Agile Software Engineering
- Research and Practice

Dr. Yael Dubinsky
December 2007

Talk at:
Software Engineering and Programming Languages Seminar
Computer Science Department
Ben Gurion University
Agenda

- The agile approach for software development
  - Customer collaboration
  - Exhaustive testing
- Research
  - Involvement of users in the process of software development
  - Governance of software development
  - Measured test driven development
- Analysis framework
The Iteration Stories

Picture
The Development Environment

Picture
Tracking the Process

Picture
A Survey by VersionOne

Value Actually Realized from Agile

Significantly Improved

Improved
Integrating User Evaluation into Software Development Environments*

Yael Dubinsky, Tiziana Catarci, Shah Rukh Humayoun, Stephen Kimani

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"

*This research is supported by the DELOS Network of Excellence on Digital Libraries (http://www.delos.info/).
Agenda

- The User Centered Design (UCD) approach
- Integrating UCD into software development environments
  - Use cases
- The UCD Management Eclipse Plug-in

User-centered design (UCD) is an iterative process to design; it grounds the process in information about the people who will use the product.

“… UCD is an iterative process whose goal is the development of usable systems, achieved through involvement of potential users of a system in system design.” [Karat 1996]

“… user-centred design emphasizes that the purpose of the system is to serve the user, not to use a specific technology, not to be an elegant piece of programming. The needs of the users should dominate the design of the interface, and the needs of the interface should dominate the design of the rest of the system.” [Norman 1986]
Integrating UCD into Software Projects

- Iterative design activities
  - The design is updated regularly as the product evolved
  - The user evaluation is fostered by performing UCD tasks in each iteration of two to four weeks, and the design is updated according to the evaluation on-going outcomes

- Measures
  - Taking measurements is a basic activity in software development processes
  - The set of user evaluation tools is built and refined during the process and is used iteratively as a complement to the process and product measures

- Roles
  - Different roles are defined to support software development environments
  - The UCD roles, like for example the *Evaluation Manager* role are defined
The User Role in Software Development Environment (Example Use Cases)

- Users involvement
  - The customer sets high priority for the following task.
    ‘Explore who are the kinds of users who should use the product that we develop; what are their characteristics; what are their needs; what are their expectations from the product.’
  - The project manager reviews the subjects for the coming reflection session, and sees that one of the subjects is ‘ways to assess the usability of our product’.
The User Role in Software Development Environment (Example Use Cases)

- User evaluation
  - The team leader browses over the details of the user experiment that is planned for tomorrow.
  - One of the teammates sees that the *User Perspective* flushes meaning new data has arrived. He clicks on it and sees that the results of the user experiment that was conducted yesterday are in. He is surprised to find a new problem with high severity ranking.
The User Role in Software Development Environment (Example Use Cases)

- Design Improvement
  - The designer of the user interface views the latest design diagrams and tries different changes that adhere to the new task in this iteration.
  - One of the teammates browses over the system reports and sees for each user experiment, which was conducted in the last two releases, what were the results and what were the implications on design. For each implication, he sees the development tasks that are related.
The UCD Management Eclipse Plug-in (Requirements)

- End-to-end UCD experiments:
  - Experiments can be defined (date and time, assign users and teammates, store description, results, and conclusions)
  - Experiments can be executed thus collecting data from User Experience (UX) according to the experiment
  - Experiments can be viewed as per the results achieved
  - Each group should select two kinds of experiments for implementations

- Evaluation manager role-perspective:
  - The evaluation manager role-perspective supports the management of the different experiments
  - Experiments can be in different states
  - The evaluation manager role-perspective supports
    - Status view of the experiments
    - view of the current work items with respect to the experiments
    - UX results view
The UCD Management Eclipse Plug-in (Requirements)

- UI designer role-perspective:
  - The designer role-perspective supports the UX-refactoring tasks (which are a special kind of work items) that emerge from the UX results
  - A UX-refactoring task is associated with one or more UX results
  - The UX results view reflects the status of the UX-refactoring tasks for example if a specific task is ‘done’ or ‘in progress’ it is marked in the UX results view

- Work items can be created and assigned.

- The system has one repository for its data.

- A set of classes to support automatic user evaluation
This template defines and organizes the way to take interviews of the real users of the system.

Domains: Interview

<table>
<thead>
<tr>
<th>Purpose</th>
</tr>
</thead>
</table>

| Relationships |

<table>
<thead>
<tr>
<th>Roles</th>
<th>Responsible</th>
<th>Modified By</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Interviewer</td>
<td></td>
</tr>
</tbody>
</table>

| Description |

| Illustrations |

<table>
<thead>
<tr>
<th>Templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users' Interview Template</td>
</tr>
</tbody>
</table>

| Key Considerations |

| More Information |

<table>
<thead>
<tr>
<th>Checklists</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interview Checklist</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Guidelines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users' Interview Guideline</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supporting Materials</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users' Interview Supporting Material</td>
</tr>
</tbody>
</table>
The role "Design Evaluator" is the person who evaluates the proposed design of the system against a set of credentials according to the current phase.

