# Agile Software Development

**Eunseok Lee, Prof.**
**College of Software**
**Sungkyunkwan University**

# Objectives

- Introduce you to agile software development method;

- Understand the rationale for agile software development methods, the agile manifesto, and the differences between agile and plan-driven development;

- Know about important agile development practices such as user stories, refactoring, pair programming and test-first development;

- Understand the Scrum approach to agile project management;

- Understand the issues of scaling agile development methods and combining agile approaches with plan-driven approaches in the development of large software systems.

# Topics covered

1. Agile methods

2. Plan-driven and agile development

3. Extreme programming

4. Agile project management

5. Scaling agile methods

# Rapid software development

- **Rapid development** and **delivery** is now often the most important requirement for software systems

  – Businesses operate in a fast–changing requirement and it is practically *impossible to produce a set of stable* software requirements

  – Software has to evolve quickly to reflect changing business needs.

- **Rapid software development**

  – Specification, design and implementation are *inter-leaved*

  – System is developed *as a series of increments* with stakeholders involved in increment evaluation

# 1. 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.

# 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:

    – *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.

# The principles of agile methods

| Principle | Description |
|-----------|-------------|
| Customer involvement | Customers should be closely involved throughout the development process. Their role is provide and prioritize 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 recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes. |
| Embrace change | Expect the system requirements to change and so design the system to accommodate these changes. |
| Maintain simplicity | Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system. |

# 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 a lot of 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 there are multiple stakeholders.

- Maintaining simplicity requires extra work.

- Contracts may be a problem as with other approaches to iterative development.

# Agile methods and software maintenance

- Most organizations spend more on maintaining existing software than they do on new software development. **So, if agile methods are to be successful, they have to support *maintenance* as well as original development.**

- **Two key issues:**

  - Are systems that are developed using an agile approach maintainable, given the emphasis in the development process of minimizing formal documentation?

  - Can agile methods be used effectively for evolving a system in response to customer change requests?

- **Problems may arise if original development team cannot be maintained.**

SUNG KYUN KWAN UNIVERSITY

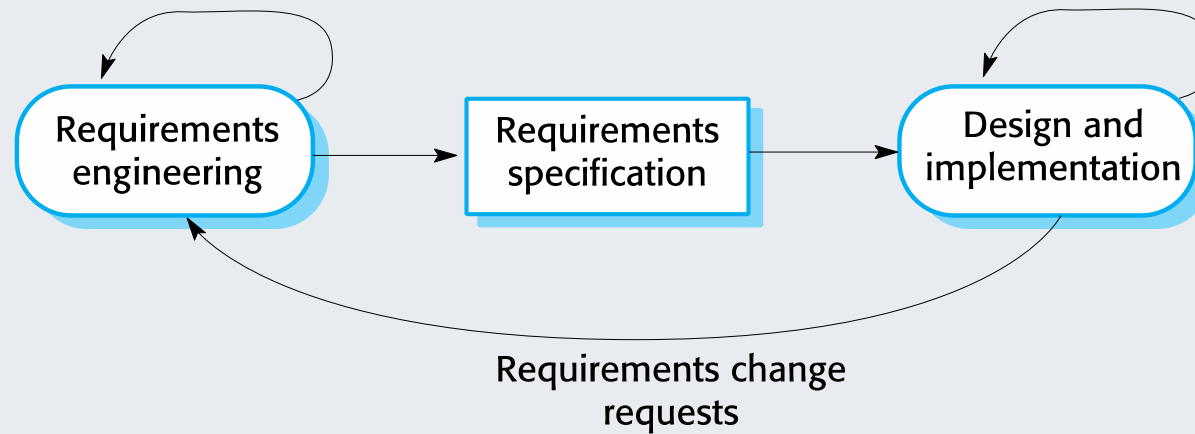# 2. Plan-driven and agile development

- **Plan-driven development**

  - A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.

  - Not necessarily waterfall model – plan-driven, incremental development is possible

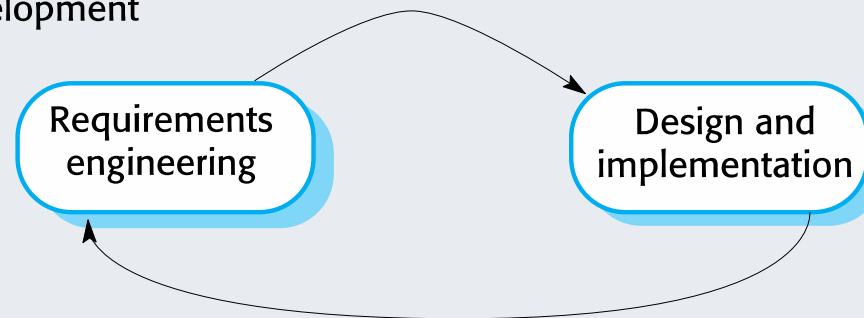  - Iteration occurs within activities.

- **Agile development**

  - Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

# Plan-driven and agile specification

Plan-based development

Requirements engineering → Requirements specification → Design and implementation

Requirements change requests

Agile development

Requirements engineering → Design and implementation

SUNG KYUN KWAN UNIVERSITY

# Technical, human, organizational issues

- **Most projects include elements of plan-driven and agile processes. Deciding on the balance depends on:**

  – Is it important to have a very detailed specification and design before moving to implementation? If so, you probably need to use a plan-driven approach.

  – Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic? If so, consider using agile methods.

