



Test Driven Development and Related Techniques For Non-Developers ;-) IBM 2012

Peter Kofler, 'Code Cop'
@codecopkofler

www.code-cop.org

Peter Kofler

- Ph.D. (Appl. Math.)
- (Java) Software Developer
- with IBM since 1 year
- SDM Costing, IGA, GBS

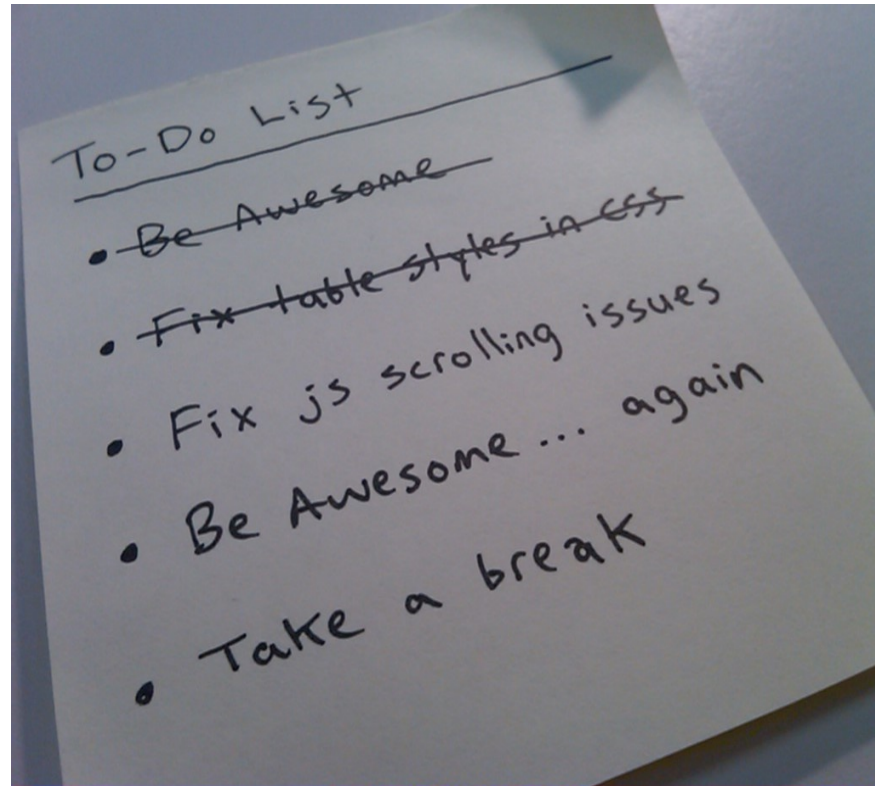


The opinions expressed here are my own and do not necessarily represent those of current or past employers.



Agenda

- 1) What does it look like?
- 2) Why is it important?
- 3) What you need to do!
- 4) Summary



1) So what is it all about?

The Prime Factors Kata (Demo)

What Exactly Will We Do Now?

- write code together
- using TDD
- see techniques and patterns
- discuss while doing



The Requirements.

- Write a class named “PrimeFactors” that has one static method: generate.
 - The generate method takes an integer argument and returns a List<Integer>.
 - That list contains the prime factors in numerical sequence.

<http://butunclebob.com/ArticleS.UncleBob.ThePrimeFactorsKata>

First Some Math.

- **Prime Number:** a natural number > 1 that has no divisors other than 1 and itself.
 - e.g. 2, 3, 5, 61, 67, ..., 997, ..., $2^{43112609}-1$, ...
- **Prime Factors:** the prime numbers that divide an integer exactly without remainder.
 - e.g. $2 = 2$,
 $4 = 2 * 2$,
 $24 = 2 * 2 * 2 * 3$
 $288 = 2^5 * 3^2$

The Requirements (for Mortals).

- code a file/module/class “PrimeFactors”
- code a function/method/routine “generate”
- accept an integer number as parameter
- return the prime factors of that number

Demo

(Code Kata)

Keep the bar green to keep the code clean.



