquals(0, account.total());
    }
```

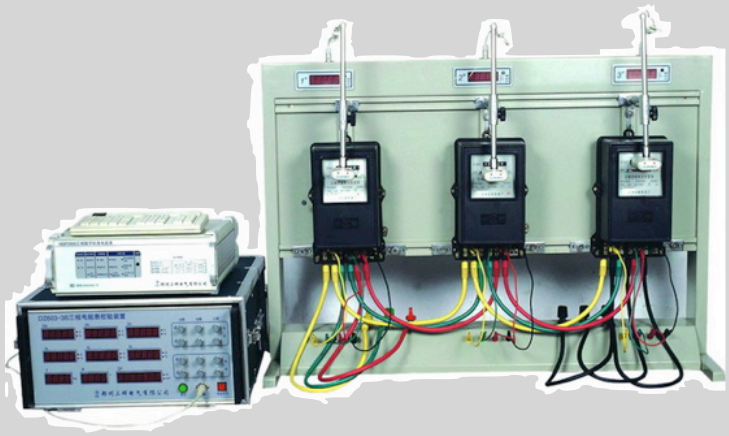


First create all objects

Then do the execution

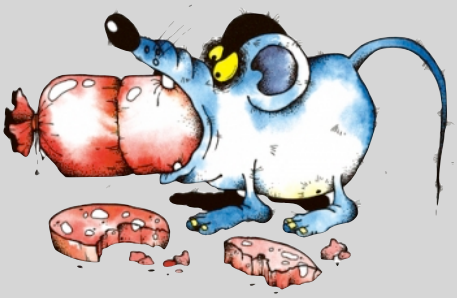
Modularity == Testability





Focus on tests first ensures modularity and other good design principles

YAGNI



Developers often ask questions such as:

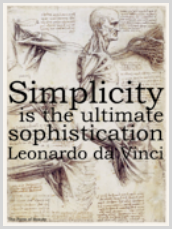
- "What if..."
- "Perhaps x will change"
- "How can I make it more flexible?"
- "This would be nice!"

Assume that:

You Ain't Gonna Need It

and prevent designs from becoming complex!

Simplicity is the ultimate sophistication



Flexible complex design on paper → Something unexpected **always** changes → More complexity than needed. Hard to maintain. Legacy

---

Simple design (not simplistic) → Something unexpected **always** changes → Easier to adopt. Simple design turns out to be easier to change. Less complexity.

Design from the perspective of use rather than implementation

What is design?

"After reviewing the software development lifecycle as I understood it, I concluded that the only software documentation that actually seems to satisfy the criteria of an engineering design is the source code listings."

– Jack Reeves,
"What is Software Design", C++ Journal, 1992.

"In the end, the source code is the design."
– Robert C. Martin,
"Agile Software Development",

"Design"

Reality

Getting Started

Deciding what test to write

- Small steps, avoid long red
- Make progress
- Depth-first vs. breadth-first
- Drive out wrong code
- Where is the risk?

Your test-writing order will influence your design

Why is it difficult to sustain?

TDD:

- Steady pace
- Speed limits
- No traffic lights (bugs)
- Might **feel** slow!!

Traditional:

- Spurts
- Fast when no problems!
- Debugging
- **Feels** fast! But often slow

Sustainability

Time developers do not notice nor plan for

Traditional development

Speculate	Code	Test	Debug

Time ---------- vs ----------

Feels slower

Test-driven development

Spec ulate	Test and Code	Debug

Teams

Team

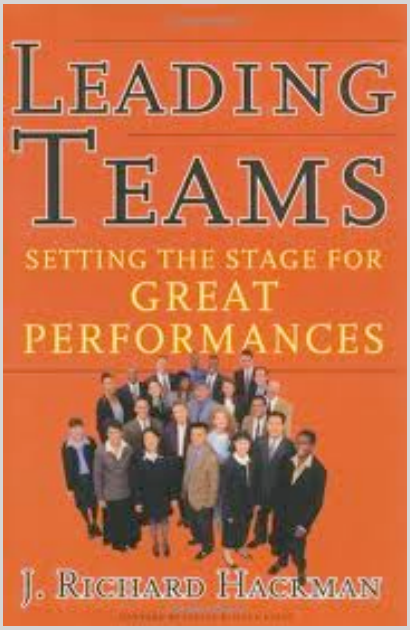
- A team comprises a group of people linked in a common purpose.

- Appropriate for conducting complex and interdependent tasks
- Team members have complementary skills and generate synergy
- Team members help one another

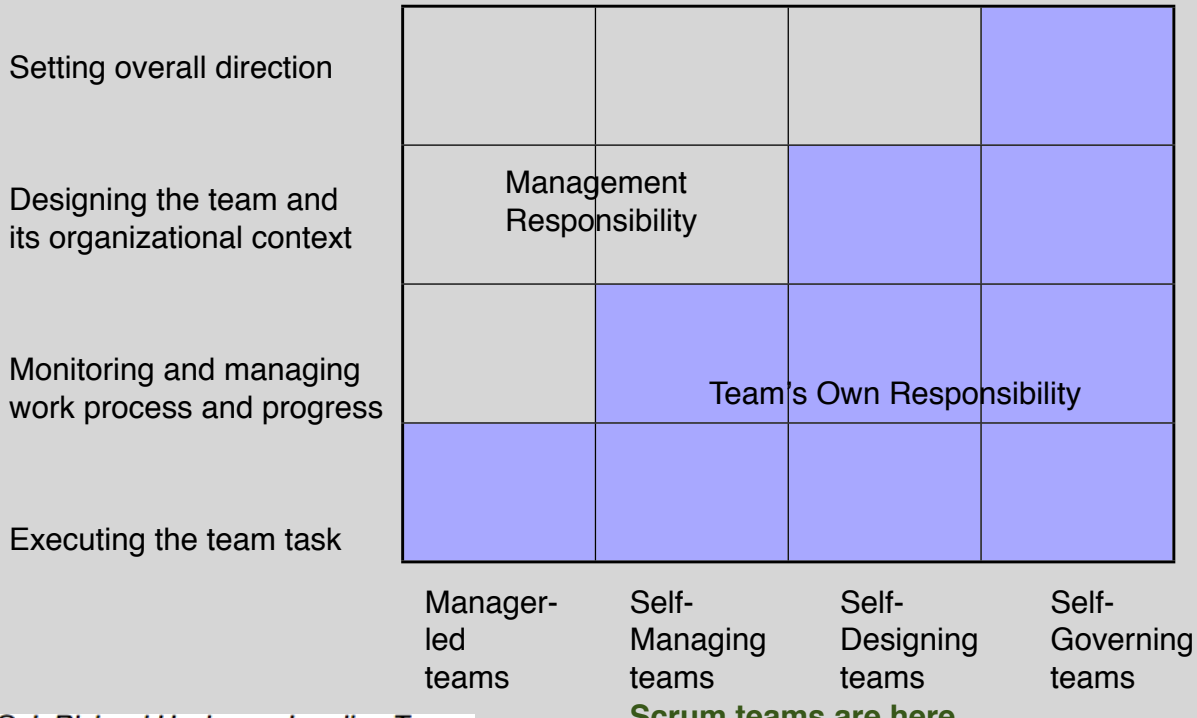
- A team has:
  - Shared work product
  - Interdependent work
  - Shared responsibility
  - Set of working agreements
  - Responsibility for managing the outside-team relationships
  - Distributed leadership

Real Team

- Appropriate task for teamwork
- Clear but moderately permeable membership boundaries
- Substantial but clearly delimited authority for managing its work
- Reasonably stable over time



Four Types of Work Teams

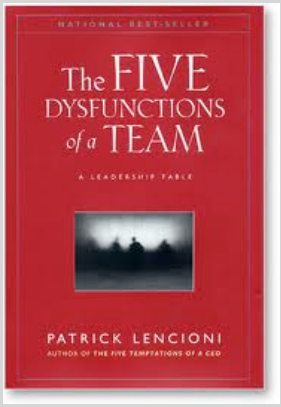


© J. Richard Hackman: Leading Teams

Scrum Team

- Self-organizing;
- Cross-functional with no roles;
- Seven plus or minus two;
- Responsible for committing to work;
- Authority to do whatever is needed to meet commitment;
- Open, collocated space;
- Resolution of conflicts;
- Rules of Etiquette:

The Five Dysfunctions of a Team



Chartering

- At the very start of product development:
  - Clarify the vision
  - Team values
  - Establish project community
  - Create working agreements
- Not part of Scrum, still useful.

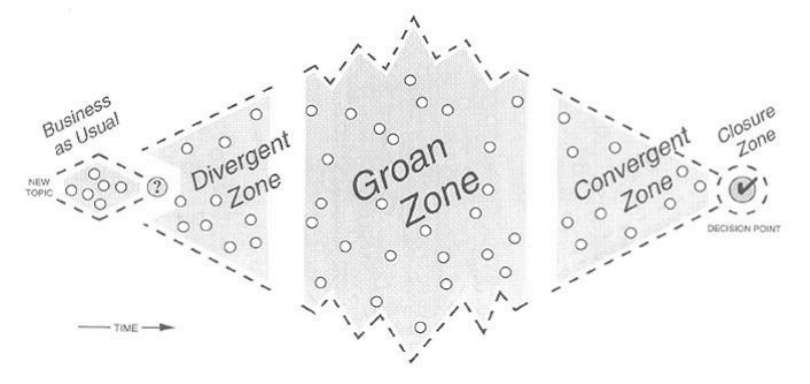
Working agreements

- Explicit statement of team accountability
- Categories:
  - Decision making
  - Timing
  - Respect
  - Technical
  - Social
- Related
  - Coding standards
  - Definition of “done”

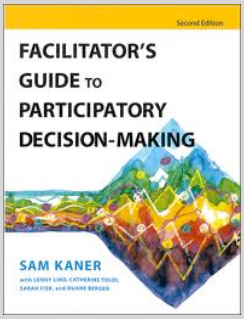
Decision making

- How does the team decide?
  - Consensus
  - Voting
  - Specialist decides
- Without an agreement on how to make decisions, team might have lots of problems.

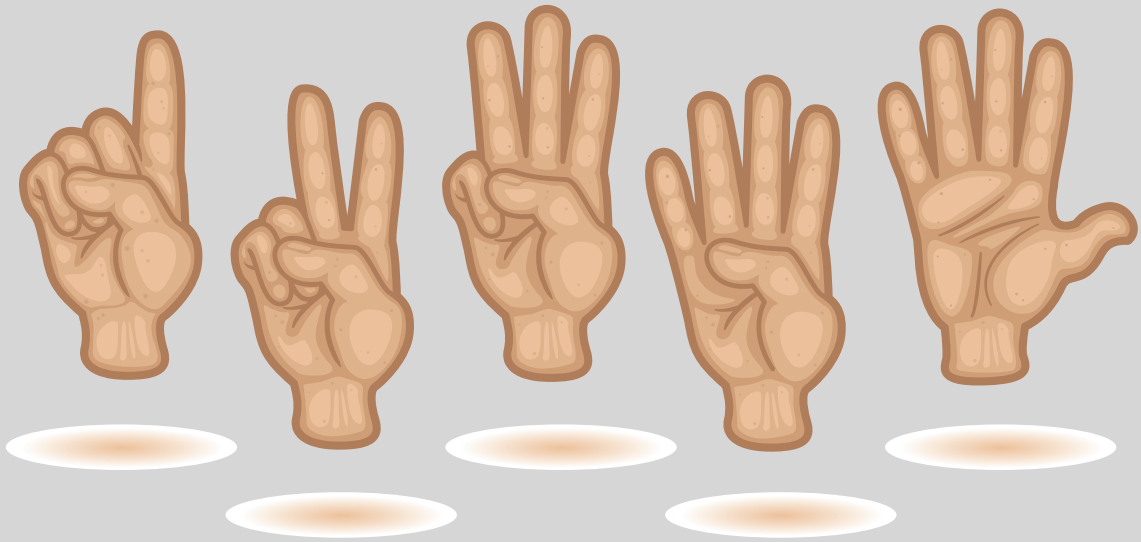
Participatory Decision Making



Encourage full participation
Promote mutual understanding
Foster inclusive solutions
Cultivate shared responsibility



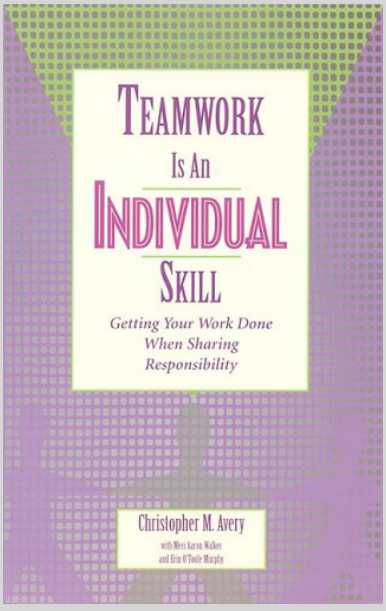
Fist of Five



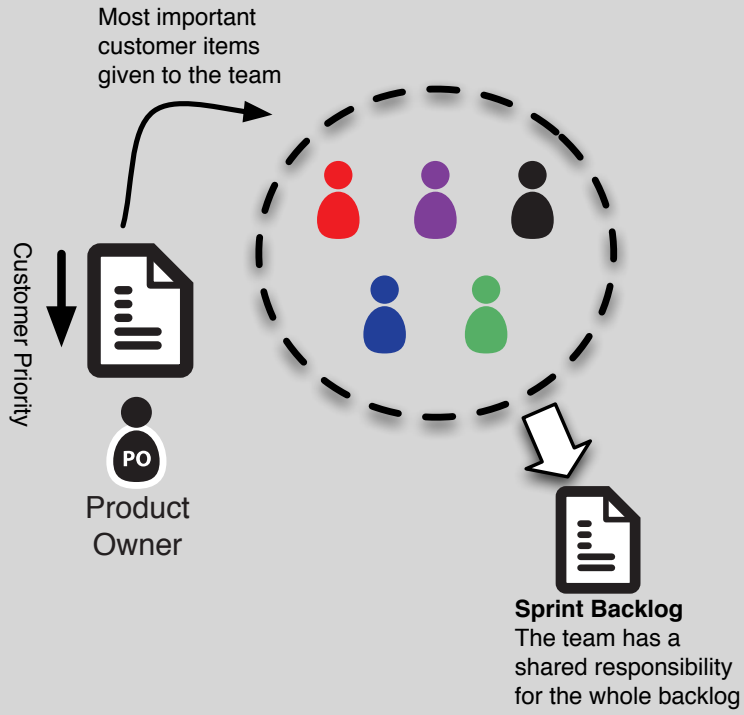
Exercise:
Create Working Agreement

Responsible team member

Personal Agility



Skills != Need



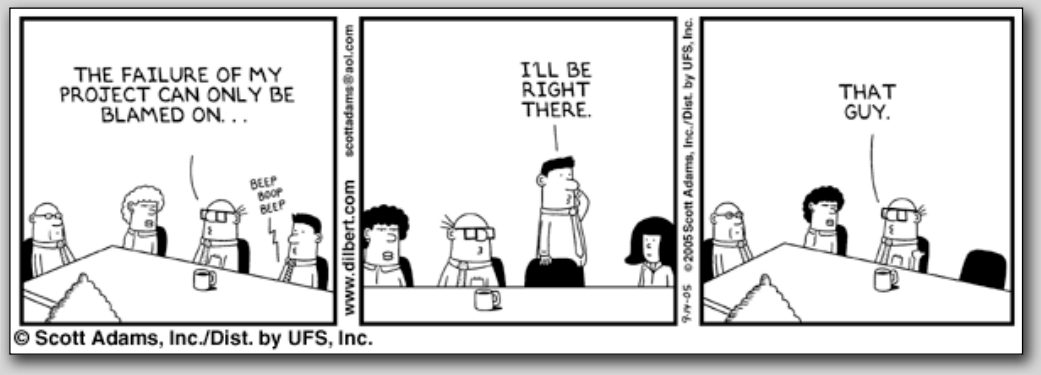
Generalizing Specialist

- Dilemma?
  - Deep expertise in a particular area
  - Experience in a large number of areas
- Generalizing specialist
  - Willing and able to learn new skills
  - Primary speciality, secondary speciality
- Benefits
  - More valuable member in self-organizing team
  - Continuous learning creates virtuous cycle

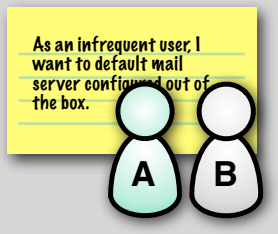
Collective Code Ownership

What is Collective Code Ownership?

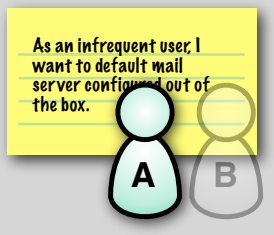
- Anyone can make necessary changes anywhere
- Expects everyone to fix any problems they find
- Everyone shares the responsibility for the quality of the code
- Everyone communicates through code



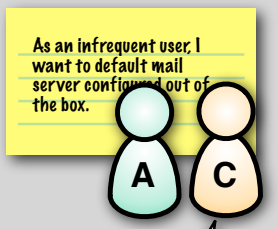
Scenario



A user story is sign up by 2 team members.

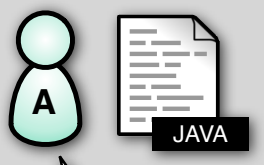


Nearing to the end of Sprint, one of them got sick and was absent.



I'm not familiar with this codebase, but I'll help so that the highest priority work is done.

Another Scenario

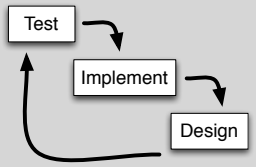


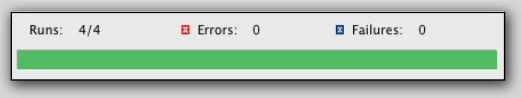
Commit!

Hey, there is a problem here. Let me fix it.

Commit!

How to make it work?





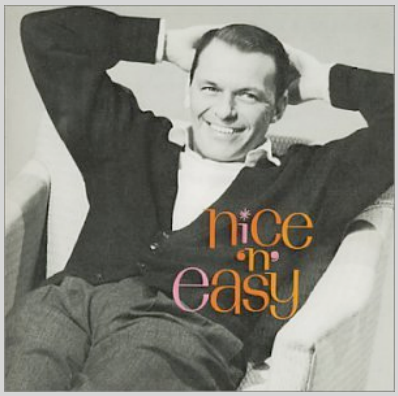
Test Driven Development

produce automated tests

You'll be dead changing without a safety net.

How to make it work?

Refactoring



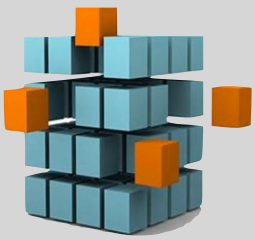
produces code that is healthier and easier to integrate.

How to make it work?

Pair program with someone familiar with the code

Benefits of Collective Code Ownership

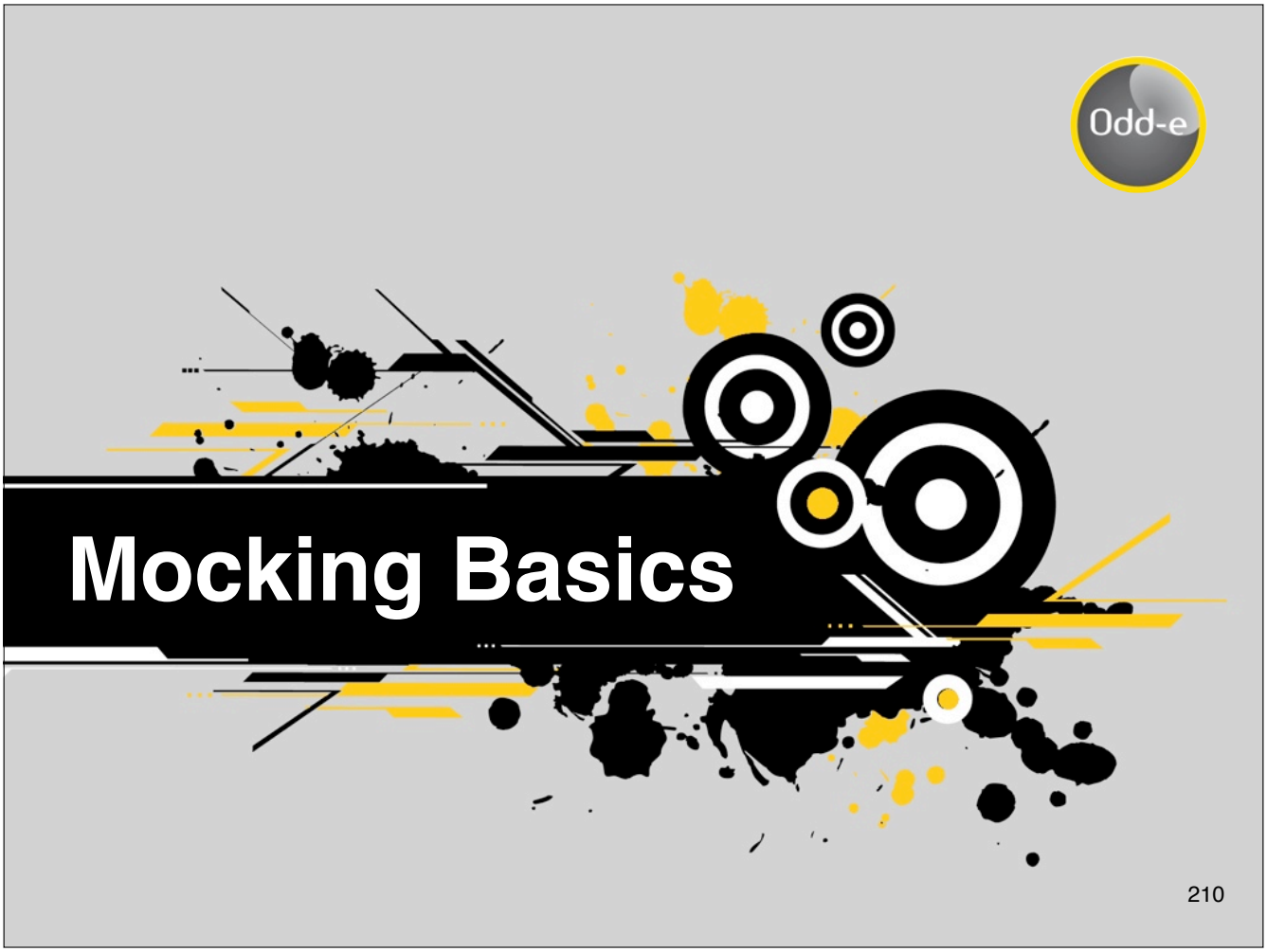
Reduce complex code



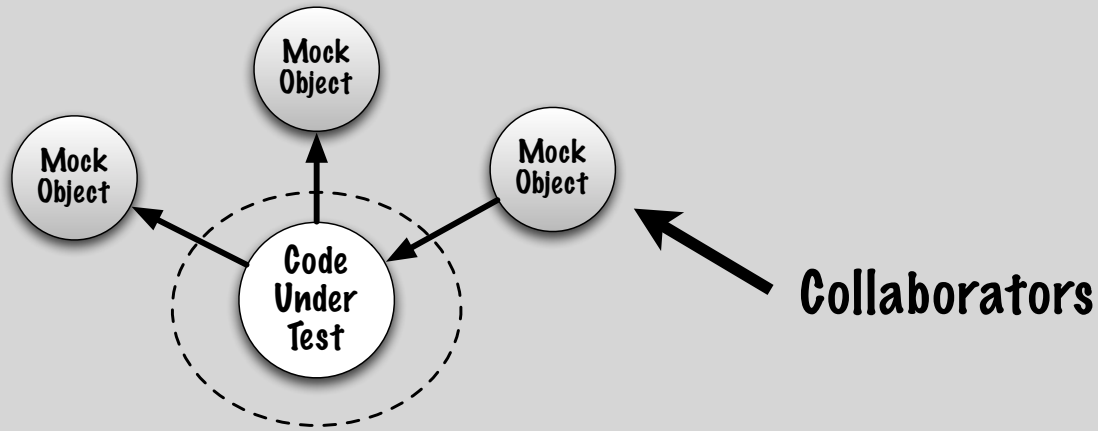
Lesser frustration, never stuck with someone else's stupidity

Spread knowledge


Odd-e
Mocking


Odd-e
Mocking Basics

What is mocking?



Use mock objects to substitute the real collaborators

Why mocking?

Isolate the code under test

Speeds up test execution

What do we do if the code under test depends on collaborators?

Test it together with the other classes it depends on.

or

Isolate it from other classes so that we can test just that class.

We choose to isolate the code under test because...

it depends on classes that has side effects (e.g. network connection, database connection).

we have no control over the source code of 3rd party libraries it is collaborating.

Speeds up test execution

Some real collaborators are expensive to create/execute.
E.g. resource intensive algorithm, complex creation.

Mock objects help return canned results which can make the tests run much faster.

You can mock by hand

```
@Test
public void sendEmailOnMakingPayment() throws Exception {
    MailServiceStub mailService = new MailServiceStub();
    ShoppingCart shoppingCart = buildShoppingCart();

    shoppingCart.setMailService(mailService);
    shoppingCart.makePayment(paymentMethod);

    assertEquals(1, mailService.numberOfMessagesSent());
}

public interface MailService {
    public void send(Message msg);
}
public class MailServiceStub implements MailService {
    List<Message> list = new ArrayList<Message>();
    public void send(Message msg) {
        list.add(msg);
    }
    public int numberOfMessagesSent() {
        return list.size();
    }
}
```

execution

verify

write stubs

Or use a mocking framework


```
private final Mockery context = new JUnit4Mockery();

@Test
public void sendEmailOnMakingPayment() throws Exception {
    final MailService mailService = context.mock(MailService.class);

