جامعة الأميرة نورة بنت عبد الرحمن

Princess Nora Bint Abdul Rahman University

SOFTWARE ENGINEERING

Networks and Communication Department

Lecture 6

By: Latifa ALrashed

Outline

- Define the concept of the software life cycle in software engineering.
- Identify the system development life cycle (SDLC).
- Describe two major types of development process, the waterfall and incremental models.
- Discuss the analysis phase and describe two separate approaches in the analysis phase: procedure-oriented analysis and object-oriented analysis.
- Discuss 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 blackbox testing.
- Show the importance of documentation in software engineering and distinguish between user documentation, system documentation and technical documentation.

The software lifecycle

- A fundamental concept in software engineering is the *software lifecycle*.
- Software, like many other products, goes through a cycle of repeating phases.

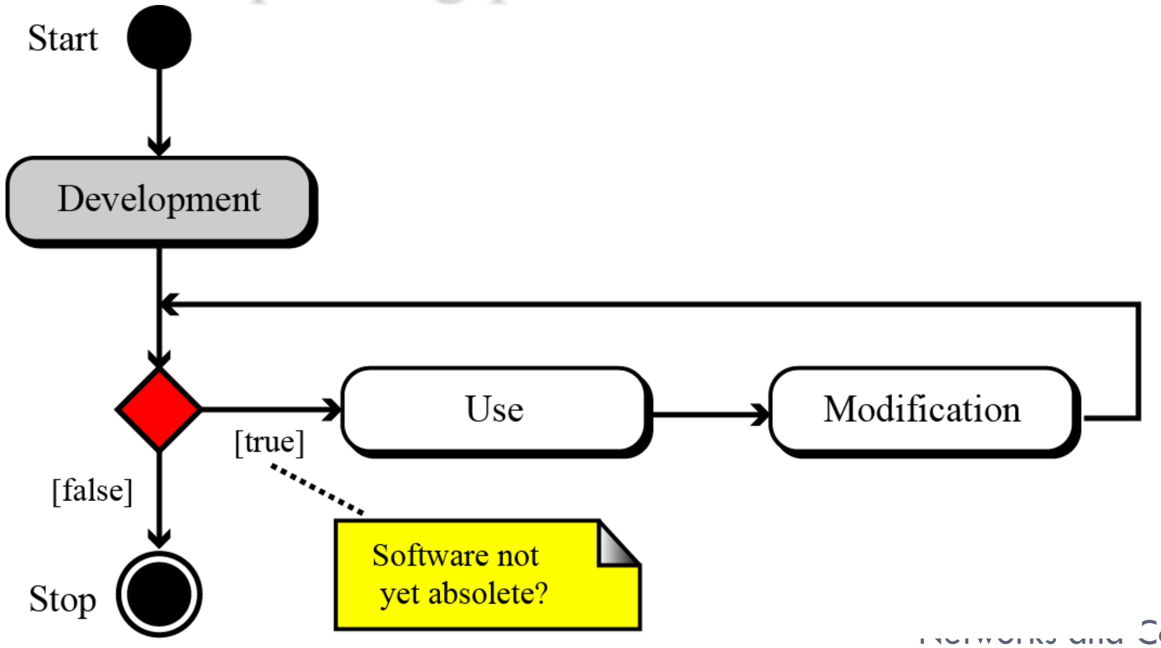


Figure 10.1 The software lifecycle

The software lifecycle (Cont.)

- Software is first developed by a group of developers.
- Usually it is in use for a while before modifications are necessary.
- The two steps, *use* and *modify*, continue until the software becomes obsolete.
- By "obsolete", we mean that the software loses its validity because of inefficiency, obsolescence of the language, major changes in user requirements, or other factors.

Developing Information Systems

- **System Development Methodology** is a standard process followed in an organization to conduct all the steps necessary to analyze, design, implement, and maintain information systems.

