Agile Software Development

L3 – Requirements management in ASD

Ezequiel Scott
ezequiel.scott@ut.ee
Agenda

• Recap
• Requirements Management in ASD
• Tools to support requirements in ASD
Agenda

• Recap

• Requirements Management in ASD

• Tools to support requirements in ASD
Recap

• An introduction to software development processes
• Agile terminology
  • Mindset, Values, Practices, Methods
• Current state of Agile worldwide
  • Most used practices and methods
  • Hybrid processes
• Test-driven Development (TDD)
Testing levels

<table>
<thead>
<tr>
<th>Level</th>
<th>Definition and Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptance Testing (AT)</td>
<td>The level of the software testing process where a system is tested for acceptability. The purpose of AT is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.</td>
</tr>
<tr>
<td>System Testing (ST)</td>
<td>The level of the software testing process where a complete, integrated system/software is tested. The purpose of ST is to evaluate the system's compliance with the specified requirements.</td>
</tr>
<tr>
<td>Integration Testing (IT)</td>
<td>The level of the software testing process where individual units are combined and tested as a group. The purpose of IT is to expose faults in the interaction between integrated units.</td>
</tr>
<tr>
<td>Unit Testing (UT)</td>
<td>The level of the software testing process where individual units/components of a software/system are tested. The purpose of UT is to validate that each unit of the software performs as designed.</td>
</tr>
</tbody>
</table>
TDD Workflow (in short)

1. Write a failing test
2. Make the test pass
3. Refactor
TDD workflow

1. write a “single” unit test describing an aspect of the program

2. run the test, which should fail because the program lacks that feature

3. write “just enough” code, the simplest possible, to make the test pass

4. “refactor” the code until it conforms to the simplicity criteria

5. repeat, “accumulating” unit tests over time
About the practice session:

What’s your progress on handout #1?

A. I completed the “Palindrome” task
B. I completed the “DNA analysis” task
C. Both tasks done
D. None
Do you have experience with TDD? (other than tutorials or course exercises)

A. I use it everyday
B. I used it few times
C. None
What’s your first feeling about TDD?
(leave your comment)

- Is it difficult?
- Would you use it? Not sure?
Does TDD work?
Benefits of using TDD

✓ The code will remain well **factored** and **testable** (higher coverage and lower coupling between objects) [2]

✓ TDD leads to more **maintainable code** since code is in a maintenance mode from the beginning

✓ TDD contributes to the **quality of the code** [1]

✓ TDD can lead to more **modularized, flexible, and extensible code** [2]


Limitations of TDD

- TDD does not seem to affect commit velocity and the number of bug fixing commits [3]:
- The correlation between TDD and productivity is inconclusive [1]
- Further evidence is necessary to conclude whether TDD is better or worse than Incremental Test Last Development (ITLD) in industrial settings [4, 7]
- Writing and maintaining an excessive number of tests costs time
- TDD is not suitable to any organizational context [6, 7] or technology
- Difficulties [7]:
  - increased development time
  - insufficient TDD experience/knowledge
  - domain and tool specific issues
  - lack of developer skill in writing test cases
  - insufficient adherence to TDD protocol
  - legacy code.

References:

Myths and Misconceptions

• You create a 100% regression test suite
  • Reusable components/frameworks and user interfaces are not usually covered by the tests

• The unit tests form 100% of your design specification
  • Design is much more than unit tests

• You only need to unit test
  • Complex systems need of other testing techniques

• TDD is sufficient for testing
  • TDD is only part of your overall testing efforts

• You do not write requirements (*)
  • You should write a to-do list (isn’t it a requirement list?)
Agenda

• Recap

• Requirements Management in ASD

• Tools to support requirements in ASD
User Stories Applied

Requirements in SE

• Software requirements is a **communication problem**

• When the **business** side dominates...
  • it mandates unrealistic functionality and dates
  • it ignores if developers understand exactly what is needed

• When the **developers** dominate the communications...
  • technical jargon replaces the language of the business
  • the developers lose the opportunity to learn what is needed by listening
Requirements – Use Cases

• **Use case analysis** is a technique used to identify the requirements of a system.