    ShoppingCart shoppingCart = buildShoppingCart();

    shoppingCart.setMailService(mailService);

    context.checking(new Expectations() {
        {
            mailService.send(with(any(Message.class)));
        }
    });

    shoppingCart.makePayment(paymentMethod);
}
```

mock the object

setup expectations to verify

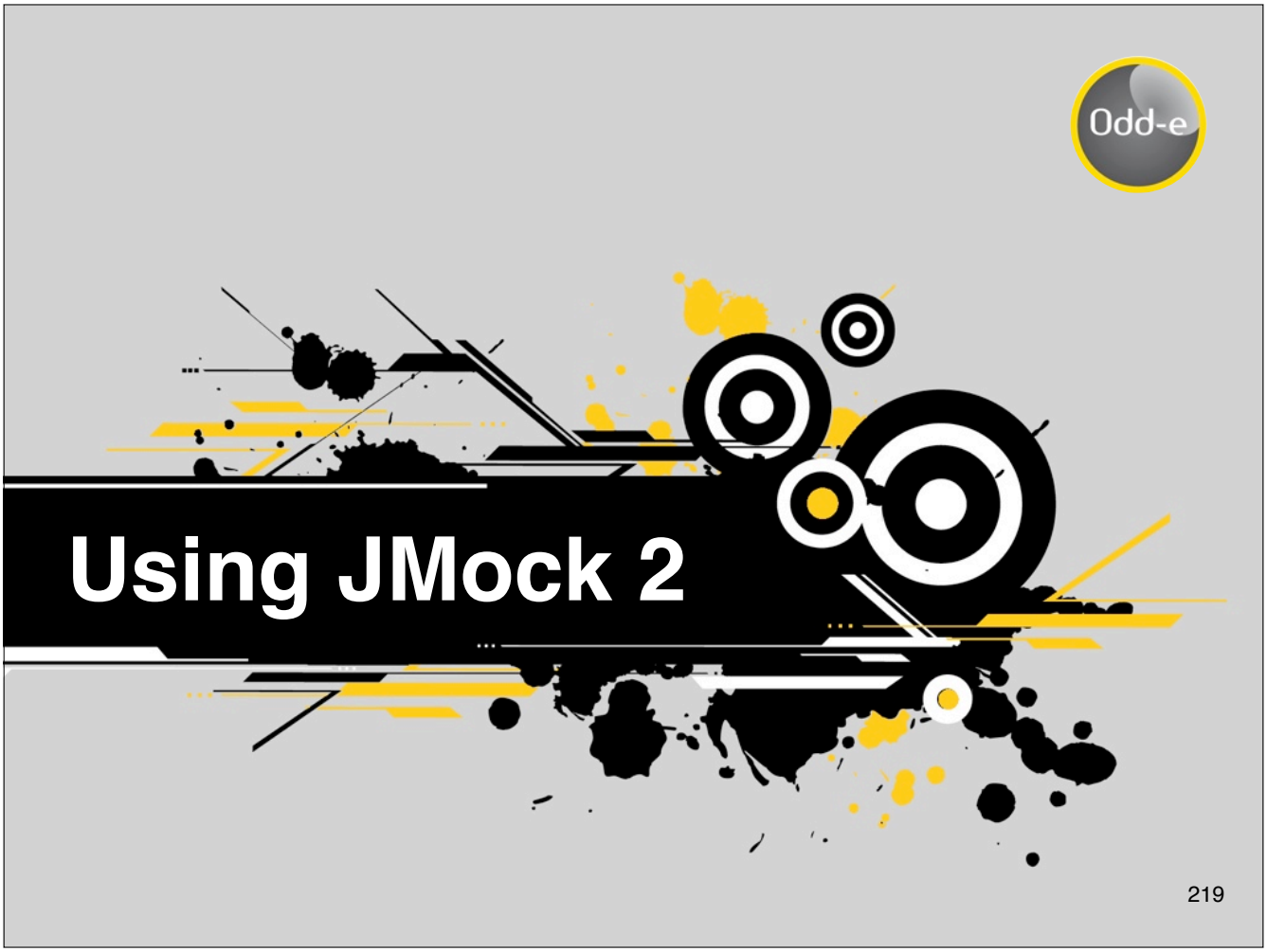
execution

Also known as interaction-based testing

How to mock properly

DO NOT re-implement the collaborator

DO NOT mock value objects



Odd-e

Main concepts of JMock

Mockery	the context of code under test
Mock Objects	substitutes for the real collaborators
Expectations	describes how the code under test should invoke its collaborators during execution

Main concepts of JMock

```
private final Mockery context = new JUnit4Mockery();

@Test
public void sendEmailOnMakingPayment() throws Exception {
    final MailService mailService = context.mock(MailService.class);

    ShoppingCart shoppingCart = buildShoppingCart();

    shoppingCart.setMailService(mailService);

    context.checking(new Expectations() {
        {
            mailService.send(with(any(Message.class)));
        }
    });

    shoppingCart.makePayment(paymentMethod);
}
```

Mockery

Mock Object

Expectations

Expectations format

```
invocation-count (mock-object).method(argument-
constraints);
    inSequence(sequence-name);
    when(state-machine.is(state-name));
    will(action);
    then(state-machine.is(new-state-name));
```

Expectations examples

Expects the person only walks 10 steps.

```
oneOf (person).walkNumberOfSteps(10);
```

Expects the person will jump at least 2 times.

```
atLeast(2).of (person).jump();
```

Expect a call and return 30.

```
oneOf (person).age(); will(returnValue(30));
```


More JMock

Getting started

http://www.jmock.org/getting-started.html

Cheat sheet

http://www.jmock.org/cheat-sheet.html

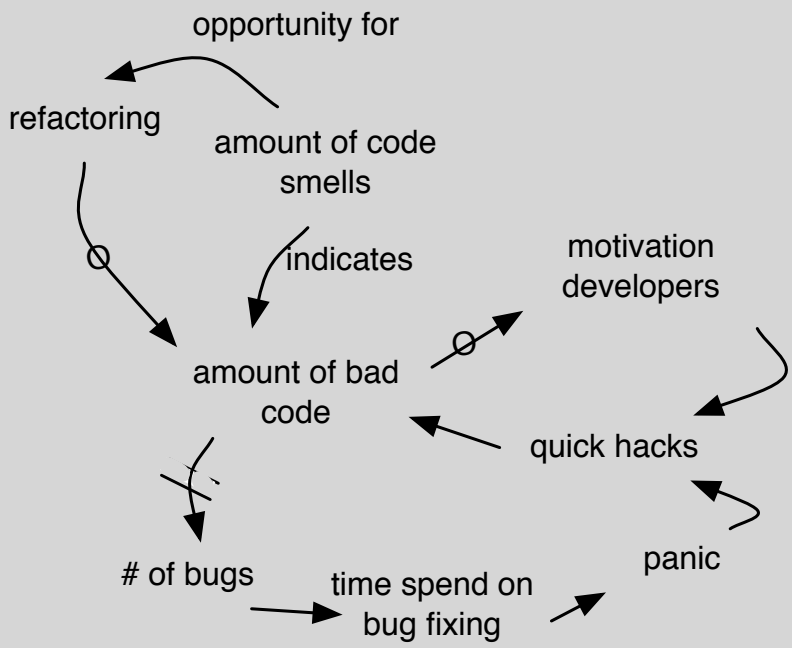
Cookbook

http://www.jmock.org/cookbook.html

Code Smells & Refactoring

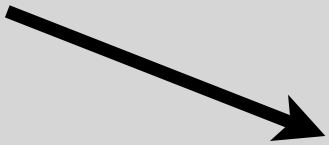
Refactor based on smells

Why? How?



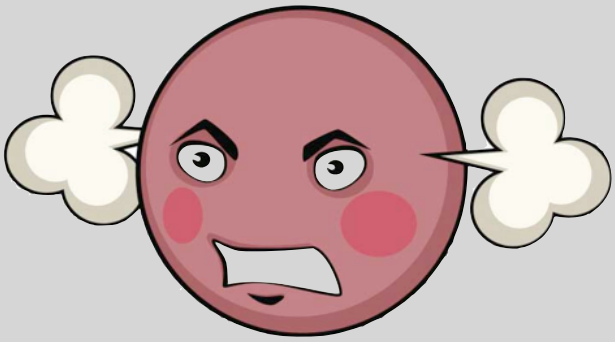
Refactoring

How do you know, when to refactor?



When your code smells!

Good code an opinion?



NO!

Good code does
not smell!

What about performance?

Not an excuse to
write smelly code.

Messy - Debug Info


```
      public long getAmount(Date startDate, Date endDate){
            if (startDate.after(endDate))
                  throw new RuntimeException("Start date cannot be before End Date!");

            Logger.getLogger(this.getClass().getName()).info("Starting to calculate the budgeted amount for " + getFullName() + " between " + star
and " + endDate + ".");

            //If Start and End are in the same budget period
            if (getBudgetPeriodType().getStartOfBudgetPeriod(startDate).equals(
                        getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
//                Logger.getLogger().info("Start Date and End Date are in the same period.");
                  long amount = getAmount(startDate);
//                Logger.getLogger().info("Amount = " + amount);
                  long daysInPeriod = getBudgetPeriodType().getDaysInPeriod(startDate);
//                Logger.getLogger().info("Days in Period = " + daysInPeriod);
                  long daysBetween = DateUtil.getDaysBetween(startDate, endDate, true);
//                Logger.getLogger().info("Days Between = " + daysBetween);

//                Logger.getLogger().info("Returning " + (long) (((double) amount / (double) daysInPeriod) * daysBetween));
//                Logger.getLogger().info("Finished calculating the budget amount.\n\n");
                  return (long) (((double) amount / (double) daysInPeriod) * daysBetween);
            }

            //If the area between Start and End overlap at least two budget periods.
            if (getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1).equals(
                        getBudgetPeriodType().getStartOfBudgetPeriod(endDate))
                        || getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1).before(
                                    getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
//                Logger.getLogger().info("Start Date and End Date are in different budget periods.");
                  long amountStartPeriod = getAmount(startDate);
//                Logger.getLogger().info("Amount Start Period = " + amountStartPeriod);
                  long daysInStartPeriod = getBudgetPeriodType().getDaysInPeriod(startDate);
//                Logger.getLogger().info("Days in Start Period = " + daysInStartPeriod);
                  long daysAfterStartDateInStartPeriod = DateUtil.getDaysBetween(startDate, getBudgetPeriodType().getEndOfBudgetPeriod(startDate),
//                Logger.getLogger().info("Days After Start Date in Start Period = " + daysAfterStartDateInStartPeriod);
                  double totalStartPeriod = (((double) amountStartPeriod / (double) daysInStartPeriod) * daysAfterStartDateInStartPeriod);
//                Logger.getLogger().info("Total in Start Period = " + totalStartPeriod);

                  double totalInMiddle = 0;
                  for (String periodKey : getBudgetPeriods(
                              getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1),
                              getBudgetPeriodType().getBudgetPeriodOffset(endDate, -1))) {
                        totalInMiddle += getAmount(getPeriodDate(periodKey));
                        Logger.getLogger(this.getClass().getName()).info("Added " + getAmount(getPeriodDate(periodKey)) + " to total for one period
between; current value is " + totalInMiddle);
                  }
//                Logger.getLogger().info("Total in Middle = " + totalInMiddle);

                  long amountEndPeriod = getAmount(endDate);
//                Logger.getLogger().info("Amount End Period = " + amountEndPeriod);
```

Long Method

Large number of line of code (anything over 5-10)

Too long!

```
public long getAmount(Date startDate, Date endDate){
      if (startDate.after(endDate))
            throw new RuntimeException("Start date cannot be before End Date!");

      //If Start and End are in the same budget period
      if (getBudgetPeriodType().getStartOfBudgetPeriod(startDate).equals(
                  getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
            long amount = getAmount(startDate);
            long daysInPeriod = getBudgetPeriodType().getDaysInPeriod(startDate);
            long daysBetween = DateUtil.getDaysBetween(startDate, endDate, true);

            return (long) (((double) amount / (double) daysInPeriod) * daysBetween);
      }

      //If the area between Start and End overlap at least two budget periods.
      if (getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1).equals(
                  getBudgetPeriodType().getStartOfBudgetPeriod(endDate))
                  || getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1).before(
                              getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
            long amountStartPeriod = getAmount(startDate);
            long daysInStartPeriod = getBudgetPeriodType().getDaysInPeriod(startDate);
            long daysAfterStartDateInStartPeriod = DateUtil.getDaysBetween(startDate, getBudgetPeriodType().getEndOfBudgetPeriod(startDate), true);
            double totalStartPeriod = (((double) amountStartPeriod / (double) daysInStartPeriod) * daysAfterStartDateInStartPeriod);

            double totalInMiddle = 0;
            for (String periodKey : getBudgetPeriods(
                        getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1),
                        getBudgetPeriodType().getBudgetPeriodOffset(endDate, -1))) {
                  totalInMiddle += getAmount(getPeriodDate(periodKey));
            }

            long amountEndPeriod = getAmount(endDate);
            long daysInEndPeriod = getBudgetPeriodType().getDaysInPeriod(endDate);
            long daysBeforeEndDateInEndPeriod = DateUtil.getDaysBetween(getBudgetPeriodType().getStartOfBudgetPeriod(endDate), endDate, true);
            double totalEndPeriod = (long) (((double) amountEndPeriod / (double) daysInEndPeriod) * daysBeforeEndDateInEndPeriod);

            return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
      }

      throw new RuntimeException("You should not be here.  We have returned all legitimate numbers from getAmount(Date, Date) in BudgetCategoryImpl.
Please contact Wyatt Olson with details on how you got here (what steps did you perform in Buddi to get this error message).");
}
```

Long method
often indicates
other smells

Safety first!

Check if the test quality

```
public class BudgetCategoryTest {

	@Test
	public void testBudgetCategory(){
		try {
			BudgetCategoryType bct = ModelFactory.getBudgetCategoryType(BudgetCategoryTypes.BUDGET_CATEGORY_TYPE_MONTH);
			BudgetCategory bc = ModelFactory.createBudgetCategory("Test", bct, false);
			bc.setAmount(DateUtil.getDate(2007, Calendar.APRIL, 1), 100);
			bc.setAmount(DateUtil.getDate(2007, Calendar.MAY, 1), 200);
			bc.setAmount(DateUtil.getDate(2007, Calendar.JUNE, 1), 240);
			bc.setAmount(DateUtil.getDate(2007, Calendar.JULY, 1), 10);
			bc.setAmount(DateUtil.getDate(2007, Calendar.AUGUST, 1), 130);
			bc.setAmount(DateUtil.getDate(2007, Calendar.SEPTEMBER, 1), 13);
			bc.setAmount(DateUtil.getDate(2007, Calendar.OCTOBER, 1), 333);
			bc.setAmount(DateUtil.getDate(2007, Calendar.NOVEMBER, 1), 331);

			assertEquals((double) 100, bc.getAmount(DateUtil.getDate(2007, Calendar.APRIL, 1)), 1);
			assertEquals((double) 100, bc.getAmount(DateUtil.getDate(2007, Calendar.APRIL, 10)), 1);
			assertEquals((double) 100, bc.getAmount(DateUtil.getDate(2007, Calendar.APRIL, 28)), 1);

			assertEquals((double) 300, bc.getAmount(DateUtil.getDate(2007, Calendar.APRIL, 1), DateUtil.getDate(2007, Calendar.MAY, 31)), 1);
			assertEquals((double) 149, bc.getAmount(DateUtil.getDate(2007, Calendar.APRIL, 15), DateUtil.getDate(2007, Calendar.MAY, 15)), 1);

		}
		catch (Exception e){
			fail("Exception: " + e);
		}
	}

	@Test
	public void budgetPeriodWeekly() throws Exception {
		BudgetCategoryType bct = ModelFactory.getBudgetCategoryType(BudgetCategoryTypes.BUDGET_CATEGORY_TYPE_WEEK);
		BudgetCategory bc = ModelFactory.createBudgetCategory("Weekly Test", bct, false);

		bc.setAmount(DateUtil.getDate(2007, Calendar.SEPTEMBER, 10), 100);
		bc.setAmount(DateUtil.getDate(2007, Calendar.SEPTEMBER, 17), 100);
		bc.setAmount(DateUtil.getDate(2007, Calendar.SEPTEMBER, 24), 100);
		bc.setAmount(DateUtil.getDate(2007, Calendar.OCTOBER, 1), 100);
		bc.setAmount(DateUtil.getDate(2007, Calendar.OCTOBER, 8), 100);
		bc.setAmount(DateUtil.getDate(2007, Calendar.OCTOBER, 15), 100);
		bc.setAmount(DateUtil.getDate(2007, Calendar.OCTOBER, 22), 100);
		bc.setAmount(DateUtil.getDate(2007, Calendar.OCTOBER, 29), 100);
		bc.setAmount(DateUtil.getDate(2007, Calendar.NOVEMBER, 5), 100);
		bc.setAmount(DateUtil.getDate(2007, Calendar.NOVEMBER, 12), 100);

		assertEquals(100l,
				bc.getAmount(DateUtil.getDate(2007, Calendar.OCTOBER, 1)));

		assertEquals(14l,
				bc.getAmount(
						DateUtil.getDate(2007, Calendar.OCTOBER, 1),
						DateUtil.getDate(2007, Calendar.OCTOBER, 1)));
		assertEquals(28l,
				bc.getAmount(
						DateUtil.getDate(2007, Calendar.OCTOBER, 1),
						DateUtil.getDate(2007, Calendar.OCTOBER, 2)));
		assertEquals(100l,
				bc.getAmount(
						bct.getStartOfBudgetPeriod(DateUtil.getDate(2007, Calendar.OCTOBER, 1)),
						bct.getEndOfBudgetPeriod(DateUtil.getDate(2007, Calendar.OCTOBER, 1))));
		assertEquals(200l,
				bc.getAmount(
						bct.getStartOfBudgetPeriod(DateUtil.getDate(2007, Calendar.OCTOBER, 1)),
						bct.getEndOfBudgetPeriod(bct.getBudgetPeriodOffset(DateUtil.getDate(2007, Calendar.OCTOBER, 1), 1))));
		assertEquals(400l,
				bc.getAmount(
						bct.getStartOfBudgetPeriod(DateUtil.getDate(2007, Calendar.OCTOBER, 1)),
						bct.getEndOfBudgetPeriod(bct.getBudgetPeriodOffset(DateUtil.getDate(2007, Calendar.OCTOBER, 1), 3))));
		assertEquals(442l,
				bc.getAmount(
						DateUtil.getDate(2007, Calendar.OCTOBER, 1),
```

Some test,
not very good ones.

Add tests

```
@Test (expected=RuntimeException.class)
public void testBeginDateIsLaterThanStartDateThrowsAnException() throws Exception {
	bc.getAmount(fourthOfJuly2011 , secondOfJuly2011);
}

@Test
public void getAmountOfARangeInsideABudgetPeriod() throws Exception {
	bc.setAmount(secondOfJuly2011, 31);
	assertEquals(3, bc.getAmount(secondOfJuly2011, fourthOfJuly2011));
}

@Test
public void getAmountOfARangeTwoNeigbouringBudgetPeriods() throws Exception {
	bc.setAmount(twentyJune2011, 300);
	bc.setAmount(secondOfJuly2011, 31);
	assertEquals(114, bc.getAmount(twentyJune2011, fourthOfJuly2011));
}

@Test
public void getAmountOfARangeWithManyBudgetPeriodsInBetween() throws Exception {
	bc.setAmount(tenthMarch2011, 31);
	bc.setAmount(twentyApril2011, 3000);
	bc.setAmount(twentyMay2011, 3100);
	bc.setAmount(twentyJune2011, 3000);
	bc.setAmount(secondOfJuly2011, 31);
	assertEquals(9100 + 2 + 22, bc.getAmount(tenthMarch2011, secondOfJuly2011));
}
```

Buckled? Ready?

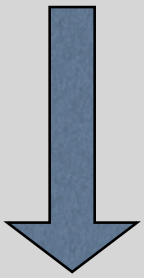
Duplicate code

Two pieces of code that are conceptually the same

```
//If Start and End are in the same budget period
if (getBudgetPeriodType().getStartOfBudgetPeriod(startDate).equals(
        getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
    long amount = getAmount(startDate);
    long daysInPeriod = getBudgetPeriodType().getDaysInPeriod(startDate);
    long daysBetween = DateUtil.getDaysBetween(startDate, endDate, true);

    return (long) (((double) amount / (double) daysInPeriod) * daysBetween);
}

//If the area between Start and End overlap at least two budget periods.
if (getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1).equals(
        getBudgetPeriodType().getStartOfBudgetPeriod(endDate))
        || getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1).before(
                getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
    long amountStartPeriod = getAmount(startDate);
    long daysInStartPeriod = getBudgetPeriodType().getDaysInPeriod(startDate);
    long daysAfterStartDateInStartPeriod = DateUtil.getDaysBetween(startDate,
        getBudgetPeriodType().getEndOfBudgetPeriod(startDate), true);
    double totalStartPeriod =
        (((double) amountStartPeriod / (double) daysInStartPeriod) * daysAfterStartDateInStartPeriod);

    double totalInMiddle = 0;
    for (String periodKey : getBudgetPeriods(
            getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1),
            getBudgetPeriodType().getBudgetPeriodOffset(endDate, -1))) {
        totalInMiddle += getAmount(getPeriodDate(periodKey));
    }

    long amountEndPeriod = getAmount(endDate);
    long daysInEndPeriod = getBudgetPeriodType().getDaysInPeriod(endDate);
    long daysBeforeEndDateInEndPeriod =
        DateUtil.getDaysBetween(getBudgetPeriodType().getStartOfBudgetPeriod(endDate), endDate, true);
    double totalEndPeriod =
        (long) (((double) amountEndPeriod / (double) daysInEndPeriod) * daysBeforeEndDateInEndPeriod);

    return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
}
```


First make duplication obvious (same)

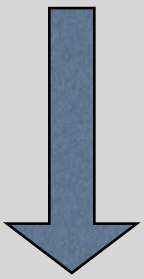
```
long amount = getAmount(startDate);
long daysInPeriod = getBudgetPeriodType().getDaysInPeriod(startDate);
long daysBetween = DateUtil.getDaysBetween(startDate, endDate, true);

return (long) (((double) amount / (double) daysInPeriod) * daysBetween);
```



Extract Local Variable (Alt-Shift-L)

```
long amount = getAmount(startDate);
long daysInPeriod = getBudgetPeriodType().getDaysInPeriod(startDate);
long daysBetween = DateUtil.getDaysBetween(startDate, endDate, true);
double amountInPeriod = (long) (((double) amount / (double) daysInPeriod) * daysBetween);

return (long) amountInPeriod;
```

```
long amountStartPeriod = getAmount(startDate);
long daysInStartPeriod = getBudgetPeriodType().getDaysInPeriod(startDate);
long daysAfterStartDateInStartPeriod =
    DateUtil.getDaysBetween(startDate, getBudgetPeriodType().getEndOfBudgetPeriod(startDate), true);
double totalStartPeriod =
    (((double) amountStartPeriod / (double) daysInStartPeriod) * daysAfterStartDateInStartPeriod);
```



Extract Local Variable (Alt-Shift-L)

```
Date endOfBudgetPeriod = getBudgetPeriodType().getEndOfBudgetPeriod(startDate);
long amountStartPeriod = getAmount(startDate);
long daysInStartPeriod = getBudgetPeriodType().getDaysInPeriod(startDate);
long daysAfterStartDateInStartPeriod = DateUtil.getDaysBetween(startDate, endOfBudgetPeriod, true);
double totalStartPeriod =
    (((double) amountStartPeriod / (double) daysInStartPeriod) * daysAfterStartDateInStartPeriod);
```


Remove useless parenthesis

```
double totalStartPeriod =
    (((double) amountStartPeriod / (double) daysInStartPeriod) * daysAfterStartDateInStartPeriod);
```

```
double totalStartPeriod =
    ((double) amountStartPeriod / (double) daysInStartPeriod) * daysAfterStartDateInStartPeriod;
```

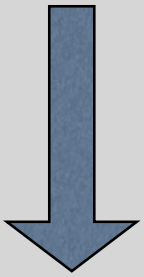
```
double totalEndPeriod =
    (long) (((double) amountEndPeriod / (double) daysInEndPeriod) * daysBeforeEndDateInEndPeriod);
```

Casts without rounding?

```
double totalEndPeriod =
    ((double) amountEndPeriod / (double) daysInEndPeriod) * daysBeforeEndDateInEndPeriod;
```

Without this, Eclipse can't detect the duplication :(

```
long amountEndPeriod = getAmount(endDate);
long daysInEndPeriod = getBudgetPeriodType().getDaysInPeriod(endDate);
long daysBeforeEndDateInEndPeriod =
    DateUtil.getDaysBetween(getBudgetPeriodType().getStartOfBudgetPeriod(endDate), endDate, true);
double totalEndPeriod =
    (long) ((double) amountEndPeriod / (double) daysInEndPeriod) * daysBeforeEndDateInEndPeriod;
```



Extract Local Variable (Alt-Shift-L)

```
Date startOfBudgetPeriod = getBudgetPeriodType().getStartOfBudgetPeriod(endDate);
long amountEndPeriod = getAmount(startOfBudgetPeriod);
long daysInEndPeriod = getBudgetPeriodType().getDaysInPeriod(startOfBudgetPeriod);
long daysBeforeEndDateInEndPeriod =
    DateUtil.getDaysBetween(startOfBudgetPeriod, endDate, true);
double totalEndPeriod =
    (long) ((double) amountEndPeriod / (double) daysInEndPeriod) * daysBeforeEndDateInEndPeriod;
```


Use start instead of end of period.