  – How large is the system that is being developed? Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used.

# Technical, human, organizational issues

- – What type of system is being developed?
  - Plan-driven approaches may be required for systems that require a lot of analysis before implementation (e.g. real-time system with complex timing requirements).
- – What is the expected system lifetime?
  - Long-lifetime systems may require more design documentation to communicate the original intentions of the system developers to the support team.
- – What technologies are available to support system development?
  - Agile methods rely on good tools to keep track of an evolving design
- – How is the development team organized?
  - If the development team is distributed or if part of the development is being outsourced, then you may need to develop design documents to communicate across the development teams.

SUNG KYUN KWAN UNIVERSITY

# Technical, human, organizational issues

- Are there cultural or organizational issues that may affect the system development?
  - Traditional engineering organizations have a culture of plan-based development, as this is the norm in engineering.
- How good are the designers and programmers in the development team?
  - It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code
- Is the system subject to external regulation?
  - If a system has to be approved by an external regulator (e.g. the FAA approve software that is critical to the operation of an aircraft) then you will probably be required to produce detailed documentation as part of the system safety case.
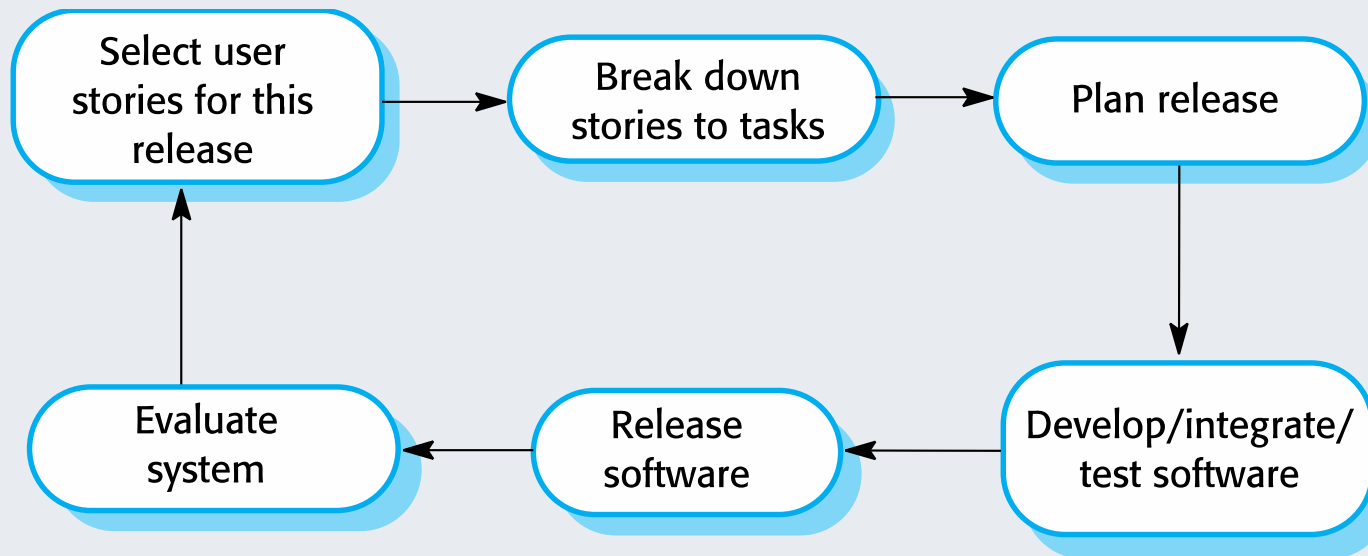
# 3. Extreme programming

- Perhaps the best-known and most widely used agile method.

- Extreme Programming (XP) takes an 'extreme' approach to iterative development.

  - New versions may be built several times per day;

  - Increments are delivered to customers every 2 weeks;

  - All tests must be run for every build and the build is only accepted if tests run successfully.

# 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.

SUNG KYUN KWAN UNIVERSITY

# The extreme programming release cycle

# Extreme programming practices (1)

| Principle or practice | Description |
|---|---|
| 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 take responsibility for all of the code. Anyone can change anything. |
| Continuous integration | As soon as the 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 overtime 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. |

# Requirements scenarios

- In XP, a customer or user is part of the XP team and is responsible for making decisions on requirements.

- 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.

# A 'prescribing medication' story

**Prescribing medication**

The record of the patient must be open for input. Click on the medication field and select either 'current medication', 'new medication' or 'formulary'.

If you select 'current medication', you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

If you choose, 'new medication', the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose 'formulary', you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click 'OK' or 'Change'. If you click 'OK', your prescription will be recorded on the audit database. If you click 'Change', you reenter the 'Prescribing medication' process.

SUNG KYUN KWAN UNIVERSITY

# Examples of task cards for prescribing medication

**Task 1: Change dose of prescribed drug**

**Task 2: Formulary selection**

**Task 3: Dose checking**

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.
Using the formulary id for the generic drug name, lookup the formulary and retrieve the recommended maximum and minimum dose.
Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low. If within the range, enable the 'Confirm' button.

SUNG KYUN KWAN UNIVERSITY

# 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

- Programming team **look for possible software improvements** and make these improvements even where there is no immediate need for them.

- This improves the *understandability* of the software and so reduces the need for documentation.

- Changes are easier to make because the code is well-structured and clear.

- However, some changes requires architecture refactoring and this is much more expensive.

# Examples of refactoring

- Re-organization of a class hierarchy to remove duplicate code.

