



SOFTWARE ENGINEERING (15B11CI513)

Credits :- 4

Contact Hours :- 3-1-0

Agile methods



- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
 - Focus on the code rather than the design
 - Are based on an iterative approach to software development
 - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

Principles of agile methods



Principle	Description
Customer involvement	The customer should be closely involved throughout the development process. Their role is provide and prioritise new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognised and exploited. The team should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and design the system so that it can accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process used. Wherever possible, actively work to eliminate complexity from the system.

Agile Methods



- Comparison with traditional methods
- Agile methods require customer involvement throughout development versus involvement only at specific phases.
 - Agile methods only develop detailed plans for the near term versus detailed planning over the entire project.
 - Agile methods use working code and prototypes versus documentation
 - Agile methods rely on early developer test case development versus separate test functions.
- Example Test Driven Development
 - Agile methods design for one feature at a time versus complete design.
 - Agile methods produce many small release versus a limited number of larger releases.

Agile method applicability



- Product development where a software company is developing a **small or medium-sized product** for sale.
- Custom system development within an organization, where there is a **clear commitment from the customer to become involved** in the development process, and where there are not many external rules and regulations that affect the software.
- Because of their focus on small, tightly-integrated teams, there are problems in **scaling agile methods to large systems**.

Problems with agile methods



- It can **be difficult to keep the interest of customers** who are involved in the process.
- Team members may be unsuited to the intense involvement that characterizes agile methods.
- Prioritizing changes can be difficult where multiple stakeholders are involved.
- Maintaining **simplicity requires extra work**.

Agile Methods



Example Agile Methods

Extreme Programming

OpenUP

LEAN Development

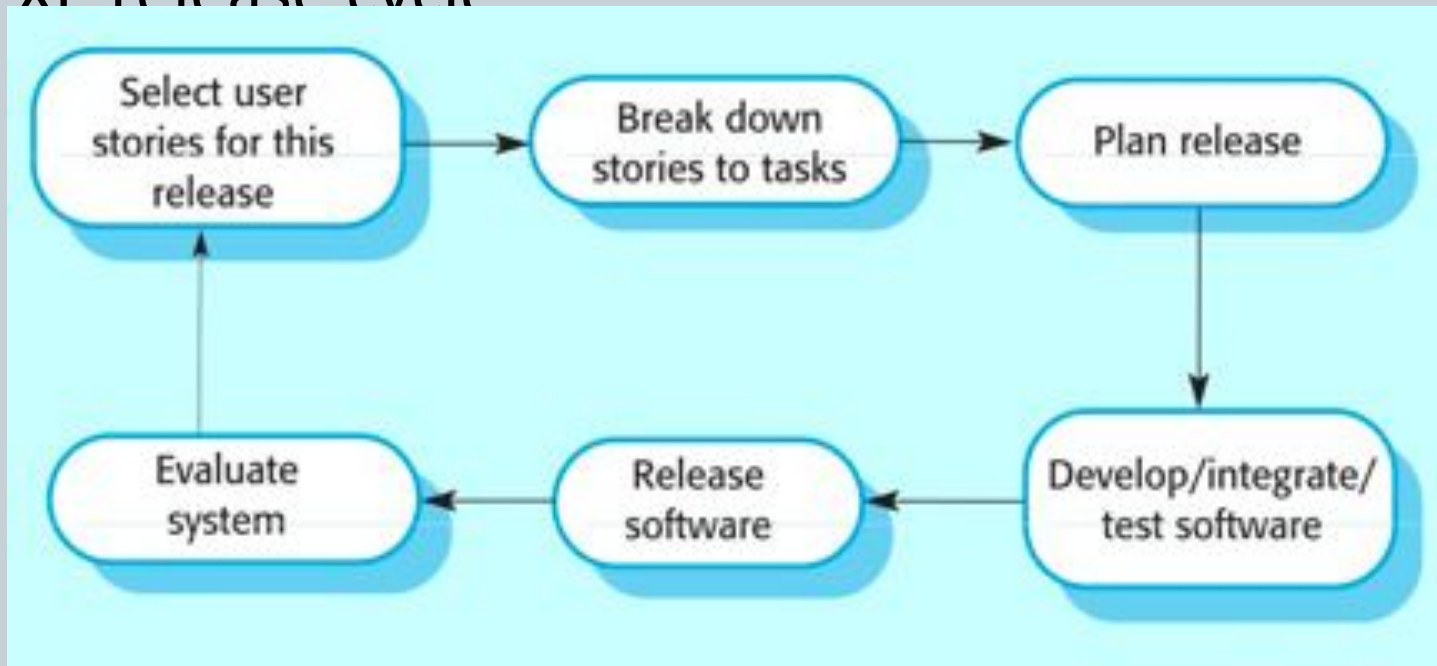
Crystal

SCRUM

Test Driven Development

Extreme Programming

- Perhaps the best-known and most widely used agile method.
- Introduced by Kent Beck in 1999
- The XP release cycle



Extreme programming practices 1



Incremental planning	Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development Tasks
Small Releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple Design	Enough design is carried out to meet the current requirements and no more.
Test first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme programming practices 2



Pair Programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective Ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.
Continuous Integration	As soon as work on a task is complete it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of over-time are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site Customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP and agile principles



- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code.

