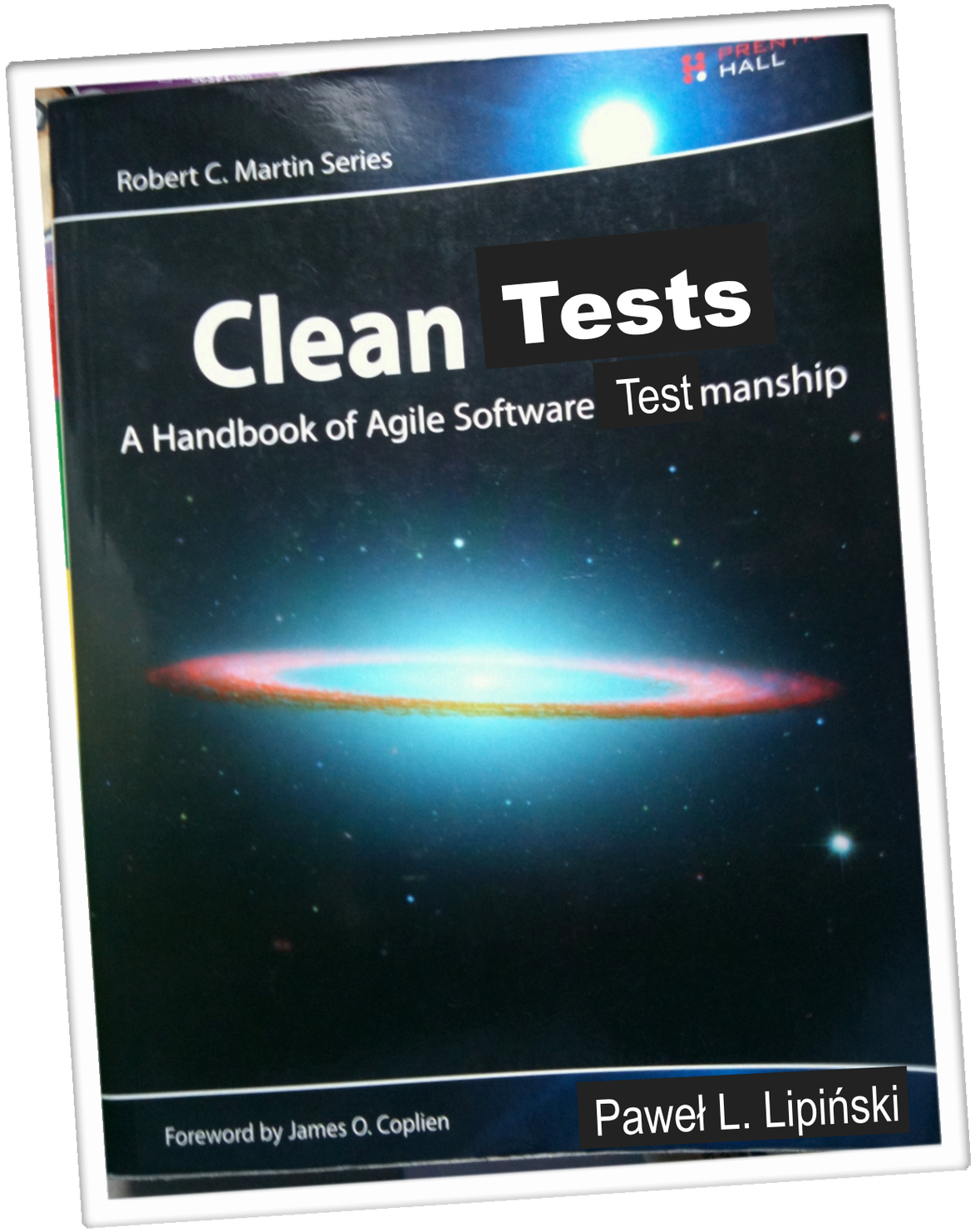


PRAGMATISTS

or how to write tests so that they serve you well

whoami

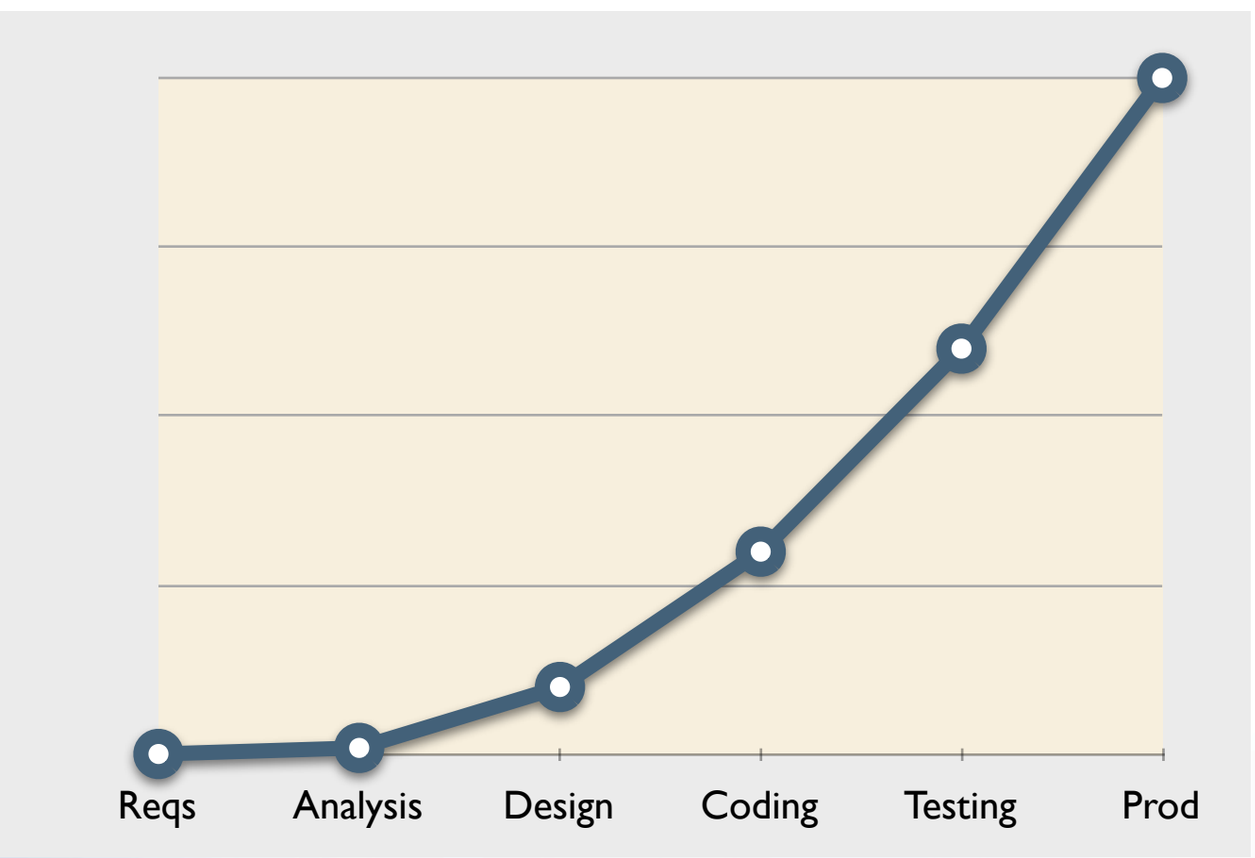
- ~16 years as a developer, ~12 years in Java
- programming, consulting, training, auditing, architecturing, coaching, team leading
- Formal & agile methods
- currently: developer, coach, ceo @ PRAGMATISTS

Cost of change grows exponentially with time

Barry Boehm

Ideas are cheaper to change than code.

Bugs found early are cheaper to fix.



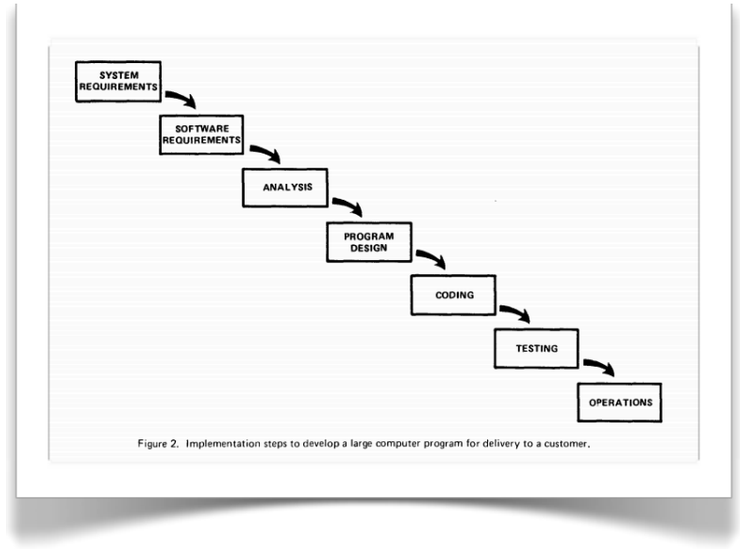
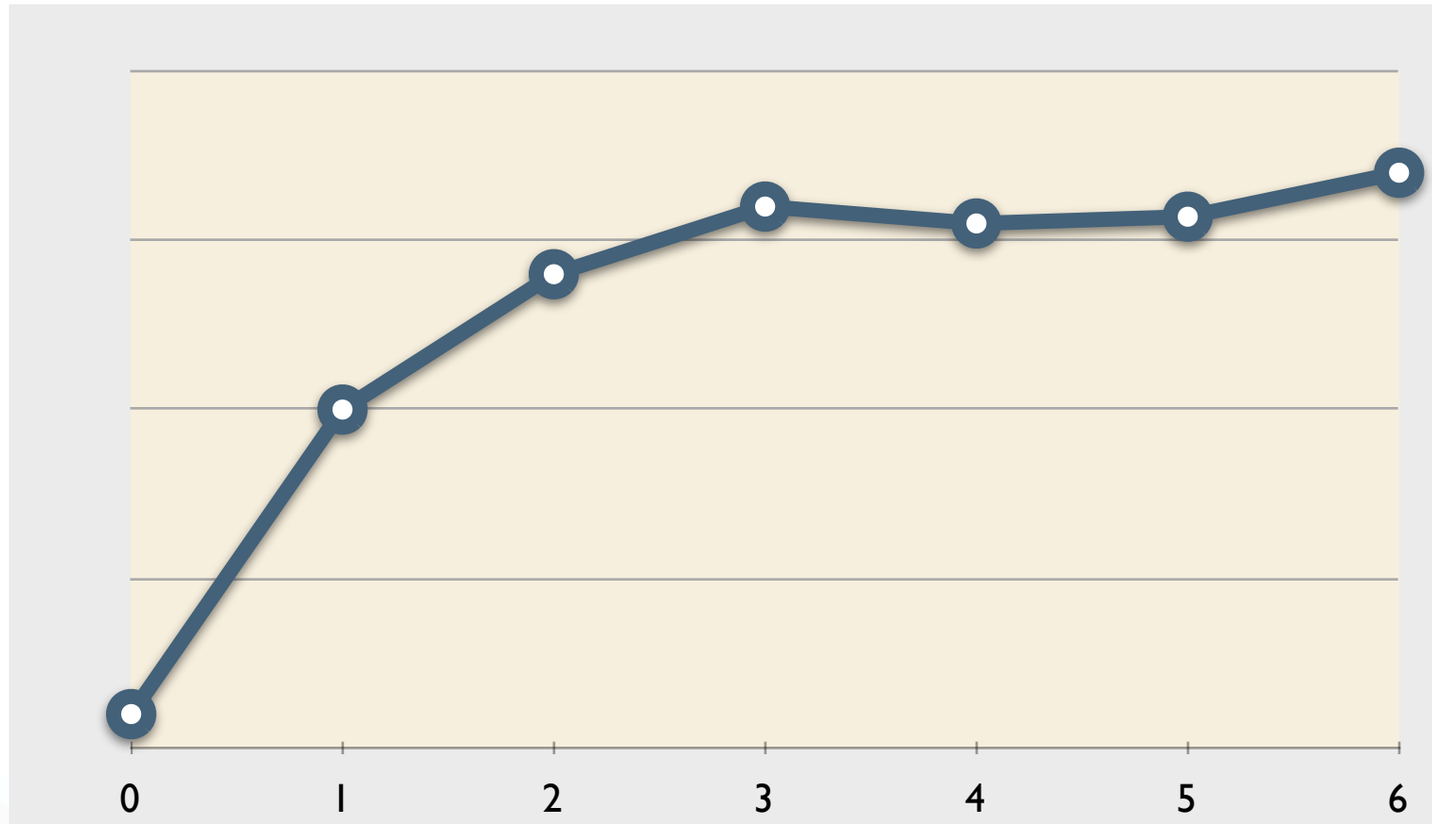