• Use case-driven development is a key characteristic of many process models and frameworks [...] With its inherent iterative, incremental, and evolutionary nature, use case also fits well for **agile development** *

Use Cases – Example

More info:

- Systems Modelling (MTAT.03.083)

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan
Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

Pssst! Did you compare the 4 values and 12 principles with your own Manifesto?
What Is a User Story?

• A user story describes **functionality** that will be **valuable** to either a user or purchaser of a software.

• User stories are composed of three aspects:
  • a written description, usually as a **Card**, of the story used for planning and as a reminder
  • **Conversations** about the story that serve to flesh out the details of the story
  • tests/document details that can be used to **Confirm** when a story is complete

• **Card, Conversation, Confirmation** (Jeffries 2001)
User Story Format

As a [role]
I want [what]
so that [benefit]

Example

As a bank customer
I want to check the strength
of my password
so that I don’t get hacked
easily
Role identification

- Most project teams consider only a single type of user → This leads to software that ignores the needs of at least some user types
- Identify the different **user roles** who will interact with the software
- Some user roles benefit from being described by **personas** → A persona is an imaginary representation of a user role
- **Brainstorm, Organize, and Consolidate** the roles
Brainstorm

• Identify the **customer**. If not available, find a **proxy**
• **Team-based** activity. If not available, find a **team customer**
• Brainstorming!

**User Role: Internal Recruiter**

Not particularly computer-savvy but quite adept at using the Web. Will use the software infrequently but intensely. Will read ads from other companies to figure out how to best word her ads. Ease of use is important, but more importantly what she learns must be easily recalled months later.
Organize

Figure 3.1 Organizing the user role cards on a table.
Figure 3.2 The consolidated role cards.
Example

As a bank customer
I want to check the strength
of my password
so that I don’t get hacked
easily
Where are the details?

• Many details can be expressed as additional stories
• It is better to have more stories than large stories
• User Story splitting

As a user I want to search for a job
Where are the details?

As a user I want to view information about each job that is matched by a search.

A user can view detailed information about a company that has posted a job.

A user can view the job description.

A user can view a job's salary range.

A user can view the location of a job.
Where are the details?

As a user I want to view information about each job that is matched by a search.

A user can view detailed information about a company that has posted a job.

Users can view information about each job that is matched by a search.

*Marco says show description, salary, and location.*

Use notes/constraints instead!
Non-functional requirements (NFR)

- NFR can be considered as **constraints** on the system’s behaviour
- “The system shall be written in Java”
- **Quality Attributes** or –ilities (usability, availability, performance, ...)

Table 16.1 *Sample constraints written for a variety of common nonfunctional requirements.*

<table>
<thead>
<tr>
<th>Area</th>
<th>Sample Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>80% of database searches will return results to the screen in less than two seconds.</td>
</tr>
<tr>
<td>Accuracy</td>
<td>The software will correctly predict the winner of a football game at least 55% of the time.</td>
</tr>
<tr>
<td>Portability</td>
<td>The system shall not make use of any technology that would make it difficult to port to Linux.</td>
</tr>
<tr>
<td>Reusability</td>
<td>The database and database access code will be reusable in future applications.</td>
</tr>
</tbody>
</table>
Themes, Epics, Stories, Tasks
**Quality criteria: INVEST**