Unit Testing

- test individual units
- isolate each part
- show that the individual parts are correct
- regression testing
- sort of living documentation
- executed within a framework

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

Test-Driven Development

- add a test
- run all tests and see if the new one fails
- write some code
- run all tests and see them succeed
- refactor code **mercilessly**
- „Red Green Refactor“

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

A minute ago all
their code worked

Refactoring

Refactoring is a technique
for **restructuring**
an existing body of code,
altering its internal structure
without changing
its external behavior.

(Martin Fowler)

Refactoring

- small behavior preserving transformations
- sequence of transformations produce a significant restructuring
- each transformation is small, less likely to go wrong
- system is kept fully working after each change
- verified by working tests

٢٦٤٦٨٩/٥٥

مكتبة الإيمان

خمس مان كيبو

مبنى
الشيخ

الجمعية الشرعية للتقوية

الرياضيات

التاريخ

الزياء

الحكم

النظام

الحكم

النظام

الحكم

النظام

الحكم

النظام

الحكم

النظام

الحكم

النظام

الحكم

النظام

الحكم

النظام

الحكم

النظام



Code Coverage

comprehensiveness of tests

Beware!

comprehensiveness \neq quality!

Small Advice

Never measure
developers by
Code Coverage!

Demo

(Coverage in Eclipse)

Demo

(Jenkins)



Continuous Integration

Continuous Integration

- Maintain a code repository
- Automate the build
- Make the build self-testing
- Everyone commits every day
- Every commit should be built
- Keep the build fast
- Everyone can see the results of the build

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

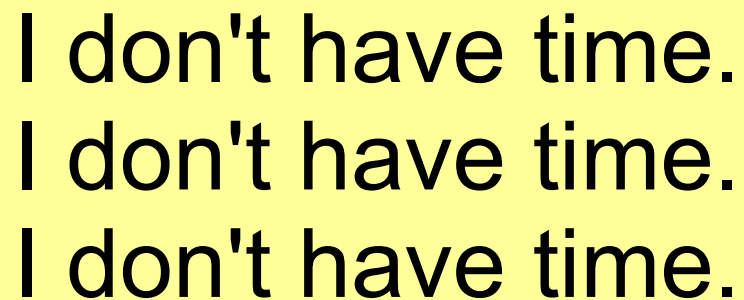
Immediate Feedback and Early Warning

Demo

(Jenkins)

2) Why should we use TDD?

- Writing tests takes time
- and we need to deliver **client value.**



I don't have time.
I don't have time.
I don't have time.

Do you still use that one?



Discussion

- What do **you** think are the benefits of TDD?
 - rapid feedback
 - ...
 - ...