Systems Development Life Cycle (SDLC)

- Traditional methodology used to develop, maintain, and replace information systems.
- Phases in SDLC:
  - Planning
  - Analysis
  - Design
  - Implementation
  - Maintenance

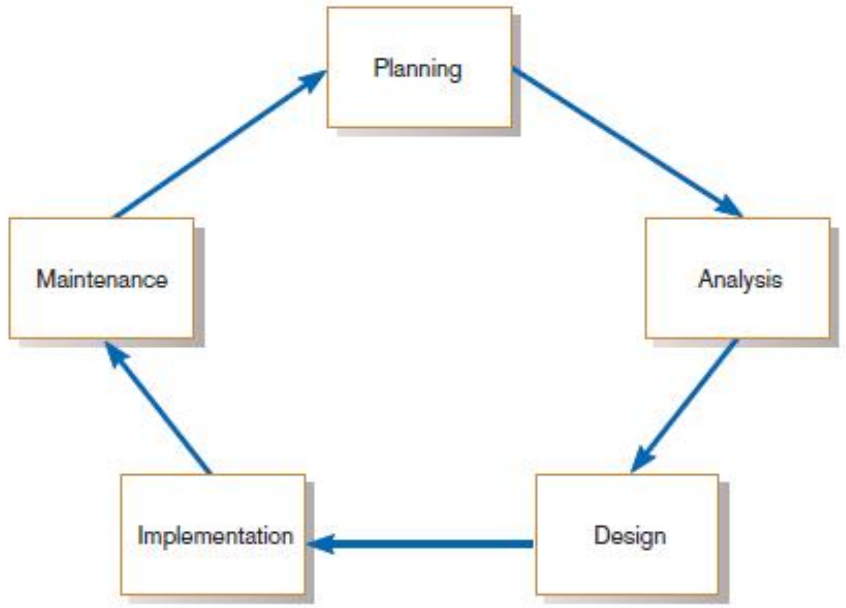


FIGURE
The systems development life cycle

Systems Development Life Cycle (SDLC) (Cont.)

- **Planning** – an organization's total information system needs are identified, analyzed, prioritized, and arranged
- **Analysis** – system requirements are studied and structured
- **Design** – a description of the recommended solution is converted into logical and then physical system specifications

Systems Development Life Cycle (SDLC) (Cont.)

- **Logical design** – all functional features of the system chosen for development in analysis are described independently of any computer platform
- **Physical design** – the logical specifications of the system from logical design are transformed into the technology-specific details from which all programming and system construction can be accomplished
- **Implementation** – the information system is coded, tested, installed and supported in the organization
- **Maintenance** – an information system is systematically repaired and improved

Development process models

- There are several models for the development process. We discuss the two most common here: the waterfall model and the incremental model.

- The waterfall model is a very popular model for the software development process.

