Software Architecture

Lecture 4

Agile Software Development

Néstor Cataño
Innopolis University

Spring 2016

Plan

1. **Agile principles**
2. Scrum
3. eXtreme Programming (XP)

Based on Book "Agile Principles, Patterns and Practices in C#", Slides "Agile Logic" by Paul Hodgetts

Agile software development

- **Diverse approaches**
  - **eXtreme Programming (XP)**,
  - **Scrum**,
  - **Crystal, and others.**
- but **shared values**
  - communication
  - simplicity
  - feedback

What is agile development?

- Agile development is the ability to develop software quickly, in the face of rapidly changing requirements. In order to achieve this agility, we need to use practices that provide the necessary discipline and feedback. We need to employ design principles that keep our software flexible and maintainable, and we need to know the design patterns that have been shown to balance those principles for specific problems.

R. and M. Martin in "Agile Principles, Patterns and Practices in C#"

Agile practices

- "My greatest fear was in adopting a process in which there is no explicit up-front design step. I had always used code as a way to verify that the diagrams were meaningful"

Agile practices

- "One practice of XP was revelation form. Test-first design sounds innocuous when you first hear it. It says to write test cases before you write production code. All production code is written to make failing test cases pass. I was not prepared for the profound ramifications that writing code this way would have. This practice has completely transformed the way I write software, and transformed it for the better."

Working Software VS Comprehensive Documentation

- Software without documentation is a disaster.
- Code is not the ideal medium for communicating the rationale and structure of a system.
- **However**, too much documentation is worse than too little.
  - Huge software documents take a great deal of time to produce and even more time to keep in sync with the code.
  - The two documents that are the best at transferring information to new team members are the **code** and the **team**.

Customer Collaboration VS Contract Negotiation

- A **contract** that specifies the requirements, schedule, and cost of a project is flawed.
- In most cases, the terms it specifies become meaningless long before the project is complete, sometimes even long before the contract is signed!
- The best contracts are those that govern the way the development team and the customer will work together.

Agile Techniques

- Agile techniques
  - pair programming
  - test driven development (TDD)
  - refactoring
  - rotation
  - fungibility

Agile software development

- **Common emphases include**
  - efficient teamwork practices
  - close coordination/collaboration with customers
  - early defect detection and elimination
  - **test-driven development** (**TDD**) → **test first**

Manifesto for Agile Software Development

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

→ **individuals and interactions** over **processes and tools**
→ **working software** over **comprehensive documentation**
→ **customer collaboration** over **contract negotiation**
→ **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

Source: www.agilemanifesto.org

Agile principles

- **Highest priority**: satisfy the customer through early and continuous delivery of valuable software.
- **Welcome changing requirements, event late in development**; harness change for customers' competitive advantage
- **Deliver working software frequently** on shorter timescales
- **Business people and developers must work together** daily throughout the project.
- **Build projects around motivated individuals**. Give them the environment and support needed, and trust them to get the job done.
- **Conveying information** to and within the development team through **face-to-face conversation.**

