Agile Software Development

L03 – 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</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</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
**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
Does TDD work?
Benefits of using TDD

✓ The code will remain well **factored** and **testable** (higher coverage and lower coupling between objects and) [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 ITLD in an **industrial setting** [4, 7]
- Writing and maintaining an **excessive number of tests** costs time
- TDD is not suitable to any **organizational context** [6, 7] or **technology** (e.g. GUI-design)

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

Mike Cohn (2004)
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** *

(*) https://en.wikipedia.org/wiki/Use_case
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
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 search for a job

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

A user can view the job description

A user can view a job’s salary range

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

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

As a user I want to search for a job

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

Theme
Increase Website Traffic

Epic
Add new Video Section
User Story
User Story
Task
Task

Epic
Improve Login Page Usability
User Story
As a User, I would like the validation on the login page to be very clear so that I can easily see when/if I make a mistake when I log in
Task
Task
Task
Task
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)?
   - Combine it with another story or otherwise reformulate it to get a good, if large, starting story.
   - Is the story size %10 to %90 of your velocity?
   - You’re done. Continue. You need to split it.

2. APPLY THE SPLITTING PATTERNS
   - Could you split the story to do that simple core first and enhance it with later stories?
   - Could you group the later stories and defer the decision about which story comes first?
   - Does the story get the same kind of data via multiple interfaces?
   - Does the story have a complex interface?
   - When you apply the obvious split, is whatever story you do first the most difficult?
   - Does the story do the same thing to different kinds of data?
   - Does the story have a simple core that provides most of the value and/or learning?
   - Does the story have a simple version you could do first?

3. EVALUATE THE SPLIT
   - Are the new stories roughly equal in size?
   - Is each story about 40-50% of your velocity?
   - Do each of the stories satisfy INVEST?
   - Are there stories you can prioritize or defer?
   - Is there an obvious story to start with that gets you early value, learning, risk mitigation, etc?
   - You’re done, though you could try another pattern to see if it works better.

WORKFLOW STEPS
   - 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 split the operations into separate stories?
   - Does the story describe a workflow?
   - Can you take a thin slice through the workflow first and enhance it with more stories later?

OPERATIONS
   - Can you split the story so you do a subset of the rules first and enhance with additional rules later?
   - Does the story have a variety of business rules (e.g., is there a domain term in the story like “flexible dates” that suggests several variations?)
   - Does the story have multiple operations (e.g., is it about “managing” or “configuring” something?)

BUSINESS RULE VARIATIONS
   - Are the new stories sufficiently small?
   - Does each of the stories satisfy INVEST?
   - Are there stories you can prioritize or defer?
   - Is there an obvious story to start with that gets you early value, learning, risk mitigation, etc?

MAJOR EFFORT
   - Can you split the story to handle data from one interface first and enhance with the others later?

INTERFACE VARIATIONS
   - Can you find a small place you understand well enough to start?

BREAK OUT A SPIKE
   - Can you define the 1-3 questions most holding you back?

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

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

<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>Bug Fix</td>
<td>Research</td>
<td>Content</td>
<td></td>
</tr>
<tr>
<td>10 hrs</td>
<td>2 hrs</td>
<td>3 hrs</td>
<td>2 hrs</td>
<td></td>
</tr>
<tr>
<td>HIGH</td>
<td>Medium</td>
<td>Medium</td>
<td>HIGH</td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 hrs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>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</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIGH</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 → **order of priority** (or “the order in which you would describe activities to explain the behaviour of the system”)
  
  • vertical axis → 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**!
User Story Mapping

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

The backbone is also the narrative flow!

Priority +

Priority -

Time

(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**

**USER TASKS**

**ACTIVITIES**

**NARRATIVE FLOW**

**DETAILS**

**TIME**

**EPICS**

**USER STORIES**

Priority +

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

*The backbone is also the narrative flow!*

Priority +

Priority -

Time

**User**

**Backbone**

**User Tasks**

**Activities (Epics)**

**Narrative Flow (User Stories)**

**Release Slice**

**Details**
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
Agenda

• Recap
• Handling requirements in ASD
• Tools to support requirements
Paper alternative
Paper alternative
Software Tools – Issue trackers

https://trello.com/b/C4Awm5lK/kanban-board
Trello + Bitbucket

https://bitbucket.org/product/features/trello-boards
## Paper or Software?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Paper</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplicity</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Interaction and discussion</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Usability</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Certification</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Traceability</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Distributed/remote development</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>