Lesson 16

RUP and the Agile Approach to Software Development

Purification Leads To Progress

RUP and Agile
What is the difference?

The Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

1. **Individuals and interactions over Processes and tools**
2. **Working software over Comprehensive documentation**
3. **Customer collaboration over Contract negotiation**
4. **Responding to change over Following a plan**

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck James Grenning Robert C. Martin
Mike Beedle Jim Highsmith Steve Mellor
Arie van Bennekum Andrew Hunt Ken Schwaber
Alistair Cockburn Ron Jeffries Jeff Sutherland
Ward Cunningham Jon Kern Dave Thomas
Martin Fowler Brian Marick

http://www.agilemanifesto.org/ 2001

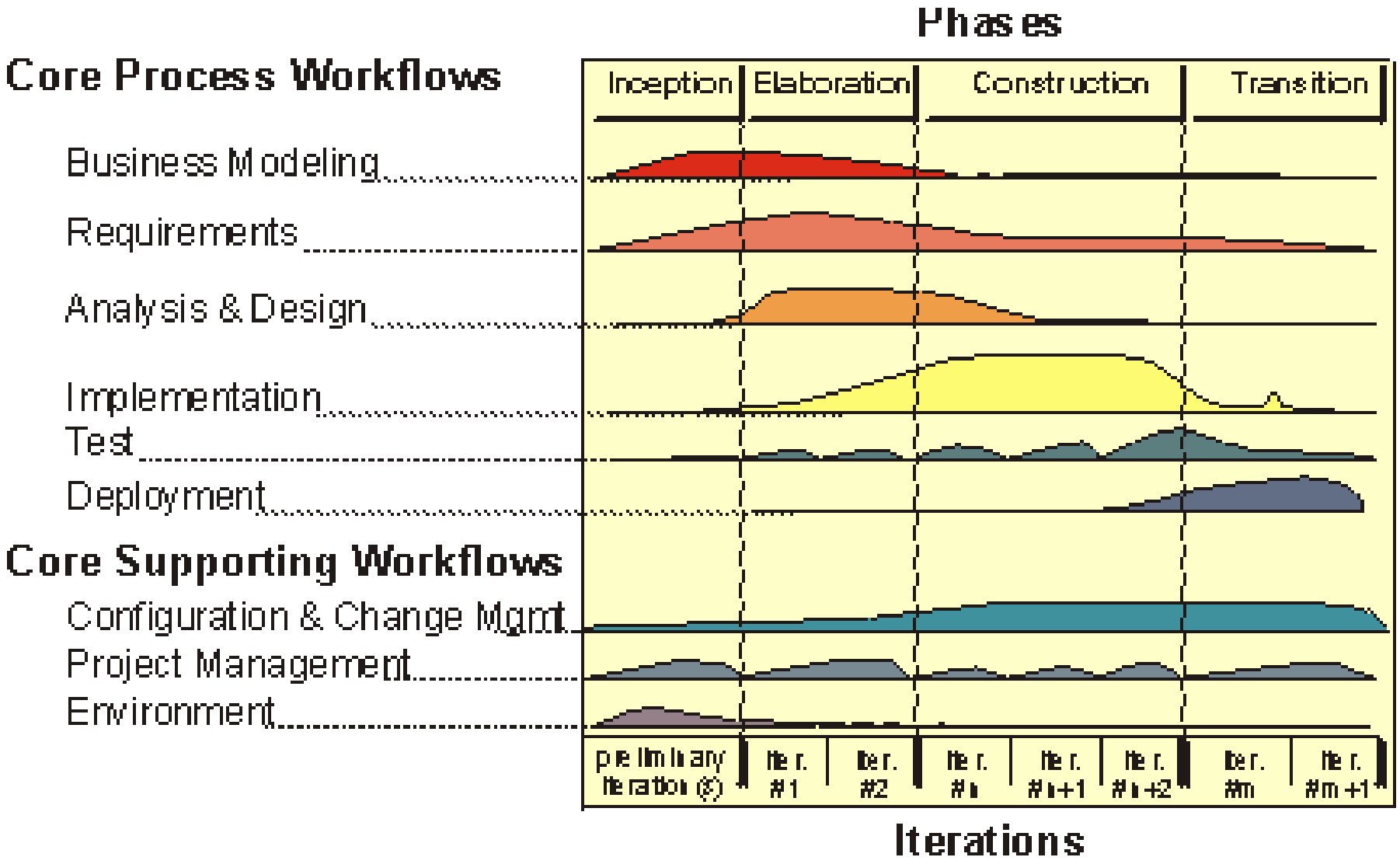
Agile Key Principles

1. Active user involvement is imperative
2. The team must be empowered to make decisions
3. Requirements evolve but the timescale is fixed
4. Capture requirements at a high level; lightweight & visual
5. Develop small, incremental releases and iterate
6. Focus on frequent delivery of products
7. Complete each feature before moving on to the next
8. Apply the 80/20 rule
9. Testing is integrated throughout the project lifecycle – test early and often
10. A collaborative & cooperative approach between all stakeholders is essential

