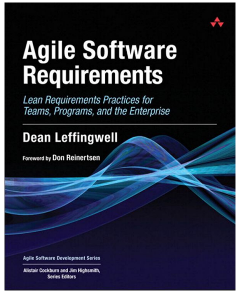
CE Department

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
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 (2011)
Dean Leffingwell

Acceptance Testing

Chapter 10

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

Introduction

Why do we feel compelled to write about testing now, in a book on agile requirements?

- ➔ The question itself reflects a traditional view, that historically, software requirements were somehow independent of their implementation.
- ➔ They lived a separate life—you could get them (reasonably right) at some point, mostly up front; the developers could actually implement them as intended; they would be tested somewhat independently to assure the system worked as intended; and the customers and users would be happy with the result.
- ➔ Of course, it never really worked that way, but it sure was easier to write about it.

Introduction

Why do we feel compelled to write about testing now, in a book on agile requirements? (Cont.)

- ➔ In thinking in lean and agile terms, however, we must take a much more systematic and holistic view.
- ➔ We understand that stories (requirements), implementation (code), and validation (acceptance tests, unit tests, and others) are not separate activities but a continuous refinement of a much deeper understanding; therefore, our thinking is different.

Introduction

Why do we feel compelled to write about testing now, in a book on agile requirements? (cont.)

- ➔ No matter what we thought earlier in the project, this functionality is what the user really needs, and it's now implemented, working, and tested in accordance with the continuous discussions and agreements we have forged during development.
- ➔ Just as importantly, we have instrumented the system (with automated regression tests) such that we can assure this functionality will **continue to work** as we **make future changes** and **enhancements** to the **system**. Then, and only then, can we declare that our work is complete for this increment.

Introduction

- ➔ we have captured the details of system behavior in a set of tests that will persist for all the time the software continues to provide value to its users. So, the requirements are implemented, complete, and tested.
- ➔ In agile, we simply can’t do that without a discussion of testing.

Contents

1. Agile Testing Overview

Agile Testing Overview

Philosophy Of Agile Testing:

- Agile testing is a style of testing, one with lessened reliance on documentation, increased acceptance of change, and the notion that a project is an ongoing conversation about quality.

  Brian Marick

  - An early XP proponent
  - a signer of the Agile Manifesto
  - Has provided much of the thought leadership in this area
  - Has developed a framework that many agilists use to think about testing in an agile paradigm.

Agile Testing Overview

He describes two main categories of testing:

1. **Business-Facing Test** is one you could describe to a business expert in terms that would (or should) interest her… . You use words drawn from the business domain: “If you withdraw more money than you have in your account, does the system automatically extend you a loan?”
2. **Technology-Facing Test** is one you describe with words drawn from the domain of the programmers: “Different browsers implement JavaScript differently, so we test whether our product works with the most important ones.”

Agile Testing Overview

- Tests that **support programming** mean that the programmers use them as an integral part of programming.
  - For example, some programmers write a test to tell them what code to write next.
  - Running the test after the [code] change reassures them that they changed what they wanted.
  - Running all the other tests reassures them that they didn't change behavior they intended to leave alone.
- Tests that **critique the product** are not focused on the act of programming. Instead, they look at a finished product with the intent of discovering inadequacies.

Agile Testing Overview

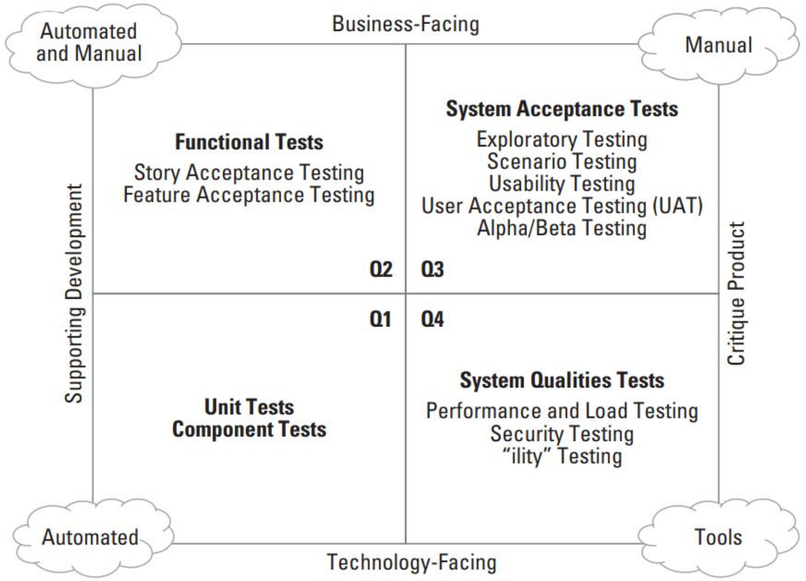
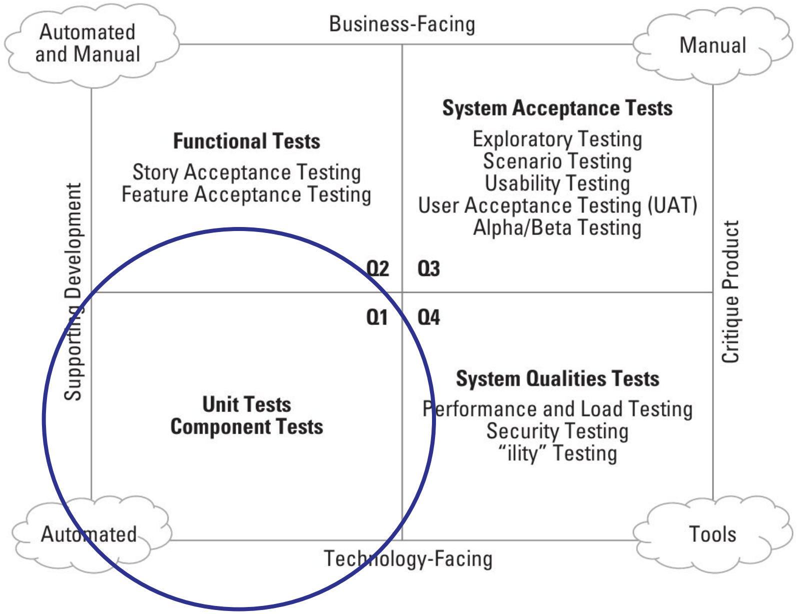


