Agile Software Development

L03 - Handling requirements in ASD

Ezequiel Scott
ezequiel.scott@ut.ee
Agenda

• Recap
• Handling requirements in ASD
• Tools to support requirements in ASD
Agenda

• Recap

• Handling requirements in ASD

• Tools to support requirements in ASD
Recap

• An introduction to software development processes
• Agile terminology
  • 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

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
Test-driven Development (TDD)

- TDD guarantees that code will remain well **factored** and **testable**
- TDD leads to more **maintainable code** since code is in a maintenance mode from the beginning
- TDD **does not seem to affect**:
  - commit velocity
  - number of bug fixing commits
  - numbers of issues
  - usage of TravisCI
  - numbers of pull requests

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.
Agenda

• Recap

• Handling requirements in ASD

• Tools to support requirements in ASD

• Iterative Software Development

• eXtreme Programming (XP)
User Stories Applied

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** *

Use Cases – Example

More info:

- Systems Modelling (MTAT.03.083)

(*) https://en.wikipedia.org/wiki/Use_case
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.
Consolidate

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 the location of a job

A user can view detailed information about a company that has posted 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 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>Quality Criteria</th>
<th>Definition</th>
</tr>
</thead>
</table>
| **Independent** | Of order of user story delivery  
|                 | Of internal and especially external dependencies |
| **Negotiable**  | Flexible scope  
|                 | None specific language  
|                 | Explain the intention, not the implementation |
| **Valuable**    | Value is clear to everyone  
|                 | Persona matches Benefit & Goal will deliver the benefit.  
|                 | Avoid technical / role specific language |
| **Estimatable** | Clear and concise explanation  
|                 | Avoid technical / role specific language |
| **Small**       | Easily fits into a Sprint. i.e. < 20% of velocity.  
|                 | Definitely not > 33% of velocity |
| **Testable**    | Can be automated  
|                 | Avoid external testing / long test suites |
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)?
     - YES
     - NO
   - Combine it with another story or otherwise reformulate it to get a good, if large, starting story.
   - Is the story size <= 10% of your velocity?
     - YES
     - NO
   - 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 other stories?
   - Could you group the later stories and defer the decision about which story comes first?
   - When you apply the obvious split, is whichever story you do first the most difficult?
   - Does the story have a simple core that provides most of the value and/or learning?
   - Does the story have a complex interface?
   - Does the story use the same kind of data?
   - Does the story use the same kind of data via multiple interfaces?
   - Can you split the story to handle data from one interface first and enhance with the others later?

3. EVALUATE THE SPLIT
   - Are the new stories roughly equal in size?
     - YES
     - NO
   - Is each story about 40 to 60% of your velocity?
     - YES
     - NO
   - Do each of the stories satisfy INVEST?
     - YES
     - NO
   - Are there stories you can dispose of or delete?
     - YES
     - NO
   - Is there an obvious story to start with that gets you early value, learning, risk mitigation, etc.?
     - YES
     - NO
   - You're done, though you could try another pattern to see if it works better.

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
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
Kanban Board

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

**Policy**
- Business case showing value, cost of delay, size estimate and design outline.
- Selection at Replenishment meeting chaired by Product Director.
- Small, well-understood, testable, agreed with PD & Team
- As per “Definition of Done” (see...)
- Risk assessed per Continuous Deployment policy (see...)

Agenda

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

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

https://bitbucket.org/product/features/trello-boards
JIRA

**Board**

**TO DO 29**
- Implement feedback collector
  - SMS-001
- Add NPS feedback to wallboard
  - SMS-011
- Add NPS feedback to email report
  - SMS-004
- Allow users to change between two tiers at the same price
  - SMS-008
- Apply a prorated discount to a user when they move from a low to a high priced tier
  - SMS-0012
- Extend the grace period to accounts with expired credit cards, where they have had at least two
  - FP-004

**IN PROGRESS 4**
- Force SSL on any page that contains account info
  - SMS-029
- Create subscription plans and discount codes in Stripe
  - SMS-036
- Add analytics to pricing page
  - SMS-039
- Add link to app usage (GA) in email report
  - SMS-036

**DONE 3**
- Automate collection of feedback for weekly email report
  - SMS-031
- Schedule weekly email report for Monday mornings to all staff
  - SMS-032
- Install SSL certificate
  - SMS-033

Create issue

Create column

Built-in filters
GitHub

https://guides.github.com/features/issues/
### 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>