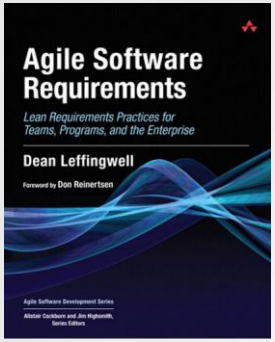


Agile Software Requirements

Software Requirements Engineering – 40688

Computer Engineering department

Sharif university of technology

Fall 402

Chapter 3:

Agile Requirements for the Team

Contents

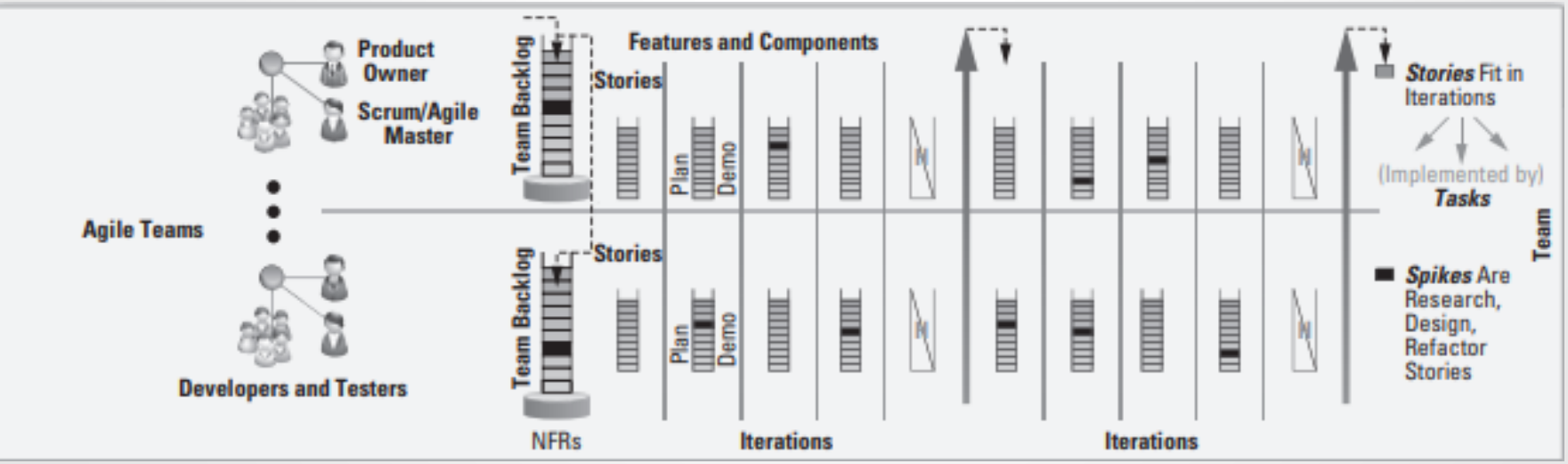
- **Introduction to the team level**
  - Why the discussion on teams?
  - Eliminating the functional silos
- **Agile team roles and responsibilities**
  - Product owner
  - Scrum/agile master
  - Developers
  - Testers
- **User stories and the team backlog**
  - Backlog
  - User stories
  - User story Basics
  - Tasks
- **Acceptance Tests**
- **Unit tests**
  - Real quality in real time
- **summary**

When you're part of a team, you stand up for your teammates. Your loyalty is to them. You protect them through good and bad, because they'd do the same for you.

—Yogi Berra

Why the Discussion on Teams? (1)

why a book on **software requirements** leads with a discussion of the organization, roles and responsibilities, and activities of the **agile project team?**



Why the Discussion on Teams? (2)

1) The nature of **agile development** is so fundamentally different from that of **traditional models** that, by necessity, **we must rethink** many of the basic **practices of software development**.

2) More importantly, in agile, **the organization of the requirements** and the **organization of the team itself** are not independent things.

3) No longer do large batches of predetermined requirements defend by others get thrown "over the transom" to a set of developers for implementation—individuals that are organized, via matrix or other, for whatever purposes.