```
long amountEndPeriod = getAmount(endDate);
long daysInEndPeriod = getBudgetPeriodType().getDaysInPeriod(endDate);
long daysBeforeEndDateInEndPeriod = DateUtil.getDaysBetween(startOfBudgetPeriod, endDate, true);
```

```
long amountEndPeriod = getAmount(startOfBudgetPeriod);
long daysInEndPeriod = getBudgetPeriodType().getDaysInPeriod(startOfBudgetPeriod);
long daysBeforeEndDateInEndPeriod = DateUtil.getDaysBetween(startOfBudgetPeriod, endDate, true);
```

Note: Parameter of getAmount and getDaysInPeriod is a Period, not a Date.

Therefore both start and end date of period will work!

Duplication is exactly the same now.

```
if (getBudgetPeriodType().getStartOfBudgetPeriod(startDate).equals(
          getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
     long amount = getAmount(startDate);
     long daysInPeriod = getBudgetPeriodType().getDaysInPeriod(startDate);
     long daysBetween = DateUtil.getDaysBetween(startDate, endDate, true);
     double amountInPeriod = ((double) amount / (double) daysInPeriod) * daysBetween;
     return (long) amountInPeriod;
}

//If the area between Start and End overlap at least two budget periods.
if (getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1).equals(
          getBudgetPeriodType().getStartOfBudgetPeriod(endDate))
          || getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1).before(
                    getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
     Date endOfBudgetPeriod = getBudgetPeriodType().getEndOfBudgetPeriod(startDate);
     long amountStartPeriod = getAmount(startDate);
     long daysInStartPeriod = getBudgetPeriodType().getDaysInPeriod(startDate);
     long daysAfterStartDateInStartPeriod = DateUtil.getDaysBetween(startDate, endOfBudgetPeriod, true);
     double totalStartPeriod = ((double) amountStartPeriod / (double) daysInStartPeriod) * daysAfterStartDateInStartPeriod;

     double totalInMiddle = 0;
     for (String periodKey : getBudgetPeriods(
               getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1),
               getBudgetPeriodType().getBudgetPeriodOffset(endDate, -1))) {
          totalInMiddle += getAmount(getPeriodDate(periodKey));
     }

     Date startOfBudgetPeriod = getBudgetPeriodType().getStartOfBudgetPeriod(endDate);
     long amountEndPeriod = getAmount(startOfBudgetPeriod);
     long daysInEndPeriod = getBudgetPeriodType().getDaysInPeriod(startOfBudgetPeriod);
     long daysBeforeEndDateInEndPeriod = DateUtil.getDaysBetween(startOfBudgetPeriod, endDate, true);
     double totalEndPeriod = ((double) amountEndPeriod / (double) daysInEndPeriod) * daysBeforeEndDateInEndPeriod;
     return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
}
```


Then extract duplication

Extract Method
Alt-Shift-M

```
//If Start and End are in the same budget period
if (getBudgetPeriodType().getStartOfBudgetPeriod(startDate).equals(
          getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
     double amountInPeriod = getAmountInPeriod(startDate, endDate);
     return (long) amountInPeriod;
}

//If the area between Start and End overlap at least two budget periods.
if (getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1).equals(
          getBudgetPeriodType().getStartOfBudgetPeriod(endDate))
          || getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1).before(
                    getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
     Date endOfBudgetPeriod = getBudgetPeriodType().getEndOfBudgetPeriod(startDate);
     double totalStartPeriod = getAmountInPeriod(startDate, endOfBudgetPeriod);

     double totalInMiddle = 0;
     for (String periodKey : getBudgetPeriods(
               getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1),
               getBudgetPeriodType().getBudgetPeriodOffset(endDate, -1))) {
          totalInMiddle += getAmount(getPeriodDate(periodKey));
     }

     Date startOfBudgetPeriod = getBudgetPeriodType().getStartOfBudgetPeriod(endDate);
     double totalEndPeriod = getAmountInPeriod(startOfBudgetPeriod, endDate);
     return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
}
```

Good Principles

DRY
Don't Repeat Yourself

Once And Only Once

DO NOT

COPY PASTE

Magic Numbers

A constant appears in the code

1 - 1

```
//If the area between Start and End overlap at least two budget periods.
if (getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1).equals(
        getBudgetPeriodType().getStartOfBudgetPeriod(endDate))
        || getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1).before(
                getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
    Date endOfBudgetPeriod = getBudgetPeriodType().getEndOfBudgetPeriod(startDate);
    double totalStartPeriod = getAmountInPeriod(startDate, endOfBudgetPeriod);

    double totalInMiddle = 0;
    for (String periodKey : getBudgetPeriods(
            getBudgetPeriodType().getBudgetPeriodOffset(startDate, 1),
            getBudgetPeriodType().getBudgetPeriodOffset(endDate, -1))) {
        totalInMiddle += getAmount(getPeriodDate(periodKey));
    }

    Date startOfBudgetPeriod = getBudgetPeriodType().getStartOfBudgetPeriod(endDate);
    double totalEndPeriod = getAmountInPeriod(startOfBudgetPeriod, endDate);
    return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
}
```

Extract to method

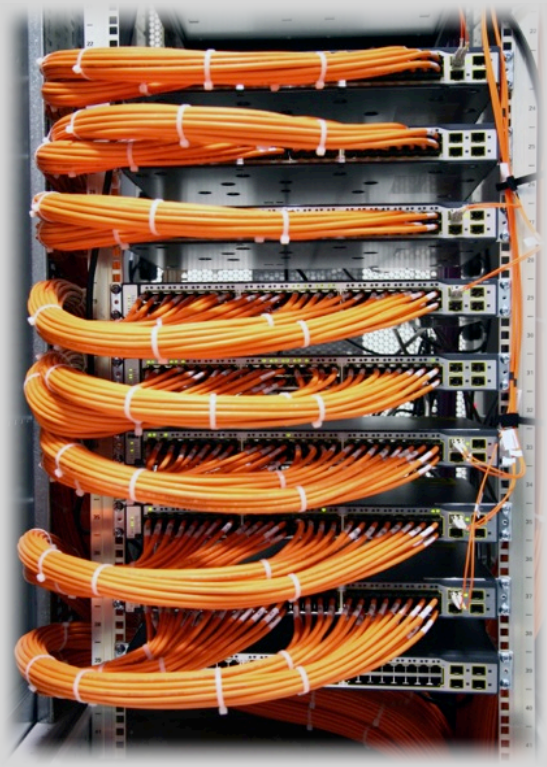
```
//If the area between Start and End overlap at least two budget periods.
if (getBudgetPeriodType().getStartOfNextBudgetPeriod(startDate).equals(
        getBudgetPeriodType().getStartOfBudgetPeriod(endDate))
        || getBudgetPeriodType().getStartOfNextBudgetPeriod(startDate).before(
                getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
    Date endOfBudgetPeriod = getBudgetPeriodType().getEndOfBudgetPeriod(startDate);
    double totalStartPeriod = getAmountInPeriod(startDate, endOfBudgetPeriod);

    double totalInMiddle = 0;
    for (String periodKey : getBudgetPeriods(
            getBudgetPeriodType().getStartOfNextBudgetPeriod(startDate),
            getBudgetPeriodType().getStartOfPreviousBudgetPeriod(startDate))) {
        totalInMiddle += getAmount(getPeriodDate(periodKey));
    }

    Date startOfBudgetPeriod = getBudgetPeriodType().getStartOfBudgetPeriod(endDate);
    double totalEndPeriod = getAmountInPeriod(startOfBudgetPeriod, endDate);
    return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
}
```


Additional cleanup

```
//If Start and End are in the same budget period
if (getBudgetPeriodType().getStartOfBudgetPeriod(startDate).equals(
        getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
    return (long) getAmountInPeriod(startDate, endDate);
}

//If the area between Start and End overlap at least two budget periods.
if (getBudgetPeriodType().getStartOfNextBudgetPeriod(startDate).equals(
        getBudgetPeriodType().getStartOfBudgetPeriod(endDate))
        || getBudgetPeriodType().getStartOfNextBudgetPeriod(startDate).before(
                getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
    double totalStartPeriod = getAmountInPeriod(startDate, getBudgetPeriodType().getEndOfBudgetPeriod(startDate));

    double totalInMiddle = 0;
    for (String periodKey : getBudgetPeriods(
            getBudgetPeriodType().getStartOfNextBudgetPeriod(startDate),
            getBudgetPeriodType().getStartOfPreviousBudgetPeriod(endDate))) {
        totalInMiddle += getAmount(getPeriodDate(periodKey));
    }

    double totalEndPeriod = getAmountInPeriod(getBudgetPeriodType().getStartOfBudgetPeriod(endDate), endDate);
    return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
}
```

Inline Temp - Alt-Shift-I

Data Clumps



A couple of items frequently appear together


```
    public long getAmount(Date startDate, Date endDate){
        if (startDate.after(endDate))
            throw new RuntimeException("Start date cannot be before End Date!");

        //If Start and End are in the same budget period
        if (getBudgetPeriodType().getStartOfBudgetPeriod(startDate).equals(
                getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
            return (long) getAmountInPeriod(startDate, endDate);
        }

        //If the area between Start and End overlap at least two budget periods.
        if (getBudgetPeriodType().getStartOfNextBudgetPeriod(startDate).equals(
                getBudgetPeriodType().getStartOfBudgetPeriod(endDate))
                || getBudgetPeriodType().getStartOfNextBudgetPeriod(startDate).before(
                        getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
            double totalStartPeriod = getAmountInPeriod(startDate,
                    getBudgetPeriodType().getEndOfBudgetPeriod(startDate));

            double totalInMiddle = 0;
            for (String periodKey : getBudgetPeriods(
                    getBudgetPeriodType().getStartOfNextBudgetPeriod(startDate),
                    getBudgetPeriodType().getStartOfPreviousBudgetPeriod(endDate))) {
                totalInMiddle += getAmount(getPeriodDate(periodKey));
            }

            double totalEndPeriod = getAmountInPeriod(getBudgetPeriodType().getStartOfBudgetPeriod(endDate),
                    endDate);
            return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
        }

        throw new RuntimeException("You should not be here.  We have returned all legitimate numbers from getAmount(Dat
BudgetCategoryImpl.  Please contact Wyatt Olson with details on how you got here (what steps did you perform in Buddi to
error message).");
    }
```

Note: Also earlier we seen the smell that Date was used as Period

Add new abstraction

```
public long getAmount(Date startDate, Date endDate){
    if (startDate.after(endDate))
        throw new RuntimeException("Start date cannot be before End Date!");

    Period period = new Period(startDate, endDate);
```

```
public class Period {

    private final Date start;
    private final Date end;

    public Period(Date start, Date end) {
        this.start = start;
        this.end = end;
    }

}
```

Abstraction not yet used...

Use new abstraction

```
        //If Start and End are in the same budget period
        if (getBudgetPeriodType().getStartOfBudgetPeriod(period.getStartDate()).equals(
                getBudgetPeriodType().getStartOfBudgetPeriod(period.getEndDate()))){
            return (long) getAmountInPeriod(period.getStartDate(), period.getEndDate());
        }

        //If the area between Start and End overlap at least two budget periods.
        if (getBudgetPeriodType().getStartOfNextBudgetPeriod(period.getStartDate()).equals(
                getBudgetPeriodType().getStartOfBudgetPeriod(period.getEndDate()))
                || getBudgetPeriodType().getStartOfNextBudgetPeriod(period.getStartDate()).before(
                        getBudgetPeriodType().getStartOfBudgetPeriod(period.getEndDate()))){
            double totalStartPeriod = getAmountInPeriod(period.getStartDate(),
                    getBudgetPeriodType().getEndOfBudgetPeriod(period.getStartDate()));

            double totalInMiddle = 0;
            for (String periodKey : getBudgetPeriods(
                    getBudgetPeriodType().getStartOfNextBudgetPeriod(period.getStartDate()),
                    getBudgetPeriodType().getStartOfPreviousBudgetPeriod(period.getEndDate()))) {
                totalInMiddle += getAmount(getPeriodDate(periodKey));
            }

            double totalEndPeriod = getAmountInPeriod(getBudgetPeriodType().getStartOfBudgetPeriod
(period.getEndDate()),
                    period.getEndDate());
            return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
        }
```

Makes the code uglier....

Extract Method

Extract Method
Alt-Shift-M

```
private long getAmount(Period period) {
    //If Start and End are in the same budget period
    if (getBudgetPeriodType().getStartOfBudgetPeriod(period.getStartDate()).equals(
            getBudgetPeriodType().getStartOfBudgetPeriod(period.getEndDate()))){
        return (long) getAmountInPeriod(period.getStartDate(), period.getEndDate());
    }

    //If the area between Start and End overlap at least two budget periods.
    if (getBudgetPeriodType().getStartOfNextBudgetPeriod(period.getStartDate()).equals(
            getBudgetPeriodType().getStartOfBudgetPeriod(period.getEndDate()))
            || getBudgetPeriodType().getStartOfNextBudgetPeriod(period.getStartDate()).before(
                    getBudgetPeriodType().getStartOfBudgetPeriod(period.getEndDate()))){
        double totalStartPeriod = getAmountInPeriod(period.getStartDate(),
                getBudgetPeriodType().getEndOfBudgetPeriod(period.getStartDate()));

        double totalInMiddle = 0;
        for (String periodKey : getBudgetPeriods(
                getBudgetPeriodType().getStartOfNextBudgetPeriod(period.getStartDate()),
                getBudgetPeriodType().getStartOfPreviousBudgetPeriod(period.getEndDate()))) {
            totalInMiddle += getAmount(getPeriodDate(periodKey));
        }

        double totalEndPeriod = getAmountInPeriod(getBudgetPeriodType().getStartOfBudgetPeriod(period.getEndDate()),
                period.getEndDate());
        return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
    }

    throw new RuntimeException("You should not be here.  We have returned all legitimate numbers from getAmount
(Date, Date) in BudgetCategoryImpl.  Please contact Wyatt Olson with details on how you got here (what steps did you
perform in Buddi to get this error message).");
}
```


And the original method

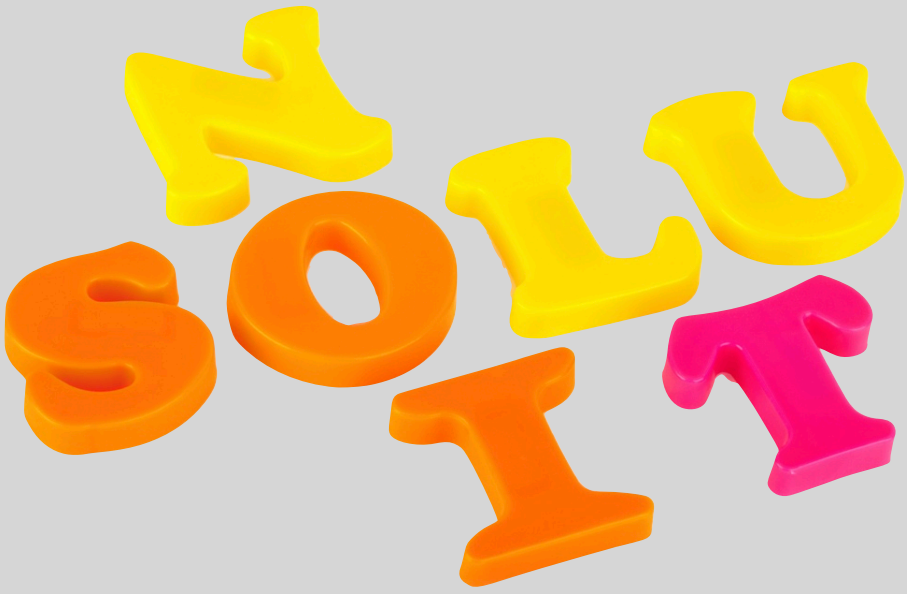
```
public long getAmount(Date startDate, Date endDate){
    if (startDate.after(endDate))
        throw new RuntimeException("Start date cannot be before End Date!");

    return getAmount(new Period(startDate, endDate));
}
```

Requires no changes in the calling code.. yet

Data Clumps and other smells often suggest a common problem:

Missing domain objects!

Uncommunicative Name & Inconsistent Names



Ambiguous names

```
private double getAmountInPeriod(Date startDate, Date endDate) {
    long amount = getAmount(startDate);
    long daysInPeriod = getBudgetPeriodType().getDaysInPeriod(startDate);
    long daysBetween = DateUtil.getDaysBetween(startDate, endDate, true);
    double amountInPeriod = ((double) amount / (double) daysInPeriod) * daysBetween;
    return amountInPeriod;
}
```

The BudgetCategory class now contains:

3 methods called getAmount
1 method called getAmounts
1 method called getAmountInPeriod

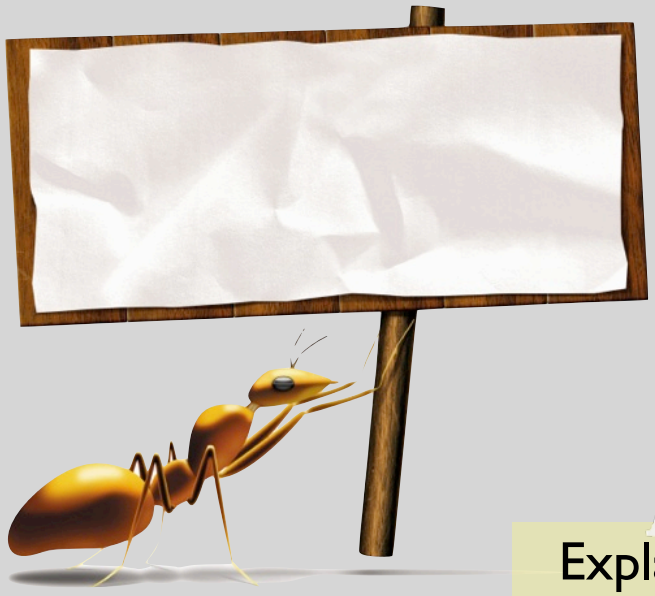
(and we caused 2 of these!)

Rename

```
public Map<String, Long> getAmounts()  ⟶  public Map<String, Long> getBudgetPeriods()

public void setAmounts(Map<String, Long> amounts)  ⟶  public void setBudgetPeriods(Map<String, Long> amounts)

public long getAmount(Date periodDate)  ⟶  public long getAmountFromBudgetPeriodContainingDate(Date periodDate)

public long getAmount(Date startDate, Date endDate)  ⟶  public long getTotalAmountPeriod(Date startDate, Date endDate)

private long getAmount(Period period)  ⟶  private long getTotalAmountPeriod(Period period)

private double getAmount(Date startDate, Date endDate)
        ⟶  private double getAmountForPeriodWithinBudgetPeriodOfStartDate(Date startDate, Date endDate)
```

New names suggest new smells and domain object

Comments



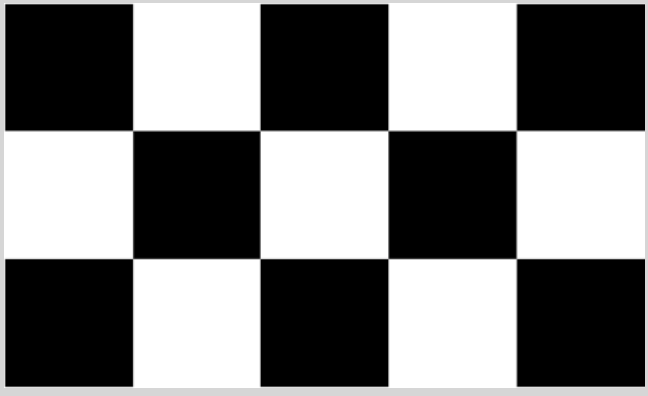
Explain how bad code works


```
public long getAmount_step6(Period period){
    Date startDate = period.getStartDate();
    Date endDate = period.getEndDate();

    //If Start and End are in the same budget period
    if (getBudgetPeriodType().getStartOfBudgetPeriod(startDate).equals(
            getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
        return (long) getAmountInPeriod(startDate, endDate);
    }

    //If the area between Start and End overlap at least two budget periods.
    if (getBudgetPeriodType().getStartOfNextBudgetPeriod(startDate).equals(
            getBudgetPeriodType().getStartOfBudgetPeriod(endDate))
            || getBudgetPeriodType().getStartOfNextBudgetPeriod(startDate).before(
                    getBudgetPeriodType().getStartOfBudgetPeriod(endDate))){
        double totalStartPeriod = getAmountInPeriod(startDate, getBudgetPeriodType().getEndOfBudgetPeriod(startDate));

        double totalInMiddle = 0;
        for (String periodKey : getBudgetPeriods(
                getBudgetPeriodType().getStartOfNextBudgetPeriod(startDate),
                getBudgetPeriodType().getStartOfPreviousBudgetPeriod(endDate))) {
            totalInMiddle += getAmount(getPeriodDate(periodKey));
        }

        double totalEndPeriod = getAmountInPeriod(getBudgetPeriodType().getStartOfBudgetPeriod(endDate), endDate);
        return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
    }


    throw new RuntimeException("...");
}
```

Comments are deodorant

They hide other smells

Not all comments are bad.



Comments that describe code because it is unreadable by itself -> Bad

Comments that describe why the code works this way -> Sometimes ok!

Lazy Class

A class isn't doing much


```
public class BudgetCategoryTypeMonthly extends BudgetCategoryType {

    public Date getStartOfBudgetPeriod(Date date) {
        return DateUtil.getStartOfMonth(date);
    }

    public Date getEndOfBudgetPeriod(Date date) {
        return DateUtil.getEndOfMonth(date);
    }

    public Date getBudgetPeriodOffset(Date date, int offset) {
        return getStartOfBudgetPeriod(DateUtil.addMonths(DateUtil.getStartOfMonth(date), 1 * offset));
    }

    public long getDaysInPeriod(Date date) {
        return DateUtil.getDaysInMonth(date);
    }

    public String getDateFormat() {
        return "MMM yyyy";
    }

    public String getName() {
        return BudgetCategoryTypes.BUDGET_CATEGORY_TYPE_MONTH.toString();
    }
}
```

Type in the classname also smells

Welcome BudgetPeriod!

```
private long getTotalAmountPeriod(Period period) {

    BudgetPeriod firstBudgetPeriod = createBudgetPeriodFromDate(period.getStartDate());

    //If Start and End are in the same budget period
    if (getBudgetPeriodType().getStartOfBudgetPeriod(period.getStartDate()).equals(
            getBudgetPeriodType().getStartOfBudgetPeriod(period.getEndDate()))){
        return (long) getAmountForPeriodWithinBudgetPeriodOfStartDate(period.getStartDate(), period.getEndDate());
    }
```

Abstraction suggested by:
- new method names
- lazy class
- comments

Use BudgetPeriod

```
BudgetPeriod firstBudgetPeriod = createBudgetPeriodFromDate(period.getStartDate());
BudgetPeriod lastBudgetPeriod = createBudgetPeriodFromDate(period.getEndDate());

if (firstBudgetPeriod.equals(lastBudgetPeriod)){
    return (long) getAmountForPeriodWithinBudgetPeriodOfStartDate(period.getStartDate(), period.getEndDate());
}
```

Now useless comment deleted

And first implementation

```
public class BudgetPeriod {

    private Period period;

    public BudgetPeriod(BudgetCategoryType type, Date date) {
        period = new Period (type.getStartOfBudgetPeriod(date), type.getEndOfBudgetPeriod(date));
    }

    @Override
    public boolean equals(Object object) {
        BudgetPeriod otherBudgetPeriod = (BudgetPeriod) object;
        return otherBudgetPeriod.period.equals(this.period);
    }

}
```

Yes, BudgetCategoryType is still there!
Removing will be gradual

Replace getBudgetPeriodType with BudgetPeriod

```
private long getTotalAmountPeriod(Period period) {

    BudgetPeriod firstBudgetPeriod = createBudgetPeriodFromDate(period.getStartDate());
    BudgetPeriod lastBudgetPeriod = createBudgetPeriodFromDate(period.getEndDate());

    if (firstBudgetPeriod.equals(lastBudgetPeriod))
        return (long) getAmountForPeriodWithinBudgetPeriodOfStartDate(period.getStartDate(), period.getEndDate());

    //If the area between Start and End overlap at least two budget periods.
    if (firstBudgetPeriod.nextBudgetPeriod().getStartDate().equals(lastBudgetPeriod.getStartDate())
            || firstBudgetPeriod.nextBudgetPeriod().getStartDate().before(lastBudgetPeriod.getStartDate())){
        double totalStartPeriod = getAmountForPeriodWithinBudgetPeriodOfStartDate(period.getStartDate(),
                firstBudgetPeriod.getEndDate());

        double totalInMiddle = 0;
        for (String periodKey : getBudgetPeriods(
                firstBudgetPeriod.nextBudgetPeriod().getStartDate(),
                lastBudgetPeriod.previousBudgetPeriod().getStartDate())) {
            totalInMiddle += getAmountFromBudgetPeriodContainingDate(getPeriodDate(periodKey));
        }

        double totalEndPeriod = getAmountForPeriodWithinBudgetPeriodOfStartDate(lastBudgetPeriod.getStartDate(),
period.getEndDate());
        return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
    }

    throw new RuntimeException("You should not be here.  We have returned all legitimate numbers from getAmount
(Date, Date) in BudgetCategoryImpl.  Please contact Wyatt Olson with details on how you got here (what steps did you
perform in Buddi to get this error message).");
}
```

Added implementation

```
public BudgetPeriod nextBudgetPeriod() {
    return new BudgetPeriod (type, type.getBudgetPeriodOffset(period.getStartDate(), 1));
}

public Date getStartDate() {
    return period.getStartDate();
}

public Date getEndDate() {
    return period.getEndDate();
}

public BudgetPeriod previousBudgetPeriod() {
    return new BudgetPeriod (type, type.getBudgetPeriodOffset(period.getStartDate(), -1));
}
```

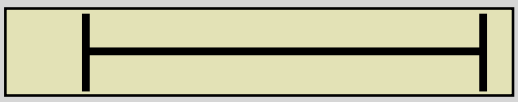
And removed nextBudgetPeriod which was added to BudgetCategoryType while removing magic numbers.
Functionality has moved in BudgetCategory

Dead code

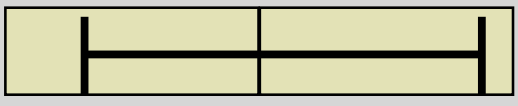
Code that is not used or not useful

Periods and BudgetPeriods

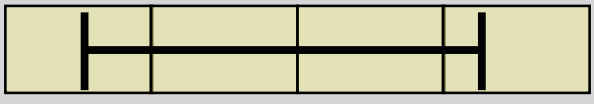
First if-statement



Second if-statement



Second if-statement



No other options!

