SE202
Introduction to Software Engineering

Lecture 2-1
Software Development Life Cycle and Processes

Pathathai Na Lumpoon

Last Lecture

>> What is Software Engineering?

>> Need of Software Engineering

- to control complex development +
- to apply effective methods and tools +
- to choose the right approach for different software development

>> Role of Software Engineering (SE)

- customer's problem + development + quality + limitation

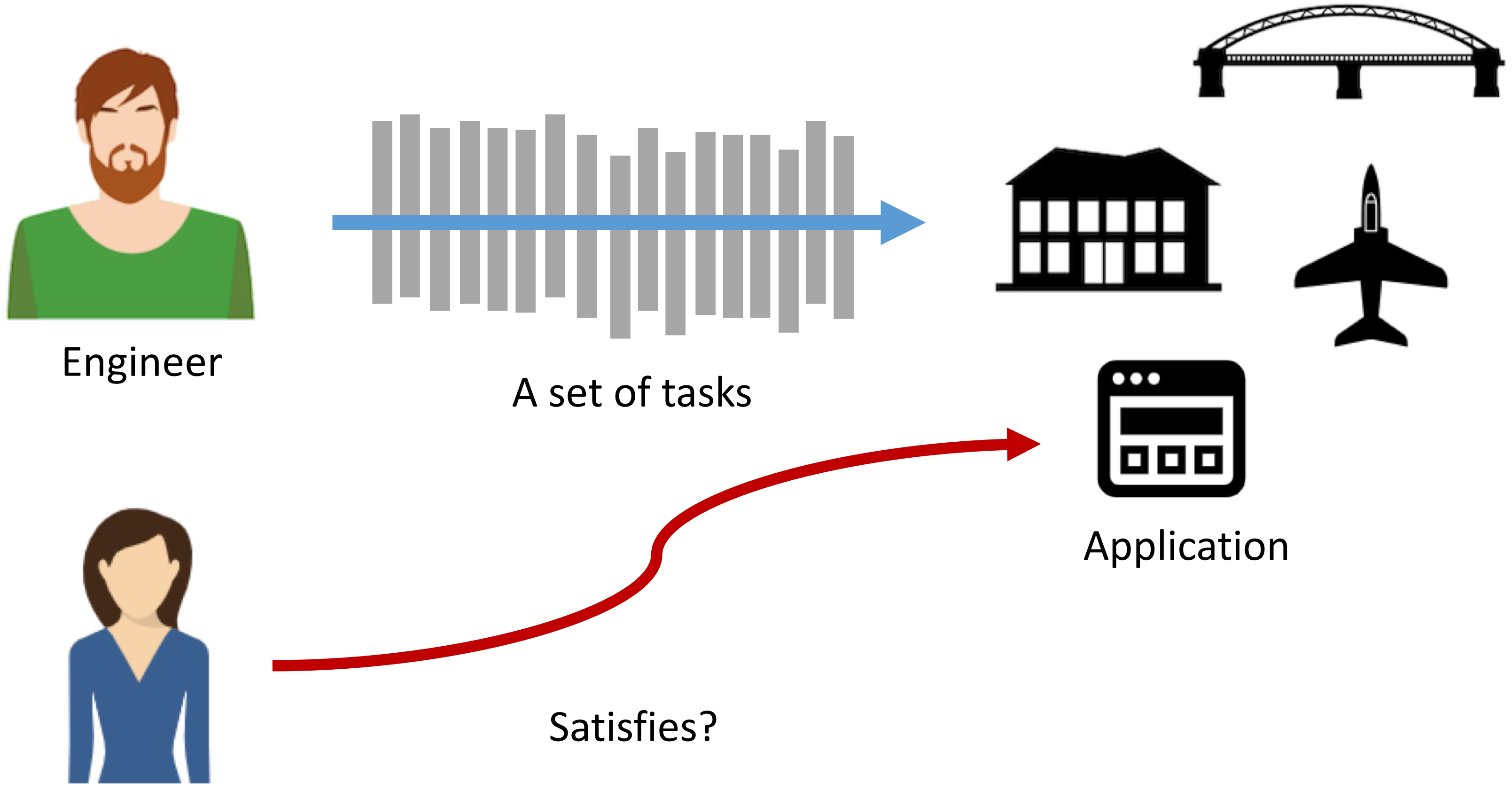
>> SE Ethics to keep in mind

- Respect to yourself + your team + your customer + your society

Today’s Topics

- Software Development Life Cycle (SDLC)
  - Core process of software development
  - Activities to perform