Customer involvement



- Customer involvement is a key part of XP where the customer is part of the development team.
- The role of the customer is:
 - To help develop stories that define the requirements
 - To help prioritise the features to be implemented in each release
 - To help develop acceptance tests which assess whether or not the system meets its requirements.

Requirements scenarios



In XP, user requirements are **expressed as scenarios or user stories**.

These are written on cards and the **development team break them down into implementation tasks**. These tasks are the basis of schedule and cost estimates.

The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

Story card for document downloading



Downloading and printing an article

First, you select the article that you want from a displayed list. You then have to tell the system how you will pay for it - this can either be through a subscription, through a company account or by credit card.

After this, you get a copyright form from the system to fill in and, when you have submitted this, the article you want is downloaded onto your computer.

You then choose a printer and a copy of the article is printed. You tell the system if printing has been successful.

If the article is a print-only article, you can't keep the PDF version so it is automatically deleted from your computer.

XP and change



- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this **reduces costs later in the life cycle**.
- XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it proposes **constant code improvement (*refactoring*)** to make changes easier when they have to be implemented.

Refactoring



- Refactoring is the process of code improvement where code is reorganised and rewritten to make it more efficient, easier to understand, etc.
- Refactoring is required because frequent releases mean that is developed incrementally and therefore tends to become messy.
- Refactoring **should not change the functionality** of the system.

Testing in XP



- Test-first development.
- Incremental test development from scenarios.
- User involvement in test development and validation.
- Automated test harnesses are used to run all component tests each time that a new release is built.

Task cards for document downloading



Task 1: Implement principal workflow

Task 2: Implement article catalog and selection

Task 3: Implement payment collection

Payment may be made in 3 different ways. The user selects which way they wish to pay. If the user has a library subscription, then they can input the subscriber key which should be checked by the system. Alternatively, they can input an organisational account number. If this is valid, a debit of the cost of the article is posted to this account. Finally, they may input a 16 digit credit card number and expiry date. This should be checked for validity and, if valid a debit is posted to that credit card account.

Test case description



Test 4: Test credit card validity

Input:

A string representing the credit card number and two integers representing the month and year when the card expires

Tests:

Check that all bytes in the string are digits

Check that the month lies between 1 and 12 and the year is greater than or equal to the current year.

Using the first 4 digits of the credit card number, check that the card issuer is valid by looking up the card issuer table. Check credit card validity by submitting the card number and expiry date information to the card issuer

Output:

OK or error message indicating that the card is invalid

Test-first development



- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
- All previous and new tests are automatically run when new functionality is added. Thus checking that the new functionality has not introduced errors.

Testing in XP



- Every iteration starts with a tests written for a new piece of functionality .
- Test cases are created before test is written
- TDD instructs developers to write new code only if an automated test has failed.
- Automated test harnesses are used to run all component tests each time that a new release is built.

Benefits of TDD



- Small Regression suite
- Since, We are doing test first development, it means reduction in bugs
- TDD is used to make the code clearer, simple and bug free
- Avoids duplication of code
- TDD drives the code design and approach

