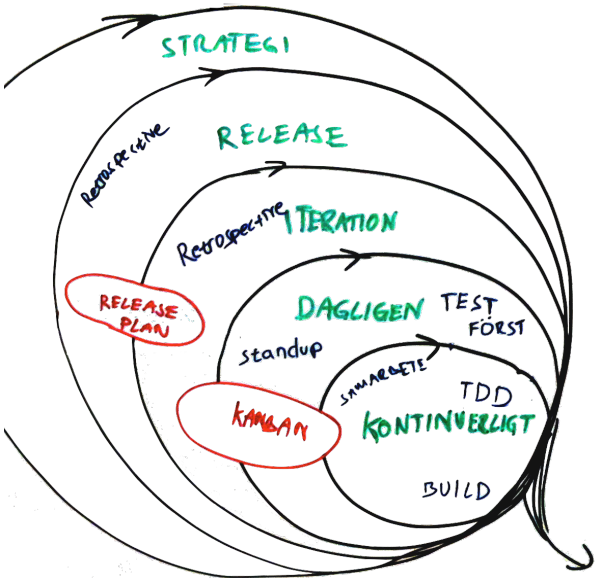
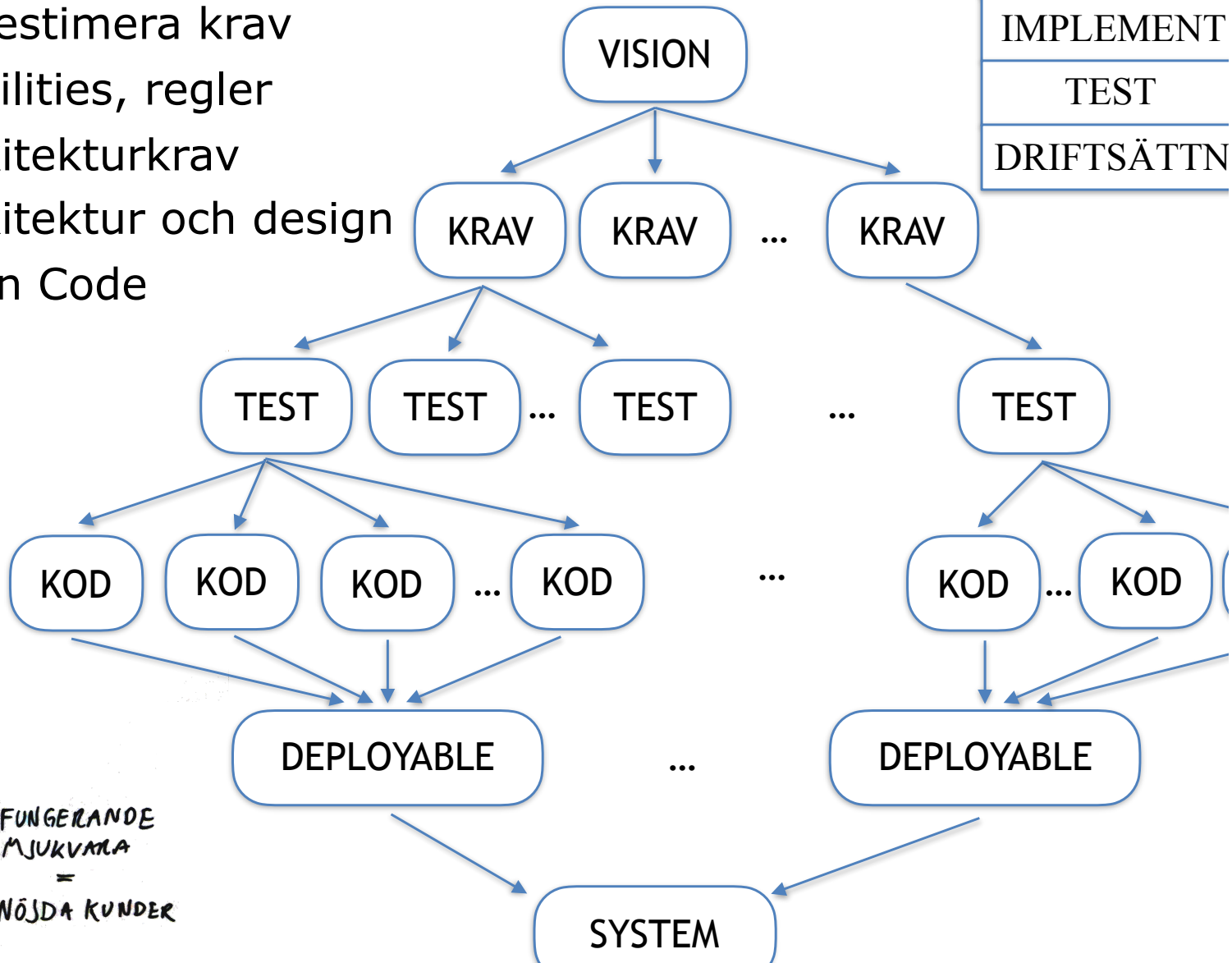
Den Röda Tråden

- Vi kan välja utvecklingsmodell
- Vi kan hantera risk och vet varför visionen behövs
- Vi kan skriva och estimera krav
  - User stories, -ilities, regler
- Vi kan ta fram arkitekturkrav
- Vi kan ta fram arkitektur och design
- Vi kan skriva Clean Code



FUNGERANDE MJUKVARA = NÖJDA KUNDER



Implementation

- Design Patterns
- Software Craftmanship
- Clean code
- Teststrategier
- **TDD**
- (Vad är en bra implementation?)

Vad betyder TDD?

Vad betyder TDD?

Test Driven Development?

Test Driven Design?

Test Driven Documentation?

Test Driven

Driven av test → test först!

Test Driven Development

google define: development

“act of improving by expanding or enlarging or refining”

Test Driven Design

google define: design

“the act of working out the form of something”

Test Driven Documentation

google define: design

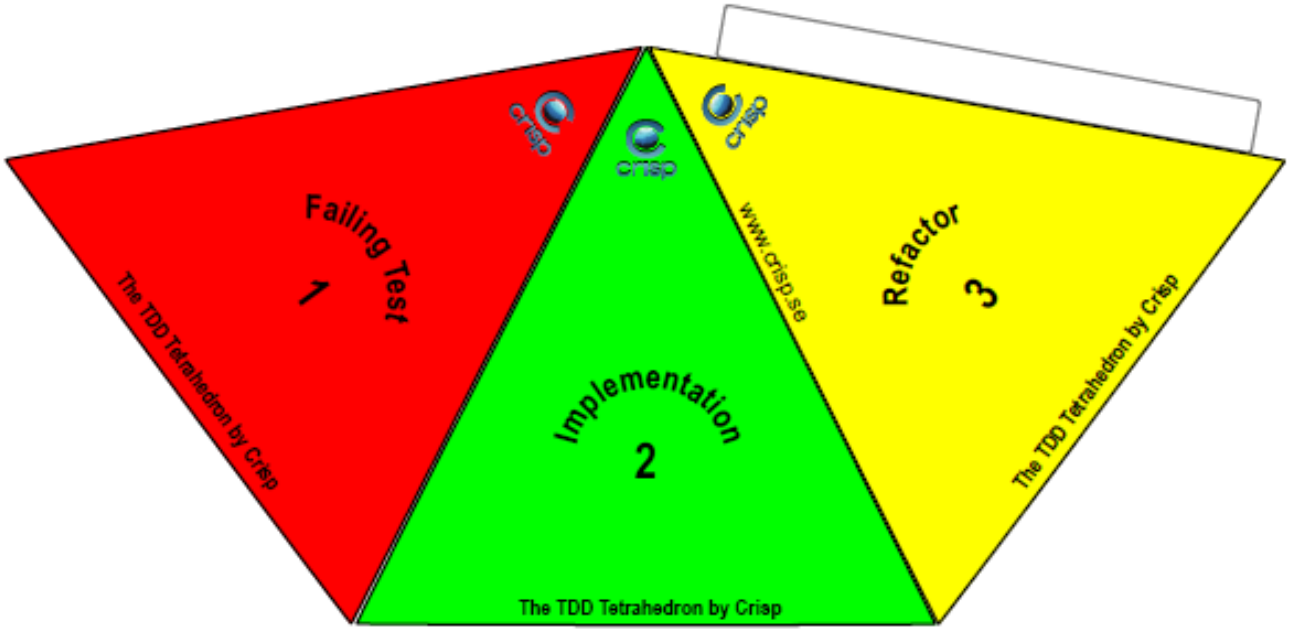
“manuals describing the operation and use of programs”