Figure 10–1 The agile testing matrix

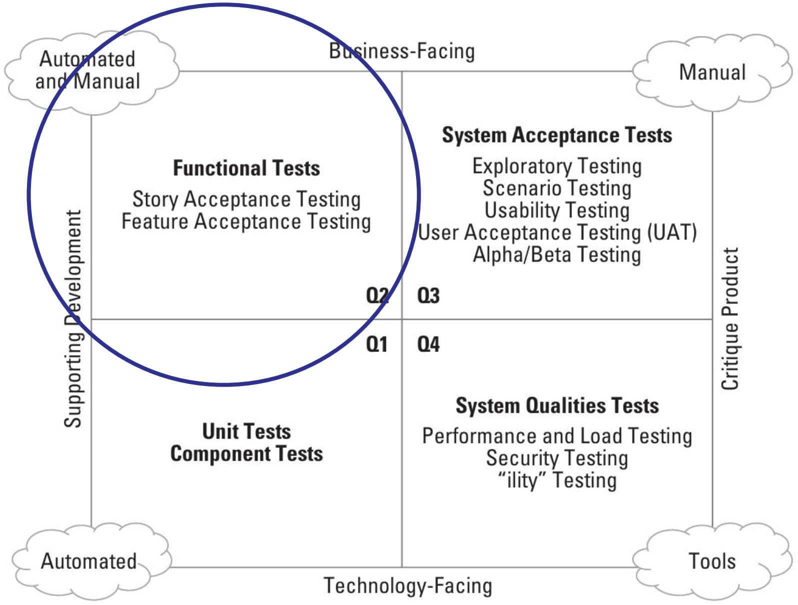
Agile Testing Overview

- ➔ These tests are **written** by **developers** to test whether the system does what they intended it to do.
- ➔ These tests are primarily **automated**, because there will be a very large number of them, and they can be implemented in the unit testing environment of choice.



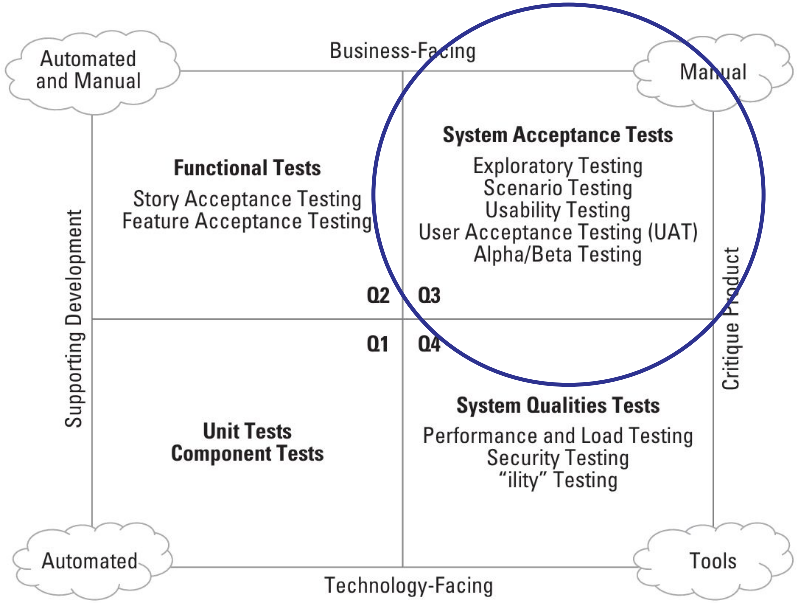
Agile Testing Overview

- These consist primarily of the **story-level acceptance tests** that the teams use to validate that each new story works the way the product owner (customer, user) intended.
- **Feature-level acceptance testing** is referenced in this quadrant as well.
- Many of these tests can be **automated**, but some of these tests are likely to be **manual**.



