Werner Wild

Test Driven Development

Silver Bullet

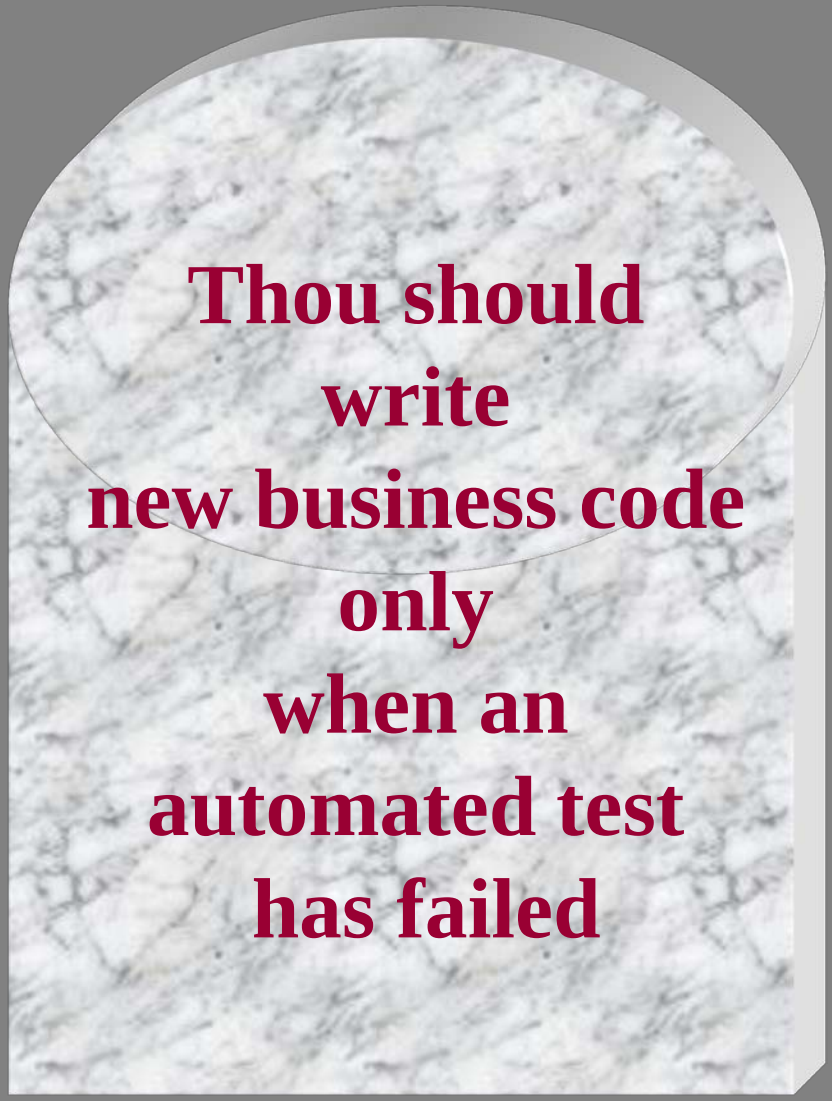
or

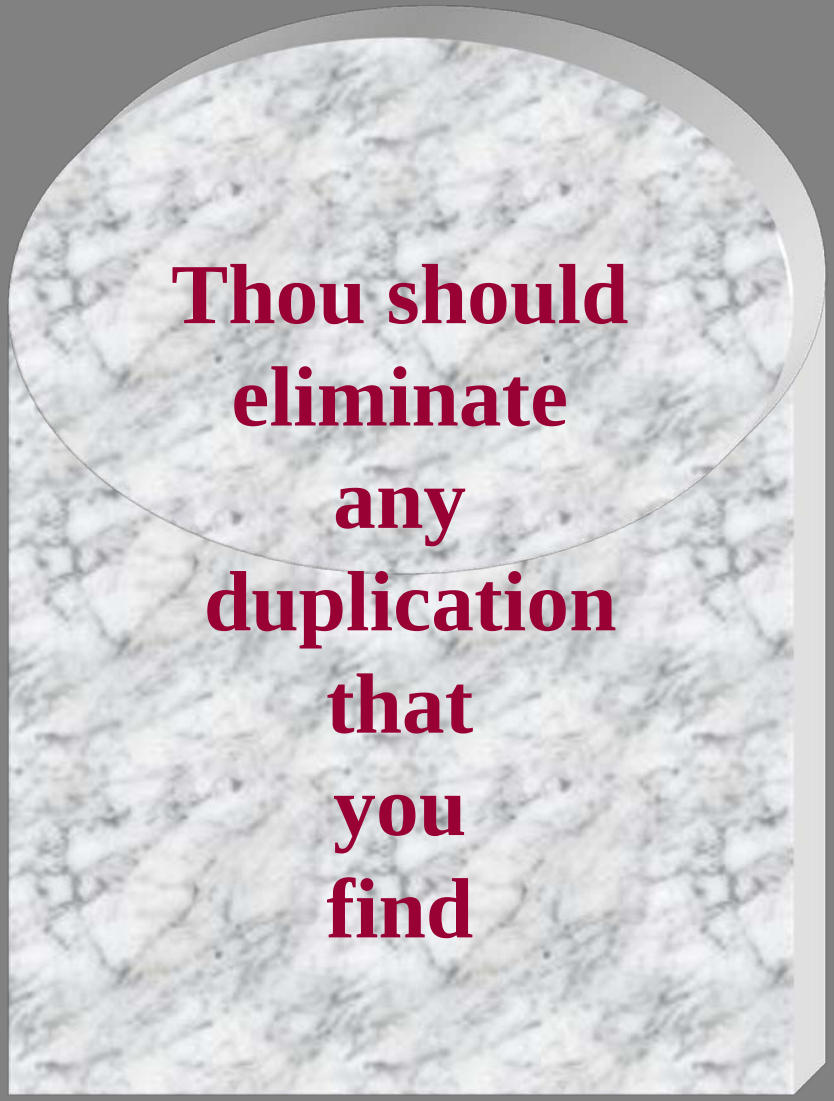
Pandora's Box ?