Benefits

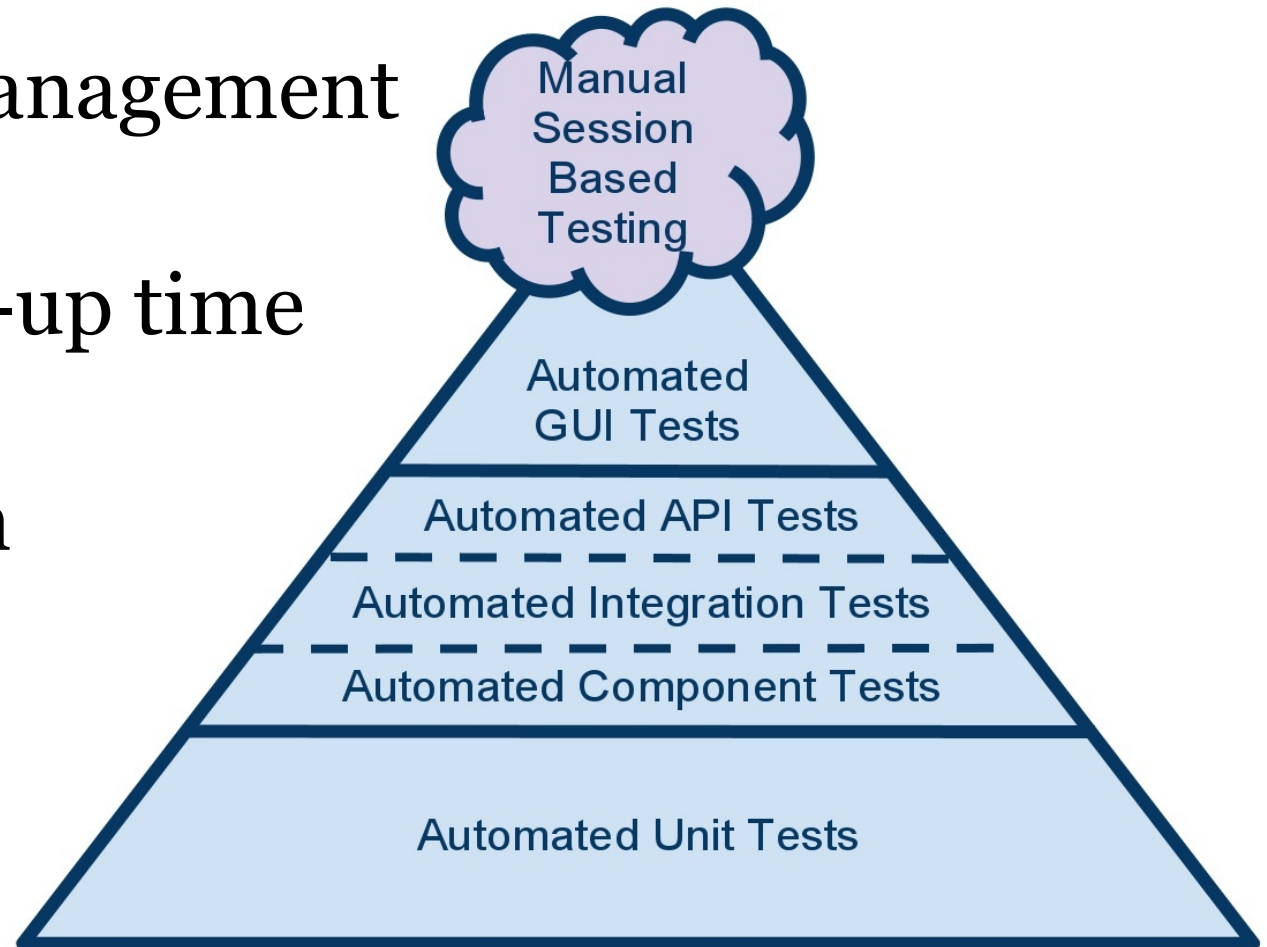
- speed up
- coverage
 - **measurable** validation of correctness
- quality
 - proven to save maintenance costs
- creates a detailed specification

Benefits

- improve design
 - drive the design of a program
 - think in exposed API

Problems

- missing management buy-in
- high ramp-up time
- still need integration tests



TDD will not fix missing skills!

- badly written tests
 - brittle
 - blinking
 - slow
 - duplicate code
 - maintenance burden
 - wrong abstractions
 - not really unit tests

Because TDD is hard!

- need to know OO, design, abstraction, ...
- new style of thinking
- need self-discipline
- (learning curve)