Agile Testing Overview

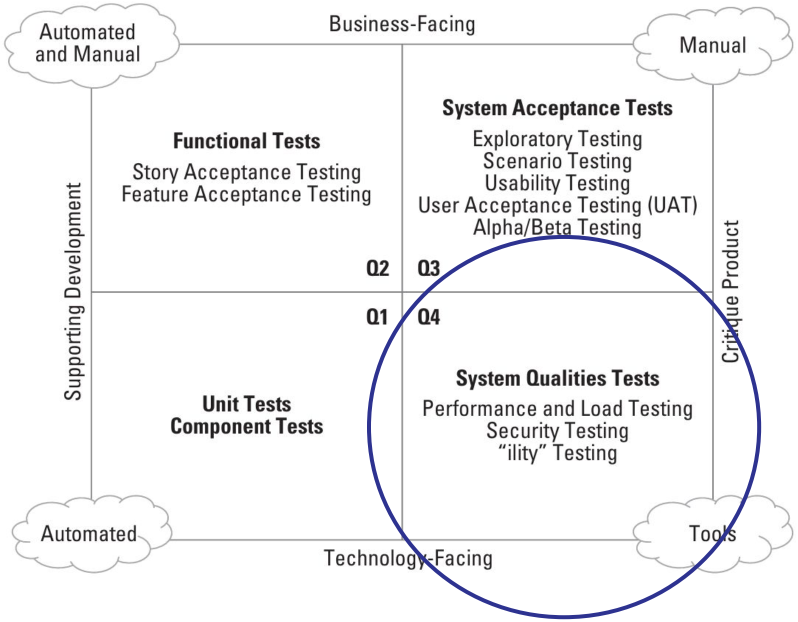
- ➔ System acceptance tests are **system-level tests** to determine whether the aggregate behavior of the system meets its usability and functionality requirements, including the many **variations** (**scenarios**) that may be encountered in actual use.
- ➔ These tests are largely **manual in nature**, because they involve users and testers using the system in actual or simulated deployment and usage scenarios.



Agile Testing Overview

- In quadrant 4, we find **system qualities tests**, which are used to determine whether the system **meets** its **nonfunctional requirements**.

- Such tests are typically supported by a class of **testing tools**, such as **load** and **performance testing tools**, which are designed specifically for this purpose.

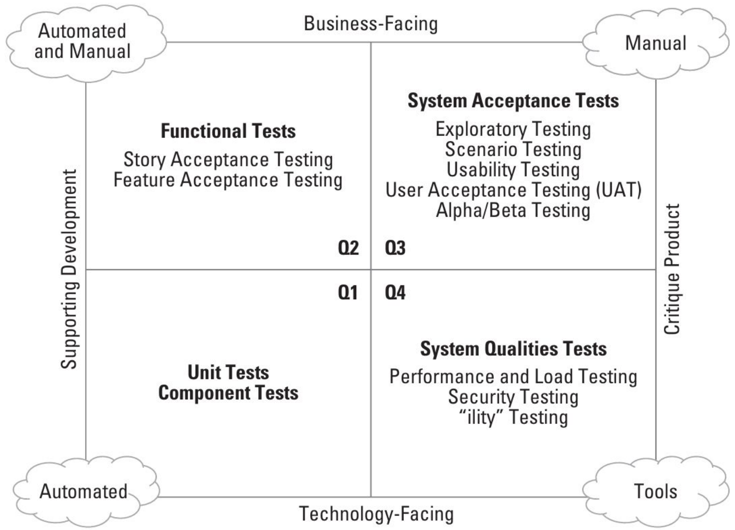


Contents

2. What Is Accepting Testing?

What Is Acceptance Testing?

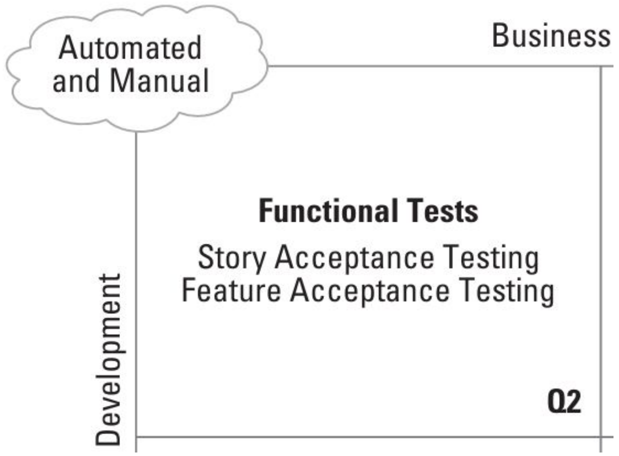
- ➔ Acceptance testing means different things to different people.
- ➔ There are two different uses of the term in the agile testing matrix.
- ➔ In quadrant 2, we see **functional** and **story** and **feature acceptance tests**, and in quadrant 3, we see **system-level acceptance tests**.
- ➔ Our focus is on quadrants Q2 and Q1.



What Is Acceptance Testing?

Story Acceptance Test (SAT)

- ➔ The majority of the testing work done by agile teams is in the development, execution, and regression testing of story acceptance tests (SATs), so we will focus on those first.
- ➔ Story acceptance tests are **functional tests** intended to **assure** that the **implementation** of **each new user story** (or other story type) **delivers** the **intended behavior**.



What Is Acceptance Testing?

Story Acceptance Test (SAT)

➔ If **all** the **new stories work as intended**, then each **new increment** of **software** is **delivering value** and provides assurances that the project is progressing in a way that will ultimately satisfy the needs of the users and business owners.