What is Test Driven Development (TDD) ?

- An iterative technique to develop software
- As much (or more) about design as testing
  - Encourages design from user's point of view
  - Encourages testing classes in isolation
  - Produces loosely-coupled, highly-cohesive systems
- As much (or more) about documentation as testing – executable documentation !
- Must be learned and practiced
  - If it feels natural at first, you're probably doing it wrong

The Two Commandments





Generated Behavior & Technical Implications

- We must design organically, with running code providing feedback between decisions
- We must write our own tests, can't wait 20 times a day for someone else to write a test
- IDE must provide rapid responses to small changes (e.g. incremental compiling)
- Designs must consist of many highly cohesive, loosely coupled components (just to make testing easier)

The Rules of Simple Design IN PRIORITY ORDER!

1. The code passes all tests
2. There is no duplication
3. The code expresses the programmer's intention
4. Using the smallest number of classes and methods

Higher priority rules must be satisfied first !

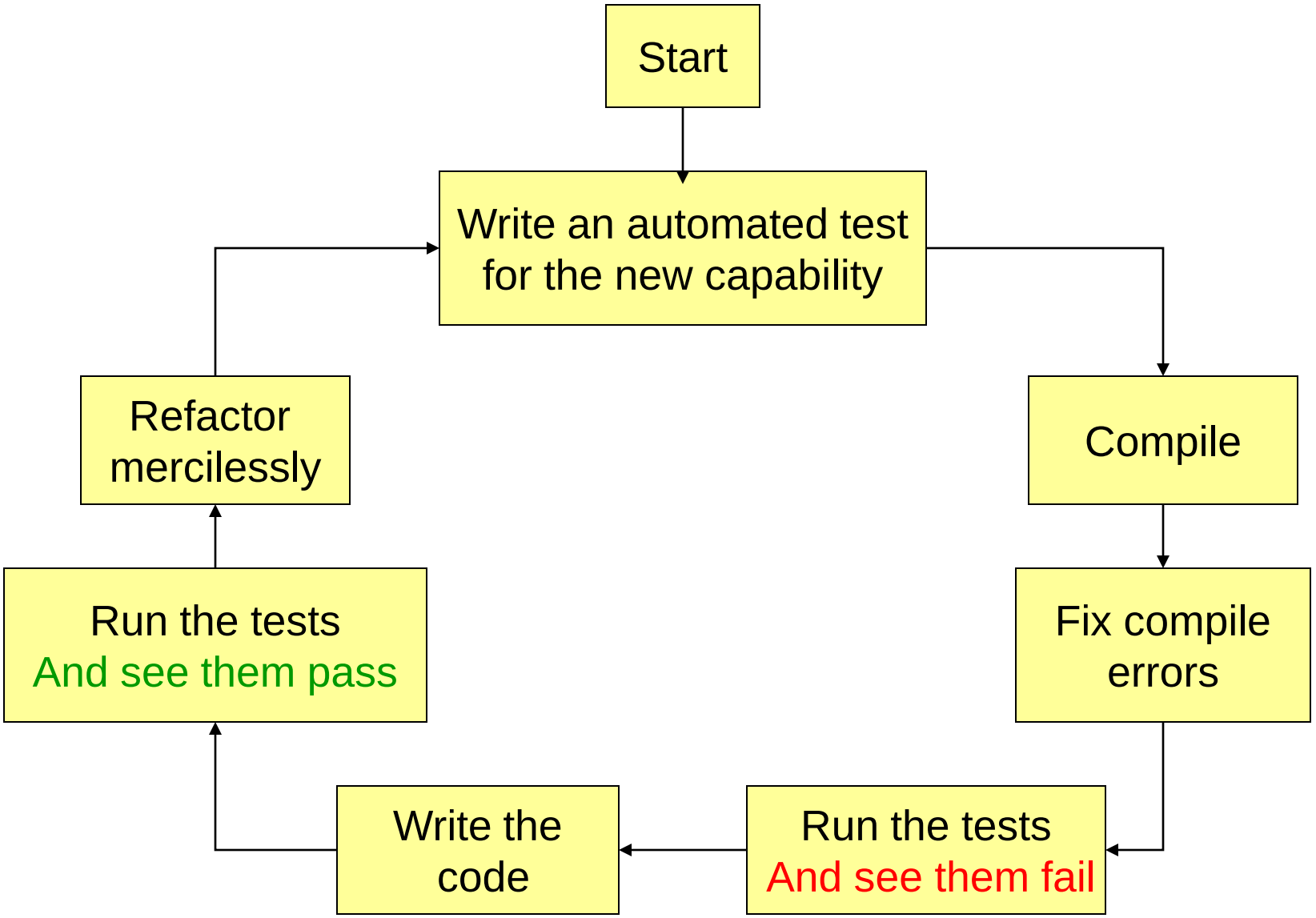
TDD's Mantra

Red / Green / Refactor

TDD's Mantra

- Red – write a little test that doesn't work – perhaps doesn't even compile – first
- Green – Make the test work quickly, committing "whatever sins" are necessary in the progress
- Refactor – Eliminate all the duplication created in merely getting the test to work

The Test Driven Development Cycle



Refactoring

Restructuring of software
by applying a series of internal changes that do not affect its observable behavior

Fowler, Refactoring, 2000

Hints

- Design evolves through coding/feedback
- Write unit tests for every piece of code that could possibly break
- Within the tests, start by writing the assertions first
- Preferred testing tool are xUnits (open source)
- Use a responsive IDE with an incremental compiler and refactoring support
- Develop in pairs, take your time and have relaxing breaks

TDD live !

Advantages

- Code becomes testable
- Great way for agreeing on what problem to solve
- Requirement changes can be incorporated without breaking existing code
- Professed to have
  - Enhanced code quality