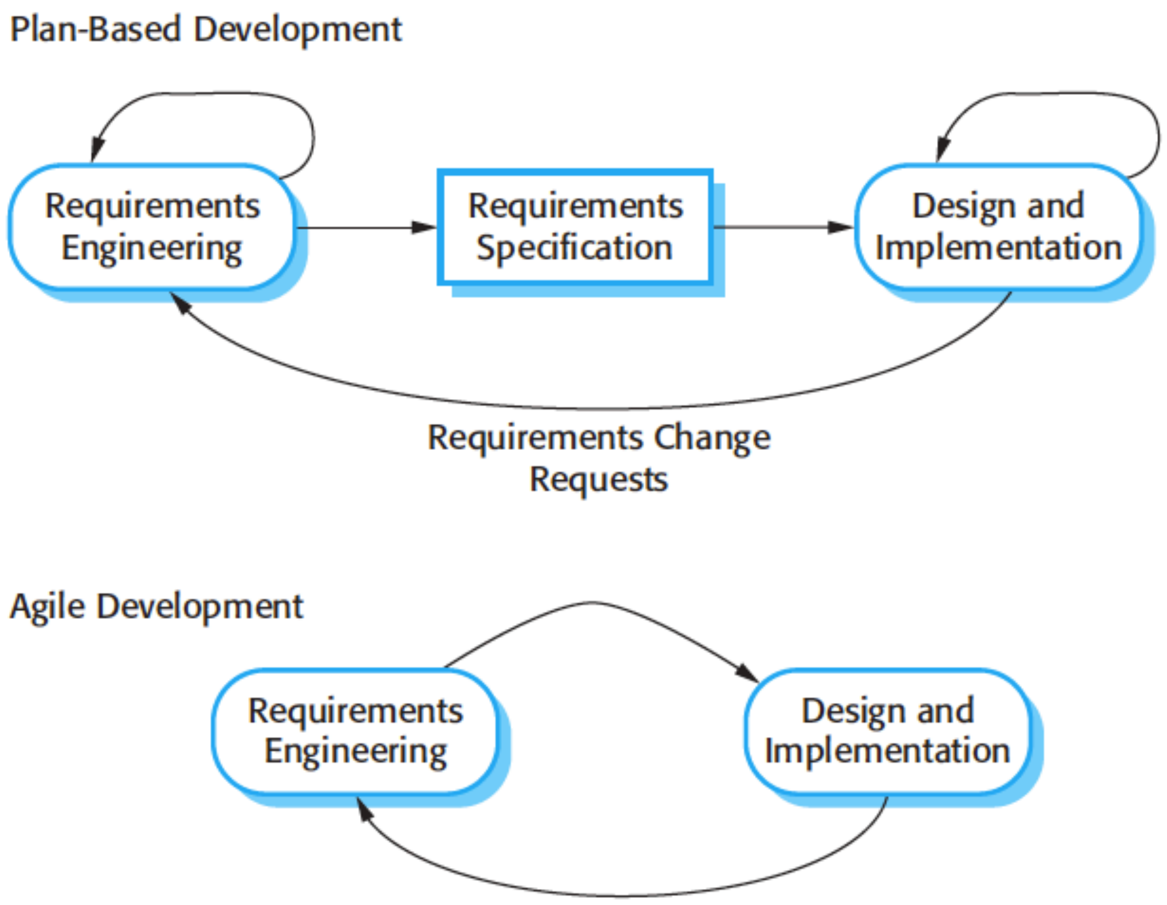
Source: www.agilemanifesto.org

Agile principles (CONT…)

- **working software** is the primary measure of progress
- **agile process** promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- **continuous attention** to technical excellence and good design enhances agility.
- **simplicity** – the art of maximizing the amount of work not done – is essential.
- The best architectures, requirements, and designs emerge from **self-organizing teams**.
- At **regular intervals**, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Source: www.agilemanifesto.org