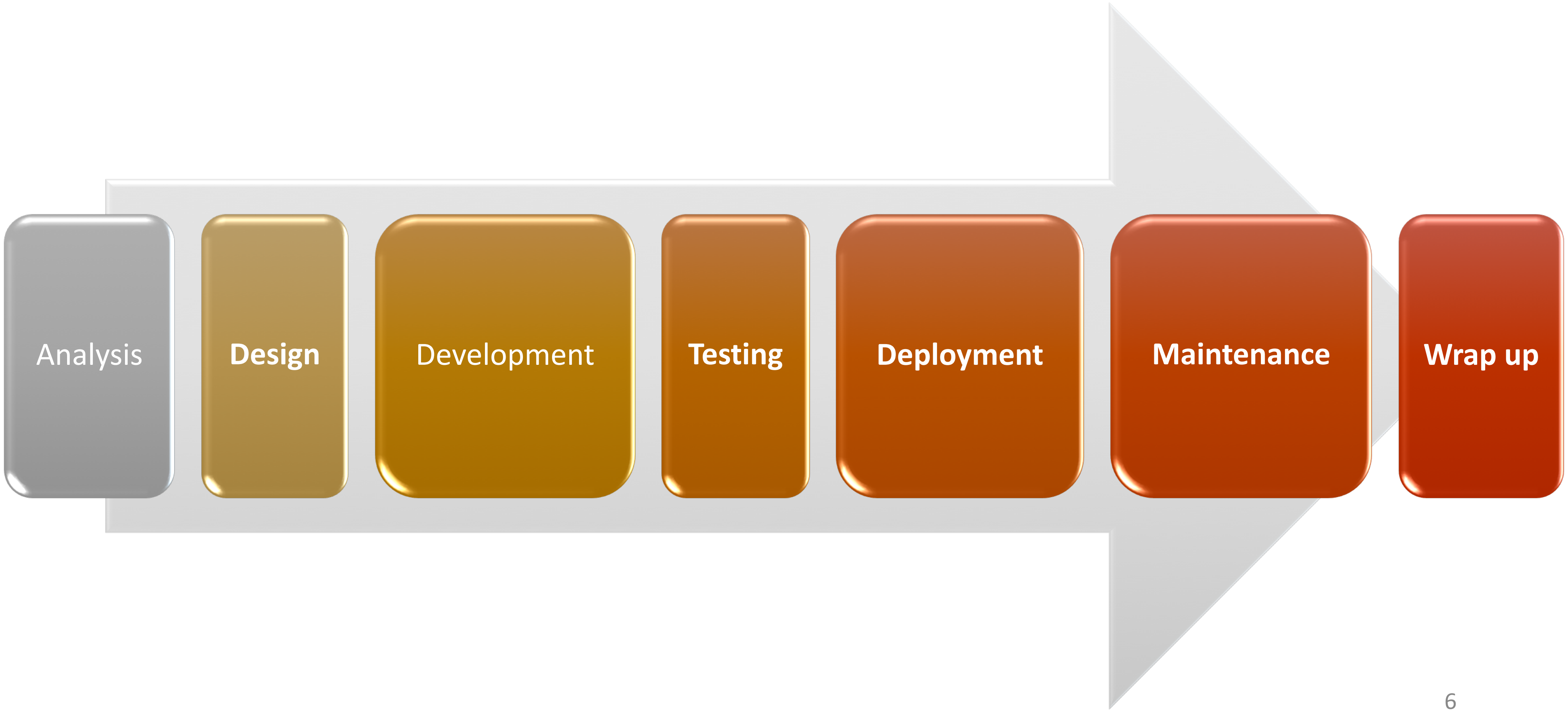
Basic Problem



Ad-hoc software development

- **Ad-hoc development**: creating software *without any formal guidelines* or process
- What are some *disadvantages* of ad-hoc development?
  - some important actions (testing, design) may go ignored
  - not clear when to start or stop doing each task
  - does not scale well to multiple people
  - not easy to review or evaluate one's work
- Cause of software development failure during 1960s which later caused the ‘Software Crisis’

