NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Tutorial #4 – Software Processes

Dr. Ma Wei

Activities included in Software Process

Specification

- Requirements Analysis

Design

- Modelling UML -> Model Checking

Implementation

Techniques

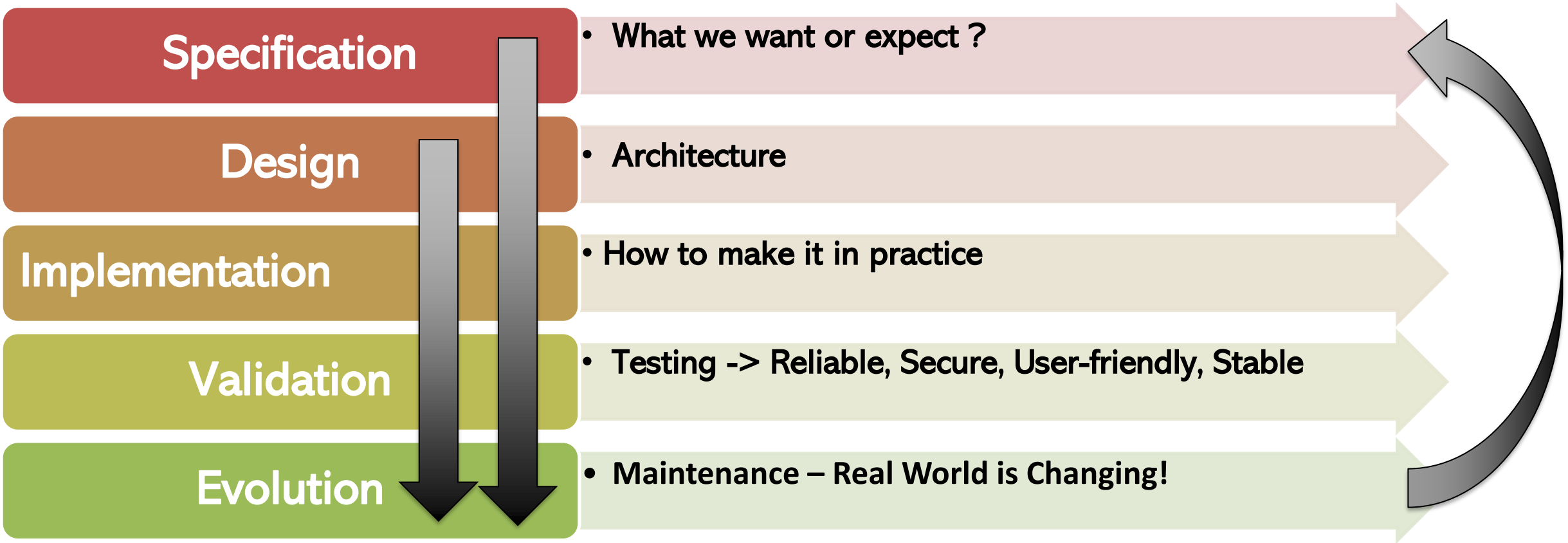
Validation

- Testing, e.g., verification, mutation testing, fuzzing, unit testing , UI testing, system testing….

