Requirements and User Stories(I)

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2024

Introduction

- Scrum and sequential product development treat requirements very differently.

- With 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 and are fleshed out *just in time* and *just enough* for the teams to start building functionality to support that requirement.

What is a requirement?

- **New system's capability**.

- A *requirement* is simply a statement of what the system must do or what characteristic it must have.

- They focus on the "what" of the system.

Requirements in sequential product development

- Are treated much as they are in manufacturing.

- They are required, nonnegotiable specifications to which the product must conform.

- These requirements are created up front and given to the development group in the form of a highly detailed document.

- It is the job of the development group, then, to produce a product that conforms to the detailed requirements.

Requirements in sequential product development(Cnt'd)

- When a change from the original plan is deemed necessary, it is managed through a formal change control process.

- Because conformance to specifications is the goal, these deviations are undesirable and expensive.

- After all, much of the work in process (WIP), in the form of highly detailed requirements (and all work based on them), might need to be changed or discarded.
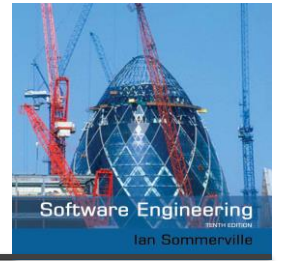
Requirements in Scrum

- Scrum views requirements as an important degree of freedom that we can manipulate to meet our business goals.

- For example, if we're running out of time or money, we can drop low-value requirements.

- If, during development, new information indicates that the cost/benefit ratio of a requirement has become significantly less favorable, we can choose to drop the requirement from the product.

- And if a new high-value requirement emerges, we have the ability to add it to the product, perhaps discarding a lower-value requirement to make room.

The fact is

- When developing innovative products, you can't create complete requirements or designs up front by simply working longer and harder.

- Some requirements and design will always emerge once product development is under way;

- No amount of comprehensive up-front work will prevent that.

Agile methods and requirements

 ✧ Many agile methods argue that producing detailed system requirements is a waste of time as requirements change so quickly.

 ✧ The requirements document is therefore always out of date.

 ✧ Agile methods usually use incremental requirements engineering and may express requirements as 'user stories'.

 ✧ This is practical for business systems but problematic for systems that require pre-delivery analysis (e.g. critical systems) or systems developed by several teams.

In Scrum,

- We don't invest a great deal of time and money in fleshing out the details of a requirement up front.

- Because we expect the specifics to change as time passes and as we learn more about what we are building, we avoid overinvesting in requirements that we might later discard.

- Instead of compiling a large inventory of detailed requirements up front, we create placeholders for the requirements, called product backlog items (PBIs).

- Each product backlog item represents desirable business value.

The Product Backlog Items

- Initially the product backlog items are large (representing large swaths of business value), and there is very little detail associated with them.

- Over time, we flow these product backlog items 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, however, more details will be exposed in conversations between the product owner and the development team.

