

---

# Human Aspects of Agile Software Engineering: Cognitive and Social Analysis

---

Orit Hazzan

Technion - Israel Institute of Technology

Based on my joint work with [Yael Dubinsky](#), [Jim Tomayko](#),  
[Uri Leron](#), [Irit Hadar](#), [Tali Seger](#) and [Meira Levy](#).

Israeli Scrum Usergroup, July 15, 2009

# Problems with software development

- **Google: "problems with software development"**
  - Requirements are complex
  - Clients usually do not know all the requirements in advance
  - Requirements may be changing
  - Frequent changes are difficult to manage
  - Process bureaucracy (documents over development)
  - It takes longer
  - The result is not right the first time
  - It costs more
  - Applying the wrong process for the product

---

# Introduction

- Traditional development processes
  - Software intangibility
  - How does **agile** software development improve software projects' results?
    - Increases transparency
    - Reduces cognitive complexity
  - Illustrations: Collaboration, Abstraction, Testing
-

---

# Part A - Collaboration

---

# Bonus Allocation

	Personal Bonus (% of the total bonus)	Team Bonus (% of the total bonus)	How this option will influence the collaboration of the team members?
a	0	100	
b	20	80	
c	50	50	
d	80	20	
e	100	0	

X% personal, Y% team means that X% of the total bonus is divided on a personal basis and Y% is divided equally between the team members.

# Bonus Allocation

- **A conflict:** Individual vs. group interests
  - **Game Theory:**
    - analyzes human behavior
    - decision making
    - decisions are mutually interdependent
    - players wish to maximize their profit
  - **Prisoner's Dilemma:** Illustrates cooperation in situations in which people can not verify that their cooperation is reciprocated.
-

# The Prisoner's Dilemma (A / B years in prison)

	B cooperates	B competes
A cooperates	No evidence <b>2, 2</b>	Evidence against A; B is released <b>10, 0</b>
A competes	Evidence against B; A is released <b>0, 10</b>	Evidence against both; punishment is decreased because they cooperated with the police <b>5, 5</b>

# The Prisoner's Dilemma: The case of software development - A's perspective, Bonus

	B cooperates	B competes
A cooperates	50%	20%
A competes	80%	0%

# The Prisoner's Dilemma: The case of software development - Bonus

	B cooperates	B competes
A cooperates	2, 2	10, 0
A competes	0, 10	15, 15

---

# The Prisoner's Dilemma:

## The case of software engineering

- Usually, team members are asked to cooperate.
  - **Software intangibility:** Team members can't verify that their cooperation will reciprocate.
  - **According to the Prisoner's Dilemma:** Team members will tend to compete.
  - **In software development:** Such behavior leads to the worst results for **all** team members.
-

---

# The Prisoner's Dilemma: The case of agile software development

- How *cooperation* and *trust* can be achieved?
    - establish an environment which enables to verify that one's cooperation is reciprocated.
      - *Transparency*
      - *A set of specific activities* (practices) that all team members are committed to apply.
-

---

# Agile Software Development: Transparency

- Whole team
    - development environment
    - daily stand-up meetings
  - Short releases
    - planning sessions
    - balanced load
  - Testing
  - Measures
  - Customer involvement
-

---

# Part B - Abstraction

---

---

## What is common to the following statements?

- “I need to gain a global view at the application in order to know how this method fits into it”.
  - “I truly believe that if I had a minute to think about these two objects more abstractly, I’d have come up with the conclusion that they can be extracted into one class. But I must move on to the next development task”.
-

# What is common to the following statements?

- "I need some time to think about the code without being swamped with all the details. I'm almost sure that if I could leave now and go to swim, I could have come up with a solution. But I must stay late as all the others on my team".
- "I wish I could join the programmer when s/he writes the code. You ask why? I'm not sure if this design can be implemented in c++."

