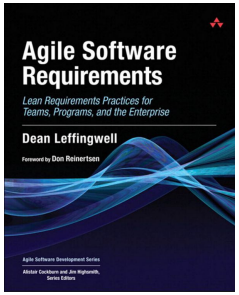
CE Department

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

Agile Requirements For The Team

Chapter 3

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

Contents

1. Introduction To The Team Level

Introduction To The Team Level

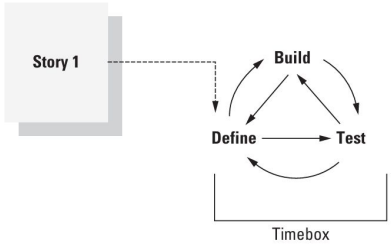
Why does a discussion on software requirements begin with the topic of the agile project team?

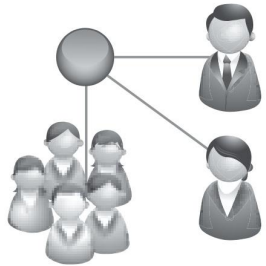
- ➔ The **nature** of **agile development** is so fundamentally **different** from that of **traditional models** that, by necessity, we must **rethink** many of **the basic practices of software development**.
- ➔ The **organization of the requirements** and the **organization of the team itself** are **not independent** things.
- ➔ 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.

Introduction To The Team Level

Let’s consider producing working code in a timebox

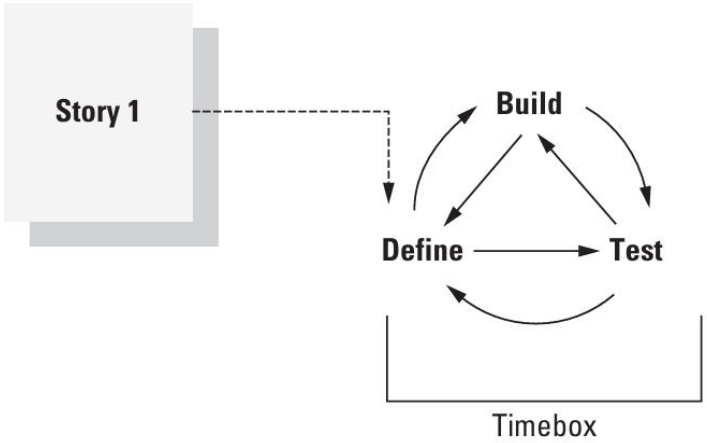
- ➔ The **basic unit** of **work** for the **team** is the **user story**.
- ➔ The **team’s objective** is to **define**, **build**, and **test some number** of **user stories** in the scope of an **iteration** and thereby achieve some even larger **value** pile in the course of a **release**.





Introduction To The Team Level

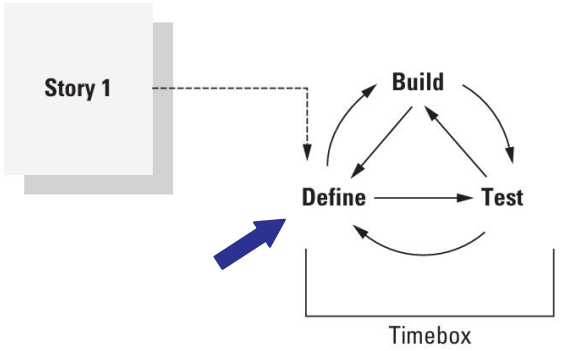
➔ Each story has a **short**, **incredibly intense development life cycle**, ideally followed by long-term residence in a software baseline that **delivers user value** for years to come.



Introduction To The Team Level

Define

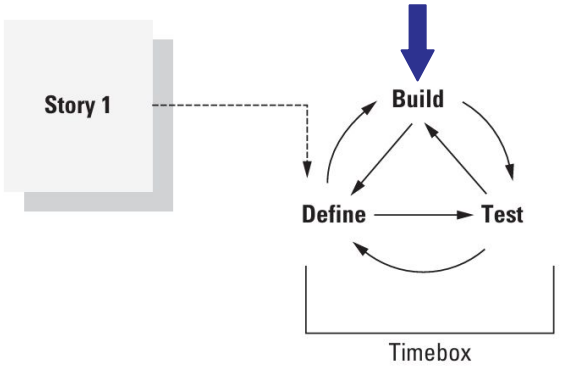
- ➔ Even if the story is well-elaborated, the **developer** will likely **still interact** with the **product owner** to understand **what is meant by the story**.
- ➔ Also, some **design** will likely be present in the developer's mind, and if not, one will **quickly** be **created** and **communicated** 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**.



