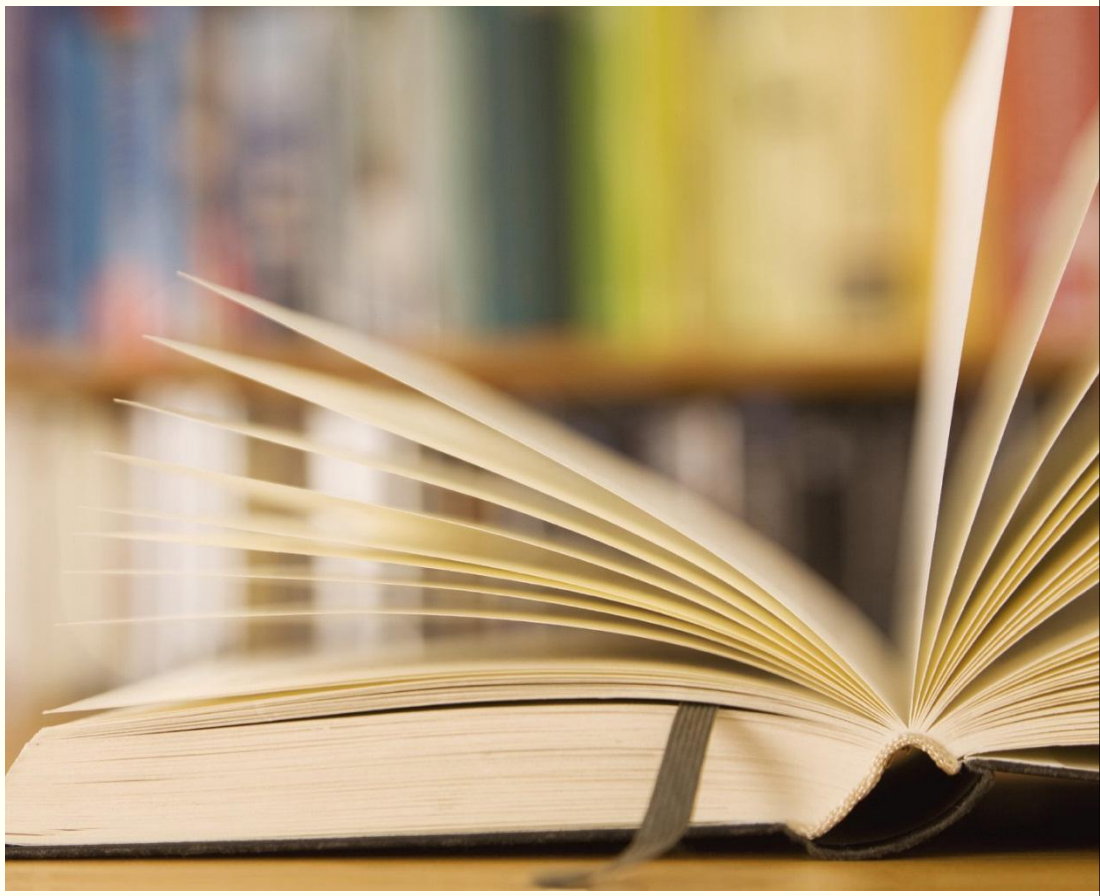
8. SOFTWARE ENGINEERING



Content

Objectives

After studying this chapter, the student should be able to:

- Understand the concept of the software lifecycle in software engineering.
- Describe two major types of development process, the waterfall and incremental models.
- Understand the analysis phase and describe two separate approaches in the analysis phase: procedure-oriented analysis and object-oriented analysis.
- Understand the design phase and describe two separate approaches in the design phase: procedure-oriented design and object-oriented design.
- Describe the implementation phase and recognize the quality issues in this phase.
- Describe the testing phase and distinguish between glass-box testing and black-box testing.
- Recognize the importance of documentation in software engineering and distinguish between user documentation, system documentation, and technical documentation.

1 - THE SOFTWARE LIFECYCLE

1. Introduction

- The **software development life cycle** (**SDLC**), also referred to as the **application development life-cycle**, is a process for planning, creating, testing, and deploying an information system.
- The systems development life cycle concept applies to a range of hardware and software configurations, as a system can be composed of hardware only, software only, or a combination of both.
- There are usually six stages in this cycle: **requirement analysis, design, development and testing, implementation, documentation, and evaluation.**

Figure 8.1 Evolution of programing

2. Development process models

- One of the basic notions of the software development process is SDLC models which stands for Software Development Life Cycle models. The models specify the various stages of the process and the order in which they are carried out. The most used, popular and important SDLC models are given below:

- **Waterfall model**
- V model
- **Incremental model**
- RAD model
- Agile model
- Iterative model
- Spiral model
- Prototype model

3. Waterfall model

- One very popular model for the software development process is known as the waterfall model (Figure 8.2). In this model, the development process flows in only one direction.
- This means that a phase cannot be started until the previous phase is completed.

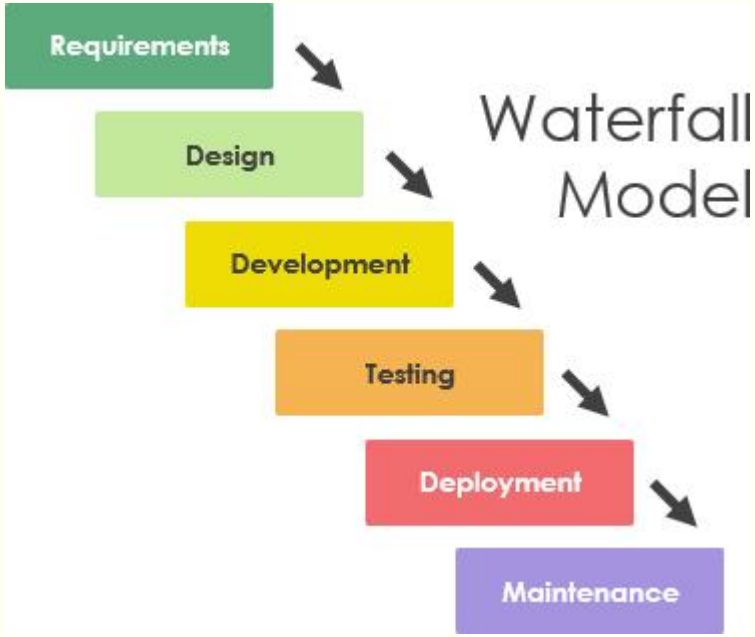


Figure 8.2 *The waterfall*

The incremental model

- In the **incremental model,** software is developed in a series of steps. The developers first complete a simplified version of the whole system. This version represents the entire system but does not include the details. Figure 10.3 shows the incremental model concept.

