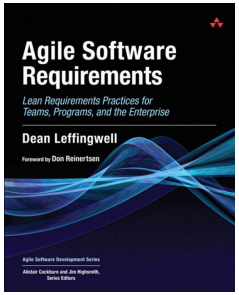
CE Department

Software Requirements Engineering

40688

These slides are designed to accompany Agile Software Requirements (2011) by Dean Leffingwell and support the university course Software Requirements Engineering, instructed by Mehran Rivadeh. Created and designed by Mahnaz Rasekhi.



Agile Software Requirements (2011)
Dean Leffingwell

User Stories_Part 1

Chapter 6

Mehran Rivadeh
mrivadeh@sharif.edu
Software Requirements Engineering
October 2025 - Fall 1404 - SUT

Contents

1. Introduction

a. User Story Overview

b. User Stories Build Bridges

c. User Stories Are Not Requirements

Introduction

➔ The user story is the workhorse of agile development, and it is the container that carries the value stream to the user. It also serves as a metaphor for our entire incremental value delivery approach, that is:

- Define a user value story
- Implement and test it in a short iteration
- Demonstrate and/or deliver it to the user
- Repeat forever

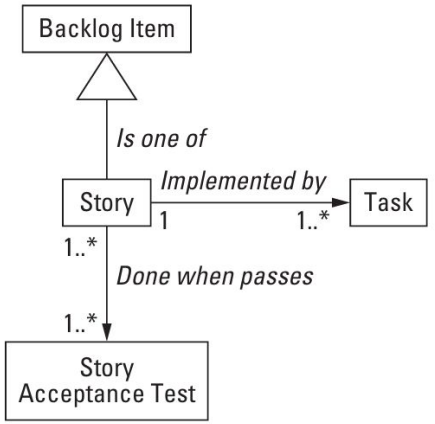


Figure 6–1 Requirements model for teams

User Story Overview

- The concept of the user story was first introduced by Extreme Programming (XP).
- XP practitioners played a key role in developing its breadth and depth.

  “The story is the **unit of functionality** in an XP project. We demonstrate progress by delivering tested, integrated code that implements a story. A story should be **understandable** to customers, developer, **testable**, **valuable** to the customer, and **small enough** that the **programmers can build** half a dozen **in** an **iteration**.”
  Beck and Fowler (2005)
- “A user story is **a brief statement of intent** that describes something the system needs to do for the user.”
  Mike Cohn(2004) - Active participant in the Scrum community.

User Story Overview

Who Writes The User Stories?

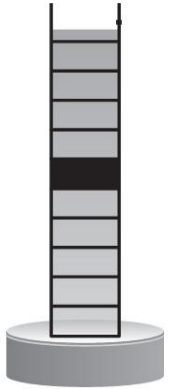
- In **XP**, user stories are often written by the **customer**, thus integrating the customer directly in the development process.
- In **Scrum**, the **product owner** often writes the user stories, with input from the customers, the stakeholders, and the team.
- In **actual practice**, **any team member** with **sufficient domain knowledge** can write user stories, but it is up to the product owner to accept and prioritize these potential stories into the product backlog.

User Story Overview

- A tool for defining a system's behavior in a way that is understandable to both the developers and the users.
- A lightweight and effective approach to managing requirements for a system.
- Captures a short statement of function on an index card or perhaps with an online tool.
- Details of system behavior do not appear in the brief statement,
  - These are left to be developed later through conversations and acceptance criteria between the team and the product owner.

User Story Overview

- In simple backlog form, stories can just be a list of things the system needs to do for the user.
- Example:
  - Log in to my web energy-monitoring portal
  - See my daily energy usage
  - Check my current electricity billing rate



User Stories Build Bridges

- User stories help bridge the Developer–Customer Communication Gap.
- In agile development, it is the **developer's job to speak the language of the user,** not the user's job to speak the language of developers.
- Effective communication is the key, and **we need a common language**.
- The **user story provides** the **common language** to build **understanding** between the user and the technical team.