Introduction To The Team Level

Build

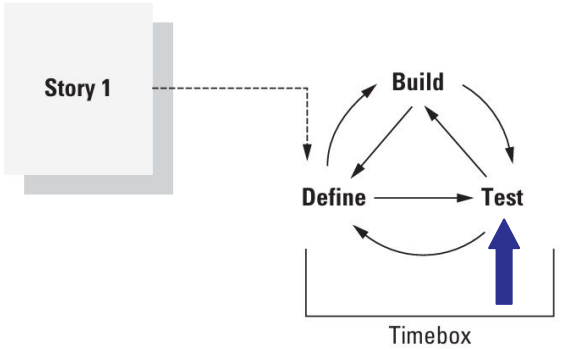
- The **actual coding** of the story provides an opportunity for **new discovery** as well.
- **Conversations** will again ensue between developer/product owner, developer/other developers, and developer/tester.
- **Story understanding evolves** during the **coding process**.



Introduction To The Team Level

Test

- ➔ A “story” is not considered complete until it has passed an **acceptance test**, which 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**.



Introduction To The Team Level

- ➔ 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?**
- ➔ 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.

Introduction To The Team Level

Eliminating The Functional Silos

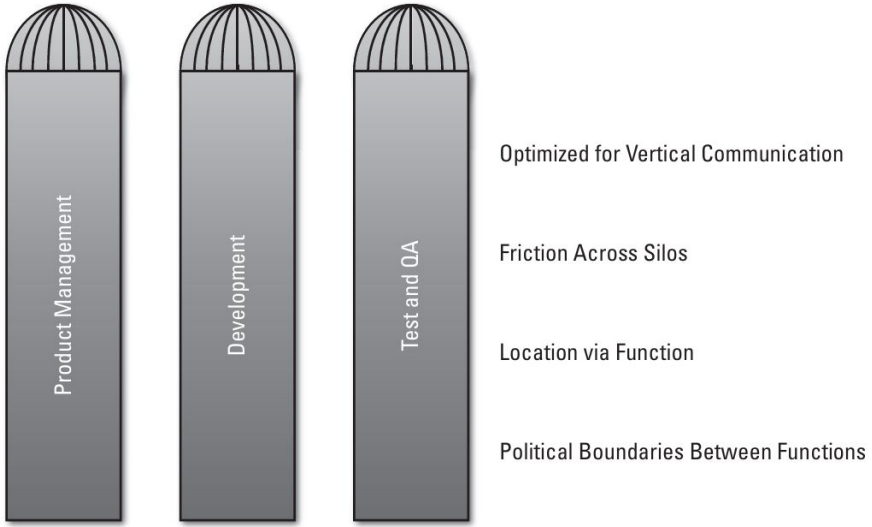
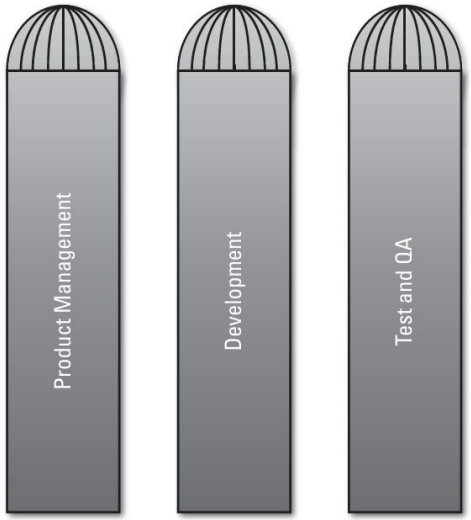


Figure 3–3 Typical functional silos

Introduction To The Team Level

- **Developers** sit with, and **communicate** with, **other developers.**
- **Product management, business analysts**, and **program managers** are **co-located** with each other and often report to different departments entirely.
- **Architects** may **work together**, may have little affinity or association with the development teams themselves.
- **Testers** probably report to—and are **co-located** with—a separate organization, rather than development.



Introduction To The Team Level

- In agile, we must redefine what makes a team a team and eliminate the silos that separate the functions.