Why the Discussion on Teams? (3)

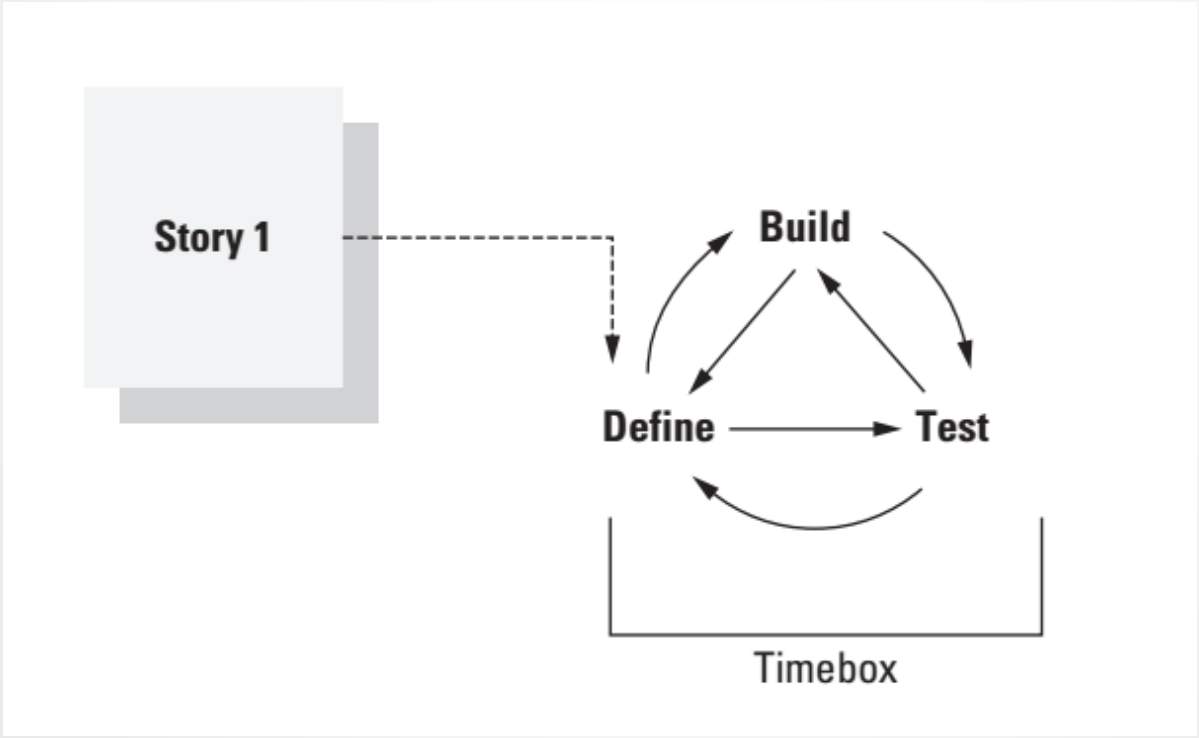
4) Rather, **the teams organize around the requirements** so as to optimize the efficiency of defining, building, and testing code that delivers value to the end users.

5) The entire team is integrally involved in defining requirements, optimizing requirements and design trade-offs, implementing them, testing them, integrating them into a new baseline, and then seeing to it that they get delivered to the customers. That is the sole purpose of the team.

Producing working code in a time box

- The basic unit of work for the team is the **user story**.
- The team's objective is to define, build, and test some number of **user stories in the scope of an iteration** and thereby achieve some even larger value pile in the course of a release.
- Each **story** has a short, incredibly intense development life cycle, ideally followed by long term residence in a software baseline that delivers user value for years to come.

Life Cycle of a Story (1)

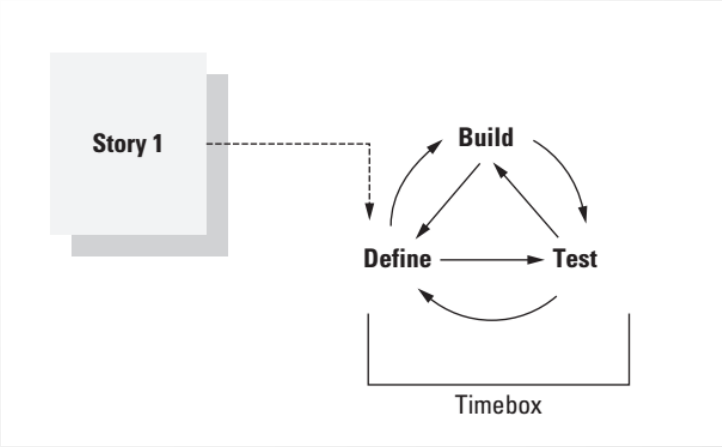


Define

- Even if the story is well-elaborated, the developer will likely still interact with the product owner to understand what is meant by the story.
- Also, some design will likely be present in the developer's mind, and if not, one will quickly be created and communicated to the product owner, peer developers, and testers.
- We use the word **define** to communicate that this **function** is a combination of both **requirements** *and* **design**.
- They are inseparable; neither has any meaning without the other. (If you don't know *how* to do IT, then you don't really know what IT is!)