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

Does cost of change really grow exponentially with time?

Postponing decisions and **reducing feedback cycles** flattens the curve.



Ariane 5 flight 501
10 years of work
5.000.000.000 €

http://www.dutchspace.nl/uploadedImages/Products_and_Services/Launchers/Ariane%205%20Launch%20512%20-%20ESA.JPG

http://www.capcomespace.net/dossiers/espace_europeen/ariane/ariane5/AR501/V88%20explosion%2003.jpg

Oops - an error has occurred - Mozilla Firefox

File Edit View History Bookmarks Tools Help

jira.com https://dev1.jira.com/secure/admin/StudioImport!showUrls.jspa

Google

Diigo Bookmark Highlight Comment Send Message (0) Read Later Unread Recent Add a filter Options Capture

Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options

[#JST-24... [#JST-28... [#JST-24... Studio sy... Oops... How to b... Apache F... [#STRM-... Maven R... Index of /... Sonatype... Index of /... CR-JST-2... Uploaded... Atlassian... JIRA Studi... Disabling... Maven D...

Oops - an error has occurred

System Error

A system error has occurred.

Please try submitting this problem via the Support Request Page
Otherwise, please create a support issue on our **support system** at http://support.atlassian.com with the following information:

1. a description of your problem
2. cut & paste the error and system information found below
3. attach the application server log file (/data/jirastudio/jira/log/atlassian-jira.log)

Cause:
java.lang.NullPointerException

Stack Trace: [hide]

```
java.lang.NullPointerException
        at com.atlassian.jira.studio.importer.StudioImport.doShowUrls(StudioImport.java:52)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at webwork.util.InjectionUtils$DefaultInjectionImpl.invoke(InjectionUtils.java:70)
        at webwork.util.InjectionUtils.invoke(InjectionUtils.java:56)
        at webwork.action.ActionSupport.invokeCommand(ActionSupport.java:433)
        at webwork.action.ActionSupport.execute(ActionSupport.java:157)
        at com.atlassian.jira.action.JiraActionSupport.execute(JiraActionSupport.java:53)
        at com.atlassian.jira.studio.importer.StudioImport.execute(StudioImport.java:42)
        at webwork.dispatcher.GenericDispatcher.executeAction(GenericDispatcher.java:139)
        at com.atlassian.jira.web.dispatcher.JiraServletDispatcher.service(JiraServletDispatcher.java:171)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:803)
        at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:269)
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:188)
        at com.atlassian.core.filters.HeaderSanitisingFilter.doFilter(HeaderSanitisingFilter.java:44)
        at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:215)
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:188)
        at com.atlassian.plugin.servlet.filter.IteratingFilterChain.doFilter(IteratingFilterChain.java:46)
        at com.atlassian.plugin.servlet.filter.ServletFilterModuleContainerFilter.doFilter(ServletFilterModuleContainerFilter.java:55)
        at com.atlassian.plugin.servlet.filter.ServletFilterModuleContainerFilter.doFilter(ServletFilterModuleContainerFilter.java:41)
        at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:215)
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:188)
        at com.atlassian.jira.web.filters.accesslog.AccessLogFilter.executeRequest(AccessLogFilter.java:99)
        at com.atlassian.jira.web.filters.accesslog.AccessLogFilter.doFilter(AccessLogFilter.java:83)
        at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:215)
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:188)
        at com.atlassian.jira.security.xsrf.XsrfTokenAdditionRequestFilter.doFilter(XsrfTokenAdditionRequestFilter.java:50)
        at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:215)
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:188)
        at com.opensymphony.module.sitemesh.filter.PageFilter.parsePage(PageFilter.java:119)
        at com.opensymphony.module.sitemesh.filter.PageFilter.doFilter(PageFilter.java:55)
```