Evolution

- Maintenance

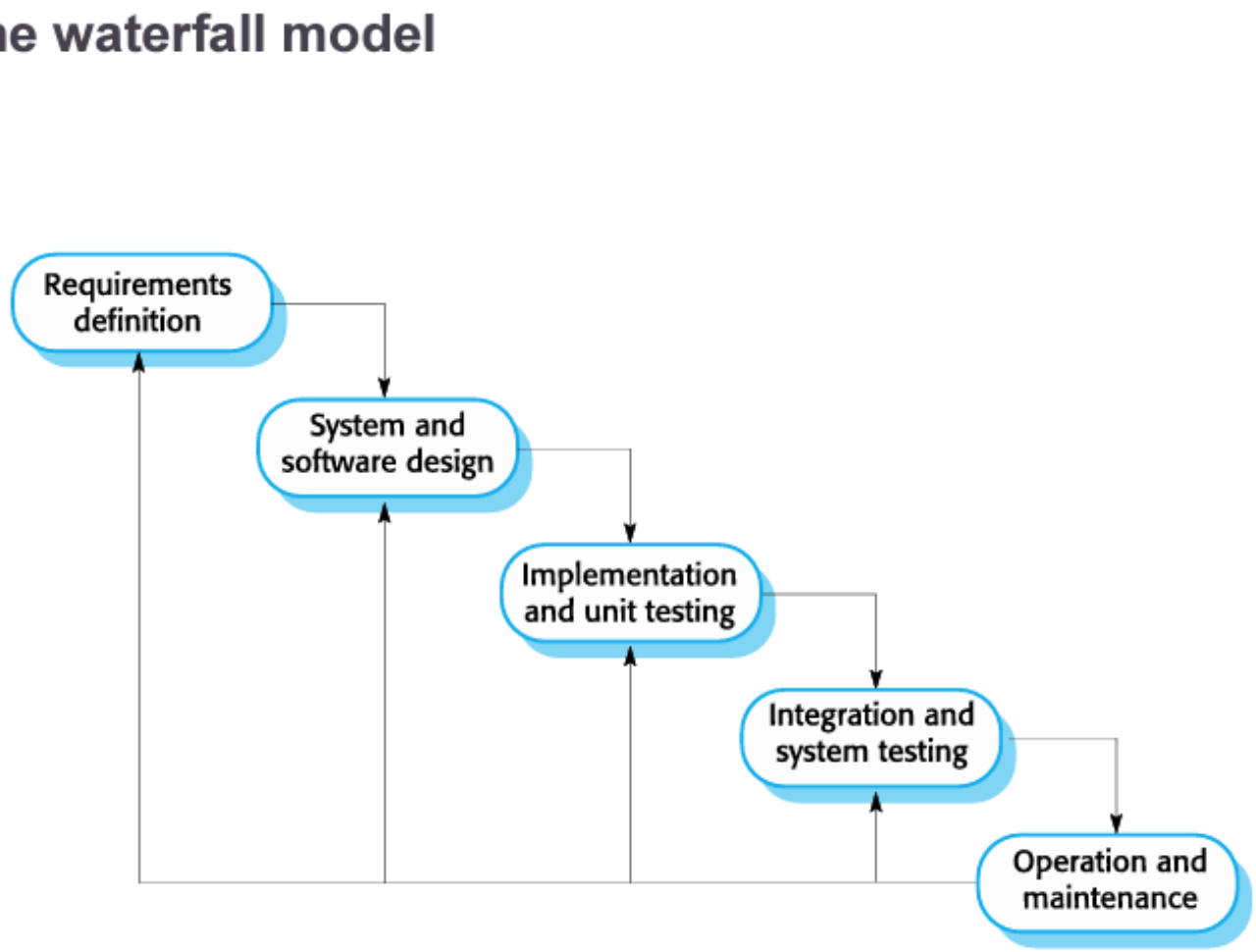
Activities included in Software Process



Software Development Model

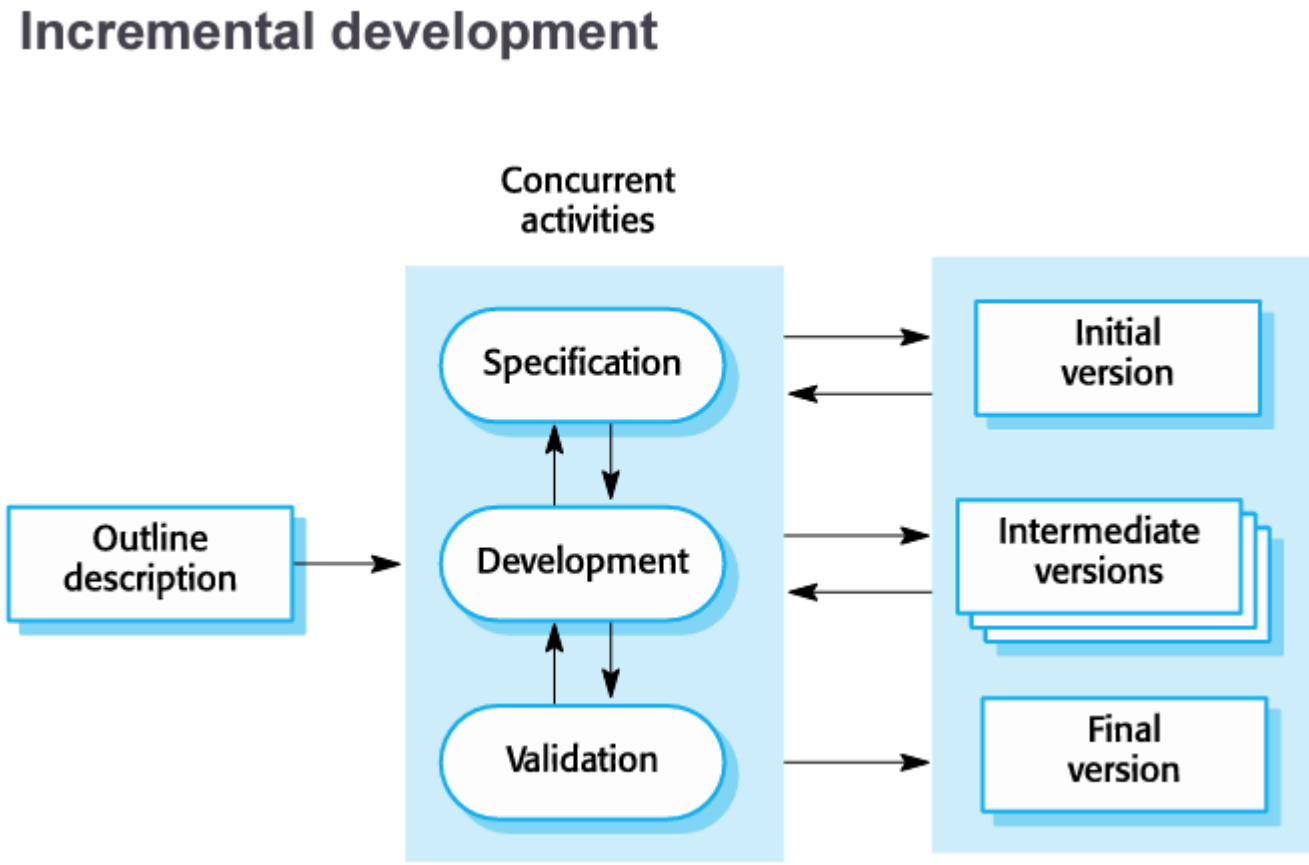
Step by Step

- Any pre-mistake will propagate
- Not Flexible
- High-Cost to maintain



Software Development Model

- Interleaved
- Flexible
- Require Expert Experience
- Frequent Change



Question 1

Why are software development costs so high?

- Unacceptably low quality requires more rework (design, implementation, and testing) than expected.
- Development of the wrong software functions requires redesign and implementation.
- Development of the wrong user interface results in redesign and implementation.
- Development of extra software functions that are not required extends the schedule.

Question 2

Why do we spend so much time and effort maintaining existing software programs?

- Software systems are subject to continuing change even after it is built.
- There are many staff and organizational reasons that make maintenance difficult
  - limited understanding;
  - morale;
  - management priorities.
- Technical problems also affect maintenance productivity
  - inadequate design specification;
  - low-quality programs and documentation;
  - testing difficulties.
- Real-time and highly synchronized systems are more difficult to change.
- A system designed to last many years is likely to require more maintenance.
- System dependent on its hardware's characteristics is likely to require many changes.

Question 3

Discuss on the characteristics of software that affect its development.

- Malleability

  Software is easy to change (all programmers are often tempted to 'tweak' their code). This malleability creates a constant pressure for software to be changed rather than replaced.

- Complexity

  Software is often complex. Complexity can usually be recognized, but it is less easy to define. One item of software can be considered more complex than another if the description of the first requires more explanation than that of the second. Part of the complexity arises from the potential variety of pathways between the components of a system.

