Agile Software Development

Lecturer: Raman Ramsin

Lecture 6

Scrum: Requirements
Requirements in Scrum

- In sequential product development, requirements are
  - nonnegotiable,
  - detailed up front, and
  - meant to stand alone.

- In Scrum, the details of a requirement are
  - negotiated through conversations that happen continuously during development,
  - fleshed out just in time and just enough for the teams to start building functionality to support that requirement, and
  - Kept in placeholders called Product Backlog Items (PBIs).
Product Backlog Items (PBIs)

- A product backlog item represents desirable business value.

- PBIs are gradually refined:
  - Initially, the PBIs are large (representing large swaths of business value), with very little detail.
  - Over time, we flow PBIs through a series of **conversations** among the stakeholders, product owner, and development team, refining them into a collection of smaller, more detailed PBIs.
  - Eventually, a product backlog item is small and detailed enough to move into a sprint, where it will be designed, built, and tested.
    - Even during the sprint, more details will be exposed in conversations between the product owner and the development team.

- Scrum does not specify any standard format for PBIs; PBIs can be represented in: **User Stories**, Use Cases, or even custom formats.
Product Backlog Items (PBIs): Progressive Refinement

- When using Scrum, not all requirements have to be at the same level of detail at the same time.
  - Requirements that we’ll work on sooner will be smaller and more detailed than ones that we will not work on for some time.
  - We employ a strategy of progressive refinement to disaggregate large, lightly detailed requirements into a set of smaller, more detailed items.

[Rubin 2012]
**User Stories**

- User stories are a convenient format for expressing the desired business value for many types of product backlog items:
  - They are crafted in a way that makes them understandable to both business people and technical people.
  - They are structurally simple and provide a great placeholder for a conversation.
  - They can be written at various levels of granularity and are easy to progressively refine.

- User stories are very convenient for expressing **features**, however, they are not suitable for all types of PBIs.
  - A typical example is using user stories for representing defects; a simple description of the defect would be preferable.
    - Example: “As a customer I would like the system to not corrupt the database.”
User Stories: The Three Cs

- User stories have been described as **the three Cs: Card, Conversation, and Confirmation**.
  - **Card**: People originally wrote (and many still do) user stories in a certain format on 3 × 5-inch index cards or sticky notes.
  - **Conversation**: Details of requirements are exposed and communicated in ongoing conversations among the development team, product owner, and stakeholders.
  - **Confirmation**: A user story also contains conditions of satisfaction; these are in fact acceptance criteria that clarify the desired behavior.
User Stories: Card

- A common template format for writing user stories is to specify:
  - a class of users (the user role),
  - what that class of users wants to achieve (the goal), and
  - why the users want to achieve the goal (the benefit).

  - The “benefit” part of a user story is optional, but unless the purpose of the story is completely obvious to everyone, it should be included.

- The card is not intended to capture all of the information that makes up the requirement.
  - We deliberately use small cards with limited space to promote brevity.
  - A card should hold a few sentences that capture the essence or intent of a requirement.
User Stories: Card

**User Story Title**

As a <user role> I want to <goal> so that <benefit>.

**Find Reviews Near Address**

As a typical user I want to see unbiased reviews of a restaurant near an address so that I can decide where to go for dinner.

[Rubin 2012]
User Stories: Conversation

- The details of a user story are exposed and communicated in a conversation among the development team, product owner, and stakeholders.
  - Conversation is typically not a one-time event, but rather an ongoing dialogue through the development effort.
  - Conversations enable the exchange of information and collaboration to ensure that the correct requirements are expressed and understood by everyone.
  - Although conversations are largely verbal, they can be supplemented with documents; e.g., they may lead to a UI sketch, or details of other system aspects.

Johnson Visualization of MRI Data

As a radiologist I want to visualize MRI data using Dr. Johnson’s new algorithm.
For more details see the January 2007 issue of the Journal of Mathematics, pages 110-118.

[ Rubin 2012 ]
User Stories: Confirmation

- A user story also contains confirmation information in the form of conditions of satisfaction.
  - These are acceptance criteria which clarify the desired behavior, usually written on the back of the user-story card.
  - Used by the development team to better understand what to build and test, and by the product owner to confirm that the user story has been implemented to his satisfaction.
  - These conditions of satisfaction can be expressed as high-level acceptance tests. The acceptance tests associated with the story exist for several reasons:
    - They are a way to capture and communicate, from the product owner’s perspective, how to determine if the story has been implemented correctly.
    - They can be a helpful way to create initial stories and refine them as more details become known, an approach called specification by example or acceptance-test-driven development (ATTD).
      - By elaborating on specific examples, we can drive the story creation and refinement process and have (automated) acceptance tests for each story.
User Stories: Confirmation

Upload File
As a wiki user I want to upload a file to the wiki so that I can share it with my colleagues.