Plan Driven VS Agile



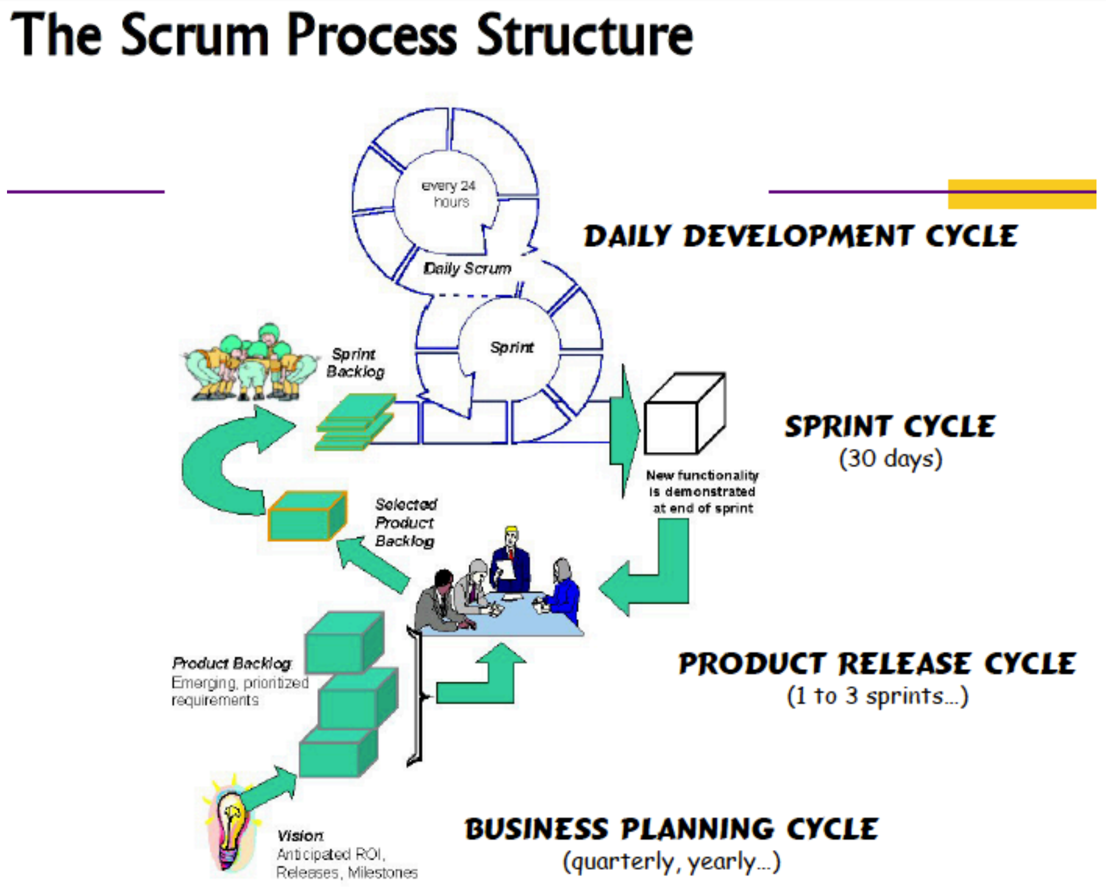
Plan-Based Development
Requirements Engineering
Requirements Specification
Design and Implementation
Requirements Change Requests
Agile Development
Requirements Engineering
Design and Implementation

Figure 3.2 Plan-driven and agile specification

Plan

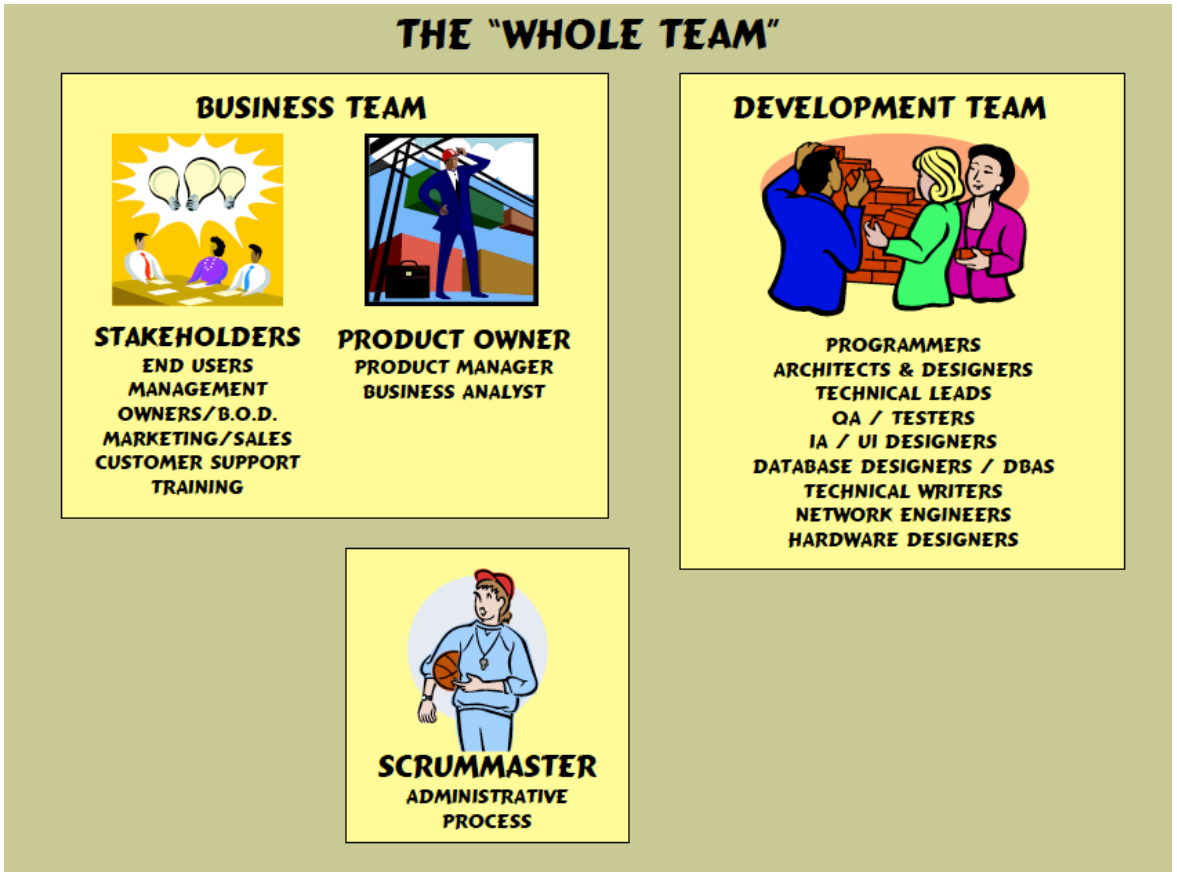
1. Agile principles
2. **Scrum**
3. eXtreme Programming (XP)

