SIEMENS

CORPORATE TECHNOLOGY

Secrets of Test Driven Development

23. TAV München

Peter Zimmerer

Senior Principal Engineer

Siemens AG

Corporate Technology Software & Engineering 1
D-81730 Munich, Germany
peter.zimmerer@siemens.com
http://www.siemens.com/research-and-development/
http://w4.siemens.com/ct/

Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

Contents

Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

Introduction *Test-driven Development (TDD)*

- **Kent Beck:**
  - Never write a single line of code unless you have a failing automated test.
  - Eliminate duplication.
- **3 steps:**
  - Write test
  - Write code
  - Refactor
- **Other related naming: test-first development**

Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

TDD is one core practice of eXtreme Programming (XP)

eXtreme TESTING

- **First code then test …**

Never write a line of functional code without a broken test case.

Any program feature without an automated test simply doesn't exist.

Kent Beck

- **First test then code …**

Test-first coding is not a testing technique.

Ward Cunningham

Remark:
From my point of view *test-first is testing* because testing is not only test execution at the end but has a lot to do with activities like creating test requirements, test specifications, test design, test cases, etc.

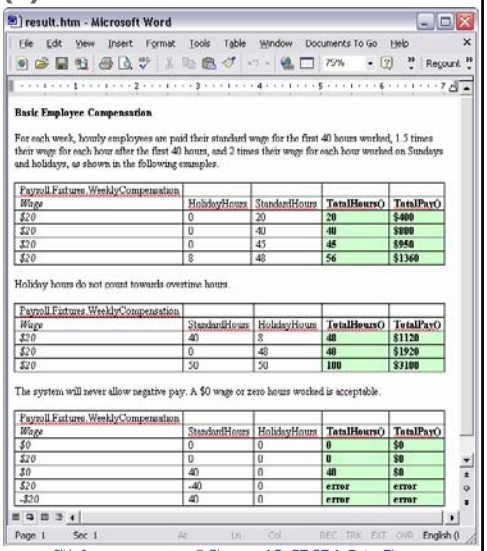
Software & Engineering Development Techniques

Typical usage of TDD (1)

- **Programmer testing (unit testing) – very often used**
  - xUnit tools (http://www.xprogramming.com/software/htm)
  - E.g. JUnit (http://www.junit.org/)
- **Customer testing (acceptance testing) – "partly used" (?)**
  - Fit, acceptance testing framework by Ward Cunningham and others (**F**rameworkFor**I**ntegrated**T**est, http://fit.c2.com/).
    - Tests are specified as HTML tables (created by Excel, Word, etc.).
    - Fixtures act as the glue between the written tests and the application's code.
    - Ideal for data-centric tests where each test does the same kind of thing to different kinds of data.
    - Available in different languages (Java, C++, C#, Python, Perl, Ruby, …).
  - FitNesse, a fully integrated standalone wiki and acceptance testing framework (based on Fit) by Robert C. Martin and Micah D. Martin (http://fitnesse.org/).
  - FitLibrary, a set of fixtures and runners that extend Fit (http://fitlibrary.sourceforge.net/).
  - Others see http://www.xprogramming.com/software/htm (main focus is on web testing over HTTP).

Typical usage of TDD (2)

- **Example: Fit result document (see http://fit.c2.com/wiki.cgi?CustomerGuide)**



SIEMENS

CORPORATE TECHNOLOGY

Typical usage of TDD (3)

- **Customer testing (acceptance testing) – "partly used" (?)**
  - Tests may be too low level to be effective
    - Often tests are expressed in terms of user interface interaction like 'enter text field', 'push button', etc.
      but not on an appropriate business process and terminology level.
    - One possible solution:
      Shift from lower-level to higher-level concepts (abstraction is the key)!
      Building the tests using a specific high level domain-specific testing language.

  - Related *keyword-driven software testing frameworks* which existed already before XP and agile methods arrived:
    - logicaCMG TestFrame (http://www.logicacmg.com/)
    - LogiGear TestArchitect (Action Based Testing™) (http://www.logigear.com/)
    - Omsphere Multiple Interface Testing Suite (MITS) (http://www.omsphere.com/)
    - K. Zambelich's table-driven method of test automation (http://www.sqa-test.com/)

CT SE 1 Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

Some basics of testing

Find bugs early during test specification, not late during test execution!!!