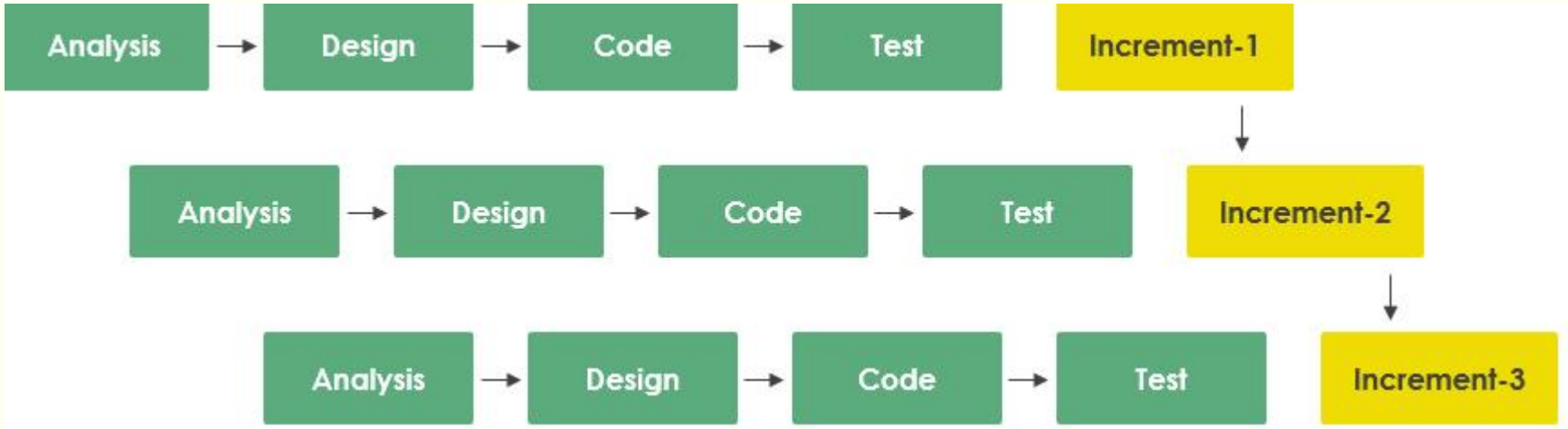


Figure 8.3 *The incremental model*

2- ANALYSIS PHASE

1. Introduction

- The development process starts with the **analysis phase**. This phase results in a specification document that shows ***what*** the software will do without specifying ***how*** it will be done.
- The analysis phase can use two separate approaches, depending on whether the **implementation phase** is done using a **procedural programming language** or an **object-oriented language**.

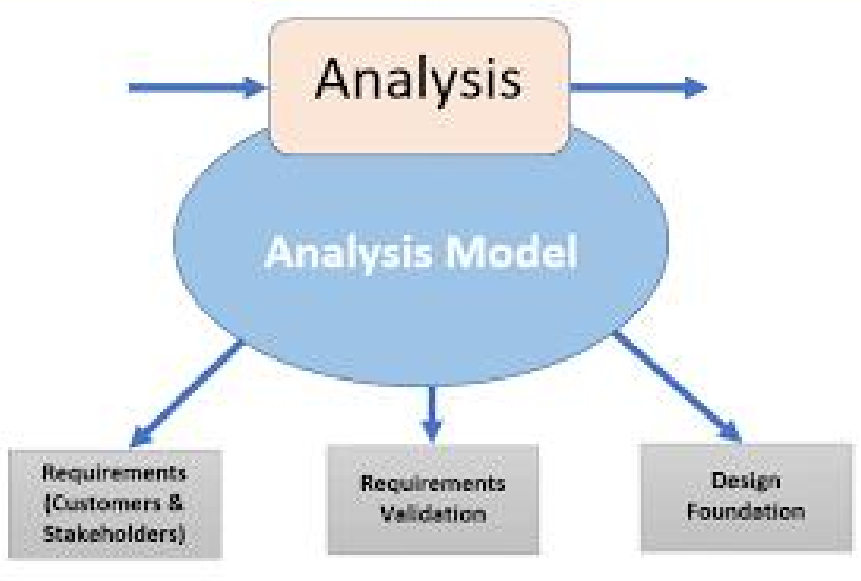


Figure 8.4 The analysis phase

2. Procedure-oriented analysis

- Procedure-oriented analysis—also called structured analysis or classical analysis—is the analysis process used if the system implementation phase will use a procedural language.
- The specification in this case may use several modeling tools, but we discuss only a few of them here.

