Test-driven development – does it really ensure quality?
Trial lecture

Anca Deak
26 November 2015
Trondheim, Norway
Agenda

• Testing overview
• Agile unit testing
• What is TDD and why use it?
• TDD in literature
• Reflections on TDD
• Summary and future work
Software testing

Software testing = process of analyzing a software item to detect the differences between existing and required conditions (bugs) and evaluate the features of the software item. (IEEE 829)

Verification
• The software should confirm to its specification
• Are we building the product right?

Validation
• The software should do what the user really requires
• Are we building the right product?

Basically: **check that the developed product meets the customer’s requirements and expectations and the customer is happy!** 🙅‍♂️
Types of testing

• White box testing
  – In white box testing internal structure of the program is taken into account
  – Knowledge of the internal program design and code required.
  – Tests are based on coverage of code statements, branches, paths, conditions.
Types of testing

• Black box testing
  – No knowledge of internal program design or code required.
  – In black box testing the internal structure of the program is hidden from the testing process
  – Tests are based on requirements and functionality
Testing levels

- **Unit testing**
  - Tests each module individually
  - Follows a white box testing approach
  - Done by developers
  - Testing level where TDD is applicable
Testing levels

• **Integration testing**
  - Checks for errors associated with interfacing.

• **System testing**
  - The entire system is tested for requirements
  - Overall system function and performance are checked

• **Acceptance Testing**
  - It is performed by the customer at its site
  - Determines if the delivered system is accepted or rejected
Agenda

• Testing overview
• **Agile unit testing**
• What is TDD and why use it?
• TDD in literature
• Reflections on TDD
• Summary and future work
Agile software development

SW Dev Method
- small iterations
- each iteration is complete dev. cycle
- product is released after each iteration

Benefits
- fast feedback regarding the product
- focus on people and interactions
- working software is delivered frequently
- face-to-face communication.
- adapted to handle changes

Agile Umbrella

Lean SD
Kanban
Xanpan
Scrum
ASD
XP
DSDM
CI

Stand up meetings
TDD
BDD
ATDD
WIP limits
Iterations
Planning poker
Retrospectives
Unit testing in agile

Tools and technique employed for quality assurance:

• continuous integration
• automated unit testing
• pair programming
• domain-driven design
• code refactoring
• test-driven development
Testing levels and TDD

- Unit Test
- Integration test
- System test
- Acceptance test
- Test individual component
- Test Component groups
- Test the integrated system
- Test the final system
- TDD
Agenda

• Testing overview
• Agile unit testing
• What is TDD and why use it?
• TDD in literature
• Reflections on TDD
• Summary and future work
TDD (Test Driven Development)

• An essential XP best practice, but can be applied as a stand alone one (Astels)

• A programming technique that promotes code development by repeating short cycles where each cycle combines Test-first development with refactoring. (Beck)

• Basically: **Write a test before coding and refactor your code!**
The TDD method involves following steps:

1. Write a test for an unimplemented functionality or behavior;
2. Supply the minimal amount of code; to pass the test;
3. Refactor the code;
4. Check that all tests are still passing after the changes were done.
Perceptions of TDD

• Primarily a development technique
• Design style
  – refactoring = refine the component design
• Testing technique
  – Does not replace traditional testing, but ensures effective unit testing
• Helps in clarifying the requirements
  – involves analyzing the requirements of a portion of an implementation
Why use TDD?

• Make you think twice about the requirements

• **Test coverage** - code is never written without first writing a test

• **Documentation** – writing tests describe your understanding of code behavior

• **Confidence** increase for developers

• Increases the **reliability** of the developed product
More on “Why use TDD?”

• Less debugging required
  – especially for the Greenfield projects

• Increased protection from defects
  – You will know that if it fails, it was the last change who broke it

• Creates a base catalog for automation
Agenda

• Testing overview
• Agile unit testing
• What is TDD and why use it?
• TDD in literature
• Reflections on TDD
• Summary and future work
How to evaluate TDD effectiveness

The body of knowledge has measured the TDD effectiveness by looking at:

• Internal code quality
• External code quality
• Productivity
What is software quality?

• Software functional quality reflects how well it complies with or conforms to a given design, based on functional requirements or specifications.