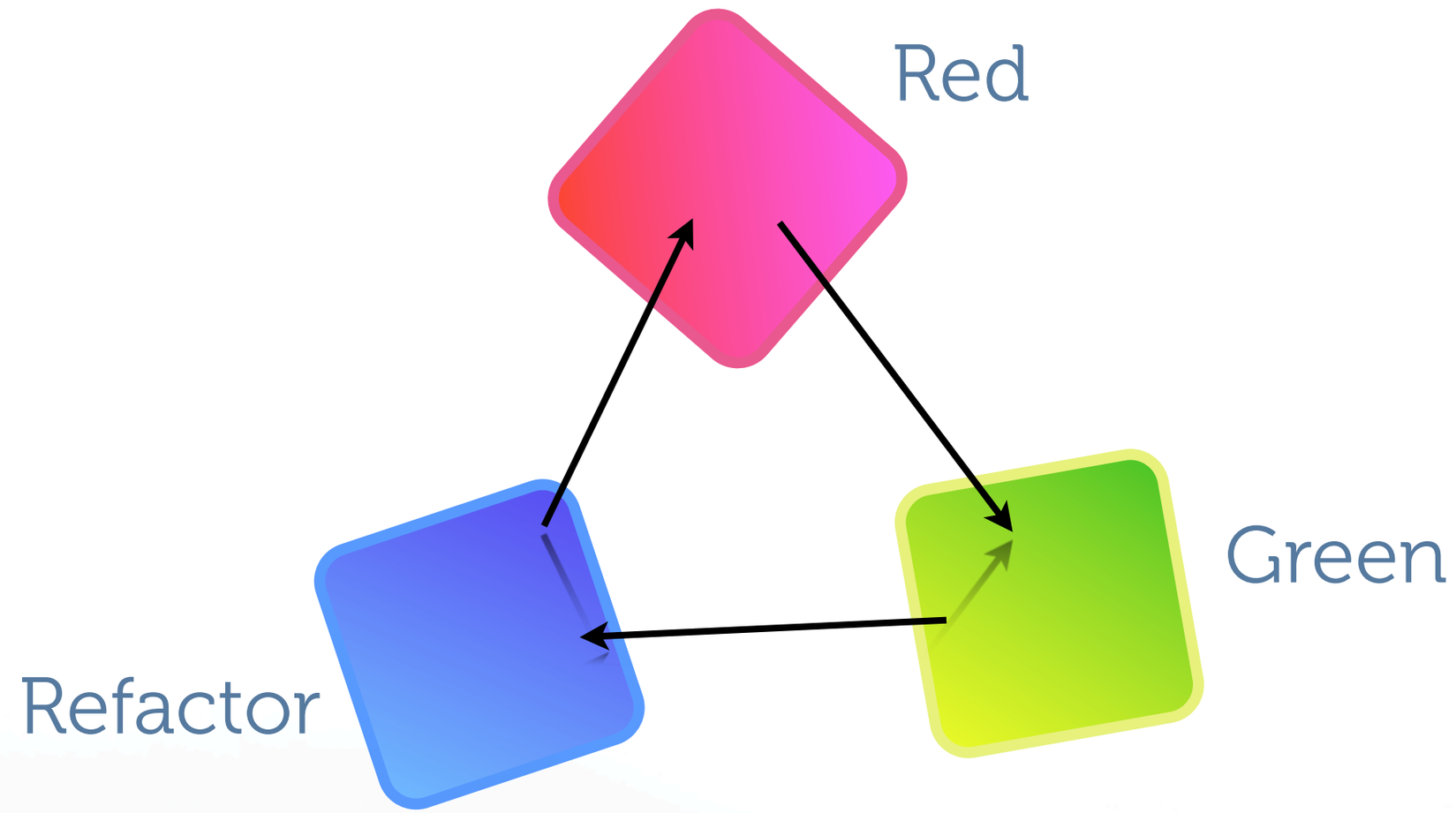
Done

0

Oops - an er... OSGi Servic... STAC - fabio Kwallet ma... Liferea 09:49 am

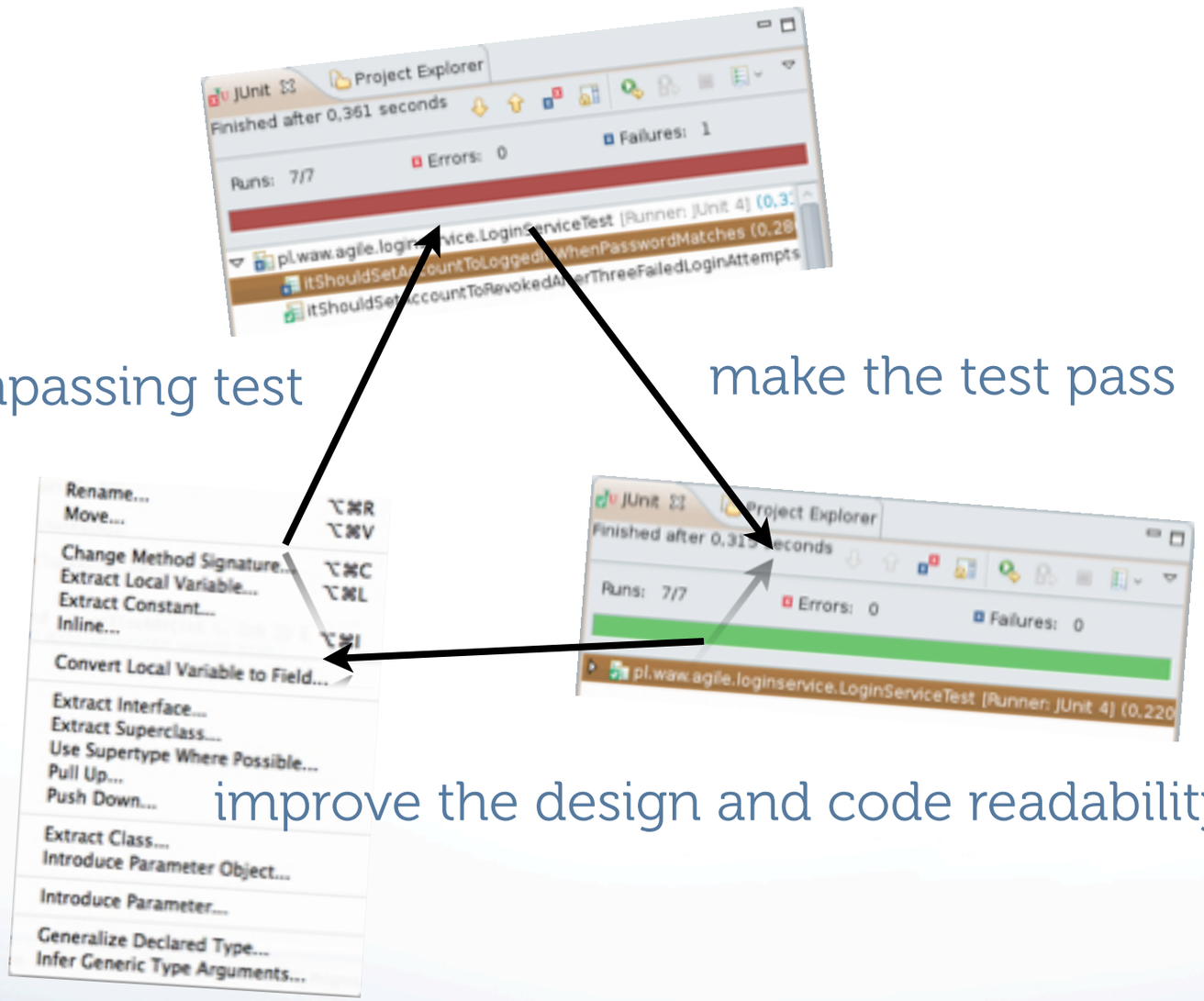
Test-Driven Development

A technique of software development based on repeating a short cycle:



Test-Driven Development

A technique of software development based on repeating a short cycle:



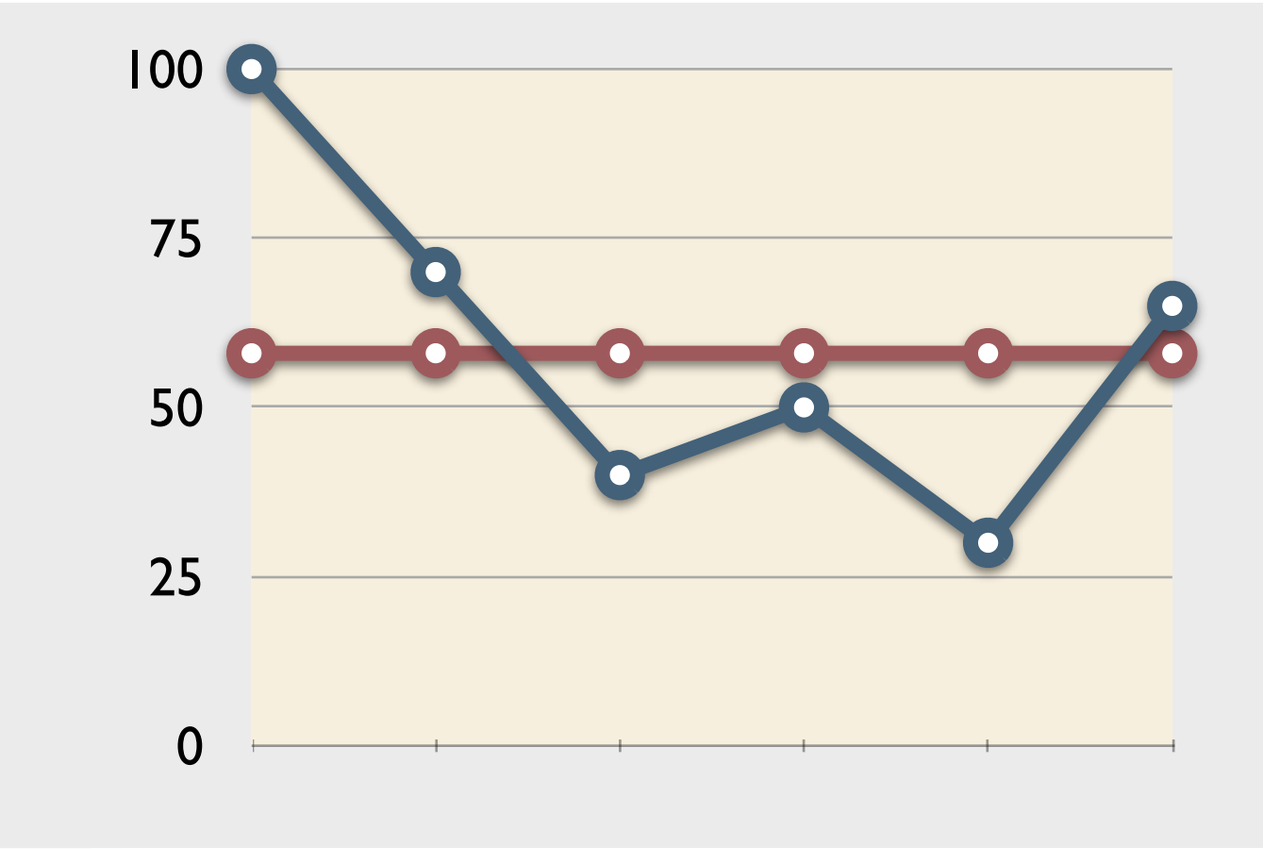
By continuously improving the design of code, we make it ***easier and easier*** *to work with.*

This is in sharp contrast to what typically happens: little refactoring and a great deal of attention paid to expediently adding new features.

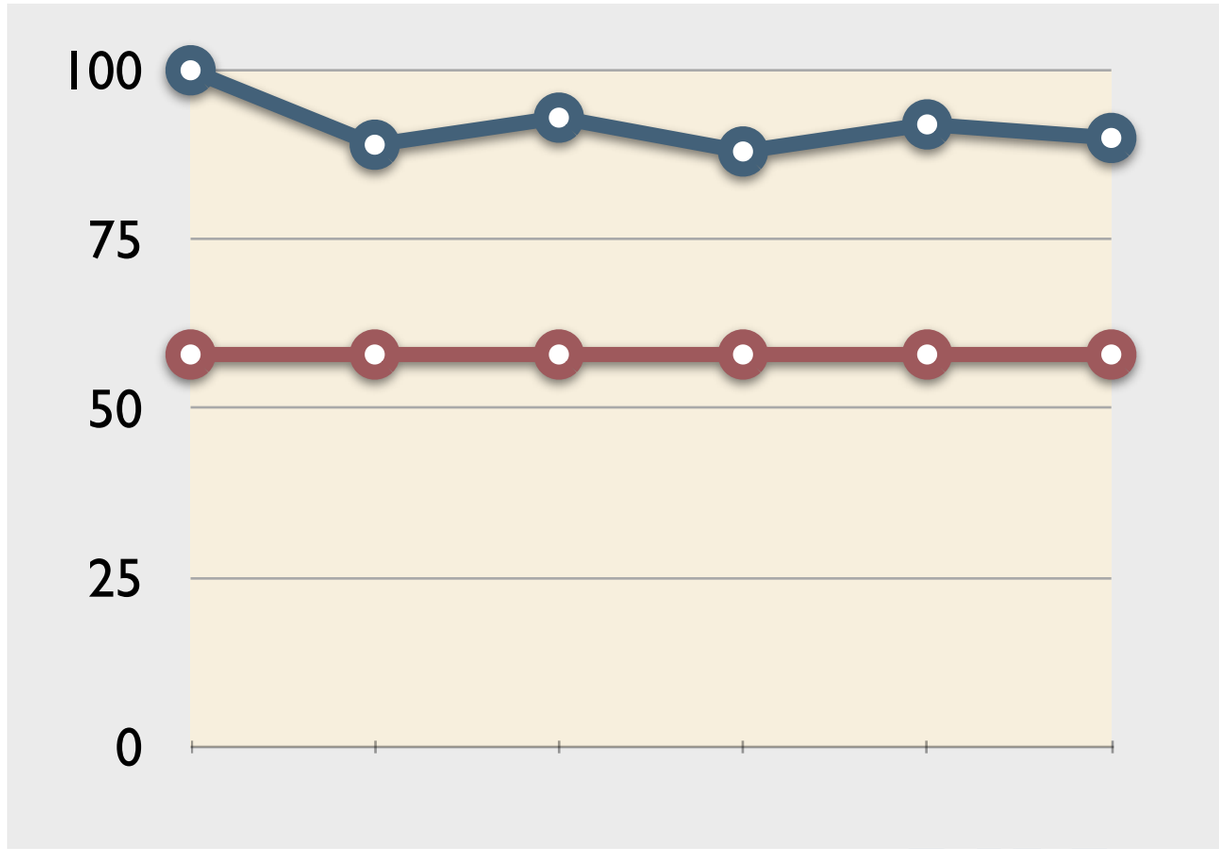
If you get into the hygienic habit of refactoring continuously, you'll find that it is easier to extend and maintain code.

Joshua Kerievsky,
Refactoring to Patterns

Acceptable quality



Traditional development



TDD

http://flagstaffclimbing.com/wp-content/themes/climbing/images/flag-climbing-bg.jpg

Ok, but... how to make it stick?

Do it with the whole team.

Get a coach to spend time with your team, on your code.