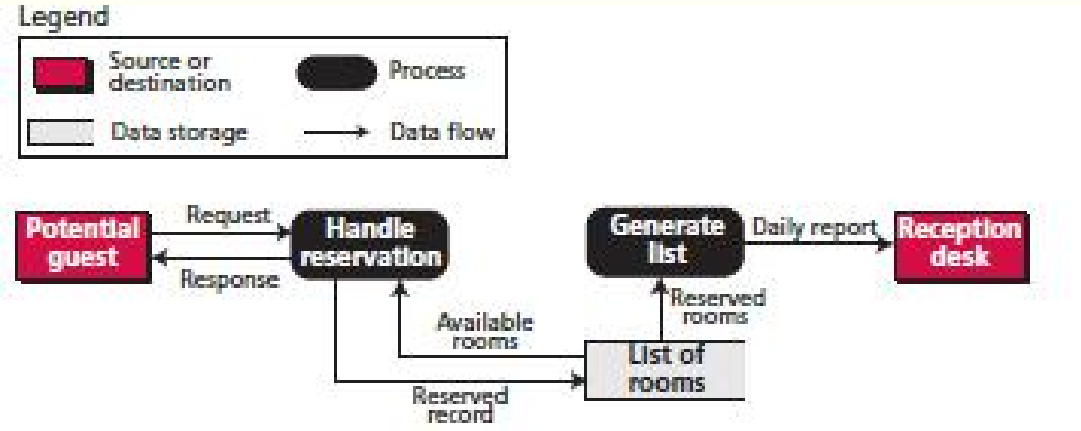


Figure 8.5 An example of a data flow diagram

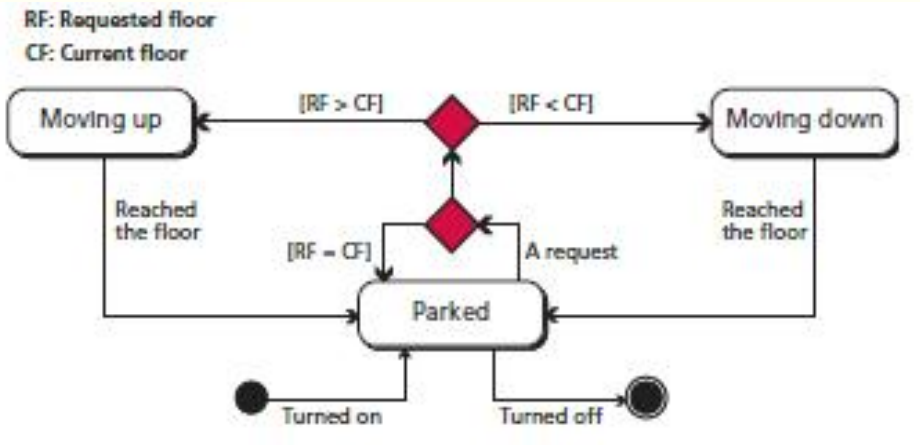


Figure 8.6 An example of a state diagram

3. Object-oriented analysis

- Object-oriented analysis is the analysis process used if the implementation uses an
- object-oriented language. The specification document in this case may use several tools, but we discuss only a few of them here.

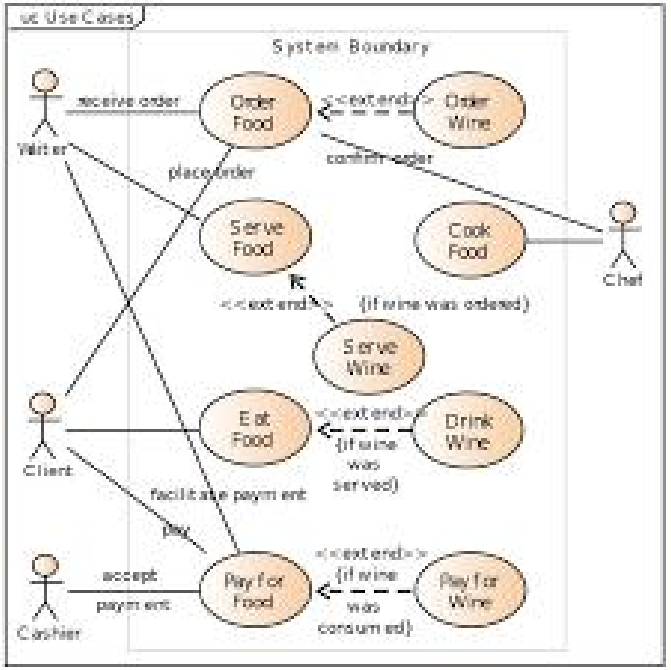


Figure 8.6 An example of a use case diagram

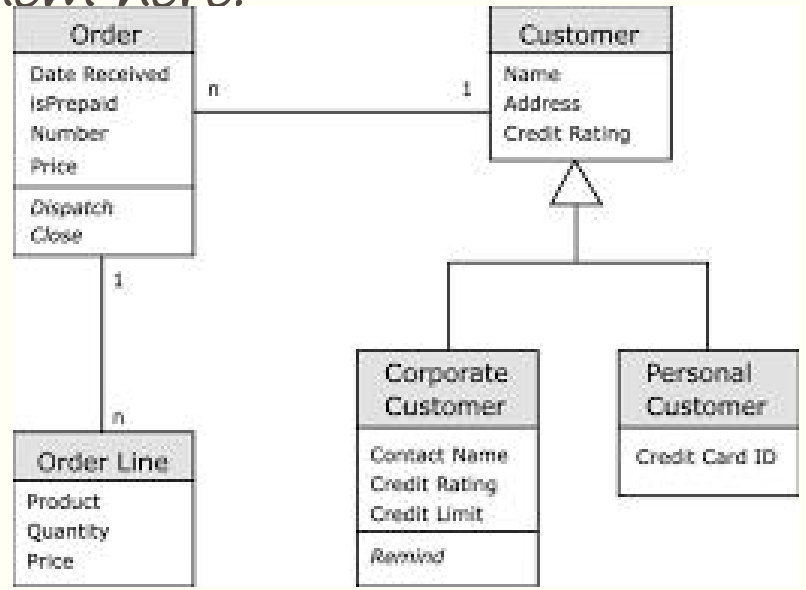


Figure 8.7 An example of a class diagram

3 - DESIGN PHASE

1. Introduction

- The design phase defines how the system will accomplish what was defined in the analysis phase. In the design phase, all components of the system are defined