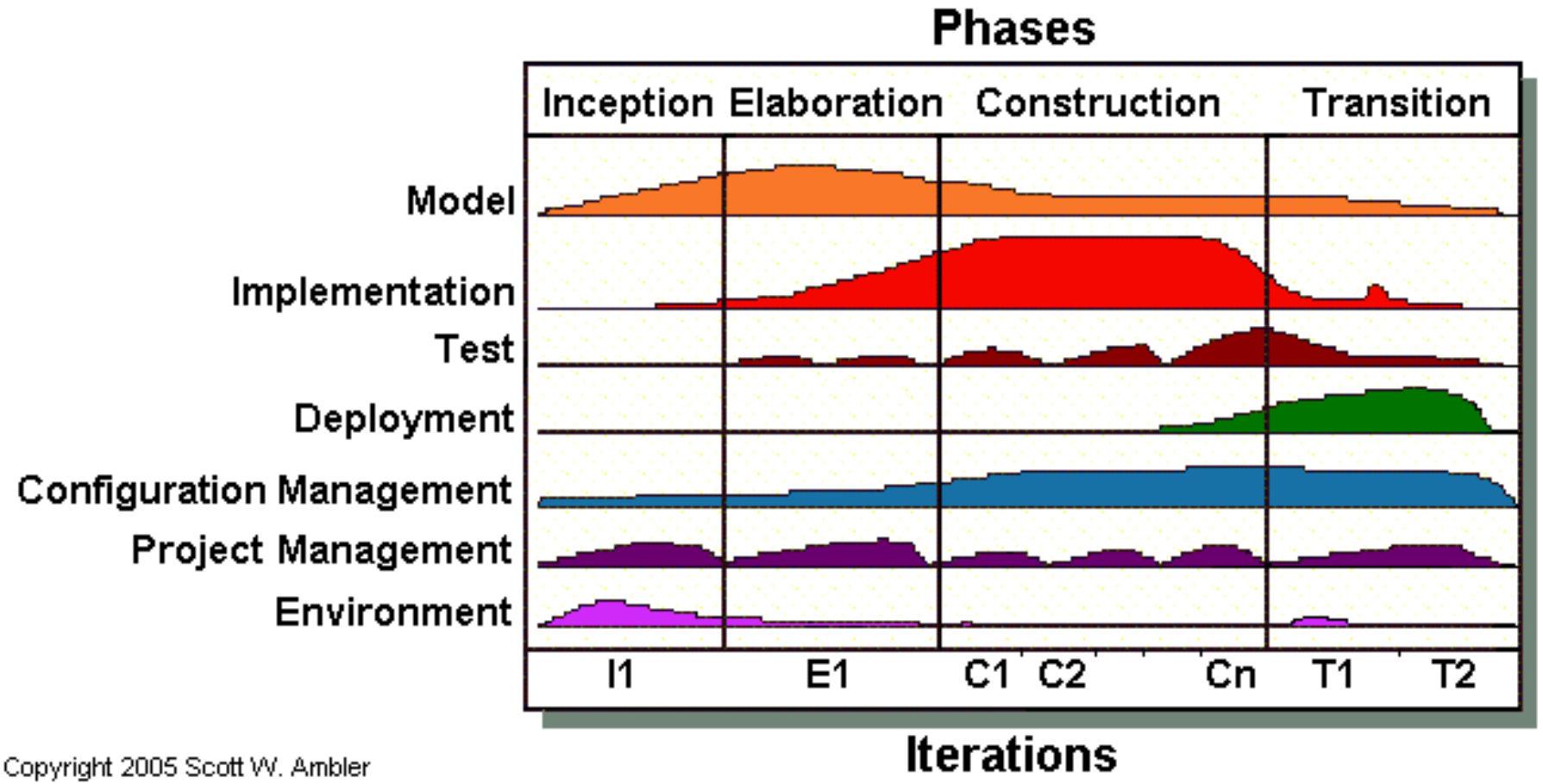
Kelly Waters

http://www.allaboutagile.com/what-is-agile-10-key-principles

Rational Unified Process Phases -Workflow



Agile Unified Process Phases- Workflow



Agile Unified Process Phases- Workflow

- **Model**. The goal of this discipline is to understand the business of the organization, the problem domain being addressed by the project, and to identify a viable solution to address the problem domain.

- **Implementation**. The goal of this discipline is to transform your model(s) into executable code and to perform a basic level of testing, in particular unit testing.

- **Notice that in AUP *Model* replaces RUP:**
  - ***Business Modeling***
  - ***Requirements***
  - ***Analysis and Design***

http://www.ambysoft.com/unifiedprocess/agileUP.html **Ambler**

Agile Process Rapid Iteration

- Pivotal VP of Cloud R&D Onsi Fakhouri discusses Agile SW Development at the August, 2016, SpringOne conference.

  https://www.youtube.com/watch?v=xdw_9dADM-4

- Nordstrom Innovation Lab Agile IPad Development, September, 2011.

  https://www.youtube.com/watch?v=szr0ezLyQHY

Microsoft's* Approach to
Acceptance Test Driven Development (ATDD)

*** Based on slides From Ihab Shaaban**

Goal?

As a developer on a Scrum Team, you want to effectively write code that is correct and delivers the required business value

Enterprise Software Development

What can cause a project to fail?

- Build the Wrong Thing (Our analysis of WHAT we will build was flawed)
- Poor quality or poor performance (Probably HOW we built it was flawed)
- Being Late (Too complex?? Too much confusion??)

Let's examine Microsoft's approach to mitigate each one of these risks.

Building the Wrong Thing

- **Communication Issues**
- *"This isn't what I agreed to"*
- We use documents: Use Cases, UML, Specs
- ***Documents don't work well***
- They're expensive to produce
- Doesn't prove intent to specifications

- **Spec Changes**

Being Late

- Requirements Analysis takes too long
- Development takes too long
- Testing takes too long
- Too many features are built
- Solution too complex

Quality and Performance Issues

- Bugs!!
- UI Problems & Usability
- Response Time

Questions

- How can we make communications more clear?
- How do we adapt with changes?
- How can we shorten time required for analysis, development and testing?
- How can we build a high quality product?