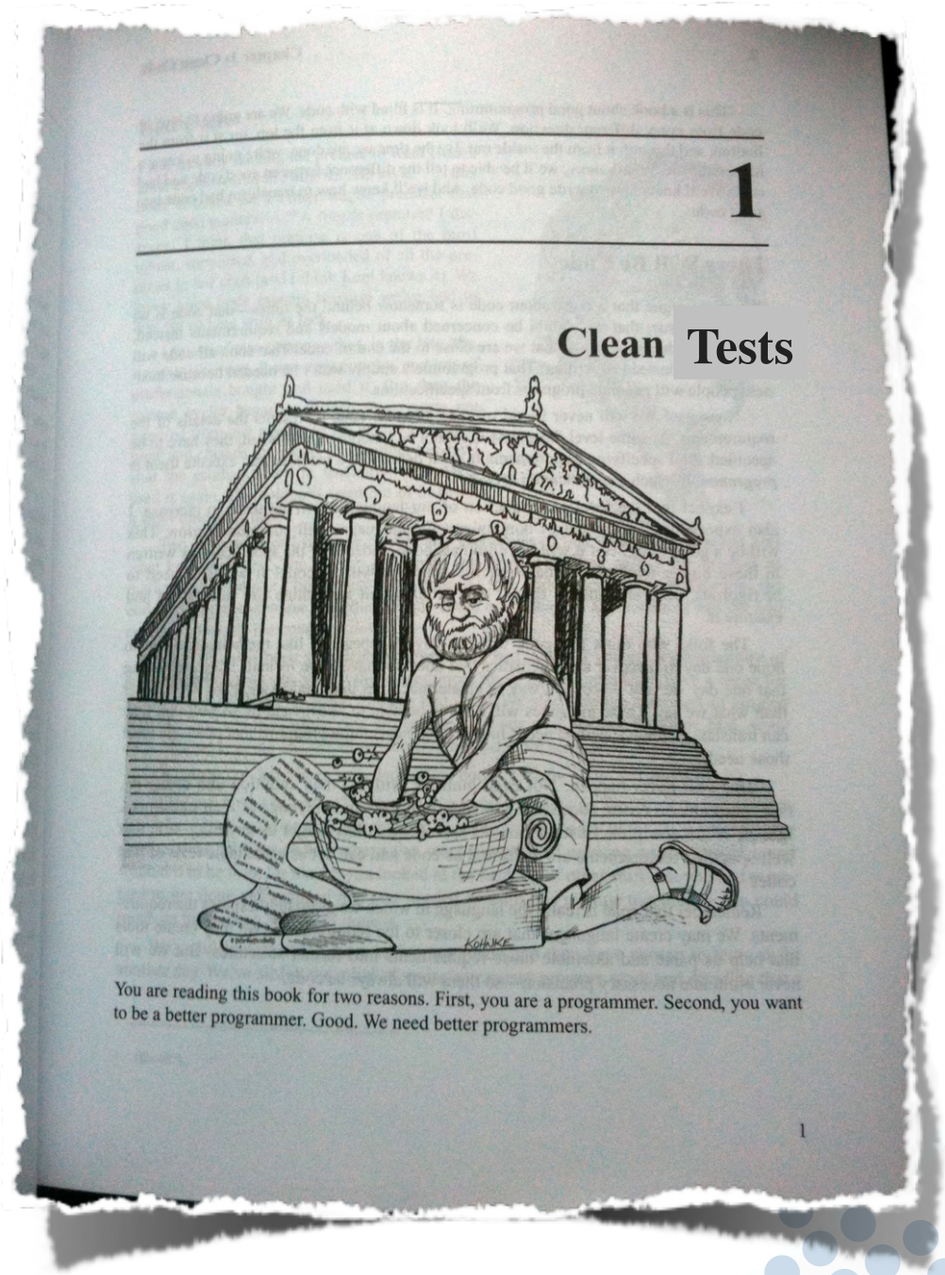
Learn it in pairs

Try kata trainings - it will build a habit of test-driving in your head.

Peer-review your test code

What do tests give us?

- awareness of what is supposed to be written
- feeling of safety while refactoring
- feeling of safety while changing code
- increase in development speed
- executable documentation



1

Clean Tests

You are reading this book for two reasons. First, you are a programmer. Second, you want to be a better programmer. Good. We need better programmers.

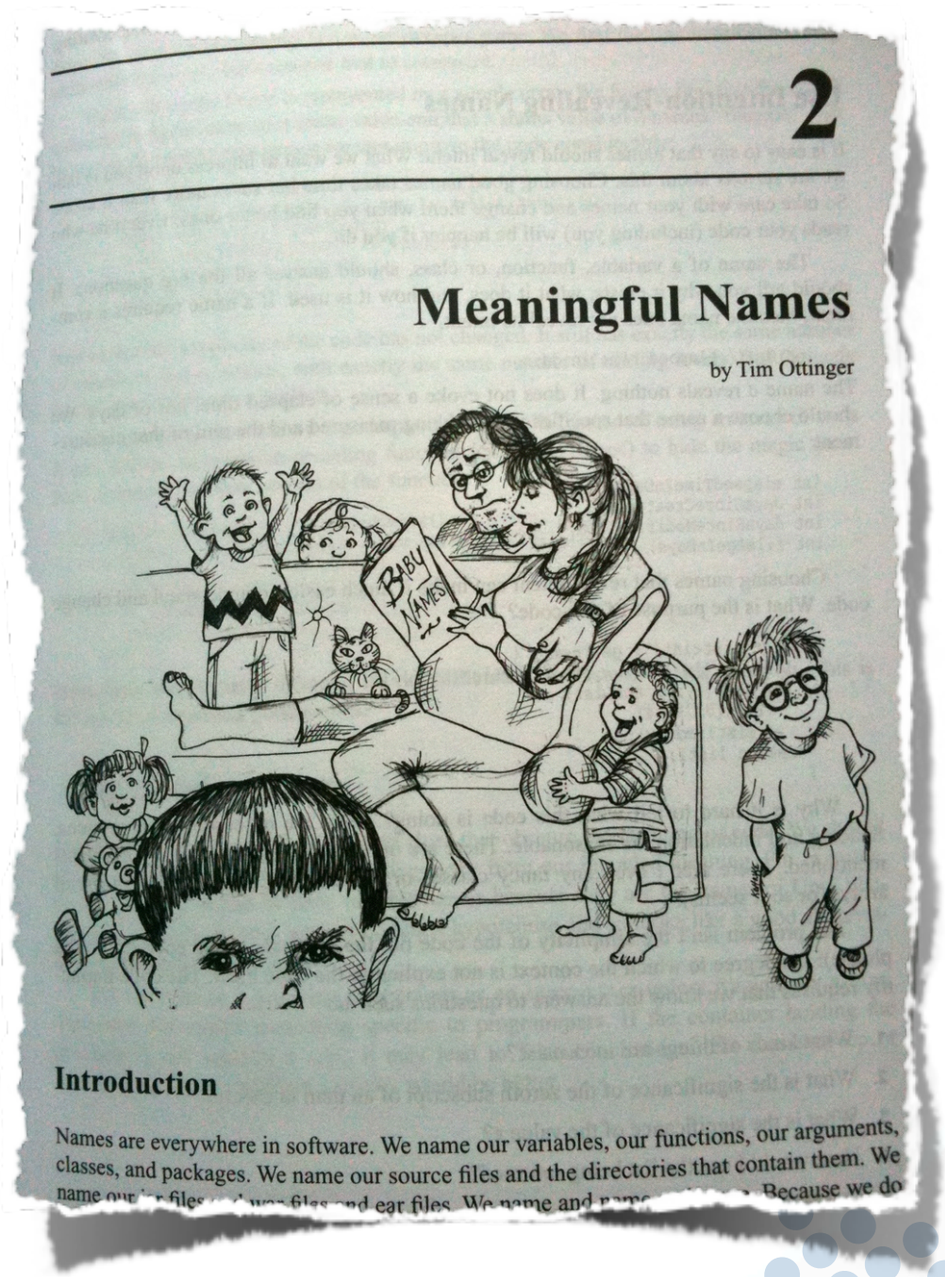
1

So what's the problem?

- Whenever you change something - they stop working
- They become harder and harder to maintain
- They soon start to look like this
- If there's many of them, it's hard to know where to look to learn something about the system

Conventions

- coherent naming of test classes
- coherent naming of tests
- test classes' names describing behaviour not the class' name
- test methods' names should describe test case, not method being tested
- code conventions



2

Meaningful Names

by Tim Ottinger

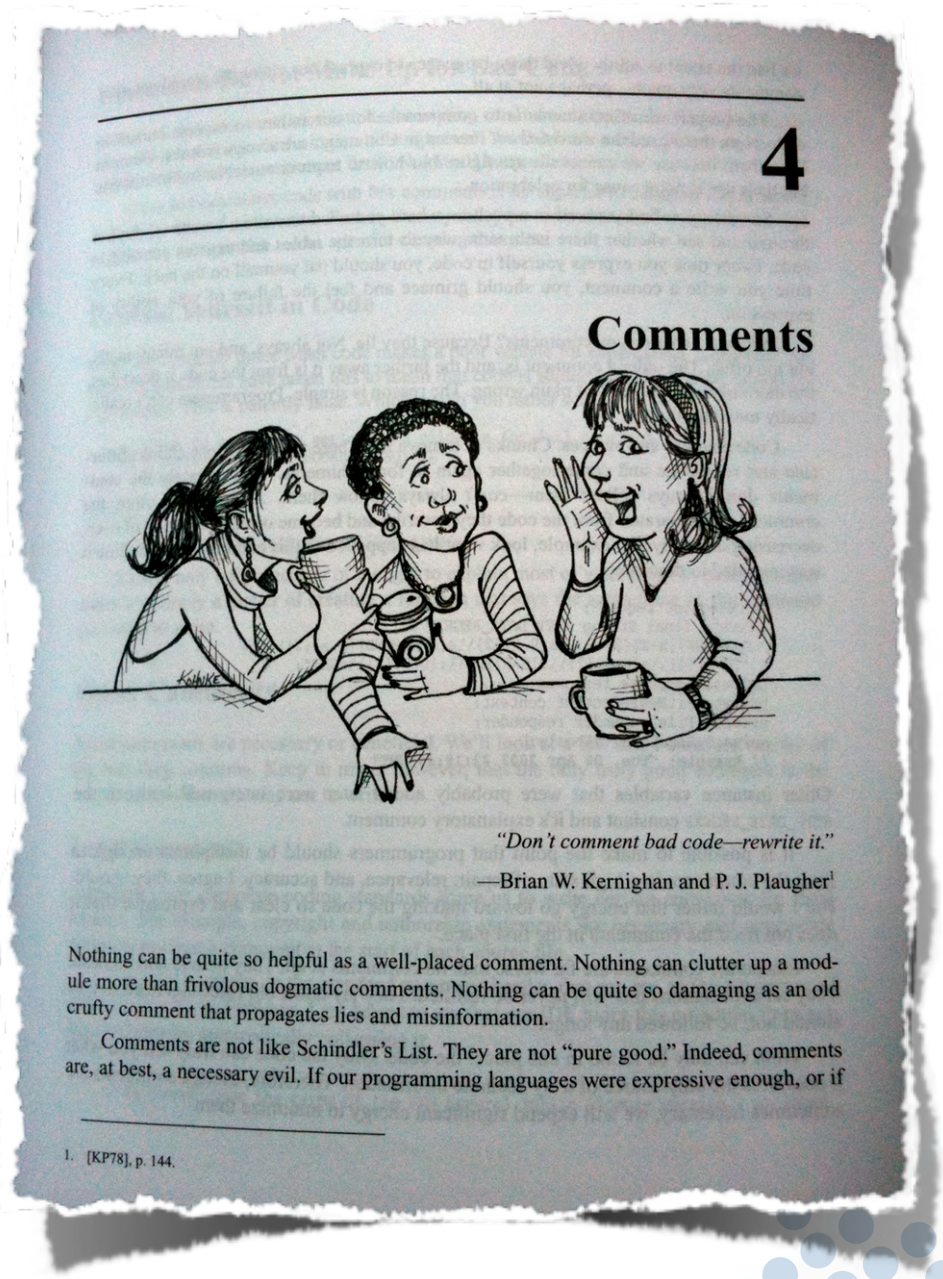
Introduction

Names are everywhere in software. We name our variables, our functions, our arguments, classes, and packages. We name our source files and the directories that contain them. We name our jar files and war files and ear files. We name and [illegible]. Because we do

Comments

- if you feel you must comment, something's wrong with your code
- tests should document code, so they must be SUPERCOMPREHENSIBLE
- though comments are useful sometimes...

```
// given

// when

// then
```



4

Comments

"Don't comment bad code—rewrite it."