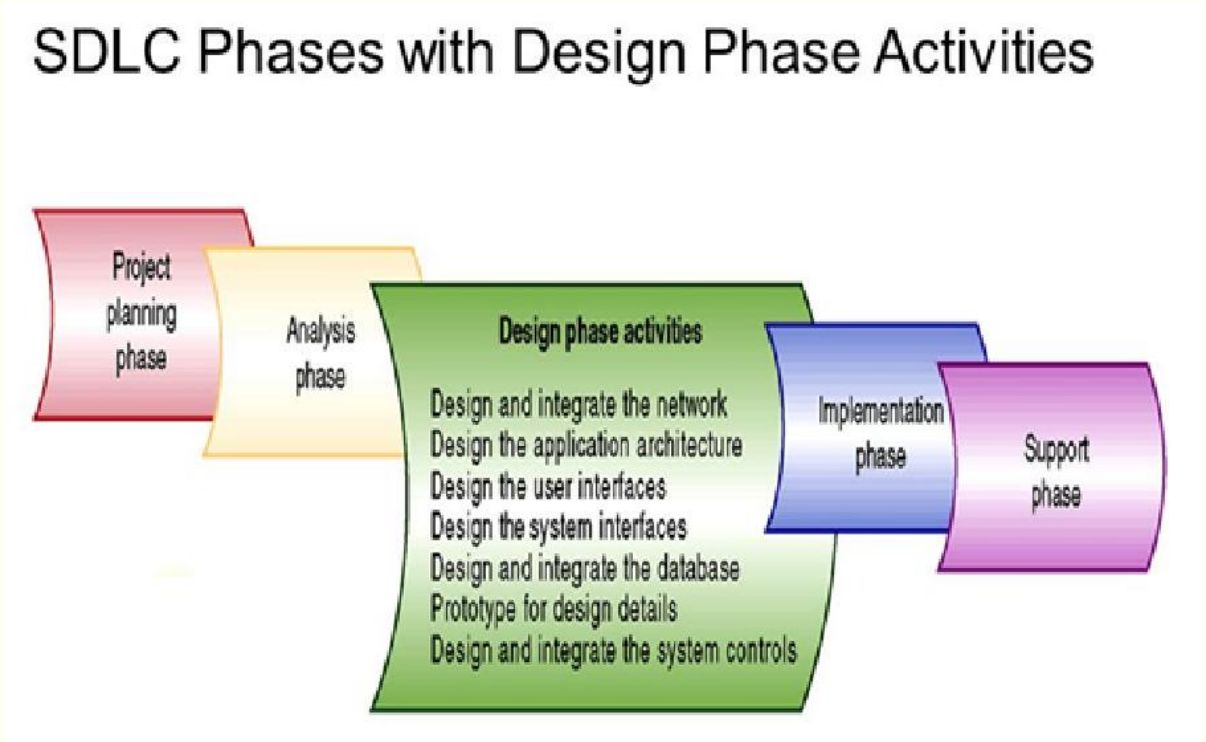


Figure 8.8 Design phase activities

2. Procedure-oriented design

- In procedure-oriented design we have both procedures and data to design. We discuss a category of design methods that concentrate on procedures. In procedure-oriented design, the whole system is divided into a set of procedure or modules.

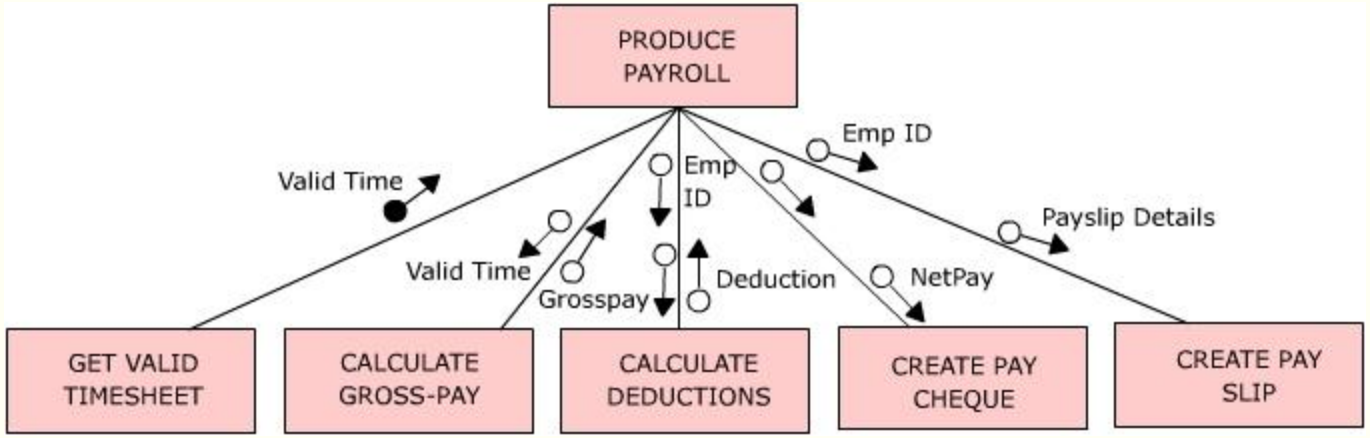


Figure 8.9 A example Structure Chart in Design phase

3. Object-oriented design

- In object-oriented design, the design phase continues by elaborating the details of classes.
- A class is made of a set of variables (attributes) and a set of methods (functions). The object-oriented design lists the details of these attributes and methods. Figure 10.9 shows an example of the details of our four classes used in the design of the old-style elevator.