```
   private long getTotalAmountPeriod(Period period) {

       BudgetPeriod firstBudgetPeriod = createBudgetPeriodFromDate(period.getStartDate());
       BudgetPeriod lastBudgetPeriod = createBudgetPeriodFromDate(period.getEndDate());

       if (firstBudgetPeriod.equals(lastBudgetPeriod))
           return (long) getAmountForPeriodWithinBudgetPeriodOfStartDate(period.getStartDate(), period.getEndDate());

       //If the area between Start and End overlap at least two budget periods.
       if (firstBudgetPeriod.nextBudgetPeriod().getStartDate().equals(lastBudgetPeriod.getStartDate())
               || firstBudgetPeriod.nextBudgetPeriod().getStartDate().before(lastBudgetPeriod.getStartDate())){
           double totalStartPeriod = getAmountForPeriodWithinBudgetPeriodOfStartDate(period.getStartDate(),
                   firstBudgetPeriod.getEndDate());

           double totalInMiddle = 0;
           for (String periodKey : getBudgetPeriods(
                   firstBudgetPeriod.nextBudgetPeriod().getStartDate(),
                   lastBudgetPeriod.previousBudgetPeriod().getStartDate())) {
               totalInMiddle += getAmountFromBudgetPeriodContainingDate(getPeriodDate(periodKey));
           }

           double totalEndPeriod = getAmountForPeriodWithinBudgetPeriodOfStartDate(lastBudgetPeriod.getStartDate(),
eriod.getEndDate());
           return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
       }

       throw new RuntimeException("You should not be here.  We have returned all legitimate numbers from getAmount
Date, Date) in BudgetCategoryImpl.  Please contact Wyatt Olson with details on how you got here (what steps did you
erform in Buddi to get this error message).");
   }
```

TRUE! It cannot possibly come here :)

Removed dead code

```
    private long getTotalAmountPeriod(Period period) {

        BudgetPeriod firstBudgetPeriod = createBudgetPeriodFromDate(period.getStartDate());
        BudgetPeriod lastBudgetPeriod = createBudgetPeriodFromDate(period.getEndDate());

        if (firstBudgetPeriod.equals(lastBudgetPeriod))
            return (long) getAmountForPeriodWithinBudgetPeriodOfStartDate(period.getStartDate(), period.getEndDate());

        double totalStartPeriod = getAmountForPeriodWithinBudgetPeriodOfStartDate(period.getStartDate(),
                firstBudgetPeriod.getEndDate());

        double totalInMiddle = 0;
        for (String periodKey : getBudgetPeriods(
                firstBudgetPeriod.nextBudgetPeriod().getStartDate(),
                lastBudgetPeriod.previousBudgetPeriod().getStartDate())) {
            totalInMiddle += getAmountFromBudgetPeriodContainingDate(getPeriodDate(periodKey));
        }

        double totalEndPeriod = getAmountForPeriodWithinBudgetPeriodOfStartDate(lastBudgetPeriod.getStartDate(),
period.getEndDate());
        return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
    }
```

Data Clump and Odd Name

```
private long getTotalAmountPeriod(Period period) {

    BudgetPeriod firstBudgetPeriod = createBudgetPeriodFromDate(period.getStartDate());
    BudgetPeriod lastBudgetPeriod = createBudgetPeriodFromDate(period.getEndDate());

    if (firstBudgetPeriod.equals(lastBudgetPeriod))
        return (long) getAmountForPeriodWithinBudgetPeriod(period, firstBudgetPeriod);

    double totalStartPeriod = getAmountForPeriodWithinBudgetPeriod(new Period (period.getStartDate(),
            firstBudgetPeriod.getEndDate()), firstBudgetPeriod);

    double totalInMiddle = 0;
    for (String periodKey : getBudgetPeriods(
            firstBudgetPeriod.nextBudgetPeriod().getStartDate(),
            lastBudgetPeriod.previousBudgetPeriod().getStartDate())) {
        totalInMiddle += getAmountFromBudgetPeriodContainingDate(getPeriodDate(periodKey));
    }

    double totalEndPeriod = getAmountForPeriodWithinBudgetPeriod(new Period(lastBudgetPeriod.getStartDate(),
            period.getEndDate()), lastBudgetPeriod);
    return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
}
```

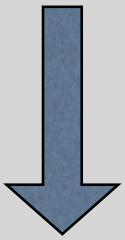
Yes, still ugly...

Feature Envy

More interested in another class than its own

Requesting too much information from Periods

```
private double getAmountForPeriodWithinBudgetPeriod(Period period, BudgetPeriod firstBudgetPeriod) {
    long amount = getAmountFromBudgetPeriodContainingDate(firstBudgetPeriod.getStartDate());
    long daysInPeriod = getBudgetPeriodType().getDaysInPeriod(firstBudgetPeriod.getStartDate());
    long daysBetween = DateUtil.getDaysBetween(period.getStartDate(), period.getEndDate(), true);
    double amountInPeriod = ((double) amount / (double) daysInPeriod) * daysBetween;
    return amountInPeriod;
}
```



Move functionality to Period and BudgetPeriod

```
private double getAmountForPeriodWithinBudgetPeriod(Period period, BudgetPeriod firstBudgetPeriod) {
    long amount = getAmountFromBudgetPeriodContainingDate(firstBudgetPeriod.getStartDate());
    long daysInPeriod = firstBudgetPeriod.getAmountOfDays();
    long daysBetween = period.getAmountOfDays();
    double amountInPeriod = ((double) amount / (double) daysInPeriod) * daysBetween;
    return amountInPeriod;
}
```


Additional Cleanup

```
private double getAmountForPeriodWithinBudgetPeriod(Period period, BudgetPeriod firstBudgetPeriod) {
    long amount = getAmountFromBudgetPeriodContainingDate(firstBudgetPeriod.getStartDate());
    long daysInPeriod = firstBudgetPeriod.getAmountOfDays();
    long daysBetween = period.getAmountOfDays();
    double amountInPeriod = ((double) amount / (double) daysInPeriod) * daysBetween;
    return amountInPeriod;
}
```

Inline Temp - Alt-Shift-I

```
private double getAmountForPeriodWithinBudgetPeriod(Period period, BudgetPeriod firstBudgetPeriod) {
    long amount = getAmountFromBudgetPeriod(firstBudgetPeriod);
    long daysInPeriod = firstBudgetPeriod.getAmountOfDays();
    long daysBetween = period.getAmountOfDays();
    return ((double) amount / (double) daysInPeriod) * daysBetween;
}
```

Primitive Obsession

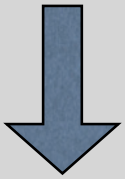
Use of primitives in higher-level abstraction methods


```
    double totalStartPeriod = getAmountForPeriodWithinBudgetPeriod(new Period (period.getStartDate(),
            firstBudgetPeriod.getEndDate()), firstBudgetPeriod);

    double totalInMiddle = 0;
    for (String periodKey : getBudgetPeriods(
            firstBudgetPeriod.nextBudgetPeriod().getStartDate(),
            lastBudgetPeriod.previousBudgetPeriod().getStartDate())) {
        totalInMiddle += getAmountFromBudgetPeriodContainingDate(getPeriodDate(periodKey));
    }

    double totalEndPeriod = getAmountForPeriodWithinBudgetPeriod(new Period(lastBudgetPeriod.getStartDate(),
            period.getEndDate()), lastBudgetPeriod);
    return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
}
```

```
double totalInMiddle = 0;
for (BudgetPeriod budgetPeriod :
        getBudgetPeriods(firstBudgetPeriod.nextBudgetPeriod(), lastBudgetPeriod.previousBudgetPeriod())) {
    totalInMiddle += getAmountFromBudgetPeriod(budgetPeriod);
}
```

And BudgetPeriods

```
public List<String> getBudgetPeriods(Date startDate, Date endDate){
    List<String> budgetPeriodKeys = new LinkedList<String>();

    Date temp = getBudgetPeriodType().getStartOfBudgetPeriod(startDate);

    while (temp.before(getBudgetPeriodType().getEndOfBudgetPeriod(endDate))){
        budgetPeriodKeys.add(getPeriodKey(temp));
        temp = getBudgetPeriodType().getBudgetPeriodOffset(temp, 1);
    }

    return budgetPeriodKeys;
}
```



```
private List<BudgetPeriod> getBudgetPeriods(BudgetPeriod firstBudgetPeriod, BudgetPeriod lastBudgetPeriod) {
    List<BudgetPeriod> budgetPeriodKeys = new LinkedList<BudgetPeriod>();

    BudgetPeriod currentBudgetPeriod = firstBudgetPeriod;

    while (currentBudgetPeriod.getStartDate().before(lastBudgetPeriod.getEndDate())){
        budgetPeriodKeys.add(currentBudgetPeriod);
        currentBudgetPeriod = currentBudgetPeriod.nextBudgetPeriod();
    }

    return budgetPeriodKeys;
}
```


And move it to BudgetPeriod (feature envy)

```
public List<BudgetPeriod> createBudgetPeriodListTill(BudgetPeriod lastBudgetPeriod) {
    List<BudgetPeriod> budgetPeriodKeys = new LinkedList<BudgetPeriod>();

    BudgetPeriod currentBudgetPeriod = this;

    while (currentBudgetPeriod.getStartDate().before(lastBudgetPeriod.getEndDate())){
        budgetPeriodKeys.add(currentBudgetPeriod);
        currentBudgetPeriod = currentBudgetPeriod.nextBudgetPeriod();
    }

    return budgetPeriodKeys;
}
```

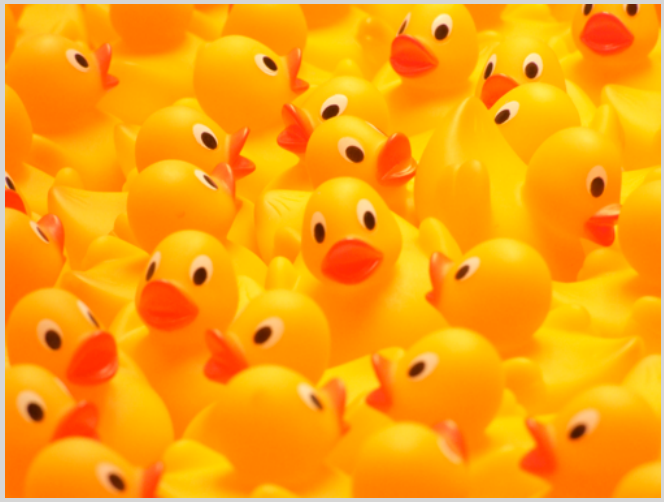
And the call

```
double totalInMiddle = 0;
for (BudgetPeriod budgetPeriod :
        firstBudgetPeriod.nextBudgetPeriod().createBudgetPeriodListTill(lastBudgetPeriod.previousBudgetPeriod()))

    totalInMiddle += getAmountFromBudgetPeriod(budgetPeriod);
}
```

Domain objects

Primitive Obsession is one of the most common and important smells.

It often suggests missing domain objects!



Abstraction Distraction

Use of different abstraction levels in the same code

```
private long getTotalAmountPeriod(Period period) {

    BudgetPeriod firstBudgetPeriod = createBudgetPeriodFromDate(period.getStartDate());
    BudgetPeriod lastBudgetPeriod = createBudgetPeriodFromDate(period.getEndDate());

    if (firstBudgetPeriod.equals(lastBudgetPeriod))
        return (long) getAmountForPeriodWithinBudgetPeriod(period, firstBudgetPeriod);

    double totalStartPeriod = getAmountForPeriodWithinBudgetPeriod(new Period (period.getStartDate(),
            firstBudgetPeriod.getEndDate()), firstBudgetPeriod);

    double totalInMiddle = 0;
    for (BudgetPeriod budgetPeriod :
            firstBudgetPeriod.nextBudgetPeriod().createBudgetPeriodListTill(lastBudgetPeriod.previousBudgetPeriod())) {
        totalInMiddle += getAmountFromBudgetPeriod(budgetPeriod);
    }

    double totalEndPeriod = getAmountForPeriodWithinBudgetPeriod(new Period(lastBudgetPeriod.getStartDate(),
            period.getEndDate()), lastBudgetPeriod);
    return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
}
```

```
BudgetPeriod firstBudgetPeriod = createFirstBudgetPeriod(period);
BudgetPeriod lastBudgetPeriod = createLastBudgetPeriod(period);
```


```
double totalStartPeriod = getAmountForPeriodWithinBudgetPeriod(new Period (period.getStartDate(),
        firstBudgetPeriod.getEndDate()), firstBudgetPeriod);
```

```
double totalStartPeriod = getAmountForOverlappingDays(period, firstBudgetPeriod);
```

And new getAmountForOverlappingDays

```
private double getAmountForOverlappingDays(Period period, BudgetPeriod firstBudgetPeriod) {
    long amount = getAmountFromBudgetPeriod(firstBudgetPeriod);
    long daysInPeriod = firstBudgetPeriod.getAmountOfDays();
    long daysBetween = period.getAmountOfOverlappingDays(firstBudgetPeriod.getPeriod());
    return ((double) amount / (double) daysInPeriod) * daysBetween;
}
```

And new getAmountOfOverlappingDays in Period

```
public long getAmountOfOverlappingDays(Period period) {
    Date largestStartDate = (start.after(period.start)) ? start : period.start;
    Date smallestEndDate = (end.before(period.end)) ? end : period.end;

    if (smallestEndDate.before(largestStartDate))
        return 0;
    return new Period(largestStartDate, smallestEndDate).getAmountOfDays();
}
```

Use more generic version

```
if (firstBudgetPeriod.equals(lastBudgetPeriod))
    return (long) getAmountForPeriodWithinBudgetPeriod(period, firstBudgetPeriod);

double totalStartPeriod = getAmountForOverlappingDays(period, firstBudgetPeriod);

double totalInMiddle = 0;
for (BudgetPeriod budgetPeriod :
        firstBudgetPeriod.nextBudgetPeriod().createBudgetPeriodListTill(lastBudgetPeriod.previousBudgetPeriod()))

    totalInMiddle += getAmountFromBudgetPeriod(budgetPeriod);
}

double totalEndPeriod = getAmountForPeriodWithinBudgetPeriod(new Period(lastBudgetPeriod.getStartDate(),
        period.getEndDate()), lastBudgetPeriod);
```

And delete getAmountForPeriodWithinBudgetPeriod

```
if (firstBudgetPeriod.equals(lastBudgetPeriod))
    return (long) getAmountForOverlappingDays(period, firstBudgetPeriod);

double totalStartPeriod = getAmountForOverlappingDays(period, firstBudgetPeriod);

double totalInMiddle = 0;
for (BudgetPeriod budgetPeriod :
        firstBudgetPeriod.nextBudgetPeriod().createBudgetPeriodListTill(lastBudgetPeriod.previousBudgetPeriod())) {
    totalInMiddle += getAmountForOverlappingDays(period, budgetPeriod);
}

double totalEndPeriod = getAmountForOverlappingDays(period, lastBudgetPeriod);
```


Hmm...

```
private long getTotalAmountPeriod(Period period) {

    BudgetPeriod firstBudgetPeriod = createFirstBudgetPeriod(period);
    BudgetPeriod lastBudgetPeriod = createLastBudgetPeriod(period);

    if (firstBudgetPeriod.equals(lastBudgetPeriod))
        return (long) getAmountForOverlappingDays(period, firstBudgetPeriod);

    double totalStartPeriod = getAmountForOverlappingDays(period, firstBudgetPeriod);

    double totalInMiddle = 0;
    for (BudgetPeriod budgetPeriod :
            firstBudgetPeriod.nextBudgetPeriod().createBudgetPeriodListTill(lastBudgetPeriod.previousBudgetPeriod())) {
        totalInMiddle += getAmountForOverlappingDays(period, budgetPeriod);
    }

    double totalEndPeriod = getAmountForOverlappingDays(period, lastBudgetPeriod);
    return (long) (totalStartPeriod + totalInMiddle + totalEndPeriod);
}
```

Last refactoring made the first and last budgetPeriod the same, so we can delete the checks

Final version (?)

```
private long getTotalAmountPeriod(Period period) {

    BudgetPeriod firstBudgetPeriod = createFirstBudgetPeriod(period);
    BudgetPeriod lastBudgetPeriod = createLastBudgetPeriod(period);

    double total = 0;
    for (BudgetPeriod budgetPeriod :  firstBudgetPeriod.createBudgetPeriodListTill(lastBudgetPeriod)) {
        total += getAmountForOverlappingDays(period, budgetPeriod);
    }

    return (long) total;
}
```

Probably not. Possible future directions:

- Introduce Budget class (not exists!!)
- Introduce Money class (not exists!!!!!)
- Remove the BudgetCategoryType
- Much more primitive obsessions

Refactorings so far

- Extract Local Variable (alt-shift-L)
  - Introduce a local variable and initialize it with a piece of code.
- Extract Method (alt-shift-M)
  - Replace a piece of code with a call to a new method
- Inline Temp (alt-shift-I)
  - Replace a variable with the code that initialized it
  - Opposite of Extract Local Variable
- Extract Class
  - Extract a piece of code into a separate class
- Preserve Whole Object
  - Pass a whole object rather than parts of it
- Move Method (alt-shift-V)
  - Move a method from one class to another

Divergent Change

Need to change the same class for different reasons

Whatever changes

We'll need to edit **this** code.

Shotgun Surgery

Responsibility is split among several classes

Whatever changes

We'll always need to edit in **many different** places.

Do your tests require shotgun surgery?

Managers

Words in the name that do not tell much about its responsiblity

What do managers do?


```
class GlobalDataHandler extends DefaultHandler {
    private GlobalData data;
    // used to save temporary object data
    private Stack<Object> tempData;
    ...

}
```

GlobalDataHandler with GlobalData and a Stack of Objects????

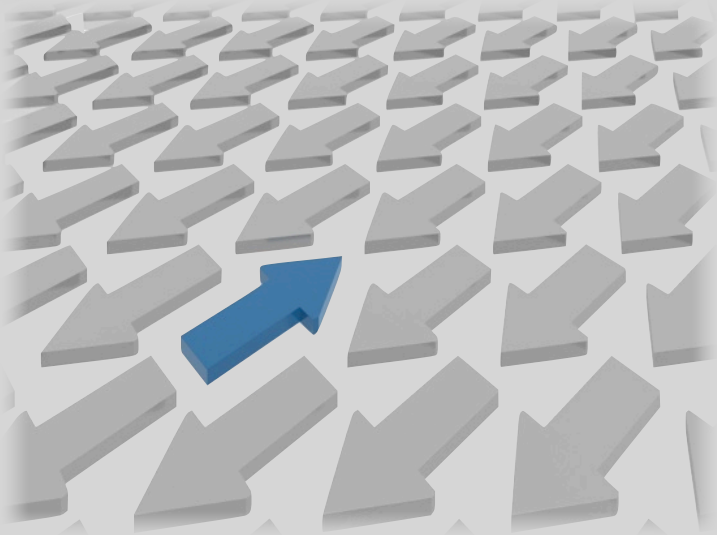
Long Parameter List

More than one or two parameters

```
boolean addSubject(String pId, String userid, String code, String title, String comment) throws Exception {
    CollabDb db = null;
    String sql = "";
    try {
        db = new CollabDb();
        Statement stmt = db.getStatement();
        SQLRenderer r = new SQLRenderer();
    ...
}
```

Often indicates other smells, such as Data Clumps

Arrows



Nested if statements generate an arrow shape

```
private static ArrayList<DataSet>[] parseDBFile(BufferedReader readFile) throws DBFileParseException
{
    if(readFile != null)
    {
        String line = readLine(readFile);
        String expression = "";
        if(line != null)
        {
            if(line.startsWith("<TRAINERDB>")) {
                // About 150 LOC deleted
            }

            else throw new DBFileParseException("Fehler in erster Zeile");
        }
        else throw new DBFileParseException("Datei leer");
    }
    throw new DBFileParseException("Datei fehlt");
}
```


Guard Clause

```
private static ArrayList<DataSet>[] parseDBFile2(BufferedReader readFile) throws DBFileParseException
{
    if(readFile == null)
        throw new DBFileParseException("Datei leer");

    String line = readLine(readFile);
    String expression = "";
    if(line == null)
        throw new DBFileParseException("Fehler in erster Zeile");
    if(line.startsWith("<TRAINERDB>")) {
        // About 150 LOC deleted
    }
}
```

Arrows can often be resolved with a revert-if refactoring and introducing a Guard Clause

They often also indicate other smells such as Long Method

Single entry - Single Exit

One major cause of arrows is developers strictly following the "single-entry, single-exit" principles (often taught at schools)

When keeping methods small, single-exit is not very relevant.

Null Checks

Many occurrences of checks for null

Remember: Design by Contract

```
public boolean accept(File f) {
    if(f != null) {
        if(f.isDirectory()) {
        return true;
        }
        String extension = getExtension(f);
        if(extension != null && mExtensions.contains(extension)) {
        return true;
        };
    }
    return false;
}
```

Null checks such as above ought to be ensured by the caller.

Design by Contract!

Primitive Obsession

Use of primitives in higher-level abstraction methods

```
public static Card createCard(List row, int rowNum) throws InvalidCsvException, IOException {

    if (row.size() == 1 && row.get(0).equals("")) {
        // ignore blank row
        return null;
    }

    if (row.size() < 5) {
        throw new InvalidCsvException("Expected 5 columns: Reserved, Reserved, Display Name, " +
                "Reserved, and Email Address.  Found " + row.size() + " columns on row " + rowNum);
    }

    String displayName = row.get(2).toString().trim();
    String address = row.get(4).toString().trim();

    if (address.indexOf("@") == -1) {
        throw new InvalidCsvException("Error found on row " + rowNum + "The following string is not " +
                "an email address in column 5: " + address);
    }

    if ("".equals(displayName)) {
        displayName = address;
    }

    Card card = new Card();
    card.setDisplayName(displayName);
    card.setEmailAddress(address);

    return card;
}
```


Domain objects

Primitive Obsession is one of the most common and important smells.

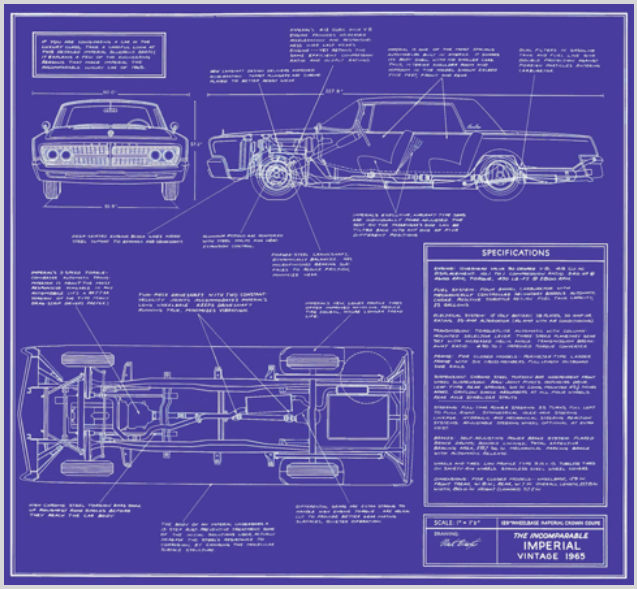
It often suggests missing domain objects!