—Brian W. Kernighan and P. J. Plaugher[1]

Nothing can be quite so helpful as a well-placed comment. Nothing can clutter up a module more than frivolous dogmatic comments. Nothing can be quite so damaging as an old crufty comment that propagates lies and misinformation.

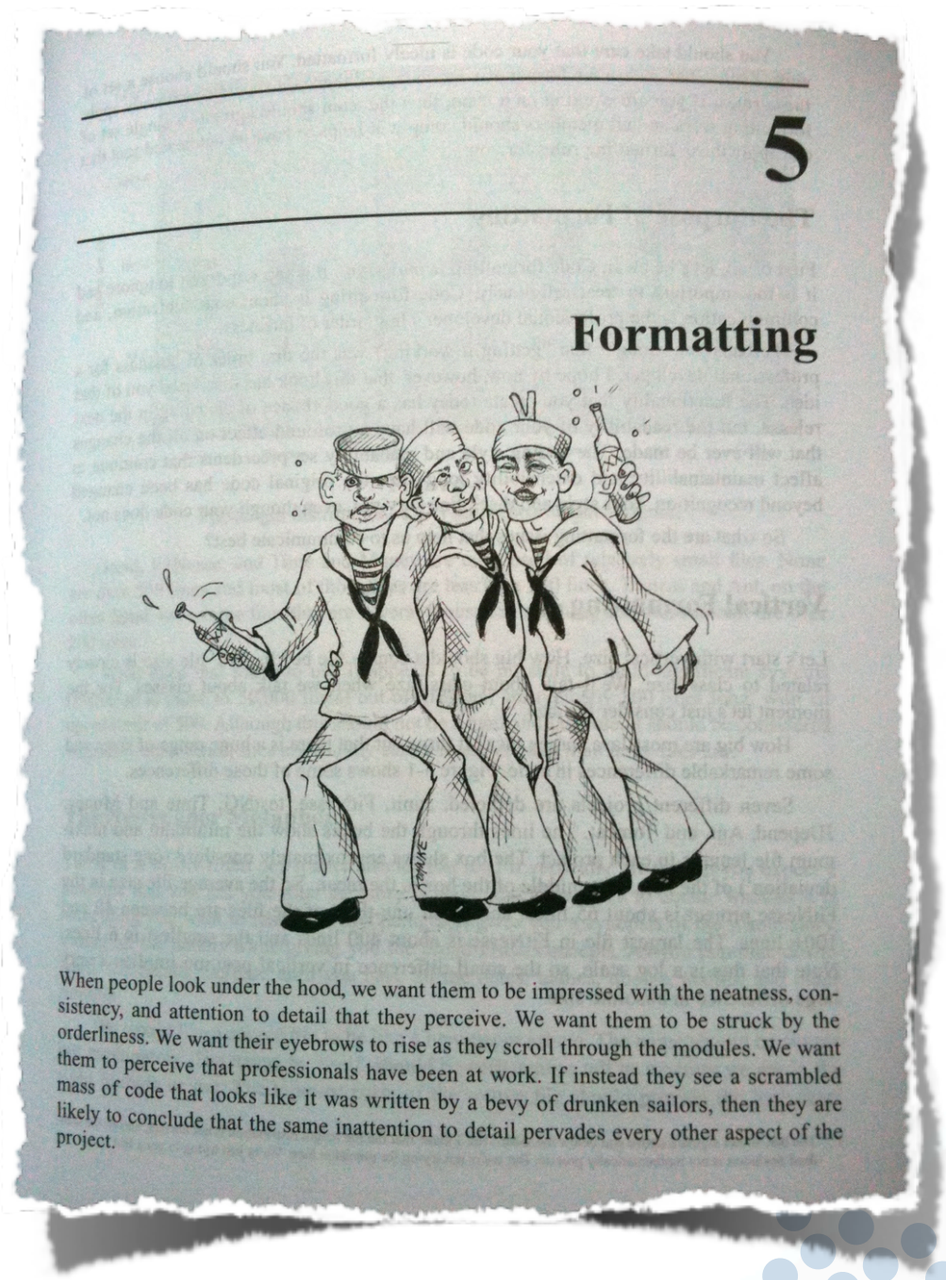
Comments are not like Schindler's List. They are not "pure good." Indeed, comments are, at best, a necessary evil. If our programming languages were expressive enough, or if

1. [KP78], p. 144.

Formatting

- coherent formatting throughout the codebase
- separation of logical fragments
- eyes used to it = quicker understanding

```
@Test
public void shouldFindStoryIfStartsWithFeature() {
    // given
    String expectedStoryName = "one";
    text = "Feature:" + expectedStoryName;

    // when
    loader.loadFrom(text);

    // then
    assertThat(nameOfFirstStory(), is(expectedStoryName));
}
```



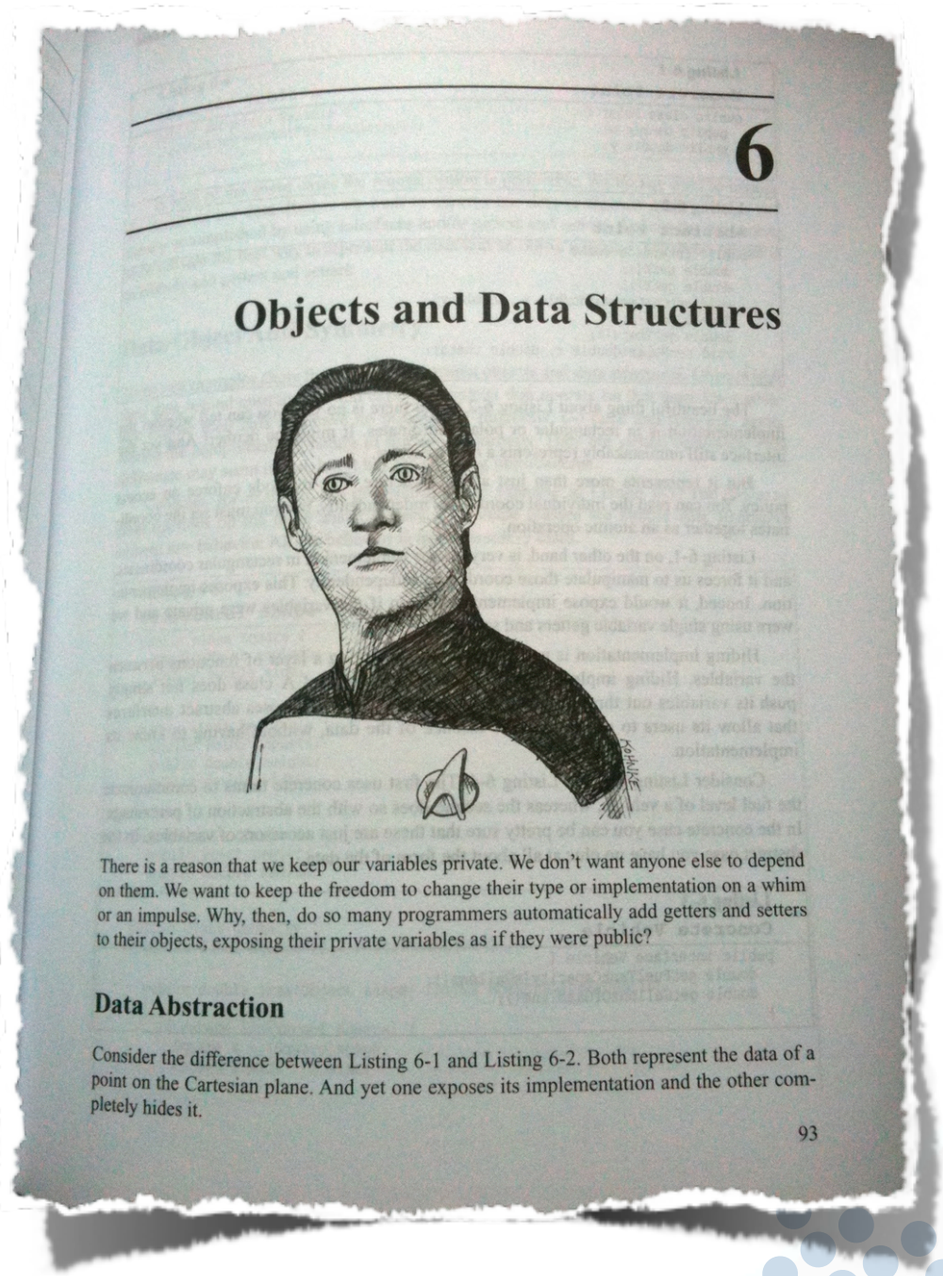
5

Formatting

When people look under the hood, we want them to be impressed with the neatness, consistency, and attention to detail that they perceive. We want them to be struck by the orderliness. We want their eyebrows to rise as they scroll through the modules. We want them to perceive that professionals have been at work. If instead they see a scrambled mass of code that looks like it was written by a bevy of drunken sailors, then they are likely to conclude that the same inattention to detail pervades every other aspect of the project.

Given (test setup)

- DRY (setup methods should be business-like)
- setup method COMPREHENSIBLE
- `@Before` vs. explicite call



6

Objects and Data Structures

There is a reason that we keep our variables private. We don't want anyone else to depend on them. We want to keep the freedom to change their type or implementation on a whim or an impulse. Why, then, do so many programmers automatically add getters and setters to their objects, exposing their private variables as if they were public?

Data Abstraction

Consider the difference between Listing 6-1 and Listing 6-2. Both represent the data of a point on the Cartesian plane. And yet one exposes its implementation and the other completely hides it.