Conditions of Satisfaction
- Verify with .txt and .doc files
- Verify with .jpg, .gif, and .png files
- Verify with .mp4 files ≤ 1 GB
- Verify no DRM-restricted files

<table>
<thead>
<tr>
<th>Size</th>
<th>Valid()</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>True</td>
</tr>
<tr>
<td>1,073,741,824</td>
<td>True</td>
</tr>
<tr>
<td>1,073,741,825</td>
<td>False</td>
</tr>
</tbody>
</table>

[Rubin 2012]
User Stories: Levels of Detail

- User stories can be specified at three levels of abstraction:
  - **Epics**: The largest type of stories, which are a few to many months in size and might span an entire release or multiple releases.
    - Helpful because gives a big-picture, high-level overview of what is desired.
    - Never moved into a sprint because it is way too big and not very detailed.
  - **Themes (Features)**: Medium-size stories, which are often on the order of weeks in size and therefore too big for a single sprint.
  - **Sprintable (Implementable) Stories**: The smallest user stories, which are on the order of days in size and therefore small enough to fit into a sprint and be implemented. Sometimes referred to as just *stories*.

- **Tasks** are the layer below stories, typically worked on by only one person, or perhaps a pair of people (on the order of hours).
  - Tasks are not user stories; they specify *how* instead of *what*. 
User Stories: Levels of Detail

Preference Training Epic
As a typical user I want to train the system on what types of product and service reviews I prefer so it will know what characteristics to use when filtering reviews on my behalf.

Keyword Training Theme
As a typical user I want to train the system on what keywords to use when filtering reviews so I can filter by words that are important to me.

[Rubin 2012]
Agile Software Development – Lecture 6

User Stories: Evaluation Criteria (1)

- Six criteria are used for evaluating whether our stories are fit for their intended use or require additional work.

- Summarized by the acronym **INVEST**, these criteria state that user stories should be:

  1. **Independent**: As much as is practical, user stories should be independent or at least only loosely coupled with one another.
     - Stories that exhibit a high degree of interdependence complicate estimating, prioritizing, and planning.
     - When applying this criterion, the goal is not to eliminate all dependencies, but instead to write stories in a way that minimizes dependencies.

  2. **Negotiable**: The details of stories should be negotiable.
     - Good stories capture the essence of what business functionality is desired and why it is desired. However, they leave room to negotiate the details.
     - Negotiability helps everyone involved avoid the blame-game mentality.
User Stories: Evaluation Criteria (2)

- **INVEST** criteria (Continued):
  3. **Valuable:** Stories need to be valuable to a customer, user, or both.
     - Customers (choosers) select and pay for the product; users use the product.
     - All stories in the backlog must be valuable (worth investing in) from the product owner’s perspective, which represents the customers and users.
     - *Technical stories* which are valuable to the developers, but are of no obvious value to the customer/user, must be approved by the product owner.
     - Most technical issues should be addressed through *tasks*, not *stories*.

---

**Migrate to New Version of Oracle**

As a developer I want to migrate the system to work with the latest version of the Oracle DBMS so that we are not operating on a version that Oracle will soon retire.

**Automatic Builds**

As a developer I want the builds to automatically run when I check in code so that regression errors are detected when they are introduced.

[Rubin 2012]
User Stories: Evaluation Criteria (3)

- **INVEST** criteria (Continued):

4. **Estimatable:** Stories should be estimatable by the team that will design, build, and test them.
   - Estimates provide an indication of the size and therefore the effort and cost of the stories (bigger stories require more effort and therefore cost more).
   - If the team is not able to size a story, the story is either just too big or ambiguous to be sized, or the team does not have enough knowledge.
     - If it’s too big, the team will need to work with the product owner to break it into more manageable stories.
     - If the team lacks knowledge, some form of exploratory activity will be needed to acquire the information (such as prototyping).