Uncle Bob:

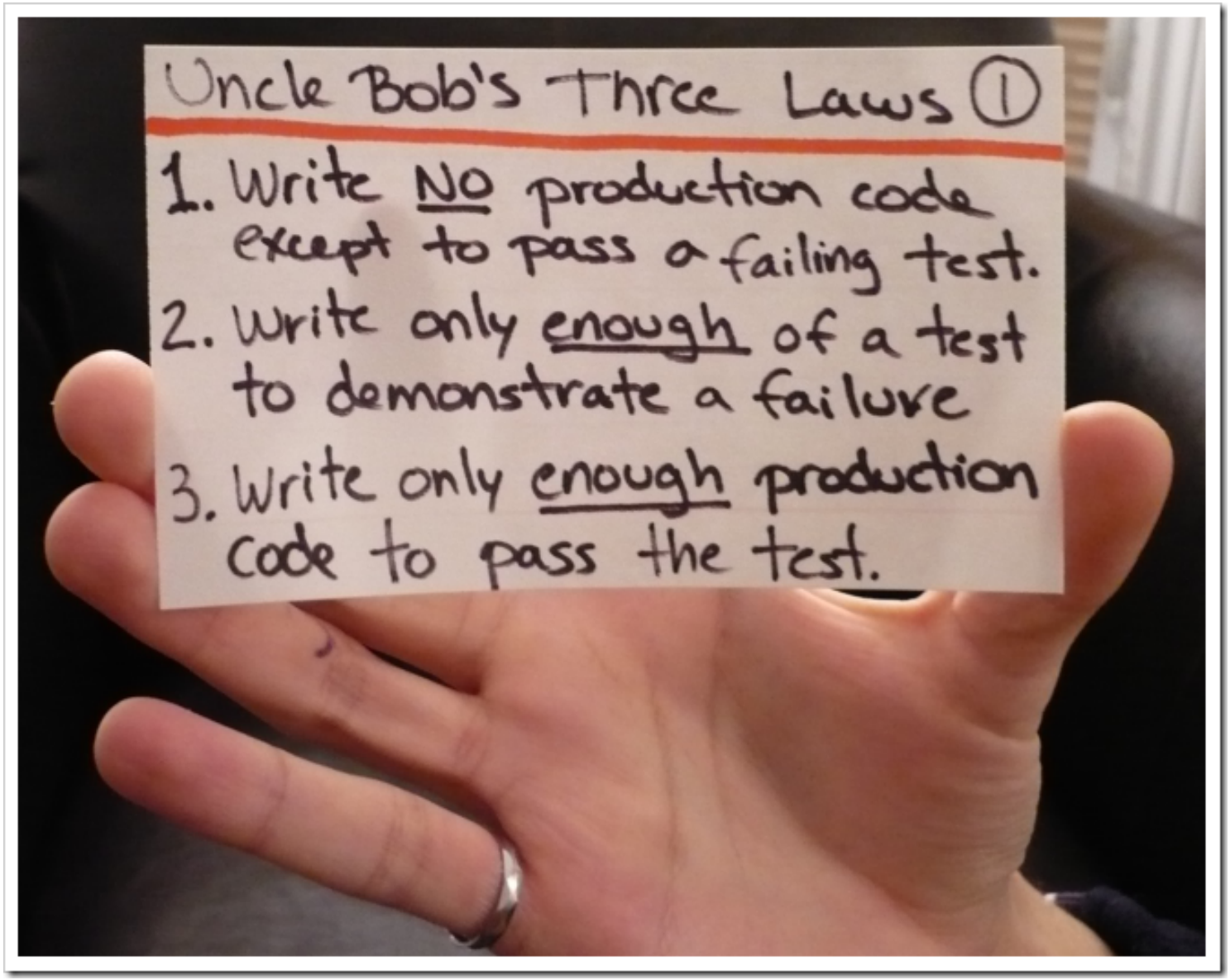
“My documentation is formal, it compiles! How about your documentation?”

TDD - hur gör man?

TDD Tetrahedon



- http://blog.crisp.se/perlundholm/2010/03/16/1268773204083.html



http://blog.briandicroce.com/2008/03/14/three-index-cards-to-easilyremember-the-essence-of-test-driven-development/
http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd

Live Coding: WordWrap

detta är en lite längre mening

→(10)

detta är en
lite längre
mening

Preferences

type filter text

- Help
- Install/Update
- Java
  - Appearance
  - Build Path
  - Code Coverage
  - Code Style
  - Compiler
  - Debug
  - Editor
    - Content Assist
    - Folding
    - Hovers
    - Mark Occurrences
    - Save Actions
    - Syntax Coloring
    - Templates
    - Typing
  - Installed JREs
  - JDepend
  - JUnit
  - Properties Files Editor
- Java EE
- Java Persistence
- JavaScript
- JSON
- Maven
- Mylyn
- Oomph
- Plug-in Development
- Remote Systems

Templates

Create, edit or remove templates:

Name	Context	Description	Auto Insert
public_stat...	Java type members	public static method	
runnable	Java	runnable	
SashForm	SWT statements	new SashForm	
Scale	SWT statements	new Scale	
ScrolledC...	SWT statements	new ScrolledComp...	
Shell	SWT statements	new Shell	
should	Java	test method (Junit 4)	on
Spinner	SWT statements	new Spinner	
static_final	Java type members	static final field	
StyledText	SWT statements	new StyledText	
StyleRange	SWT statements	new StyleRange fo...	
switch	Java statements	switch case statem...	
synchroniz...	Java statements	synchronized block	
syserr	Java statements	print to standard e...	on
sysout	Java statements	print to standard out	on
systrace	Java statements	print current meth...	on

New...

Edit...

Remove

Restore Removed

Revert to Default

Import...

Export...