Software Development Life Cycle (SDLC)



SDLC

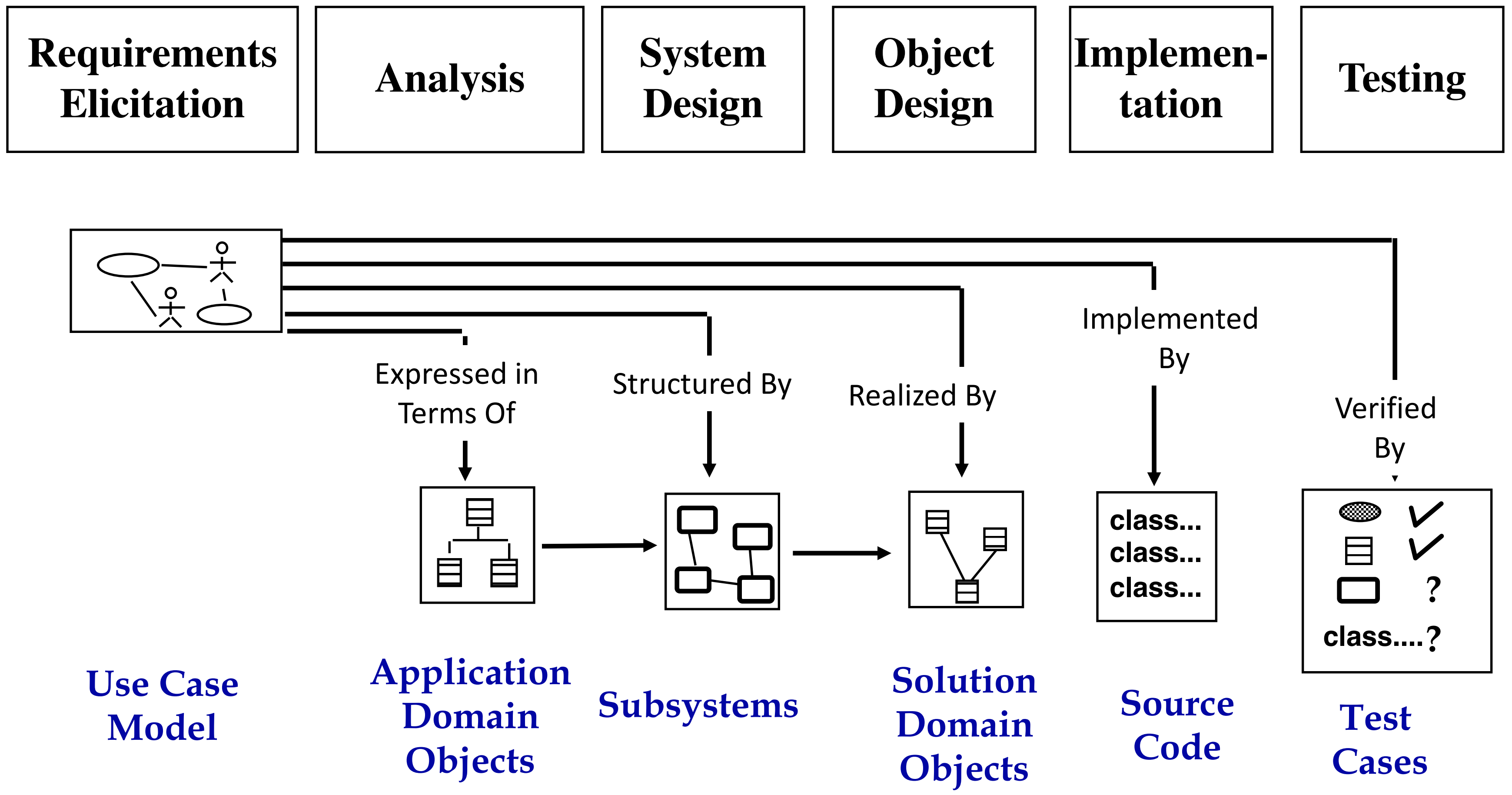
- Originated in 1960s
- SDLC is a framework defining tasks to perform in each software development process.
    1. **Requirement Analysis and Specification** – defining what the system should do
    2. **Design** – defining design a software structure that realises the specification
    3. **Development, Integration** – coding and combining code of the system
    4. **Testing** – checking that it does work correctly and does what the customer wants
    5. **Deployment** – rolling the system out to the users
    6. **Maintenance** – changing the system in response to changing customer needs.
    7. **Wrap up** – evaluating the whole project

- It can take months or years to complete.