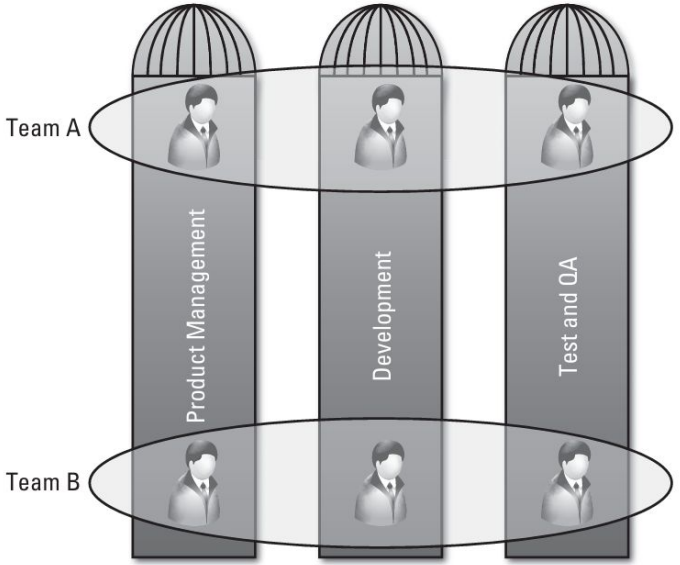


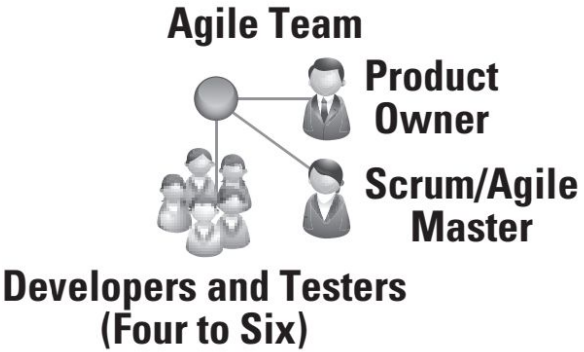
Figure 3–4 Reorganizing into agile teams

Contents

2. **Agile Team Roles And Responsibilities**

Agile Team Roles And Responsibilities

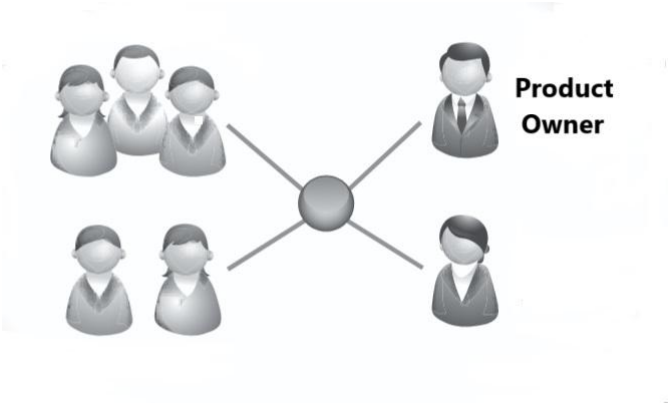
- ➔ The basic organizational structure of each agile team is largely the same.
- ➔ There are 3 roles on an agile project team:
  - The product owner
  - The Scrum master
  - The Rest of the team
    - Developers
    - Testers



Agile Team Roles And Responsibilities

Product Owner

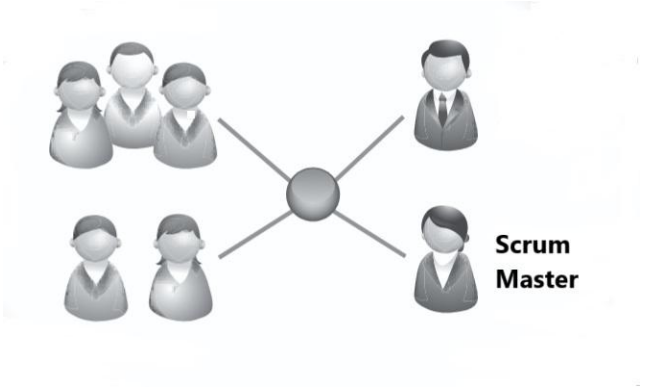
- ➔ Working with product managers, business analysts, customers, and other stakeholders to determine the requirements.
- ➔ Maintaining the backlog and setting priorities based on relative user value.
- ➔ Setting objectives for the iteration.
- ➔ Elaborating stories, participating in progress reviews,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