- Size

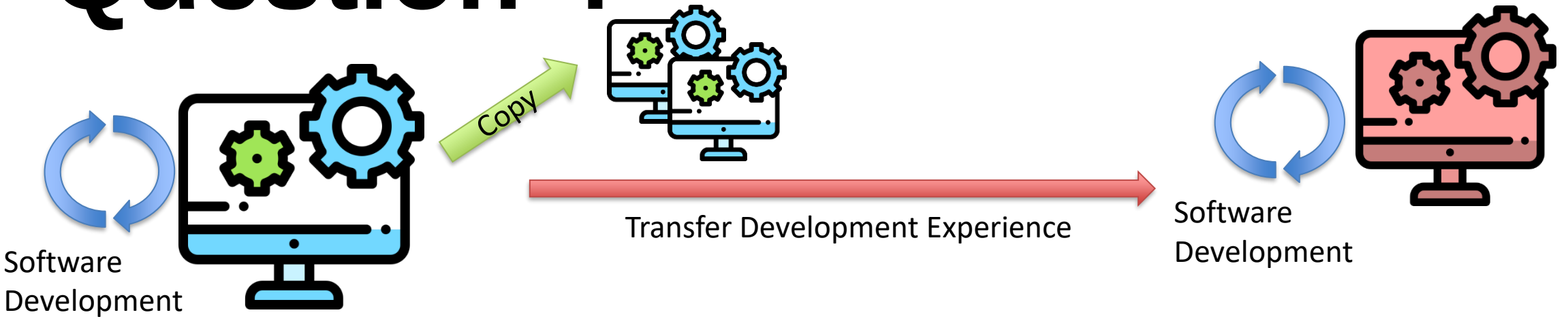
  It is likely that there will be more errors in a large piece of software than there will be in a small one. Evidence suggests that the number of errors increases roughly in proportion to the square of the size of the software. All things being equal, an item of software that is twice as big as another is likely to have four times the number of errors.

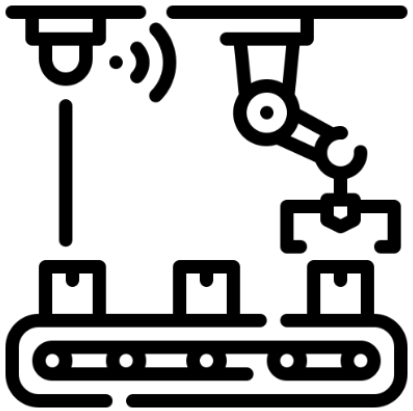
Question 4

What are the two distinct differences between software development and hardware manufacturing?

- In software development,
  - we do not produce the same product again and again based on the same process and design,
  - but we do that in manufacturing: In manufacturing, for producing an identical copy of the product, we need to run the same process based on the same design. As such, we produce the same product again and again based on the same design and process. Proper statistical analysis can be carried out to analyze the earlier feedbacks and experiences to improve the latter production.
  - In software development, for producing an identical product, we just need to make another copy without any further effort. When we carry out a new software development, it is surely that we will base on different requirement and/or different design.
  - It is hard for the later development to benefit from the earlier experiences and feedbacks through proper statistical analysis as they are carried out on different basis.
- Software does not wear out, as a result, maintenance does not mean component replacement: As such, there is no clear cut timing to phase out a piece of software. As a result, users are likely to pressurize software engineers to incorporate more and more new requirements after a system is implemented. This is likely to corrupt the overall structure of the software and deteriorate its quality.
- Due to the above key differences, requirements engineering are carried out more frequently in software development.

Question 4





1. Work pipeline to make the same products
2. Maintenance -> Relace broken components
3. Statistical analysis, e.g., what is the ratio of the flaw products?

Question 5

Sum up Agile manifesto in four sentences, then discuss the key Agile principles.

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Link: https://agilemanifesto.org/

Question 5 (cont'd)

Sum up Agile manifesto in four sentences, then discuss the key Agile principles.

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Question 6

Discuss the following characteristics of eXtreme Programming (XP):

i. When to use XP

ii. Pair programming

iii. Test-driven development (TDD)

iv. Code refactoring

- Involve new or prototype technology, where the requirements change rapidly, or some development is required to discover unforeseen implementation problems.
  - Are research projects, where the resulting work is not the software product itself, but domain knowledge.
  - Are small and more easily managed through informal methods.