As a <role>, I can <activity> so that <business value>.

User Stories Are Not Requirements

- They are not detailed requirements specifications (something a system shall do) but are rather **negotiable expressions** of intent (it needs to do something about like this).
- They are **short**, **easy to read**, and **understandable** to developers, stakeholders, and users.
- They represent **small increments** of **valued functionality** that can be developed **in a period** of days to weeks.

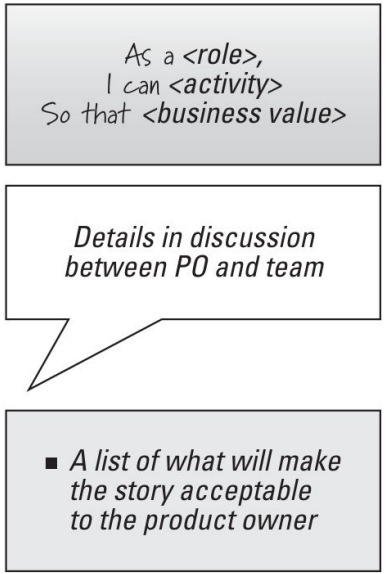
User Stories Are Not Requirements

- They are **relatively easy to estimate**, so effort to implement the functionality can be rapidly determined.
- They are **not carried** in **large**, **unwieldy documents** but rather organized in lists that can be more easily arranged and rearranged as new information is discovered.
- They need **little** or **no maintenance** and can be safely discarded after implementation
- **User stories**, and the **code** that is created quickly thereafter, **serve as inputs** to **documentation**, which is then developed incrementally as well.

Contents

2. User Story Form

User Story Form



- **Card** represents two to three sentences used to describe the intent of the story.
- **Conversation** represents a discussion between the team, customer, product owner, and other stakeholders, which is necessary to determine the more detailed behavior required to implement the intent.
- **Confirmation** represents the acceptance test, which is how the customer or product owner will confirm that the story has been implemented to their satisfaction

User Story Voice

- In the last few years, a newer, fairly standardized form has been applied that strengthens the user story construct significantly.

As a <role>, I can <activity> so that <business value>.

- **<role>**: represents who is performing the action or perhaps one who is receiving the value from the activity. It may even be another system, if that is what is initiating the activity.
- **<activity>**: represents the action to be performed by the system.
- **<business value>**: represents the value achieved by the activity.

User Story Detail

Example:

As a <role>, I can <activity> so that <business value>.

- ➔ As s consumer (<role>), I want to be able to see my daily energy usage (<what I do with the system>) so that I can lower my energy costs and usage (<business value I receive>).

User Story Detail

- The details for user stories are conveyed primarily through conversations between the product owner and the team, keeping the team involved from the outset.
- If more details are needed about the story, they can be provided in the form of
  - an attachment
  - mock-up
  - spreadsheet
  - algorithm, or whatever
- In addition to the statement of the user story, additional notes, assumptions, and acceptance criteria can be kept with a user story.

User Story Acceptance Criteria

Example (The Topic Of Chapter 10):

- **Story:**

  As s consumer (<role>), I want to be able to see my daily energy usage (<what I do with the system>) so that I can lower my energy costs and usage (<business value I receive>).

- **Acceptance Criteria:**

  - Read DecaWatt meter data every 10 seconds and display on portal in 15-minute increments and display in-home display every read.
  - Read KiloWatt meters for new data as available and display on portal every hour and on the in-home display after every read.
  - No multi day trending for noe (another story)

Contents

3. INVEST In Good User Stories

This part is covered in Chapter 6 – Part 2.