The Scrum process structure



Taken from slides “Agile Logic” by Paul Hodgetts

The Scrum team



The Scrum Team

- ScrumMaster
  - Facilitator and coach
- Product Owner
  - manages product vision
- Development Team
  - realizes the product
- External Roles

The ScrumMaster

- masters **Scrum** philosophy
- ensures team stays on process
- helps **Product Owner** and **Development Team**
- protects the team from impediments

The Product Owner

- represents the interests of the **stakeholder**
- **collaborates** daily with the **Development Team**
- **communicates** product requirements
- prioritizes requirements based on business value and risk
- **inspects increments**

Development Team

→ estimates cost and communicates trade-offs

→ mutually commits to **Sprint backlog**

→ self-organizes and self-manages tasks

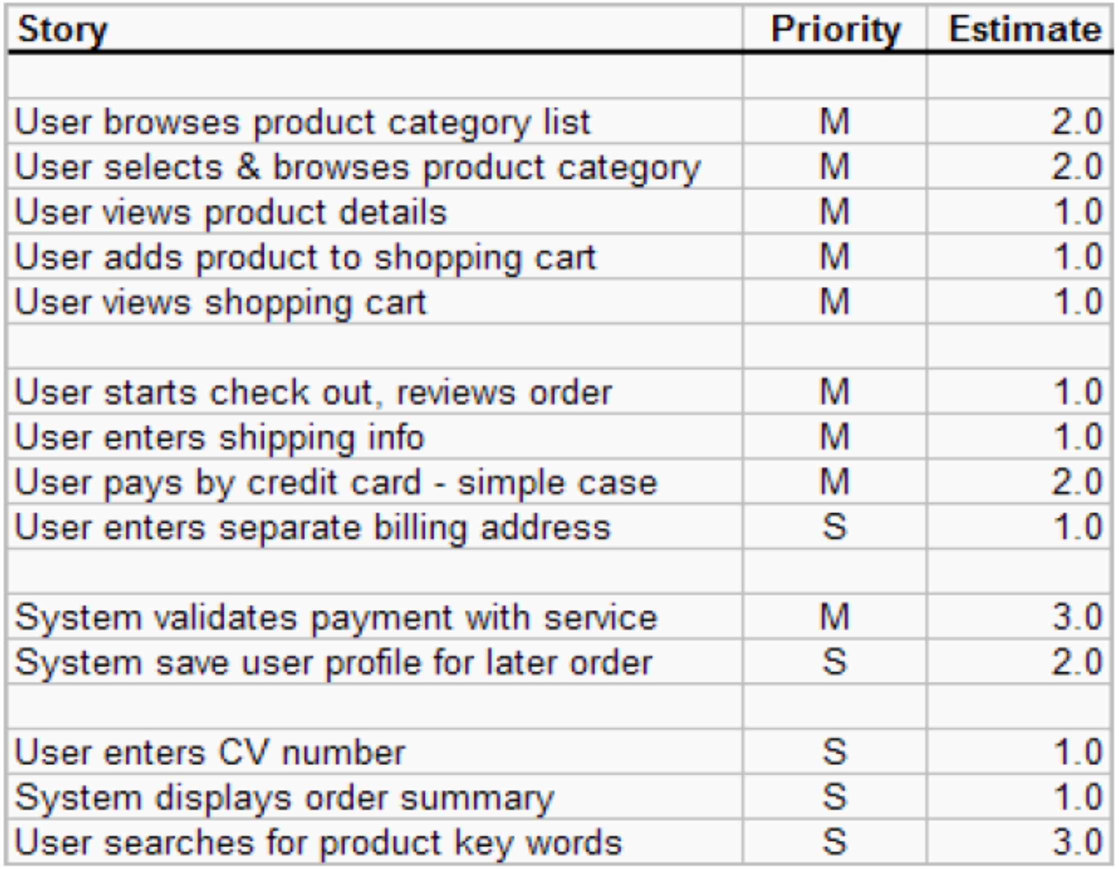
→is responsible for standards and practices