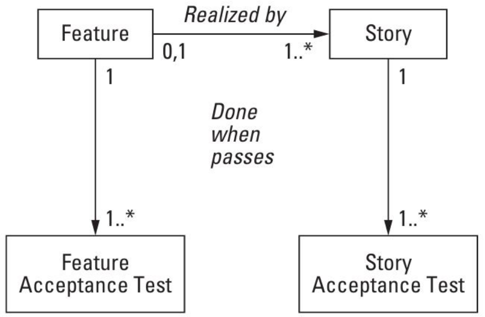


Figure 10–2 Features and stories cannot be considered done until they pass one or more acceptance tests.

What Is Acceptance Testing?

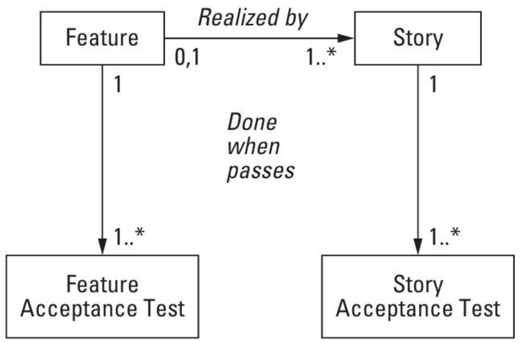
Story Acceptance Test (SAT)

- ➔ They are **written** in the **language** of the **business domain** (business facing tests from quadrant 2).
- ➔ They are **developed** in a **conversation** between the **developers**, **testers**, and **product owner**.
- ➔ Although anyone can write tests, the **product owner**, as **business owner/customer proxy**, is the **primary owner** of the **tests**.
- ➔ They are **black-box tests** in that they **verify only** that the **outputs** of the **system meet** the **conditions** of **satisfaction**, without **concern** for how the result is achieved.
- ➔ They are **implemented during** the **course** of the **iteration** in which the **story itself** is **implemented**.

What Is Acceptance Testing?

Story Acceptance Test (SAT)

- ➔ This means that **new acceptance tests** are **developed** for **every new story**.
- ➔ If a story does not pass its test, the teams get no credit for the story, and the story is carried over into the next iteration, where the code or the test, or both, are reworked until the test passes.



Contents

3. Characteristics Of Good Story Acceptance Test

Characteristics Of Good Story Acceptance Tests

- If **stories** are the **workhorse** of **agile development**—the key proxy artifact that carries the value stream to the customer—then **story acceptance tests** are the **workhorse** of **agile testing**, so teams spend much time defining, refining, and negotiating the details of these tests.
- That's because, in the end, it is the **details** of **these tests** that **define** the **final**, **agreed-to behavior** of the **system**.
- Therefore, writing good story acceptance tests is a prime factor in delivering a quality system.

Characteristics Of Good Story Acceptance Tests

They Test Good User Stories

- ➔ We described the INVEST model for user stories.
- ➔ The **quality** of our **acceptance tests** are **fully dependent** on the **native quality** of the **story** itself.
- ➔ In particular, the user story to which a test is associated has to be independent, small, and testable.
- ➔ If the development of an acceptance test illustrates that the story is otherwise, then the story itself must be refactored until it meets these criteria.

Characteristics Of Good Story Acceptance Tests

They Test Good User Stories

- To get a good acceptance test, we may need to **refactor** the **story first**.
- Example:
  - As a consumer, I am always aware of my current energy costs.

  becomes this:

  - As a consumer, I always see current energy pricing reflected on my portal and on-premise devices so that I know that my energy usage costs are accurate and reflect any utility pricing changes.

Characteristics Of Good Story Acceptance Tests

They Are Relatively Unambiguous And Test All The Scenarios

- ➔ Since the **story** itself is a **lightweight** (even potentially throwaway) **expression** of **intent**, the **acceptance test carries** the **detailed requirements** for the **story**, now and into the future.
- ➔ As such, there can be little **ambiguity** about the details in the story acceptance test.
- ➔ In addition, the **acceptance test** must **test all** the **scenarios** implied by the story. Otherwise, the team won't know when the story is sufficiently complete in order to be able to be presented to the product owner for acceptance.

Characteristics Of Good Story Acceptance Tests

They Are Relatively Unambiguous And Test All The Scenarios

Example:

- **Story:**
  - As a consumer, I always see current energy pricing reflected on my portal and on-premise devices so that I know that my energy usage costs are accurate and reflect any utility pricing changes.

Characteristics Of Good Story Acceptance Tests

They Are Relatively Unambiguous And Test All The Scenarios

Example:

- **Story:**
  - As a consumer, I always see current energy pricing reflected on my portal and on-premise devices so that I know that my energy usage costs are accurate and reflect any utility pricing changes.
- **Acceptance Test 1:**
  - Verify the current pricing is always used and the calculated numbers are displayed correctly on the portal and each on-premise device (see attachment for formats).

Characteristics Of Good Story Acceptance Tests

They Are Relatively Unambiguous And Test All The Scenarios

Example:

- **Story:**
    - As a consumer, I always see current energy pricing reflected on my portal and on-premise devices so that I know that my energy usage costs are accurate and reflect any utility pricing changes.
- **Acceptance Test 2:**
    - Verify the pricing and the calculated numbers are updated correctly when the price changes.

Characteristics Of Good Story Acceptance Tests

They Are Relatively Unambiguous And Test All The Scenarios

Example:

- **Story:**
  - As a consumer, I always see current energy pricing reflected on my portal and on-premise devices so that I know that my energy usage costs are accurate and reflect any utility pricing changes.
- **Acceptance Test 3:**
  - Verify the “current price” field itself is updated according to the scheduled time.