5. **Sized Appropriately (Small):** Stories should be sized appropriately for when we plan to work on them.
   - Stories worked on in sprints should be small.
   - It’s OK to have epics/themes that will not be worked on in the near future. They should be broken down only when the time comes to work on them.
User Stories: Evaluation Criteria (4)

- **INVEST** criteria (Continued):
  
  6. **Testable**: Stories should be testable in a binary way—they either pass or fail their associated tests.

  - Being testable means having good acceptance criteria (related to the conditions of satisfaction) associated with the story (confirmation).

  - It may not always be necessary or possible to test a story; but the requirement is still valuable as it will drive the design.

    - Epic-size stories typically do not have tests, nor do they need them.
    
    - There might not be a practical way to test non-functional requirements, such as “As a user, I want the system to have 99.999% uptime.”
Non-functional Requirements

- Non-functional requirements represent system-level constraints, and can be written as user stories; but they can also be written in a different format if deemed appropriate.

<table>
<thead>
<tr>
<th>Internationalization</th>
<th>Web Browser Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>As a user I want an interface in English, a Romance language, and a complex language so that there is high statistical likelihood that it will work in all 70 required languages.</td>
<td>System must support IE8, IE9, Firefox 6, Firefox 7, Safari 5, and Chrome 15.</td>
</tr>
</tbody>
</table>

[Rubin 2012]
Non-functional Requirements: Importance

- Non-functional requirements are very important since they affect the design and testing of most or all stories in the product backlog.
  - For example, having a “Web Browser Support” non-functional requirement would be common on any website project.
    - When the team develops the website features, it must ensure that the site features work with all of the specified browsers.
- Each non-functional requirement is a prime target for inclusion in the team’s definition of done.
  - If “Web Browser Support” is included in the definition of done, the team will have to test any new features added in the sprint with all of the listed browsers.
    - If it does not work with all of them, the story is not done.
  - Teams must try to include as many of the non-functional requirements in their definitions of done as they possibly can, so that they are tested continuously.
Knowledge-Acquisition Stories

Sometimes we need to create a product backlog item that focuses on knowledge acquisition through exploration.

- Such exploration is known by many names: Prototype, proof of concept, experiment, study, spike, and so on.
- If the knowledge-acquisition story is a technical story, its business value has to be justifiable to the product owner.

  - The question for the Scrum team is whether the value of the acquired information exceeds the cost of getting it.
  - If it does not, a **fail-fast** strategy might be a better option (try something, get fast feedback, and rapidly inspect and adapt).

> Filtering Engine Architecture Eval
> As a developer I want to prototype two alternatives for the new filtering engine so that I know which is a better long-term choice.

> Conditions of Satisfaction
> Run speed test on both prototypes.
> Run scale test on both prototypes.
> Run type test on both prototypes.
> Write short memo describing experiments, results, and recommendations.

[Rubin 2012]
Gathering Stories

- In Scrum, gathering user stories involves the users as part of the team that is determining what to build and is constantly reviewing what is being built.

- Two techniques are usually employed:
  - **User-story-writing workshops**: Used for generating at least the initial set of user stories through brainstorming.
  - **Story mapping**: Used to organize and provide a user-centered context to the stories.
Gathering Stories: User-Story-Writing Workshops

- The workshop frequently includes the product owner, Scrum Master, and development team, in conjunction with internal and external stakeholders.
- Most workshops last anywhere from a few hours to a few days.
- Some teams prefer to work top-down and others prefer to work bottom-up.
  - The top-down approach involves the team starting with a large story (like an epic) and then breaking it into smaller stories.
  - The bottom-up approach starts brainstorming the sprintable stories that are associated with the next release of the system.
Gathering Stories: Story Mapping

- The idea is to break a high-level user activity into a workflow that can be further decomposed into a set of detailed tasks.
  - At the highest level are the epics, representing the large activities.
  - Next we think about the sequence or common workflow of user tasks that make up the epic (represented by themes).
    - We lay out the themes along a timeline, where themes in the workflow that would occur sooner are positioned to the left of those that would occur later.
      - For example, the “Search for Product” theme would be to the left of the “Manage Shopping Cart” theme.
  - Each theme is then decomposed into a set of sprintable stories that are arranged vertically in order of priority.
    - Not all stories within a theme need to be included in the same release.
- Story mapping can be used as a complement to the story-writing workshop, in order to help visualize the prioritization of stories.
Gathering Stories: Story Map

[Rubin 2012]
References