- Tidying up and renaming attributes and methods to make them easier to understand.

- The replacement of code with calls to methods that have been included in a program library.

# Key points

- **Agile** methods are **incremental development methods** that focus on rapid development, frequent releases of the software, reducing process overheads and producing high-quality code. They involve the customer directly in the development process.

- The decision on whether to use an agile or a plan-driven approach to development should depend on the type of software being developed, the capabilities of the development team and the culture of the company developing the system.

- **XP**(Extreme programming) is a well-known agile method that integrates a range of good programming practices such as *frequent releases* of the software, *continuous software improvement* and *customer participation* in the development team.

# Agile Software Development

## Part 2

# Testing in XP

- Testing is central to XP and XP has developed an approach where **the program is tested after every change has been made.**

- **XP testing features:**

  - 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.

# 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.**

  – Usually relies on a testing framework such as JUnit.

- **All previous and new tests are run automatically when new functionality is added,** thus checking that the new functionality has not introduced errors.
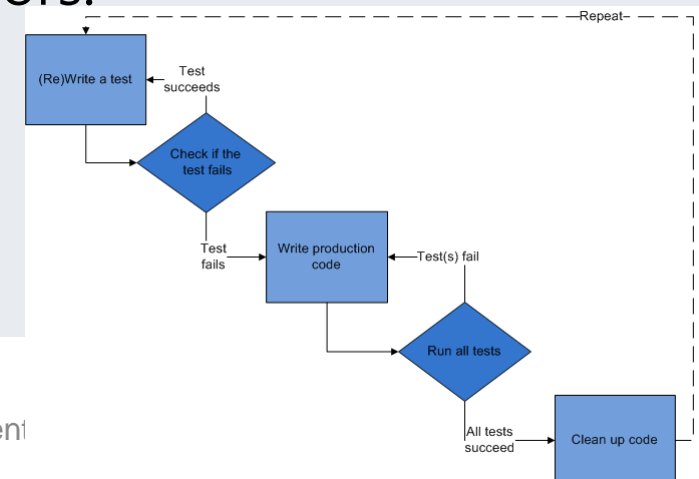
# Customer involvement

- **The role of the customer** in the testing process is to help *develop acceptance tests* for the stories that are to be implemented in the next release of the system.

- The customer who is **part of the team writes tests** as development proceeds. **All new code is therefore validated** to ensure that it is what the customer needs.

- However, **people adopting the customer role** have limited time available and so **cannot work full-time** with the development team. They may feel that **providing the requirements was enough** of a contribution and so may be **reluctant to get involved** in the testing process.

# Test case description for dose checking

**Test 4: Dose checking**

**Input:**
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

**Tests:**
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

**Output:**
OK or error message indicating that the dose is outside the safe range.

# Test automation

- **Test automation means that tests are written as executable components before the task is implemented**
  - These testing components should be stand-alone, should simulate the submission of input to be tested and should check that the result meets the output specification. An automated test framework (e.g. Junit) is a system that makes it easy to write executable tests and submit a set of tests for execution.
- **As testing is automated, there is always a set of tests that can be quickly and easily executed**
  - Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately.

# Problems with test-first development

- Programmers **prefer programming to testing** and sometimes they take short cuts when writing tests. For example, they may write incomplete tests that do not check for all possible exceptions that may occur.

- Some tests can be very difficult to write incrementally. For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the 'display logic' and workflow between screens.

- It is difficult to judge the completeness of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage.

# 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.

SUNG KYUN KWAN UNIVERSITY
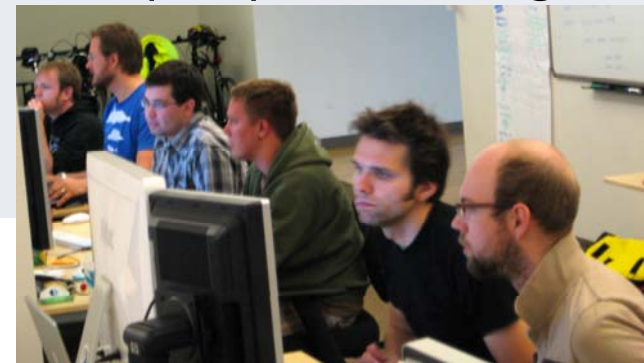
# 4. Agile project management

- The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.

- The standard approach to project management is plan-driven. Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.

- Agile project management requires a different approach, which is **adapted to incremental development** and the particular strengths of agile methods.

# Scrum

- The Scrum approach is a general agile method but its focus is on **managing iterative development** rather than specific agile practices.

- **There are three phases in Scrum.**

  - The initial phase is an **outline planning phase** where you establish the general objectives for the project and design the software architecture.

  - This is followed by **a series of sprint cycles**, where each cycle develops an increment of the system.