Question 6 (cont'd)

Discuss the following characteristics of eXtreme Programming (XP):

i. When to use XP

ii. Pair programming

iii. Test-driven development (TDD)

iv. Code refactoring

- All production software in XP is built by two programmers, sitting side by side, at the same machine.
- While the first developer focuses on writing, the other one reviews code, suggests improvements, and fixes mistakes along the way.
- All production code is reviewed by at least one other programmer, and results in better design, better testing, and better code and ultimately a higher-quality software.
- Faster knowledge sharing.

Question 6 (cont'd)

Discuss the following characteristics of eXtreme Programming (XP):

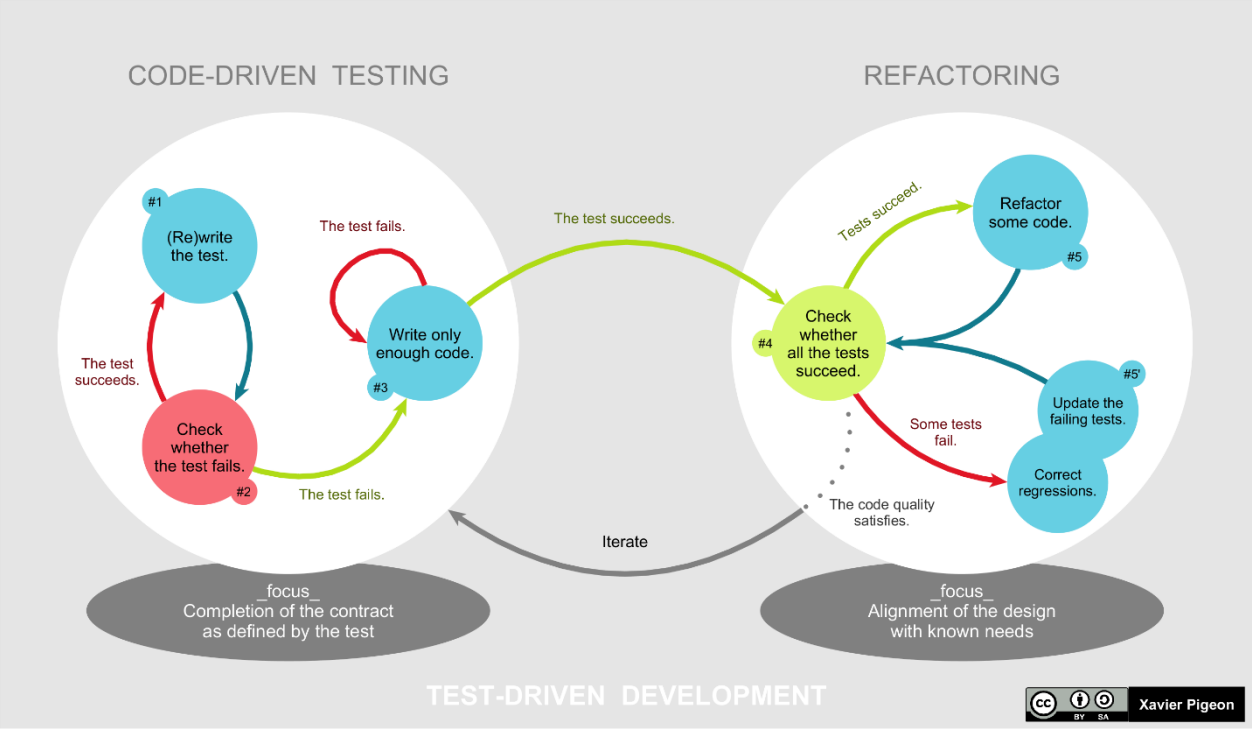
i. When to use XP

ii. Pair programming

iii. Test-driven development (TDD)

iv. Code refactoring

- Refers to a style of programming in which 3 activities are tightly interwoven:
  - Coding
  - Testing (in the form of writing unit tests)
  - Design (in the form of refactoring).