The waterfall model

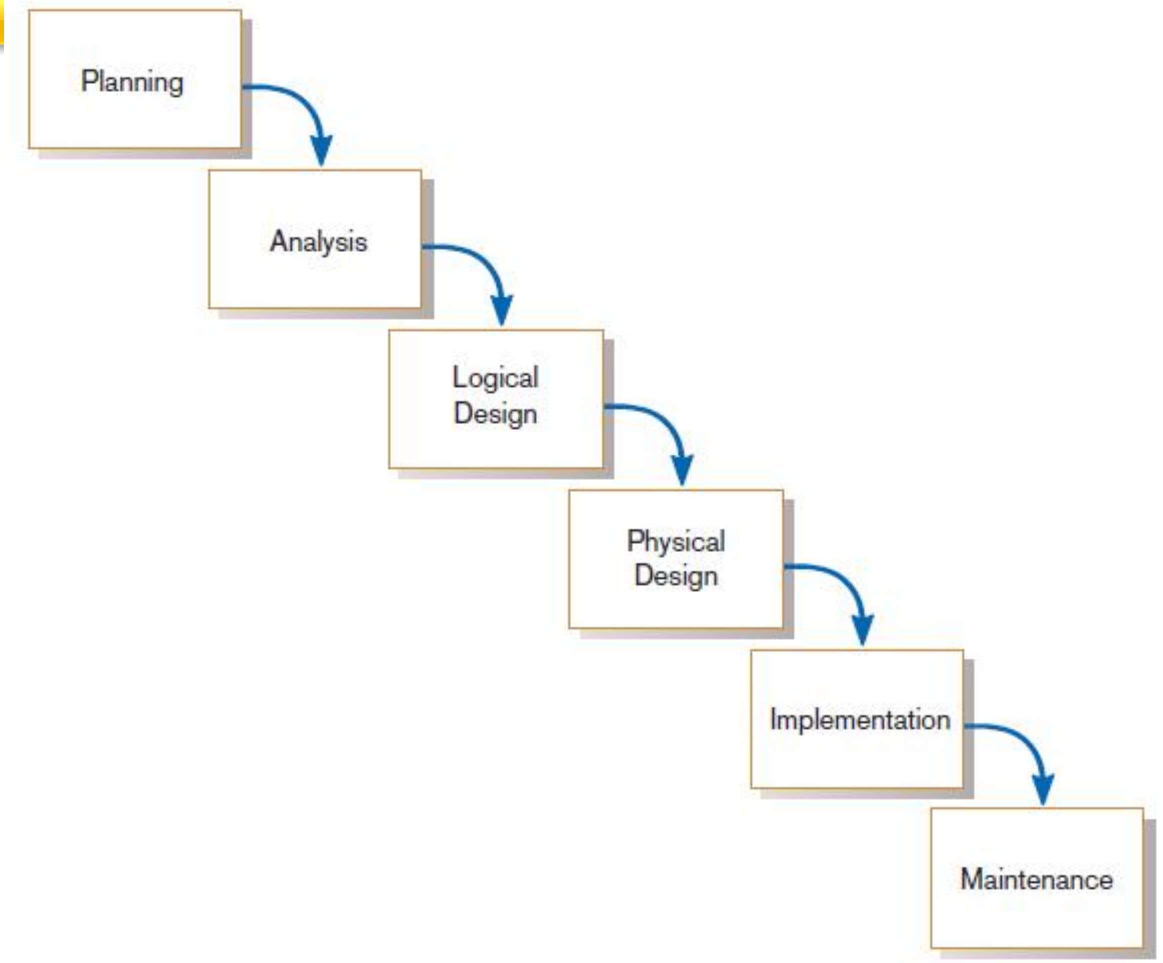


Figure 10.2 The waterfall model

The waterfall model (Cont.)

- 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.
- For example, the entire design phase should be finished before the implementation phase can be started.
- There are advantages and disadvantages to the waterfall model.

The waterfall model (Cont.)

- Pros:
  - Each phase is completed before the next phase starts;
  - For example, the group that works on the design phase knows exactly what to do because they have the complete results of the analysis phase.
  - The testing phase can test the whole system because the entire system under development is ready.

The waterfall model (Cont.)

- Cons:
  - System requirements “locked in” after being determined (can't change)
  - Limited user involvement (only in requirements phase)
  - Too much focus on milestone deadlines of SDLC phases to the detriment of sound development practices.
  - The difficulty in locating a problem: if there is a problem in part of the process, the entire process must be checked.

The incremental model

- In the incremental model, software is developed in a series of steps.