→builds and completes increments

Artifacts of a Scrum Projects

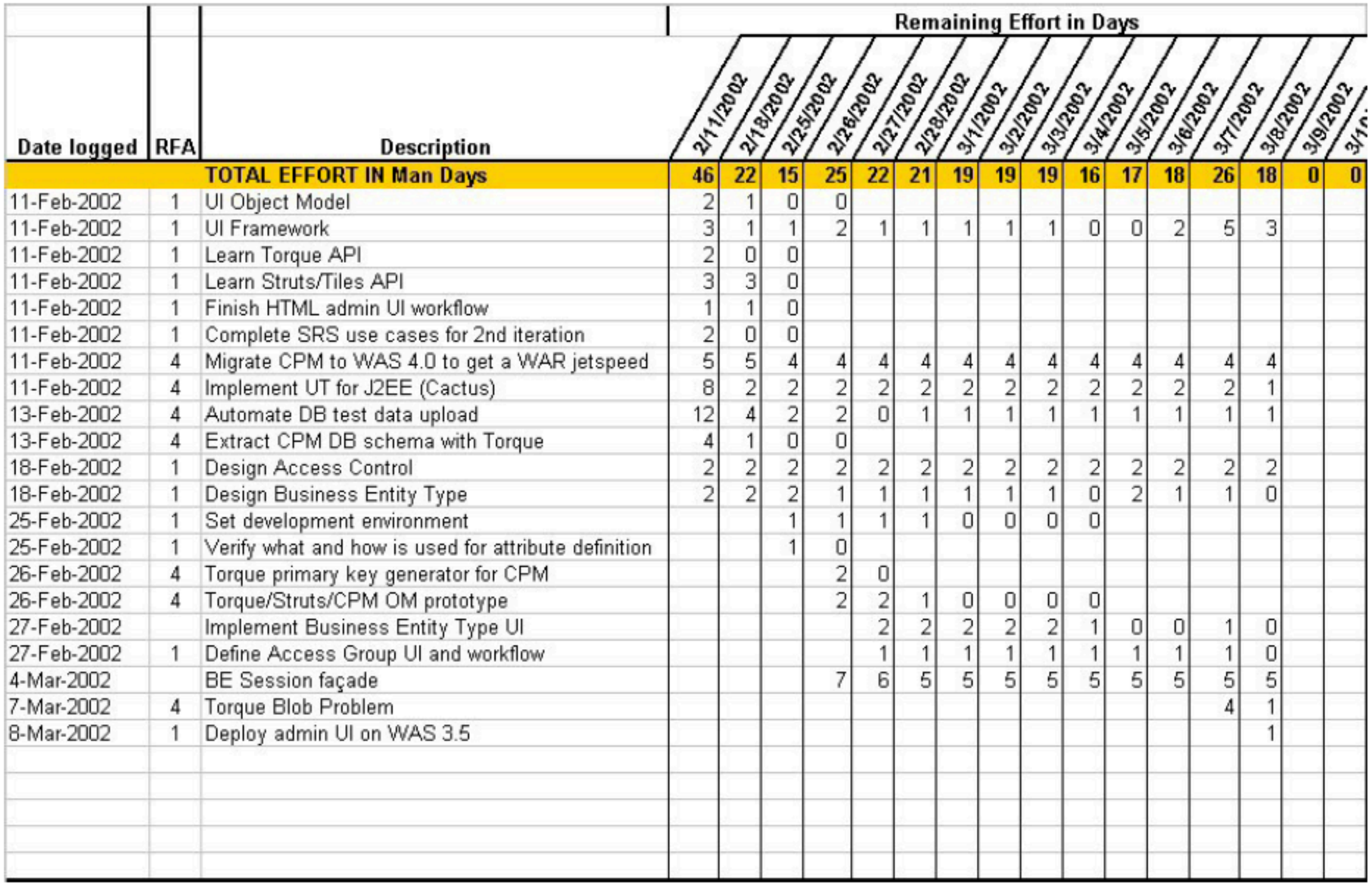
- **Product Backlog**
  - owned by **Product Owner**
  - captures product requirements
  - prioritized by business value and risk
  - supports coarse-grained estimating and planning

