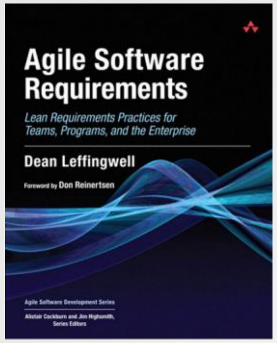


Agile Software Requirements

Lean Requirements Practices for Teams, Programs, and the Enterprise

Dean Leffingwell

Foreword by Don Reinertsen

Agile Software Development Series

Alistair Cockburn and Jim Highsmith, Series Editors

Agile Software Requirements

Software Requirements Engineering – 40688

Computer Engineering department

Sharif university of technology

Fall 402

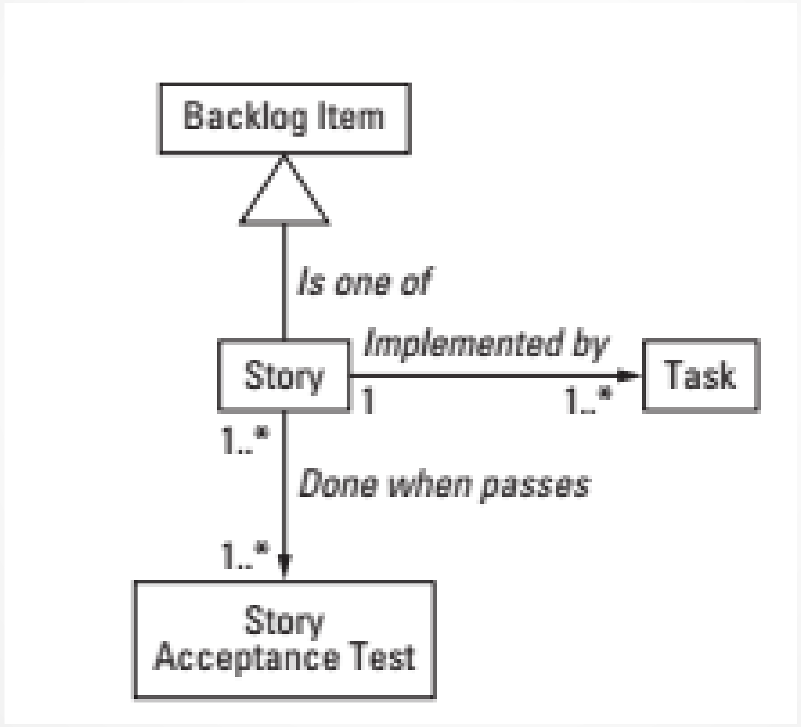
Chapter 6:

User Stories

Contents

- Introduction
  - User Stories are not requirement
- User Story Form
  - Card, Conversation, and Confirmation
  - User Story Voice
  - User Story Detail
  - User Story Acceptance Criteria
- Invest in Good User stories
- Splitting User Stories
- Spikes
- Story Modeling with Index Card
- Summary

User Story



Requirements model for teams

User Story Overview (1)

As Beck and Fowler explain:

- *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.*

User Story Overview (2)

- A user story is **a brief statement of intent** that describes something **the system needs to do for the user**.
- 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.

User Stories Are Not Requirements (1)

1) They are not detailed requirements specifications.
2) They are short, easy to read, and understandable to developers, stakeholders, and users.
3) They represent small increments of valued functionality that can be developed in a period of days to weeks.
4) They are relatively easy to estimate, so effort to implement the functionality can be rapidly determined.
5) 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.

User Stories Are Not Requirements (2)

5) They are not detailed at the outset of the project but are elaborated on a Justin-time basis, thereby avoiding too-early specificity, delays in development, requirements inventory, and an over-constrained statement of the solution.

6) They need little or no maintenance and can be safely discarded after implementation .

7) User stories, and the code that is created quickly thereafter, serve as inputs to documentation, which is then developed incrementally as well.

User Story Form (1)
Card, conversation, confirmation

1) **Card** represents 2 to 3 sentences used to describe the intent of the story.

2) **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.