```
public static Card createCard(CardFileRow row) throws InvalidCsvException, IOException {

    if (row.isValid()) {
        return null;
    }

    if (!row.emailAddressIsValid()) {
        throw new InvalidCsvException("Error found on row " + row.number() + "The following string is not " +
                "an email address in column 5: " + row.getEmailAddress().toString());
    }

    String displayName = row.getDisplayName();
    if (row.getDisplayName().isEmpty()) {
        displayName = row.getEmailAddress().toString();
    }

    Card card = new Card();
    card.setDisplayName(displayName);
    card.setEmailAddress(row.getEmailAddress());

    return card;
}
```

Introduced domain object
CardFileRow and EmailAddress

Upfront Design

A common cause of missing domain objects is to do design in "the design phase"



During implementation, new things are discovered, but "the design is already done"..
Causing... primitive obsession

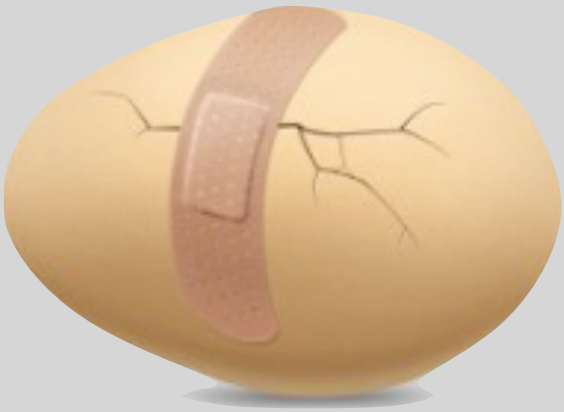
Data Class

A class consisting of only data

```
public class SubTransaction implements Serializable, Cloneable {
    private static final long serialVersionUID = 1L;

    private double amount;
    private String description;
    private Category category;

    public SubTransaction(double amount, String description, Category category) {
        super();
        this.amount = amount;
        this.description = description;
        this.category = category;
    }
    public double getAmount() {
        return amount;
    }
    public String getDescription() {
        return description;
    }
    public Category getCategory() {
        return category;
    }
    @Override
    public Object clone() {
        return new SubTransaction(amount, description, category);
    }

    @Override
    public String toString() {
        return "[{"+description+"}{"+category+"}{"+amount+"}]";
    }
}
```

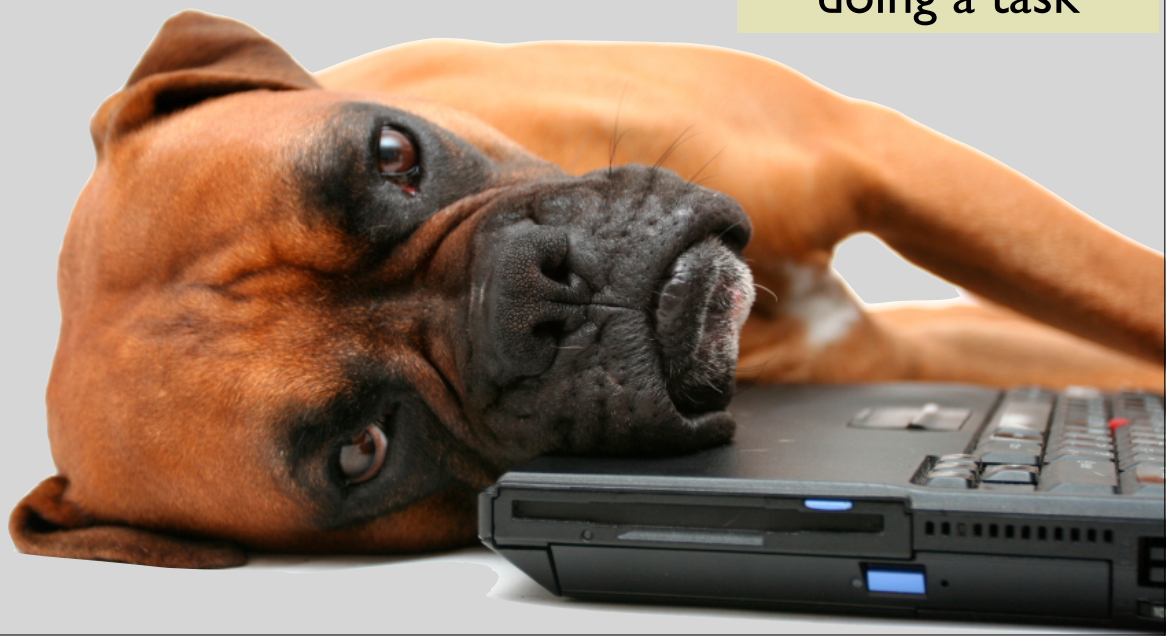
Temporary!

Data classes are common when the domain objects start to emerge. Over time they ought to get logic, otherwise it is a smell.



Boredom

Feeling bored doing a task



```
private void  fillBookToChapters() {
    bookToChapters.put(new Integer(1), new Integer(50)); //genesis
    bookToChapters.put(new Integer(2), new Integer(40)); //exodus
    bookToChapters.put(new Integer(3), new Integer(27)); //leviticus
    bookToChapters.put(new Integer(4), new Integer(36)); //numbers
    bookToChapters.put(new Integer(5), new Integer(34)); //deuteronomy
    bookToChapters.put(new Integer(6), new Integer(24)); //joshua
    bookToChapters.put(new Integer(7), new Integer(21)); //judges
    bookToChapters.put(new Integer(8), new Integer(4)); //ruth
    bookToChapters.put(new Integer(9), new Integer(31)); //1 samuel
    bookToChapters.put(new Integer(10), new Integer(24)); //2 samuel
    bookToChapters.put(new Integer(11), new Integer(22)); //1 kings
    bookToChapters.put(new Integer(12), new Integer(25)); //2 kings
    bookToChapters.put(new Integer(13), new Integer(29)); //1 chronicles
    bookToChapters.put(new Integer(14), new Integer(36)); //2 chronicles
    bookToChapters.put(new Integer(15), new Integer(10)); //ezra
    bookToChapters.put(new Integer(16), new Integer(13)); //nehemia
    bookToChapters.put(new Integer(17), new Integer(10)); //esther
    bookToChapters.put(new Integer(18), new Integer(42)); //job
    bookToChapters.put(new Integer(19), new Integer(150)); //psalms
    bookToChapters.put(new Integer(20), new Integer(31)); //proverbs
    bookToChapters.put(new Integer(21), new Integer(12)); //ecclesiastes
    bookToChapters.put(new Integer(22), new Integer(8)); //song of songs
    bookToChapters.put(new Integer(23), new Integer(66)); //isaiah
    bookToChapters.put(new Integer(24), new Integer(52)); //jeremiah
    bookToChapters.put(new Integer(25), new Integer(5)); //lamentations
    bookToChapters.put(new Integer(26), new Integer(48)); //ezekiel
    bookToChapters.put(new Integer(27), new Integer(12)); //daniel
    bookToChapters.put(new Integer(28), new Integer(14)); //hosea
    bookToChapters.put(new Integer(29), new Integer(3)); //joel
    bookToChapters.put(new Integer(30), new Integer(9)); //amos
    bookToChapters.put(new Integer(31), new Integer(0)); //obadiah
    bookToChapters.put(new Integer(32), new Integer(4)); //jonah
    bookToChapters.put(new Integer(33), new Integer(7)); //micah
    bookToChapters.put(new Integer(34), new Integer(3)); //nahum
    bookToChapters.put(new Integer(35), new Integer(3)); //habakkuk
    bookToChapters.put(new Integer(36), new Integer(3)); //zephaniah
```

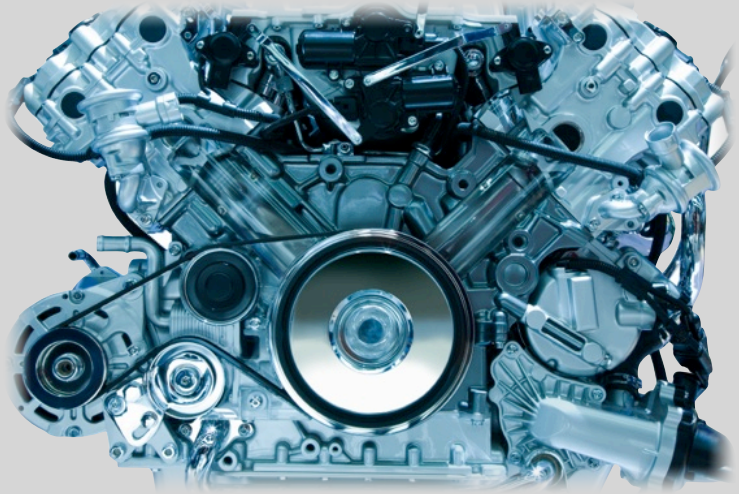
Inappropriate Intimacy

One class accesses internal information of another class

Casting interface to implementation...

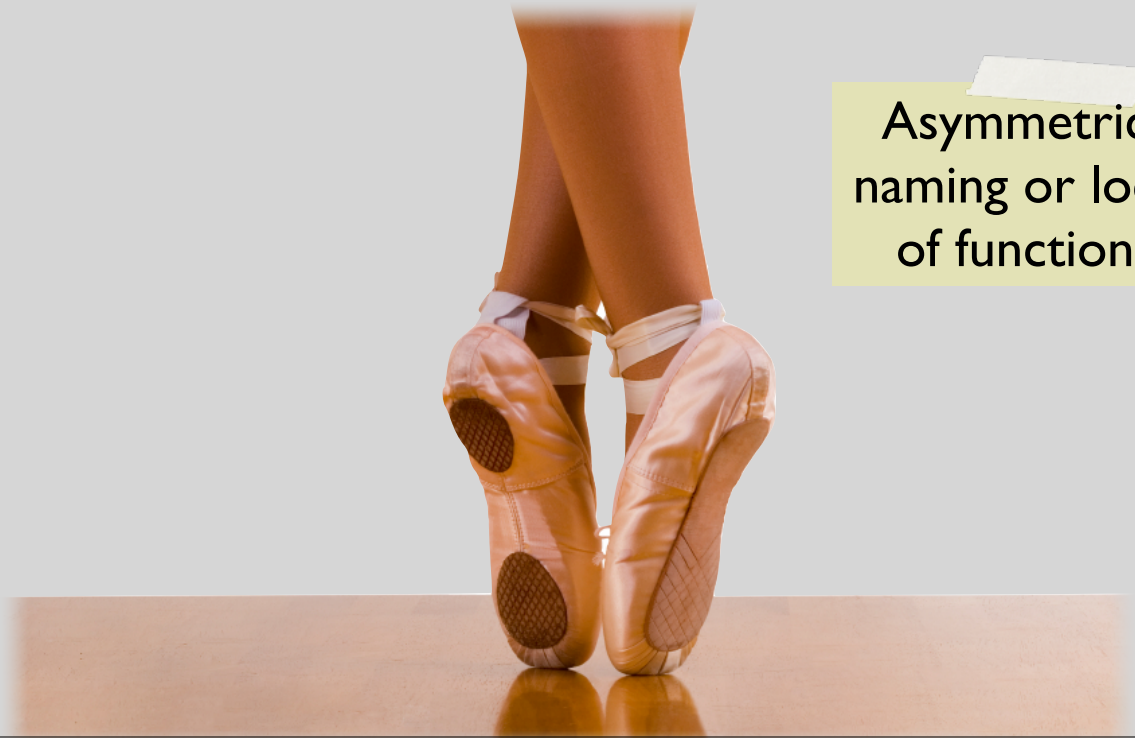
```
for (BudgetCategory bc : document.getBudgetCategories()) {
    boolean changed = false;
    Map<String, Long> newAmountMap = new HashMap<String, Long>();

    for (String key : ((BudgetCategoryImpl) bc).getBudgetPeriods().keySet()) {
        String[] splitKey = key.split(":");
        if (splitKey.length == 2){
            changed = true;
            Date dateKey = new Date(Long.parseLong(splitKey[1]));
            Logger.getLogger(ModelFactory.class.getName()).finest("dateKey: " + dateKey);
            long amount = ((BudgetCategoryImpl) bc).getBudgetPeriods().get(key);
            String newDateKey =
                bc.getBudgetPeriodType().getName()
                + ":" + DateUtil.getYear(dateKey)
                + ":" + DateUtil.getMonth(dateKey)
                + ":" + DateUtil.getDay(dateKey);
            newAmountMap.put(newDateKey, amount);
        }
        else if (splitKey.length == 4){
            newAmountMap.put(key, ((BudgetCategoryImpl) bc).getBudgetPeriods().get(key));
        }
    }

    if (changed){
        ((BudgetCategoryImpl) bc).setBudgetPeriods(newAmountMap);
    }
}
```

Speculative Generality



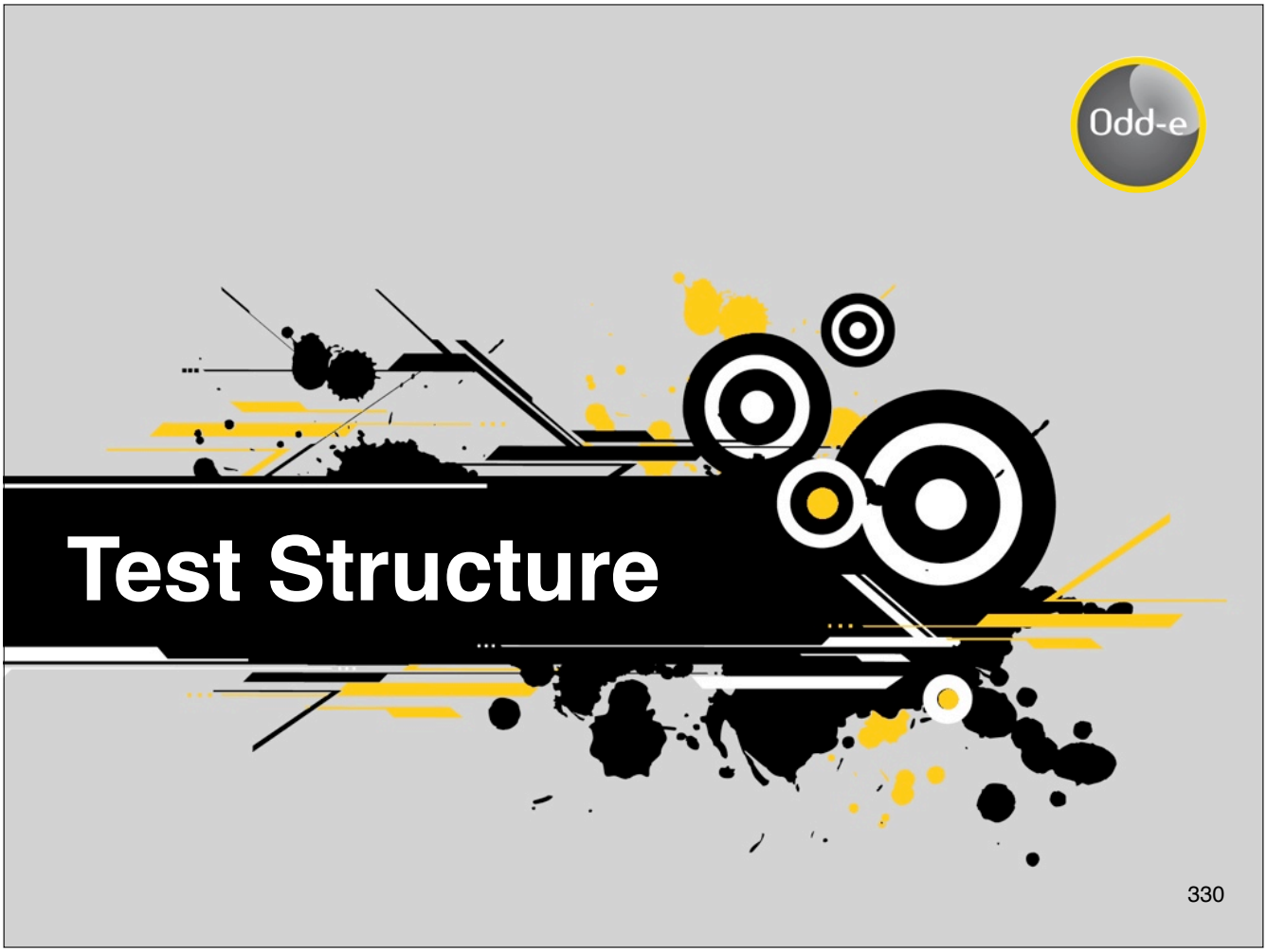
Code that is more complicated than to fulfill the current requirements

Asymmetrical Code

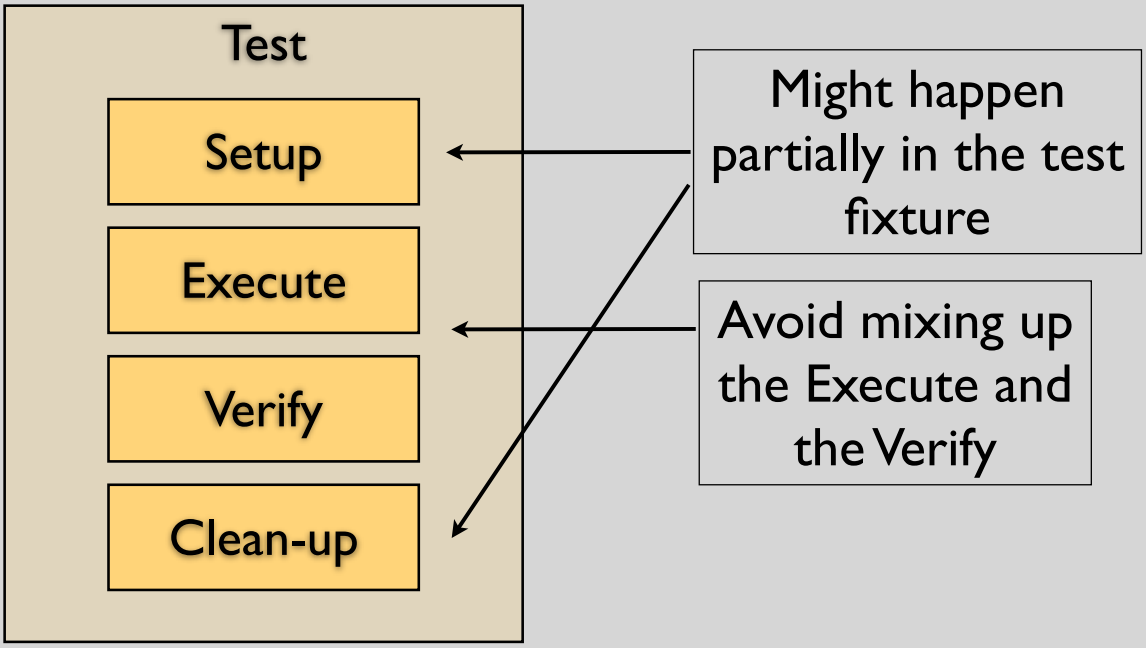


Asymmetrical in naming or location of functionality


Odd-e
Good unit tests


Odd-e
Test Structure

Non-Mock test structure



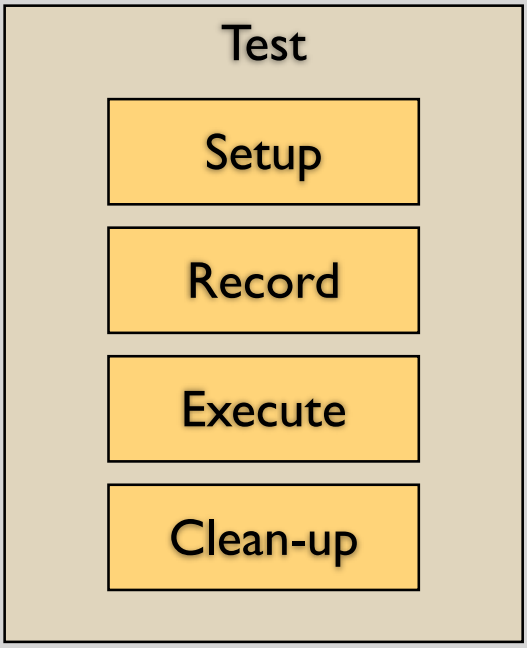
Also called: Arrange-Act-Assert

Consistent code, Clear Structure

- Arrange-Act-Assert / Given-When-Then
- One assertion per test / Test one behavior per class

```
@Test
public void makesReservationThroughReservationService() {
        // ARRANGE
        final Reservation reservation = new Reservation();
        Reservations reservations = stubReservationsToReturn(reservation);
        HouseKeeping houseKeeping = stubHouseKeeping();
        HotelRoom room = new HotelRoom(reservations, houseKeeping);


        // ACT
        Reservation reservationReturned = room.bookFor(new Customer());

        // ASSERT
        assertSame(reservation, reservationReturned);
}
```

Mock test structure



Tests with mocks first specify the expectations and then execute.

Properties of good tests

Odd-e

It is not a unit test when:

- It talks to the database
- It communicates across the network
- It touches the file system
- It can't run at the same time as any of your other unit tests
- You have to do special things to your environment (such as editing config files) to run it.

Michael Feathers

Odd-e

GOALS

✓ Tests should reduce risk, not introduce it.

✓ Tests should be easy to run again and again.

✓ Tests should be easy to maintain as time passes.

HOW TO DO IT?

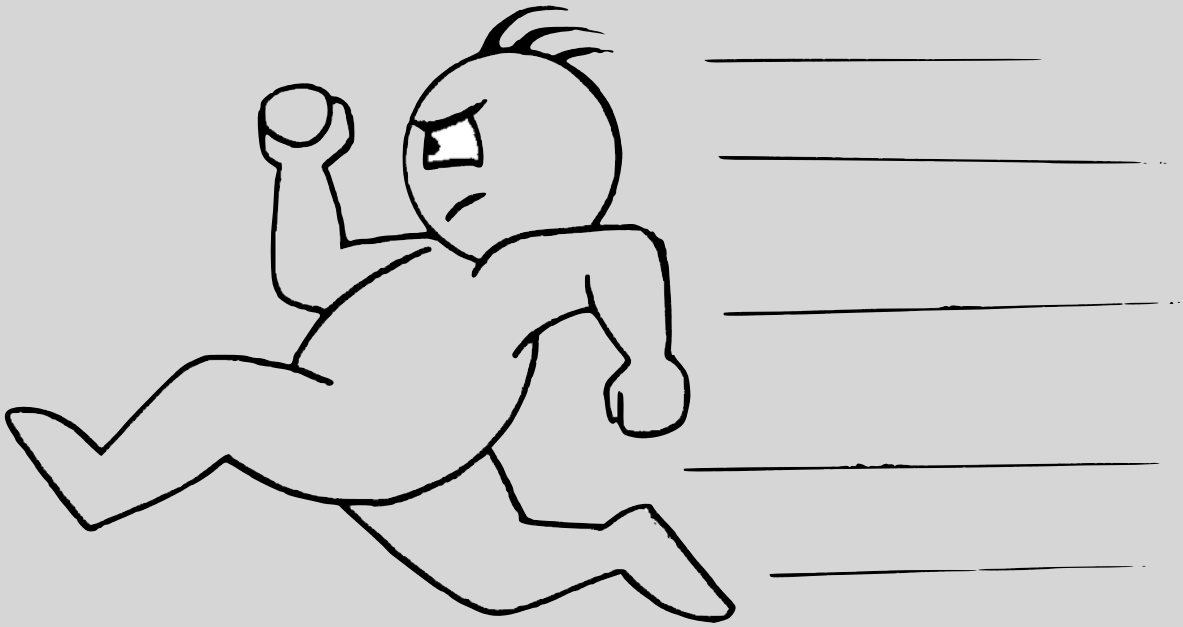
✓ Tests should only fail because of one reason.

✓ Tests should only test one thing

✓ Minimize test dependencies (no dependencies on databases, files, ui etc.)

Odd-e

✓ Fast! (seconds)



Odd-e

✓ A-TRIP

★ Automated

★ Thorough

★ Repeatable

★ Independent

★ Professional

✓ Readable

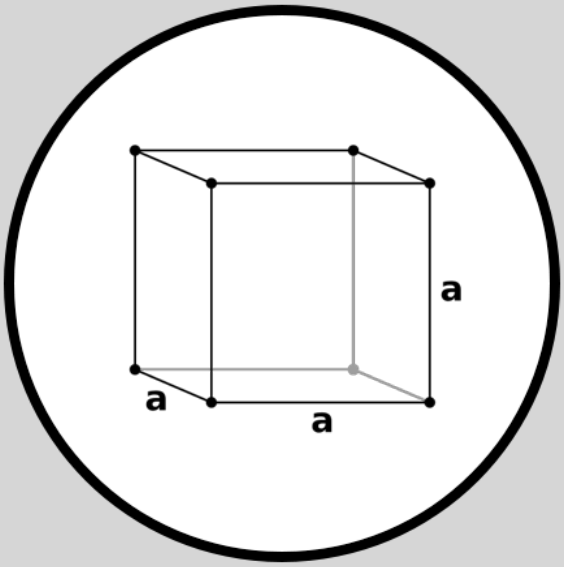
NOT

*l*Egi*b*lE lEt*t*eR*s*

BUT

Readable Words

✓ small & simple



✓ To the .