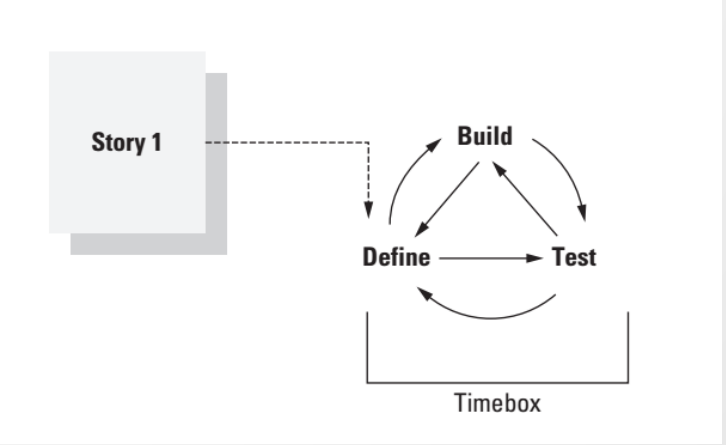
Build

- The actual coding of the story provides an opportunity for new discovery as well.
- Conversations will again ensue between developer or product owner, developer or other developers, and developer or tester.
- Story understanding evolves during the coding process.



Test

- A “story” is not considered complete until it has passed an **acceptance test.** (the topic of Chapter 10)
- It assures that the **code meets the intent of the story**.
- Building functional acceptance tests (plus unit tests) before, or in parallel with the code, again tests the team’s understanding of the subject story.



Life Cycle of a Story (2)

Of course, this process happens every day; it happens in real time, and it happens multiple times a day for each story in the iteration!

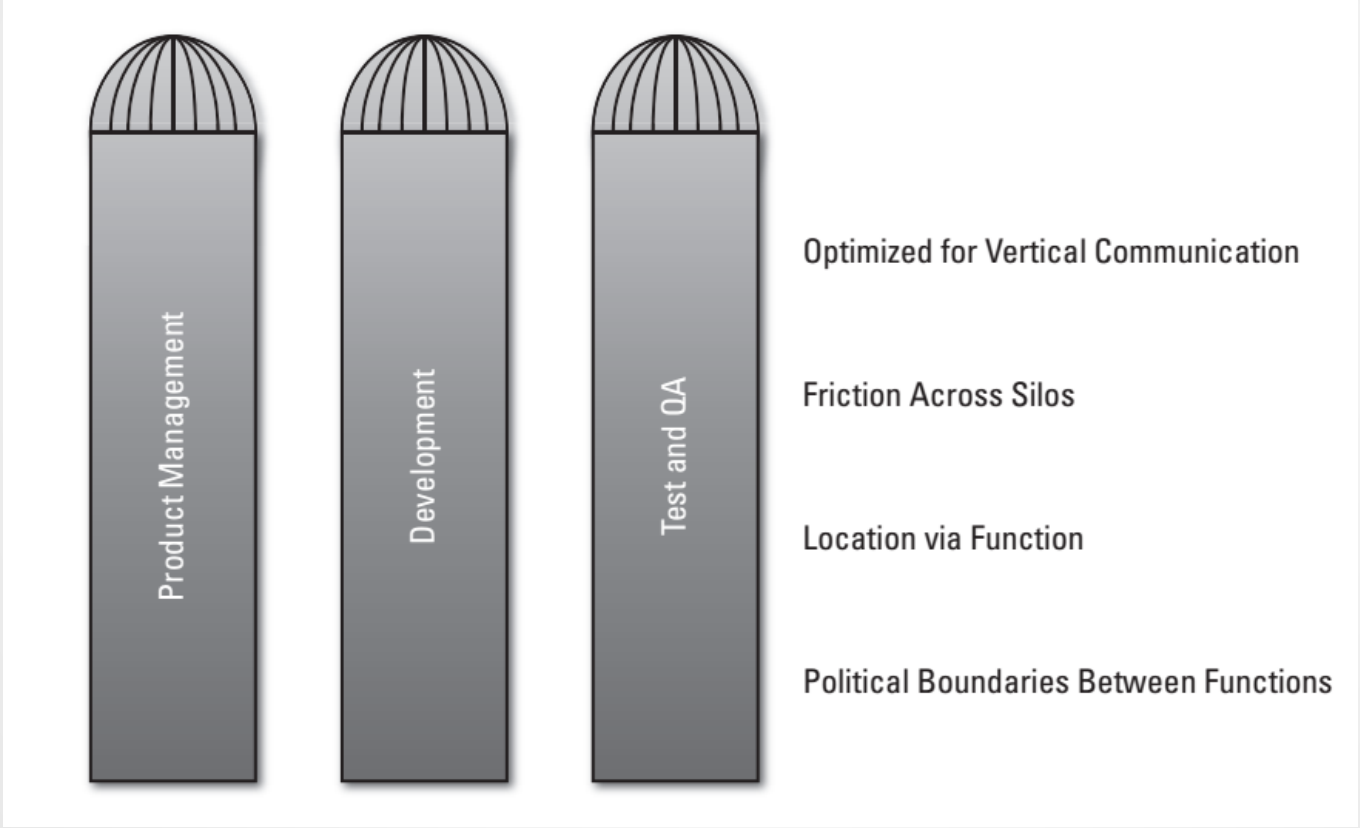
- *How could such a process work* in a traditional environment where a product owner or manager may not exist or has been called away on another mission?
- *How could it work* if the developer is multiplexed across multiple projects or is working part-time "on assignment" from a resource pool?
- *How could it work* if the test resources are not dedicated and available at the same time that the code is written?