  - The project **closure phase** wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

# Scrum terminology (a)

| Scrum term | Definition |
|---|---|
| Development team | A self-organizing group of software developers, which should be no more than 7 people. They are responsible for developing the software and other essential project documents. |
| Potentially shippable product increment | The software increment that is delivered from a sprint. The idea is that this should be 'potentially shippable' which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable. |
| Product backlog | This is a list of 'to do' items which the Scrum team must tackle. They may be feature definitions for the software, software requirements, user stories or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation. |
| Product owner | An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative. |

# Scrum terminology (b)

| Scrum term | Definition |
|---|---|
| Scrum | A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team. |
| ScrumMaster | The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference. |
| Sprint | A development iteration. Sprints are usually 2-4 weeks long. |
| Velocity | An estimate of how much product backlog effort that a team can cover in a single sprint.  Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance. |

# The Scrum process (sprint cycle)
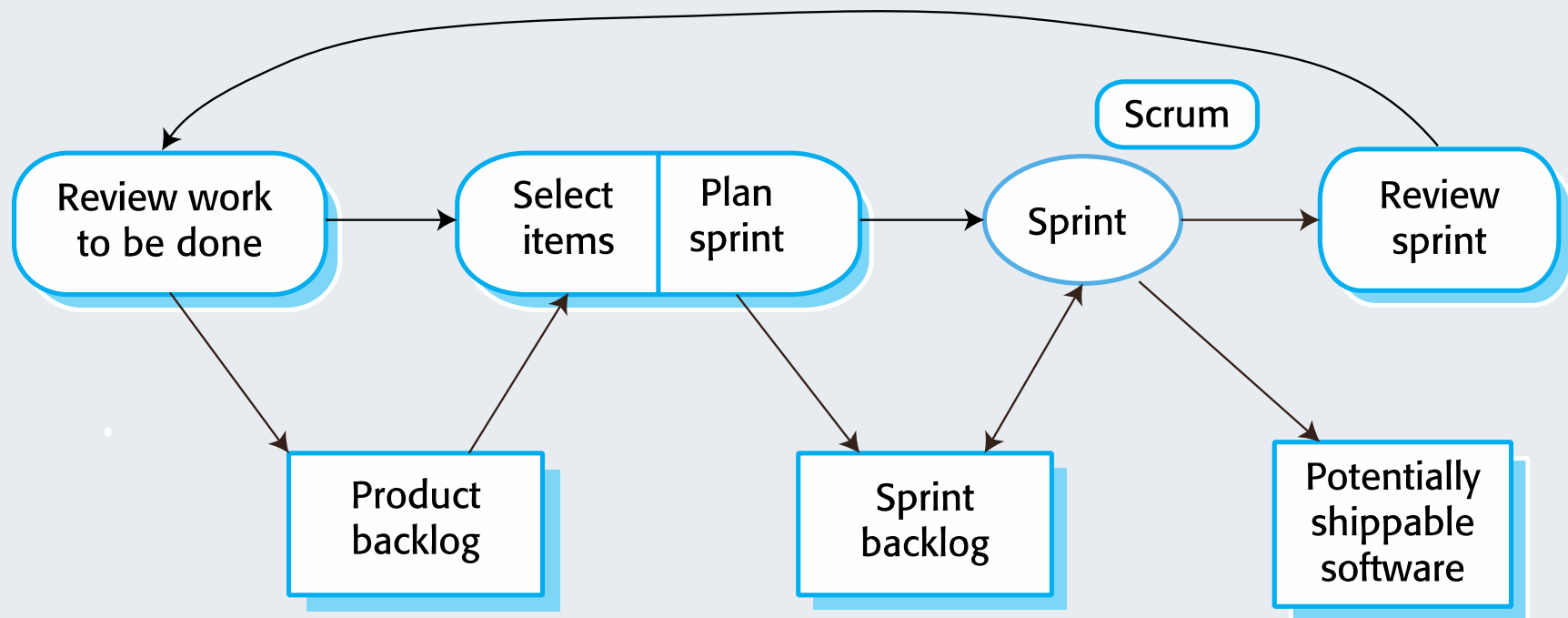
# The Sprint cycle

- Sprints are fixed length, **normally *2–4* weeks**. They correspond to the development of a release of the system in XP.

- The starting point for planning is the <span style="color:red">**product backlog**</span>, which is the **list of work to be done** on the project.

- The *selection phase* involves all of the project team who work with the customer to select the features and functionality to be developed during the sprint.

SUNG KYUN KWAN UNIVERSITY

# The Sprint cycle

- Once these are *agreed*, the team organize themselves to *develop* the software. **During this stage the team is isolated from the customer** and the organization, with all communications channelled through the so-called 'Scrum master'.

- The role of the Scrum master is to **protect the development team** from external distractions.