```
@Test
public void iteratorReturnsAllObjectsOfGivenTypeInObjectSpace() {
        IRubyObject o1 = runtime.newFixnum(10);
        IRubyObject o2 = runtime.newFixnum(20);
        IRubyObject o3 = runtime.newFixnum(30);
        IRubyObject o4 = runtime.newString("hello");
        objectSpace.add(o1);
        objectSpace.add(o2);
        objectSpace.add(o3);
        objectSpace.add(o4);

        List storedFixnums = new ArrayList(3);
        storedFixnums.add(o1);
        storedFixnums.add(o2);
        storedFixnums.add(o3);

        Iterator strings = objectSpace.iterator(runtime.getString());
        assertSame(o4, strings.next());
        assertNull(strings.next());

        Iterator numerics = objectSpace.iterator(runtime.getNumeric());
        for (int i = 0; i < 3; i++) {
                Object item = numerics.next();
                assertTrue(storedFixnums.contains(item));
        }
        assertNull(numerics.next());
}
```

```
@Test
public void iteratorReturnsAllObjectsOfGivenTypeInObjectSpace() {
        givenObjectSpaceContains(1, 2, 3);
        givenObjectSpaceContains("string");
        assertThat(iteratorFor(runtime.getString()), containsExactly("hello"));
        assertThat(iteratorFor(runtime.getNumeric()), containsExactly(1, 2, 3));
}
```


✓ Focused

```
public class HotelRoom {
        public Reservation bookFor(Customer customer) {
                Reservation reservation = reservations.reserveFor(customer, this);
                housekeeping.notifyAbout(reservation);
                return reservation;
        }
}

@Test
public void makesReservationThroughReservationService() {
        final Reservation reservation = new Reservation();
        Reservations reservations = stubReservationsToReturn(reservation);
        HouseKeeping houseKeeping = stubHouseKeeping();

        HotelRoom room = new HotelRoom(reservations, houseKeeping);

        assertSame(reservation, room.bookFor(new Customer());
}
```

Whether housekeeping gets notified is another test's concern!

✓ Named well

```
@Test
public void booksRoomThroughReservations()
```


✓ Potty trained

GOOD TESTS

✓ Clean up after themselves.

✓ Don't leave the lid up.

✓ Expressive

Literals vs. Variables

```
@Test
public void alternative1() {
   int votes = 2;
   assertEquals(WEIGHT * votes * votes, new Topic(votes).popularity());
}

@Test
public void alternative2() {
   assertEquals(4.56, new Topic(votes).popularity());
}

@Test
public void alternative3() {
   int votes = 2;
   assertEquals(1.14 * votes * votes, new Topic(votes).popularity());
}
```

Which of these tests breaks most easily?

Which of these tests expresses its intent best?

Sometimes it's *good* if a test breaks easily.

Odd-e

Expensive setup

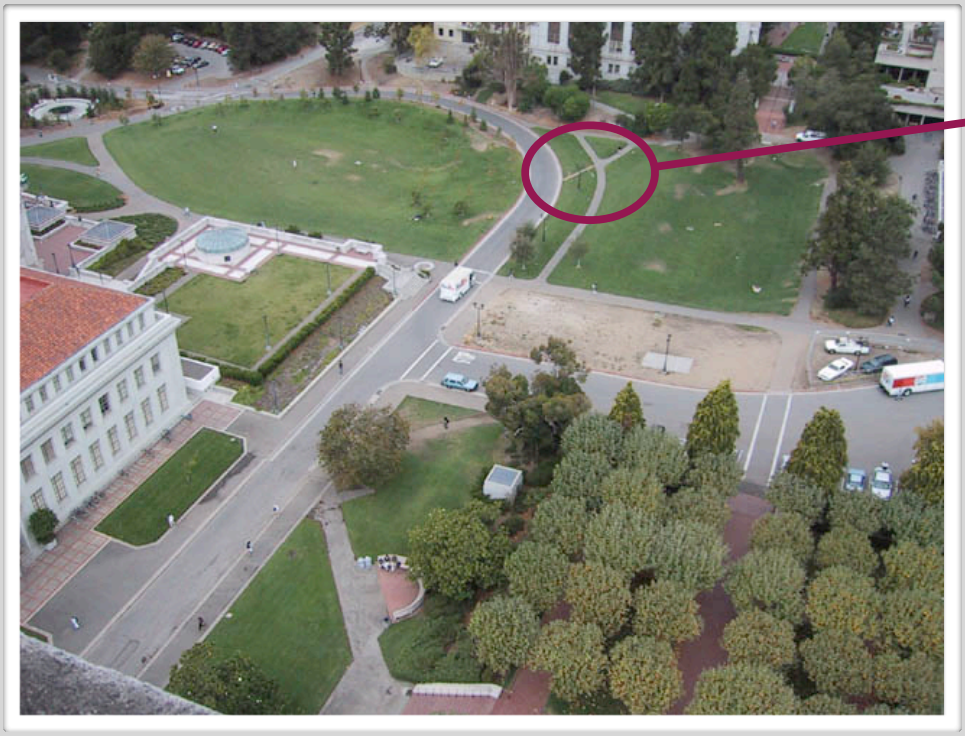

Odd-e
Thinking about Design


Odd-e
Emergent Design

Traditional Design

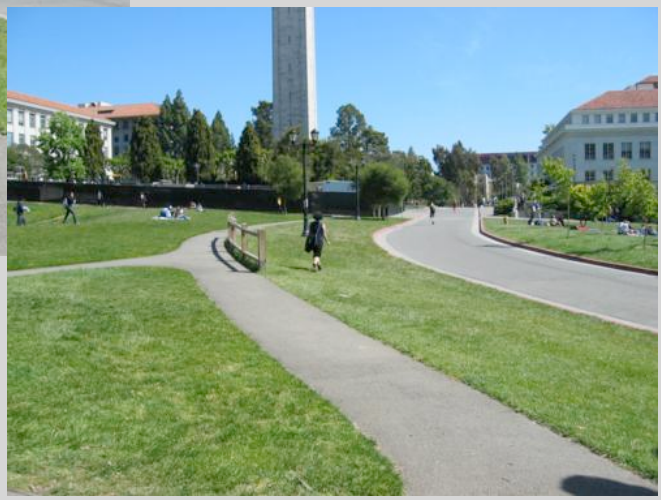
Ref: peterme.com

Reality



Forced design

Reality



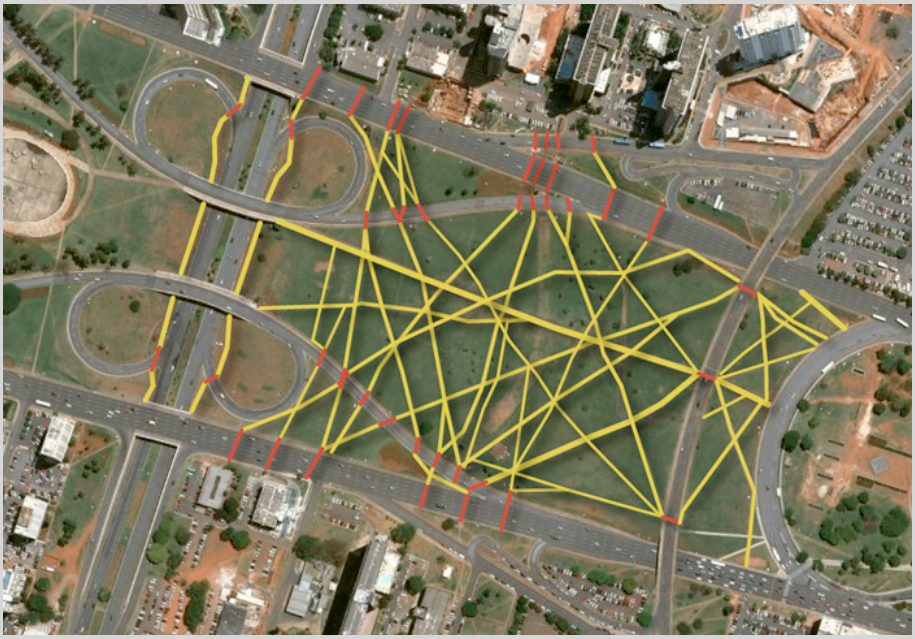
Larger problems!

Beginning of a new path

Emergent design

Ref: Discovering Urbanism

Reality



Traditional vs Emergent

Design (try to make **all** decisions)

Implement the initial design

Traditional

Emergent

Design (decide an initial direction)

Design/Code

Reflect and Decide Direction

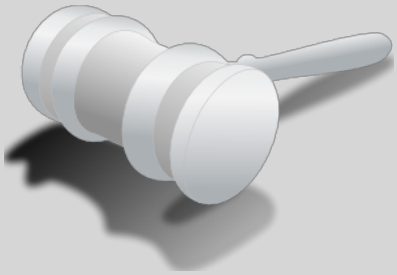
Design/Code

Reflect and Decide Direction

Design/Code

Traditional vs Emergent

Traditional design is about decision

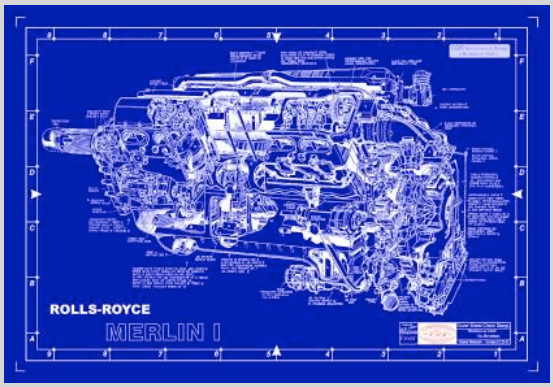


Decide the whole blueprint

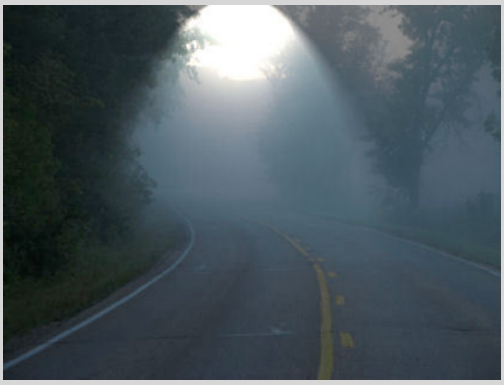
Emergent design is about direction

Make sure we are going in the right direction

Traditional vs Emergent



Mindset:
We can think of most things beforehand. That will be most efficient



Mindset:
Premature speculation is probably wrong. Gradual discovery and learning leads to better solutions

Emergent Design

Steering the design

Steering the direction of the design based on:

- Order of test-driving
- Principles
- Patterns


Odd-e
Principles
363


From: IosTechnies.com
SOLID
Software Development is not a Jenga game

SOLID

Principles for Object-Oriented Design collected by Uncle Bob Martin:

- S - Single Responsibility Principle
- O - Open-Close Principle
- L - Liskov Substitution Principle
- I - Interface Segregation Principle
- D - Dependency Inversion Principle

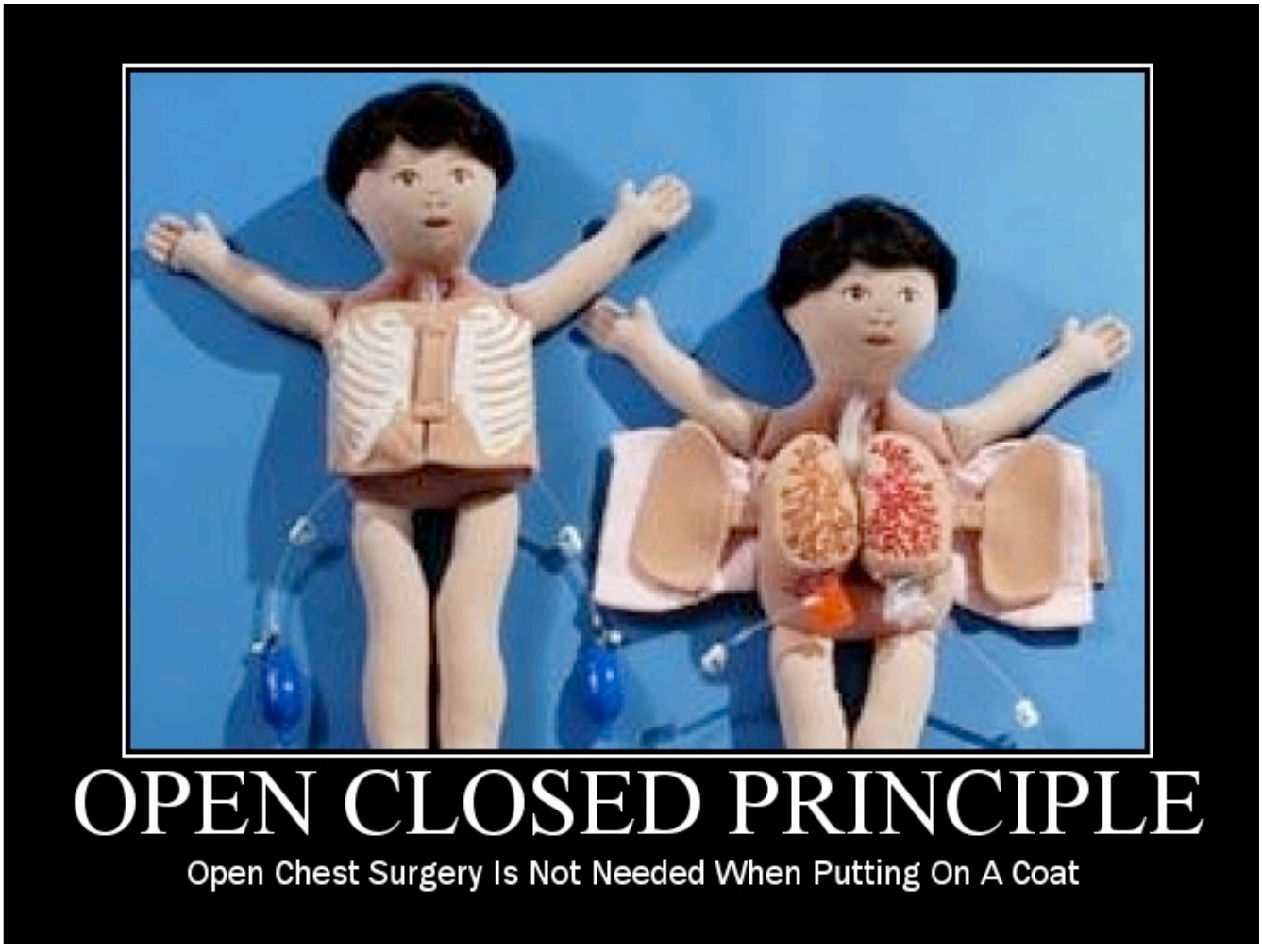
SRP - Single-Responsibility Principle

```
public final class CurrencyConverter {
    private static final String ECB_RATES_URL = "http://www.ecb.int/stats/eurofxref/eurofxref-daily.xml";

    transient private File cacheFile = null;
    private String cacheFileName = null;
    private HashMap<String, Long> fxRates = new HashMap<String, Long>(40);
    private Proxy proxy = Proxy.NO_PROXY;

    public void setProxy(Proxy proxy) { ... }

    public double convert(double amount, String fromCurrency, String toCurrency) { ... }

    public String getCacheFileName() { ... }

...
```

"A class should have only one reason to change"

This class breaks the SRP:

- Converts Currencies
- Updates exchange rates via the web
- Caches rates locally



OCP - Open-Close Principle

```
public final class CurrencyConverter {
    private static final String ECB_RATES_URL = "http://www.ecb.int/stats/eurofxref/eurofxref-daily.xml";

    transient private File cacheFile = null;
    private String cacheFileName = null;
    private HashMap<String, Long> fxRates = new HashMap<String, Long>(40);
    private Proxy proxy = Proxy.NO_PROXY;

    public void setProxy(Proxy proxy) { ... }

    public double convert(double amount, String fromCurrency, String toCurrency) { ... }

    public String getCacheFileName() { ... }

...
```

Software entities should be open for extension, but closed for modification

This class breaks the OCP:

- It doesn't allow for extension such as retrieving the currencies from a non-web source.
- Yet, even getting the currencies from a different URL will require modification.

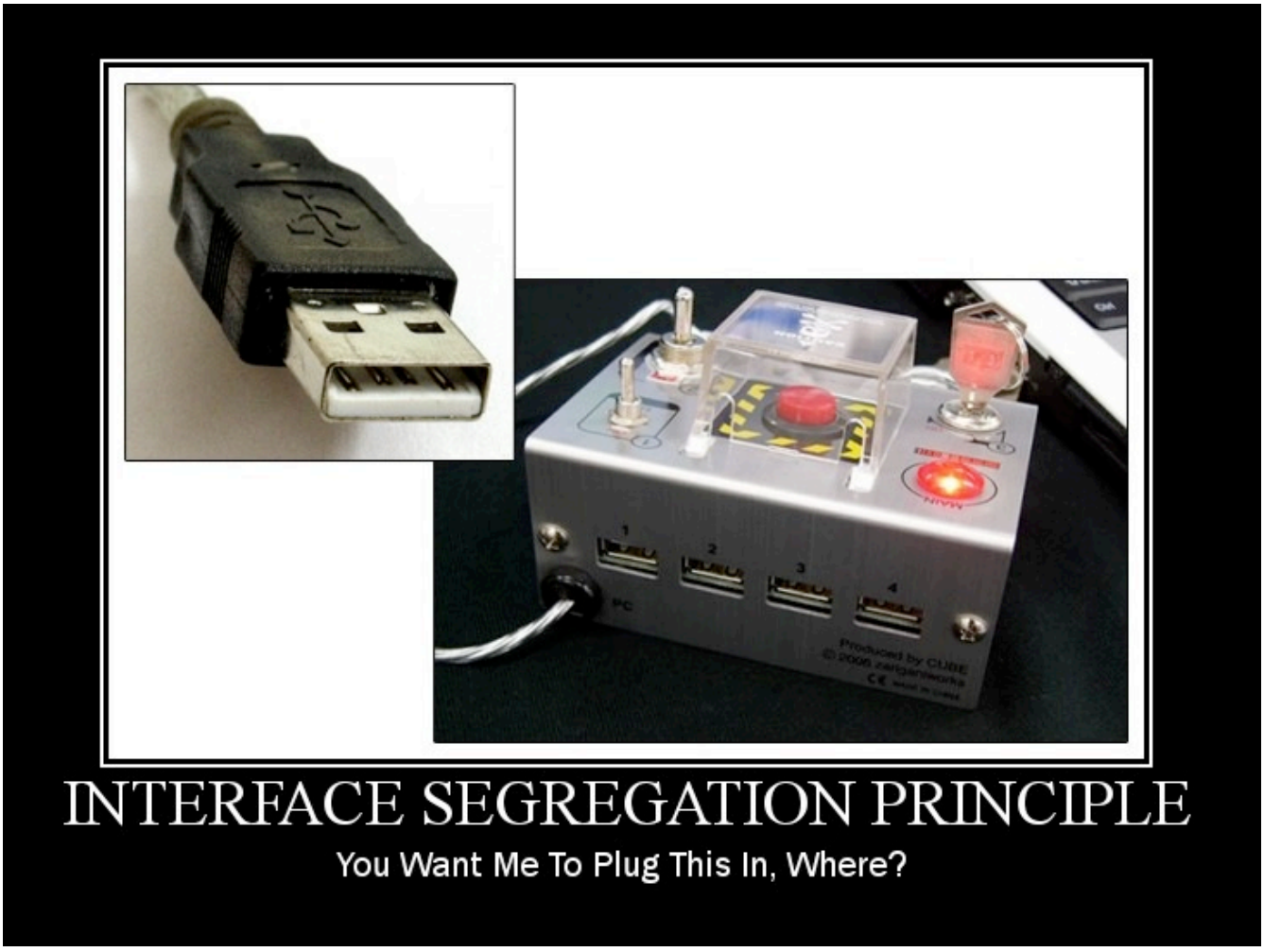
LSP - Liskov Substitution Principle

```
public class SortedArrayList<T extends Comparable<? super T>> extends AbstractList<T> {
    private ArrayList<T> backingList;

    public void add(int location, T object) {
        //We don't care about what position you try for, foo!
        add(arg1);
    }

    public boolean add(T object) {
        int index = Collections.binarySearch(backingList, arg0);
        if (index < 0)
            index = (index + 1) * -1;
        backingList.add(index, arg0);
        return true;
    }
```

Subtypes must be substitutable for their base types

This class breaks the LSP:

- The add to a position in the AbstractList breaks the base type interface.
- After adding it to position x, the get from position x will return a different object.



ISP - Interface Segregation Principle

```
public interface Document extends ModelObject, StandardDocument {

      public static final int RESET_PASSWORD = 1;  //Should we change the current password?
      public static final int CHANGE_PASSWORD = 2; //Should we prompt for a password?

      public void addAccount(Account account) throws ModelException;
      public void addAccountType(AccountType type) throws ModelException;
      public void addBudgetCategory(BudgetCategory budgetCategory) throws ModelException;
      public void addScheduledTransaction(ScheduledTransaction scheduledTransaction) throws ModelException;
      public void addTransaction(Transaction transaction) throws ModelException;
      public String doSanityChecks();
      public Account getAccount(String name);
      public List<Account> getAccounts();
      public AccountType getAccountType(String name);
      public List<AccountType> getAccountTypes();
      public List<BudgetCategory> getBudgetCategories();
      public BudgetCategory getBudgetCategory(String fullName);
      public long getNetWorth(Date date);
      public ModelObject getObjectByUid(String uid);
      public List<ScheduledTransaction> getScheduledTransactions();
      public List<Source> getSources();
      public List<Transaction> getTransactions();
      public List<Transaction> getTransactions(Date startDate, Date endDate);
      public List<Transaction> getTransactions(Source source);
      public List<Transaction> getTransactions(Source source, Date startDate, Date endDate);
      public void refreshUidMap() throws ModelException;
      public void removeAccount(Account account) throws ModelException;
      public void removeAccountType(AccountType type) throws ModelException;
      public void removeBudgetCategory(BudgetCategory budgetCategory) throws ModelException;
      public void removeScheduledTransaction(ScheduledTransaction scheduledTransaction) throws ModelException;
      public void removeTransaction(Transaction transaction) throws ModelException;
      public void save() throws DocumentSaveException;
      public void saveAs(File file) throws DocumentSaveException;
      public void saveToStream(OutputStream os) throws DocumentSaveException;
      public void setFlag(int flag, boolean set);
      public void updateAllBalances();
      public void updateScheduledTransactions();
      public Document clone() throws CloneNotSupportedException;
}
```

Clients should not be forced to depend on methods that they do not use

DIP - Dependency Inversion Principle

```
public class users implements IStructuredContentProvider {
    private Vector<user> usersList=new Vector<user>();

    private user activeUser=null;


    public void getUsersFromDB() {
        usersList.clear();

        List retrievals = DB.getSession().createQuery("from user WHERE outdated IS NULL").list();
        for (Iterator iter = retrievals.iterator(); iter.hasNext();) {
            user currentlyRetrieved =  (user)iter.next();
            usersList.add(currentlyRetrieved);
        }
```

Abstractions should not depend on details
Details should depend on abstractions

This class breaks the DIP:

- getUsersFromDB details ought to depend on an abstraction rather than direct SQL queries

Other principles

DRY. Don't Repeat Yourself!

Law of Demeter

Don't Ask, Tell.

Design by Contract

DRY - Don't Repeat Yourself

Every piece of knowledge must have a single, unambiguous, authoritative representation within a system

DRY principle covers more than obvious duplication, focus on duplication of knowledge

Law of Demeter

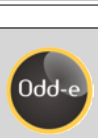
Any method of an object should only call methods belonging to:

- Itself
- Its parameters
- Any objects it creates
- Its direct component objects

```
public String replaceValues(TemplatePlaceholderValues replacementValues) {
    return template.replace(replacementValues.firstPlaceholder().name(),
           replacementValues.firstPlaceholder().value());
}
```

This class breaks the law of Demeter:

- The call to name() doesn't belong to any of these four groups



Don't ask, tell!

You should endeavor to tell objects what you want them to do instead of asking for their data

Design by Contract

- Pre-conditions for derived types must be same or weaker than base type
- Post-conditions for derived types must be the same or stronger than base type

Design Contract

Client

```
public class User {

    private Printer printer = new Printer();
    private String content;

    public void displayEverything() {
        printer.printLines(content, 10);
    }
}
```

Supplier

```
public class Printer {

    public void printLines(String content, int i) {

    }
}
```


Design Contract: The pre-condition

Client

```
public class User {

    private Printer printer = new Printer();
    private String content;

    public void displayEverything1() {
        printer.printLines1(content, 10);
    }
}
```

Supplier

```
public class Printer {

    public int printLines1(String content, int linesToPrint) {
        if (content == null)
            throw new IllegalArgumentException();
        if (linesToPrint < 0)
            throw new IllegalArgumentException();

        int amountOfLinesPrinted = 0;

        // Do the work

        return amountOfLinesPrinted;
    }
}
```

Contract:
Precondition: <anything>

Whatever the client passes is, Printer should work correctly.

Design Contract: The pre-condition

Client

```
private String content2 = new String();

public void displayEverything2() {
    printer.printLines2(content2, 10);
}
```

Supplier

```
public int printLines2(String content, int linesToPrint) {
    if (linesToPrint < 0)
        throw new IllegalArgumentException();

    int amountOfLinesPrinted = 0;

    // Do the work

    return amountOfLinesPrinted;
}
```

Contract:
Precondition:
- content is a valid String

Now, client is responsible for ensuring that content is a valid String. If not it is not a valid String, then the bug is in client and not in Supplier!

Design Contract: The pre-condition

Client

```
private String content3 = new String();

public void displayEverything3() {
    printer.printLines3(content3, 10);
}
```

Supplier

```
public int printLines3(String content, int linesToPrint) {

    int amountOfLinesPrinted = 0;

    // Do the work

    return amountOfLinesPrinted;
}
```

Contract:
Precondition:
- content is a valid String
- linesToPrint must be >= 0

More expectations more to the client by stricter pre-conditions

Design Contract: The post-condition

Client

```
public class User {

    private Printer printer = new Printer();
    private String content;

    public void displayEverything4() {
        printer.printLines4(content, 10);
    }
}
```

Supplier

```
    public int printLines4(String content, int linesToPrint) {
        return (int) ((Math.random() - 0.5)
            * (double) Integer.MAX_VALUE) * 2;
    }
```

Contract:

Postcondition:

- printLines returns a valid int

Client cannot assume anything!

Design Contract: The post-condition

Client

```
public class User {

    private Printer printer = new Printer();
    private String content;

    public void displayEverything5() {
        printer.printLines5(content, 10);
    }
}
```

Supplier

```
    public int printLines5(String content, int linesToPrint) {
        int amountOfLinesPrinted = 0;

        // Do the work

        return amountOfLinesPrinted;
    }
```

Contract:

Postcondition:

- printLines returns >= 0
- printLines won't throw an exception

Client can assume a sane return value

Design Contract: The post-condition

Client

```
public class User {

    private Printer printer = new Printer();
    private String content;

    public void displayEverything() {
        printer.printLines(content, 10);
    }
}
```

Supplier

```
    public int printLines5(String content, int linesToPrint) {
        int amountOfLinesPrinted = 0;

        // Do the work

        return amountOfLinesPrinted;
    }
```

Contract:

Postcondition:
- printLines returns >= 0
- printLines won't throw an exception
- the sizeOfBufferedText() == 0

The post-condition expresses what happened to the state of the object!

Design Contract: Class Invariant

Supplier