Preview:

```
@${testType:newType(org.junit.Test)}
public void ${should}() {
}
```

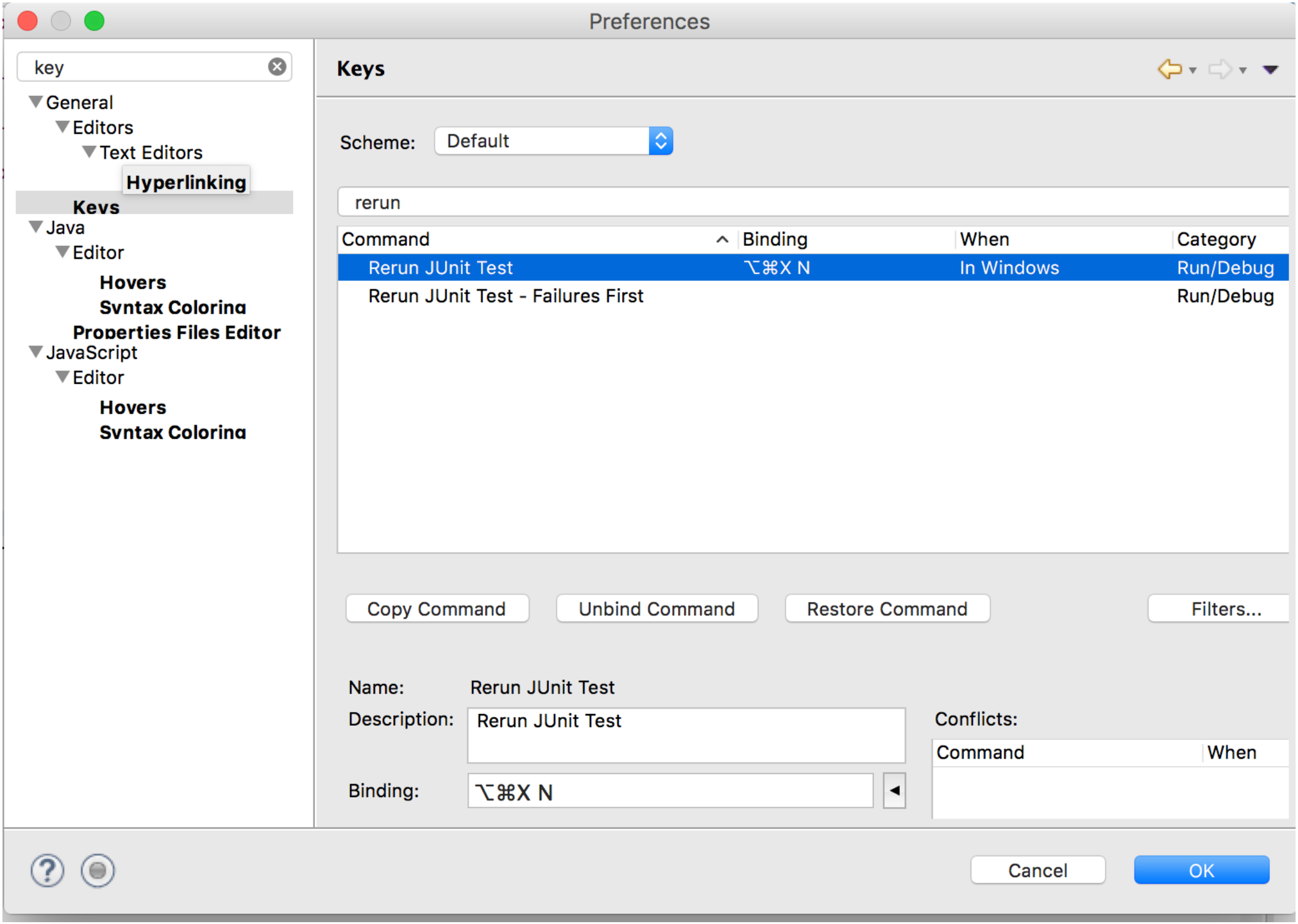
Use code formatter

Restore Defaults

Apply

Cancel

OK


Preferences
key
General
Editors
Text Editors
Hyperlinking
Keys
Java
Editor
Hovers
Syntax Coloring
Properties Files Editor
JavaScript
Editor
Hovers
Syntax Coloring
Keys
Scheme:
Default
rerun
Command
Binding
When
Category
Rerun JUnit Test
⌥⌘X N
In Windows
Run/Debug
Rerun JUnit Test - Failures First
Run/Debug
Copy Command
Unbind Command
Restore Command
Filters...
Name:
Rerun JUnit Test
Description:
Rerun JUnit Test
Binding:
⌥⌘X N
Conflicts:
Command
When
Cancel
OK

TDD – varför?

Design och Kvalitet

- Scope – Definition of Done
- Safe refactoring
- Coding by Intention
- Incremental Design
- Loose coupling – easy to test
- High cohesion – easy to test
- …

TDD - kodkvalitet

Hög täckningsgrad →
Trygg refaktorering →
Frihet att refaktorera kod →
Modifierbarhet →

- Yrkesstolthet
  - Jag vill inte leverera undermålig kod
  - Jag vill ha frihet att skapa ”clean code”

Test first

- Design
  - Safe refactoring
  - Coding by Intention
- Dokumentation
- Kvalitet
- Regressionstester
- Continuous Integration
- Continuous Delivery