3) **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 Form (2)
User Story Voice

As a 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 Form (3)
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.
- However, if **more details** are needed about the story, they can be provided in the form of an **attachment (mock-up**, **spreadsheet**, **algorithm**, **or whatever** which is attached to the user story).

User Story Form (4)
User Story Acceptance Criteria

- ***As a consumer, I want to be able to see my daily energy usage so that I can lower my energy costs and usage.***
- Acceptance Criteria:

✓ Read DecaWatt meter data every 10 seconds and display on portal in 15-minute increments and display on in-home display every read.

✓ Read Kilowatt meters for new data as available and display on the portal every hour and on the in-home display after every read.

✓ No multiday trending for now (another story).

INVEST in good user stories

1) **I**ndependent
2) **N**egotiable
3) **V**aluable
4) **E**stimable
5) **S**mall
6) **T**estable

The INVEST model is now fairly ubiquitous, and many agile teams evaluate their stories with respect to these attributes.

Independent (1)

Independence means that a story can be **developed**, **tested**, and potentially even **delivered on its own**. Therefore, it can also be independently *valued*.

An Example:

As an administrator*, I can set the consumer's password security rules so that users are required to create and retain secure passwords, keeping the system secure.*

As a consumer*, I am required to follow the password security rules set by the administrator so that I can maintain high security to my account.*

Independent (2)

- ***As an administrator***, *I can set the password expiration period so that users are forced to change their passwords periodically.*
- ***As an administrator***, *I can set the password strength characteristics so that users are required to create difficult-to-hack passwords.*

Valuable (1)

Think of a **whole story as a multi-layer cake**, e.g., a network layer, a persistence layer, a logic layer, and a presentation layer. When we split a story [**horizontally**], we're serving up only part of that cake.

We want to **give the customer the essence of the whole cake**, and the best way is to **slice vertically** through the layers.

Developers often have an inclination **to work on only one layer at a time** (and get it "right"); but a full database layer (for example) has little value to the customer if there's no presentation layer.

Valuable (2)

- ***As a consumer**, I can see other energy pricing programs that appeal to me so that I can enroll in a program that better suits my lifestyle.*
- ***As a utility marketing director**, I can present users with new pricing programs so that they are more likely to continue purchasing energy from me.*

Valuable (3)

Refactor the error logging system:

- ***As a consumer***, *I can receive a consistent and clear error message anywhere in the product so that I know how to address the issue.*

OR

- ***As a technical support member***, *I want the user to receive a consistent and clear message anywhere in the application so they can fix the issue without calling support.*

Small (Increased Throughput)

- User stories should be small enough to be able to be completed in an iteration.
- From queuing theory, we know that smaller batch sizes go through a system faster.

$$\text{Cycle Time} = \frac{\text{Work In Process}}{\text{Throughput}}$$

- Where **throughput**, **the amount of work that can be done in a unit of time**, is constant we have to decrease the amount of things we are working on in order to decrease cycle time.

Fewer, smaller stories in process will come out faster.

Small (Decreased Complexity)