SDLC

- Goals of each phase:
  - mark out a clear *set of steps* to perform
  - produce a *tangible document* or item
  - allow for *review* of work
  - specify *actions* to perform *in the next phase*

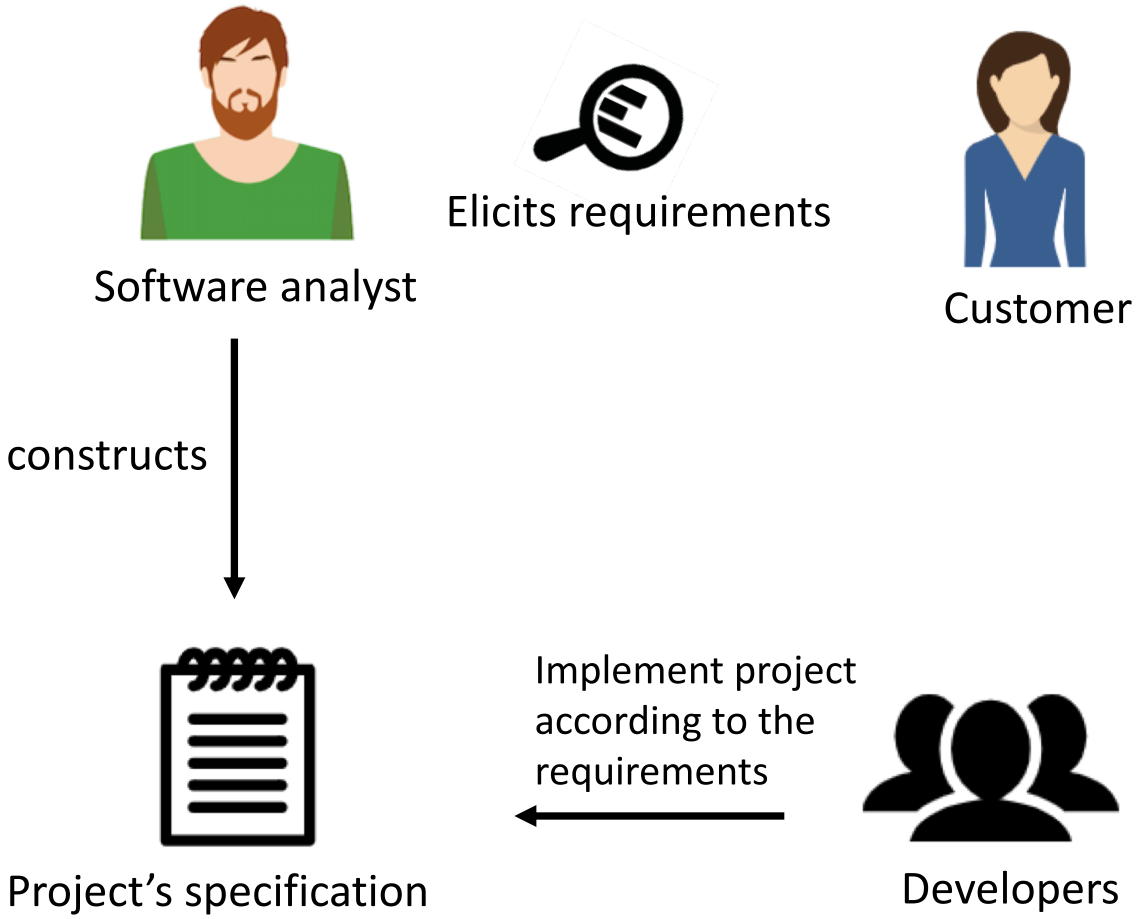
One view of SW cycle phases



Requirement Analysis and Specification

- To solve problems by understanding
  - Customer’s problems
  - Business environment
  - Available technologies
- The process of establishing *what products and services* are required and the *constraints* on the system’s operation and development.
- Keys
  - Identify customers and try to interact with them as much as possible to insight what the customers want and what the customers need