Limitations of TDD



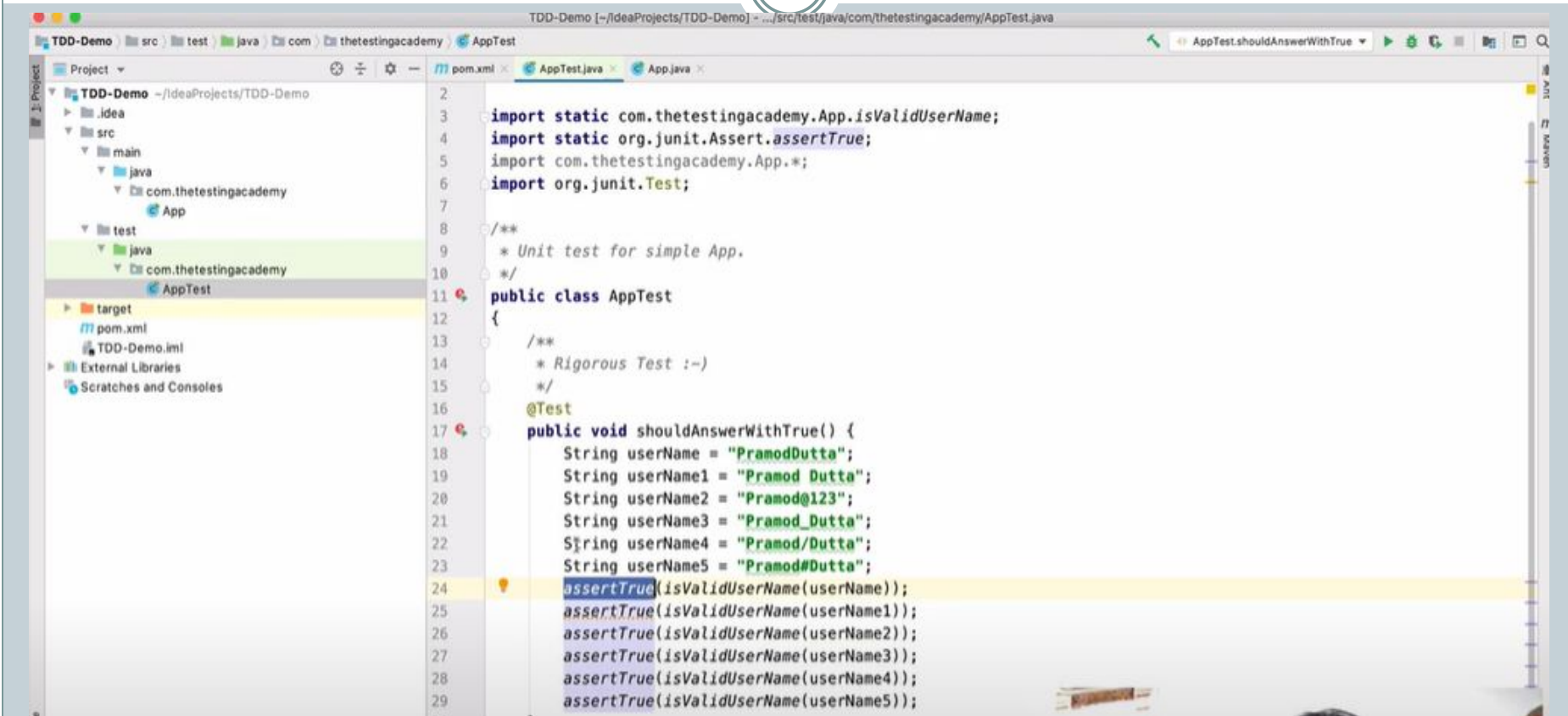
- Early test cases creation may cause heavy test maintenance
- Requirement change is paid in TDD

Sequence of steps inTDD



- Read the requirement (User story)
- Add a test
- Run test if fails
- Write a code to fix it
- Run test
- Refactor code
- Repeat
-