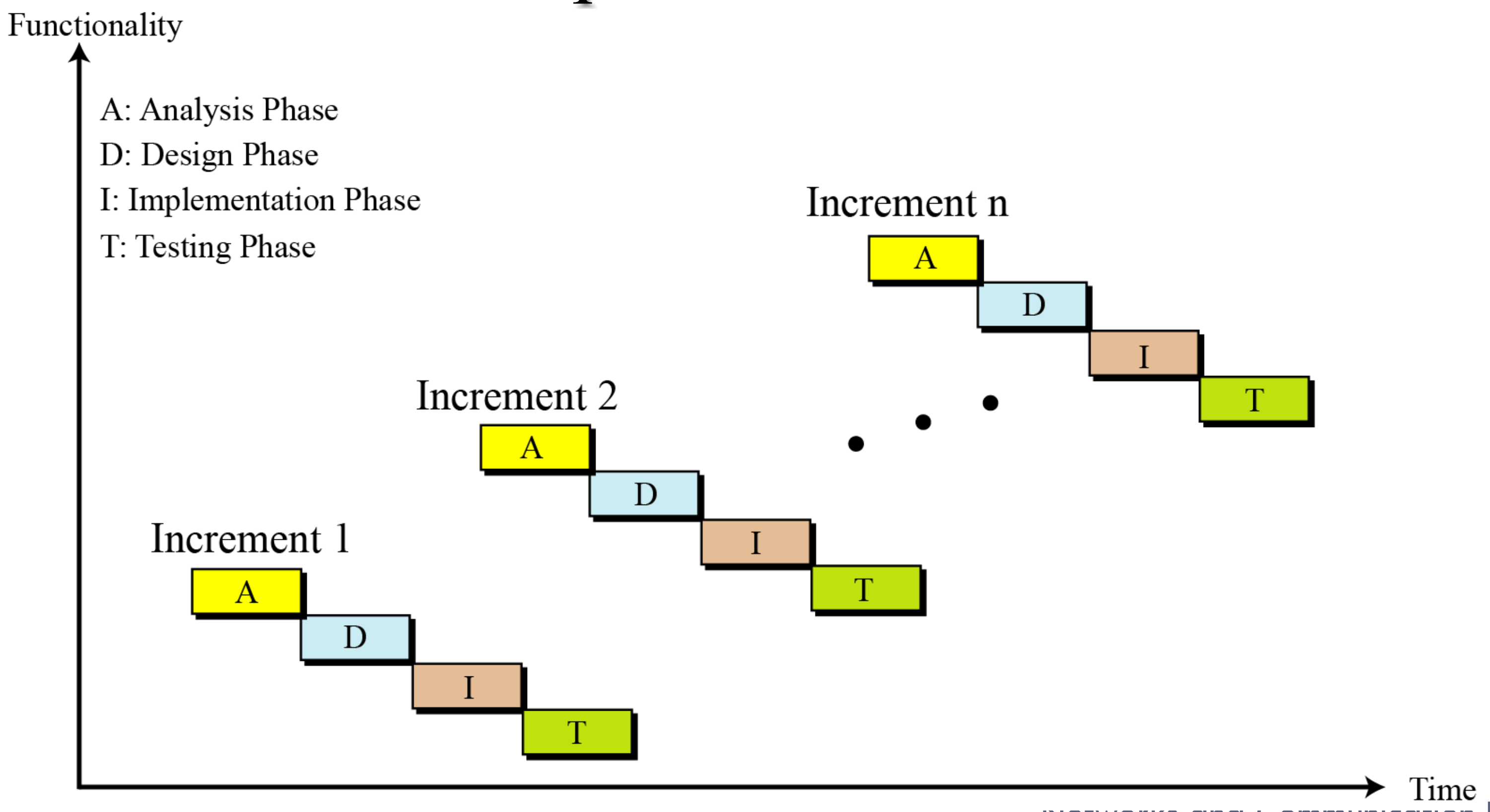


Figure 10.3 The incremental model

The incremental model (Cont.)

- The developers first complete a simplified version of the whole system.
- This version represents the entire system but does not include the details.
- In the second version, more details are added, while some are left unfinished, and the system is tested again.
- If there is a problem, the developers know that the problem is with the new functionality,
- they do not add more functionality until the existing system works properly.
- This process continues until all required functionality has been added.

ANALYSIS PHASE

- 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. We briefly discuss both in this section.

Procedure-oriented analysis

- Procedure-oriented 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.