✓The answer is, **it doesn't**.

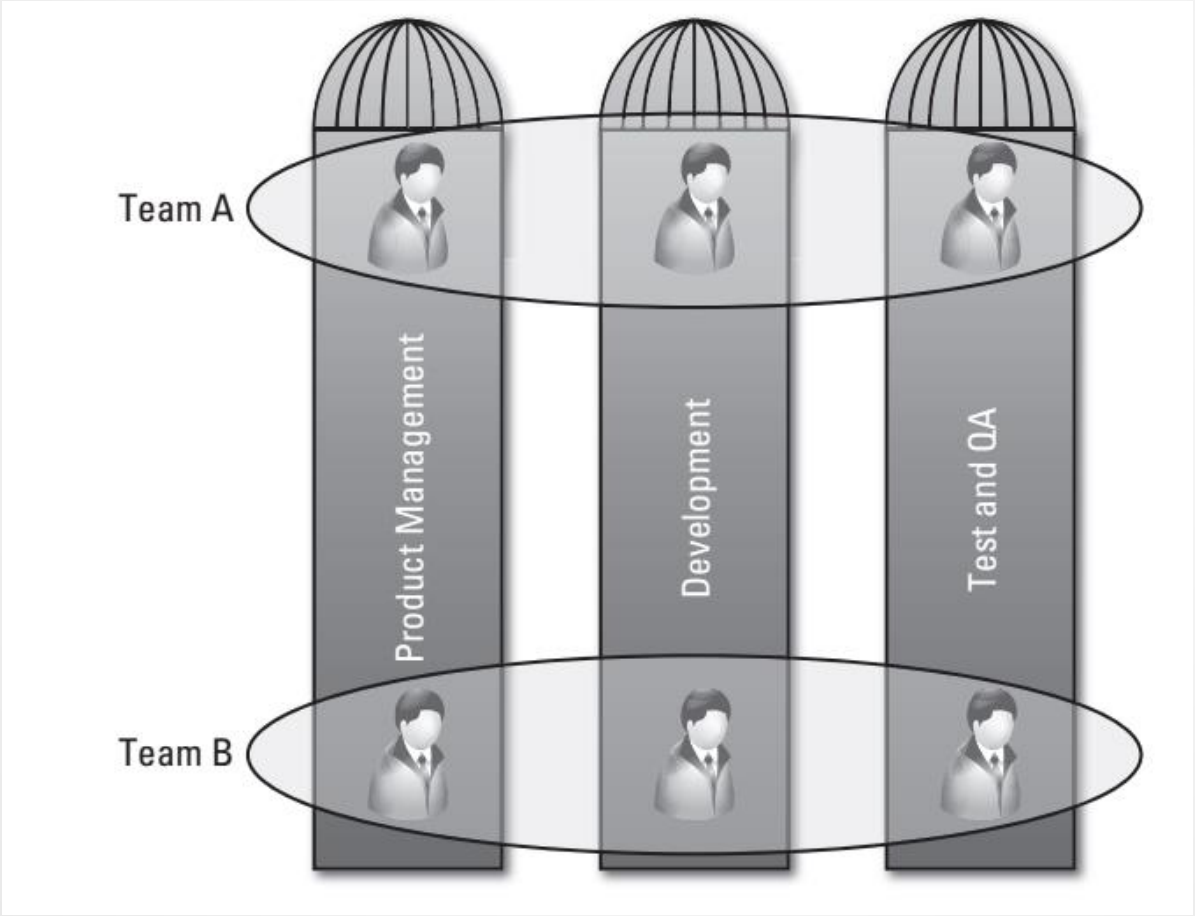
Life Cycle of a Story (3)

- Clearly, we are going to have **refactor our organization** to achieve this agile efficiency.
- We are going to have to **organize around the requirements stream** and **build teams** that have the full capability *to define, build, test, and accept* stories into the baseline every day.