Agile Team Roles And Responsibilities

Scrum Master

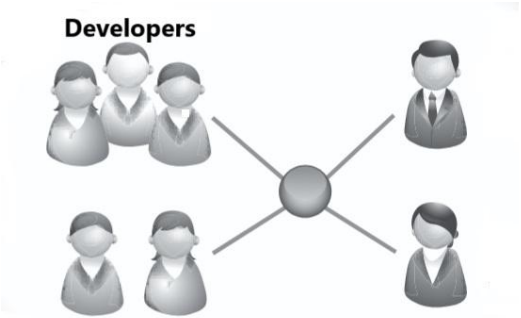
- ➔ Facilitating the team's progress toward the goal
- ➔ Leading the team's efforts in continuous improvement
- ➔ Enforcing the rules of the agile process
- ➔ Eliminating impediments



Agile Team Roles And Responsibilities

Developers

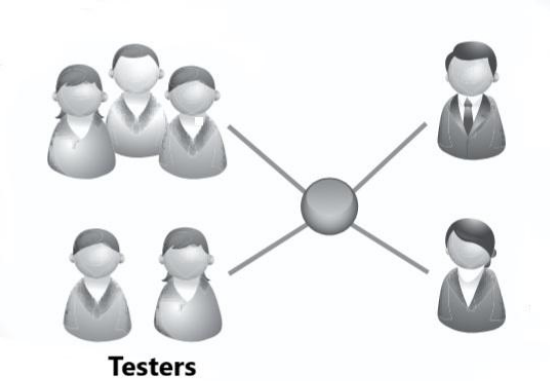
- ➔ Collaborating with product owners and testers to make sure the right code is being developed.
- ➔ Writing the code.
- ➔ Writing and executing the unit test for the code.
- ➔ Writing methods as necessary to support automated acceptance tests and other testing automation.
- ➔ Checking new code into the shared repository every day.
- ➔ developers actively participate in improving the development environment.



Agile Team Roles And Responsibilities

Testers

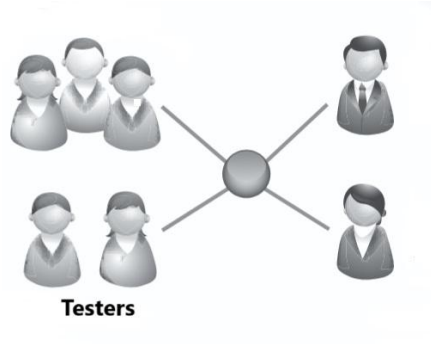
- ➔ Testers are an integral part of every agile team.
- ➔ They become part of the team just as soon as new code starts to be laid down, and they continue with the team throughout the release process.
- ➔ Their cycle is the same as the development cycle.
- ➔ Every new story that reaches an iteration boundary is subject to immediate review and analysis for acceptance testability.

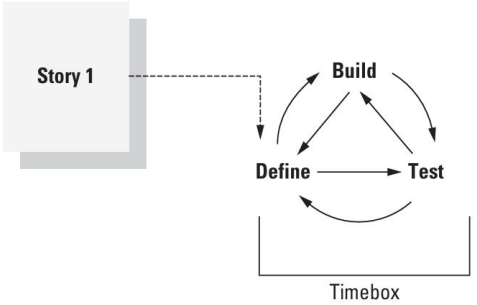


Agile Team Roles And Responsibilities

Testers

- ➔ Writing the acceptance test case while the code is being written.
- ➔ 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.
- ➔ Testing the code against the acceptance test.
- ➔ Checking the test cases into the shared repository every day.
- ➔ Developing ongoing test automation to integrate acceptance and component tests into the continuous testing environment.