Test after

- Design
  - ~Safe refactoring
  - ~~Coding by Intention~~
- Dokumentation
- ~~~Kvalitet
- ~Regressionstester
- ~Continuous Integration
- ~Continuous Delivery

No tests?

- Design?
  - ~~Safe refactoring~~
  - ~~Coding by Intention~~
- ~~Dokumentation~~
- Kvalitet?
- ~~Regressionstester~~
- Continuous Integration?
- Continuous Delivery?

TDD och kvalitet

- Studier visar tydligt på ökad kvalitet
  - Pre-release defect density 40-90%
  - 2.6-4.2 times lower number of defects
  - Passing 18% more blackbox tests
  - Etc

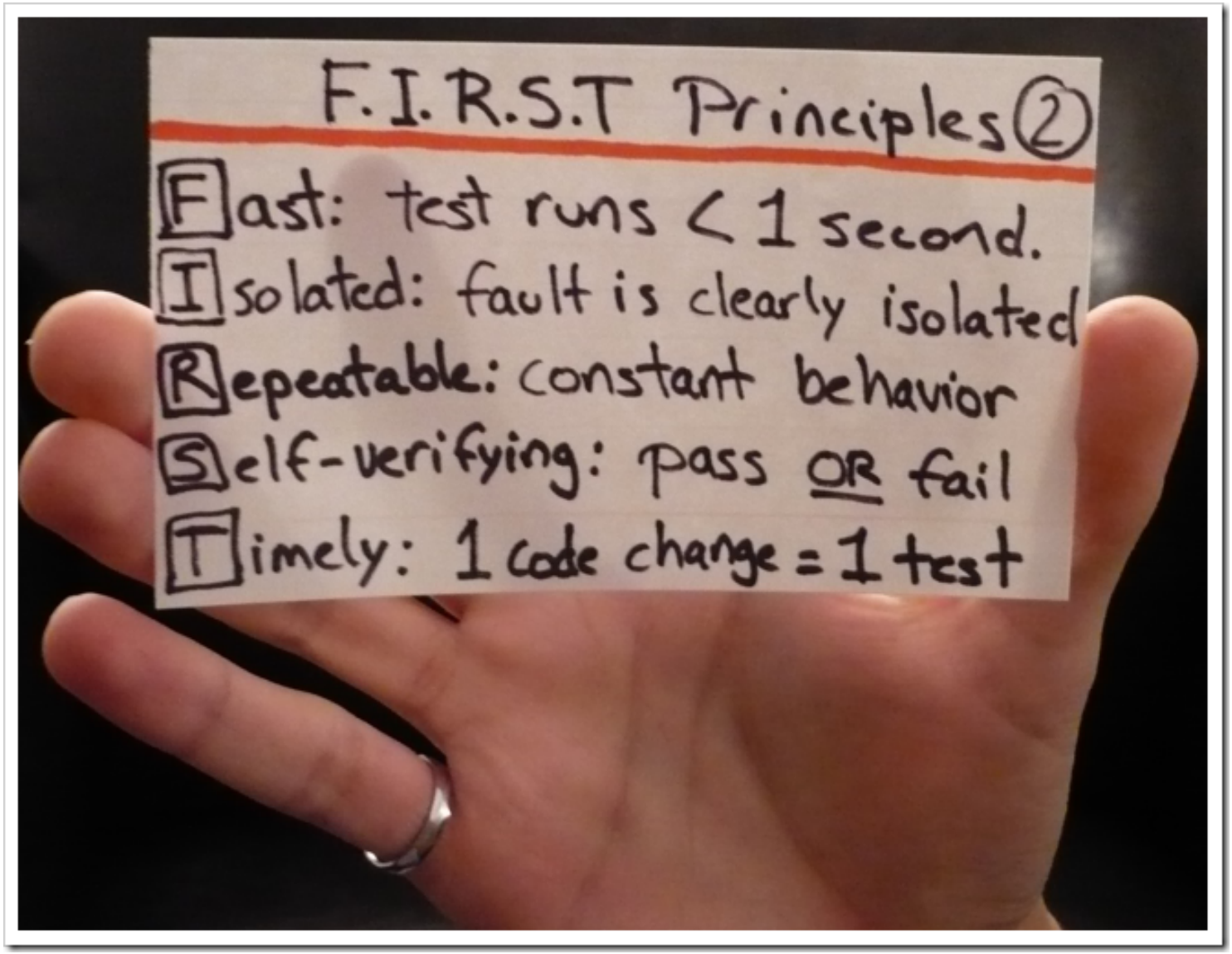
http://biblio.gdinwiddie.com/biblio/StudiesOfTestDrivenDevelopment

TDD och tidsåtgång

- Studier visar inte tydligt på minskad tidsåtgång
  - 15-35% increase in initial development time
  - No increase
  - etc
- Men vad händer sen?

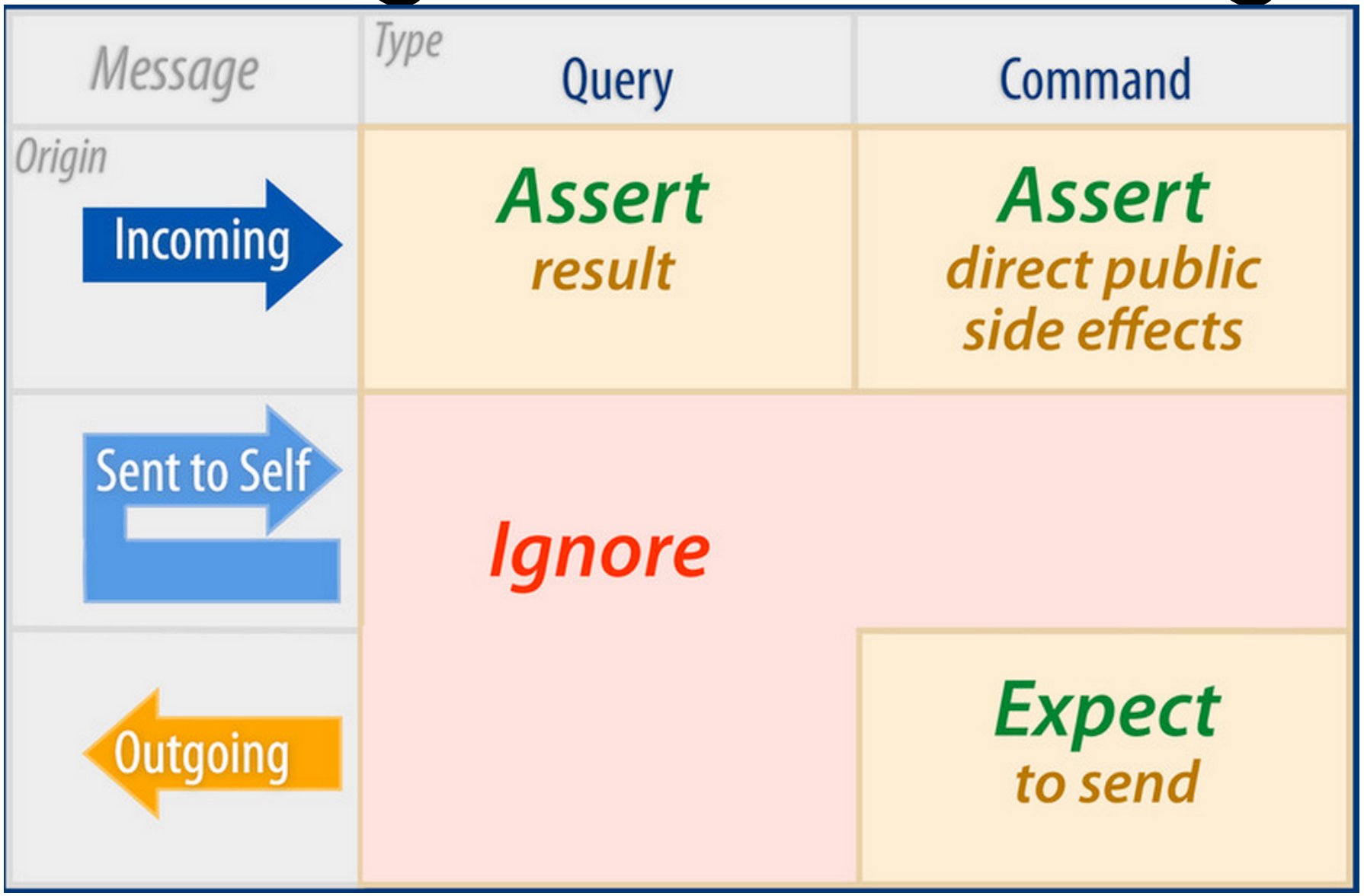
http://biblio.gdinwiddie.com/biblio/StudiesOfTestDrivenDevelopment

Vad är ett bra test?



http://blog.briandicroce.com/2008/03/14/three-index-cards-to-easilyremember-the-essence-of-test-driven-development/
http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd

The Magic Tricks of Testing



https://www.youtube.com/watch?v=URSWYvyc42M

Magic Tricks

- Thorough
- Stable
- Fast
- Few

Guidelines

Keep tests clean!

One Assert per Test (at least minimized)

Single Concept per Test

Uncle Bob on privates

#1 Don’t test private or protected methods.

#2 Testing trumps privacy

Make private methods package protected

Or extract into own class

#3 Testing privates implies a design error

Kata

- Sätt rätt rörelsemönster i ryggmärgen

Shu-Ha-Ri

- Shu
  - Focus on practice – start forming habit.
- Ha
  - Learn underlying principles and theory behind technique.
- Ri
  - Decide for yourself, adapt to your particular context

http://martinfowler.com/bliki/ShuHaRi.htm

Alternativ: http://en.wikipedia.org/wiki/

Kata 1 i em

- String Calculator Kata
  - http://osherove.com/tdd-kata-1/
- Ping Pong TDD
  - http://coderetreat.org/facilitating/activities/ping-pong
- PreReqs:
  - JUnit
  - https://github.com/jonananas/tdd-workshop

Pair Programming

- Driver-Navigator
- Ping Pong TDD

  http://coderetreat.org/facilitating/activities/ping-pong
- Promiscuous Pairing and Beginner's Mind

  http://csis.pace.edu/~grossman/dcs/XR4-PromiscuousPairing.pdf

Pairing – Varför?

Better design
Better decisions