Typical Functional Silos



Reorganizing into agile teams



Product Owner

In summary form, the product owner role is responsible for the following:

1) Working with product managers, business analysts, customers, and other stakeholders to determine the requirements.
2) Maintaining the backlog and setting priorities based on relative user value.
3) Setting objectives for the iteration.
4) Elaborating stories, participating in progress reviews, and accepting new stories.

Scrum /Agile Master

1) Facilitating the team's progress toward the goal.
2) Leading the team's efforts in continuous improvement.
3) Enforcing the rules of the agile process.
4) Eliminating impediments.

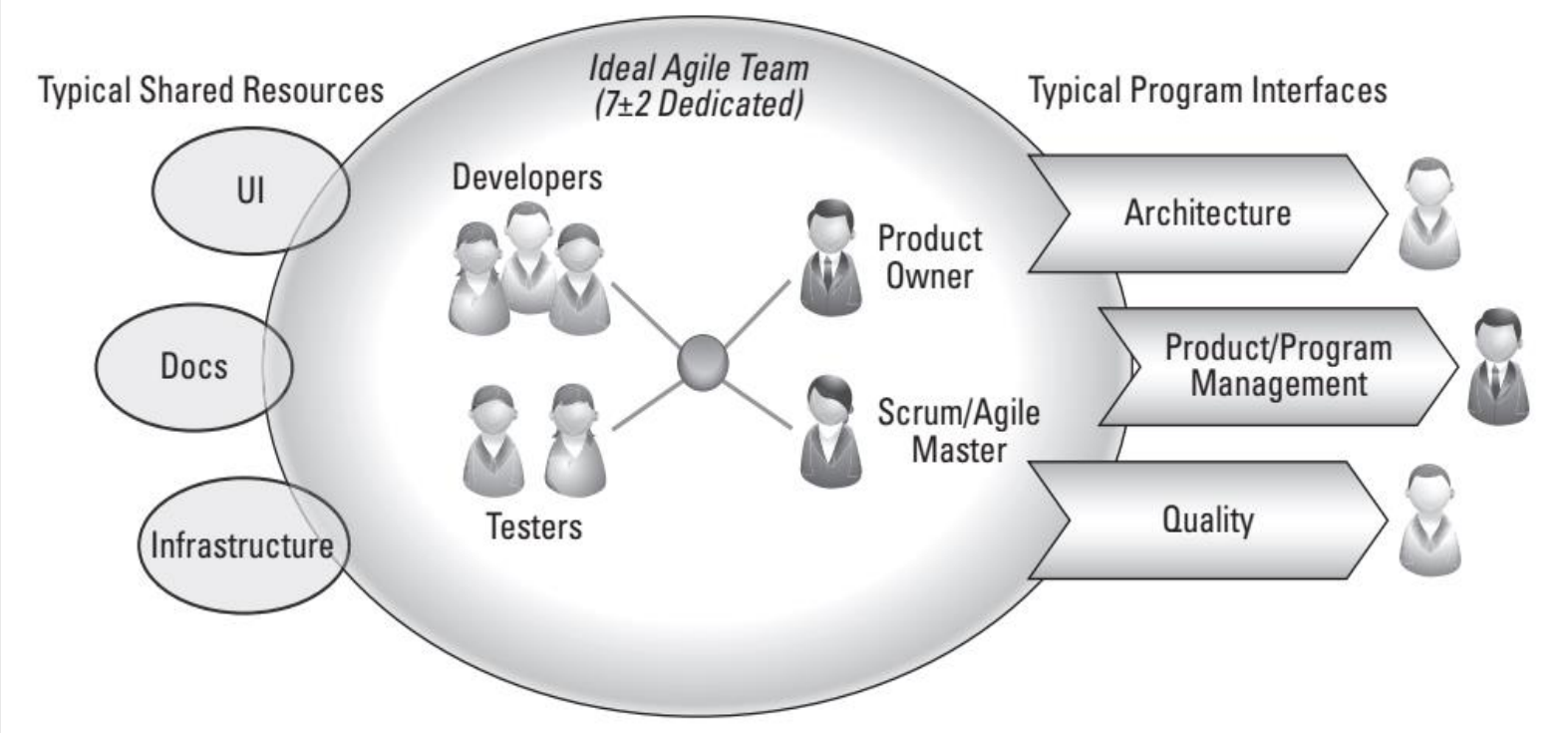
Developers

1) Collaborating with product owners and testers to make sure the right code is being developed.
2) Writing the code.
3) Writing and executing the unit test for the code.
4) Writing methods as necessary to support automated acceptance tests and other testing automation.
5) Checking new code into the shared repository every day.