• Software structural quality refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability.
Internal quality: evaluates the internal design of the code as seen by developers

- code and test coverage
- good auto-documentation
- code complexity
- concision
- cohesion
External quality: indicated by number of defects per code size unit or passing test cases

- all the properties of the software as a product that make this product usable and enjoyable
- reliability
- accuracy
- ease of use and comfort
- adaptability
Productivity measures the development effort as the number of given units in a given time frame

- lines of code per hour (code standard, right metric)
- number of user stories implemented

“Measuring programming progress by lines of code is like measuring aircraft building progress by weight.” (Bill Gates)
TDD in Literature

Findings from literature show contradictory and varying results for TDD

**Positive influence**
- External quality is positively influenced by the TDD (Nagappan et al.)
- Improvement of productivity (Erdogmus et al.)
- High test coverage

**Negative or inconclusive results**
- Decrease in quality
- Lower productivity
- No difference
Findings from literature reviews show contradictory and varying results for TDD

<table>
<thead>
<tr>
<th>Literature reviews</th>
<th>Year</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kollanus</td>
<td>2010</td>
<td>TDD effects on Quality and Productivity</td>
</tr>
<tr>
<td>Sfetsos and Stamelos</td>
<td>2010</td>
<td>Agile practice(TDD, pair programming) influence on Quality and Productivity</td>
</tr>
<tr>
<td>Shull et al.</td>
<td>2010</td>
<td>TDD effects on Internal Quality, External Quality and Productivity</td>
</tr>
<tr>
<td>Turhan et al.</td>
<td>2010</td>
<td>TDD effects on Internal Quality, External Quality and Productivity</td>
</tr>
<tr>
<td>Rafique and Misic</td>
<td>2011</td>
<td>TDD effects on External Quality and Productivity</td>
</tr>
<tr>
<td>Munir et al.</td>
<td>2015</td>
<td>Review of literature reviews with focus on rigor and relevance</td>
</tr>
</tbody>
</table>
Results from literature reviews - rigor and relevance

• A lack of rigor was highlighted in the existing research
  – Insufficient details about context
  – Poor study design
  – Threats to validity not addressed

• Relevance of the studies were also under discussion
  – Practical impact of the results
  – Realism of the setup
Results from literature reviews (1)

• Using small sets of observed parameters/metric made it difficult to compare the results between studies

• Different results between experiments and case studies

• Experiments with students are not best suited, longer and more complex projects are recommended

• Semi-industrial environments where proposed
Results from literature reviews (2)

• Seasoned developers were more willing to work with TDD than fresh programmers

• Further industry experiments and longitudinal case studies are recommended

• **External quality is positively influenced by TDD but comes with a decrease of productivity**
Limitations of TDD (1)

- Adopting TDD leads to reduced productivity in the beginning. (more time needs to be allocated for creating tests)

- Not suited for non-functional requirements, data base testing, network configuration, or hardware related issues running these tests will take too much time or require specific settings

- Change of business requirements requires rewriting of the test or renders them unusable
Limitations of TDD (2)

• Not over rely on TDD: all the tests pass should does not imply that the code is bug free

• Developers have to learn to write unit tests

• As the product complexity increases, more tests will take more time to write, to run and to maintain

• Depends on programmer motivation and discipline

• Some developers simply don’t like to write tests
Agenda

- Testing overview
- Agile unit testing
- What is TDD and why use it?
- TDD in literature
- Reflections on TDD
- Summary and future work
Reflections on TDD (1)

- It is used in the first phase of implementing a software product
- It’s a practical way of documenting and building automation
- Coupled with pair programming, it increases communication in the team and can be used as a mentoring process
Reflections on TDD (2)

• Not practical for handling legacy code but refactoring legacy code provides opportunity to write unit tests.

• The use of TDD tends to decrease when the software complexity increases
Reflections on TDD (3)

• **Testing frameworks** provide a practical approach for performing TDD and tracking test status by running reports. (JUnit, Google Test)

• **Continuous Integration** provides an automatic and fast way of checking the unit test status
Agenda

• Testing overview
• Agile unit testing
• What is TDD and why use it?
• TDD in literature
• Reflections on TDD
• **Summary and future work**
Summary

- TDD effectiveness is likely to vary from context to context (team to team, product to product).

- Claims of improved quality which are sometimes coupled with a decrease of productivity due to more time allocated to writing tests.

- There is little data to support claims of improved productivity or software design (low level design).
Recommendations for future work

Further empirical studies on TDD:

- Increased validity
- Common metrics for internal code quality
- Longitudinal studies in industry
Thank you!

Q&A
References - 1


References - 2