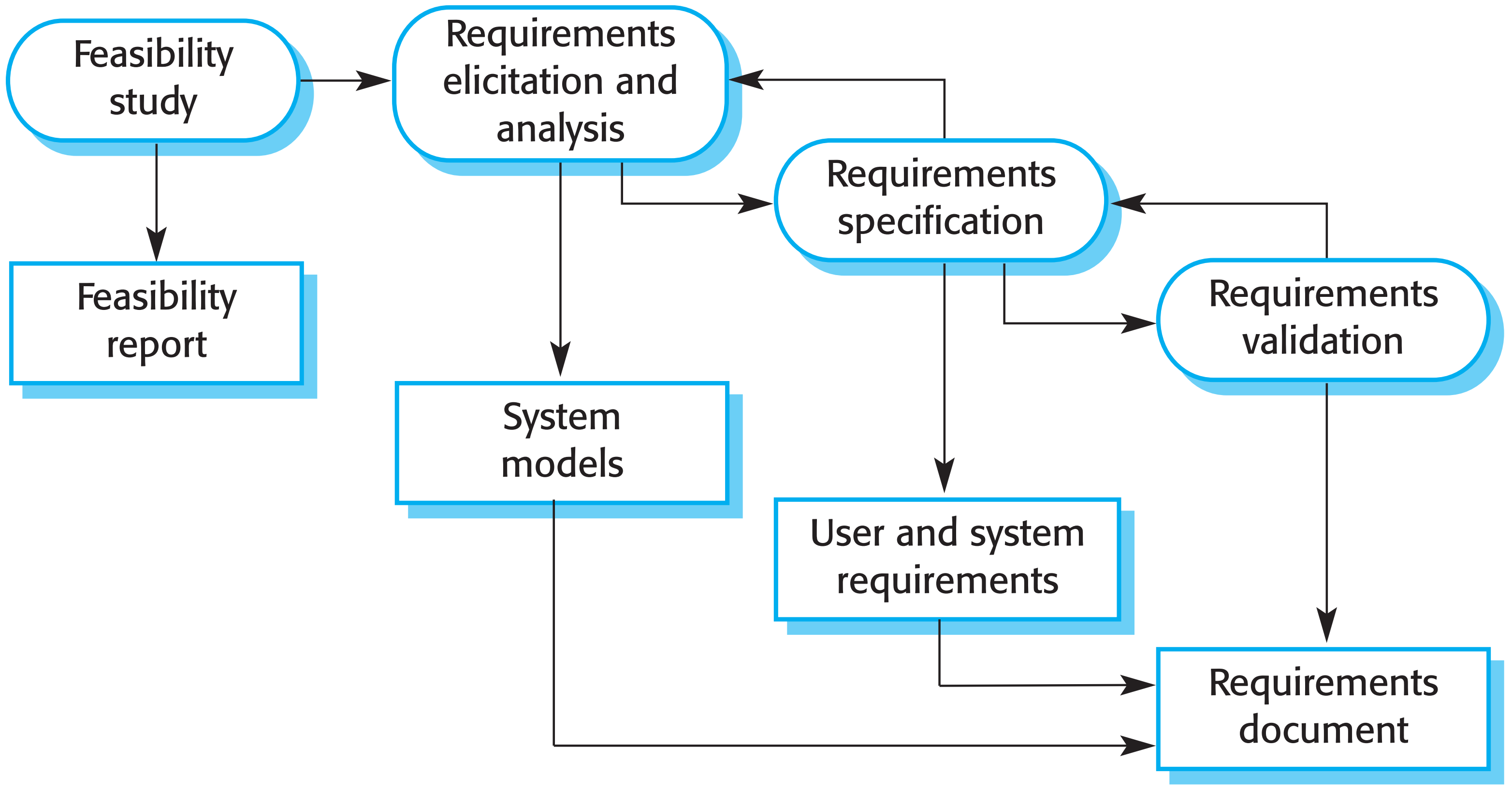
Requirement Analysis and Specification



Changes often happen.

What can happen when the project gets change request?