Testers

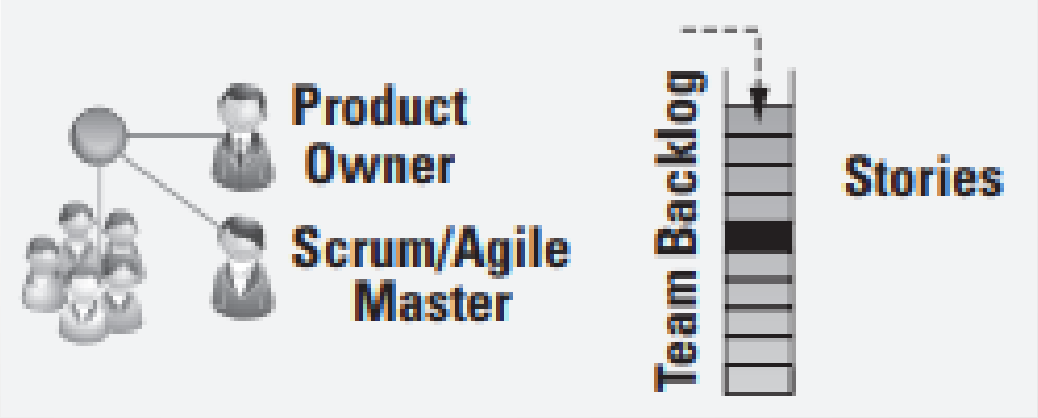
1) Writing the acceptance test case while the code is being written.
2) Interfacing with the developer and product owner to make sure the story is understood and that the acceptance tests track the desired functionality of the story.
3) Testing the code against the acceptance test.
4) Checking the test cases into the shared repository every day.
5) Developing ongoing test automation to integrate acceptance and component tests into the continuous testing environment.

Ideal agile team with shared resources and typical interfaces



Backlog (1)

The term **backlog** was first introduced by **Scrum**, where it was described as a **product backlog**. However, in our enterprise model, **product** can be a pretty nebulous thing because various teams may be working at various levels, so there are **multiple types of backlogs** in the Big Picture. Therefore, our use of the term *backlog* is more generalized than in Scrum.

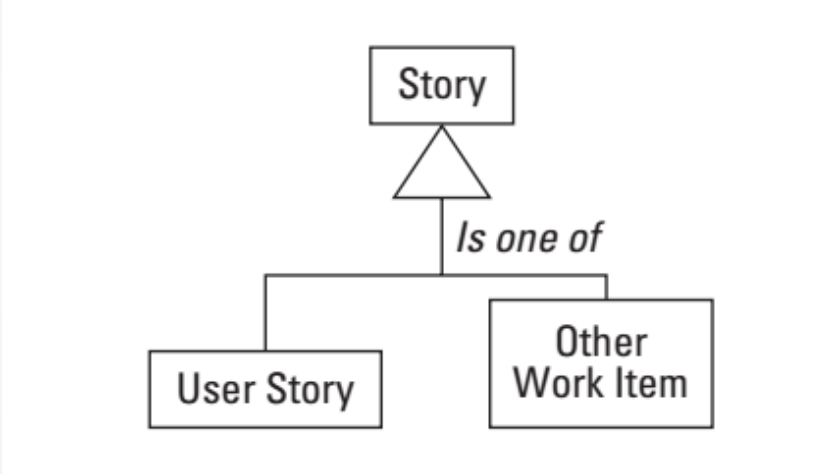


Backlog (2)

- Within the team, **maintenance** and **prioritization** of the **backlog** are the responsibility of the **product owner**, who is a resident of the team
- The team's backlog consists of all the work items the team has identified.
- In the meta-model, we generically call these work items **stories**.
- A **story** is a work item contained in the team's backlog.

User Stories

- The backlog is composed of user stories and other work items.



- Other work items include thing such as refactors, defects, support and maintenance, and tooling and infrastructure work.

User Story Basics (1)

- **User stories** are the agile replacement for most of what has been **traditionally** expressed as **software requirements statements** (or use cases in RUP and UML).
- Developed initially within the constructs of XP, they are now endemic to agile development in general and are taught in most Scrum classes as well.