YAGNI

Quality
Learning
Satisfaction
Economics
    +15% time
    -15% defects

String Calculator

The following is a TDD Kata- an exercise in coding, refactoring and test-first, that you should apply daily for at least 15 minutes (I do 30).

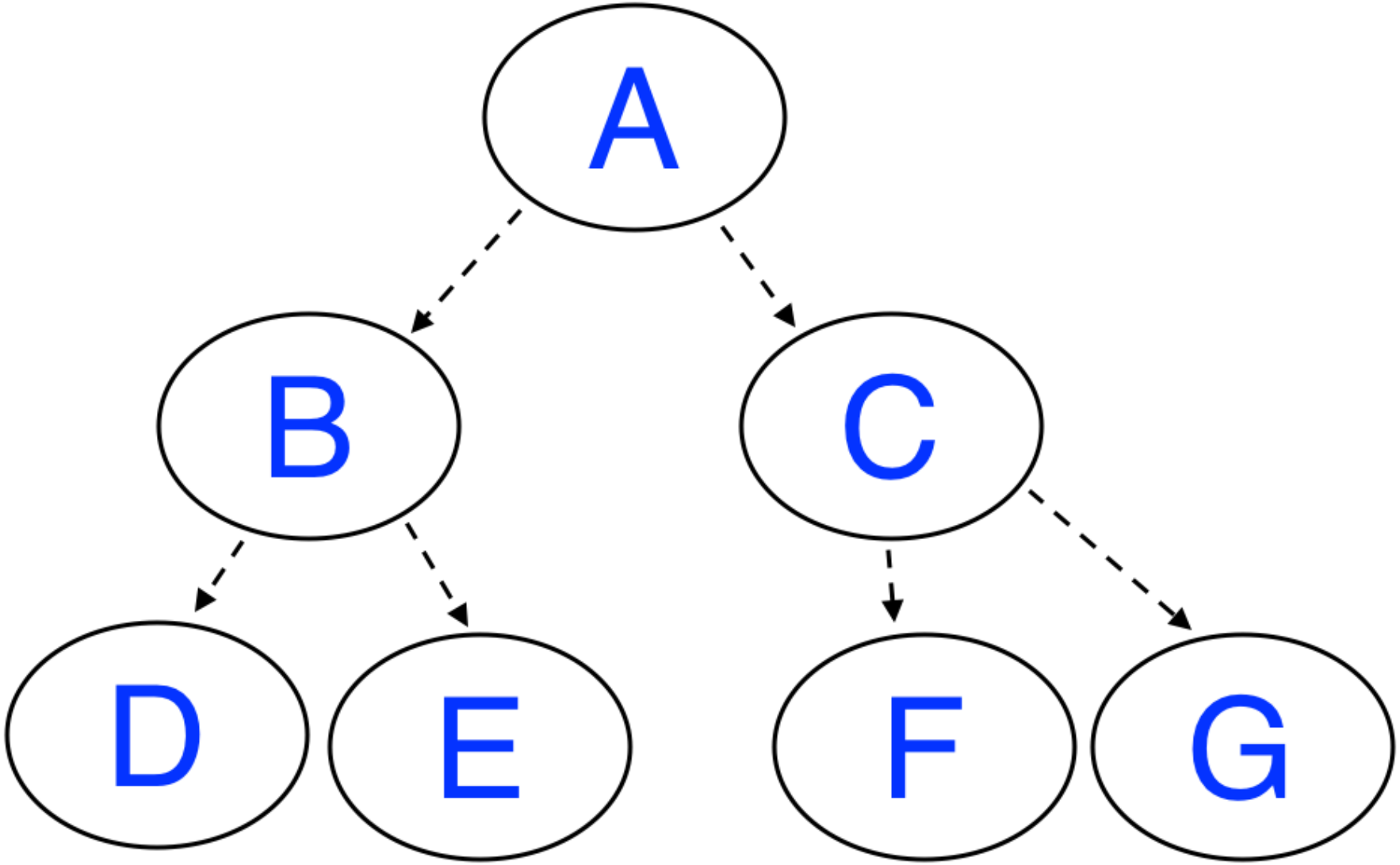
Before you start:

- Try not to read ahead.
- Do one task at a time. The trick is to learn to work incrementally.
- Make sure you only test for correct inputs. there is no need to test for invalid inputs for this kata

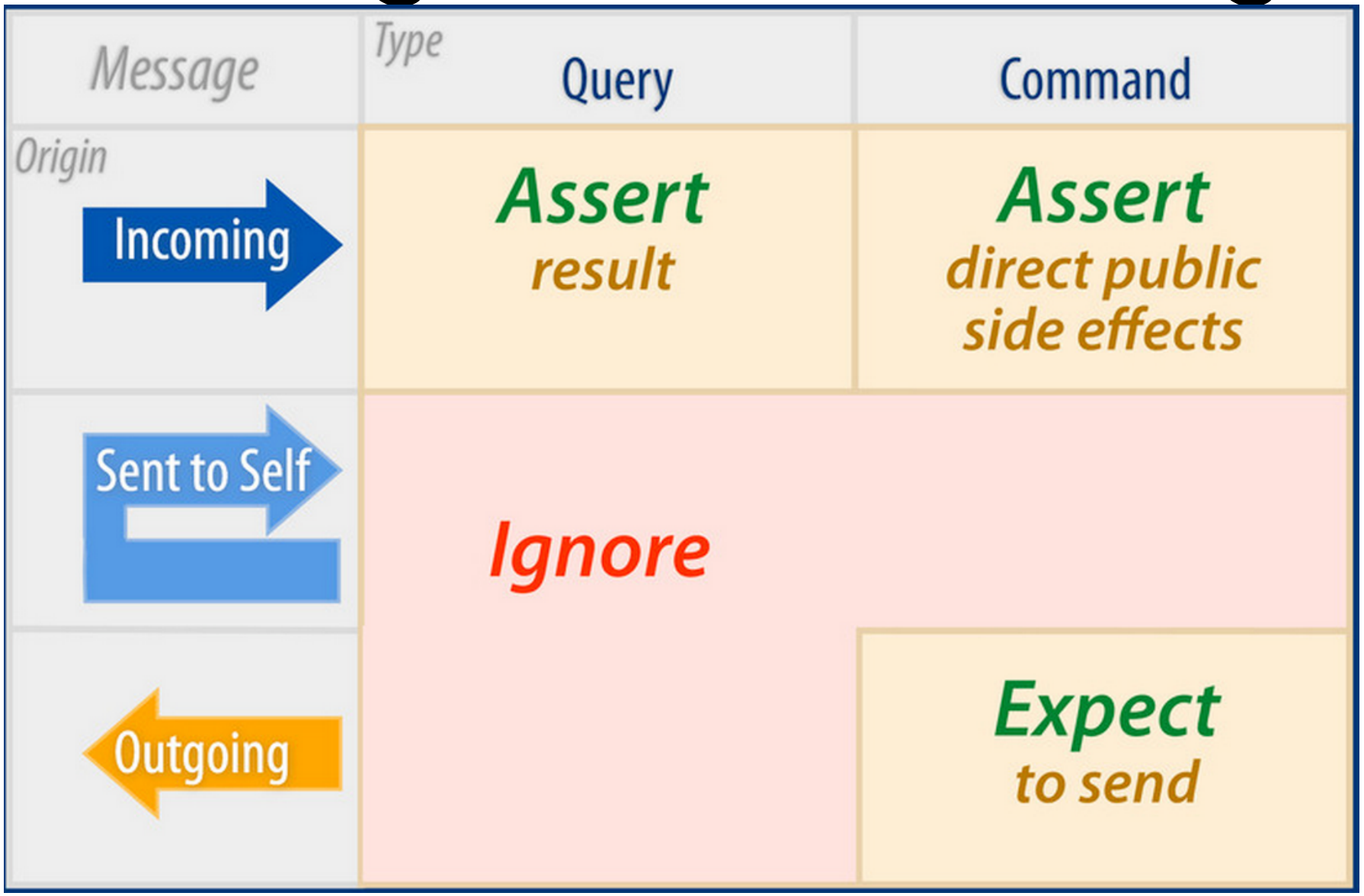
String Calculator

1. Create a simple String calculator with a method int Add(string numbers)
    1. The method can take 0, 1 or 2 numbers, and will return their sum (for an empty string it will return 0) for example “” or “1” or “1,2”
    2. Start with the simplest test case of an empty string and move to 1 and two numbers
    3. Remember to solve things as simply as possible so that you force yourself to write tests you did not think about
    4. Remember to refactor after each passing test
2. Allow the Add method to handle an unknown amount of numbers
3. Allow the Add method to handle new lines between numbers (instead of commas).
    1. the following input is ok: “1\n2,3” (will equal 6)
    2. the following input is NOT ok: “1,\n” (not need to prove it - just clarifying)
4. Support different delimiters
    1. to change a delimiter, the beginning of the string will contain a separate line that looks like this: “//[delimiter]\n[numbers…]” for example “//;\n1;2” should return three where the default delimiter is ‘;’ .
    2. the first line is optional. all existing scenarios should still be supported
5. Calling Add with a negative number will throw an exception “negatives not allowed” - and the negative that was passed.if there are multiple negatives, show all of them in the exception message

Dependencies?



The Magic Tricks of Testing



https://www.youtube.com/watch?v=URSWYvyc42M

Test doubles

- Dummy – Gör ingenting, returnerar null/0. Skickas ofta bara runt.
- Stub – Dummy som returnerar korrekta värden. State verification.
- Spy – Stub som spelar in anrop.
- Fake – En enkel simulator, t.ex. RepositoryInMem.
- Mock – Vet vilka metoder som ska anropas - expectations. Behaviour verification.

Test doubles

- F - Långsamma tester
- I - Ej isolerade
- R – Sköra

Mockito