The requirements engineering process



HL Design

High Level Design

- Deciding how the requirements should be ***structured***
  - what platform to use
    - such as desktop, laptop, tablet, or phone
  - what data design to use
    - such as direct access, 2-tier, or 3-tier
  - And interfaces with other systems
    - such as external purchasing systems

Note: make sure that the high-level design covers every aspect of the requirements

- E.g. A system to manage the results of ostrich races

➤ Database (to hold the data)
➤ Classes (for example, Race, Ostrich, and Jockey classes)
➤ User interfaces (to enter Ostrich and Jockey data, enter race results, produce result reports, and create new races)
➤ External interfaces (to send information and spam to participants and fans via e-mail, text message, voice mail, and anything else we can think of)

LL Design

Low Level Design

- After high-level designs are done, the low-level designs begin.
- The low-level design includes information about how that piece of the project should work
- E.g.
  - the ostrich racing application’s database piece
    - An initial design for the database.
      - The tables of the race, ostrich, and jockey information.
- Changing the design here and there may be happened since the interactions between the different pieces of the project is discovered

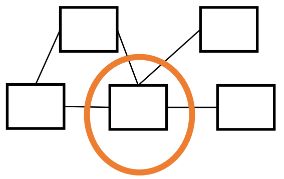
Development

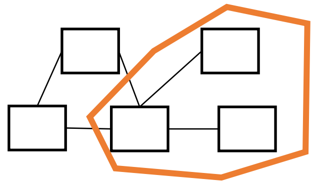
Software development

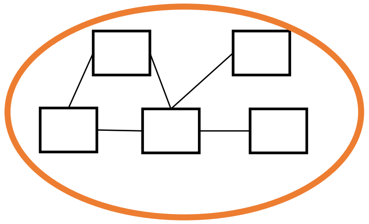
- Development
  - Translate *design structure* into an *executable program*;
- The programmers continue refining the low-level designs until they know how to implement those designs in code.
- As the programmers write the code, they test it to find and remove as many bugs as they reasonably can.

Software Testing

- Software testing involves checking the system does work correctly and does what the customer wants.
- The systems are tested by testers who didn't write the code.