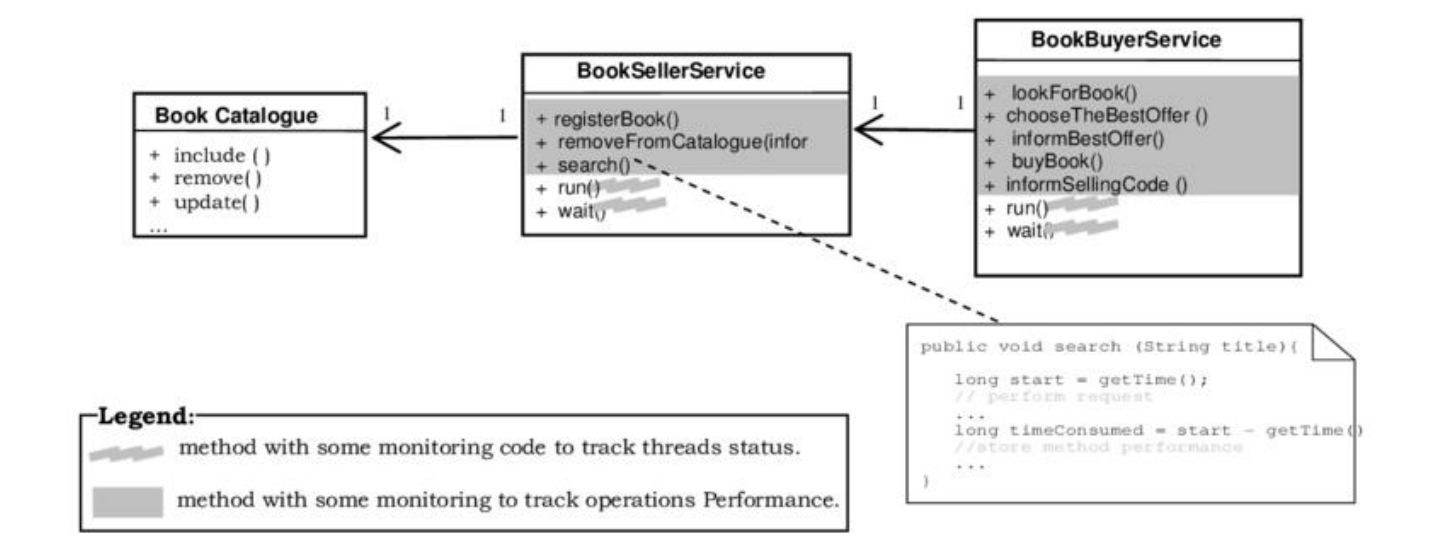


Figure 8.10 A Object-Oriented design of Book Trading system

4- IMPLEMENTATION PHASE

1. Introduction

- In the waterfall model, after the design phase is completed, the **implementation phase** can start.
- In this phase the programmers write the code for the modules in procedure-oriented design, or write the program units to implement classes in object-oriented design. There are several issues to mention in each case.

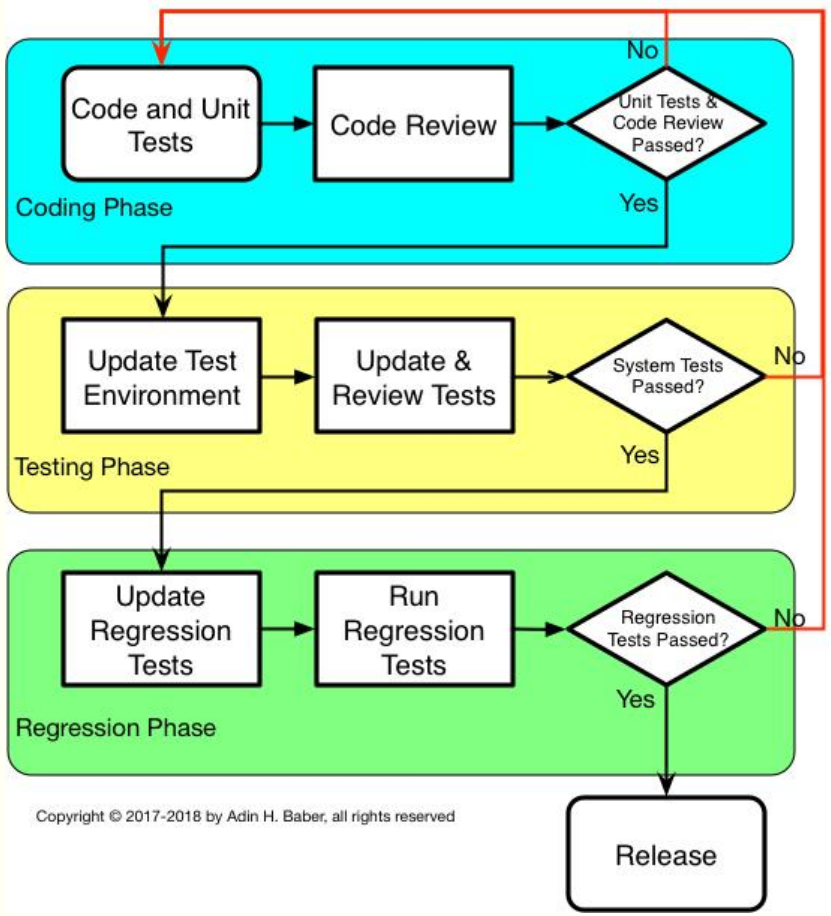


Figure 8.11 A example Implementation phase (coding & testing) in Water Fall model

2. Choice of language

- In a procedure-oriented development, the project team needs to choose a language or a set of languages from among the procedural languages.
- Although some languages like C++ are considered to be both a procedure—and an object-oriented language—normally an implementation uses a purely procedural language such as C. In object-oriented cases, both C++

```
// Get the size of the result string
DWORD dataSize = 0; // size of data, in bytes
const DWORD flags = RRF_RT_REG_SZ;
LONG retCode = ::RegGetValueW(
    hKey,
    nullptr,    // no subkey
    L"Connie",
    flags,
    nullptr,    // type not required
    nullptr,    // output buffer not needed now
    &dataSize
);
if (retCode != ERROR_SUCCESS)
{
    // Handle error...
}
```

Java8Example.java

```
1  package com.vogella.eclipse.ide.java8;
2
3  public class Java8Example {
4
5      public static void main(String[] args) {
6          Runnable runnable = new Runnable() {
7              @Override
8              public void run() {
9                  System.out.println("Hello Lambdas");
10             }
11         };
12
13         new Thread(runnable);
14
15     }
16
17 }
18
```

Figure 8.12 A example language programing (C++ & Java) in implementation phase

3. Software quality

- The quality of software created at the implementation phase is a very important issue. A software system of high quality is one that satisfies the user's requirements, meets the operating standards of the organization, and runs efficiently on the hardware for which it was developed. However, if we want to achieve a software system of high quality, we must be able to define some attributes of quality.