- **Finding bugs during test execution is *not* the optimum.**
- **Specification artifacts (requirements, use cases, models, architecture, design) must be improved and completed for testing which results in changes.**
  - Testing prevents bugs → *building the test specification* (e.g. building a test model or creating abstract non-executable test cases) *is testing.*
  - Tests represent a set of *executable specifications.*
- **Design for testability**
  - Visibility/Observability and control
  - *We (testers and developers) are sitting in the same boat …*

- **So, why should we restrict TDD only to unit and acceptance testing with detailed implemented and executable tests?**

CT SE 1 Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

TDD ≈ Preventative Testing

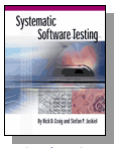
- ***Preventative Testing is built upon the observation that one of the most effective ways of specifying something is to describe (in detail) how you would accept (test) it if someone gave it to you.***
  Bill Hetzel, <1990
- **Use testing to influence and control requirements, architecture, design, implementation, deployment, and maintenance:**
  → *testware development leads software development.*
- ***That means that the idea of TDD is***
  - *nothing actually new*
  - *nothing new brought to us by XP or agile methods and the hype around it*
  - *rather a quite old idea …*

Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

TDD – Example



Ref.: R. Craig, S.P. Jaskiel
Systematic Software Testing

- **Automated Teller Machine (ATM) requirement:**

  A valid user must be able to withdraw up to $200 or the maximum amount in the account.

- **The first 2 test cases**

  TC1: Withdraw $200 from an account with $165 in it. Result ???

  TC2: Withdraw $168.46 from an account with $200 in it. Result ???

already help us to discover two ambiguities in the requirements:

Some people will interpret it to mean that the ATM user can withdraw the lesser of the two values ($165), while other people will interpret it to mean they can withdraw the greater of the two values ($200).

Does the bank want the ATM to dispense coins to the users???

Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

TDD secret 1 – My view of TDD

- **TDD = Test-first *design* ⊕ test-first *implementation***
  - Including early creation of abstract non-executable test cases as well as detailed implemented and executable test cases.
- **TDD is possible and strictly recommended on *every* test level, *not* only for unit and acceptance testing (→ *preventative testing*)!**
- **Emphasizes the importance and benefits of early testing activities.**
  - *Building the test specification is testing.*
  - Tests represent a set of *executable specifications.*
  - Proactive design for testability.
- **These things are NOT actually new and are already well known for a long time.**
- **What's new is that these things are more and more really used in projects today. From my point of view that's the real big benefit of the *TDD-hype brought to us by XP and agile methods*.**

Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

TDD secret 2 – Test-first implementation

- **Not always 100% possible in real life**
  - Usage might be costly and time-consuming dependent on what test environment (e.g. by creating mock objects) is needed.
  - Legacy code with low testability.
  - GUI testing (e.g. Java Swing, capture/replay tools)
  - Web applications (using ASP.NET, JSP, servlets):
    Tests could be created to check the HTML output of the code, but that doesn't really test that the HTML code itself is properly displayed within the browser.
  - Distributed objects (e.g. EJB) deployed on application servers
  - Code running on different types of machines and interacting with a complex environment: e.g. communication servers, middleware servers, database servers, content management systems, web interfaces, etc.
  - Event-based reactive systems
  - Embedded systems

Software & Engineering Development Techniques

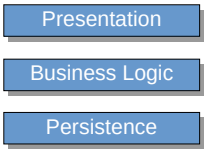
SIEMENS

CORPORATE TECHNOLOGY

TDD strategy for GUI testing

- **The more code you can make testable the more reliable the system will be.**
- **Divide the code into appropriate components that can be built, tested, and deployed separately.**
- **Build most of the functionality (business logic, services) outside the context of the user interface code using TDD.**
- **Let the user interface code be just a very thin layer on top of rigorously tested code. I.e. build as much functionality as possible outside the GUI.**

- **Again, this "good" basic architecture style of a clear separation between business logic and user interface is NOT new and is already well known for a long time: e.g. 3-tier architecture.**



Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

TDD secret 3 – Innovation and invention (1)

- **Not always 100% possible in real life**
  - Is it an enemy of innovation and invention?
  - Example:
    Invention of the car
    by Carl Benz and Gottlieb Daimler
    in 1886.

Only 70 years later:

Invention of the crash test
by Mercedes-Benz in the fifties.
Systematic crash tests
since 1959.

Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

TDD secret 3 – Innovation and invention (2)

- **Not always 100% possible in real life**
  - Example **Software technologies (e.g. object-orientation (OO), web technologies, aspect-oriented programming (AOP), grid computing):** When looking back in history we can see that first there was the idea, vision and invention of these new software technologies and later people thought about needed strategies, methods, and tools for testing them.
  - Example **Architectural and design patterns:** These patterns claim to contain innovative approved best practices. Here again the patterns have been invented first (or some have been reinvented based on already known knowledge) and later people started to think about how to test a specific design pattern, i.e. also design patterns have not been developed in a core TDD manner.
  - Example **Testing tools:** What's about all the innovative testing tools we get from the commercial testing tool industry? Do you think these tools are usually developed in a TDD manner? I don't think that …

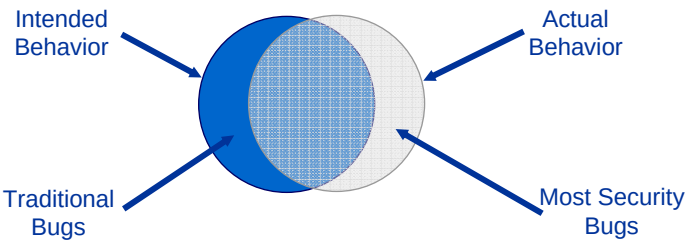
Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