Data flow diagrams

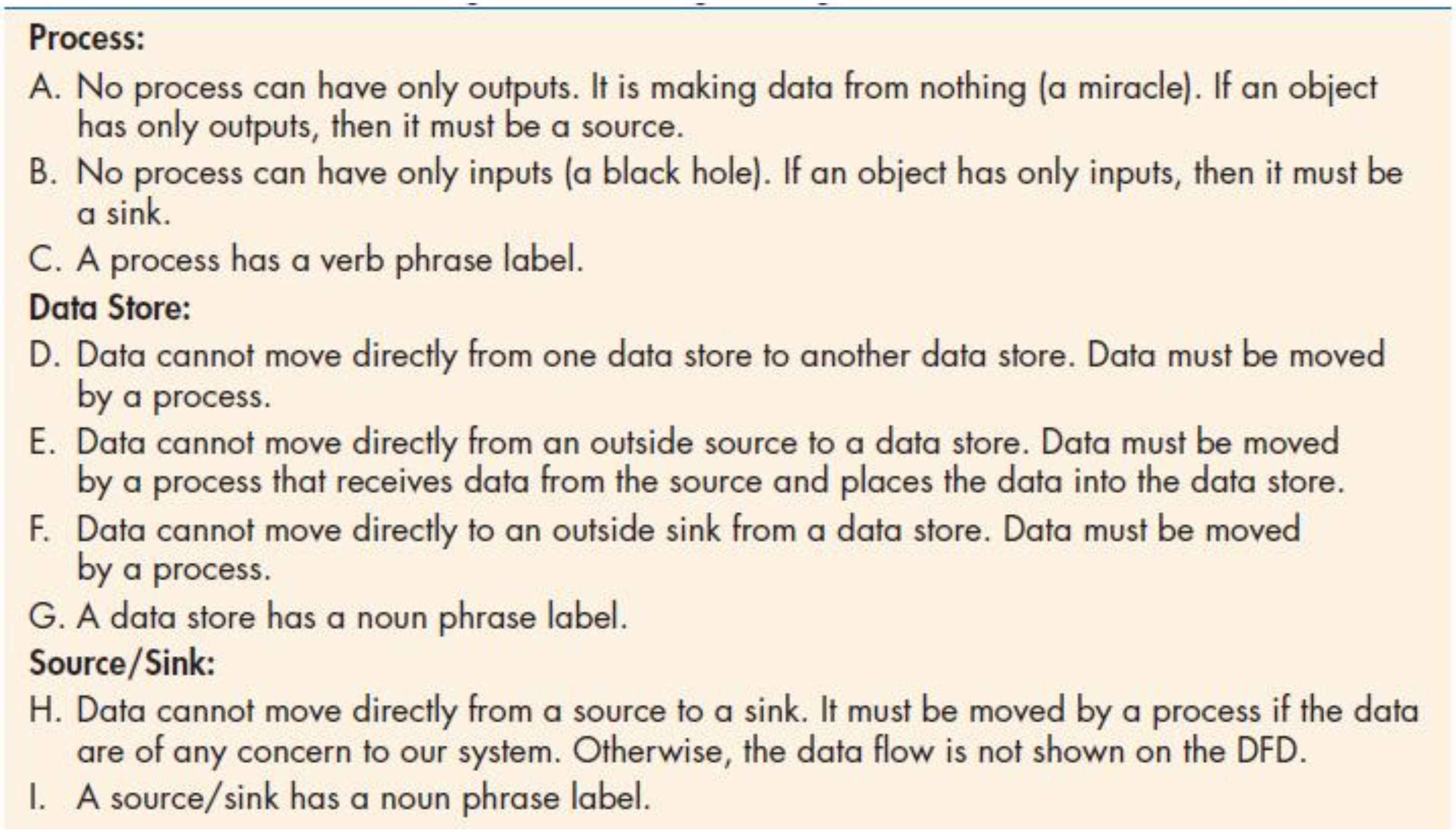
- Data flow diagrams show the movement of data in the system.

- They use four symbols: a square box shows the source or destination of data, a rectangle with rounded corners shows the process (the action to be performed on the data), an open-ended rectangle shows where data is stored, and arrows shows the flow of data.

Figure 10.4 An example of a data flow diagram

Data Flow Diagramming Rules

TABLE 7-2 Rules Governing Data Flow Diagramming



Process:

A. No process can have only outputs. It is making data from nothing (a miracle). If an object has only outputs, then it must be a source.

B. No process can have only inputs (a black hole). If an object has only inputs, then it must be a sink.

C. A process has a verb phrase label.

Data Store:

D. Data cannot move directly from one data store to another data store. Data must be moved by a process.

E. Data cannot move directly from an outside source to a data store. Data must be moved by a process that receives data from the source and places the data into the data store.