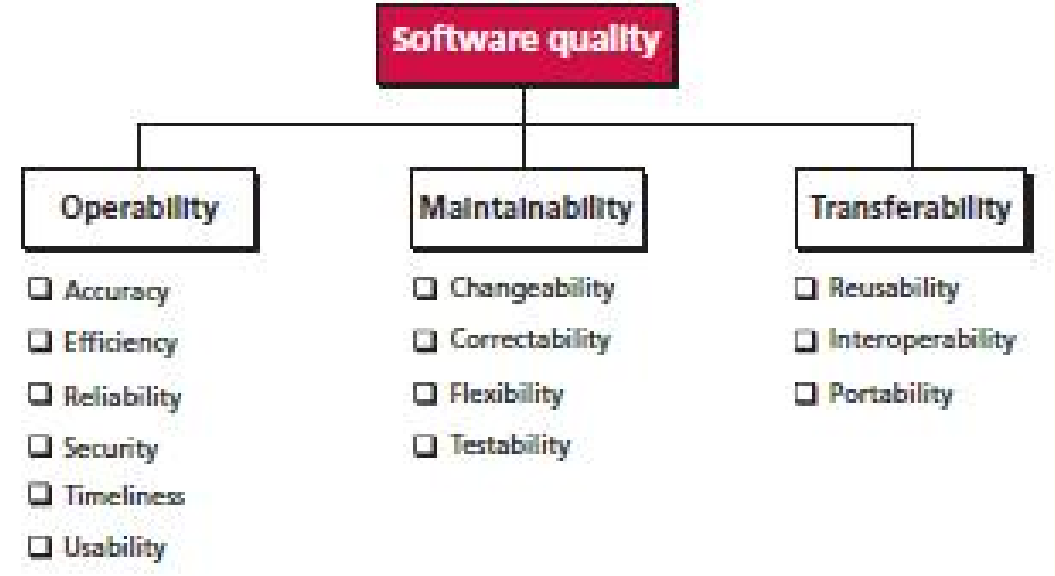


Figure 8.13 Quality factors

4. Operability

- Operability refers to the basic operation of a system. Several measures can be mentioned for operability, as shown in Figure 8.14: **accuracy**, **efficiency**, **reliability**, **security**, **timeliness**, and **usability**.

Figure 8.14 Operability factors

5. Maintainability

- Maintainability refers to the ease with which a system can be kept up to date and running correctly. Many systems require regular changes, not because they were poorly implemented, but because of changes in external factors.
- For example, the payroll system for a company might have to be changed often to meet changes in government laws and regulations

Figure 8.15 – Maintainability factors

6. Transferability

- Transferability refers to the ability to move data and/or a system from one platform to another and to reuse code. In many situations this is not an important factor. On the other hand, if we are writing generalized software, it can be critical.

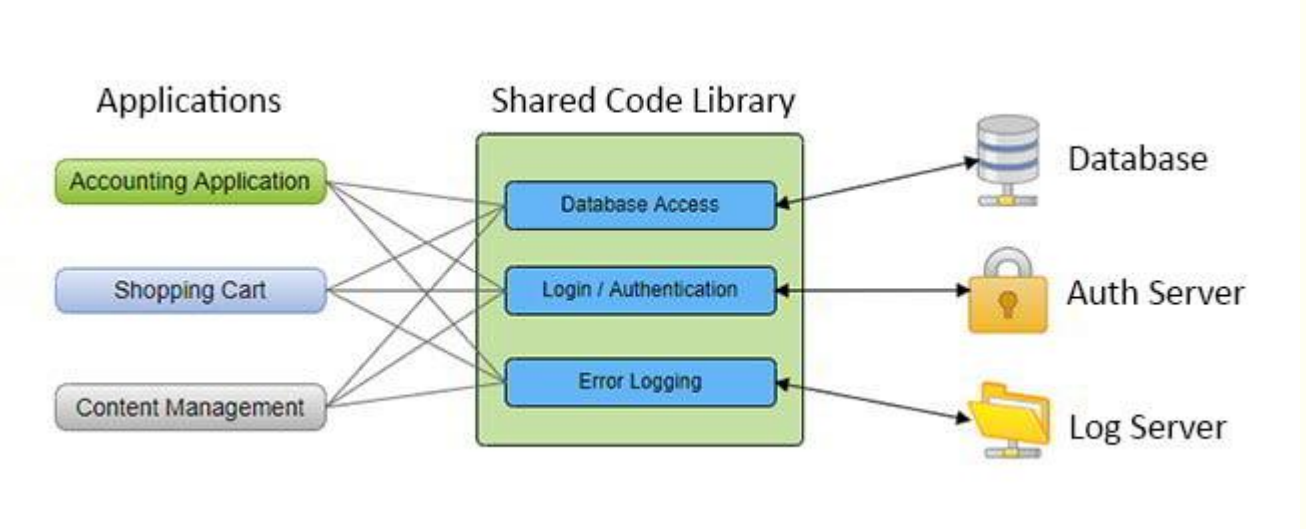


Figure 8.16 – A reuse code example

5 - TESTING PHASE

1. Introduction

- The goal of the testing phase is to find errors, which means that a good testing strategy is the one that finds most errors. There are two types of testing: **white-box** and **black-box** (Figure 10.11).