<table>
<thead>
<tr>
<th>INVEST by Bill Wake</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Independent</strong></td>
</tr>
<tr>
<td>- Of order of user story delivery</td>
</tr>
<tr>
<td>- Of internal and especially external dependencies</td>
</tr>
<tr>
<td><strong>Negotiable</strong></td>
</tr>
<tr>
<td>- Flexible scope</td>
</tr>
<tr>
<td>- None specific language</td>
</tr>
<tr>
<td>- Explain the intention, not the implementation</td>
</tr>
<tr>
<td><strong>Valuable</strong></td>
</tr>
<tr>
<td>- Value is clear to everyone</td>
</tr>
<tr>
<td>- Persona matches Benefit &amp; Goal will deliver the benefit.</td>
</tr>
<tr>
<td>- Avoid technical / role specific language</td>
</tr>
<tr>
<td><strong>Estimatable</strong></td>
</tr>
<tr>
<td>- Clear and concise explanation</td>
</tr>
<tr>
<td>- Avoid technical / role specific language</td>
</tr>
<tr>
<td><strong>Small</strong></td>
</tr>
<tr>
<td>- Easily fits into a Sprint. i.e. &lt; 20% of velocity.</td>
</tr>
<tr>
<td>- Definitely not &gt; 33% of velocity</td>
</tr>
<tr>
<td><strong>Testable</strong></td>
</tr>
<tr>
<td>- Can be automated</td>
</tr>
<tr>
<td>- Avoid external testing / long test suites</td>
</tr>
</tbody>
</table>
User Story Splitting patterns

1. Prepare the input story
   • Check INVEST
   • Check the size

2. Apply the splitting patterns
   • Operations, Data, Workflow steps...

3. Evaluate the split
   • Are the new stories equal in size?

https://agileforall.com/patterns-for-splitting-user-stories/
HOW TO SPLIT A USER STORY

1. PREPARE THE INPUT STORY
   - Does the big story satisfy INVEST* (except, perhaps, small)?
     - NO: Combine it with another story or otherwise reformulate it to get a good, if large, starting story.
     - YES: Is the story size 10% to 20% of your velocity?
       - NO: You’re done.
       - YES: Continue. You need to split it.

2. APPLY THE SPLITTING PATTERNS
   - Can you split the story so you do the beginning and end of the workflow first and enhance with stories from the middle of the workflow?
   - Can you take a thin slice through the workflow first and enhance it with more stories later?
   - Can you split the story to just make it work first and then enhance it to satisfy the non-functional requirements?
   - Does the story describe a workflow?
   - Does the story describe a workflow?
   - Does the story describe a workflow?

3. EVALUATE THE SPLIT
   - Are the new stories roughly equal in size?
     - YES: Try another pattern on the original story or the larger post-split stories.
     - NO: Try another pattern.
   - Do each of the stories satisfy INVEST?
     - YES: Try another pattern.
     - NO: Are there stories you can de-prioritize or delete?
   - Is there an obvious story to start with that gets you early value, learning, risk mitigation, etc.?
     - YES: You’re done, though you could try another pattern to see if it works better.
     - NO: Try another pattern to see if you can get this.

* INVEST - Stories should be:
  - IN - Independent
  - N - Negotiable
  - E - Estimatable
  - V - Valuable
  - S - Small
  - T - Testable

** Visit http://www.richardlawrence.info/splitting-user-stories/ for more info on the story splitting patterns

Copyright © 2011-2018 Agile for All. All rights reserved.
Last updated 2/3/2018

https://agileforall.com/patterns-for-splitting-user-stories/
Organizing User Stories

How to envisage the entire product or service as a series of tasks which the user completes?

Two mechanisms:

• Kanban board
• User Story Mapping
Organizing User Stories

How to envisage the entire product or service as a series of tasks which the user completes?

Two mechanisms:

• **Kanban board**

• **User Story Mapping**
## Kanban Board

- It visualizes work and the process it goes through
- Generally more sophisticated than “mere” task boards

![Kanban Board Diagram]

<table>
<thead>
<tr>
<th>Backlog</th>
<th>To Do</th>
<th>In Progress</th>
<th>Testing</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 hrs HIGH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 hrs Low</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 hrs HIGH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bug Fix</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 hrs Medium</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 hrs Medium</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Organizing User Stories

How to envisage the entire product or service as a series of tasks which the user completes?