```
@Test
public void basicFlow() throws Exception {
    // Build
    String collaborator = mock(String.class);
    when(collaborator.length()).thenReturn(3);

    // operate
    int length = collaborator.length();

    // check
    assertThat(length).isEqualTo(3);
    verify(collaborator).length();
}
```

Test doubles

- Exempel med Mockito

- OrderServiceTest

Test doubles

- Driv fram interface – implementera senare

Mockito - captor

```
@Test
public void usingCaptor() throws Exception {
    ArgumentCaptor<String> captor = ArgumentCaptor.forClass(String.class);

    log.info("a very complicated message");

    verify(log).info(captor.capture());
    assertThat(captor.getValue()).matches(".+ message");
}
```

Mockito - DI

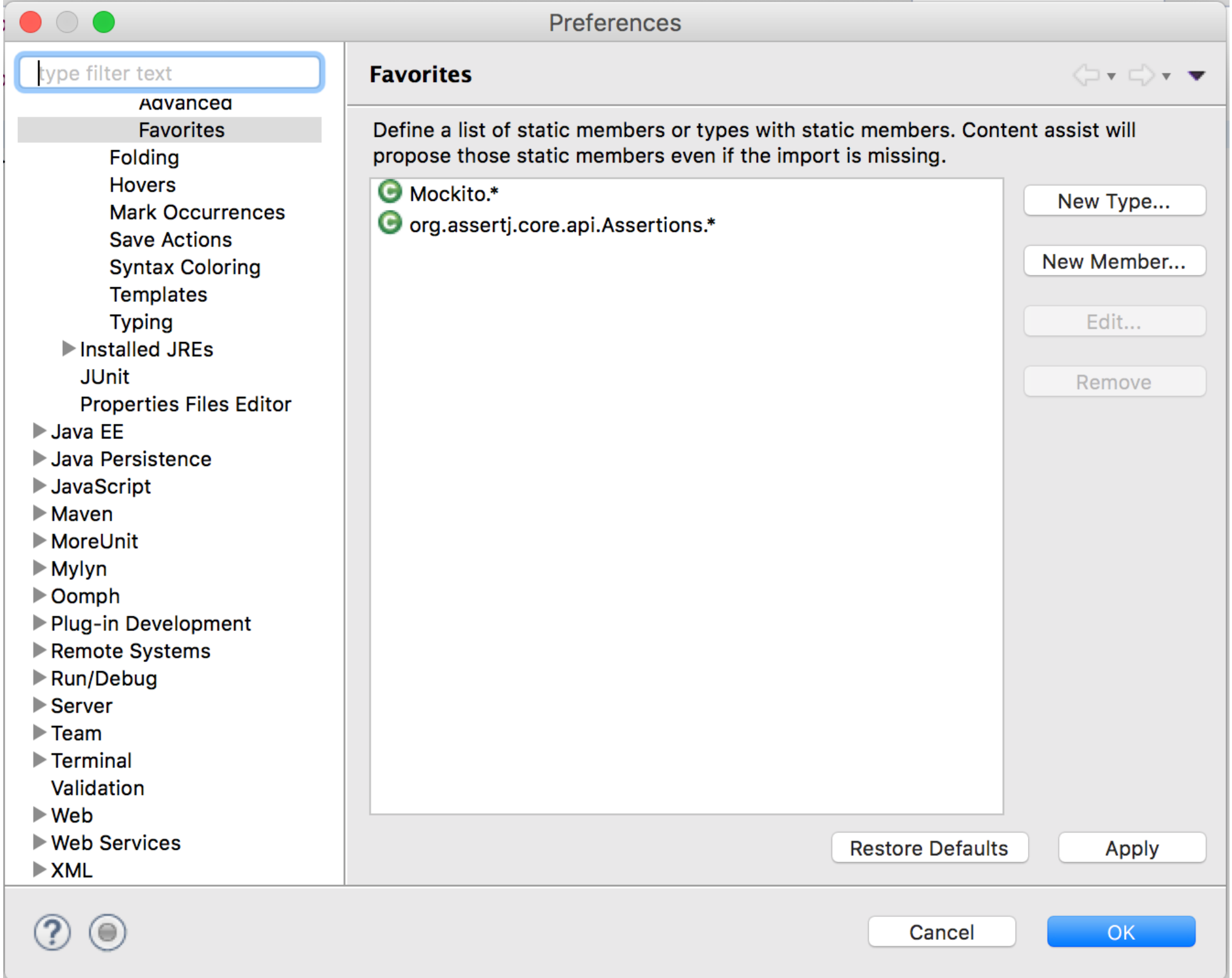
```java
@RunWith(MockitoJUnitRunner.class)
public class MockitoBeyondBasics {

    @Mock
    Logger log;

    @Captor
    ArgumentCaptor<String> captor;