Characteristics Of Good Story Acceptance Tests

They Are Relatively Unambiguous And Test All The Scenarios

Example:

- **Story:**
  - As a consumer, I always see current energy pricing reflected on my portal and on-premise devices so that I know that my energy usage costs are accurate and reflect any utility pricing changes.
- **Acceptance Test 4:**
  - Verify the info/error messages when there is a fault in the pricing (see approved error messages attached).

Characteristics Of Good Story Acceptance Tests

They Persist

- “If developers don’t document much and there are no software requirements specifications as such, how are we supposed to keep track of what the system actually does?
- After all, we are the ones responsible for assuring that it actually works, now and in the future.
- Isn’t that something we have to know, not just once, but in perpetuity?”
- “The answer is yes, indeed, we do have to know how it works, and we have to routinely regression test it to make sure it continues to work. We do that primarily by persisting and automating (wherever possible) acceptance tests and unit tests.

Characteristics Of Good Story Acceptance Tests

They Persist

➔ User stories can be safely thrown away after implementation.

➔ That keeps them lightweight, keeps them team friendly, and fosters negotiation,

➔ but acceptance tests persist for the life of the application.

➔ We have to know that the current price field didn’t just get updated once when we tested it but that it gets updated every time the price changes, even when the application itself has been modified.

Contents

4. Acceptance Test Template

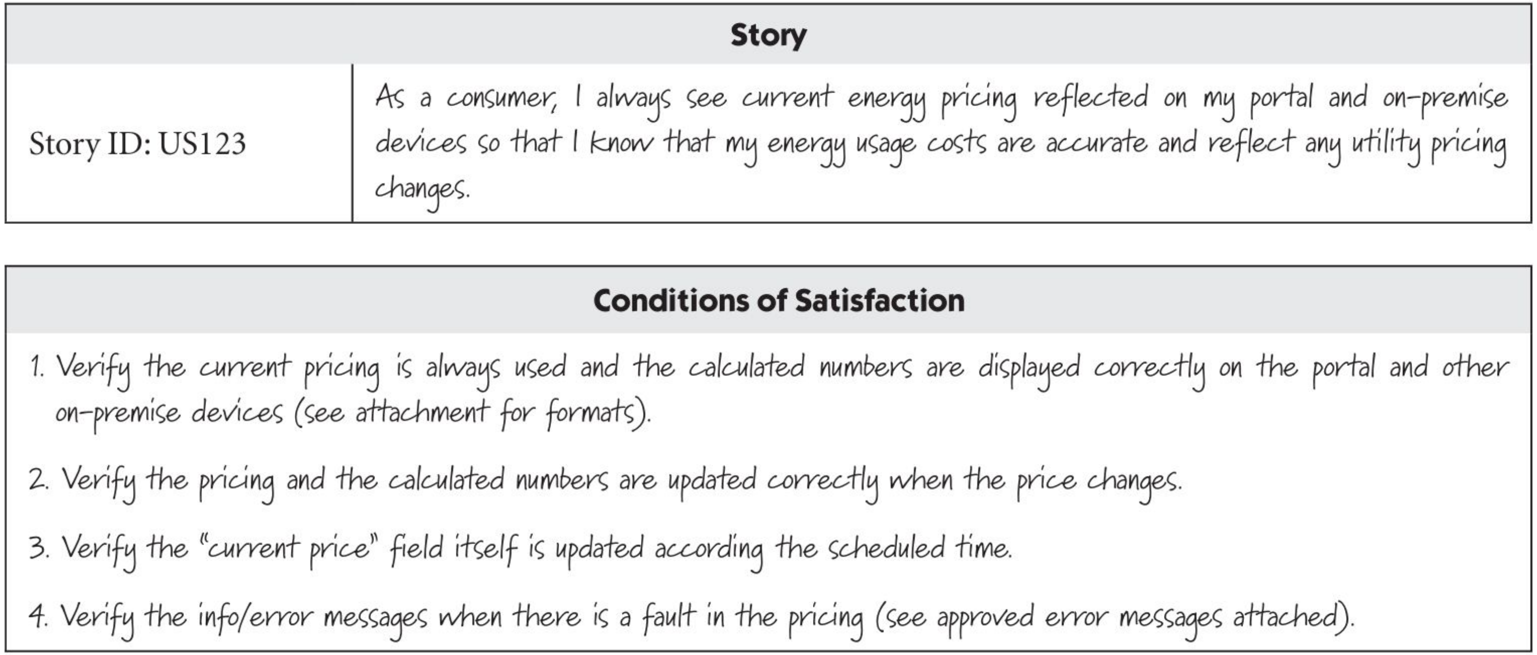
Acceptance Test Template

- At each iteration boundary or whenever a story is to be implemented, it comes as no surprise to the team that they need to **create** an **acceptance test** that **further refines** the **details** of a **new story** and **defines** the **conditions** of **satisfaction** that will tell the team **when** the **story** is **ready** for **acceptance by** the **product owner**.
- In addition, in the context of a team and a current iteration, the domain of the story is pretty well established, and certain patterns of activities result, which can guide the team to the work necessary to get the story accepted into the baseline.

Acceptance Test Template

- To assist in this process, it can be convenient to the team to have a checklist—a simple list of things to consider—to fill out, review, and discuss each time a new story appears.
- Crispin and Gregory [2009] provide an example of such a story acceptance testing checklist in their book.
- Based on our experience using this checklist, we provide an example from the case study in the table of next slide.