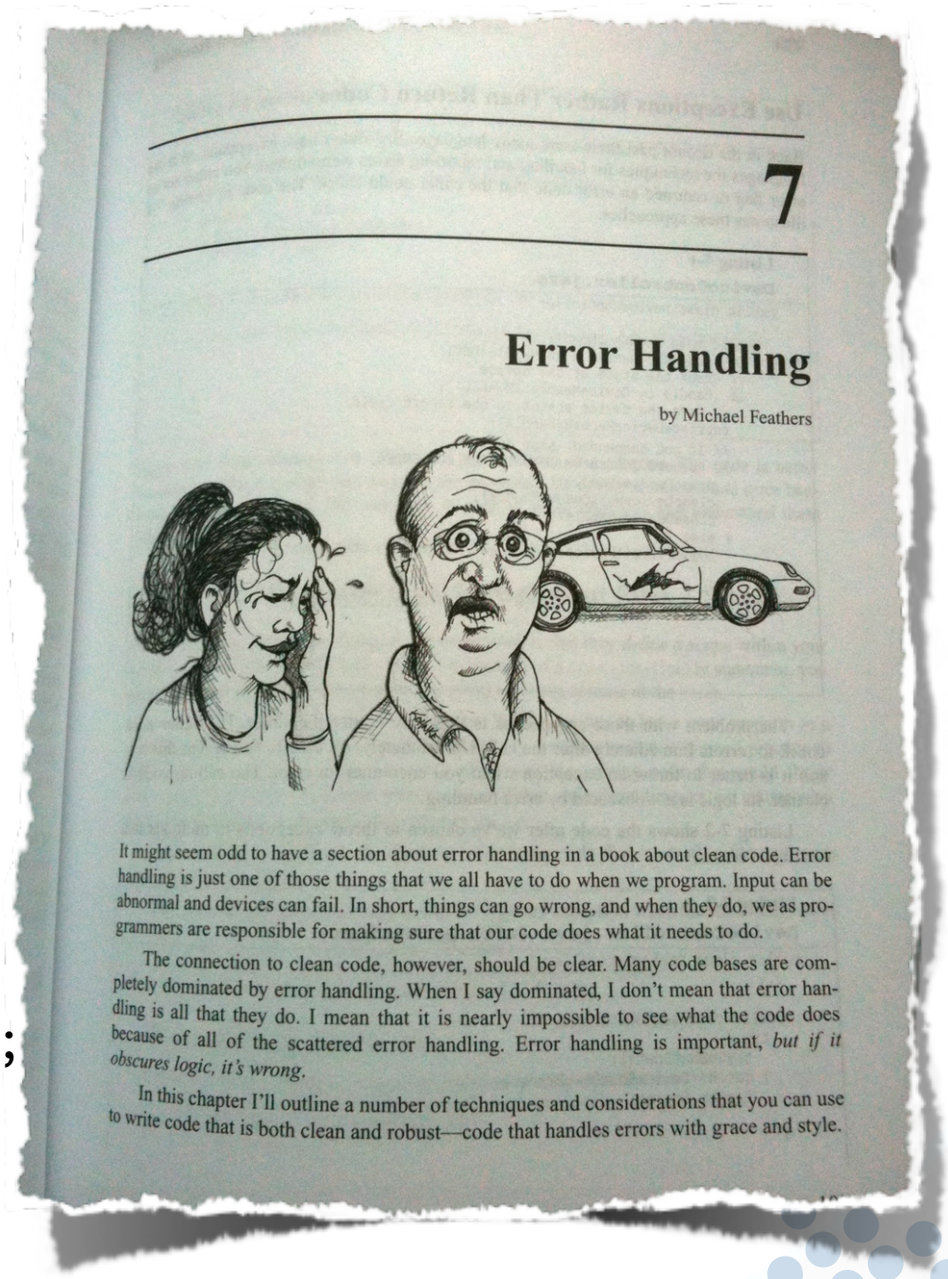
93

Error handling

- One test, one exception
- NEVER:

  `@Test(expected = Exception.class)`
  Use only CONCRETE, UNIQUE exception

- try / catch



7

Error Handling

by Michael Feathers

It might seem odd to have a section about error handling in a book about clean code. Error handling is just one of those things that we all have to do when we program. Input can be abnormal and devices can fail. In short, things can go wrong, and when they do, we as programmers are responsible for making sure that our code does what it needs to do.

The connection to clean code, however, should be clear. Many code bases are completely dominated by error handling. When I say dominated, I don't mean that error handling is all that they do. I mean that it is nearly impossible to see what the code does because of all of the scattered error handling. Error handling is important, *but if it obscures logic, it's wrong.*

In this chapter I'll outline a number of techniques and considerations that you can use to write code that is both clean and robust—code that handles errors with grace and style.

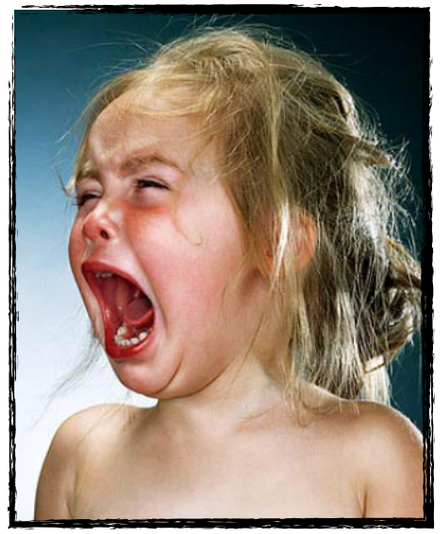
```
@Rule
public ExpectedException throwing =
                            ExpectedException.none();

...
...

// when
throwing.expect(ParseException.class);
parser.parse(text);
// then exception is thrown
```

Error handling

```
import static com.googlecode.catchexception.CatchException.*;
import static com.googlecode.catchexception.apis.CatchExceptionBdd.*;

// given: an empty list
List myList = new ArrayList();

// when: we try to get the first element of the list
when(myList).get(1);

// then: we expect an IndexOutOfBoundsException
then(caughtException())
        .isInstanceOf(IndexOutOfBoundsException.class)
        .hasMessage("Index: 1, Size: 0")
        .hasNoCause();
```

https://code.google.com/p/catch-exception/

Behaviour Driven Development



- define behaviours not tests
- behaviours constitute a functional spec of the application
- behaviours should be worked upon by business people together with developers
- focus on why the code should exist
- naming in code is the same as names used by the business people (ubiquitous language)

Defining behaviour

As a conference attendee **I want to** get a restration status after signing up for a conference **so that** I know if the registration went fine:

Given a conference „Joker”

When I try to register to it and the registration is successful

Then I get a confirmation message: „You are registered to Joker. An email with details has been sent to you.”

Given a conference „Joker”

When I try to register to it but there are no free places

Then I get a message: „Sorry. No free places left. Try the next year!”

Examples, not Tests

- BDD helps to think about objects from the perspective of their behaviours, so the code is more object-oriented
- examples help you create a „mental” model of the system
- test class shows examples of use of a functionality
- test code is an example of behaviour
- test code is also an example of use

BDD naming

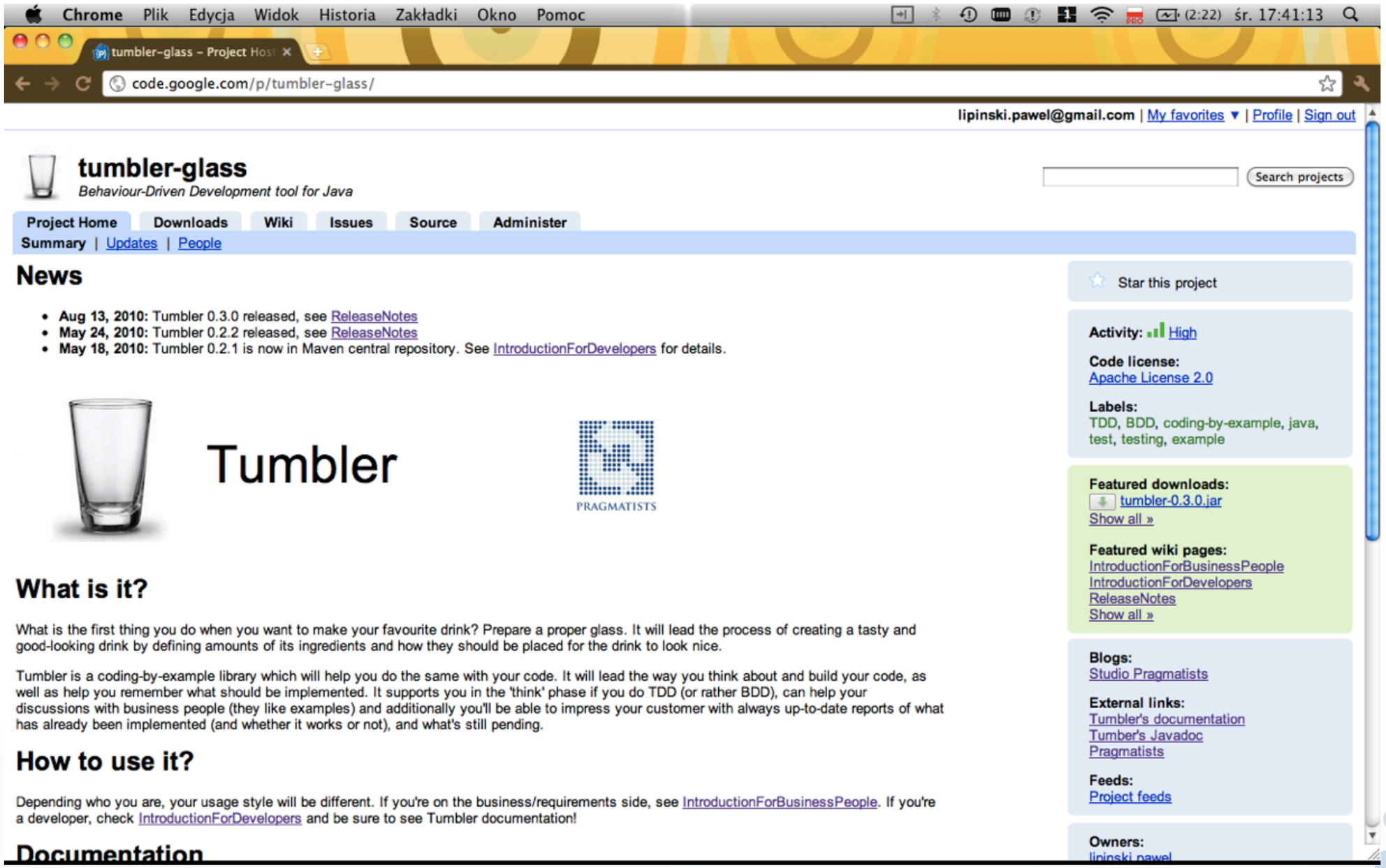
Story, Scenario

```
public class AddingBooksToLibraryTest {
    @Mock
    private BookRepository bookRepository;

    @Test
    public void shouldEnableAddingNewBooks() {
        // given
        Library library = new Library(bookRepository);
        Book book = new Book("Children from Bullerbyn");

        // when
        library.add(book);

        // then
        assertThat(library.size()).is(1);
        verify(bookRepository).save(book);
    }
}
```

BDD rules

- test names should be sentences
- simple constant template of the sentence helps you focus in the test on one thing
- understandable test name helps when the test fails
- test are examples - think about them not as a means for future verification, but as a documentation of behaviour

