Översikt

- Implementation
  - Vad är en bra implementation och hur når vi dit?
- Quality Assurance
  - Hur säkerställer vi kvalitet?

Planering

- F1 (idag)
  - Repetition Design
  - Responsible Programmer
  - **TDD**
- F2 (Imorgon)
  - Jan-Erik
- F3 (Nästa måndag)
  - Clean Code
  - Quality Assurance i agil process

Design sammanfattning

- Vad är design?
- Domain Driven Design
- Process: Evolutionary Design
- Designprinciper
- Design Patterns

- Vad är arkitektur?
- Process: Kandidatarkitekturer
- Architectural Patterns
- Dokumentation
- UML

Grundläggande Designprinciper

- Abstraction
- Decomposition and modularization
- Coupling and Cohesion
- Encapsulation and information hiding
- Completeness and primitiveness (DoD)
- Consistency (Ubiquituous language)

Grundläggande Designprinciper

- Reveal Intent - *Naming*
- SRP: Single Responsibility Principle
  - One and one reason only to change
- DRY: Don't Repeat Yourself
  - Duplication: Missed opportunity for abstraction!
- Simple design
- YAGNI: You ain't gonna need it!

Design patterns

Responsible Programmer

Habits of a Responsible Programmer
by Anders Janmyr

TDD:

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

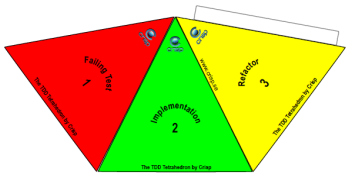
google define: design

"manuals describing the operation and use of programs"

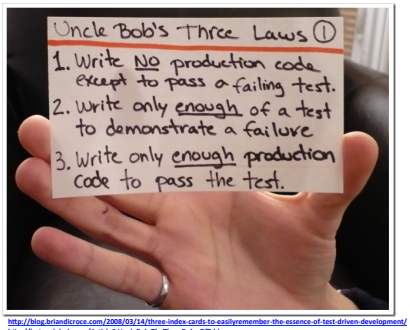
Uncle Bob:
"My documentation is formal, it compiles! How about your documentation?"

TDD – hur gör man?

TDD Tetrahedon



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



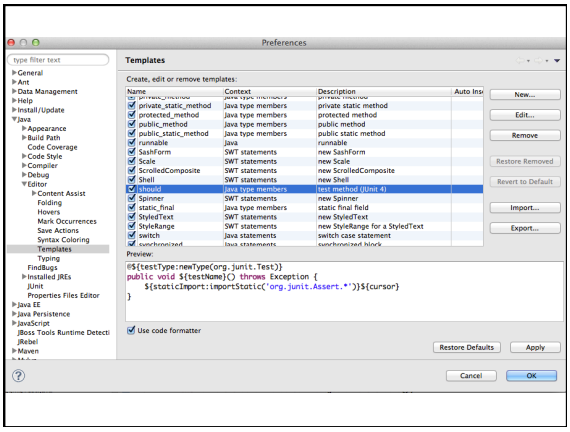
http://blog.brandicroce.com/2008/03/14/three-index-cards-to-easilyremember-the-essence-of-test-driven-development/
http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd

Live Coding: WordWrap

detta är en lite längre mening

→(10)

detta är en
lite längre
mening



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

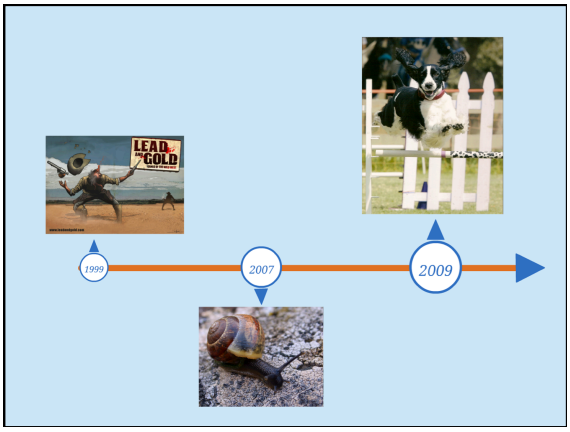
- Design
  - Safe refactoring
  - Coding by Intention
- Dokumentation
- Kvalitet
- Regressionstester
- Continuous Integration
- Continuous Delivery

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 – varför gör du det inte?

TDD – varför inte?

Press att leverera snabbt

Tar lång tid

Måste fortfarande testa ”rätt saker”

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

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

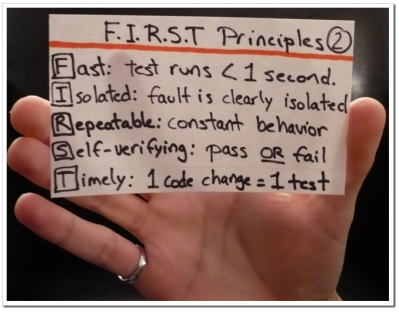
- Studier visar inte tydligt på minskad tidsåtgång
  - 15-35% increase in initial development time
  - No increase
  - etc
- Men vad händer sen?

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

Vad är ett bra test? FIRST!

- Fast
- Isolated (independent)
- Repeatable
- Self verifying
- Timely

- Next slide…



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

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

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

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

Dags för kata!

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

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?

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

Test doubles

- Driv fram interface – implementera senare

Test doubles

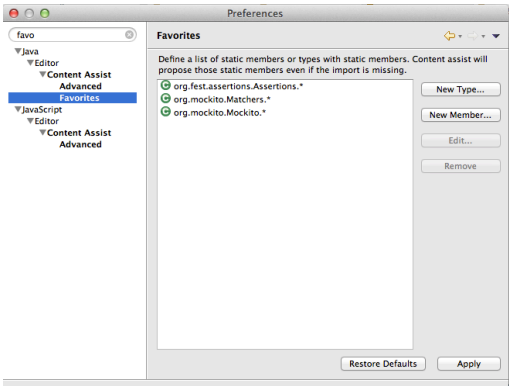
- Exempel med Mockito
- OrderServiceTest

Mockito

- Mockito Maven-coords

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-all</artifactId>
    <version>1.10.19</version>
</dependency>
```

  - http://mvnrepository.com/artifact/org.mockito/mockito-all/1.10.19



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 – typer av test

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

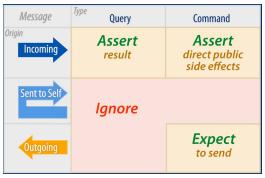
Sköra tester

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

Filmtips

The Magic Tricks of Testing by Sandi Metz 32 min
https://www.youtube.com/watch?v=URSWYvyc42M

För tips om hur man undviker sköra tester



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

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)

Läs!

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

Till nästa föreläsning(?)

- Kolla in filmen, från 19:46 till slutet 40:24

Habits of a Responsible Programmer, Anders Janmyr:

http://oredev.org/2013/wed-fri-conference/habits-of-a-responsible-programmer

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 obestadle 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 Tyvärr inte riktigt lika bra :)
  - Verktyg för .Net
  - - nUnit
  - - kodtäckning: ncover, .Test (samma företag som jTest ovan)
  - - pex