Acceptance Test Template



Story	
Story ID: US123	As a consumer, I always see current energy pricing reflected on my portal and on-premise devices so that I know that my energy usage costs are accurate and reflect any utility pricing changes.

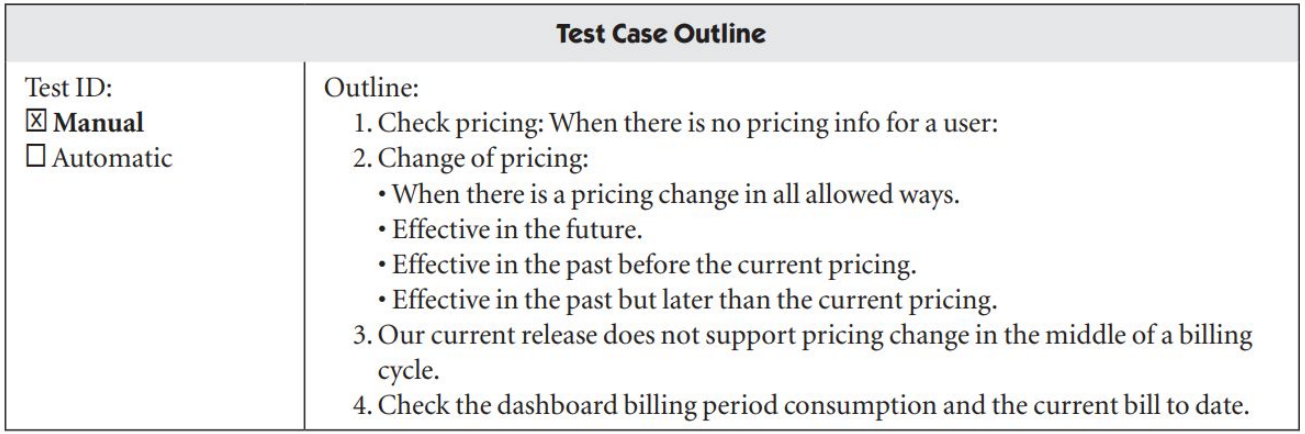
Conditions of Satisfaction
1. Verify the current pricing is always used and the calculated numbers are displayed correctly on the portal and other on-premise devices (see attachment for formats).
2. Verify the pricing and the calculated numbers are updated correctly when the price changes.
3. Verify the "current price" field itself is updated according the scheduled time.
4. Verify the info/error messages when there is a fault in the pricing (see approved error messages attached).

Acceptance Test Template

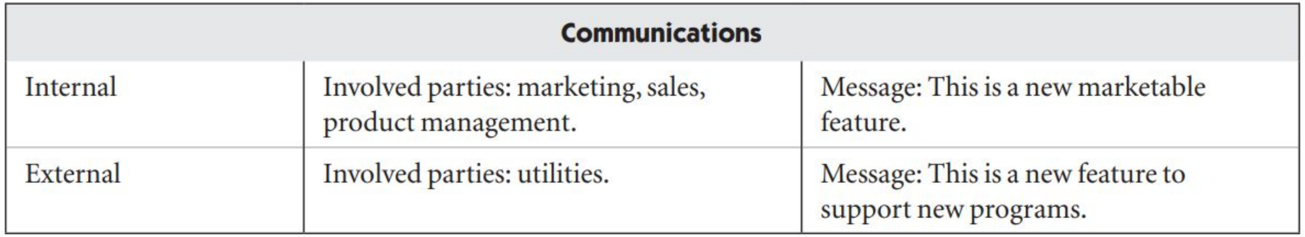
Modules Impacted	
Pricing RESTlet API	Impact: Amend protocol to allow pricing data.
In-home display	Impact: Refactor pricing schedule to support pricing programs to display on the in-home display.
Portal	Impact: Refactor pricing schedule to support pricing programs to display on the portal.

Documents Impacted	
User guide	Impact: Add new section on pricing.
Online help	Impact: Update online help to reflect pricing programs.
Release notes	Impact: Document defects in release notes.
Utility guide	Impact: Document pricing schedule changes.

Acceptance Test Template



Test Case Outline	
Test ID: ☒ **Manual** ☐ Automatic	Outline: 1. Check pricing: When there is no pricing info for a user: 2. Change of pricing: • When there is a pricing change in all allowed ways. • Effective in the future. • Effective in the past before the current pricing. • Effective in the past but later than the current pricing. 3. Our current release does not support pricing change in the middle of a billing cycle. 4. Check the dashboard billing period consumption and the current bill to date.



Communications		
Internal	Involved parties: marketing, sales, product management.	Message: This is a new marketable feature.
External	Involved parties: utilities.	Message: This is a new feature to support new programs.

Acceptance Test Template

➔ Since each team is in a different context, their templates will differ, but the simple act of creating a template as a reminder of all the things to think about benefits the team and increases the velocity with which they can further elaborate and acceptance test a new story.