Agile Team Roles And Responsibilities

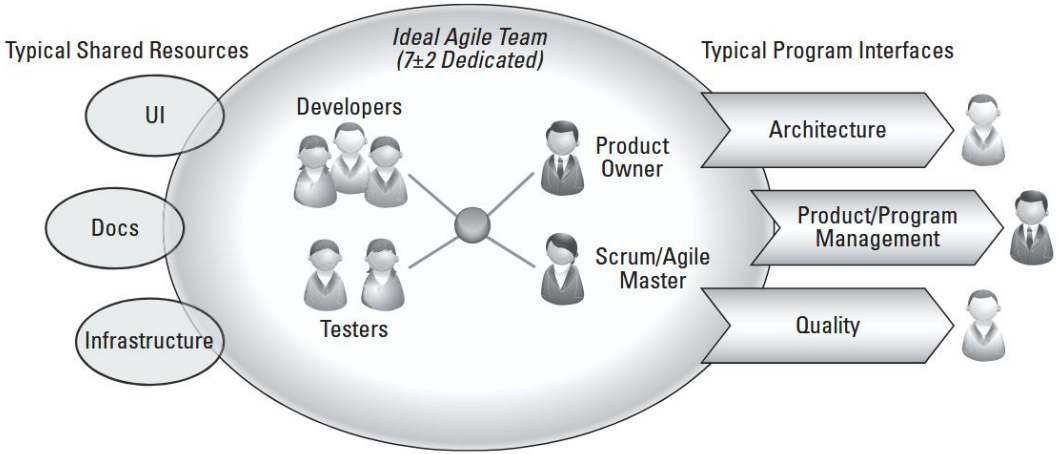


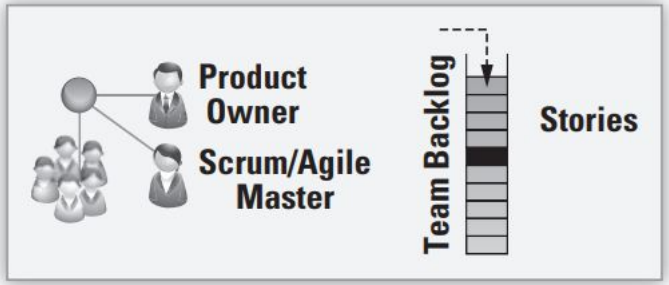
Figure 3–5 Ideal agile team with shared resources and typical interfaces

Contents

3. **User Stories And Team Backlog**

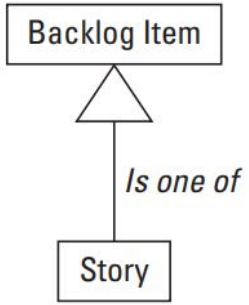
Team Backlog

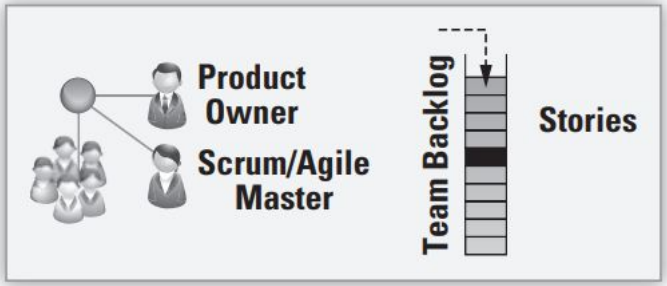
- The **only definite source** of **work** for the **team**.
- It is **local to them** and is **managed** by them.
- It is a **repository** for **all identified work items** (primarily user stories) in order to meet the team's iteration objectives.
- **Maintenance** and **prioritization** of the backlog are the responsibility of the **product owner**.



Team Backlog

- ➔ The team's backlog consists of all the work items the team has identified.
- ➔ A story is a work item contained in the team's backlog.





User Stories

- Agile teams use User Story to **define** the **system behavior** and **value** for the **user**.
- Other work Items: **Refactors**, **defects**, **support**, **maintenance**, **tooling**, and **infrastructure work**.
- They **help team to keep track of all the work they have to do to deliver value**.
- They also **help the team better estimate the time** it will take to actually deliver the user stories.

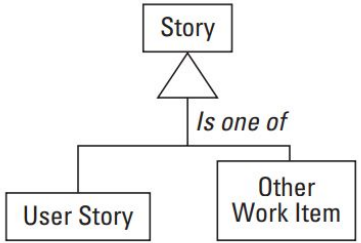
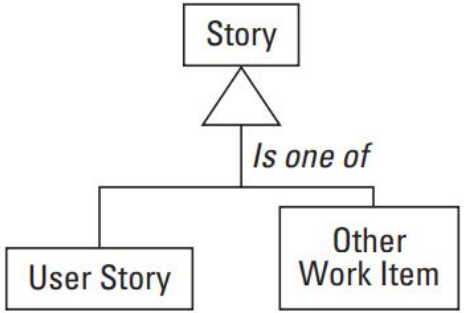


Figure 3–8 A story may be a user story or other work item.

User Stories

User Story Basics

- ➔ Replace traditional software requirements and use cases
- ➔ Serve as the core units of agile development
- ➔ Originated in Extreme Programming (XP)
- ➔ Widely adopted across agile frameworks
- ➔ Commonly taught in Scrum training



User Stories

User Story Basics

- It 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>

  - User Role → Persona
  - Business value → elements of problem space
  - Activity → Solution Space

User Stories

User Story Basics

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

Example:

➔ As a salesperson, I want to paginate my leads when I send mass emails so that I can quickly select a large number of leads.