F. Data cannot move directly to an outside sink from a data store. Data must be moved by a process.

G. A data store has a noun phrase label.

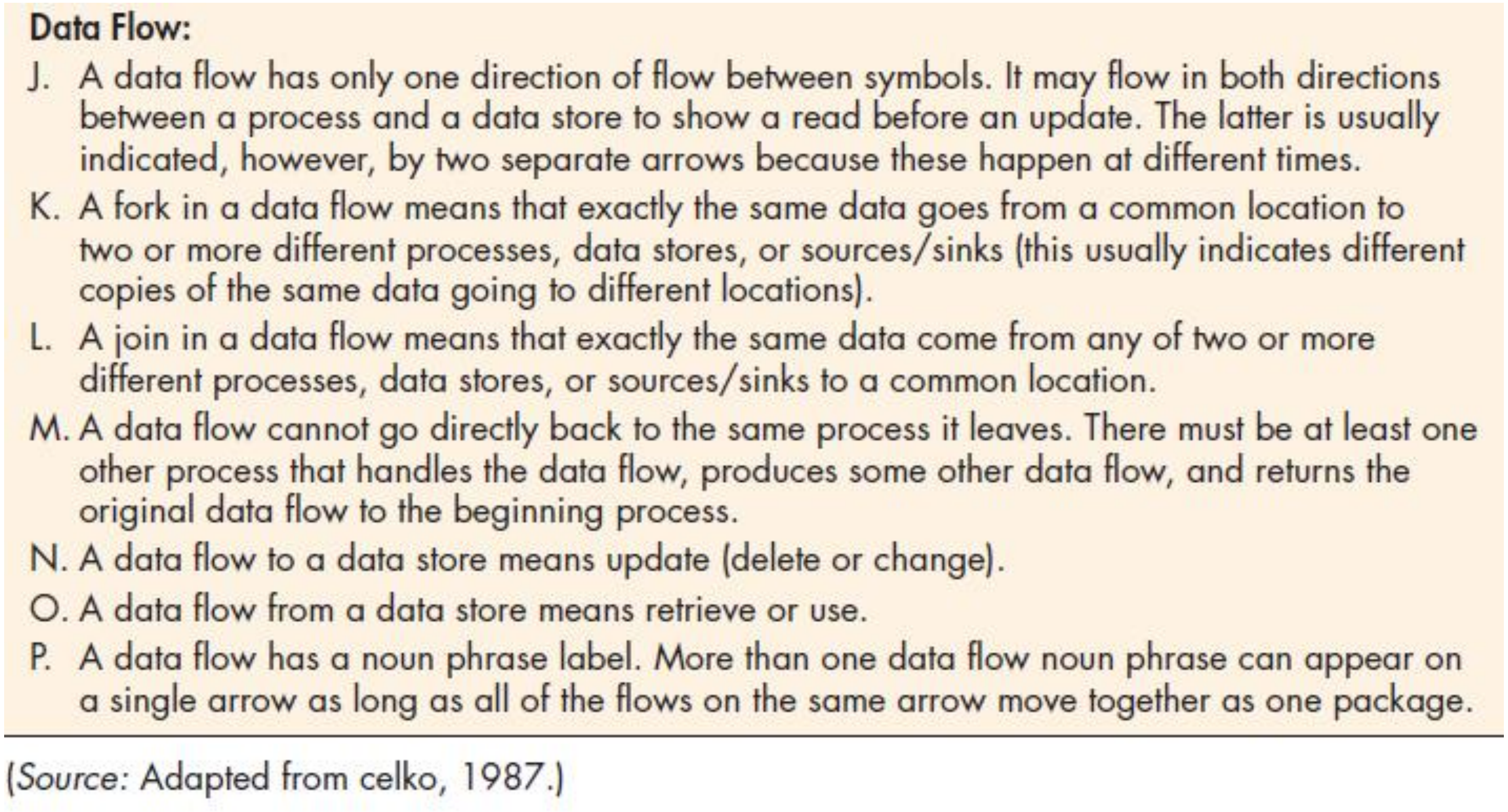
Source/Sink:

H. Data cannot move directly from a source to a sink. It must be moved by a process if the data are of any concern to our system. Otherwise, the data flow is not shown on the DFD.

I. A source/sink has a noun phrase label.

Data Flow Diagramming Rules (Cont.)

TABLE 7-2 Rules Governing Data Flow Diagramming (cont.)



Data Flow:

J. A data flow has only one direction of flow between symbols. It may flow in both directions between a process and a data store to show a read before an update. The latter is usually indicated, however, by two separate arrows because these happen at different times.

K. A fork in a data flow means that exactly the same data goes from a common location to two or more different processes, data stores, or sources/sinks (this usually indicates different copies of the same data going to different locations).

L. A join in a data flow means that exactly the same data come from any of two or more different processes, data stores, or sources/sinks to a common location.

M. A data flow cannot go directly back to the same process it leaves. There must be at least one other process that handles the data flow, produces some other data flow, and returns the original data flow to the beginning process.