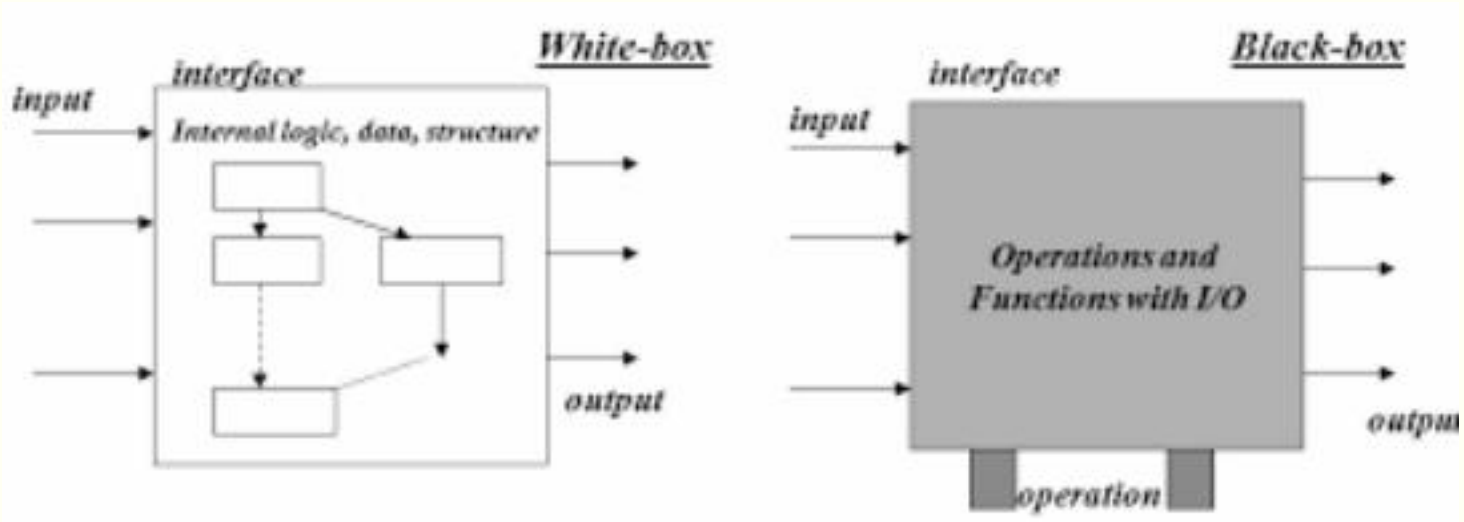


Figure 8.17 Software testing

2. Glass-box testing (or white-box testing)

- **Glass-box testing** (or **white-box testing**) is based on knowing the internal structure of the software. The testing goal is to check to determine whether all components of the software do what they are designed to do.
- Glass-box testing assumes that the tester knows everything about the software. In this case, the software is like a glass box in which everything inside the box is visible.
- Glass-box testing is done by the software engineer or a dedicated team.

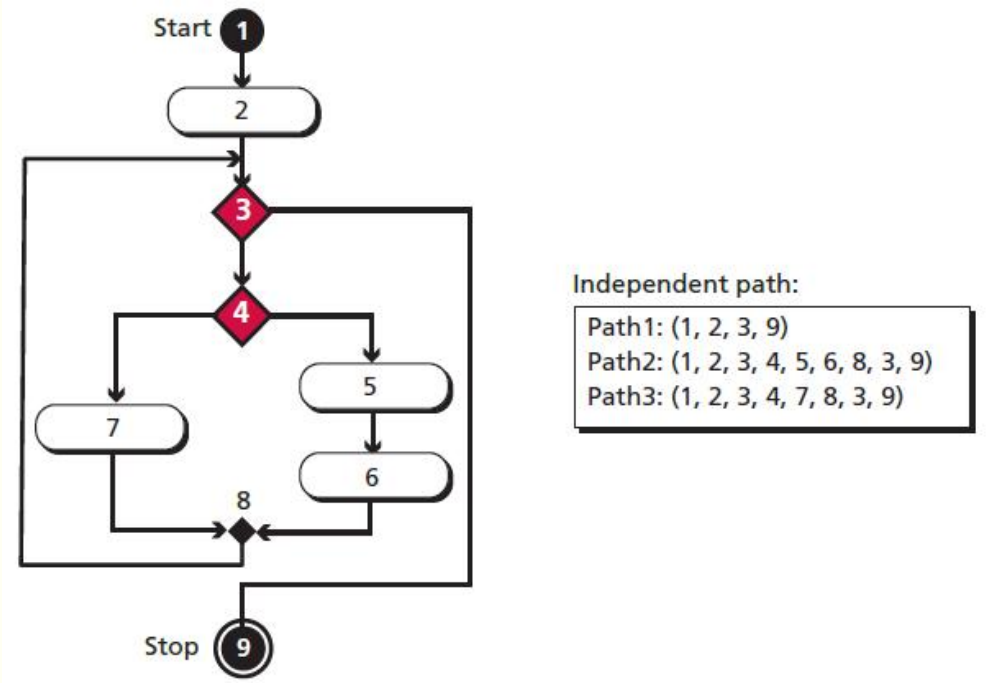


Figure 8.18 An example of basis path testing

3. Black testing

- **Black-box testing** gets its name from the concept of testing software without knowing what is inside it and without knowing how it works. In other words, the software is like a black box into which the tester cannot see.
- Black-box testing tests the functionality of the software in terms of what the software is supposed to accomplish, such as its inputs and outputs. Several methods are used in black-box testing, discussed below.

Test Case #	Inputs (Causes)		Expected Output (Effects)
	Sex	Age	Premium
1	Male	<25	$3000
2	Male	>=25 and < 65	$1000
3	Male	>= 65	$1500
4	Female	>= 65	$1500
5	Female	<25	$500
6	Female	>=25 and < 65	$500

Figure 8.19 A example Black-box Testing

4. Exhaustive testing

- The best black-box test method is to test the software for all possible values in the input domain. However, in complex software the input domain is so huge that it is often impractical to do so.