Product backlog over time

Time

Legend

Placeholder size

Larger

Smaller

Amount of detail

A little   A lot
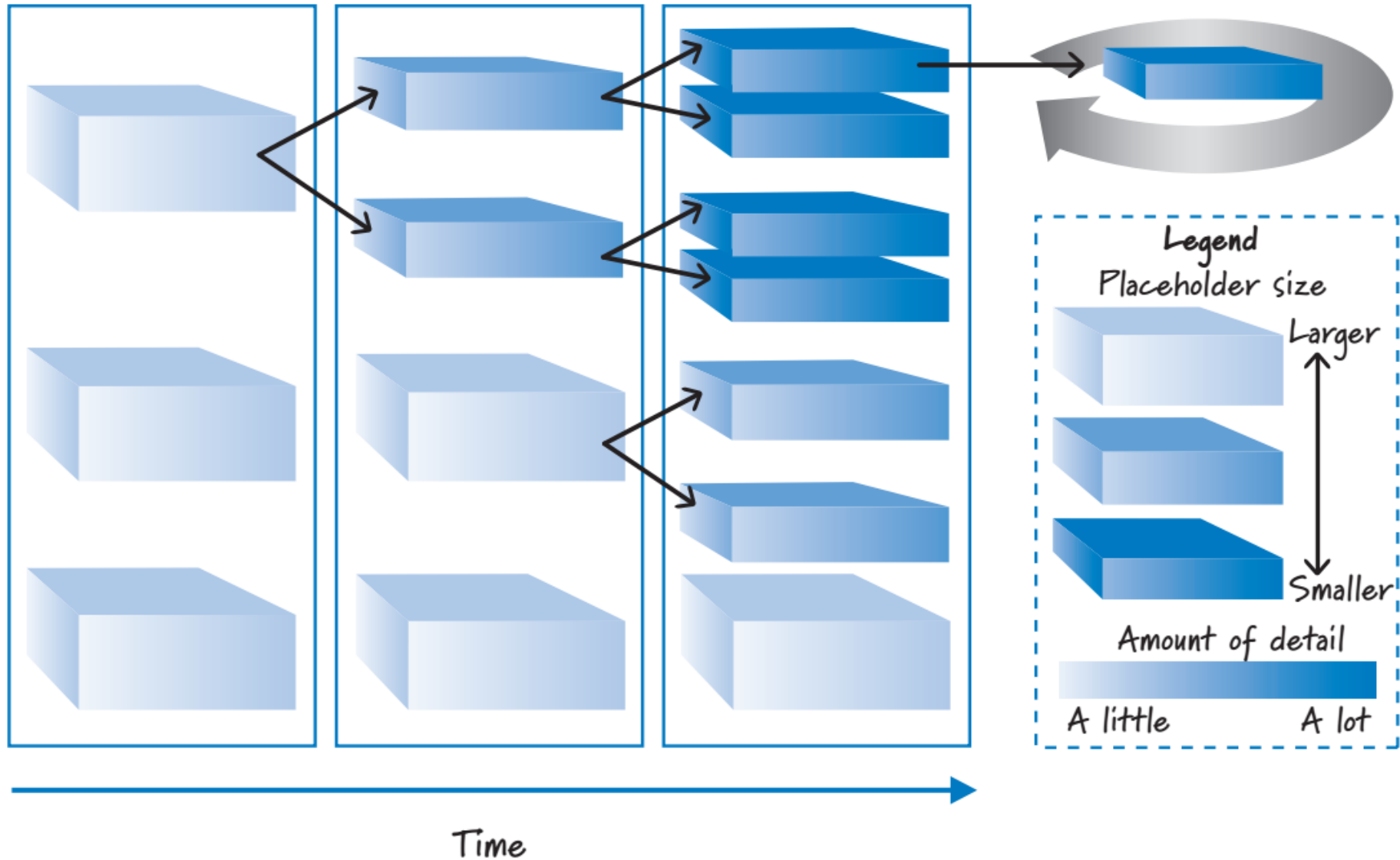
The product backlog is simply a snapshot of the current collection of items and their associated details.

Format of requirements

- While Scrum doesn't specify any standard format for these product backlog items, many teams represent PBIs as user stories.

- You don't have to. Some teams prefer use cases, and others choose to represent their PBIs in their own custom formats.

Using Conversations

■ As a communication vehicle, requirements facilitate a shared understanding of what needs to be built.

■ They allow the people who understand what should be created to clearly communicate their desires to the people who have to create it.

Sequential Product Development

- Relies heavily on written requirements, which look impressive but can easily be misunderstood.

- It seemed unlikely that there would be no ambiguities in document, detailed use case document. English just isn't that precise; even if it were, people just aren't that precise with their writing.

A way to better ensure that the desired features are being built is

for the people who know what they want

to have timely conversations with

the people who are designing, building, and testing features.

Using conversation in Scrum

■ We leverage conversation as a key tool for ensuring that requirements are properly discussed and communicated.

■ Verbal communication has the benefit of being high-bandwidth and providing fast feedback, making it easier and cheaper to gain a shared understanding.

■ Conversations enable bidirectional communication that can spark ideas about problems and opportunities—discussions that would not likely arise from reading a document.

Using conversation in Scrum (Cnt'd)

- Conversation is just a tool.

- It doesn't replace all documents.

- In Scrum, the product backlog is a "living document," available at all times during product development.

- Those who still want or must have a requirements specification document can create one at any time, simply by collecting the product backlog items and all of their associated details into a document formatted however they like.

Progressive Refinement

- With sequential product development all requirements must be at the same level of detail at the same time.

- Approved requirements document must specify each and every requirement so that the teams doing the design, build, and test work can understand how to conform to the specifications.

- There are no details left to be added.

Disadvantages of forcing all requirements to be at the same level of detail at the same time

- We must predict all of these details early during product development when we have the least knowledge that we'll ever have.

- We treat all requirements the same regardless of their priority, forcing us to dedicate valuable resources today to create details for requirements that we may never build.

- We create a large inventory of requirements that will likely be very expensive to rework or discard when things change.

- We reduce the likelihood of using conversations to elaborate on and clarify requirements because the requirements are already "complete."

Refinement is Scrum

- 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 won't work on for some time.

- We employ a strategy of progressive refinement to disaggregate, in a just-in-time fashion, large, lightly detailed requirements into a set of smaller, more detailed items.

What Are User Stories?

- User stories are a convenient format for expressing the desired business value for many types of product backlog items.

- User stories 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.

- Additionally, they can be written at various levels of granularity and are easy to progressively refine.