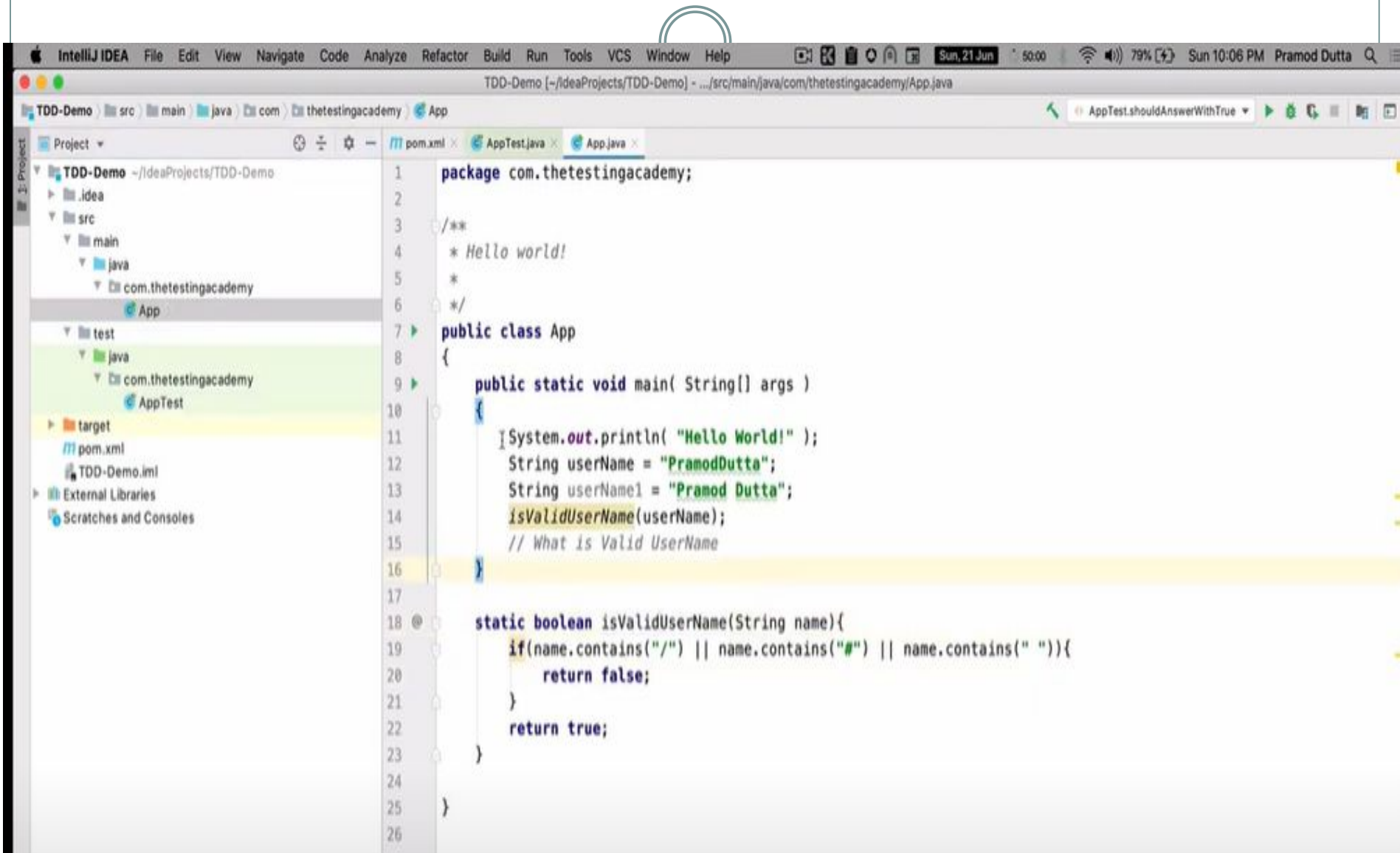
Sequence of steps inTDD



The screenshot shows an IDE window for a project named 'TDD-Demo'. The left sidebar displays the project structure, including 'src/main/java/com/thetestingacademy/App' and 'src/test/java/com/thetestingacademy/AppTest'. The main editor area shows the code for 'AppTest.java'. The code includes imports for 'com.thetestingacademy.App', 'org.junit.Assert', and 'org.junit.Test'. It features a unit test method 'shouldAnswerWithTrue()' that uses 'assertTrue()' to verify the 'isValidUserName()' method of the 'App' class for various input strings.

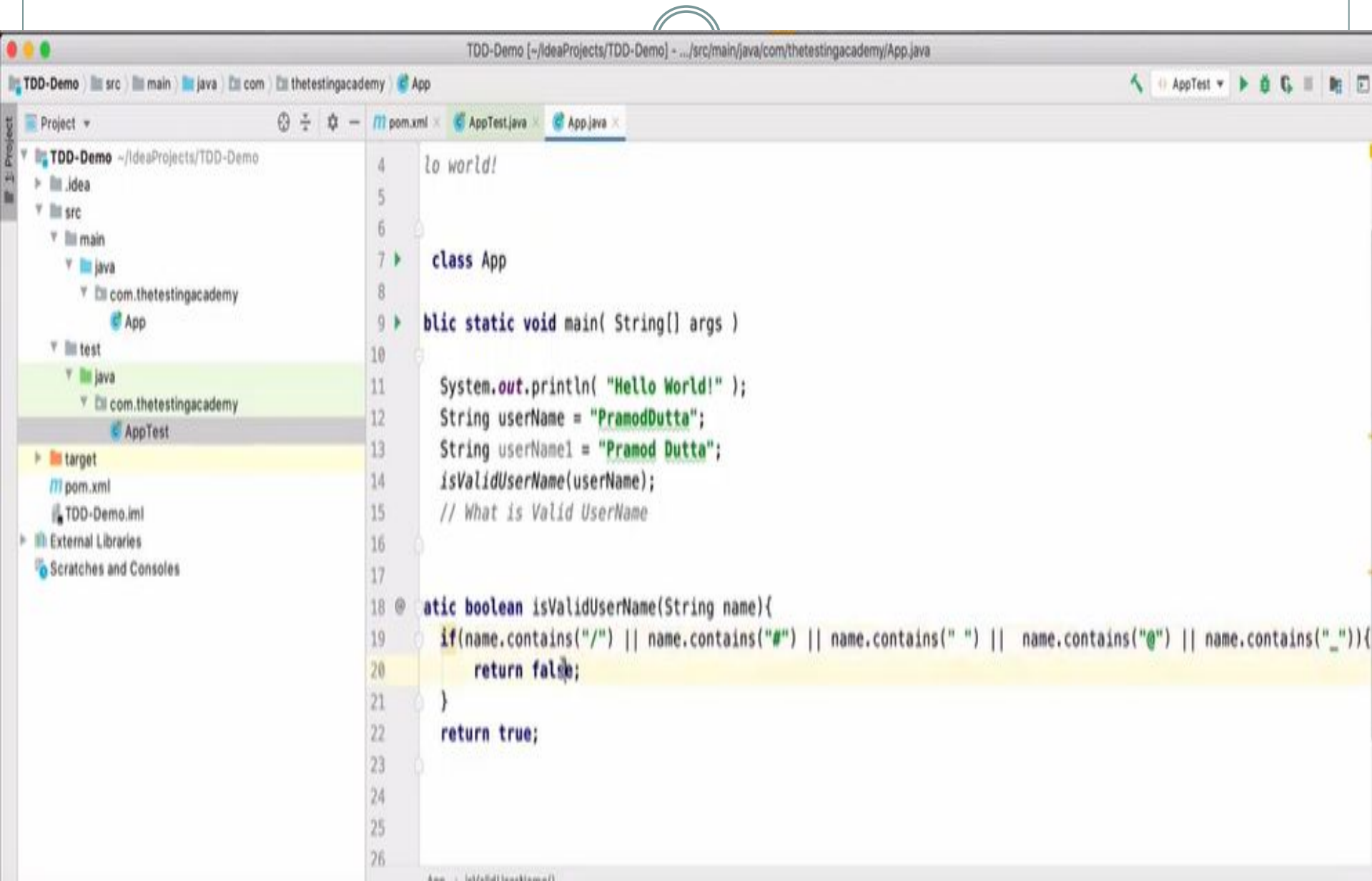
```
1 2
3  import static com.thetestingacademy.App.isValidUserName;
4  import static org.junit.Assert.assertTrue;
5  import com.thetestingacademy.App.*;
6  import org.junit.Test;
7
8  /**
9   * Unit test for simple App.
10  */
11  public class AppTest
12  {
13      /**
14       * Rigorous Test :-
15       */
16      @Test
17      public void shouldAnswerWithTrue() {
18          String userName = "PramodDutta";
19          String userName1 = "Pramod Dutta";
20          String userName2 = "Pramod@123";
21          String userName3 = "Pramod_Dutta";
22          String userName4 = "Pramod/Dutta";
23          String userName5 = "Pramod#Dutta";
24          assertTrue(isValidUserName(userName));
25          assertTrue(isValidUserName(userName1));
26          assertTrue(isValidUserName(userName2));
27          assertTrue(isValidUserName(userName3));
28          assertTrue(isValidUserName(userName4));
29          assertTrue(isValidUserName(userName5));
30  }
```