3) What you need to do



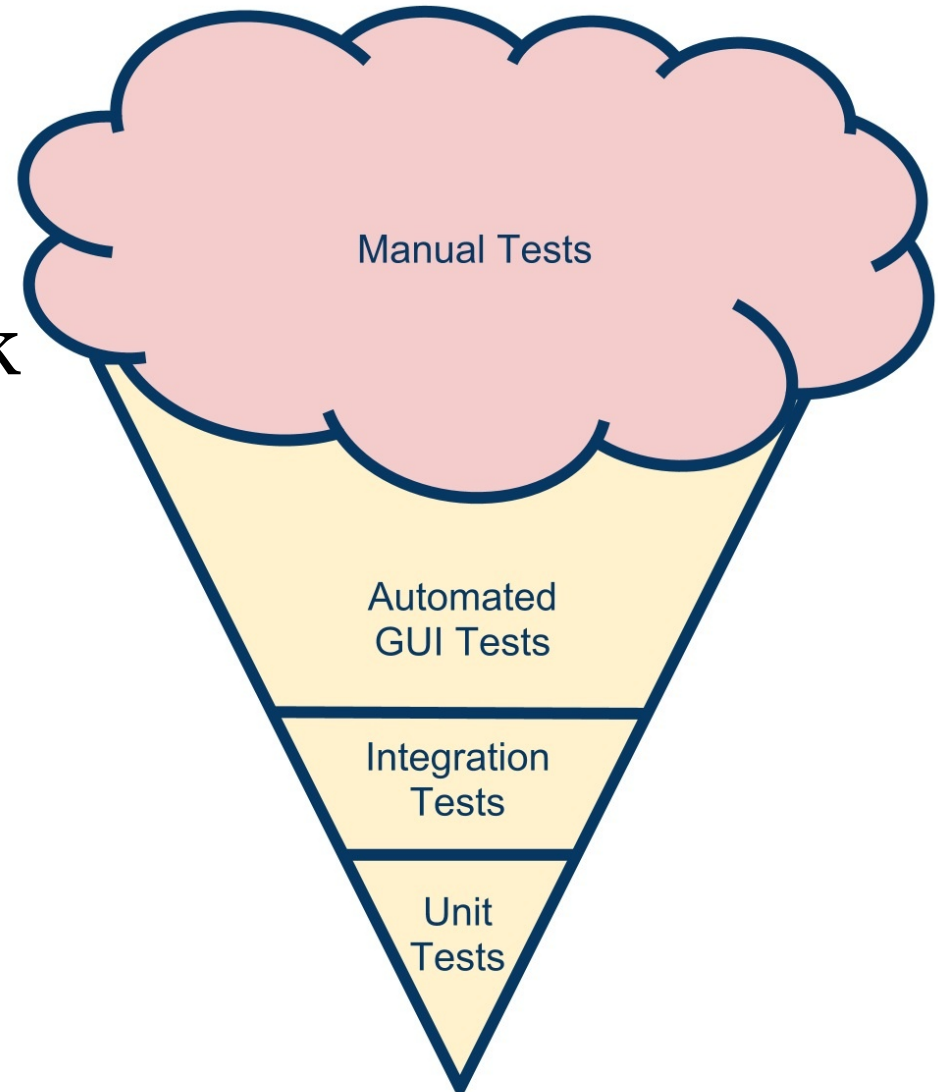
Problem Indicators: CI

- blinking builds
- builds broken for long time
- increasing build execution time
- decreasing code coverage



Problem Indicators: Tests

- slow
- fragile
- no detailed feedback
- “Ice-Cream Cone”