```
public class Printer {

    public int capacityOfPrintBuffer() {
        return capacity;
    }

    public int sizeOfBufferedText() {
        return sizeOfBufferedText;
    }
```

Contract:

Class Invariant:
-sizeOfBufferedText <= capacityOfPrintBuffer

Client can at any time assume the supplier's class invariant

Design Contract: Example contracts

Contract:

Precondition:
-<anything>
Postcondition:
- printLines returns >= 0
- printLines can throw IllegalArgumentException
Class Invariant:
-sizeOfBufferedText <= capacityOfPrintBuffer

With a weak pre-condition, the supplier can't guarantee much...

Often leads to ugly code

With a strong pre-condition, the supplier can guarantee more, less error handling in client

Contract:

Precondition:
- content is a valid String
- linesToPrint >= 0
Postcondition:
- printLines returns >= 0
- the sizeOfBufferedText() == 0
Class Invariant:
-sizeOfBufferedText <= capacityOfPrintBuffer

Design by Contract

Strong/weak pre/post conditions are not good/bad, they just are.

Always make expectations and assumptions clear by thinking in contracts between client/supplier

Design by Contract Summary

Pre-condition: must be true to execute correctly:

- The supplier assumes it to be true
- The user ensures it is true!

Post-condition: a state prevails upon completion i.e. a description of the results of the actions performed.

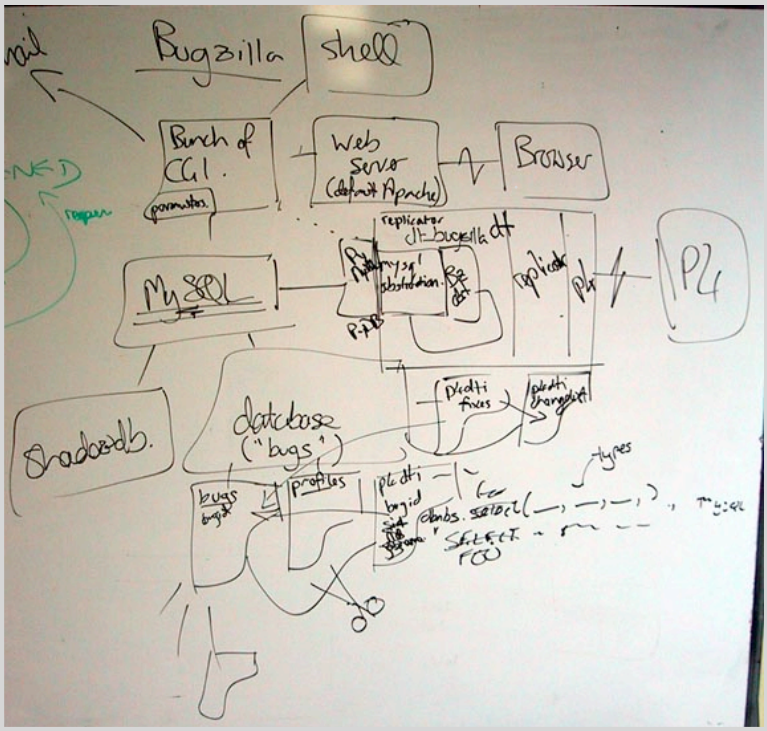
- The supplier must make it so!
- The user assumes it will be true

Modeling in Agile

First law of diagramming

The primary value in diagrams is in the discussion while diagramming—we model to have a conversation

Model on whiteboards



Ref: Ravenbook

Model together

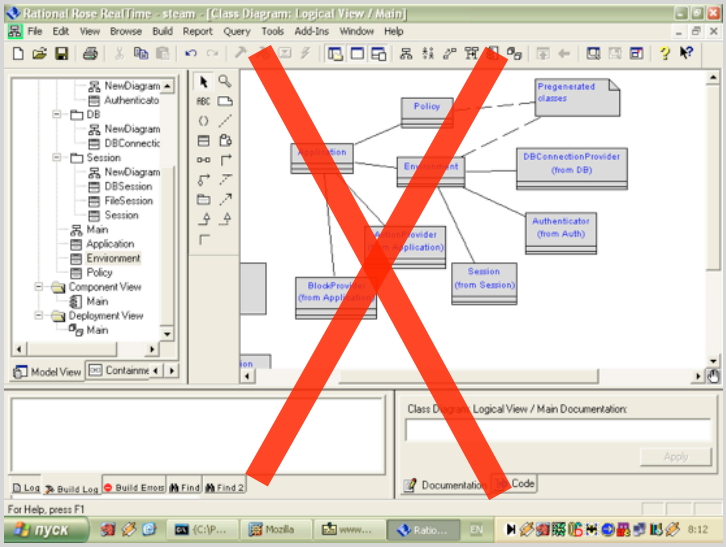
Ref: Technical University Dortmund

Principles

- Software is your primary goal
- Assume simplicity
- Model with a purpose
- Multiple models
- Content is more important than representation
- Everyone can learn from everyone else

Ref: Agile Modeling

Avoid electronic tools



Avoid electronic modeling tools. They limit the space and its hard to model together.

Even for documentation, photos often work better

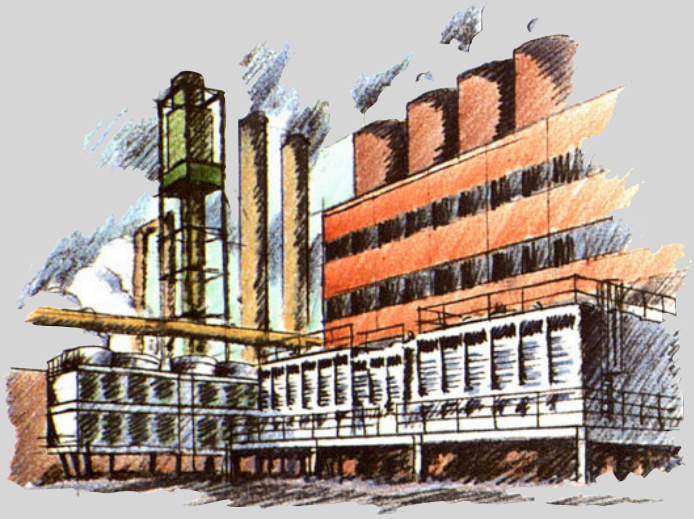
Object Creation

Hard dependencies

```
void clear(double initialBalance) {
    this.currentBalanceDate = DateUtils.dateToInteger(new Date());
    this.balanceHistory = new BalanceHistory(initialBalance);
    this.finalBalance = initialBalance;
    this.currentBalance = initialBalance;
    this.checkedBalance = initialBalance;
}

void enableEvents(boolean enabled) {
    super.setEventsEnabled(enabled);
    if (enabled) fireEvent(new EverythingChangedEvent(this));
}
```

Object creation are hard dependencies. They need to be separated

Factories



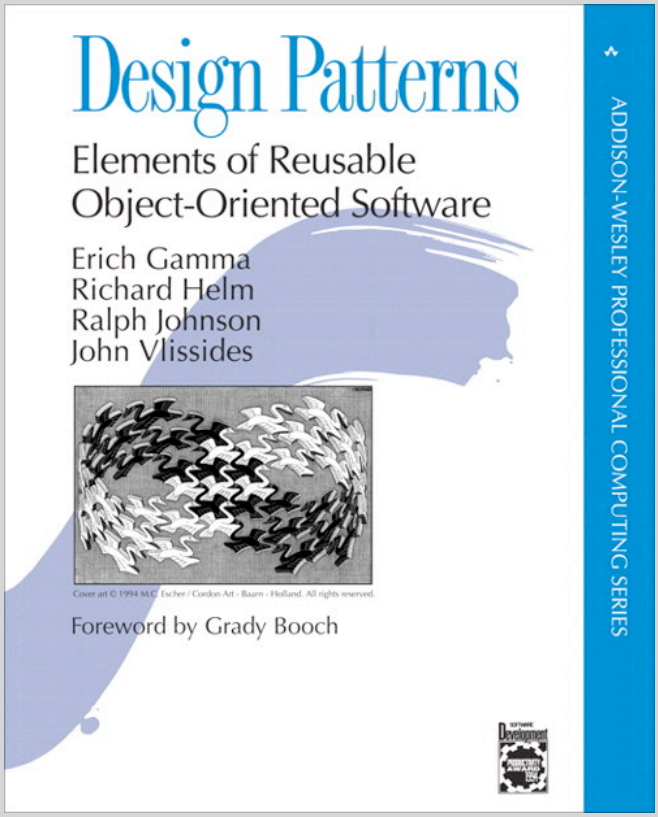
Move creation to separate classes

Dependency Injection

Pass objects in rather than create them yourself

- Setter injection
- Constructor injection

Patterns



Design Pattern

"descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context"

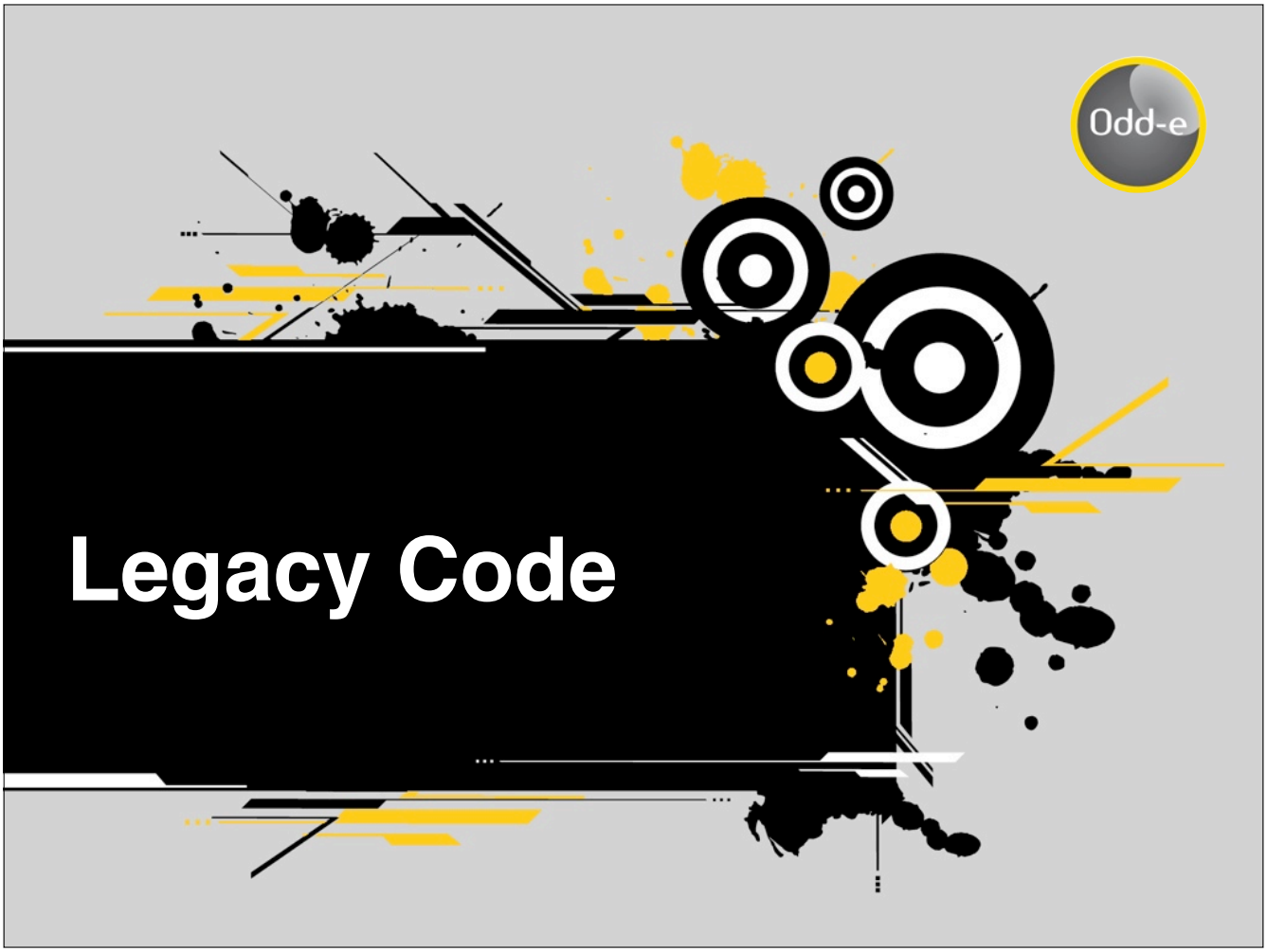
-- Gang of four

Common patterns

- Composite
- Bridge
- Decorator
- Observer
- Null Object
- Builder
- Chain of responsibility
- Template method

Avoid Singleton

(well, no need to avoid this Singleton)


Odd-e
Legacy Code


Odd-e
Legacy Overview

Agenda

- What is Legacy Code?
- Overview of Test-drive legacy development
- Dependency breaking
- Understanding the code
- Seeing Responsibilities
- Towards better responsibility assignment

What is legacy code?

Code without unit test

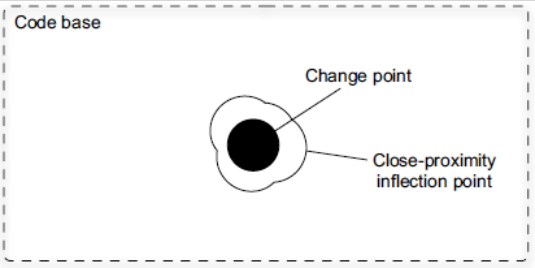
What can you do about it

1. Identify change point
2. Find test points
3. Break dependencies
4. Write tests
5. Make changes and refactor

The Legacy Code Dilemma

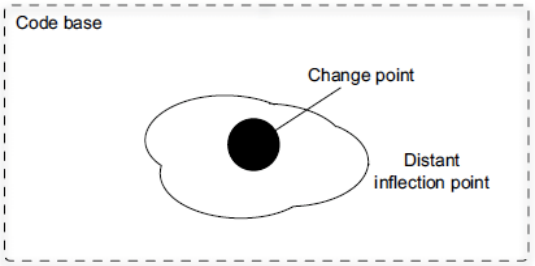
When we change code, we should have tests in place. To put tests in place, we often have to change code.

Change point and Test point



System boundaries – e.g. Network, database

Close proximity test points tend to provide a more localised checkpoint without too much noise around.



Distant test points are more likely to catch side effects

Seam

A seam is a place where you can alter behaviour in your program without editing in that place

- Link Seam
- Preprocessor Seam
- Metaprogramming Seam (e.g function pointer)
- Object (polymorphism) Seam

Enabling point is a place where you can make the decision to use one behaviour or another in each seam.

I don’t want to break anything

- Automatic Refactoring tool
- Lean on Compiler
  - Let your compiler tell where you need to fix
- Signature Preservation
- Single Goal Editing
  - Do one thing at a time
- Pair Programming

Understanding the code

Characterisation Tests

Tests that record the actual behaviour of a piece of code

1. Use a piece of code in a test harness
2. Write an assertion that you know will fail
3. Let the failure tell you what the behaviour is
4. Change the test so that it expects the behaviour that the code produces

Heuristics

- Write tests for area where you will make your changes
- Try writing tests for specific things that you are going to change
- If you are extracting or moving some functionality, write tests that verify the existence and connection of those behaviours on a case-by-case basis
- Verify that you are exercising the code that you are going to move and that is connected properly

Sensing Variable

Add a variable to a class and use it to sense conditions in the method that you want to refactor

Example:

```
public static final CharSequence getDisplayLabel(Context context, int kind, int type, CharSequence label) {
    CharSequence display = "";
    if (type != People.ContactMethods.TYPE_CUSTOM) {
        CharSequence[] labels = context.getResources().getTextArray( com.android.internal.R.array.emailAddressTypes);
        try {
            display = labels[type - 1];
        } catch (ArrayIndexOutOfBoundsException e) {
            display = labels[ContactMethods.TYPE_HOME - 1];
        }
    } else {
        if (!TextUtils.isEmpty(label)) {
            if (label.toString().equals(MOBILE_EMAIL_TYPE_NAME)) {
                display = context.getString( com.android.internal.R.string.mobileEmailTypeName);
            } else {
                display = label;
            }
        }
    }

    return display;
}
```


Introduce sensing variable

```
public boolean STANDARD_TYPE = false;

public static final CharSequence getDisplayLabel(Context context, int kind, int type, CharSequence label) {
    CharSequence display = "";
    if (type != People.ContactMethods.TYPE_CUSTOM) {
        CharSequence[] labels = context.getResources().getTextArray( com.android.internal.R.array.emailAddressTypes);
        try {
            display = labels[type - 1];
        } catch (ArrayIndexOutOfBoundsException e) {
            display = labels[ContactMethods.TYPE_HOME - 1];
        }
    } else {
        if (!TextUtils.isEmpty(label)) {
            if (label.toString().equals(MOBILE_EMAIL_TYPE_NAME)) {
                display = context.getString( com.android.internal.R.string.mobileEmailTypeName);
            } else {
                display = label;
            }
        }
        STANDARD_TYPE = true;
    }

    return display;
}
```

Add a test that covers the condition

```
public void testGetDisplayLabelForStandardType() {
    Context context = new Context();
    /*
    We still need to construct the inputs to produce a case that covers the condition
    */

    Contacts.getDisplayLabel(context, kind, type, label);
    assertTrue(Contacts.STANDARD_TYPE);
}
```

Refactor to understand

- Rename attributes to convey roles
- Rename methods to convey intent
- Rename classes to convey purpose
- Remove duplicated code
- Replace condition branches by methods
- Refactor method bodies to consistent level of abstraction

Scratch Refactoring

- Refactor it whatever way you want to get a better understanding of it
- Don’t check it in again
- Throw that code away

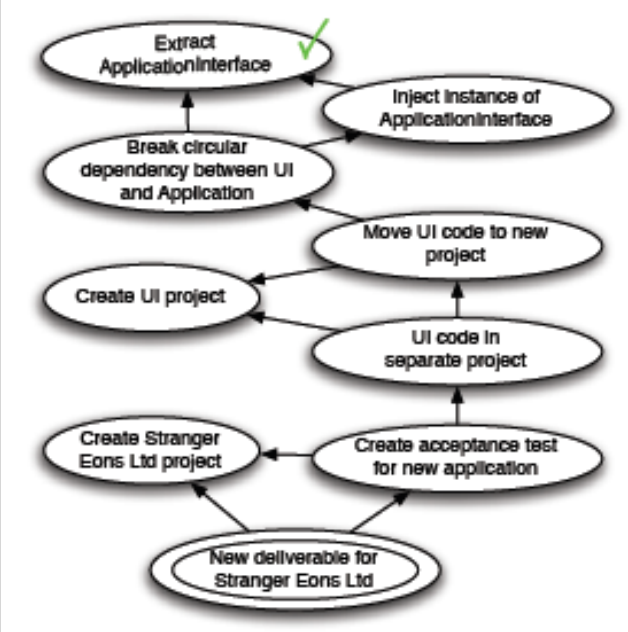
Effect Propagation

- Identify a method that will change
- If the method has a return value, look at its callers
- See if the method modifies any values, if it does, look at the methods use those values, and the methods that use those methods
- Make sure to look for superclasses and subclasses for instance variables
- Look at parameters to the methods to see if they are used by the code that you want to change
- Look for global variables and static data that is modified

So we have layers of layers of dependencies?

Mikado Method

- It's not mean to construct the big whole graph at one step
- Avoid "Analysis Paralysis"
- "Undo" and revert whenever needed



Further reading: http://mikadomethod.wordpress.com/book/

Dependency Breaking

Common Strategies

- Add
  - Extra Constructor
  - Static Setter
  - Parameterise
- Replace
  - Test specific subclass
  - Static instance
- Extract
  - Method
  - Interface / Implementation
  - Method to Class
  - Globals to Class

You may need to use it in combination!

Parameterise functions

- Create a new method with the internally created object as an argument
- Copy the code from the original method into the old method, deleting the creation code
- Cut the code from the original method and replace it with a call to the new method, using a “new expression” in the argument list

```
void run() {
   m_result = new TestResult();
   runTest(m_result);
}
```

```
void run() {
   run(new TestResult());
}

void run(TestResult result) {
   m_result = result;
   runTest(m_result);
}
```

Extra Constructor

```
public class LogFileMerge {
  private URL logFileA, logFileB;

  public LogFileMerge() {
    this(new URL("http://server1/system.log"),
    new URL("http://server2/system.log"));
  }

  LogFileMerge(URL a, URL b) {
    this.logFileA = a;
    this.logFileB = b;
  }

  ...
}
```

Original delegates to extra constructor

Dependencies exposed to tests

Test-Specific SubClass

```
public class CreditCardProcessing {

    public boolean isValid(String cardnumber) {
        return validationCodeMatches(cardnumber)
            && cardIsActive(cardnumber);
    }

    protected boolean validationCodeMatches(String cardnumber) {
        // validation logic omitted for brevity...
    }

    protected boolean cardIsActive(String cardnumber) {
        // access to merchant system's web service
        // omitted for brevity...
    }
}
```

Extract and Override Call

Break dependence in a local method

Dependencies

1.

```
public void rebindStyles() {
   styles = StyleMaster.formStyles(template, id);
   ...
}
```

2.

```
public void rebindStyles() {
   styles = formStyles(template, id);
   ...
}

protected List formStyles(StyleTemplate template, int id) {
    return StyleMaster.formStyles(template, id);
}
```

Create a new method and copy the call

3.

```
public class TestingPageLayout extends PageLayour {

    protected List formStyles(StyleTemplate template, int id) {
        ...
        return new ArrayList();
    }
}
```

Introduce a testing subclass and override this new method

Extract Interface

```
Transaction t = new PaydayTransaction(getTestingDatabase(), new TransactionLog());
```

dependencies that we want to get rid of

```
void testPayday()
{
    FakeTransactionLog aLog = new FakeTransactionLog();
    Transaction t = new PaydayTransaction(getTestingDatabase(), aLog);
    t.run();