TDD secret 4 – Non-functional requirements

- **Not always 100% possible in real life**
  - Non-functional requirements: performance, usability, etc.
  - Example: Security (Ref. James Whittaker)



Software & Engineering Development Techniques

TDD secret 5 – Cost efficiency and predictability

- **Think about cost efficiency again …**
  - What about continuously changing early requirements and architectural prototypes → rework in testing?
  - Find the right balance in your project …
- **Not enough in real life**
  - Test cases created using a test-first approach or generated from (always incomplete) requirements/design are never enough.
  - You cannot predict everything
    → use techniques like exploratory testing as well.
  - Decisions made during implementation won't be well-tested by tests created only from the design.

Experiences

- **TDD increases visibility and importance of testing.**
- **TDD needs changes in development:**
  process, people (including management!), tooling.
- **TDD results in a closer cooperation of testers and developers.**
- **Question from the developers:**
  How do I test private member functions?

  From a technical point there are different answers dependent on the used programming language (e.g. friends in C++, reflection in Java).

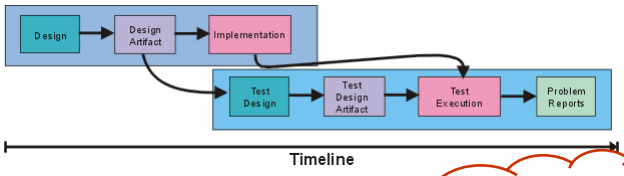
  "In *core TDD* this question is NOT allowed, i.e. it does not make sense, because in TDD we *first* have the designed and implemented test and *then* do *some* implementation for this test.
  I.e. the details of the implementation do not matter so much …"

SIEMENS

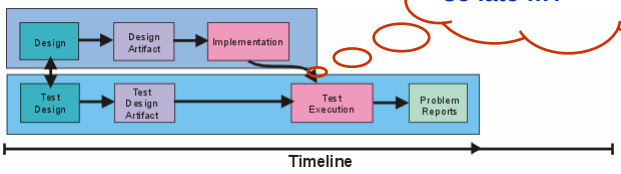
CORPORATE TECHNOLOGY

TDD – Changes (Ref. IBM Rational Unified Process)

- **Traditional**



- **Test-first design**



But, why does the test execution happen so late …?

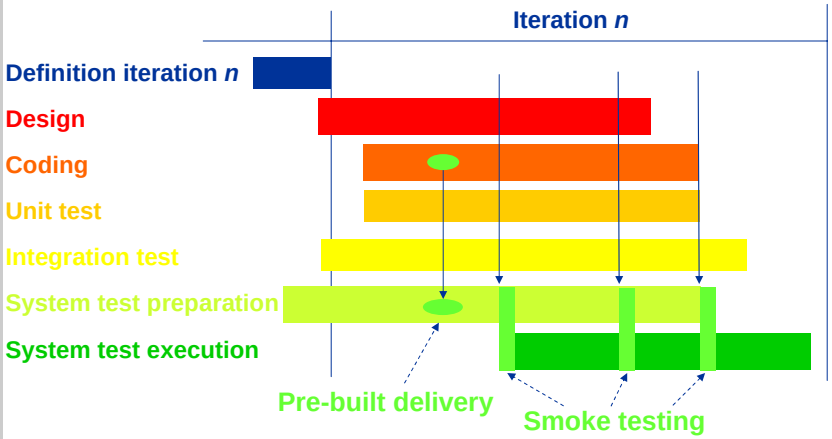
Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

Example for a test workflow visualizing TDD

Test-driven development is often used in the context of an iterative/incremental, agile development process but TDD is not restricted to it.



Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

Summary Test-First ⊕ Test-Second

- TDD = Test-first design ⊕ test-first implementation
- TDD is possible and strictly recommended on every test level, not only for unit and acceptance testing (→ *preventative testing*): *Let testing drive your development and maintenance at all!*
- TDD needs changes in development: process, people, tooling.
- TDD results in a closer cooperation of testers and developers.
- TDD is neither 100% possible nor sufficient in real-world projects.
- TDD does not completely replace conventional "afterwards" software testing: so, do test-first as well as *test-second*.
- The right project specific balance is the key for cost efficiency.
- If not already done then start with TDD tomorrow!!!

CT SE 1 Software & Engineering Development Techniques

SIEMENS

CORPORATE TECHNOLOGY

The end

Only by doing and experiencing it you will get the feeling what's really behind TDD ...

Experience is a hard teacher because she gives the test first, the lesson afterwards.

Vernon Sanders Law (*1930)
Baseball pitcher at Pittsburgh Pirates

Every test should result in a lesson learned. Doing TDD you will get the lessons early ...

I wish you much success for your own experiences in the world of test-first & *test-second*.

CT SE 1 Software & Engineering Development Techniques