    @Test
    public void usingInjectedCaptor() throws Exception {
        log.info("a very complicated message");

        verify(log).info(captor.capture());
        assertThat(captor.getValue()).matches(".+ message");
    }
```


Preferences
type filter text
Advanced
Favorites
Folding
Hovers
Mark Occurrences
Save Actions
Syntax Coloring
Templates
Typing
Installed JREs
JUnit
Properties Files Editor
Java EE
Java Persistence
JavaScript
Maven
MoreUnit
Mylyn
Oomph
Plug-in Development
Remote Systems
Run/Debug
Server
Team
Terminal
Validation
Web
Web Services
XML
Favorites
Define a list of static members or types with static members. Content assist will propose those static members even if the import is missing.
Mockito.*
org.assertj.core.api.Assertions.*
New Type...
New Member...
Edit...
Remove
Restore Defaults
Apply
Cancel
OK

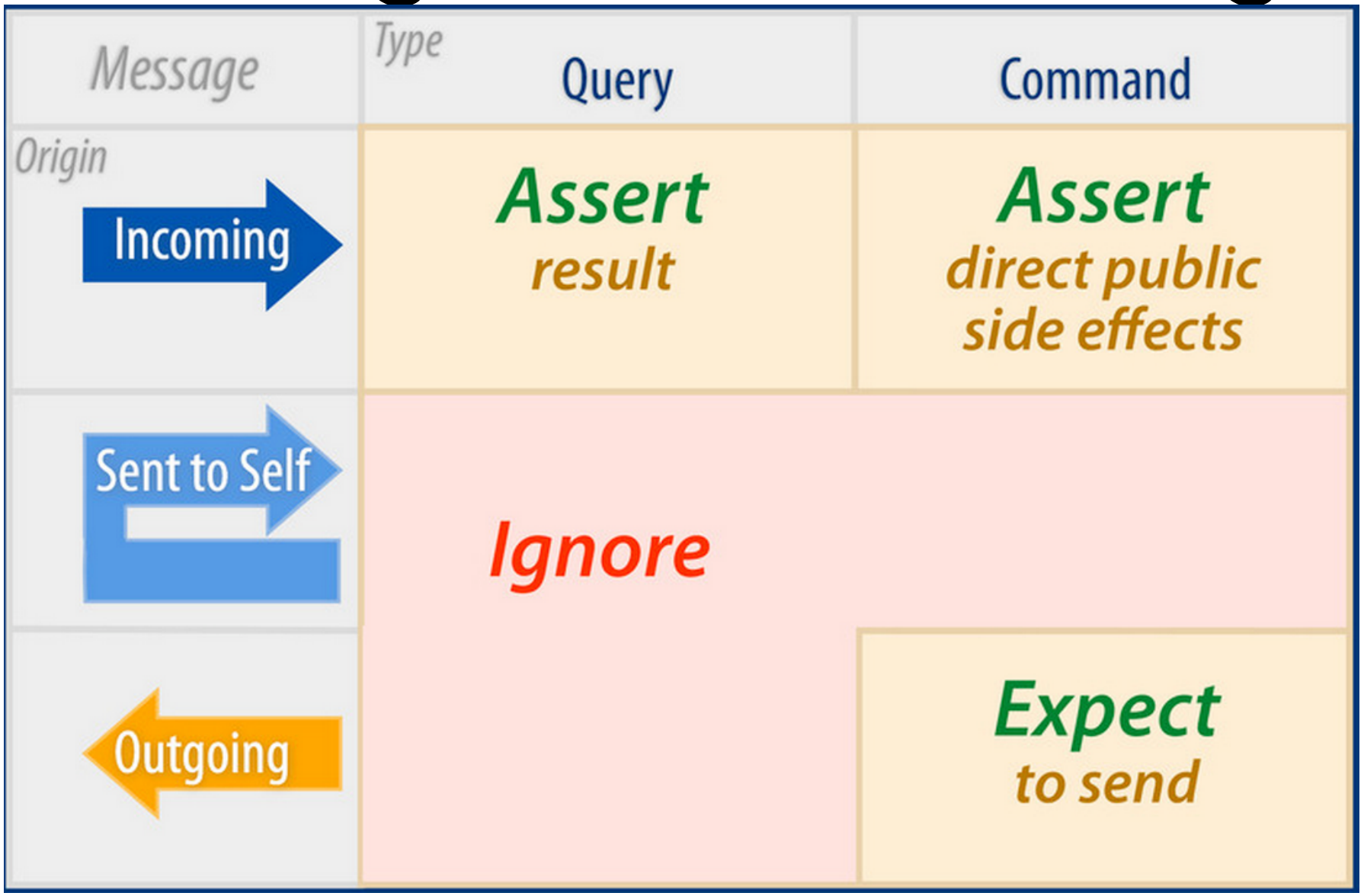
Kata: Log-interface till StringCalculator

- Utgå från StringCalculator.

1. Logga resultatet med log.info varje gång add() anropas. Test först!
2. Om log.info slänger exception så ska det propagera ut från add().

TDD – varför gör du det inte?

The Magic Tricks of Testing



https://www.youtube.com/watch?v=URSWYvyc42M

TDD – varför inte?

Press att leverera snabbt

Tar lång tid att skriva

Tar lång tid att köra

Går sönder hela tiden - sköra!

http://butunclebob.com/ArticleS.UncleBob.GreenWristBand

Sköra tester

- Interface sensitivity
  - UI
- Data sensitivity
  - Live data
- Context sensitivity
  - Time/Date/Memory/Network/OS/Hardware
- Overspecification
  - Något vi inte avsett att testa förändras

Underhåll testerna

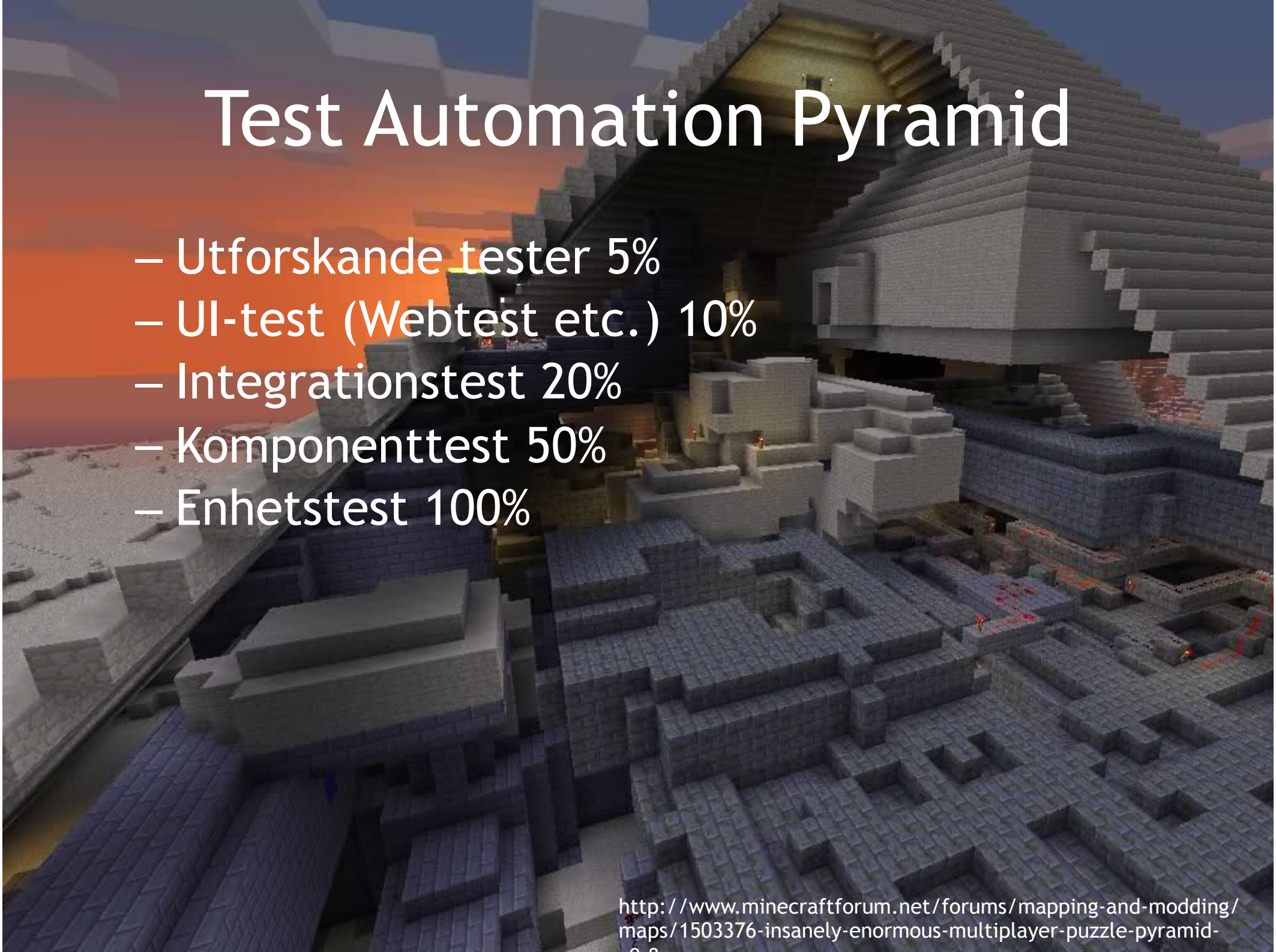
- Tester slås av när de är i vägen
  - Om de tar för lång tid
  - Om de är sköra
  - Om man inte kan lita på dem
- När tester slås av lämnas systemet oskyddat
- Exempel: UI-tester med live-data

Underhåll testerna

- Lösning
  - Fallerande tester är ”stop the line”-händelser
  - Refaktorera sköra tester
  - Refaktorera långsamma tester

Test Automation Pyramid

- Utforskande tester 5%
- UI-test (Webtest etc.) 10%
- Integrationstest 20%
- Komponenttest 50%
- Enhetstest 100%



http://www.minecraftforum.net/forums/mapping-and-modding/maps/1503376-insanely-enormous-multiplayer-puzzle-pyramid-

Other stuff

- Code Retreat
- TDD as if you meant it

TDD as if you meant it

The rules for *TDD as if you meant it* are:

1. Write *exactly one* new test. It should be the smallest test which seems to point in the direction of a solution
2. Run the test to make sure it fails
3. Make the test from (1) pass by writing the least amount of implementation code you can *IN THE TEST METHOD.*
4. Refactor to remove duplication or otherwise as required to improve the design. Be strict about the refactorings. Only introduce new abstractions (methods, classes, etc) when they will help to improve the design of the code. Specifically:
    1. ONLY Extract a new method if there is sufficient code duplication in the test methods. When extracting a method, initially extract it to the test class (don't create a new class yet).
    2. ONLY create a new class when a clear grouping of methods emerges and when the test class starts to feel crowded or too large.
5. Repeat the process by writing another test (go back to step #1).

http://coderetreat.org/facilitating/activities/tdd-as-if-you-meant-it

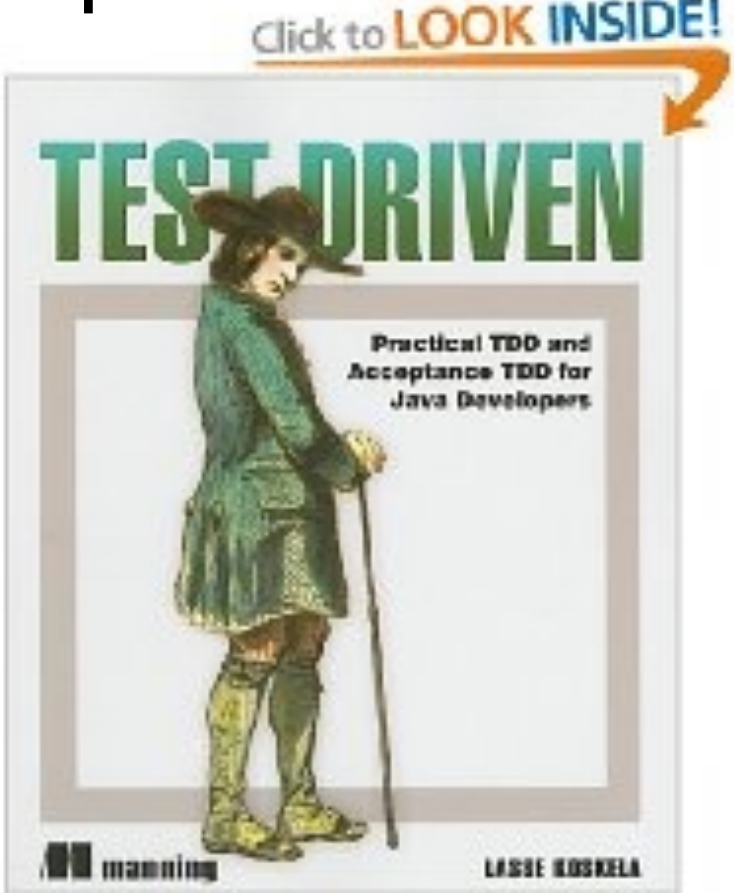
Sammanfattning

- Skriv kod först efter failande test!
- Test → Code → Refactor
- Clean Code
- Clean Tests

- (Exempel på katornas lösningar finns på https://github.com/jonananas/tdd-workshop)

Boktips TDD

- kap 2 och 9 finns på
- http://www.manning.com/koskela/



Fler kator

- http://kata-log.rocks/

Länkar

- Ett antal länkar som kom upp under diskussionerna:
  - Working effectively with Legacy code - http://www.amazon.com/Working-Effectively-Legacy-Michael-Feathers/dp/0131177052
  - Handlar om hur man gör otestade system testbara
  - Bowling kata - http://butunclebob.com/files/downloads/Bowling%20Game%20Kata.ppt
  - Detta är ett exempel på hur man utför TDD, motsvarande Wordwrap som vi körde. Tanken med en kata (från kampsport) är att man utför den tills man kan den utantill, då har man möjlighet att gradera till nästa nivå (bälte).
  - Testdriven webutveckling med wicket: http://www.crisp.se/per.lundholm/doc/wicket_rd.pdf
  - Verktyg för automatiska regressionstester
  - .Net: http://research.microsoft.com/en-us/projects/Pex/ Ska vara riktigt bra!
  - java: http://www.parasoft.com/jsp/products/jtest.jsp Tydligen inte riktigt lika bra :)
  - Verktyg för .Net
  - - nUnit
  - - kodtäckning: ncover, .Test (samma företag som jTest ovan)
  - - pex

Implementation

- Design Patterns
- Software Craftmanship
- Clean code
- Teststrategier
- **TDD**
- (Vad är en bra implementation?)