Problem Indicators: Developers

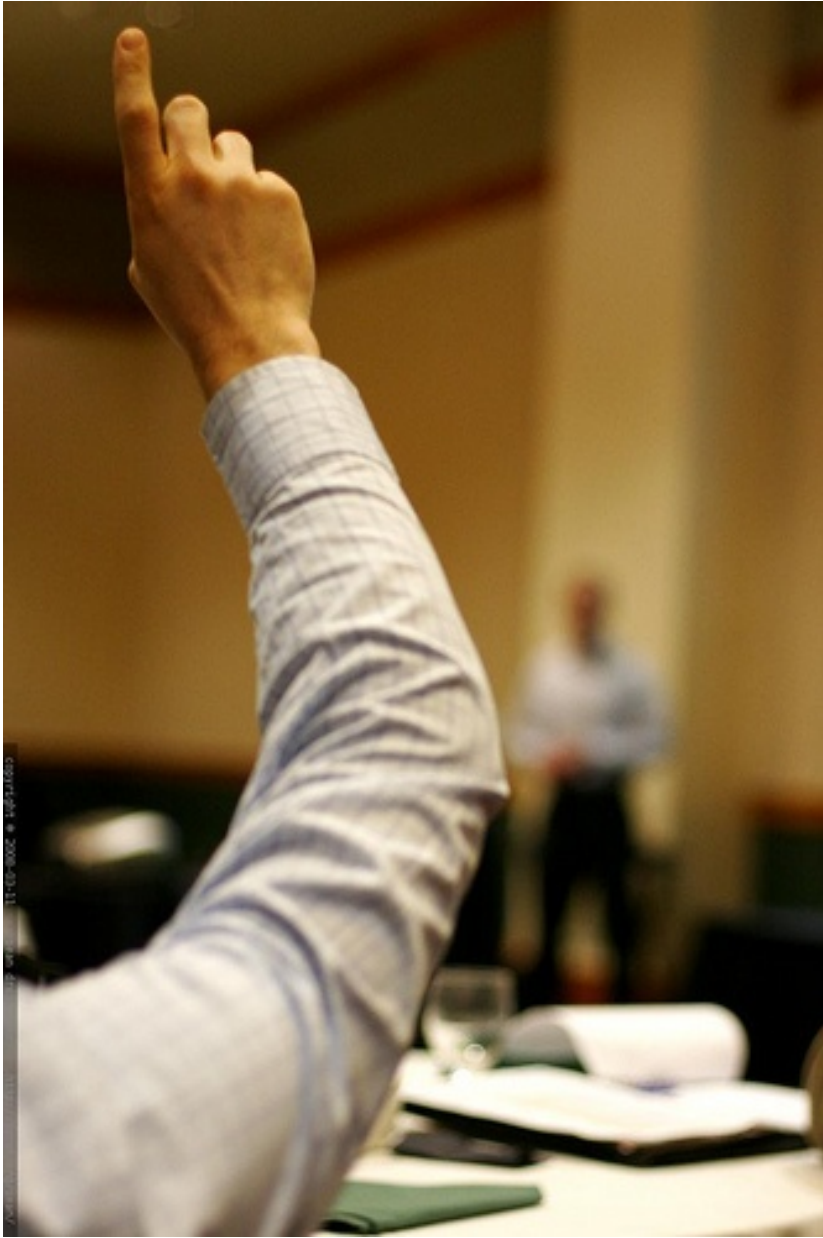
- claiming that they can't write tests
- claiming that they can't fix tests
- wanting to delete tests

“Buzzwords” Summary

- Techniques
 - Unit Testing
 - Test Driven Development
 - Refactoring
 - Code Coverage
 - Continuous Integration
- Management Buy-In
- Keep an eye on the CI server

Support TDD

Do not compromise
on techniques!



Thank
You



Peter Kofler



@codecopkofler

www.code-cop.org

CC Images

- Drive: <http://www.flickr.com/photos/hjem/367306587/>
- Judge: <http://www.flickr.com/photos/eldave/6169431454/>
- List: <http://www.flickr.com/photos/kylesteeddesign/3724074594/>
- Question mark: <http://www.flickr.com/photos/oberazzi/318947873/>
- Fence: <http://www.flickr.com/photos/30830597@No8/3630649274/>
- Coverage: <http://www.flickr.com/photos/paulk/3166328163/>
- Works: <http://www.codinghorror.com/blog/archives/000818.html>
- Cash: <http://www.flickr.com/photos/mindfire/315274981/>
- Steep: <http://www.flickr.com/photos/worldofoddy/229501642/>
- Want You: <http://www.flickr.com/photos/shutter/105497713/>
- Warn: <http://www.flickr.com/photos/hugosimmelinck/2252095723/>
- Questions: <http://www.flickr.com/photos/seandreilinger/2326448445/>