SCRUM

- **SCRUM**
- Iterative and incremental
- Team based
- Inspect and Adapt as you go

build less.. deliver faster

- **Agile Development Practices**
- **Acceptance Test-Driven Development**

What is ATDD?

- **Sources of requirements:**
- System requirements come from many sources.
- Business team often has less knowledge about software development
- Requirements are constantly changing
- User Stories: usually used to defer the details, they often do not describe in detail what the system is actually doing.
- There are always a gap between user stories and code.
- User stories start the discussion, what happens after discussion?

Prove Requirements Understanding

- Write code and depend on SCRUM feedback
- Ask questions to validate assumptions
- Write requirements using different documents

- Or…

- Use ATDD

What is ATDD?

- ATDD, is -- ***understanding and stating back the requirements as tests***
- Consists of Automated tests written and failing before the code is complete
- Tests built on specific examples of expected behavior
- After acceptance tests pass, they don't fail.
- Start with a User Story and create scenarios that provide detailed behavior of the system. ***How many scenarios per use case?***
- Write one acceptance test for each scenario.

Try ATTD

Each person pick one of their use cases and redo the use case description like a user story as used with ATDD.

We'll use the following example.

How to Create Acceptance Tests?

Example

“Cancel an Order in an Ecommerce system”

Example Building Requriements

- **User story**
  Represents requirements in the following format:
  - ***As a*** <application role>
  - ***I want*** <some feature>
  - ***So that*** <I get some benefit>
- **Scenario**
  To document the conversation
  - ***Given*** <initial context>
  - ***When*** <this happens>
  - ***Then*** <this should be the result>

Example Requirements: (Cont.)

- **User story : Cancel an Order**
  - As an administrator
  - I want to cancel placed orders
  - So that they no longer exist in the system
- **Scenario**

  Cancel order from admin order detail page
  - Given I am viewing order detail
  - When I click cancel order
  - Then order is marked as cancelled *and* doesn't appear in the order list page

Difference From Use Case Definitions?

- We create "***Executable Specifications***"
- Requirements are represented as automated tests

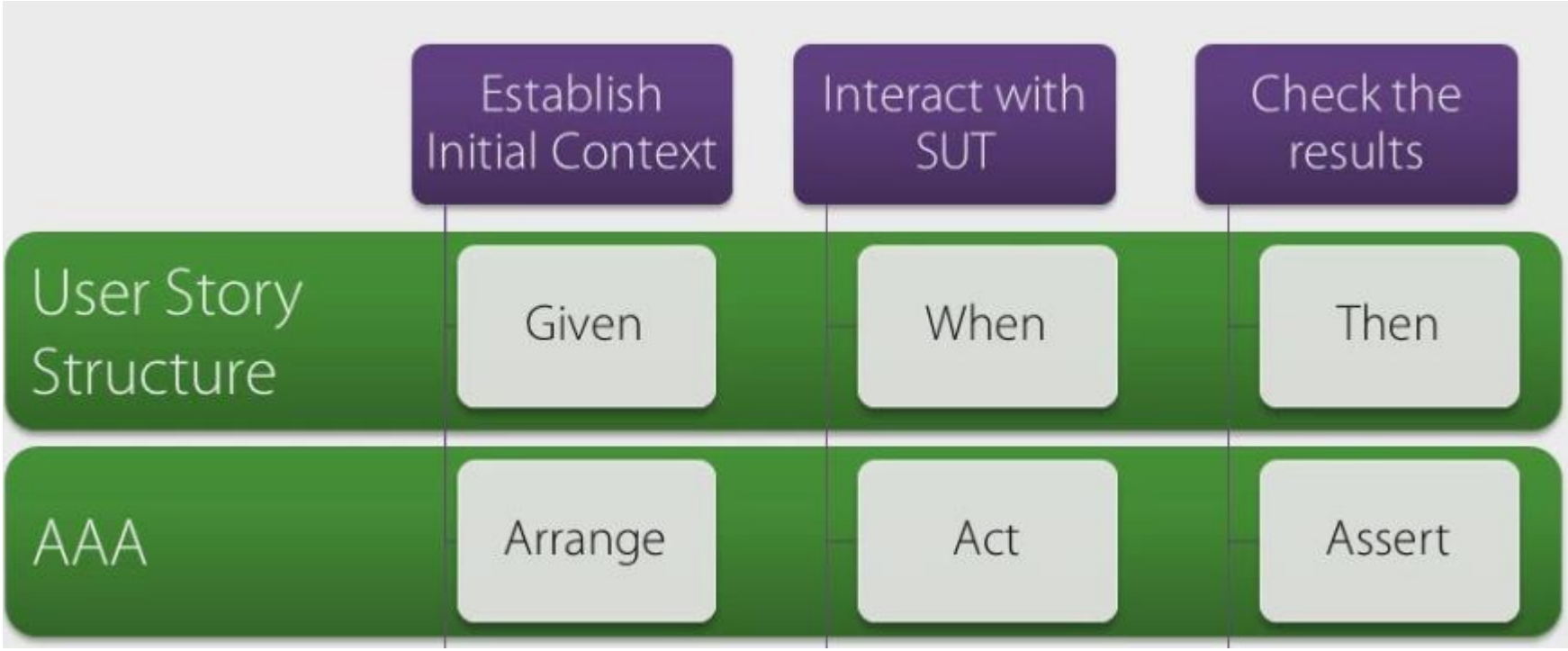
Building Requirements

- **Let's map a User story and scenario to a RUP use case description**
- **User story**
  - ***As a*** <application role>
  - ***I want*** <some feature>
  - ***So that*** <I get some benefit>
- **Scenario**
  - ***Given*** <initial context>
  - ***When*** <this happens>
  - ***Then*** <this should be the result>

Building Requirements

- **Let's map a User story and scenario to a RUP use case description**
- **User story**
  - ***As a*** <application role> **RUP Actor**
  - ***I want*** <some feature> **RUP Short Description**
  - ***So that*** <I get some benefit> **RUP Post-condition**
- **Scenario**
  - ***Given*** <initial context> **RUP Pre-condition**
  - ***When*** <this happens> **RUP User input**
  - ***Then*** <this should be the result> **RUP System output**

Create Automated Tests



Create Automated Tests (Cont.)