Tumbler

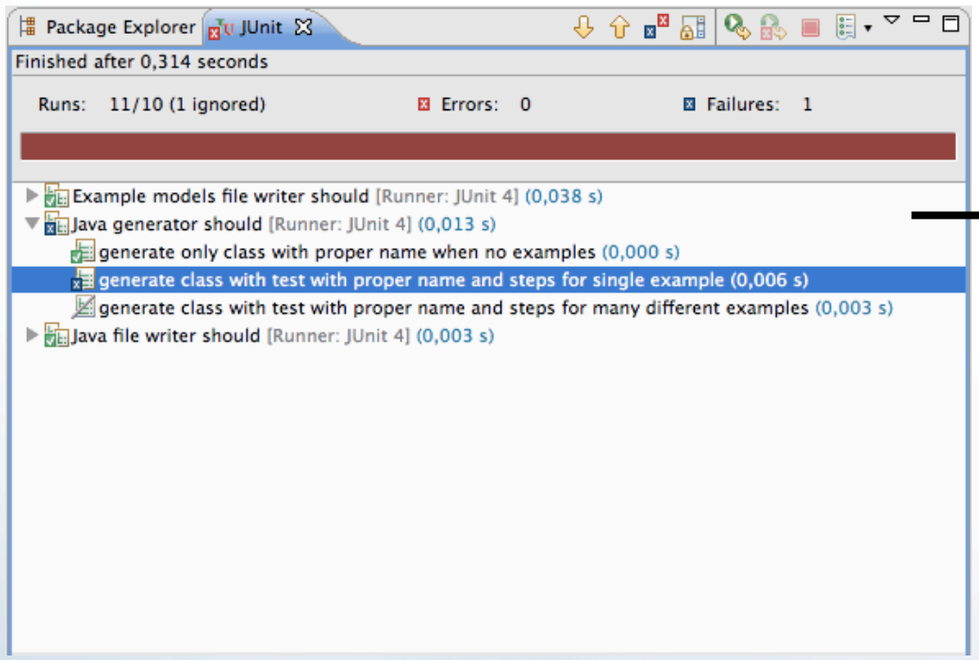


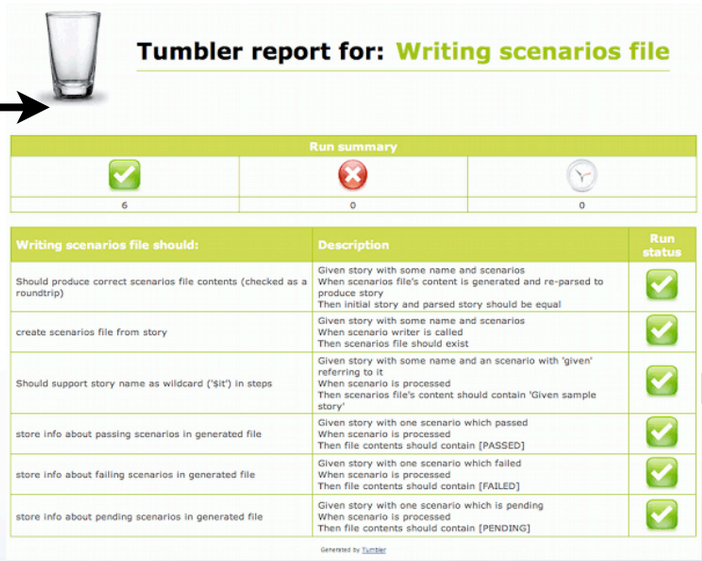
http://tumbler-glass.googlecode.com

```
Story: Library
    Scenario: lend an existing book from the library
        Given 'Children from Bullerbyn' book in the library
        When this book is borrowed from the library
        Then the library doesn't contain it anymore.

    Scenario: not lend a nonexisting book from the library
        Given empty library
        When we try to borrow 'Children from Bullerbyn' from the library
        Then the library doesn't let it to be borrowed.

    Scenario: accept back a book previously borrowed
        Given 'Children from Bullerbyn' has been borrowed from the library
        When this book is given back
        Then the library contains it.

    Scenario: not accept back a book which does not belong to the library
        Given 'Children from Bullerbyn' book has not been borrowed the library
        When this book is given back
        Then the library does not accept it.
```

```java
import org.junit.*;

@RunWith(TumblerRunner.class)
@Story("Library")
public class LibraryScenarios {
    public LibraryScenarios() {
        Narrative("As a library user, " +
                "In order to borrow a book, " +
                "I want librarian to give me that book, " +
                "So that I can take it home");
    }

    @Scenario(pending = true)
    public void shouldLendAnExistingBookFromTheLibrary() {
        Given("'Children from Bullerbyn' book in the library " +
                "and a pretty librarian.");

        When("this book is borrowed from the library " +
                "and the librarian is blinking at you");

        Then("the library doesn't contain it anymore " +
                "and the librarian wants to go out with you " +
                "but you're already married, so no way.");
    }

    @Scenario(pending = true)
    public void shouldNotLendANonexistingBookFromTheLibrary() {
        Given("empty library");

        When("we try to borrow 'Children from Bullerbyn' from the library");

        Then("the library doesn't let it to be borrowed.");
    }

    @Scenario(pending = true)
    public void shouldAcceptBackABookPreviouslyBorrowed() {
        Given("'Children from Bullerbyn' has been borrowed from the library");

        When("this book is given back");

        Then("the library contains it.");
    }
}
```

```java
@Scenario(pending = true)
public void shouldNotLendANonexistingBookFromTheLibrary() {
    Given("empty library");
    Library library = new Library();
    Book sampleBook = new Book("Children from Bullerbyn");

    When("we try to borrow 'Children from Bullerbyn' from the library");
    library.lend(sampleBook).to(reader);

    Then("the library doesn't let it to be borrowed.");
    assertThat(reader.books().size(), is(0));
}
```





Tumbler report for: Writing scenarios file

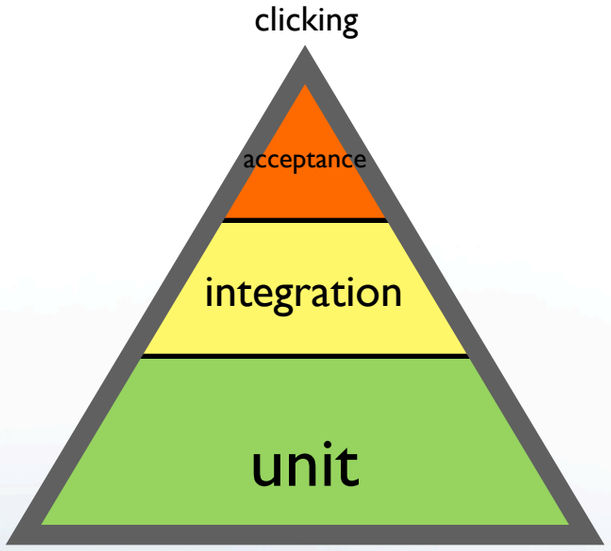
Run summary		
6	0	0

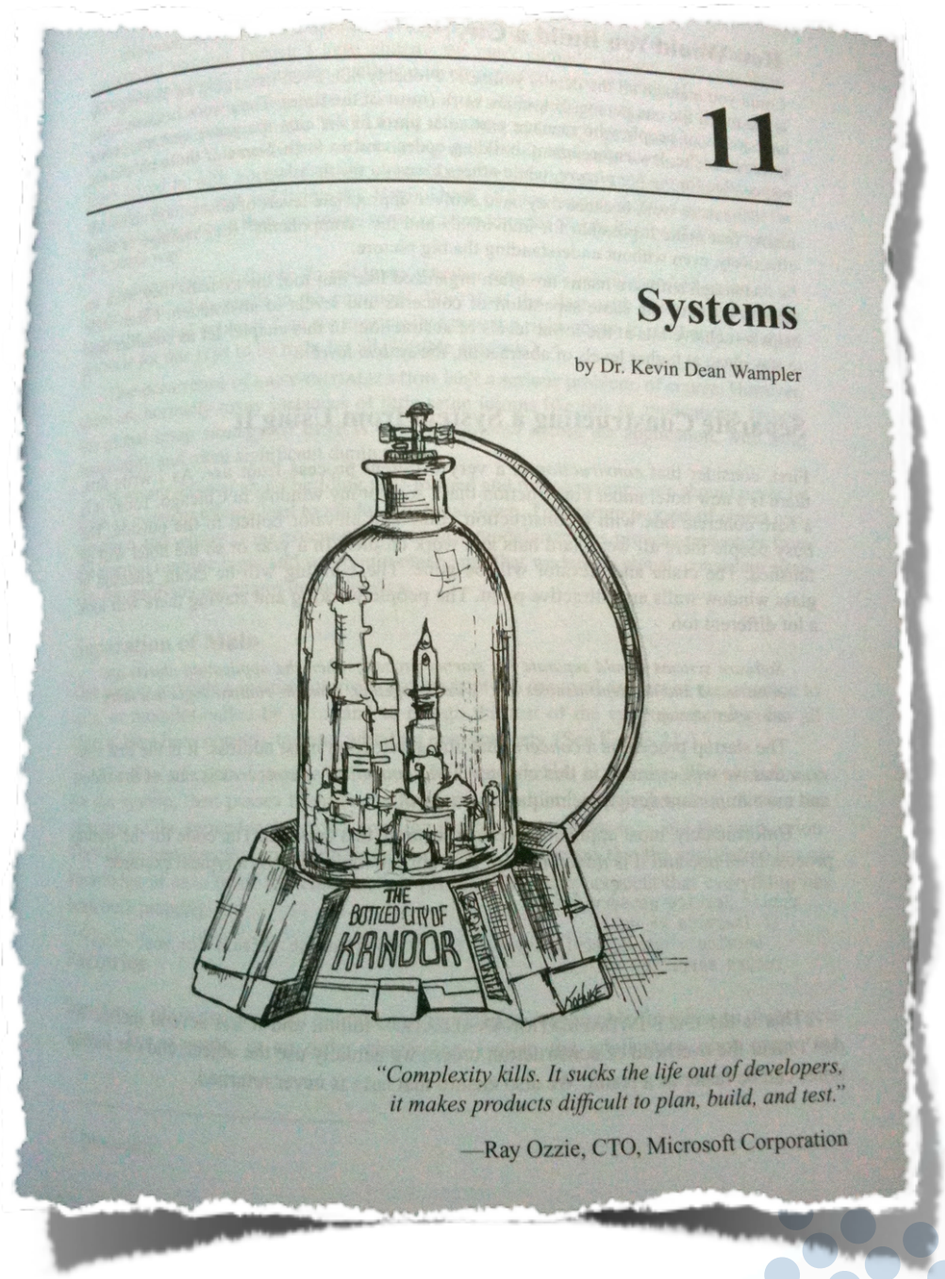
Writing scenarios file should:	Description	Run status
Should produce correct scenarios file contents (checked as a roundtrip)	Given story with some name and scenarios When scenarios file's content is generated and re-parsed to produce story Then initial story and parsed story should be equal	✓
create scenarios file from story	Given story with some name and scenarios When scenario writer is called Then scenarios file should exist	✓
Should support story name as wildcard ('$it') in steps	Given story with some name and an scenario with 'given' referring to it When scenario is processed Then scenarios file's content should contain 'Given sample story'	✓
store info about passing scenarios in generated file	Given story with one scenario which passed When scenario is processed Then file contents should contain [PASSED]	✓
store info about failing scenarios in generated file	Given story with one scenario which failed When scenario is processed Then file contents should contain [FAILED]	✓
store info about pending scenarios in generated file	Given story with one scenario which is pending When scenario is processed Then file contents should contain [PENDING]	✓