Product Backlog



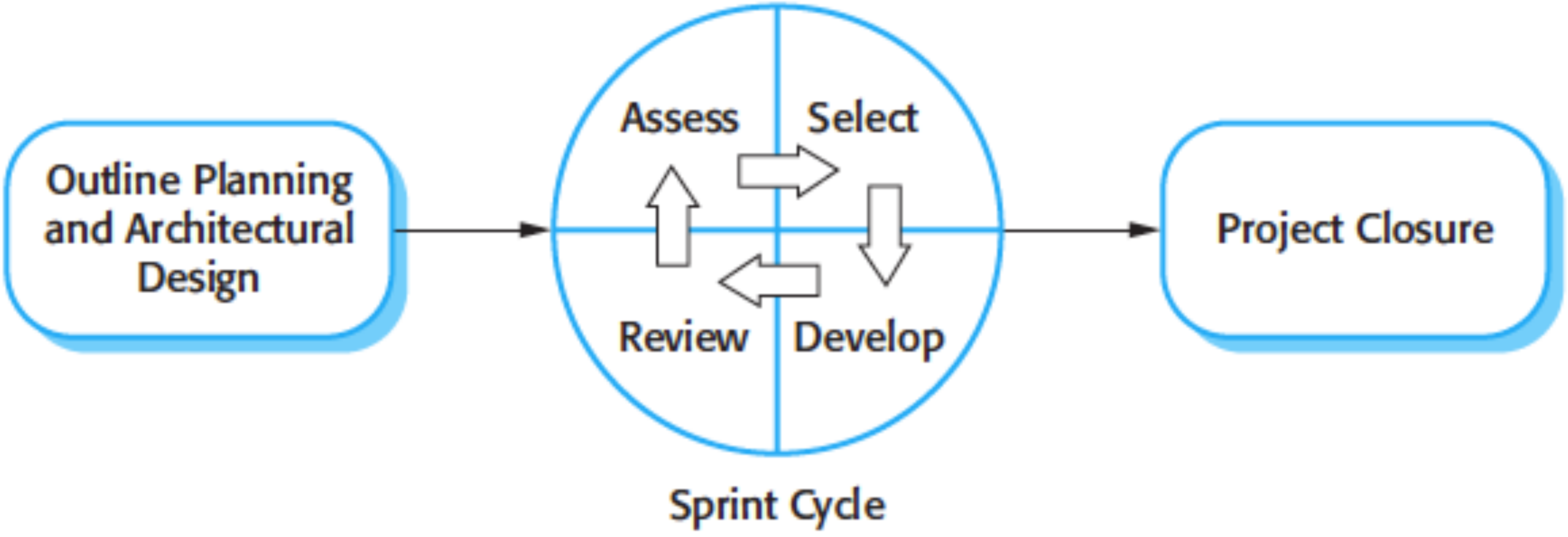
Story	Priority	Estimate
User browses product category list	M	2.0
User selects & browses product category	M	2.0
User views product details	M	1.0
User adds product to shopping cart	M	1.0
User views shopping cart	M	1.0
User starts check out, reviews order	M	1.0
User enters shipping info	M	1.0
User pays by credit card - simple case	M	2.0
User enters separate billing address	S	1.0
System validates payment with service	M	3.0
System save user profile for later order	S	2.0
User enters CV number	S	1.0
System displays order summary	S	1.0
User searches for product key words	S	3.0