-  At the end of the sprint, the work done is *reviewed* and *presented* to stakeholders. The next sprint cycle then begins.
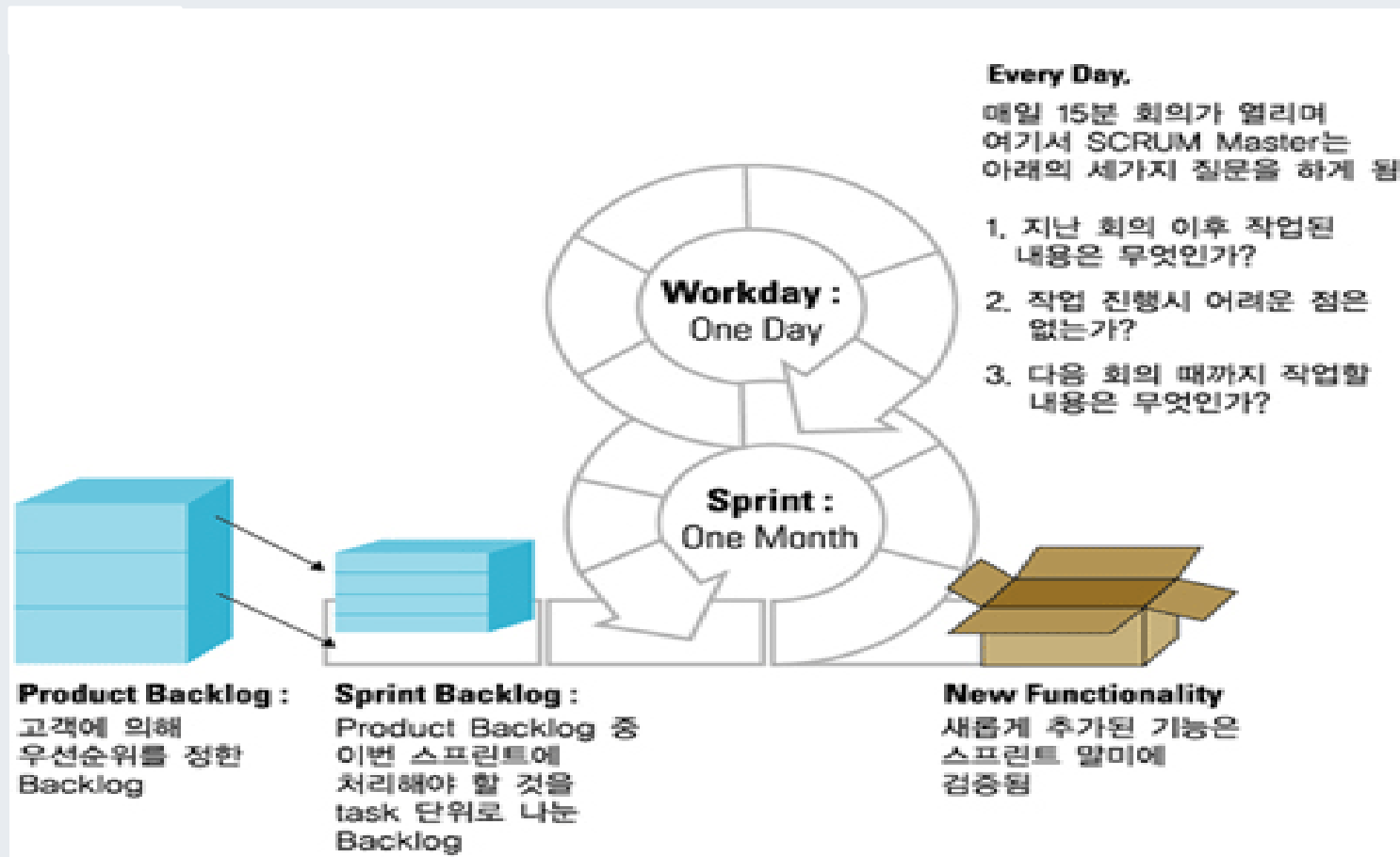
SUNG KYUN KWAN UNIVERSITY

# Teamwork in Scrum

- The 'Scrum master' is a facilitator who *arranges* daily meetings, *tracks* the backlog of work to be done, *records* decisions, *measures* progress against the backlog and *communicates* with customers and management outside of the team.

- The whole team attends short daily meetings where all team members *share* information, *describe* their progress since the last meeting, problems that have arisen and what is planned for the following day.

  – This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.
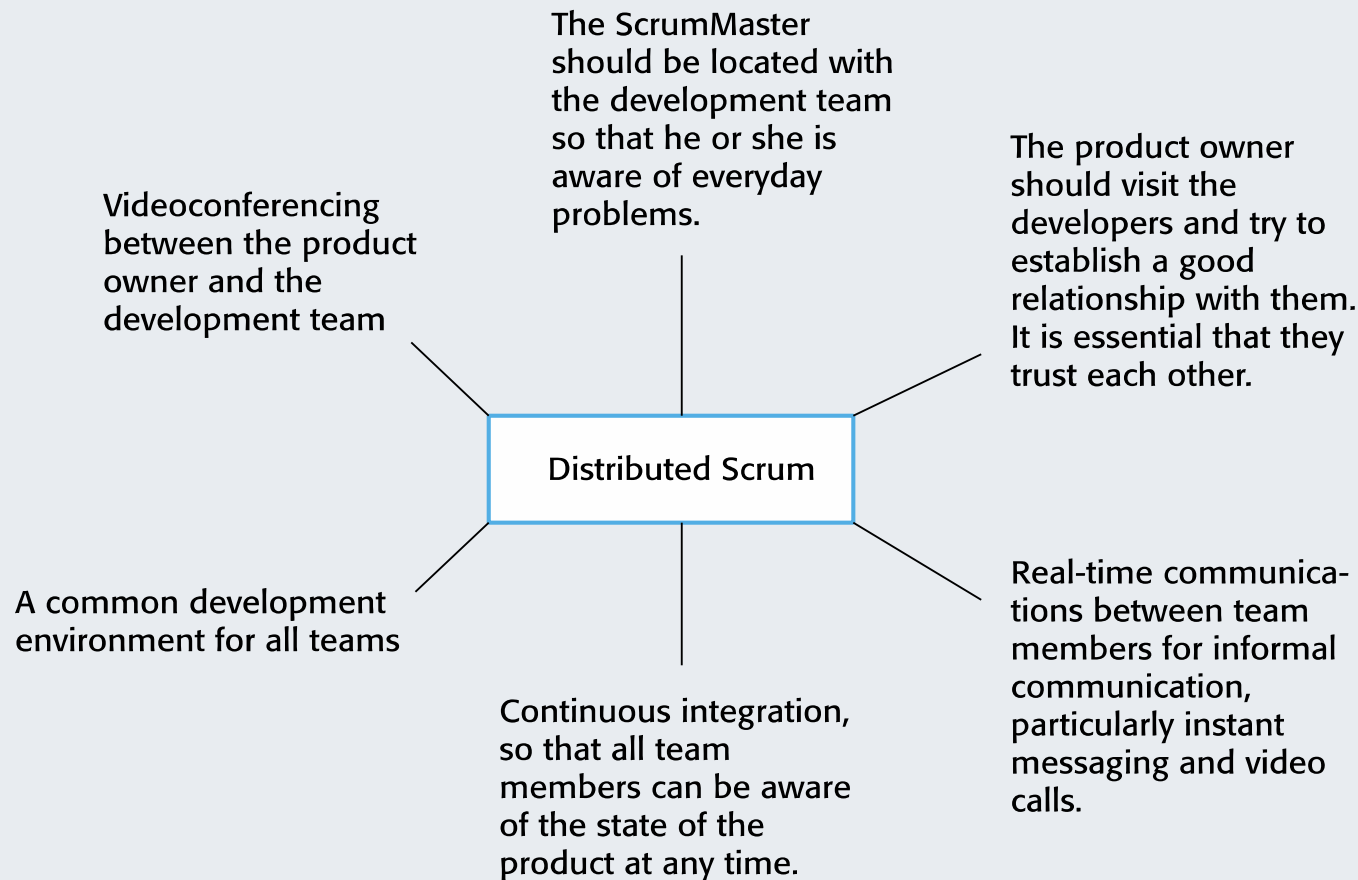
# Scrum benefits