  - Faster development time
  - Proper implementation of requirements
  - Simple design
  - The needed – and only the needed !- functionality

Cons

- Needs some practice, is VERY unusual first (especially to people with traditional coding backgrounds)
- Will rarely work among groups of only inexperienced people
- Meaningless metrics like LOC/day will go downhill … However:
- Delivered, trustable functionality per day is significantly higher than usual (suspicion “your not working at the limit, so here is some more work” ☺)
- Your sophisticated, expensive debugger is rarely needed ☺
- SW might be developed too fast, business/customer’s side can’t keep up with the speed ☺

Less Suited

- Security Software
  - human judgment needed
- Concurrency
  - hard to duplicate subtle problems reliably

Higher Level Advantages (cont here)

- Design on demand
  - Code only what is needed to pass the tests
  - Remove all Duplication
  - => get a perfectly adapted design (to the current requirements)
  - => design is equally prepared for all future stories
- Continuous delivery
- Early and ongoing concrete feedback helps managing risks and reduces fear
- Fosters agility (IT and business)

Something to Reflect

Do Agile Projects – not Projectiles

Klaus Wuestefeld, Creator of the Prevayler, OOPSLA 2003

Final thought

The real problem is bad coders

Bill Caputo

Literature

- Kent Beck:
  Test Driven Development by Example (Addison Wesley 2003)
  eXtreme Programming Explained – Embrace Change (Addison Wesley 2000)
  oder
  Extreme Programming Explained: Embrace Change (2nd Edition) (Addison Wesley 2004)
- Martin Fowler:
  Refactoring – Improving the Design of Existing Code (Addison Wesley 2000)
- Mark Denne, Jane Cleland-Huang:
  Software by Numbers (Prentice Hall 2004)
- Johannes Link:
  Unit Testing in Java – How the Test Drives the Code (Morgan Kaufmann 2003)

Literature

- Frank Westphal:
  Testgetriebene Entwicklung mit Junit und FIT(dpunkt Verlag, 2005)
- Mary & Tom Poppendieck:
  Lean Development – An Agile Toolkit (Addison Wesley 2003)
  Implementing Lean Software Development: From Concept to Cash (Addison Wesley 2006)
- Tom DeMarco & Timothy Listener:
  Waltzing with Bears: Managing Risk on Software Projects (Dorset House 2003)

Thanks for your attention!