Generated by Tumbler

How to organize all these tests

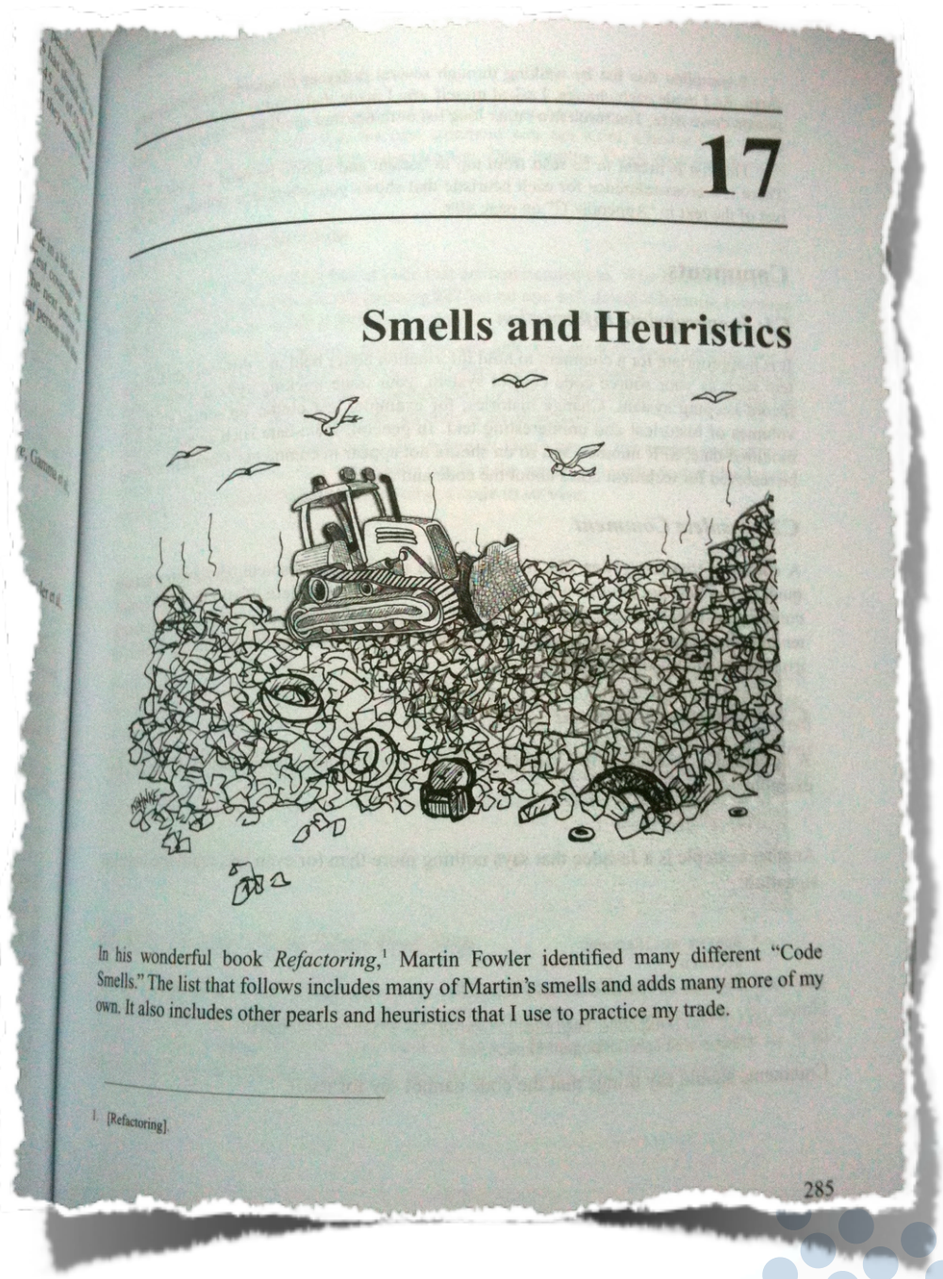
- Tests in the same packages as SUT vs. separately
- Tests named by the functionality vs. tests named by tested classes
- Tests' speed
- Test levels





Test smells

- Long setup - lack of factory method, or perhaps a problem with the structure of created objects?
- Long tests - perhaps you're testing more than one behaviour at a time?
- Many assertions - doesn't the tested method have too many responsibilities?
- Too many test methods in a test class - class under test has too many responsibilities?



17

Smells and Heuristics

In his wonderful book *Refactoring*,[1] Martin Fowler identified many different "Code Smells." The list that follows includes many of Martin's smells and adds many more of my own. It also includes other pearls and heuristics that I use to practice my trade.

1. [Refactoring].

285

Maintainable tests

- Reuse assertions, create „business” assertions
- Reuse object construction methods - don’t be sensitive to changes of constructors
- Reuse test setup
- Tests should not depend on environment and order of execution
- Avoid many assertions - when the first one fails, others are not executed (they could’ve give you a clue about possible reasons)
- Separate assertion from the action to increase readability (or better use given/when/then pattern)
- Use variables to pass and verify values in tests
- Remove tests ONLY when you remove a functionality or when tests’ responsibilities overlap

Pair programming

http://static.guim.co.uk/sys-images/Guardian/Pix/pictures/2011/10/25/13195652461 30/Russian-President-Dmitry--007.jpg

The biggest challenge for me personally was essentially mourning for the death of “Programmer Man”.

Programmer Man is how I think of myself when I’ve got my headphones in, speed metal blaring in my ears, and I’m coding like a motherfucker. My fingers can’t keep up with my brain. I’m In The Zone.

For most of my career, this image is what I’ve considered to be the zenith. When I come home and was in Programmer Man Mode most of the day, I feel like I’ve had a good day.

Pair Programming undeniably killed Programmer Man. This was a tough adjustment, since I’ve considered that mode to be my favorite for so long. I now see, however, that **Programmer Man was, without me knowing it, Technical Debt Man.**

http://www.nomachetejuggling.com/2009/02/21/i-love-pair-programming/

> Don’t be afraid of pair-programming - you’re not as good as you think, but you’re not as bad as you fear.

Ron Jeffries

Ok, but how to start doing it?

Comfortable position

Do harder bits in pairs first

Get a shower.

Do it the right way - one person codes,
the other gets the bigger picture.

Switch pairs.

And how to make it stick?

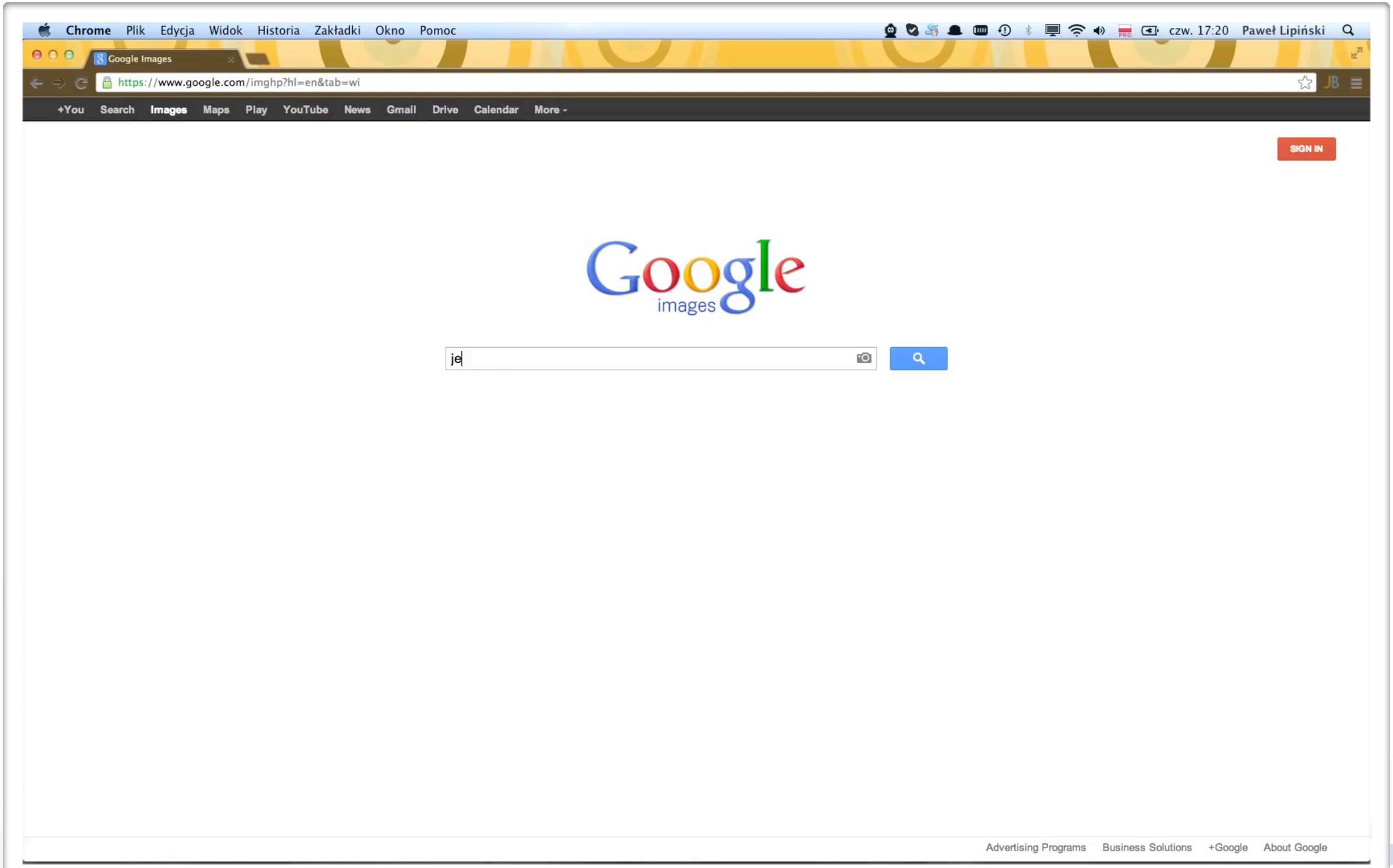
2 keyboards

2 mice

Check which setting fits you best.

http://www.nomachetejuggling.com/2011/08/25/mechanics-of-good-pairing/

Initially everybody MUST pair. Make it optional once you know it well.


Chrome Plik Edycja Widok Historia Zakładki Okno Pomoc
czw. 17:20 Paweł Lipiński
Google Images
https://www.google.com/imghp?hl=en&tab=wi
+You Search Images Maps Play YouTube News Gmail Drive Calendar More
SIGN IN
Google
images
je
Advertising Programs Business Solutions +Google About Google

Ok, but how to start doing it?

Check-in often... clean builds.

Never leave the office if the build is broken.

Don’t comment out failing tests...

Jenkins

cruise-control

gump

Team Foundation Server

TeamCity

Continuum

Hudson

Bamboo

Time-box fixing build revert if needed.

You’re responsible if your build broke something.

And how to make it stick?

Optimize your building process.

Sonar

Lots of tests.

Keep your tests fast.

Integration tests.

End-to-end tests.

Notifications which piss you off.

Fancy info radiator.

Thank you!

pawel.lipinski@pragmatists.pl

PRAGMATISTS

All pictures were used exclusively to set the presentation context and advertise the original book.