User Stories

Tasks

- ➔ The **lowest-granularity thing** in the model and represent **activities** that **must be performed** by **specific team members** to accomplish the story.
- ➔ Tasks have an **owner.**
- ➔ They are **estimated in hours** (typically 4 to 8).
- ➔ They are **children** to their **associated story.**

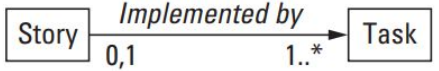


Figure 3–9 Stories are implemented by tasks.

User Stories

Tasks

Example: 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
- ➔ Test 51.4: Get it to pass _ Juha and Bill
- ➔ Task 51.5: Document in user help _ Cindy

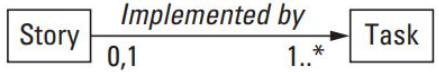


Figure 3–9 Stories are implemented by tasks.

Contents

4. Acceptance Tests

Story Acceptance Test

- All code is **tested code**.
- **Quality** is achieved **during**, rather than after, actual **code development**.
- **Acceptance test confirm** the **story** has been **implemented correctly**.
- It is a **distinct artifact** from other user stories.
- **Acceptance tests** are **functional tests** that **verify** that the **system implements** the **story** as intended.

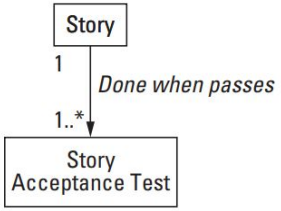


Figure 3–10 Every story has one or more story acceptance tests.

Story Acceptance Test

- **Focus**: Validates a user story — a feature or requirement from the user's perspective.
- **Written by**: Often testers or product owners, sometimes with developers.
- **Purpose**: Confirms that the story meets its acceptance criteria and delivers the expected behavior.
- **Example**: Verifying that a user can successfully log in with valid credentials.

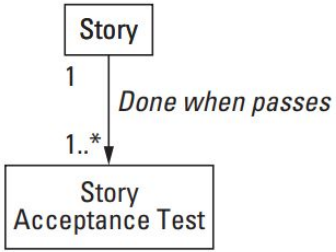


Figure 3–10 Every story has one or more story acceptance tests.

Contents

5. **Unit Tests**

Unit Test

- **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**.
- In any case, the **test** should be **written**, **passed**, and **built into** an **automated testing framework before** a **story** can be **considered done**.

Unit Test

- ➔ Mature agile teams provide comprehensive practices for unit testing and automated functional (story acceptance) testing.
- ➔ Also, for those in the process of tooling their agile project, implementing this meta-model can provide inherent traceability of story-to-test, without any overhead on the part of the team.

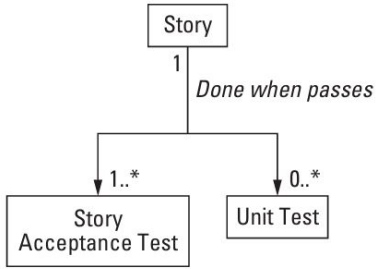


Figure 3–11 The code that implements the story should also be unit tested.

Unit Test

Real Quality In Real Time

- The combination of **creating** a lightweight **story description**, having a **conversation** about the **story**, **elaborating** the **story** into **functional tests**, **augmenting** the **acceptance** of the **story** with **unit tests**, and then **automating testing** is how agile teams achieve **high quality in the course of each iteration**.
- In this way: **Quality is built in**, one story at a time.
- **Continued assurance of quality** is achieved by **continuous** and **automated execution** of the aggregated **functional** and **unit tests**.

Summary

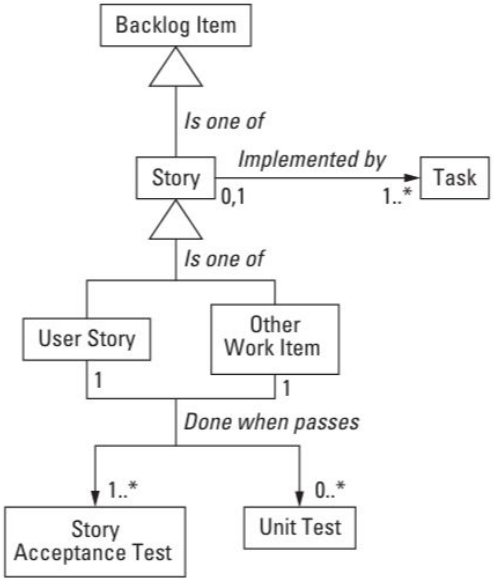


Figure 3–12 Agile team-level requirements artifacts and relationships

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.