- The **product is _broken down_ into a set of manageable and understandable chunks**.

- Unstable requirements _do not hold up progress_.

- The whole team have _visibility_ of everything and consequently team _communication_ is improved.

- Customers see _on-time delivery_ of increments and _gain feedback_ on how the product works.

- _Trust_ between customers and developers is established and a _positive culture_ is created in which everyone expects the project to succeed.

SUNG KYUN KWAN UNIVERSITY

# An Example of SCRUM Sprint Cycle in Korea

# Distributed Scrum

The ScrumMaster should be located with the development team so that he or she is aware of everyday problems.

Videoconferencing between the product owner and the development team

The product owner should visit the developers and try to establish a good relationship with them. It is essential that they trust each other.

**Distributed Scrum**

A common development environment for all teams

Continuous integration, so that all team members can be aware of the state of the product at any time.

Real-time communications between team members for informal communication, particularly instant messaging and video calls.

# 5. Scaling agile methods

- Agile methods have proved to be successful for small and medium sized projects that can be developed by a small co-located team.

- It is sometimes argued that the success of these methods comes because of *improved communications* which is possible when everyone is working together.

- Scaling up agile methods involves changing these **to cope with larger, longer projects** where there are multiple development teams, perhaps working in different locations.

# Large systems development

- Large systems are usually collections of separate, communicating systems, where separate teams develop each system. Frequently, these teams are working in different places, sometimes in different time zones.

- Large systems are 'brownfield systems', that is they include and interact with a number of existing systems. Many of the system requirements are concerned with this interaction and so don't really lend themselves to flexibility and incremental development.

- Where several systems are integrated to create a system, a significant fraction of the development is concerned with *system configuration* rather than original code development.

# Large systems development

- Large systems and their development processes are often *constrained by* external rules *and* regulations *limiting the way that they can be developed.*

- Large systems have a long procurement *and* development time. It is *difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects.*

- Large systems usually have a *diverse set of stakeholders. It is practically impossible to involve all of these different stakeholders in the development process.*

# Scaling out and scaling up

- '<span style="color:red">Scaling up</span>' is concerned with using agile methods *for developing* <u>*large software systems*</u> that cannot be developed by a small team.

- '<span style="color:red">Scaling out</span>' is concerned with how agile methods can be *introduced across a* <u>*large organization*</u> with many years of software development experience.

- When scaling agile methods it is essential to *maintain agile fundamentals*

  – Flexible planning, frequent system releases, continuous integration, test-driven development and good team communications.

# Contractual issues of agile method

- Most software contracts for custom systems are based around a specification, which sets out what has to be implemented by the system developer for the system customer.
- However, this precludes interleaving specification and development as is the norm in agile development.
- A contract that pays for developer time rather than functionality is required.
  - However, this is seen as a high risk my many legal departments because what has to be delivered cannot be guaranteed.

# Agile method and maintenance

- Most organizations spend more on maintaining existing software than they do on new software development.

- Key problems are:

  – Lack of product documentation

  – Keeping customers involved in the development process

  – Maintaining the continuity of the development team

- Agile development relies on the development team knowing and understanding what has to be done.

- For long-lifetime systems, this is a real problem as the original developers will not always work on the system.

SUNG KYUN KWAN UNIVERSITY

# Key points

- A particular strength of extreme programming is the development of automated tests before a program feature is created. All tests must successfully execute when an increment is integrated into a system.

- The Scrum method is an agile method that provides a project management framework. It is centred round a set of sprints, which are fixed time periods when a system increment is developed.