Contents

4. Splitting User Stories

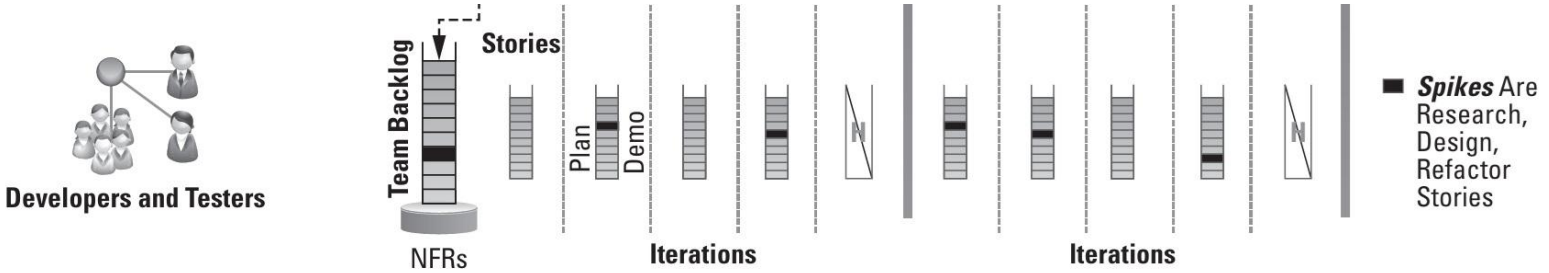
This part is covered in Chapter 6 – Part 3.

Contents

5. **Spikes**

Spikes

- ➔ Another invention of XP
- ➔ A special type of story used to drive out risk and uncertainty in a user story or other project facet.



Spikes

Spikes may be used for a number of reasons:

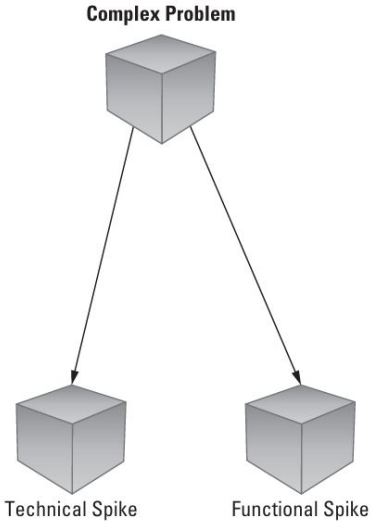
- For **basic research** to familiarize the team with a new technology or domain.
- The story may be too big to be estimated appropriately, and the team may use a spike **to analyze the implied behavior** so they can split the story into estimable pieces.
- The story may contain significant **technical risk**, and the team may have **to do some research** or **prototyping to gain confidence** in a **technological approach** that will allow them to commit the user story to some future timebox.
- The story may contain significant **functional risk**, in that although the intent of the story may be understood, **it's not clear how the system needs to interact with the user** to achieve the benefit implied.

Technical Spikes

Technical spikes are used **to research various technical approaches** in the **solution domain.**

- ➔ Example: A technical spike may be used to:
  - Determine a build-versus-buy decision
  - To evaluate potential performance or load impact of a new user story
  - To evaluate specific implementation technologies that can be applied to a solution, or for any reason

when the team needs to develop a more confident understanding of a desired approach before committing new functionality to a timebox.



Functional Spikes

- Functional spikes are used whenever there is **significant uncertainty** as to **how a user might interact with the system**.
- Functional spikes are often best evaluated through some level of prototyping, whether it be:
  - User interface mockups
  - Wireframes
  - Page Flows
  - or whatever techniques are best suited to get feedback from the customer or stakeholders.

Functional Spikes

Some user stories may require both types of spikes. Here's an example:

- As a consumer, I want to see my daily energy use in a histogram so that I can quickly understand my past, current, and projected energy consumption.
  - **Technical Spike:** Research how long it takes to update a customer display to current usage, determining communication requirements, bandwidth, and whether to push or pull the data.
  - **Functional Spike:** Prototype a histogram in the web portal and get some user feedback on presentation size, style, and charting attributes.