1, 2, 3

```
[TestMethod]
public void
Total_Should_Be_1_When_1_Product_Added()
{
                    var cart = new ShoppingCart("TEST");

                   cart.AddItem(new Product("SKU"));

                  Assert.AreEqual(1, cart.TotalItems);
}
```

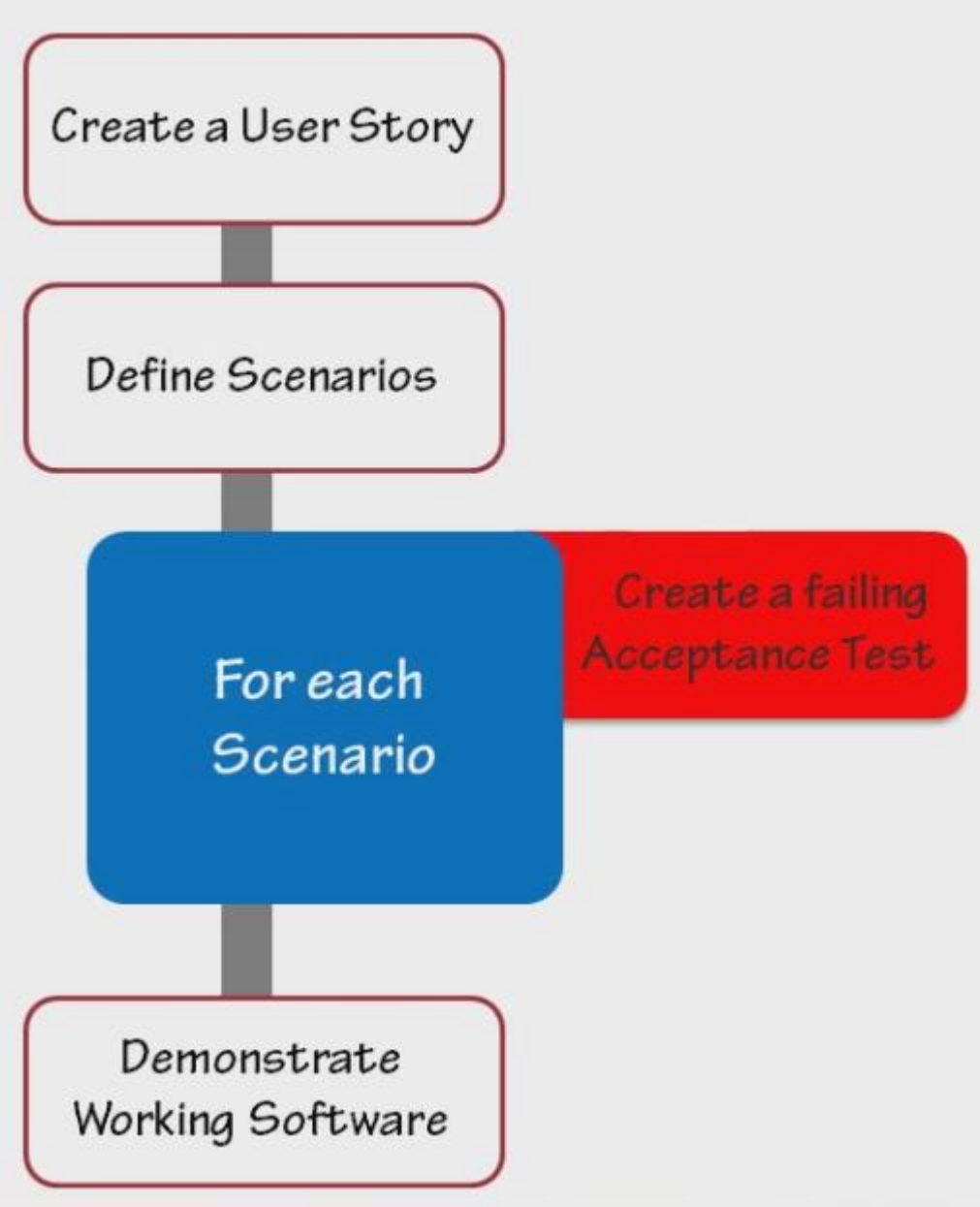
Create Automated Tests (Cont.)