Role Set: Design Evaluators

- **Checklists**
  - Evaluation Checklist

- **Guidelines**
  - Evaluation Guideline

- **Supporting Materials**
  - Evaluation Supporting Material
High-level Design: Experiment Attributes
High-level Design: Evaluation Manager Perspective
Summary

- We present an UCD management plug-in to better support UCD activities in software development processes.
- We base the capabilities suggested on use cases that were emerged when performing UCD activities with agile teams.
- Future directions:
  - continue developing the plug-in to support the use cases that are emerged, thus achieving a complete set of refined requirements for UCD management.
  - evaluate the plug-in with software development teams.
Agenda

- The agile approach for software development
  - Customer collaboration
  - Exhaustive testing
- Research
  - Involvement of users in the process of software development
  - Governance of software development
  - Measured test driven development
- Analysis framework
Enterprise in Transition: Governance Meets Agile

Yael Dubinsky, Avi Yaeli, Yishai Feldman
{dubinsky,aviy,yishai}@il.ibm.com
IBM Haifa Research Lab
To govern is to steer, to direct

A commonly used definition within IBM

- Establishing chains of responsibility, authority and communications to empower people to make decisions
- Establishing policy, control and measurement mechanisms to enable people to carry out their roles and responsibilities

The ultimate goal of governance is to ensure that the organization successfully reaches its strategy and goals

Software Development Governance: transition to agile

- **Context**
  - Agile becomes main stream
  - A ‘Transition to Agile’ processes take place – involves conceptual and practical changes

- **Goals**
  - Study the ‘Transition to Agile’ from a governance perspective
    - Evolution of the governance throughout its lifecycle
    - The governance artifacts, their implementation, documentation and evolution with each iteration
  - Identify patterns and tooling capabilities that could help to support governance processes in Agile

- **Research method**
  - Field research – accompanying transition processes
    - Participate in business days every two weeks; interview key roles along the process; guide a retrospective process; assist in refining a measures set; consult to higher management in adopting agile into the work procedures.
A Governance Model
A Governance Model
Governance Lifecycle

Estimation vs. Actual Time

Iteration 2

Iteration 4
Estimation vs. Actual Time
(Summary of 4 iterations)
The Governance Perspective

- **Governance direction**
  - Usually top-down, here bottom-up
  - Need appropriate governance mechanisms

- **Goals and measures**
  - Incompatible between agile team and rest of enterprise
  - Need a way to integrate governance structures

- **Governance as framework for scalability**
  - Need to formalize governance mechanisms to help the process of transition to agile

- **Governance control loop**
  - An essential part of governance
Summary

- Towards a Governance Tool
  - Governance Editor
    - E.g., Manipulate the entities
  - Governance Engine
    - E.g., calculate the compliance
  - Governance Dashboard
    - E.g., role-based views
Agenda

- The agile approach for software development
  - Customer collaboration
  - Exhaustive testing

- Research
  - Involvement of users in the process of software development
  - Governance of software development
  - Measured test driven development

- Analysis framework
Test Driven Development (TDD)

- The goal is clean code that works
- The TDD actions are:
  - Write new code only if an automated test has failed
  - Eliminate duplication

Test Driven Development (TDD)

- The TDD mantra: red/green/refactor
  - **Red** - Write a little test that doesn't work (or compile)
  - **Green** - write the minimum and simplest code that makes the test run
  - **Refactor** - Eliminate duplication created in merely getting the test to work

- Small-steps iterative manner

Why TDD?

- Not enough time to test
- Testing provides negative feedback
- Responsibility of testing is transferred
- Testing is a low status job
- Testing is hard to manage
- Testing is hard

Problem: difficult to introduce
TDD Steps

![Graph showing TDD Steps and average across functions.]

- Number of TDD Steps
- Average
Reasons for TDD Steps

- Checking function/class existence
- Checking parameters correctness
- Checking exceptional Cases
- Checking a part of a feature
- Checking input
- Checking number of elements
- Checking code improvements

Number of TDD Transitions
Lessons Learned

- TDD steps are a useful unit to explore the TDD behavior patterns
- Working in TDD steps, participants produced unit tests but did not refactor while in the process
- A tighter procedure of work is required
The TDD mantra: red/green/refactor

- Red - Write a little test that doesn't work (or compile)
- Green - write the minimum and simplest code that makes the test run
- Refactor - Eliminate duplication created in merely getting the test to work

Measure Size and Complexity

Small-steps iterative manner
Measured TDD
Measured TDD

![Graph showing cyclomatic complexity for Measured TDD steps per task.](image)

- **Axes:**
  - Y-axis: Cyclomatic complexity
  - X-axis: Measured TDD steps per task

- **Legend:**
  - Step 1
  - Step 2
  - Step 3
  - Step 4
  - Step 5
  - Step 6

- **Data Points:**
  - Step 1: Low complexity for most steps.
  - Step 2: Moderate to high complexity across steps.
  - Step 3: Consistently low complexity.
  - Step 4: Moderate to high complexity across steps.
  - Step 5: Variability in complexity across steps.
  - Step 6: Low complexity for most steps.
Summary & Lessons Learned

- Average of number of steps - 4.16 vs. 4.66; low levels of complexity -> most participants selected simple functions

- Asking specifically to relate to the measures outcomes, increases refactoring
Summary & Lessons Learned

- Specific task for example:

- Number of Lines vs. Measured TDD steps
- Cyclomatic complexity vs. Measured TDD steps
Agenda

- The agile approach for software development
  - Customer collaboration
  - Exhaustive testing
- Research
  - Involvement of users in the process of software development
  - Governance of software development
  - Measured test driven development
- Analysis framework
Human Organizational Technological Analysis Framework

- **Human**
  - cognitive and social aspects
  - refers to learning and interpersonal processes

- **Organizational**
  - managerial and cultural aspects
  - refers to software project management and control

- **Technological**
  - practical and technical aspects
  - refers to design, testing, and coding, as well as to integration, delivery, and maintenance of software products