Sequence of steps inTDD



```
1 package com.thetestingacademy;
2
3 /**
4  * Hello world!
5  *
6  */
7 public class App
8 {
9     public static void main( String[] args )
10    {
11        System.out.println( "Hello World!" );
12        String userName = "PramodDutta";
13        String userName1 = "Pramod Dutta";
14        isValidUserName(userName);
15        // What is Valid UserName
16    }
17
18    static boolean isValidUserName(String name){
19        if(name.contains("/") || name.contains("#") || name.contains(" ")){
20            return false;
21        }
22        return true;
23    }
24
25 }
26
```

Sequence of steps inTDD



```
4 // Hello
5
6
7 class App
8
9 public static void main( String[] args )
10
11 System.out.println( "Hello World!" );
12 String userName = "PramodDutta";
13 String userName1 = "Pramod Dutta";
14 isValidUserName(userName);
15 // What is Valid UserName
16
17
18 public boolean isValidUserName(String name){
19 if(name.contains("/") || name.contains("#") || name.contains(" ") || name.contains("@") || name.contains("_")){
20 return false;
21 }
22 return true;
23
24
25
26
```

Pair programming



- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.

Problems with XP



● Customer involvement

This is perhaps the most difficult problem. It may be difficult or impossible to find a customer who can represent all stakeholders and who can be taken off their normal work to become part of the XP team. For generic products, there is no 'customer' - the marketing team may not be a typical of real customers.

● Architectural design

- The incremental style of development can mean that inappropriate architectural decisions are made at an early stage of the process.
- Problems with these may not become clear until many features have been implemented and refactoring the architecture is very expensive.

Problems with XP cont...



● Test complacency

- It is easy for a team to believe that because it has many tests, the system is properly tested.
- Because of the automated testing approach, there is a tendency to develop tests that are easy to automate rather than tests that are 'good' tests.

Key points



- Extreme programming includes practices such as systematic testing, continuous improvement and customer involvement.
- Customers are involved in developing requirements which are expressed as simple scenarios.
- The approach to testing in XP is a particular strength where executable tests are developed before the code is written.
- Key problems with XP include difficulties of getting representative customers and problems of architectural design.

Scrum



- Scrum is an agile process that allows us to focus on delivering the highest business value in the shortest time.
- It allows us to rapidly and repeatedly inspect actual working software (every two weeks to one month).
- The business sets the priorities. Teams self-organize to determine the best way to deliver the highest priority features.
- Every two weeks to a month anyone can see real working software and decide to release it as is or continue to enhance it for another sprint.

Characteristics



- Self-organizing teams
- Product progresses in a series of month-long “sprints”
- Requirements are captured as items in a list of “product backlog”
- No specific engineering practices prescribed
- Uses generative rules to create an agile environment for delivering projects
- One of the “agile processes”