---

## What is common to the following statements?

- “I need to gain a **global view** at the application in order to know how this method fits into it”.
  - “I truly believe that if I had a minute to think about these two objects **more abstractly**, I’d have come up with the conclusion that they can be extracted into one class. But I must move on to the next development task”.
-

---

# What is common to the following statements?

- “I need some time to think about the code without **being swamped with all the details**. I’m almost sure that if I could leave now and go to swim, I could have come up with a solution. But I must stay late as all the others on my team”.
  - “I wish I could join the programmer when s/he writes the code. You ask why? I’m not sure if this design **can be implemented** in c++.”
-

# What is common to the following statements?

- "I need to gain **global view** ..."
- "I truly believe that had I had only a minute to think about these two objects **more abstractly**..."
- "I need some time to think about the code without being swamped with all the **details**..."
- "...I'm not sure if this design can be **implemented** into C++."

---

**The need to move between abstraction levels**

---

# Abstraction

*single*



*multiple*

---

---

# Multiple abstraction levels in agile software development environments

## ■ Planning Sessions

- the **release** planning session: **high** abstraction level;
  - the **iteration** planning session: **lower** abstraction level.
  - the **entire team** participates in the planning game; developers see the **entire picture** of the system as well as **its parts**.
-

---

# Multiple abstraction levels in agile software development environments

- **Short Releases**

- guide not to stay for too long a time in too high or too low level of abstraction

- **Pair Programming**

- **Refactoring**

---

---

# Part C - Testing

---

---

# Outline - Testing

- Why people don't like testing?
    - cognitive, social, affective, managerial reasons
  - TDD includes
    - automatic tests
    - test first approach
  - How Test Driven Development (TDD) may help?
    - increases transparency
    - reduces cognitive complexity
-

---

# Why people don't like testing?

- **Observation 1:** In traditional development environments, testing appears as one of the last stages and is usually done under pressure.
  - TDD is done all the time and is not postponed to the end of the process.
-

---

# Why people don't like testing?

- **Observation 2:** Testing may give a **negative feedback** (and who likes it?)
  - TDD ends with a success.
-

---

# Practitioners' reflection

- Code Unit Test First

Why don't people like testing? Well, the traditional way of testing is tough to take. You write what seems to be perfectly sensible code, then you write a test and the test tells you that you failed. No one wants to hear that.

Let's turn it around. Write the test first; run it. Of course it fails.. You haven't written the code under test yet. Start writing code.. keep testing. Soon, the test will tell you that you've succeeded! MichaelFeathers

---

---

# Why people don't like testing?

- **Observation 3:** **Testing** in traditional environments **is done by someone else**
    - **The responsibility is transferred**
  - **TDD is done by the developers who write the code.**
-

---

# Why people don't like testing?

- **Observation 4:** Testing in traditional software development environments is carried out at the end of the production line, and, inspired by tradition working class jobs, gets **low status**.
  - **All developers write tests.**
-

---

# Why people don't like testing?

- **Observation 5: Managerial difficulties**
    - Testing slows down development.
    - TDD fosters development processes
    - Automatic (not manually) process.
  
    - It's hard to manage testing (when to test).
    - TDD turns development (and testing) to be a controlled process.
-

---

# Practitioner's reflection

- <http://xp.c2.com/UnitTest.html>

It's a wonderful ... experience ... .

I don't write code any other way anymore.

**My code has less problems, I have more confidence and management has more confidence. - sg**

---

---

# Why people don't like testing?

- **Observation 6: Cognitive difficulties**
    - It's hard to know what to test
    - It's hard to know how much testing should be done
  - TDD improves understanding of what you develop
    - What to test
    - How much testing should be performed
-

---

# Practitioner's reflection

- A key aspect of this process: don't try to implement two things at a time, don't try to fix two things at a time. Just do one. When you get this right, development turns into a very pleasant cycle of testing, seeing a simple thing to fix, fixing it, testing, getting positive feedback all the way. Guaranteed flow. And you go so fast! Try it, you'll like it.

[RonJeffries](#)

---

---

# Conclusion - Part C

- TDD may help in coping with
    - cognitive
    - affective
    - social
    - managerialchallenges of testing.
  - How may TDD help?
    - increases transparency
    - reduces cognitive complexity
-

---

# Conclusion

Main challenge of Software Engineering:

**Software intangibility**

- The agile approach:
    - Increases transparency
    - Reduces cognitive complexity
-

---

Thank you.

Questions?

---

Orit Hazzan

Email: [oritha@techunix.technion.ac.il](mailto:oritha@techunix.technion.ac.il)