Two mechanisms:

• Kanban

• User Story Mapping
Story Mapping

- **Story mapping** consists of ordering user stories along two independent dimensions
  - horizontal axis $\rightarrow$ **order of priority** (or “the order in which you would describe activities to explain the behaviour of the system”)
  - vertical axis $\rightarrow$ it represents increasing sophistication of the **implementation**

- The first horizontal row represents a “**walking skeleton**”, a barebones but usable version of the product
- Working through successive rows fleshes out the product with **additional functionality**
User Story Map in 7 steps

1. Frame the journey
2. Build your story backbone
3. Identify and group activities
4. Break large tasks into subtasks
5. Fill in the blanks
6. Prioritize tasks and subtasks (but leave your backbone as is)
7. “Slice” groups of tasks into iterations

https://plan.io/blog/user-story-mapping/
User Story Mapping

https://plan.io/blog/user-story-mapping/
User Story Mapping

https://plan.io/blog/user-story-mapping/
User Story Mapping

https://plan.io/blog/user-story-mapping/

The **backbone** is also the **narrative flow**!
User Story Mapping

https://plan.io/blog/user-story-mapping/

The backbone is also the narrative flow!

Priority +

Priority -

Time

(EPICS)

(USER STORIES)

The backbone is also the narrative flow!
User Story Mapping

https://plan.io/blog/user-story-mapping/

The backbone is also the narrative flow!

Priority +

Priority -

Time

(EPICS)

(USER STORIES)

Details

Activities

Narrative Flow

(EPICS)

(User Stories)
User Story Mapping

https://plan.io/blog/user-story-mapping/

The **backbone** is also the **narrative flow**!
User Story Mapping
https://plan.io/blog/user-story-mapping/

The backbone is also the narrative flow!
User Story Mapping

https://plan.io/blog/user-story-mapping/

The **backbone** is also the **narrative flow**!

**Priority +**

**Priority -**

**Time**

**Activities (EPICS)**

**Narrative Flow (USER STORIES)**

**Details**

**User Tasks**

**User**

**Backbone**

**Release Slice**
User Story Mapping: Example

Organize Email
- Search Email
- File Emails
- Compose Email
- Read Email
- Delete Email
- View Calendar
- Manage Calendar
- Create Appt
- Update Appt
- View Appt
- Create Contact
- Update Contact
- Delete Contact

Manage Email
- Search by Keyword
- Move Emails
- Create and send basic email
- Open basic email
- Delete email
- View lists of appts
- Create basic appt
- Update contents/location
- View Appt

Manage Calendar
- View Monthly formats
- Create RTF appt
- Accept/Reject/Tentative
- View Contact info

Manage Contacts
- Limit search to one field
- Set email priority
- Open Attachments
- Mandatory/Optional

Release 1
- Send HTML email
- Open HTML email
- Empty Deleted Items
- View Daily Format
- Create HTML appt
- Propose new time
- Add address data
- Update Address Info
- Delete Contact

Limit search to 1+ fields

Release 2
- Search attachments
- Get address from contacts
- View Weekly Formats
- Get address from contacts
- View Attachments
- Import Contacts
- Export Contacts

Search sub folders
- Search Calendar
- Add Attachments

Example story map created by Steve Rogalsky
http://winnipegagilist.blogspot.com

Release 3
- Send RTF email
- Open RTF email
- View Monthly formats
- Create RTF appt
- Accept/Reject/Tentative
- View Contact info
Releases

• A release is made up of one or more iterations

• Release planning refers to determining a balance between a projected timeline and a desired set of functionality

• Iteration planning refers to selecting stories for inclusion in this iteration

• The customer team and the developers are both involved in release and iteration planning
Team activity

Your team has been hired to add new features to a social network

1. Choose your team’s favourite social network
2. One of you must play the role of Product Owner (PO). Others can play different roles (e.g. Engineering, Sales, Marketing, Finance...)
3. Agree on some epics to add (3 should be enough)
4. Create a user story mapping to plan at least the next two releases (https://miro.com/app/board/o9JkzXoM_8=/?fromEmbed=1)
5. Identify gaps, dependencies, technical requirements, and alternatives.
6. Share your results on Moodle!