The Agile Manifesto-a statement of values

Individuals and
interactions

over

Process and tools

Working software

over

Comprehensive
documentation

Customer
collaboration

over

Contract negotiation

Responding to
change

over

Following a plan

SCRUM

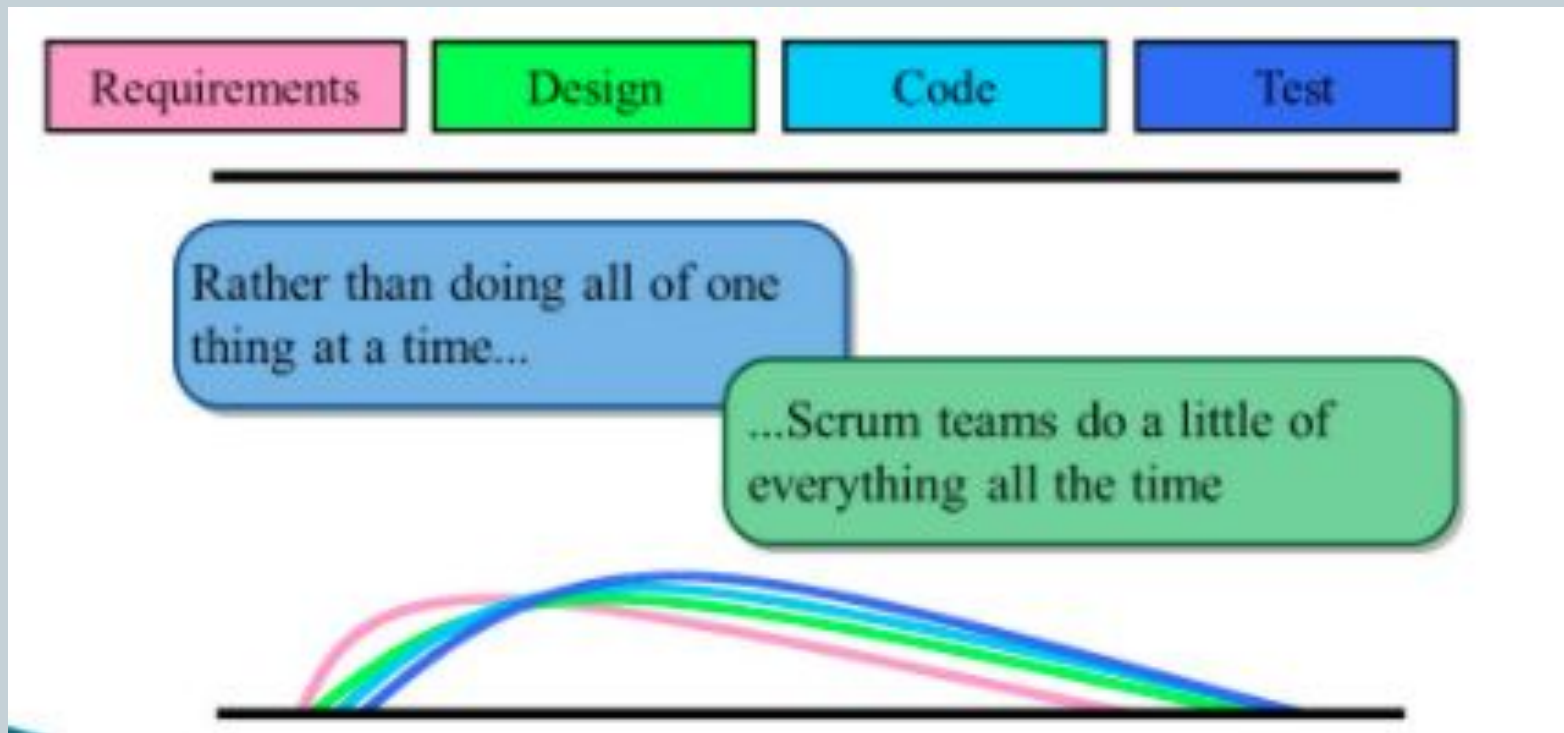


Sprints



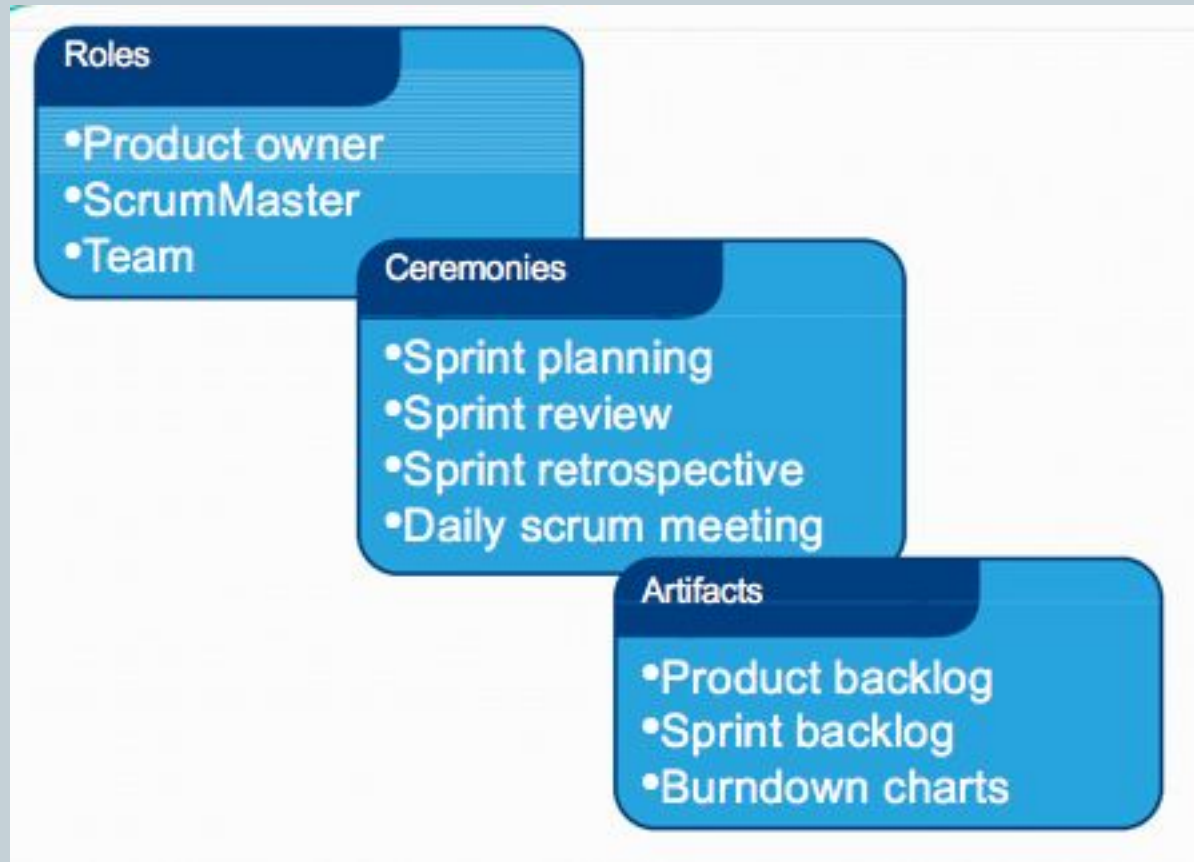
- Scrum projects make progress in a series of “sprints”
 - Analogous to Extreme Programming iterations
- Typical duration is 2-4 weeks or a calendar month at most
- A constant duration leads to a better rhythm
- Product is designed, coded, and tested during the sprint