Question 6 (cont'd)

Discuss the following characteristics of eXtreme Programming (XP):

i. When to use XP

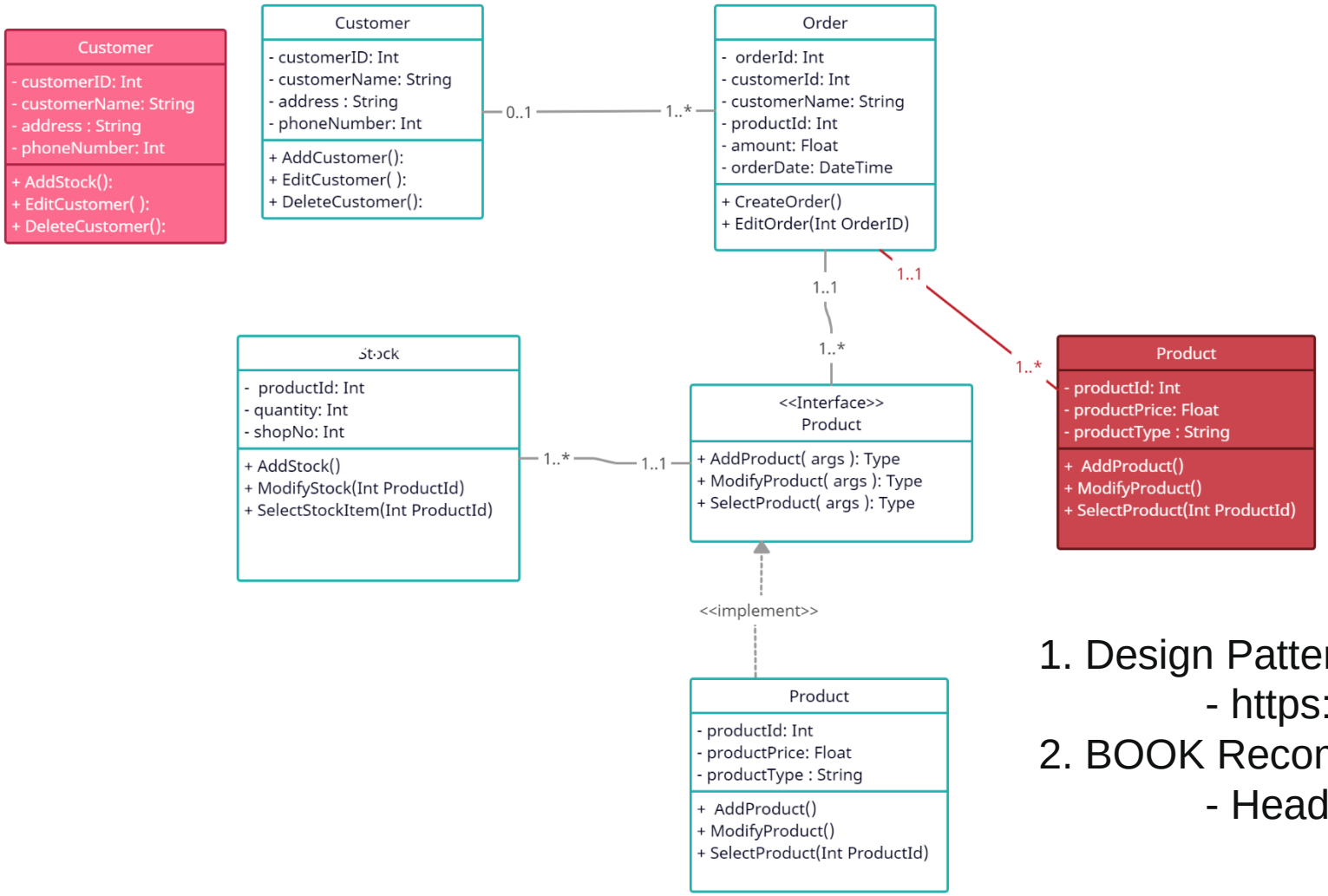
ii. Pair programming

iii. Test-driven development (TDD)

iv. Code refactoring

- The refactoring process focuses on:
  - Removal of duplication
  - Increasing the "cohesion" of the code, while lowering the "coupling"
- A well designed code must be HIGH cohesion and LOW coupling

HIGH cohesion and LOW coupling



Design Pattern

1. Design Pattern Tutorial
   - https://github.com/iluwatar/java-design-patterns
2. BOOK Recommendation for Beginners
   - Head First Design Patterns