Contents

5. Automated Acceptance Testing

Automated Acceptance Testing

- ➔ Because **acceptance tests run** at a **level above** the **code**, there are a variety of approaches to executing these tests, including **manual tests.**
- ➔ However, manual tests pile up very quickly (the faster you go, the faster they grow), and eventually, the number of **manual tests required** to **run a regression slows down the team** and **introduces delays** in the **value stream**.
- ➔ To avoid this problem, most **teams** know that they **have to automate most** of **their acceptance tests**.
- ➔ They use a variety of tools to do so, including database-driven tests, Web UI testing tools, and automated tools for record and playback.

Automated Acceptance Testing

- However, **many agile teams** have **discovered** that **some** of **these methods** are **labor-intensive** and can be somewhat brittle and **difficult** to **maintain** because they often **couple** so tightly to the **specific implementation**.
- A better approach is to take a **higher level** of **abstraction** that **works directly against** the **business logic** of the **application** and one that is not encumbered by the presentation layer or other implementation details.
- One such method is the Framework for Integrated Tests (FIT) method.

Automated Acceptance Testing

- ➔ During the course of **each iteration**, **new acceptance tests** are **developed** and **validated** for **each new story in** the **iteration**, and **these tests** are **then added** to the **regression test suite**.
- ➔ These **suites of acceptance tests** can be **run automatically against** the **system** under test at **any time** to **assure** that the **build** is **not broken by** the **new code**.
- ➔ As always, the team's **goal** is to **develop** and **automate tests within** the **course** of the **iteration** in which the **new functionality** is **introduced**.
- ➔ In **some application domains**, **teams** must build **custom frameworks** that **mirror** this **approach** but work in the technologies of their implementation.

Automated Acceptance Testing

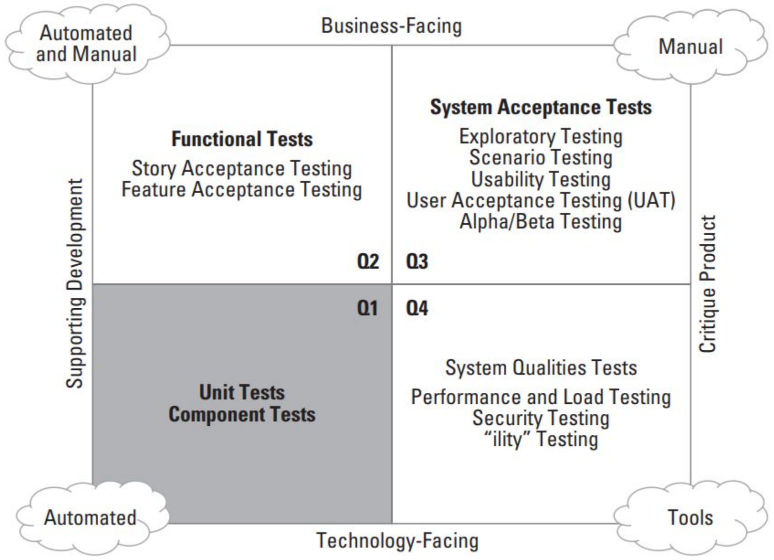
- ➔ In any case, **whatever can’t** be **automated eventually slows** the **team down**, so **continuous investments** in **testing automation infrastructure** are **routine items** on a **team’s backlog**, and like any other backlog item, they **must be** appropriately **prioritized** by the **product owner** to achieve **sustainable high velocity**.

Contents

6. Unit And Component Testing

Unit And Component Testing

- In quadrant 1, we see unit tests and component tests, which are technology-facing tests as they are implemented and executed by the development team.
- Together with feature and story acceptance tests, they also complete the support development side of the agile testing picture.



Unit Testing

- ➔ Unit testing is the **white-box testing** whereby **developers write test code** to **test** the **production code** they **developed** for the **system**.
- ➔ In so doing, the **understanding** of the **user story** is **further refined**, and **additional details** about a **user story** can be **found in** the **unit tests** that **accompany** the **code**.

Example 1:

- ➔ The presentation syntax and range of legal values for current price field can likely be found in the unit tests, rather than the acceptance tests, because otherwise the acceptance tests are long, are unwieldy, and cause attention to the wrong level of detail.

Unit Testing

➔ In so doing, the **understanding** of the **user story** is **further refined**, and **additional details** about a **user story** can be **found in** the **unit tests** that **accompany** the **code**.

Example 2:

➔ Imagine you're building a calculator app. A unit test might check that when you add 2 + 2, the program correctly returns 4. This kind of detail belongs in unit tests, not in acceptance tests, because acceptance tests should focus on bigger goals like "the calculator can perform basic arithmetic operations," rather than every tiny input/output combination.

Unit Testing

- ➔ In so doing, the **understanding** of the **user story** is **further refined**, and **additional details** about a **user story** can be **found in** the **unit tests** that **accompany** the **code**.

Example 3:

- ➔ The rules about how many items a customer can add to their shopping cart — such as preventing negative quantities or limiting the maximum number of units per product — can likely be found in the unit tests, rather than the acceptance tests. Otherwise, the acceptance tests become too long, too detailed, and shift focus away from the bigger goal of "a customer can successfully purchase items."