Sprint Backlog



			Remaining Effort in Days															
Date logged	RFA	Description	2/11/2002	2/18/2002	2/25/2002	2/26/2002	2/27/2002	2/28/2002	3/1/2002	3/2/2002	3/3/2002	3/4/2002	3/5/2002	3/6/2002	3/7/2002	3/8/2002	3/9/2002	3/1
		TOTAL EFFORT IN Man Days	**46**	**22**	**15**	**25**	**22**	**21**	**19**	**19**	**19**	**16**	**17**	**18**	**26**	**18**	**0**	**0**
11-Feb-2002	1	UI Object Model	2	1	0	0												
11-Feb-2002	1	UI Framework	3	1	1	2	1	1	1	1	1	0	0	2	5	3		
11-Feb-2002	1	Learn Torque API	2	0	0													
11-Feb-2002	1	Learn Struts/Tiles API	3	3	0													
11-Feb-2002	1	Finish HTML admin UI workflow	1	1	0													
11-Feb-2002	1	Complete SRS use cases for 2nd iteration	2	0	0													
11-Feb-2002	4	Migrate CPM to WAS 4.0 to get a WAR jetspeed	5	5	4	4	4	4	4	4	4	4	4	4	4	4		
11-Feb-2002	4	Implement UT for J2EE (Cactus)	8	2	2	2	2	2	2	2	2	2	2	2	2	1		
13-Feb-2002	4	Automate DB test data upload	12	4	2	2	0	1	1	1	1	1	1	1	1	1		
13-Feb-2002	4	Extract CPM DB schema with Torque	4	1	0	0												
18-Feb-2002	1	Design Access Control	2	2	2	2	2	2	2	2	2	2	2	2	2	2		
18-Feb-2002	1	Design Business Entity Type	2	2	2	1	1	1	1	1	1	0	2	1	1	0		
25-Feb-2002	1	Set development environment			1	1	1	1	0	0	0	0						
25-Feb-2002	1	Verify what and how is used for attribute definition			1	0												
26-Feb-2002	4	Torque primary key generator for CPM				2	0											
26-Feb-2002	4	Torque/Struts/CPM OM prototype				2	2	1	0	0	0	0						
27-Feb-2002		Implement Business Entity Type UI					2	2	2	2	2	1	0	0	1	0		
27-Feb-2002	1	Define Access Group UI and workflow					1	1	1	1	1	1	1	1	1	0		
4-Mar-2002		BE Session façade				7	6	5	5	5	5	5	5	5	5	5		
7-Mar-2002	4	Torque Blob Problem													4	1		
8-Mar-2002	1	Deploy admin UI on WAS 3.5														1		

Scrum Project Management



Plan

1. Agile principles
2. Scrum
3. **eXtreme Programming (XP)**

eXtreme VS Scrum

- **Scrum teams** typically work in **iterations** (called **SPRINTS**) that are from **two weeks** to **one month** long.
- **XP teams** typically work in iterations that are **one** or **two weeks** long.

eXtreme VS Scrum

- **Scrum teams do not allow changes into their SPRINTS**. Once the SPRINT planning meeting is completed and a commitment made to delivering a set of product backlog items, that set of items remains unchanged through the end of the SPRINT.
- **XP teams are much more amenable to change within their iterations**. As long as the team hasn't started work on a particular feature, a new feature of equivalent size can be swapped into the XP team's iteration in exchange for the unstarted feature.

eXtreme VS Scrum

- **eXtreme Programming** teams work in a strict priority order. Features to be developed are prioritized by the Customer and the team is required to work on them in that order.
- **Scrum's Product Owner** (the Customer) prioritizes the product **backlog** but the team determines the sequence in which they will develop the **backlog items**.

eXtreme VS Scrum

- **Scrum** doesn’t prescribe any **engineering practices**; **XP** does
  - **user stories**
  - **acceptances tests**
  - **pair programming**
  - **test driven development (TDD)**
  - **the planning game**
  - **automated testing**
  - **simple design**
  - **refactoring**

User Stories

- **As a <role>, I want <goal/desire>**

- “As a **user**, I want **to search for my customers** by their first and last names.”