1, 2, 3

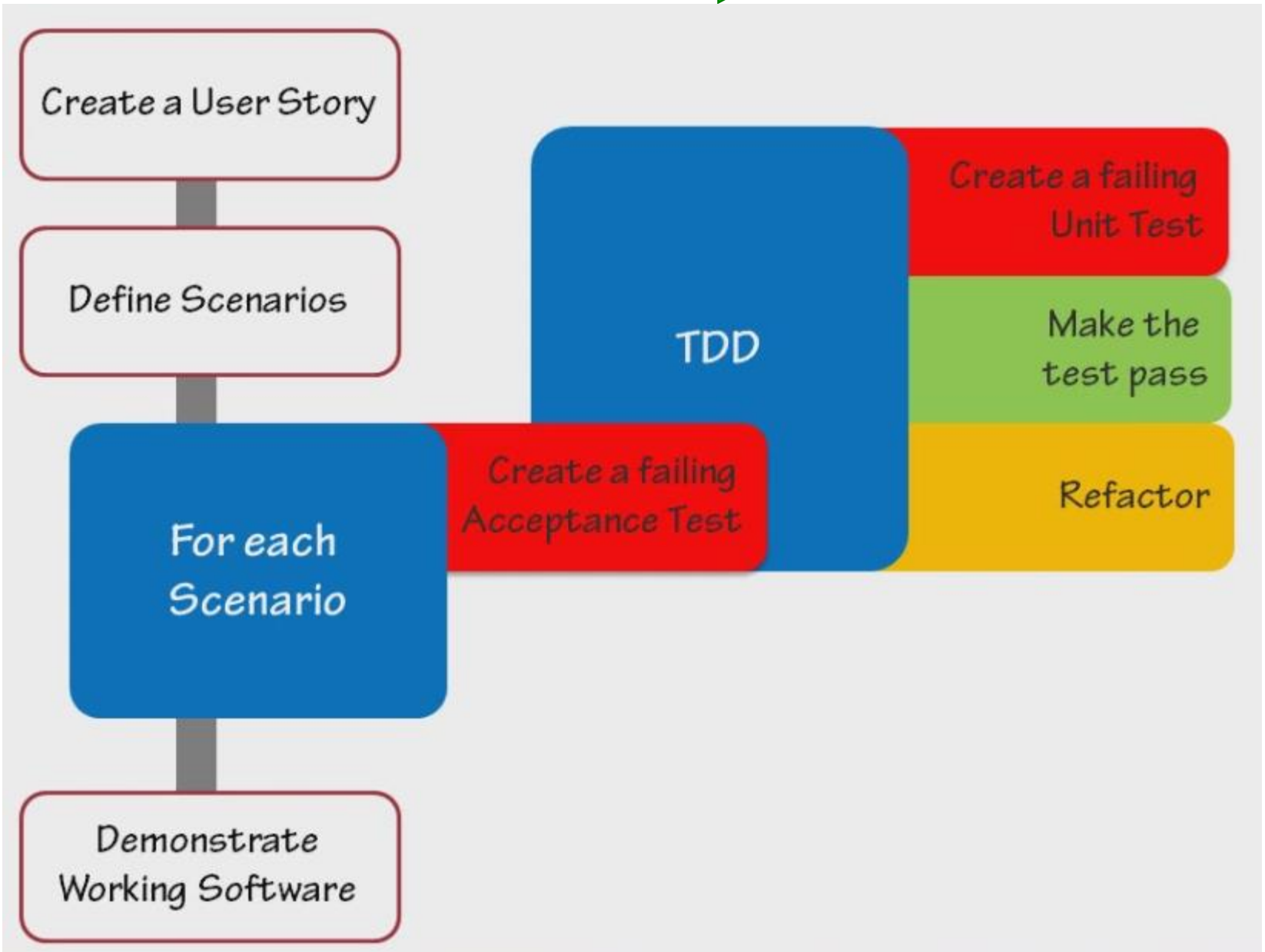
```
[TestMethod]
public void
Total_Should_Be_1_When_1_Product_Added()
{
    Given   var cart = new ShoppingCart("TEST");

    When    cart.AddItem(new Product("SKU"));

    Then    Assert.AreEqual(1, cart.TotalItems);
}
```

Cycle

Cycle (Cont.)


Create a User Story
Define Scenarios
For each Scenario
Create a failing Acceptance Test
Demonstrate Working Software

Cycle (Cont.)

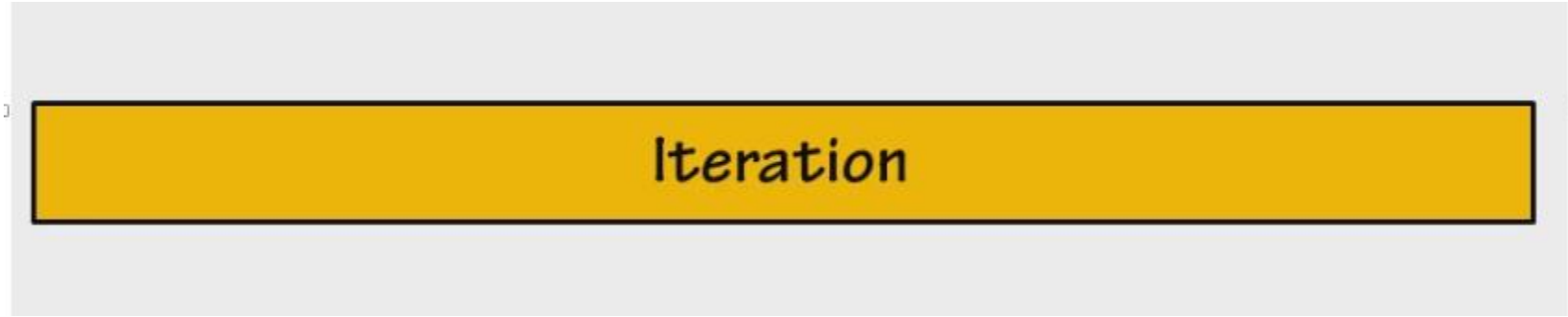

Create a User Story
Define Scenarios
For each Scenario
Demonstrate Working Software
TDD
Create a failing Acceptance Test
Create a failing Unit Test
Make the test pass
Refactor