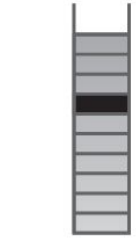
Guidelines For Spikes

- Since spikes do **not directly deliver** user **value**, they should be used sparingly and with caution.
- The following are some guidelines for applying user spikes.
  - Estimable, demonstrable, and acceptable
  - The exception, not the rule
  - Implement the spike in a separate iteration from the resulting stories

Guidelines For Spikes

Estimable, Demonstrable, And Acceptable

- ➔ Like other stories, spikes are put in the backlog, estimated, and sized to fit in an iteration.
- ➔ Spike results are different from a story, because they generally produce information, rather than working code.
- ➔ A spike may result in a decision, prototype, storyboard, proof of concept, or some other partial solution to help drive the final results.
- ➔ In any case, the spike should develop just the information sufficient to resolve the uncertainty in being able to identify and size the stories hidden beneath the spike.



■ ***Spikes*** Are Research, Design, Refactor Stories

Guidelines For Spikes

Estimable, Demonstrable, And Acceptable

- ➔ The output of a spike is demonstrable, both to the team and to any other stakeholders.
- ➔ This brings visibility to the research and architectural efforts.
- ➔ It also helps build collective ownership and shared responsibility for the key decisions that are being taken.
- ➔ And, like any other story, spikes are accepted by the product owner when the acceptance criteria for the spike have been fulfilled.

Guidelines For Spikes

The Exception, Not The Rule

- ➔ Every user story has uncertainty and risk—this is the nature of agile development.
- ➔ The team discovers the right solution through discussion, collaboration, experimentation, and negotiation.
- ➔ Thus, in one sense, every user story contains spike-level activities to flush out the technical and functional risk.
- ➔ The goal of an agile team is to learn how to embrace and effectively address this uncertainty in each iteration.
- ➔ A spike story, on the other hand, should be reserved for the more critical and larger unknowns.
- ➔ When considering a spike for future work, first consider ways to split the story through the strategies discussed earlier. Use a spike as a last option.

Guidelines For Spikes

Implement The Spike In A Separate Iteration From The Resulting Stories

- ➔ Since a spike represents uncertainty in one or more potential stories, planning for both the spike and the resultant stories in the same iteration is risky and should generally be avoided.
- ➔ However, if the spike is small and straightforward and a quick solution is likely to be found, there is nothing wrong with completing the stories in the same iteration. Just be careful.

Contents

5. Story Modeling With Index Cards

Story Modeling With Index Card

- ➔ Writing and modeling user stories using **physical** index **cards** provides a **powerful visual** and kinesthetic **means** for **engaging** the **entire team** in **backlog development**.
- ➔ This interactive approach has a number of **advantages**:
  - ◆ The physical size of index cards forces **a text length limit**, requiring the writer to articulate their ideas in just **a sentence or two**.
    - This helps **keep user stories small** and **focused**, which is a key attribute.
    - The tangible and physical nature of the cards gives teams the ability to **visually** and **spatially arrange them** in **various configurations** to help **define** the **backlog**.

Story Modeling With Index Card

- **Cards** may be **arranged** by **feature** (or epic) and may be written on the same colored cards as the feature for visual differentiation.
- Cards can also be **arranged** by **size** to help developers “see” the size relationships between different stories.
- Cards can be **arranged** by **time** or **iteration** to help evaluate dependencies, understand logical sequencing, see the impact on team velocity, and better align and communicate differing stakeholder priorities.
- **The more cards** you have, **the more work** you see, so scoping is a more natural process.

Story Modeling With Index Card

- Any team member can write a story card, and the physical act of moving these small, tangible “value objects” around the table creates an interactive learning setting where participants “see and touch” the value they are about to create for their stakeholders.
- Experience has shown that teams with a shared vision are more committed to implementing that vision.
- Modeling value delivery with physical story cards provides a natural engagement model for all team members and stakeholders—one that results in a shared, tangible vision for all to see and experience.

End of Chapter 6 _ Part 1

Contributions

- Author of Reference Book: **Dean Leffingwell**
- Course Instructor: **Mehran Rivadeh**
- Slide Creator: **Mahnaz Rasekhi**
  - These slides are primarily based on Agile Software Requirements by Dean Leffingwell, with occasional adaptations to enhance clarity and engagement.