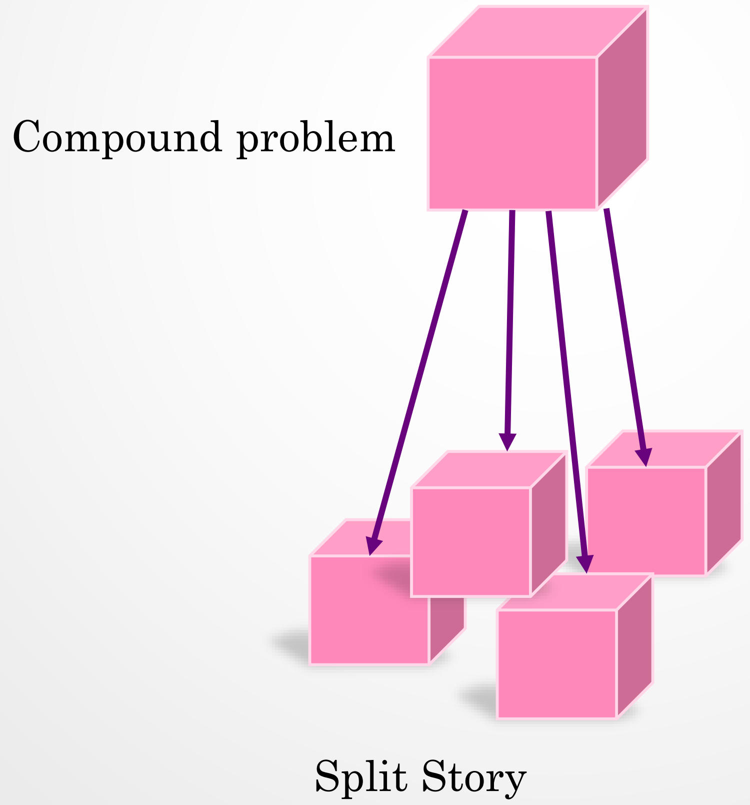
- Smaller stories go through faster because of their **decreased complexity**.
- Complexity has a nonlinear relationship to size.
- This is seen most readily in testing.
- Robert Martin:
  - Rule 1: Do one thing.
  - Rule 2: Keep them small
  - Rule 3: Make them smaller than that.
- Fibonacci estimating sequence

Testable

- If a story does not appear to be testable, then the story is probably:
  - ill-formed,
  - Overly complex,
  - Or perhaps dependent on other stories in the backlog
- Write-the-test-first approach.
- If a team really knows how to test a story, then they likely know how to code it.
- Words such as *quickly*, *manage*, *nice*, *clean*, and so on should be avoided.

Splitting user stories

Team must break feature and epic to user stories.



Patterns of Splitting User Stories (1)

1) **Workflow Steps pattern:**
   - Identify steps that takes to accomplish a specific workflow.
   - Implement the workflow in incremental stages.
2) **Business Rule Variations:**
   - Break the story into several stories to handle the business rule complexity.
3) **Major Effort**
   - Sometimes a story can be split into several parts where most of the effort will go toward implementing the first one.

Patterns of Splitting User Stories (2)

4) **Simple/Complex**
   - When the team is discussing a story and the story seems to be getting larger and larger ("What about x? Have you considered y?").
   - Stop and ask, "What's the simplest version that can possibly work?"
   - Version as its own story, and then break out all the variations and complexities into their own stories.

5) **Variations in Data**
   - Data variations and data sources are another source of scope and complexity.

Patterns of Splitting User Stories (3)

6) **Data Entry Methods**
   - Sometimes complexity is in the user interface rather than the functionality itself.
   - Split the story to build it with the simplest possible UI, and then build the richer UI later.

7) **Defer System Qualities**
   - Sometimes, the initial implementation isn't all that hard, and the major part of the effort is in making it fast or reliable or more precise or more scalable.

8) **Operations (Example: Create Read Update Delete (CRUD))**
   - Words like manage or control are a giveaway that the story covers multiple operations, which can offer a natural way to split the story.

Patterns of Splitting User Stories (4)

9) **Use-Case Scenarios**
   - If use cases have been developed to represent complex user-to-system or system-to-system interaction,
   - The story can often be split according to the individual scenarios of the use case.

10) **Break Out a Spike**
   - In some cases, a story may be too large or overly complex, or perhaps the implementation is poorly understood.
   - In that case, build a technical or functional spike to figure it out; then split the stories based on that result.

Spike

- ✓ *Spikes* are a **special type of story** used to drive out risk and uncertainty in a user story or other project facet.
  - Familiarize the team with a new technology or domain.
  - Use a spike to analyze the implied behavior so they can split the story into estimable pieces.
  - The story may contain significant technical risk.
  - The story may contain significant functional risk.
- ✓ Estimable, Demonstrable, and Acceptable.

Spike 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.

Story modeling with index cards

- Writing user stories using physical index cards provides a powerful visual means for engaging the entire team in backlog development.
- This helps keep user stories small and focused, which is a key attribute.
- Cards may be arranged by feature (or epic).
- 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.
- The more cards you have, the more work you see.