What Are User Stories (Cnt'd)?

■ They are simply a lightweight approach that match nicely with core agile principles and our need for an efficient and effective placeholder.

■ User stories are central placeholder to attach any other relevant information and they are helpful for detailing a requirement.

What Are User Stories (Cnt'd)?

- Ron Jeffries offers a simple yet effective way to think about user stories (Jeffries 2001).

- He describes them as three Cs: **C**ard, **C**onversation, and **C**onfirmation.

Card

- The card idea is pretty simple.

- People originally wrote (and many still do) user stories directly on 3 × 5-inch index cards or sticky.

Card (Cnt'd)

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

■ The "so that" part of a user story is optional, but unless the purpose of the story is completely obvious to everyone, we should include it with every user story.

Card (Cnt'd)

- The card isn't 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.

- It serves as the placeholder for more detailed discussions that will take place among the stakeholders, product owner, and development team.

Conversation

■ The details of a requirement are exposed and communicated in a conversation among the development team, product owner, and stakeholders.

■ The user story is simply a promise to have that conversation.

Conversation (Cnt'd)

- Conversation is typically not a one-time event, but rather an ongoing dialogue.

- There can be an initial conversation when the user story is written, another conversation when it's refined, yet another when it's estimated, another during sprint planning (when the team is diving into the task-level details), and finally, ongoing conversations while the user story is being designed, built, and tested during the sprint.

Conversation (Cnt'd)

- One of the benefits of user stories is that they shift some of the focus away from writing and onto conversations.

- These conversations enable a richer form of exchanging information and collaborating to ensure that the correct requirements are expressed and understood by everyone.

- Although conversations are largely verbal, they can be and frequently are supplemented with documents.

User story with additional data attached

User story can reference an entire article for future reading and conversation.

Conversation (Cnt'd)

- User stories are simply a good starting point for eliciting the initial essence of what is desired, and for providing a reminder to discuss requirements in more detail when appropriate.

- However, user stories can and should be supplemented with whatever other written information helps provide clarity regarding what is desired.

Confirmation

■ A user story also contains confirmation information in the form of conditions of satisfaction.

■ These are **acceptance criteria** that clarify the desired behavior.

■ They are used by the development team to better understand what to build and test by the product owner to confirm that the user story has been implemented to his satisfaction.

Confirmation (Cnt'd)

- If the front of the card has a few-line description of the story, the back of the card could specify the conditions of satisfaction.

Upload File	Conditions of Satisfaction
As a wiki user I want to upload a file to the wiki so that I can share it with my colleagues.	Verify with .txt and .doc files Verify with .jpg, .gif, and .png files Verify with .mp4 files <= 1 GB Verify no DRM-restricted files

Confirmation (Cnt'd)

- These conditions of satisfaction can be expressed as high-level acceptance tests.

- These tests would not be the only tests that are run when the story is being developed.

- In fact, for the handful of acceptance tests that are associated with a user story, the team will have many more tests (perhaps 10 to 100 times more) at a detailed technical level that the product owner doesn't even know about.

Reasons of Confirmation

- First, they are an important way to capture and communicate, from the product owner's perspective, how to determine if the story has been implemented correctly.

- These tests can also be a helpful way to create initial stories and refine them as more details become known. This approach is sometimes called specification by example or **acceptance-test-driven development (ATTD)**.

An example

■ Initially, let's limit uploaded file sizes to be 1 GB or less. Also, make sure that we can properly load common text and graphics files. And for legal reasons we can't have any files with digital rights management (DRM) restrictions loaded to the wiki.

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

Confirmation (Cnt'd)

- If we were using a tool, we could conveniently define these tests in a table, which shows examples of different file sizes and whether or not they are valid.

- By elaborating on specific examples, we can drive the story creation and refinement process and have (automated) acceptance tests available for each story.

Size	Valid()
0	True
1,073,741,824	True
1,073,741,825	False

Level of Detail

■ User stories are an excellent vehicle for carrying items of customer or user value through the Scrum value-creation flow.

■ If we have only one story size (the size that would comfortably fit within a short-duration sprint), it will be difficult to do higher-level planning and to reap the benefits of progressive refinement.

Level of Detail(Cnt'd)

■ Small stories used at the sprint level are too small and too numerous to support higher-level product and release planning. At these levels we need fewer, less detailed, more abstract items.

■ Otherwise, we'll be mired in a swamp of mostly irrelevant detail.

■ Imagine having 500 very small stories and being asked to provide an executive-level description of the proposed product to secure your funding. Or try to prioritize among those 500 really small items to define the next release.

Level of Detail(Cnt'd)

- Having only small stories precludes the benefit of progressively refining requirements on a just enough, just-in-time basis.