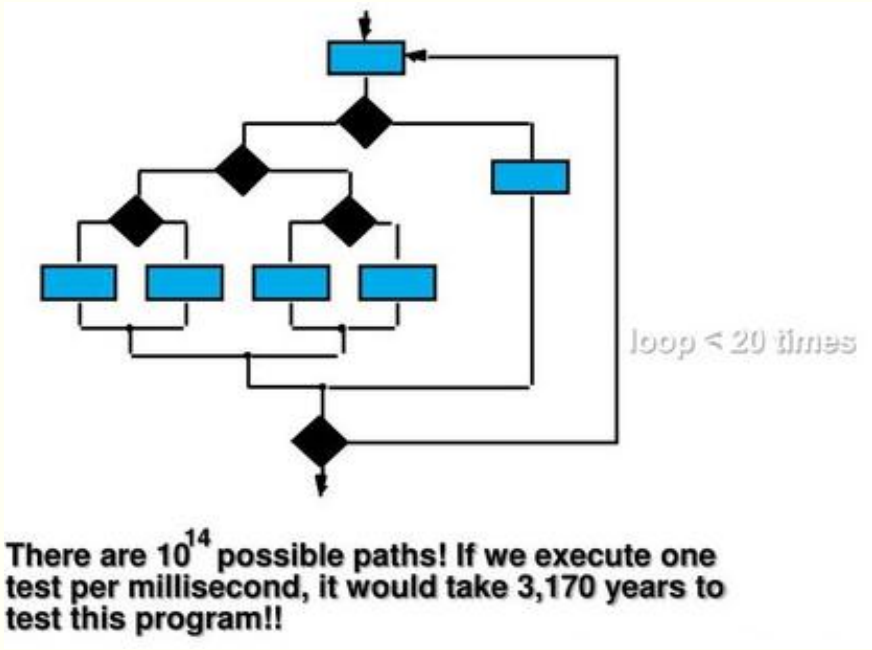


Figure 8.20 A example Black-box Testing

5. Random testing

- In random testing, a subset of values in the input domain is selected for testing. It is very important that the subset be chosen in such a way that the values are distributed over the domain input. The use of random number generators can be very helpful in this case.

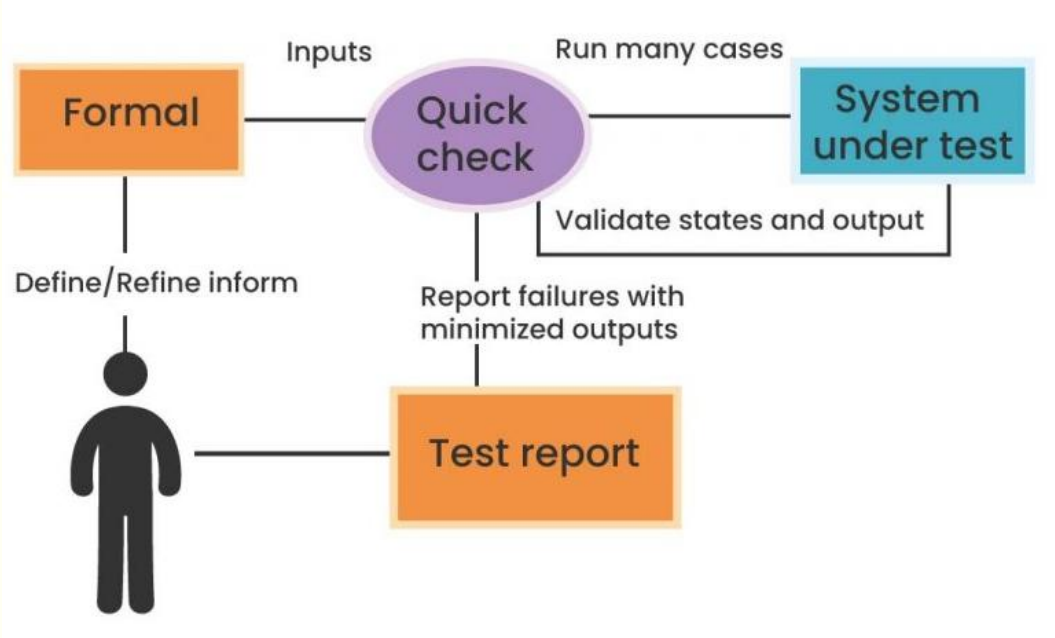
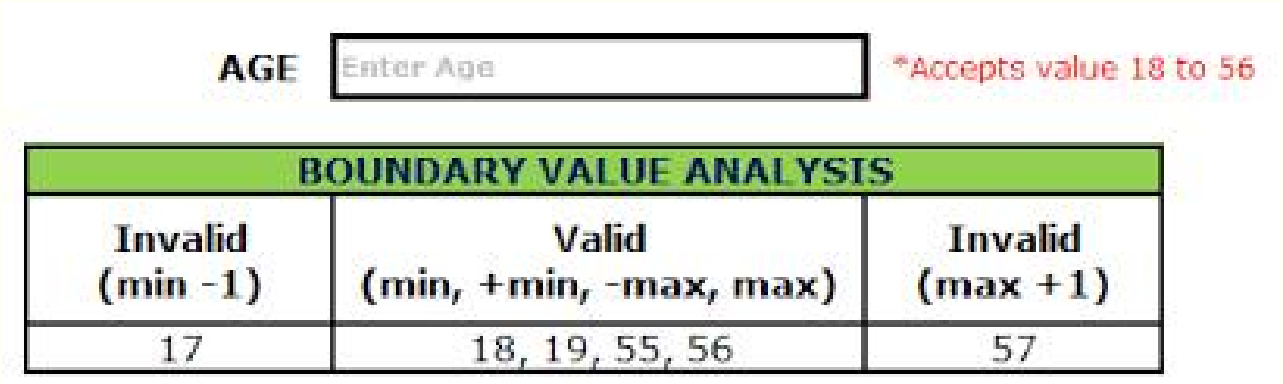


Figure 10.15 A example Black-box Testing

6. Boundary-value testing

- Errors often happen when boundary values are encountered. For example, if a module defines that one of its inputs must be greater than or equal to 100, it is very important that module be tested for the boundary value 100. If the module fails at this boundary value, it is possible that some condition in the module's code such as x . 100 is written as x . 100.



BOUNDARY VALUE ANALYSIS		
Invalid (min -1)	**Valid (min, +min, -max, max)**	**Invalid (max +1)**
17	18, 19, 55, 56	57

Figure 10.16 A example Black-box Testing