Unit testing


Integration testing

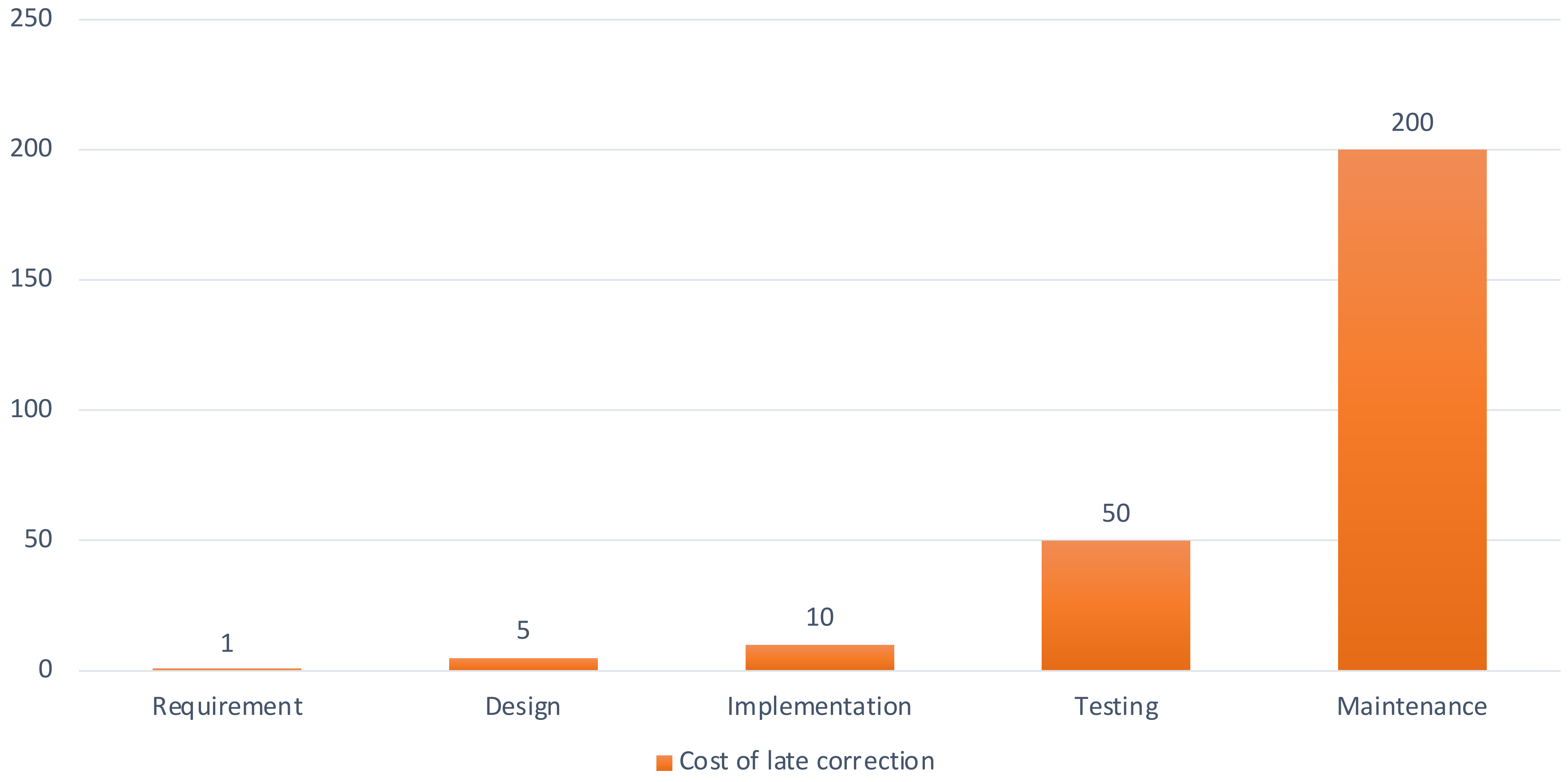

System testing

- Fixing a bug can lead to other bugs.
  - The bug fix may be incorrect.
  - The fix may break other code that depended on the original buggy behavior.
  - The fix might change some correct behavior to a new correct behavior, but another piece of code might depend on the original behavior.
- When to stop retesting?

Quiz: Counting bugs

- Suppose requirements analysis, high-level design, low-level design, and development works as follow
  - Every time you make a decision, the next task in the sequence includes two more decisions that depend on the first one.
  - For example, when you make a requirements decision, the high-level design includes two decisions that depend on it.
- Now suppose you made a mistake during requirements gathering.
  - How many incorrect decisions If you detect the error during the requirements analysis phase?
  - How many incorrect decisions If you detect the error during development phase?

Cost of late correction



Deployment

Software deployment

- All of the activities that make a software system available for use.
  - E.g. releasing a new variant of Tetris on the Internet
- Deployment can be difficult, time-consuming, and expensive.
  - E.g. suppose you've written a new billing system to track payments from your company's millions of customers.
  - Deployment might involve any or all of the following:
    - New computers for the back-end database
    - A new network
    - New computers for the users
    - User training
    - On-site support while the users get to know the new system
    - Parallel operations while some users get to know the new system and other users keep using the old system
    - More bugs found

Maintenance

Software maintenance

- Software is inherently flexible and can *change*.
- Many things can be happened after the products release to the customers
  - *Environment change*
    - New libraries or new system that our software has to operate with
    - Adaptive maintenance
  - *Requirements change*
    - through changing business circumstances, the software that supports the business must also evolve and change → Feature request
    - Perfective maintenance
  - *Bug report*
    - Problems with the software are found
    - Corrective maintenance

Wrap up

Wrap up

- At the end of project, the team should evaluate the project and decide what went right and what went wrong.
  - Figuring out how to make the things that went well occur more often in the future.
  - Determining how to prevent the things that went badly in the future.

SDLC in nature

- Different development models handle the basic tasks in different ways, such as making some less formal or repeating tasks many times.
- The basic tasks often occur at the same time, with some developers working on one task while other developers work on other tasks.
- Work sometimes flows backward with later tasks requiring changes to earlier tasks.
- The longer a mistake (bug) remains undetected, the harder it is to fix.

Quiz: Analyze how the programmer "A"

Analyze how the programmer "A" working on the following tasks and explain where those tasks are performed in this kind of SDLC activities

a. The customer sends me an e-mail describing the problem.
b. I reply telling what I think the customer wants (and sometimes asking for clarification).
c. The customer confirms my guesses or gives me more detail.
d. I crank out a quick example program.
e. I e-mail the example to the customer.
f. The customer examines the example and asks more questions if necessary.
g. I answer the new questions.

Next Class

- ***PEOPLE***

  involve in a software project

- ***DOCUMENTs***

  need for software development