Unit Testing

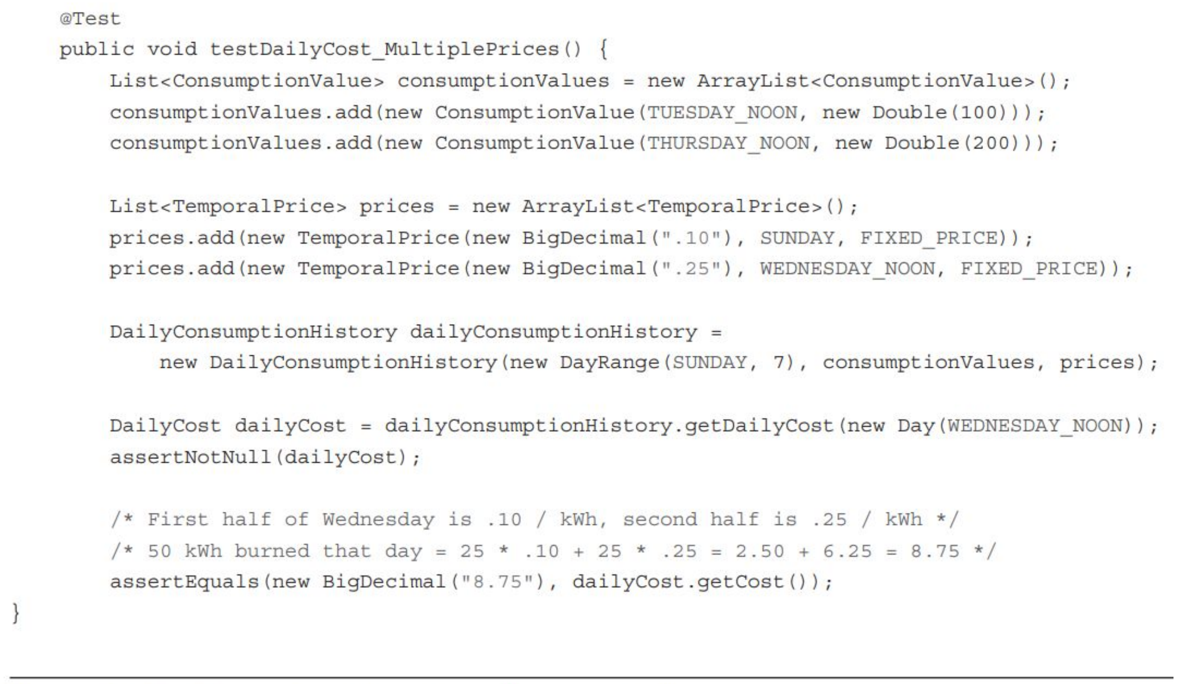
- A **comprehensive unit test strategy prevents QA** and **test personnel** from spending most of their time finding and reporting on **code-level bugs** and allows the team to **move its focus to more system-level testing challenges**.
- Indeed, for many agile teams, **the addition of a comprehensive unit test strategy** is a key pivot point in their **move toward true agility**—and one that delivers the "best bang for the buck" in determining overall system quality.

Unit Testing

Unit Testing In The Course of An Iteration

- ➔ Because the **unit tests** are written **before** or **concurrently with** the **code** and because the **unit testing frameworks** include **test execution automation**, all unit testing can be **accomplished within the iteration**.
- ➔ Moreover, the **unit test frameworks hold** and **manage** the **accumulated unit tests**, so **regression testing automation** for unit tests is **largely free** for the **team**.
- ➔ Unit testing is a cornerstone practice of software agility, and **any investments** a **team makes toward** more **comprehensive unit testing** will be well **rewarded** in **quality** and **productivity.**

Unit Testing



```
    @Test
    public void testDailyCost_MultiplePrices() {
        List<ConsumptionValue> consumptionValues = new ArrayList<ConsumptionValue>();
        consumptionValues.add(new ConsumptionValue(TUESDAY_NOON, new Double(100)));
        consumptionValues.add(new ConsumptionValue(THURSDAY_NOON, new Double(200)));

        List<TemporalPrice> prices = new ArrayList<TemporalPrice>();
        prices.add(new TemporalPrice(new BigDecimal(".10"), SUNDAY, FIXED_PRICE));
        prices.add(new TemporalPrice(new BigDecimal(".25"), WEDNESDAY_NOON, FIXED_PRICE));

        DailyConsumptionHistory dailyConsumptionHistory =
            new DailyConsumptionHistory(new DayRange(SUNDAY, 7), consumptionValues, prices);

        DailyCost dailyCost = dailyConsumptionHistory.getDailyCost(new Day(WEDNESDAY_NOON));
        assertNotNull(dailyCost);

        /* First half of Wednesday is .10 / kWh, second half is .25 / kWh */
        /* 50 kWh burned that day = 25 * .10 + 25 * .25 = 2.50 + 6.25 = 8.75 */
        assertEquals(new BigDecimal("8.75"), dailyCost.getCost());
}
```

Figure 10–7 A sample unit test from the case study

Component Testing

- ➔ Component testing is used to **test larger-scale components** of the **system**.
- ➔ Many of these components are present in **various architectural layers**, where they **provide services needed by features or other components.**
- ➔ **Testing tools** and **practices** for **implementing component tests vary** according to the **nature** of the **component**.
    - ◆ For example, unit testing frameworks can hold arbitrarily complex tests written in the framework language (Java, C, and so on), so many teams use their unit testing frameworks to build component tests. They may not even think of them differently.

Component Testing

- **Acceptance testing frameworks**, especially those at the level of http Unit and XML Unit, are also employed.
- In other cases, developers may use **testing tools** or **write fully custom tests** in **any language** or **environment** that is **most productive** for them.

End of Chapter 10

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.