User Story Basics (2)

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

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

"**As a** *Salesperson* (<role>), **I want to** *paginate my leads when I send mass e-mails* (<what I do with the system>) **so that** *I can quickly select a large number of leads* (<business value I receive>)."

Tasks (1)

To assure that the teams really understand the work to be done and to assure that they can meet their commitments, many agile teams take a very detailed approach to estimating and coordinating the individual work activities necessary to complete a story.

They do this via the **task**, which we'll represent as an additional model element.

Tasks (2)

- Tasks have an **owner** (the person who has taken responsibility for the task) and are **estimated in hours** (typically four to eight).
- The burndown (completion) of task hours represents one form of iteration status.
- In most cases, tasks are **"children"** to their **associated story** (deleting the story parent deletes the task). However, for flexibility, the model also supports stand-alone tasks and tasks that support other team objectives.

Task (3)

Story 51: Select photo for upload

Task 51.1: Define acceptance test—Juha, Don, Bill

Task 51.2: Code story—Juha

Task 51.3: Code acceptance test—Bill

Task 51.4: Get it to pass—Juha and Bill

Task 51.5: Document in user help—Cindy

Acceptance tests (1)

Ron Jeffries, one of the creators of XP, described what has become our favorite way to think about user stories.

He used the neat alliteration *card, conversation, and confirmation* to describe the three elements of a user story.

1) ***Card*** represents the two to three sentences used to describe the **intent of the story**.

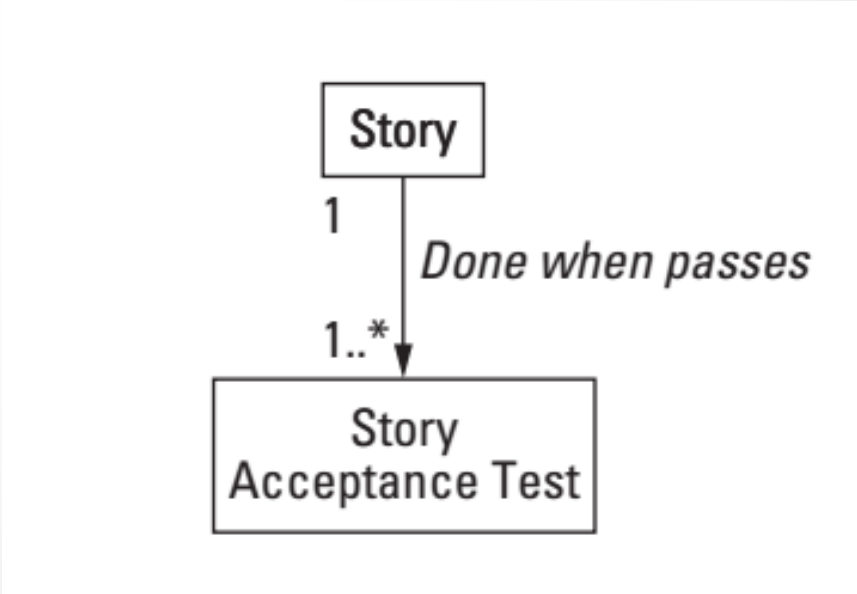
Acceptance tests (2)

2) ***Conversation*** represents fleshing out the **details** of the intent of the card in a **conversation** with the **customer** or **product owner**. In other words, the card also represents a "promise for a conversation" about the intent.

3) ***Confirmation*** represents how the team, via the customer or customer proxy, comes to understand that **the code meets the full intent of the story.**

Acceptance test (3)

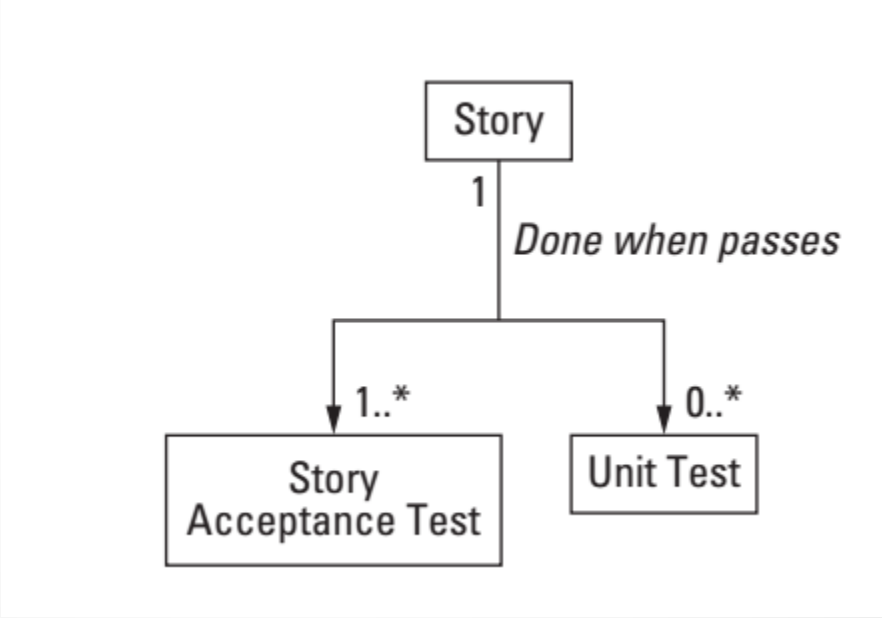
In our model, we represent the **confirmation function** as a type of **acceptance test**, one that confirms the story has been implemented correctly.