Try ATTD

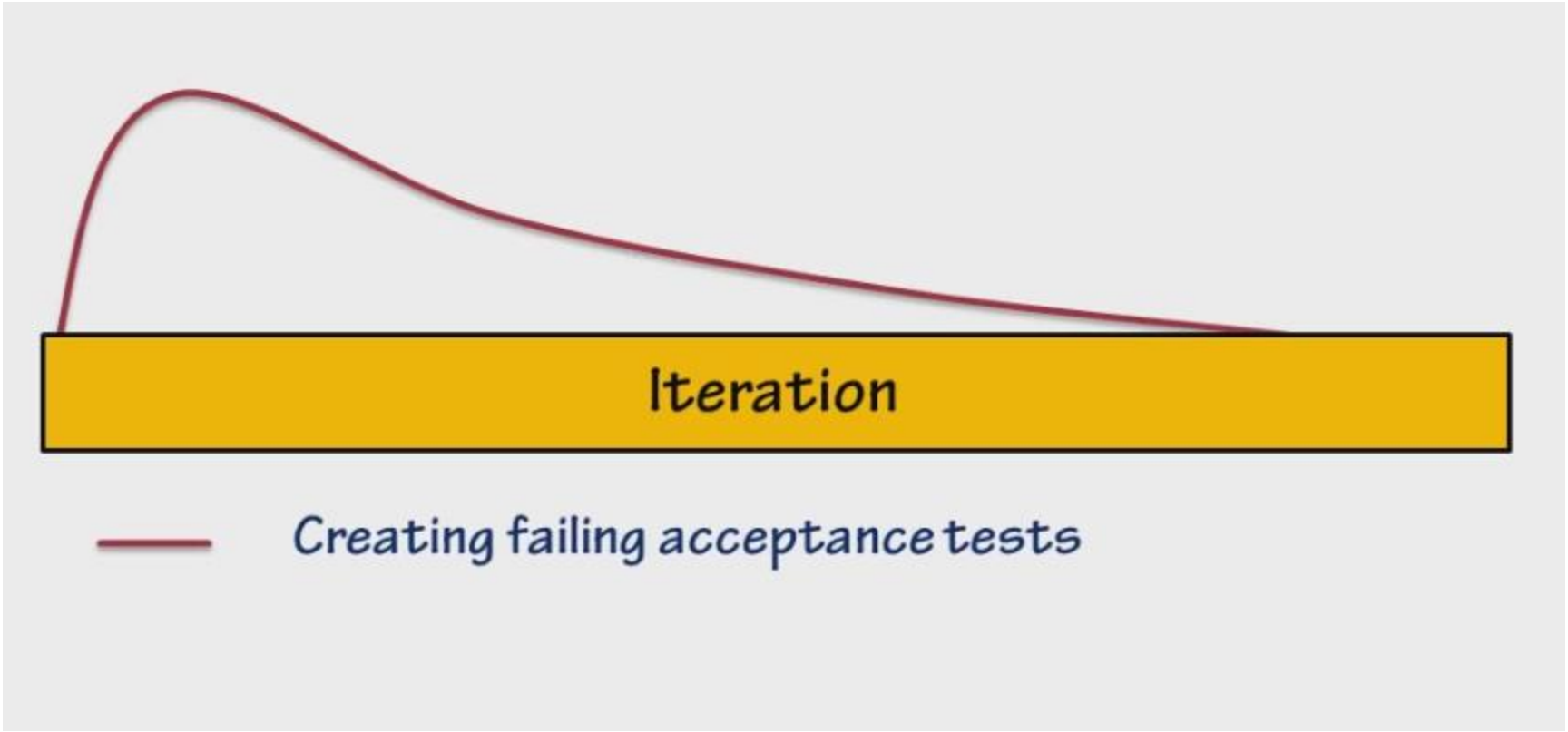
Each person take one of their ATDD User story scenarios and write a test following