Pair Programming

- **code** is written by **pairs of programmers** working together at the same workstation.
  - one member of each pair **types the code**.
  - the other member **finds errors** and **improvements**.
  - **roles change frequently!!**

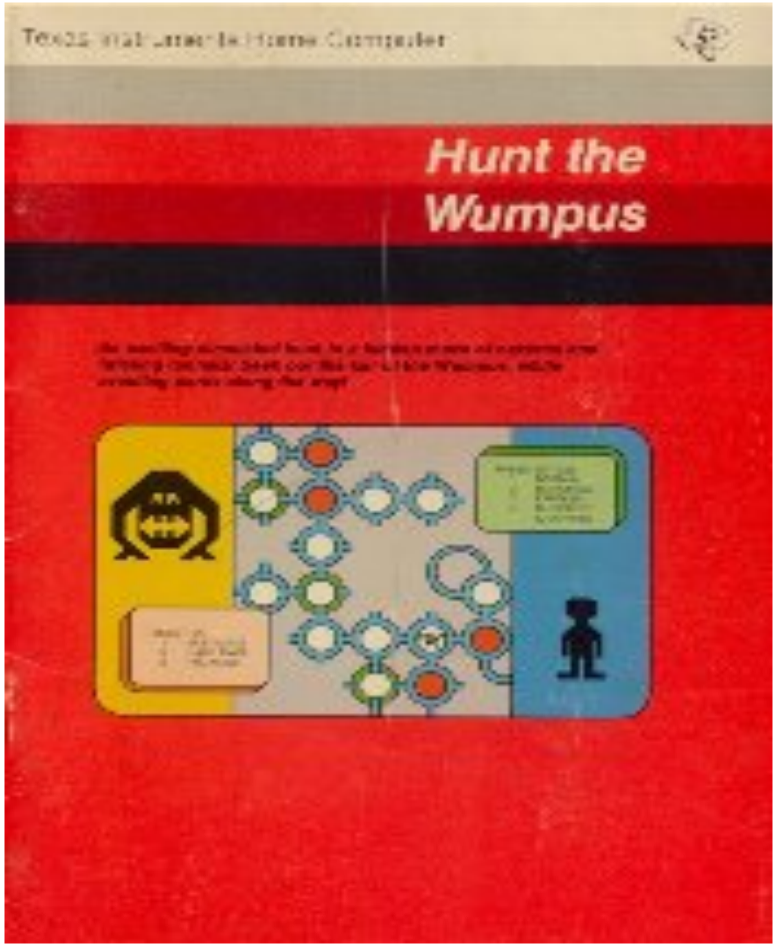
Test Driven Development (TDD)

- production code is written to make a failing **unit test** pass.
  - we write a unit test that fails because the functionality it is testing does not exist.
  - Then, one writes the code that makes that test pass.
- **test cases and code evolve together**, with the test cases leading the code by a very small fraction.

Test Driven Development (TDD)

1. Don't write any production code until you have written a failing unit test.
2. Don't write more of a unit test than is sufficient to fail or fail to compile.
3. Don't write any more production code than is sufficient to pass the failing test.

Unit test –
Hunt the Wumpus



Unit test – Hunt the Wumpus

```
public void TestMove() {
 WumpusGame g = new WumpusGame();
 g.Connect(4,5,"E");
 g.GetPlayerRoom(4);
 g.East();
 Assert.AreEqual(5,g.GetPlayerRoom());
}
```

Acceptance Tests

- acceptance tests for a **user story** are written immediately preceding, or even concurrently with, the implementation of that story.
- they are written in a **scripting language** that allows them to be run automatically and repeatedly.
- acceptance tests are written by business analysts, quality assurance specialists, and testers during the iteration.

http://www.fitnesse.org/

The Planning Game

- At the beginning of each release and each iteration, the developers give the customers a **budget**.
- The customers choose **user stories** whose costs total up to that budget and are not allowed to exceed their budget.
- Developers **determine their budget** based on how much they were able to get done in the **previous iteration** (Sprint) or in the **previous release**.

The Planning Game – Splitting Stories

- **“users can securely transfer money into, out of, and between their accounts” →**
  - – users can log in.
  - – users can log out.
  - – users can deposit money into their accounts.
  - – users can withdraw money from their accounts.
  - – users can transfer money from one of their accounts to another account.