- Fortunately, user stories can be written to capture customer and user needs at various levels of abstraction.

User story abstraction hierarchy
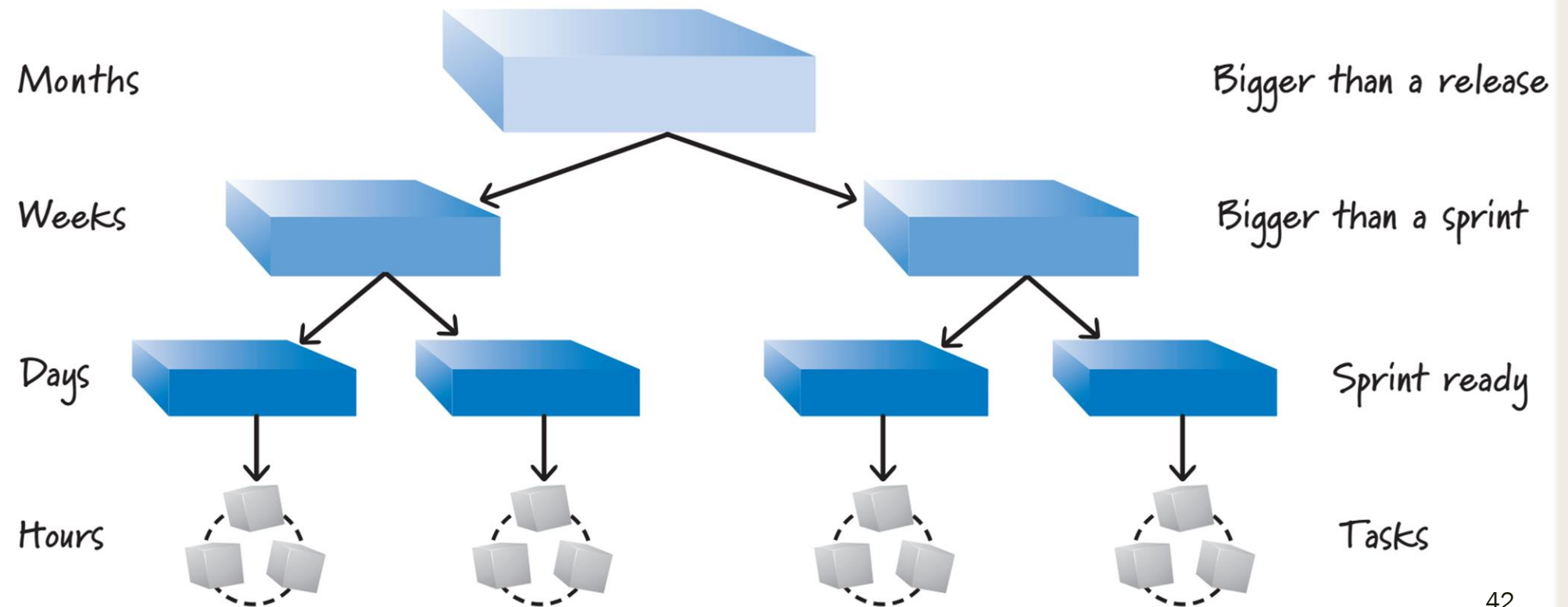
Stories at multiple levels of abstraction- epic

■ The largest would be stories that are a few to many months in size and might span an entire release or multiple releases. Many people refer to these as **epics**.

■ Epics are helpful because they give a very big-picture, high-level overview of what is desired.

■ We would never move an epic into a sprint for development because it is too big and not very detailed.

■ Instead, epics are excellent placeholders for a large collection of more detailed stories to be created at an appropriate future time.

Example epic

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.

Stories at multiple levels of abstraction-feature and story

- The next-size stories are often on the order of weeks in size and therefore too big for a single sprint. Some teams might call these **features.**

- The smallest forms of user stories are **stories.**

- Call these stories either **sprintable stories** or **implementable stories** to indicate that they are on the order of days in size and therefore small enough to fit into a sprint and be implemented.

Stories at multiple levels of abstraction- task

■ **Tasks** are the layer below stories, typically worked on by only one person, or perhaps a pair of people.

■ Tasks typically require hours to perform. When we go to the task layer, we are specifying *how* to build something instead of *what* to build (represented by epics, features, and stories).

■ Tasks are not stories, so we should avoid including task-level detail when writing stories.

Stories at multiple levels of abstraction

■ It really doesn't matter what labels you use as long as you use them consistently.

■ What does matter is recognizing that stories can exist at multiple levels of abstraction, and that doing so nicely supports our efforts to plan at multiple levels of abstraction and to progressively refine big items into small items over time.

Reference

1- K. S. Rubin, "Essential Scrum, A Practical guide to the most popular agile process," 2013.