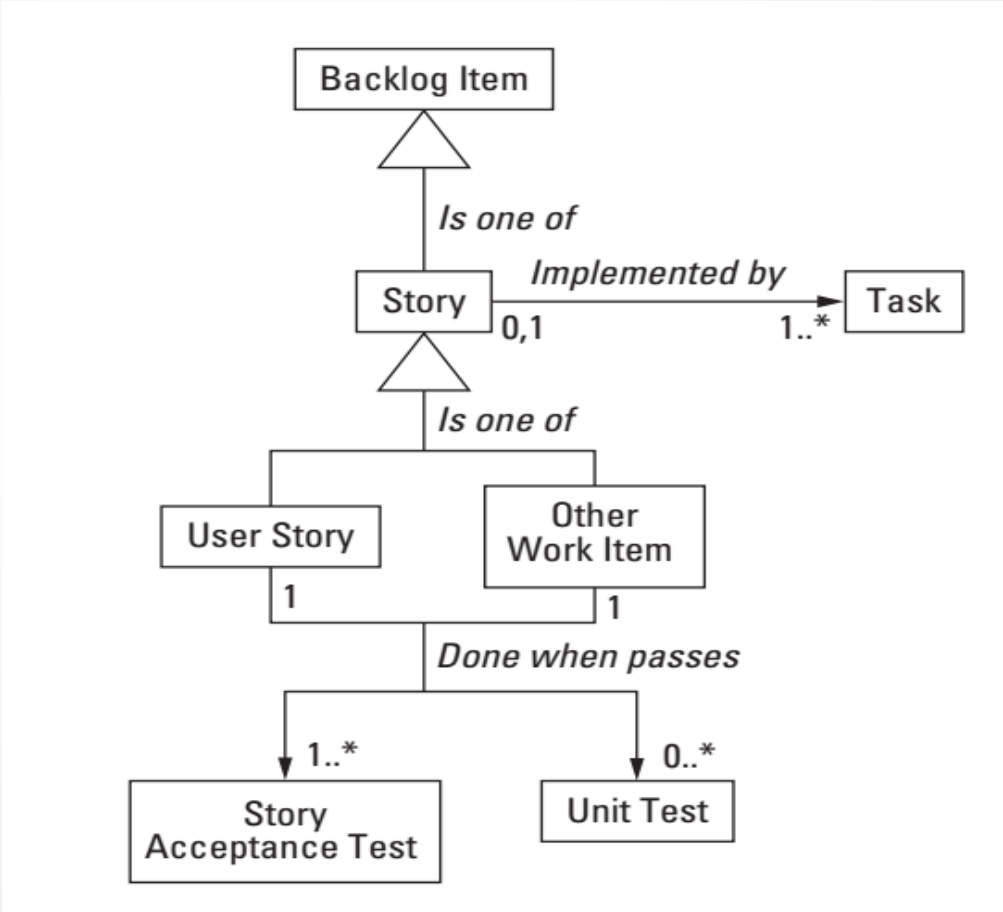
Unit Tests

- To further assure quality, we can augment the acceptance with *unit tests*.
- Unit tests are used to confirm that the **lowest-level module** of an application (a class or method in object-oriented programming; a function or procedure in procedural programming) **works as intended**.
- Unit tests are written **by the developer** to test that the code executes the logic of the subject module.
- In test-driven development(TDD), the test is written before the code.

Real Quality



Summary


Backlog Item
Is one of
Story
Implemented by
0,1
1..*
Task
Is one of
User Story
Other
Work Item
1
1
Done when passes
1..*
0..*
Story
Acceptance Test
Unit Test