Sequential vs. overlapping development



Source: "The New New Product Development Game" by
Takeuchi and Nonaka. *Harvard Business Review*, January 1986.

Scrum framework



Roles: Product owner



- Define the features of the product
- Decide on release date and content
- Be responsible for the profitability of the product (ROI)
- Prioritize features according to market value
- Adjust features and priority every iteration, as needed
- Accept or reject work results

Roles: The Scrum Master



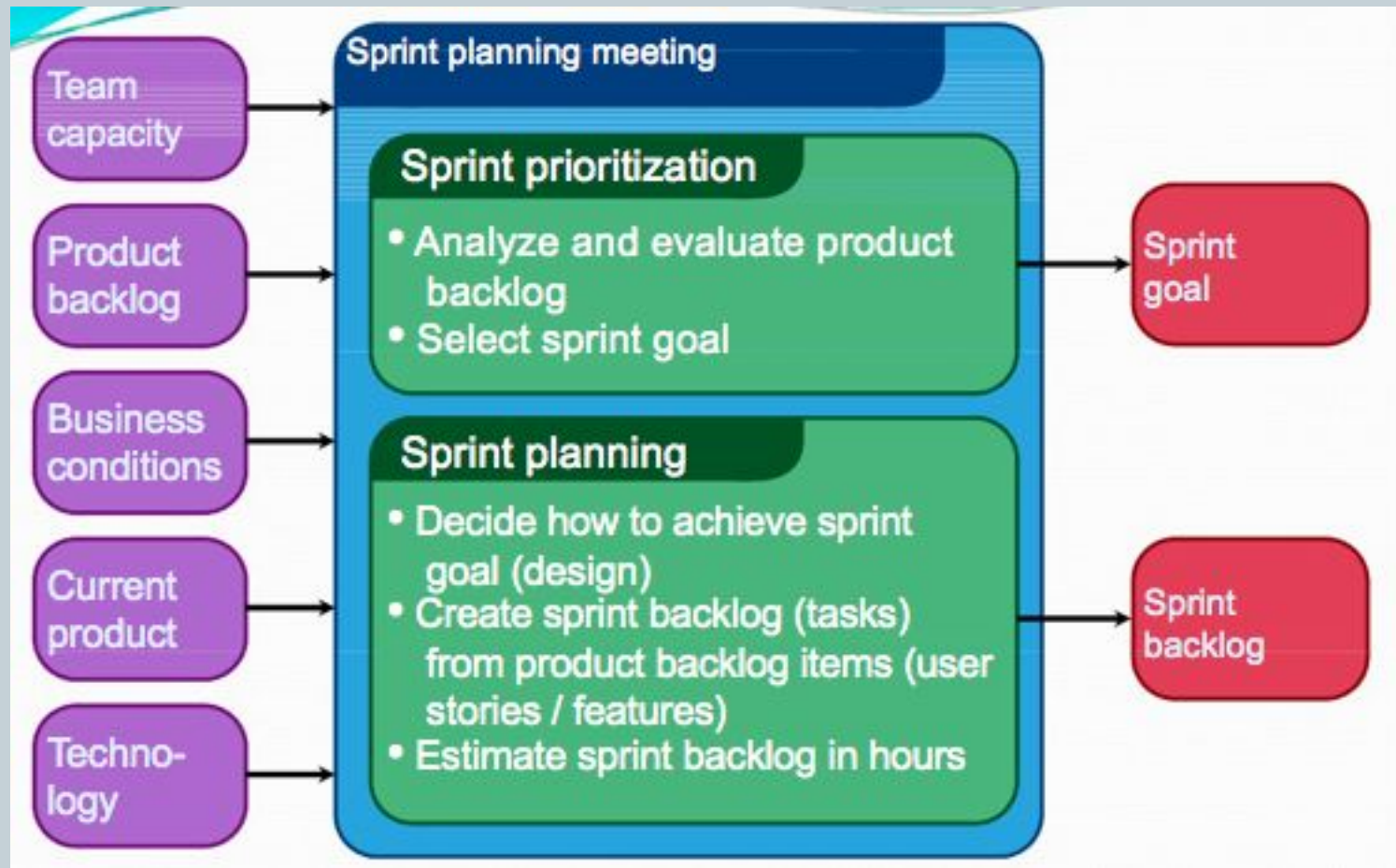
- Represents management to the project
- Responsible for enacting Scrum values and practices
- Removes impediments(obstruction)
- Ensure that the team is fully functional and productive
- Shield the team from external interferences