- Scaling agile methods for large systems is difficult. Large systems need up-front design and some documentation.
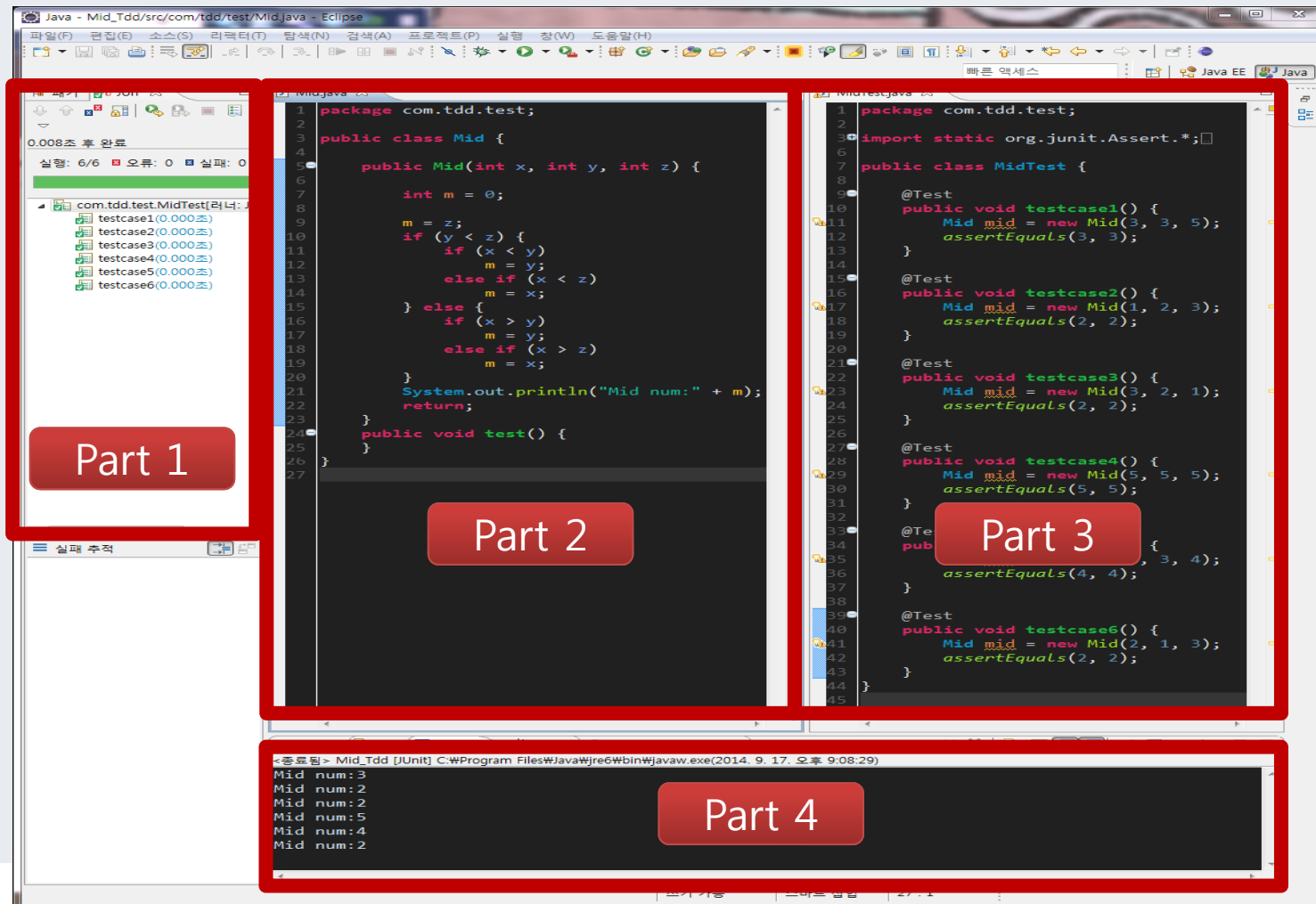
# Appendix

## Example of TDD
## - Mid function

SUNG KYUN KWAN UNIVERSITY

# Appendix –Test Driven Development

- **Mid()**
  - Function to return an intermediate value when three integers are inserted

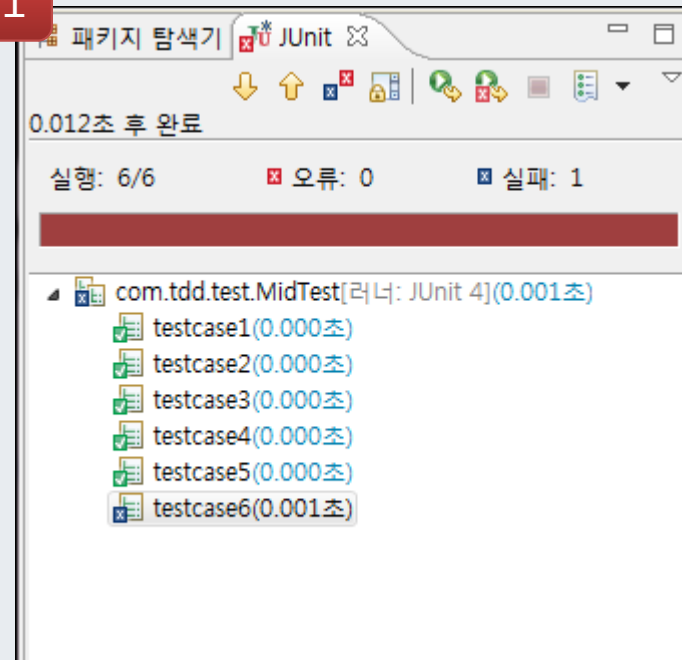| Mid() {<br>int x,y,z,m; | Test Cases | | | | | |
|---|---|---|---|---|---|---|
| | 3,3,5 | 1,2,3 | 3,2,1 | 5,5,5 | 5,3,4 | 2,1,3 |
| 1:  read("Enter 3 numbers:",x,y,z); | ● | ● | ● | ● | ● | ● |
| 2:  m = z; | ● | ● | ● | ● | ● | ● |
| 3:  if (y<z) | ● | ● | ● | ● | ● | ● |
| 4:    if (x<y) | ● | ● | | | ● | ● |
| 5:      m = y; | | ● | | | | |
| 6:    else if (x<z) | ● | | | | ● | ● |
| **7:      m = y;  // *** bug *** ** | ● | | | | | ● |
| 8:  else | | | ● | ● | | |
| 9:    if (x>y) | | | ● | ● | | |
| 10:      m = y; | | | ● | | | |
| 11:    else if (x>z) | | | | ● | | |
| 12:      m = x; | | | | | | |
| 13: print("Middle number is:",m); | ● | ● | ● | ● | ● | ● |
| }            Pass/Fail Status | P | P | P | P | P | F |