Arrange --Act--Assert

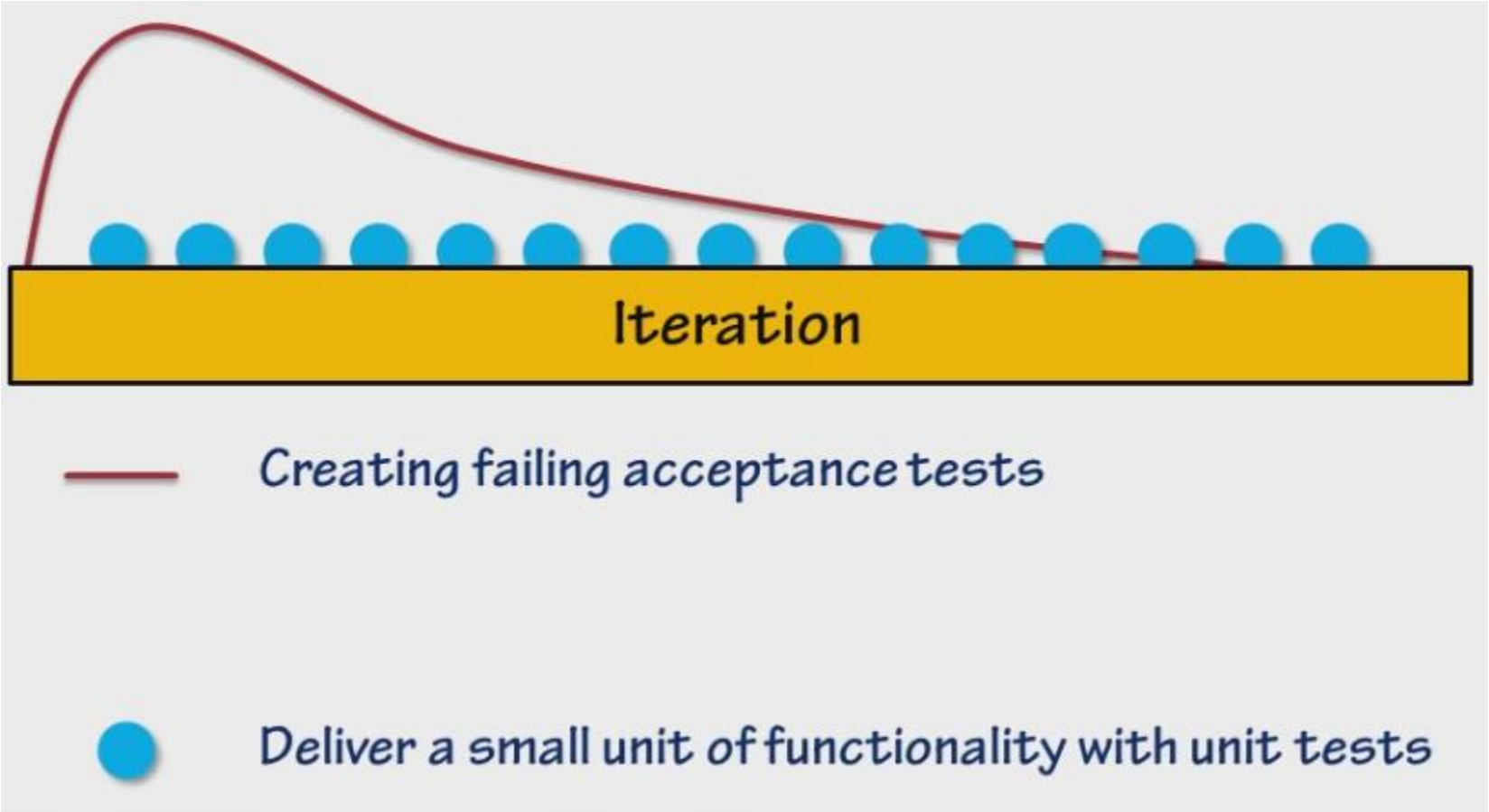
Development Iteration Cycle



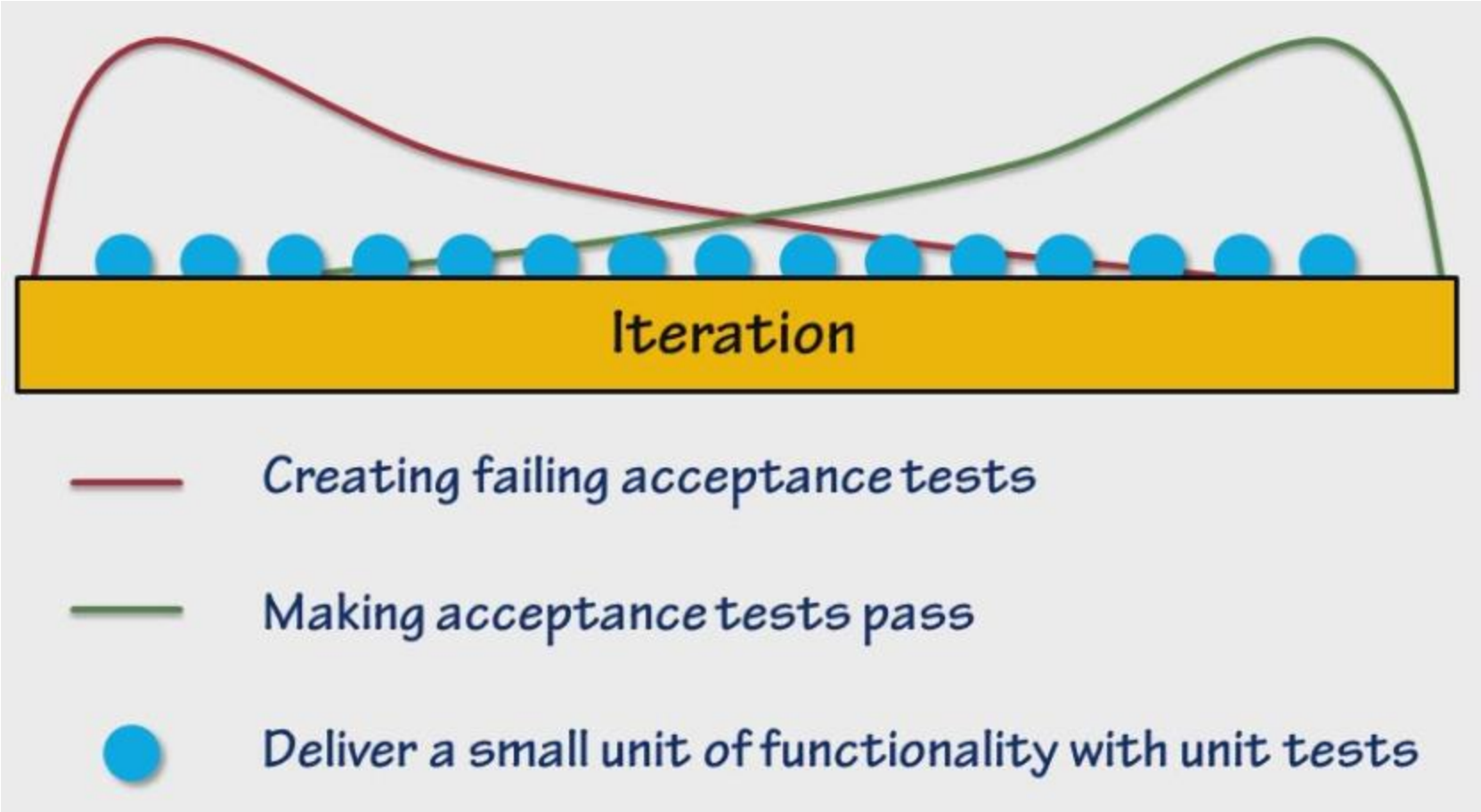
Development Iteration Cycle (Cont.)