Roles: The team



- Typically consists of 5-9 people
- Cross-functional:
 - Programmers, testers, user experience designers, etc.
- Teams are self-organizing

Ceremonies: Sprint planning



Ceremonies: Sprint planning

- Team selects items from the product backlog they can commit to completing
- Sprint backlog is created
 - Tasks are identified and each is estimated
 - Collaboratively, not done alone by the Scrum Master
- High-level design is considered

As a vacation planner, I want to see photos of the hotels.

Code the middle tier (8 hours)
Code the user interface (4)
Write test fixtures (4)
Code the xyz class (6)
Update performance tests (4)

Ceremonies: The daily scrum



- Parameters
 - Daily
 - 15-minutes
 - Stand-up
- Not for problem solving
 - Only team members, Scrum Master, product owner, can talk
- Helps avoid other unnecessary meetings

Ceremonies: The sprint review



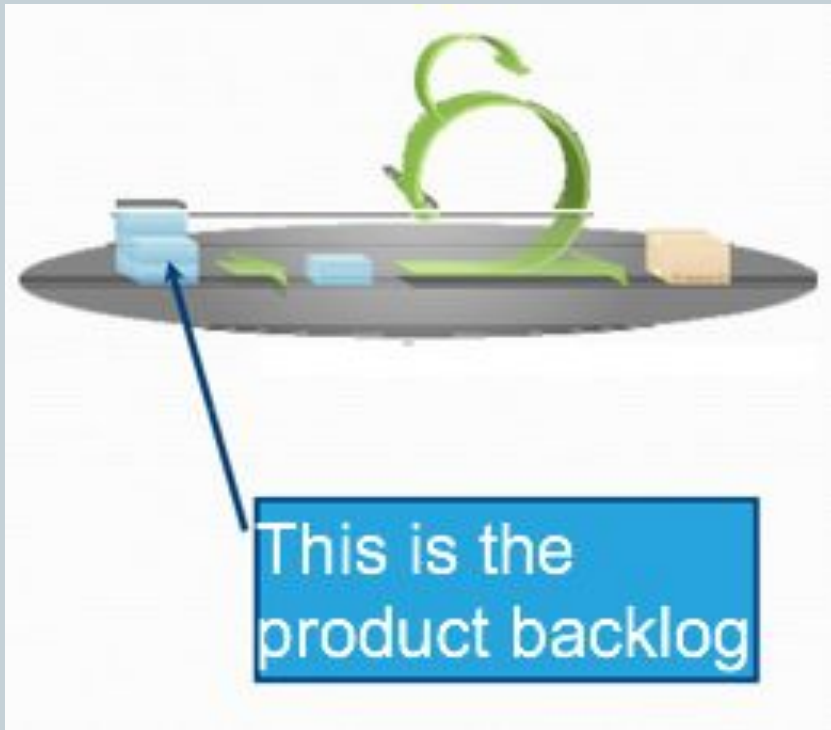
- Team presents what it accomplished during the sprint
 - Typically takes the form of a demo of new features or underlying architecture
 - Informal
 - 2-hour prep time rule
 - No slides
 - Whole team participates

Ceremonies: Sprint retrospective



- Periodically take a look at what is and is not working
- Typically 15-30 minutes
- Done after every sprint
- Whole team participates
 - Scrum Master
 - Product owner
 - Team
 - Possibly customers and others

Artifacts: Product backlog



- The requirements
- A list of all desired work on the project
- Ideally expressed such that each item has value to the users or customers of the product
- Prioritized by the product owner
- Reprioritized at the start of each sprint

A sample product backlog



Backlog item	Estimate
Allow a guest to make a reservation	3
As a guest, I want to cancel a reservation.	5
As a guest, I want to change the dates of a reservation.	3
As a hotel employee, I can run RevPAR reports (revenue-per-available-room)	8
Improve exception handling	8
	30
	50

Cons



- The basic premise that the team is committed to the project. If the team is not committed then process collapses
- The management's comfort level in delegation of tasks
- The size of the team is restricted due to the involvement of all team members
- Suited for development of new products and not for enhancement of an existing product
- Reliance on experience