# Appendix –Test Driven Development

- **Java code for Mid() with Eclipse**

# Appendix –Test Driven Development

- **Part 1**
- Indicates the success or failure of the tested test case

# Appendix –Test Driven Development

Part 2

- **Part 2**
- Java code for Mid()

```java
package com.tdd.test;

public class Mid {

    public Mid(int x, int y, int z) {

        int m = 0;

        m = z;
        if (y < z) {
            if (x < y)
                m = y;
            else if (x < z)
                m = y;   //bug
        } else {
            if (x > y)
                m = y;
            else if (x > z)
                m = x;
        }
        System.out.println("Mid num:" + m);
        return;
    }
    public void test() {
    }
}
```

SUNG KYUN KWAN UNIVERSITY

# Appendix –Test Driven Develo...

- **Part 3**
- Task for TDD
  - Conduct development (Part 2), while performing test at the same time.

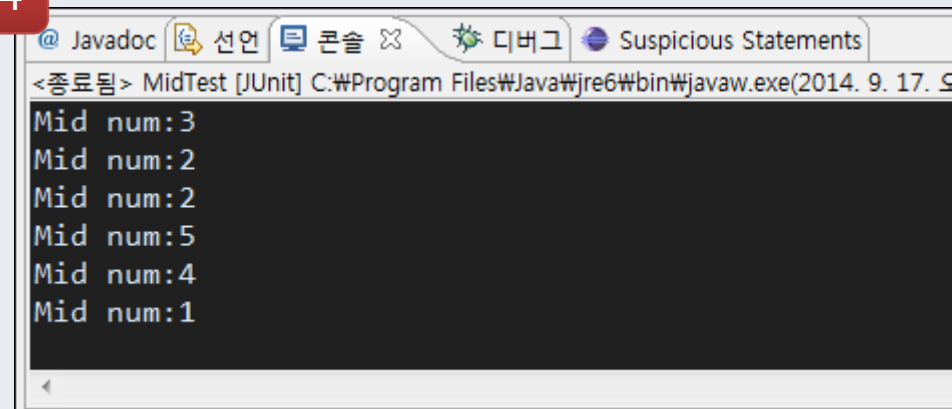| Mid() {<br>int x,y,z,m; | Test Cases | | | | | |
|---|---|---|---|---|---|---|
| | 3,3,5 | 1,2,3 | 3,2,1 | 5,5,5 | 5,3,4 | 2,1,3 |
| 1: read("Enter 3 numbers:",x,y,z); | ● | ● | ● | ● | ● | ● |
| 2: m = z; | ● | ● | ● | ● | ● | ● |
| 3: if (y<z) | ● | ● | ● | ● | ● | ● |
| 4: if (x<y) | ● | ● | | | ● | ● |
| 5: m = y; | | ● | | | | |
| 6: else if (x<z) | ● | | | | ● | ● |
| 7: **m = y; // *** bug ***** | ● | | | | | ● |
| 8: else | | | ● | ● | | |
| 9: if (x>y) | | | ● | ● | | |
| 10: m = y; | | | | ● | | |
| 11: else if (x>z) | | | | ● | | |
| 12: m = x; | | | | | | |
| 13: print("Middle number is:",m); | ● | ● | ● | ● | ● | ● |
| } Pass/Fail Status | P | P | P | P | P | F |

```java
package com.tdd.test;

import static org.junit.Assert.*;

public class MidTest {

    @Test
    public void testcase1() {
        Mid mid = new Mid(3, 3, 5);
        assertEquals(3, 3);
    }

    @Test
    public void testcase2() {
        Mid mid = new Mid(1, 2, 3);
        assertEquals(2, 2);
    }

    @Test
    public void testcase3() {
        Mid mid = new Mid(3, 2, 1);
        assertEquals(2, 2);
    }

    @Test
    public void testcase4() {
        Mid mid = new Mid(5, 5, 5);
        assertEquals(5, 5);
    }

    @Test
    public void testcase5() {
        Mid mid = new Mid(5, 3, 4);
        assertEquals(4, 4);
    }

    @Test
    public void testcase6() {
        Mid mid = new Mid(2, 1, 3);
        assertEquals(2, 1);
    }
}
```

# Appendix –Test Driven Development

- **Part 4**
- Console - When you insert a test case, the intermediate value is printed to the console via Mid ().

Part 4