    assertEquals(getSampleCheck(12), getTestingDatabase().findCheck(12));
}
```

1. Extract Interface of TransactionLog
2. Implement fake class

Odd-e

Seeing Responsibilities

One annoying case about legacy code: God Class

Classes that assume too many responsibilities is hard to maintain:

- Difficult to read
- Difficult to maintain
- Difficult to identify the impact of change

Remember: Single Responsibility Principle!

Responsibilities

- Doing:
  - Doing something itself:
    - Creating an object or doing a calculation
  - Initiating action in other objects
  - Controlling and co-ordinating activities in other projects
- Knowing:
  - about private data
  - about related objects
  - about things it derive or calculate

- Bigger responsibilities take several classes
  - Domain model helps with “knowing”
  - Interaction diagrams help with “doing”

Heuristic: Group methods

- Look for similar method names
- Write down all of the methods on a class, along with their access types
- Try to find ones that seem to go together

Heuristic: Hidden methods

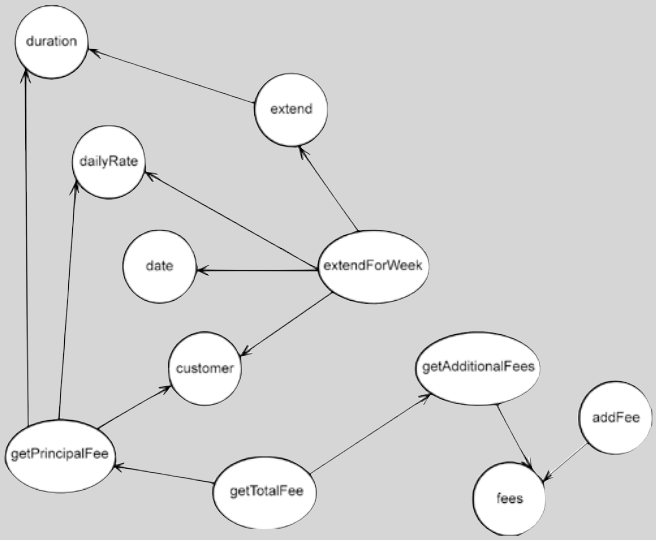
- If a class has many of private and protected methods, it often indicates that there is another class in the class
- If you have the urge to test a private method, the method should not be private; and
- if making the method public bothers you, chances are, it is because it is part of a separate responsibility.

Heuristic: Decisions that can change

- Is there some way of doing something that seems hard-coded? for example:
  - Talking to a database
- Can you imagine it changing?

Heuristic: Internal relationships

- Are certain instance variables used by some methods and not others?
- Can we identify cohesive subsets of instance variables?



Heuristic: Primary Responsibility

- Try to describe the responsibility of the class in a single sentence

Odd-e

Hidden state in a method?

- Long method?
- Single responsibility principle

Difficult setup?

- Too much coupling?
- Enough indirection
- Dependency injection

State-leak across tests?

- Improper use of singletons?

Testing private methods?

- Single Responsibility Principle violations?
- High cohesion?

Fragile mocking?

- Why not using the real thing? bad coupling?
- Law of Demeter
- Dependency injection

Odd-e

Software engineering?

- When you look at software, do you get the feeling that it has been developed without adult supervision?
- Is software more critical to your business and yet becoming bloated and buggier?
- Are projects taking longer and delivering less than promised?
- Is licensing software professionals really going to make any differences?
- The “software crisis” has existed for years and there is still a shortage of good software developers, because mechanical metaphors for software development don’t work

What is craftsmanship

- Nature of learn programming
  - Innate knowledge
  - Learned skills
  - Hard-won rules of thumb

The only way to learn is by learning from those more experience than you
This is craftsmanship

Software craftsmanship Manifesto

Manifesto for Software Craftsmanship

Raising the bar.

As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft. Through this work we have come to value:

Not only working software,
but also **well-crafted software**

Not only responding to change,
but also **steadily adding value**

Not only individuals and interactions,
but also **a community of professionals**

Not only customer collaboration,
but also **productive partnerships**

That is, in pursuit of the items on the left we have found the items on the right to be indispensable.

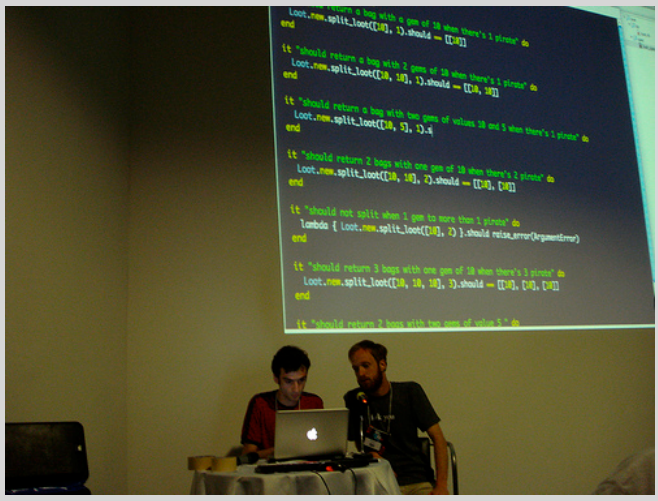
Software Craftsman

- Take responsibility
- No broken windows
- Quality is not an option
- Growing yourself
- Communicate!
- Professionalism

Practising

- Coding Dojo
  - Kata
  - Randoori
- Code Retreat

- Further broadening with OSS



Coding Dojo in Rio 2009 http://tinyurl.com/3pjyprt

Are you professional?

Don't write bad code

Never be blocked

Use Good Tools

Avoid Debugging

Apprenticeship

Short Iterations

Incremental Improvement

Avoid Turgid Viscous Architectures

100% Code Coverage

Clean Code

QA should find nothing

Test through right interface

TDD

Abstract away volatility

Progressive Deepening

De-couple from others

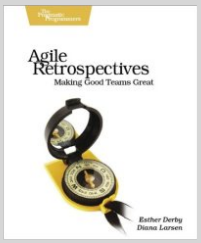
Definition of Done

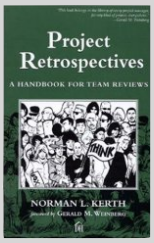
No Grand Redesigns

Manual Test Scripts are Immoral

Sprint Retrospective

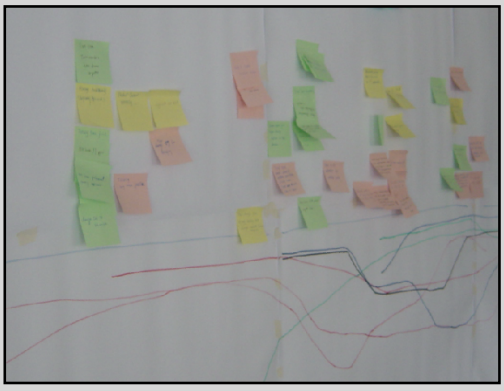
- Process improvement at end of every Sprint
- Facilitated by ScrumMaster
- What went well, what could be improved.
- Team devises solution to most vexing problems
- “Project Retrospectives,” Norman Kerth

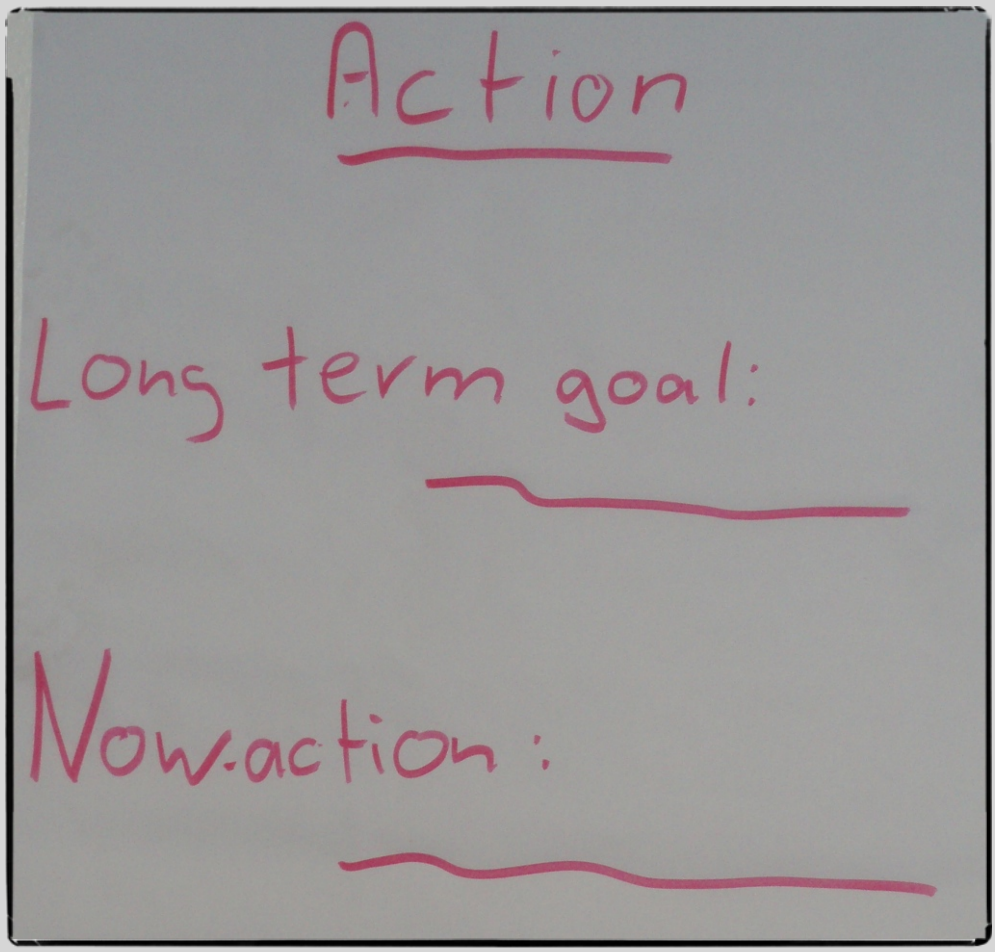




Retrospective Techniques

- Check previous actions first. If not done, retrospect on them.
- Only select a couple actions and really DO them.
- Focus on an constructive attitude.
  - “What can WE do”.
- Action planning: http://www.scrumalliance.org/articles/61-plan-of-action



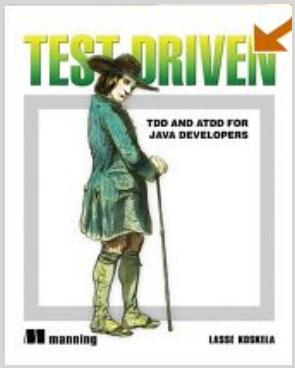


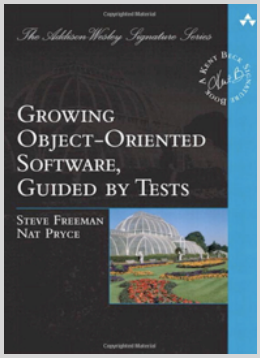
Things to check in Retrospectives

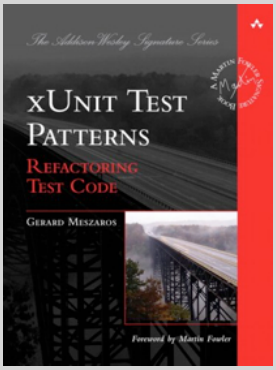
- Actions:
  - Do we only have a few actions?
  - Are they considered useful?
  - Are they implemented?
- Is “done” extended?
- Updating our working agreements?
- Do we require:
  - Tasks in the Sprint backlog?
  - Items in the Product backlog?

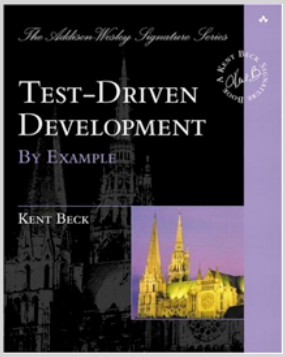
References

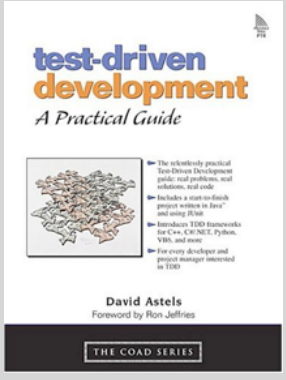
Test-Driven Development



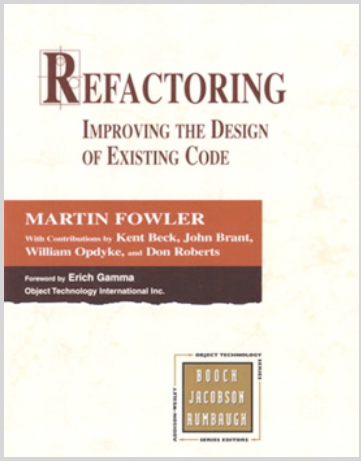


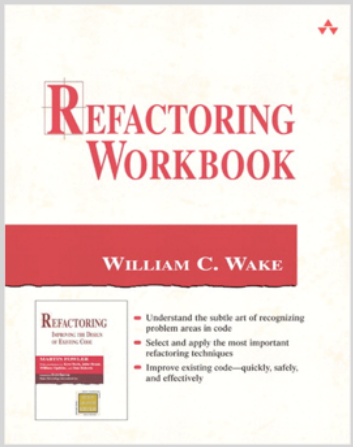


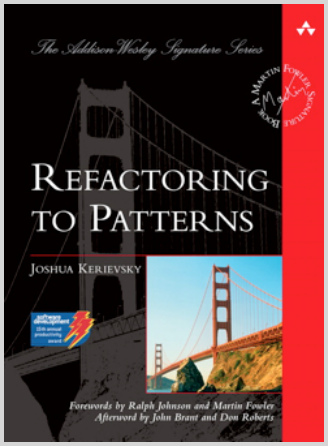




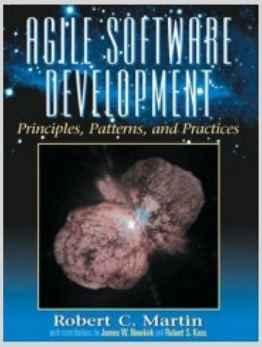
Refactoring

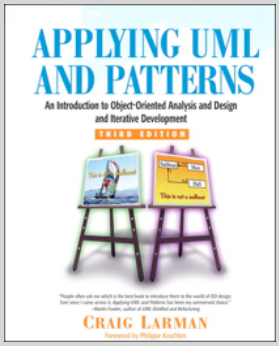


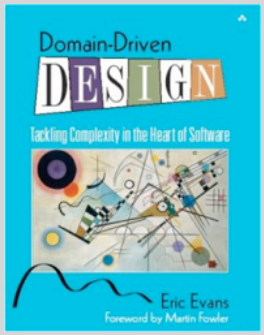


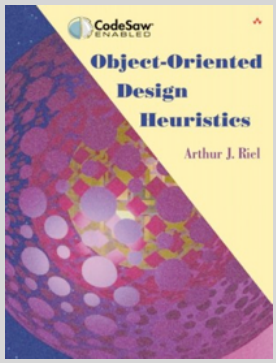


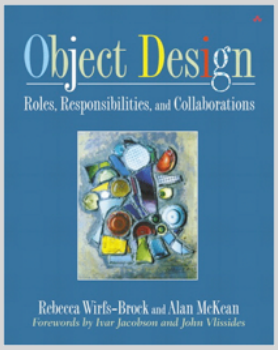
Object-Oriented Design



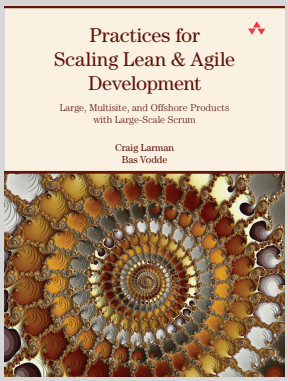


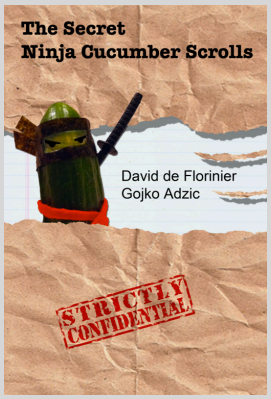


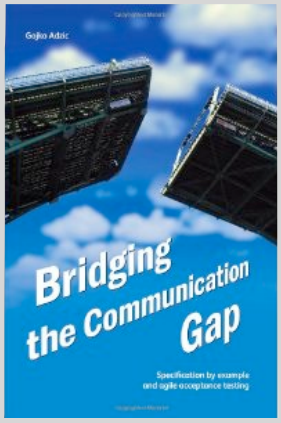


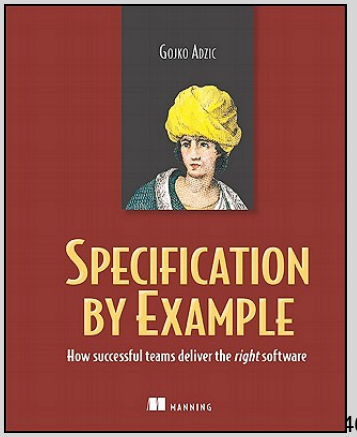


Acceptance-TDD

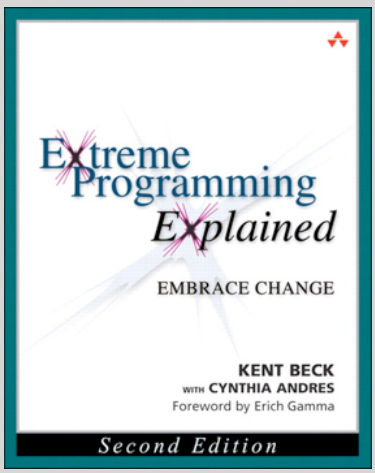


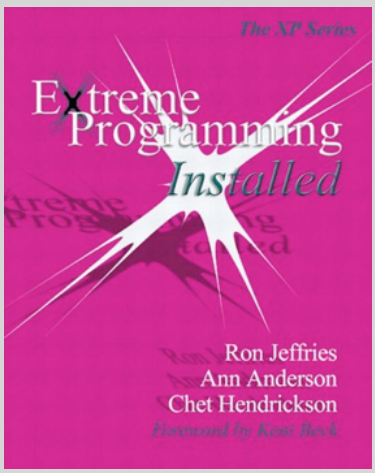




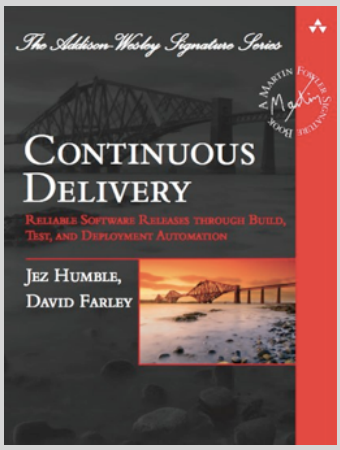


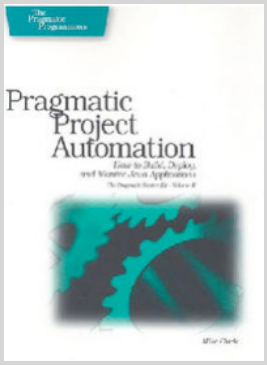
Extreme Programming

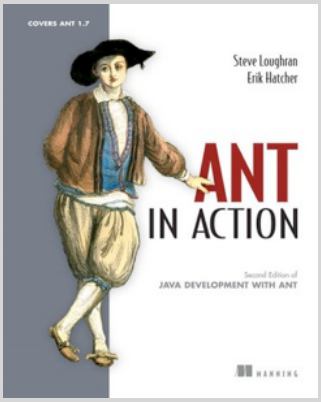


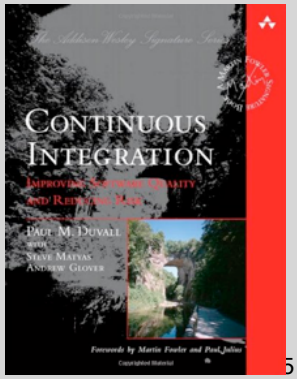


Automation



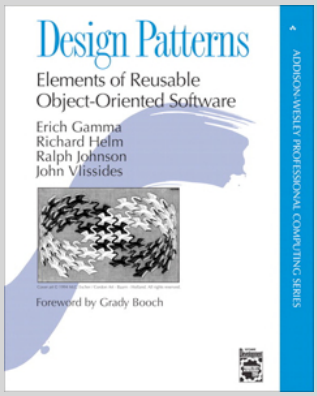


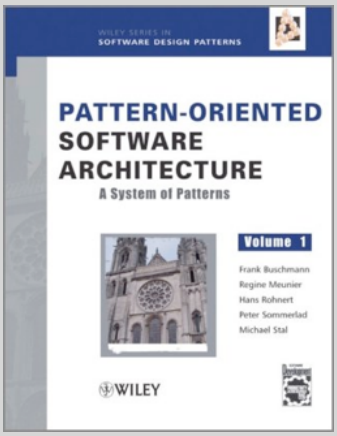


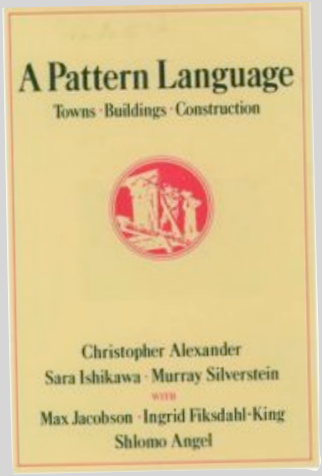


5

Patterns

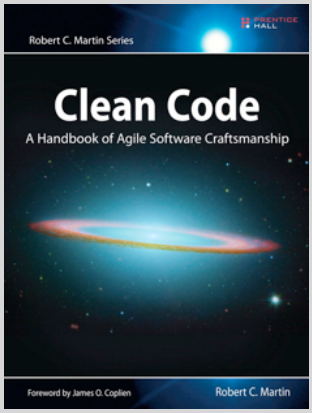


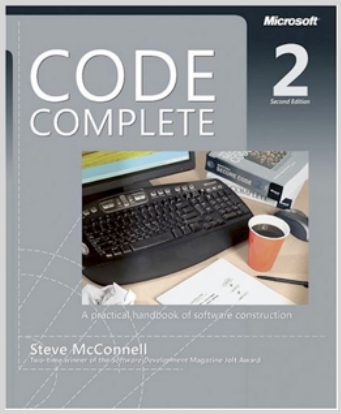


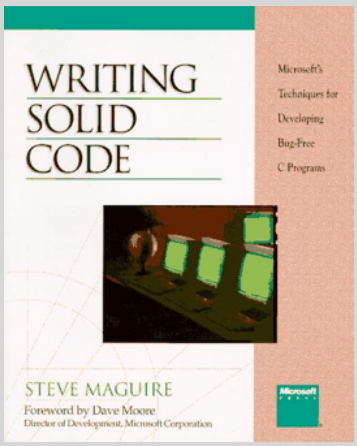


Not software

Code

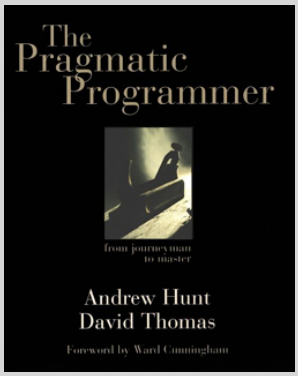


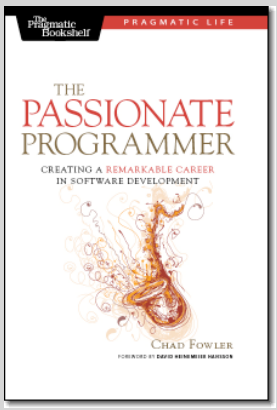


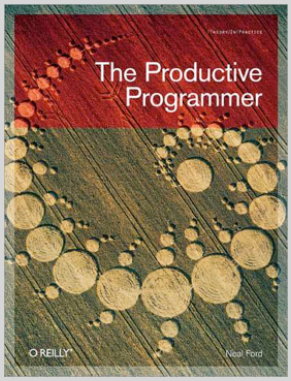


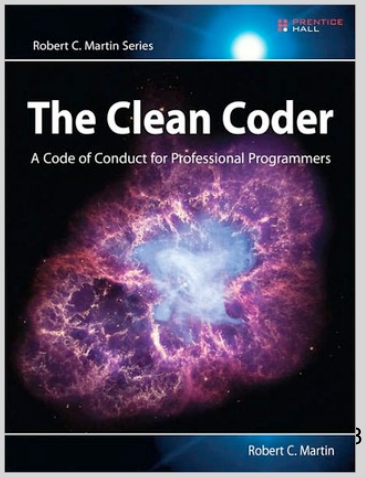
Old

Attitude!

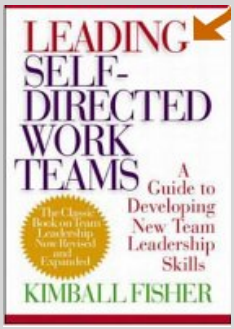


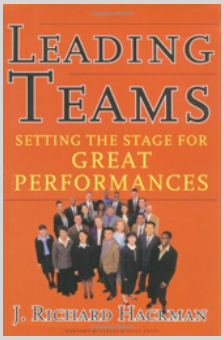


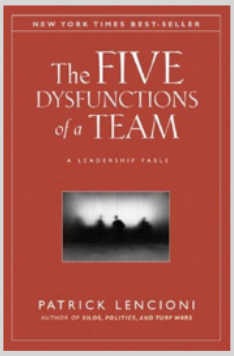


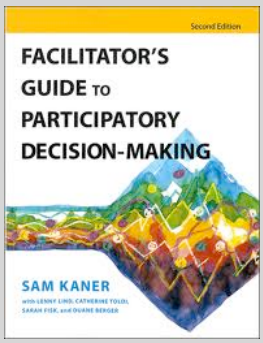


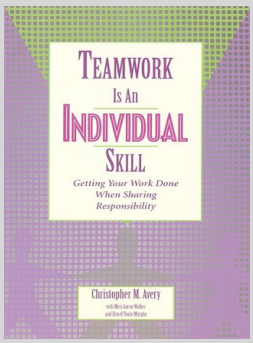
Books - Teams











Software that got harmed during this course

- **Card Me** http://sourceforge.net/projects/cardme/
- **Church Software** http://sourceforge.net/projects/churchsoftware/
- **Buddi** http://sourceforge.net/projects/buddi/
- **Bulkmailer** http://sourceforge.net/projects/bulkmailer/
- **GnuAccounting** http://sourceforge.net/projects/gnuaccounting/
- **StudyTrainer** http://sourceforge.net/projects/studytrainer/
- **University ERP System** http://sourceforge.net/projects/university/
- **Yapbam** http://sourceforge.net/projects/yapbam/
- **Yamm** http://sourceforge.net/projects/yammjava/

We mean no harm to the open source software movement, these open source projects, nor their authors. We are happy that they've contributed their source code to the software development community at large and that we have the opportunity to learn from others.

#	Item	Area	Value	Size
Refinement				
1	As an infrequent user, I want to send a pre-written email to multiple recipients at once. (by uploading a text file)	core	1	8
2	As an infrequent user, I can edit an email before sending it.	core	5	3
3	As a contact admin, I can add a recipient.	core	10	5
4	Add attribute to a recipient.	core	10	3
5	As an infrequent user, I want to know when the mail server reports an error for an email address.	core	10	5
6	As an infrequent user, I don't want to be able to type a newline in the subject field.	core	90	2
7	As an infrequent user, I can write in HTML.	rich content	15	3
8	As an infrequent user, I want to have a quick summary of all the people that will be mailed. (dry-run)	core	15	3
9	As any user, I want to register so that I can use the application.	user management	15	5
10	As an infrequent user I want to import contacts from vCard files. (We've bought a license for MAD vCard Pro for this.)	io	15	21
11	As any user I want to login with secret credentials so that only I have access to my data.	user management	20	8
12	As a contact admin, I want to have validity check on my email address when I add it.	core	20	2
13	As an infrequent user, I can add an attribute to a mail subject.	personalization	15	13
14	As an infrequent user, I want to be notified when the email gets bounced.	core	20	8
15	As an infrequent user, I want to preview the email before I send.	core	25	5
16	As any user, I want to setup custom SMTP server. Username/password/login	config	25	3
17	SSL for SMTP	config	30	5
18	Import CSV	io	15	21
19	Import from Google Spreadsheet	io	65	21
				Total of 144 points in Refinement
Current Release				
	Remove recipient from named list	core	30	2
	As an infrequent user, I want to automatically select recipients based on an attribute.	personalization	85	13
	As an infrequent user, I want to sort recipients based on attribute value.	core	85	21
	As an infrequent user, I want to be warned when the email subject is empty.	core	30	2
	As an infrequent user, I want to add multiple recipients to CC.	core	65	2
	Add recipient to named list	core	24	5
	Delete all recipients	core	55	2
	Create a "standard attribute"	core	40	5
	As an infrequent user, I can select some recipients from all recipients so that I have more flexibility.	core	45	3
	As any user I want to select from a list of popular mail servers so that it is easy to configure without knowledge of technical details. (Gmail, Yahoo, Hotmail)	config	30	5
	As an infrequent user, I can save the email for later use.	core	85	3
	As an infrequent user, I want to list all contacts who have a certain attribute.	core	65	8
	As an infrequent user, I want to list all contacts where an attribute has a specific value.	core	70	8
	As a contact admin, I want to compose lists from a list of lists so that I don't have to add all of those contacts to the new list by hand.	core	95	13

#	Item	Area	Value	Size
	As an infrequent user, I want to be able to fix and resend rejected email addresses.	core	20	8
	As a frequent user, I want to select email from all emails to send.	template	95	8
	As a contact admin, I want to create a named recipient list so that I can easily send future emails to the same people.	core	20	5
	Import Mac address book	io	70	21
	As a system admin, I can create any user so that they can send emails.	user management	110	5
	As an infrequent user, I can send a mail with an attribute in the email body so that the recipient feels personal.	personalization	20	8
	Import Excel	io	25	13
	As any user, I want the emails to continue sending even when I logout so that I do not have to wait for all emails to finish sending.	core	95	21
	As any user, I want to logout so that I can leave my computer knowing that somebody else won't send spam on my behalf.	user management	50	1
	Remove an attribute from a recipient.	contacts	65	2
	As a system admin, I want to list all users so that I can have an overview.	user management	70	2
	As an infrequent user, I want to be able to add recipients to CC from my contacts.	core	75	2
	Edit the name of an attribute.	contacts	55	2
	As an infrequent user, I want to be able to add one person to CC.	core	50	2
	Edit the value of an attribute.	contacts	25	2
	Delete recipient from contacts.	contacts	40	2
	As an infrequent user, I want to add one person to BCC.	core	60	2
	Delete named list.	contacts	90	2
	As a system admin, I want to remove any user from all users list.	user management	80	2
	As a frequent user, I want to keep a list of emails.	template	40	5
				Total of 351 points for whole release