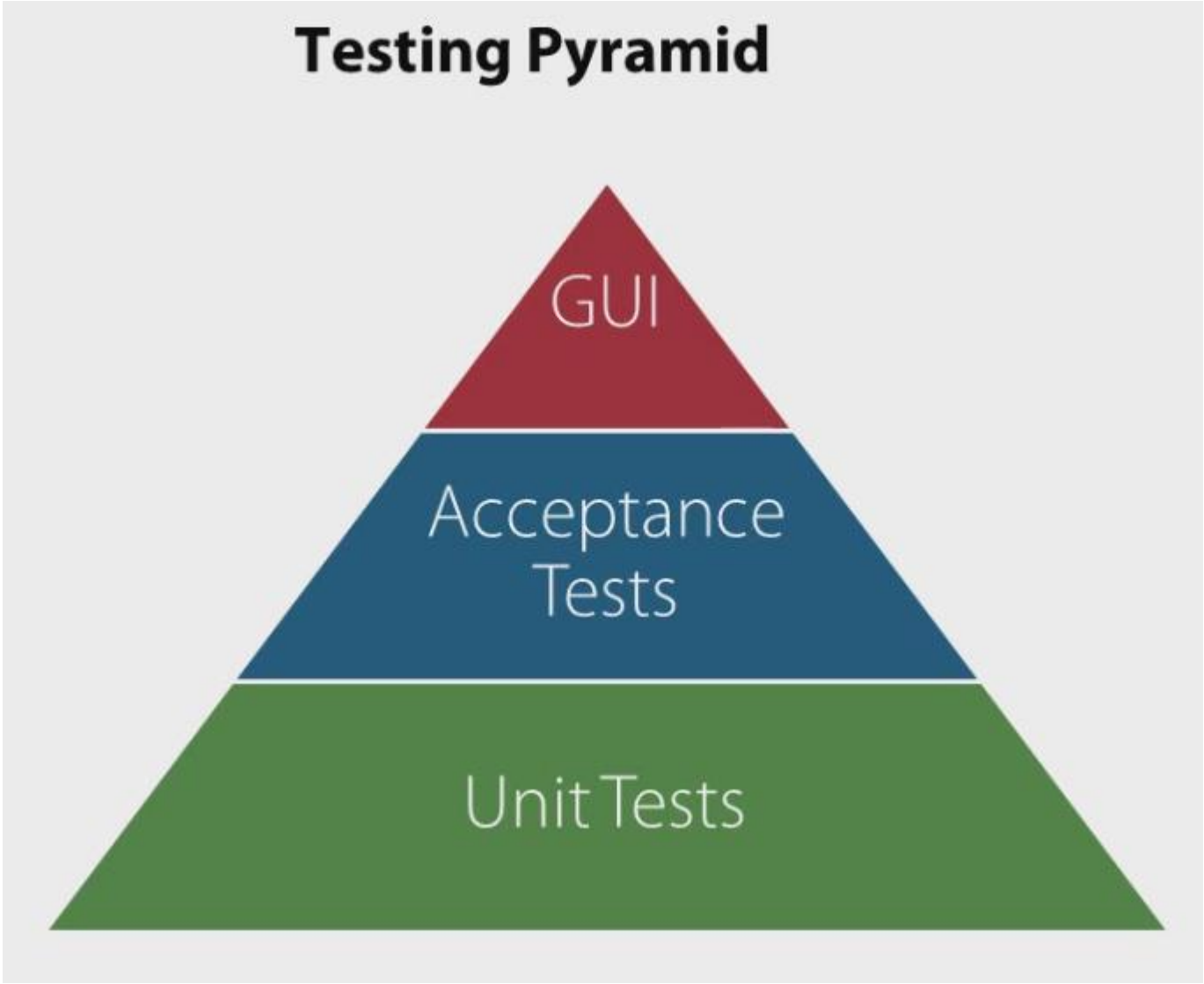
Development Iteration Cycle (Cont.)



Development Iteration Cycle (Cont.)



Where Do Acceptance Tests Fit?



User Story Automated Testing Example

- Gherkin is a language for the Cucumber automated testing tool.
- A Gherkin source file follows the Given/When/Then structure of a User Story Scenario.
- For example see: https://github.com/cucumber/cucumber/wiki/Gherkin

User Story Automated Testing Example

An example Gherkin source file from the above link:

1: **Feature**: Some terse yet descriptive text of what is desired
2: Textual description of the business value of this feature
3: Business rules that govern the scope of the feature
4: Any additional information that will make the feature easier to understand
5:
6: **Scenario**: Some determinable business situation
7: **Given** some precondition
8: And some other precondition
9: **When** some action by the actor
10: And some other action
11: And yet another action
12: **Then** some testable outcome is achieved
13: And something else we can check happens too
14:
15: **Scenario**: A different situation ---- and so on------

Review

- What are the problems that ATDD is attempting to solve?
- How does ATDD solve those problems?
- Compare the ATDD approach to defining requirements to our RUP approach.
- How do Acceptance Tests fit with other developer testing?

Summary

Acceptance Tests:

- Are full integration tests
- Are readable by humans and machines
- Evolve with requirements
- Ideally, stand in place of requirement documents
- Continuous Validation

SCI Question

- Would you consider TM to be an agile technique for improved brain performance?