N. A data flow to a data store means update (delete or change).

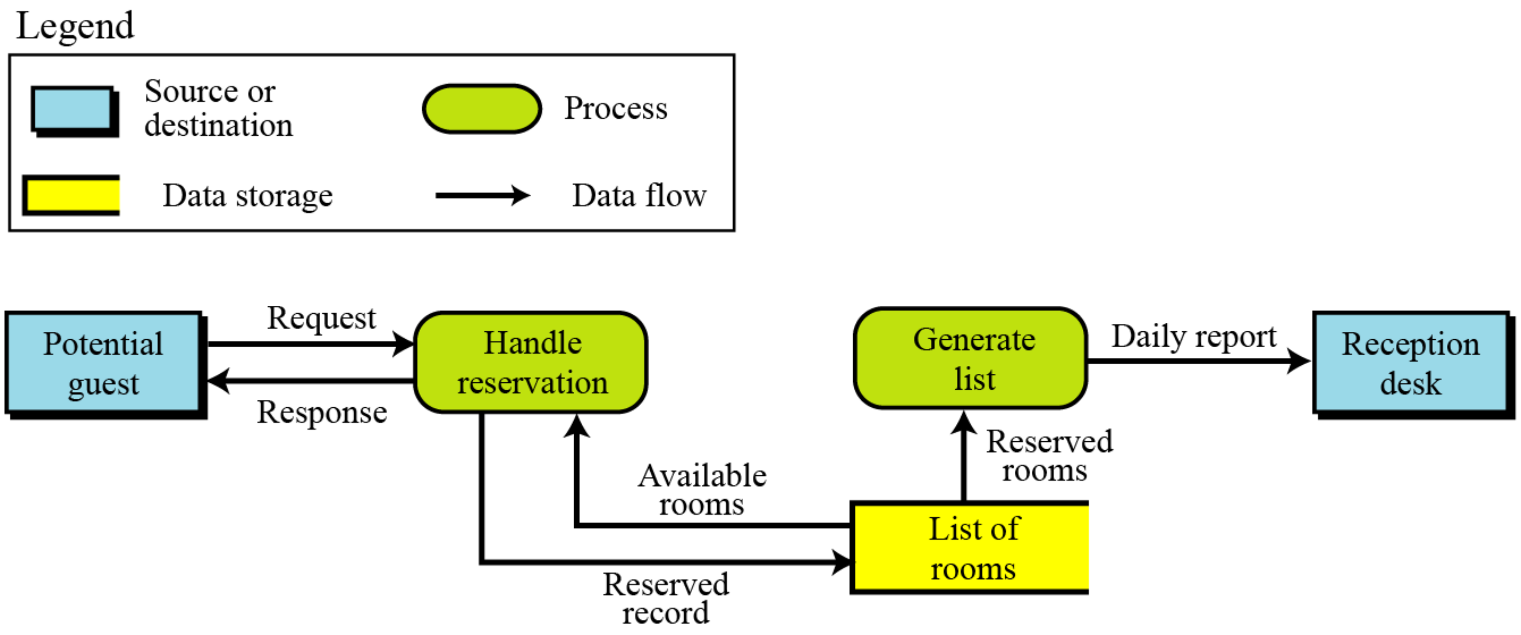
O. A data flow from a data store means retrieve or use.

P. A data flow has a noun phrase label. More than one data flow noun phrase can appear on a single arrow as long as all of the flows on the same arrow move together as one package.

(*Source:* Adapted from celko, 1987.)

Data flow diagram example

- Figure 10.4 shows a simplified version of a booking system in a small hotel that accepts reservation from potential guests through Internet and confirms or denies the reservation based on available vacancies.



Procedure-oriented analysis (Cont.)

- Entity-relationship diagrams
  - Another modeling tool used during the analysis phase is the entity-relationship diagram.
  - The database designer creates an ER diagram to show the entities for which information needs to be stored and the relationship between those entities.
- State diagrams
  - State diagrams provide another useful tool that is normally used when the state of the entities in the system will change in response to events.
  - As an example of a state diagram, we show the operation of a one-passenger elevator. When a floor button is pushed, the elevator moves in the requested direction. It does not respond to any other request until it reaches its destination.

State diagrams (Cont.)

- State is represented by rounded rectangle in the state diagram

RF: Requested floor

CF: Current floor

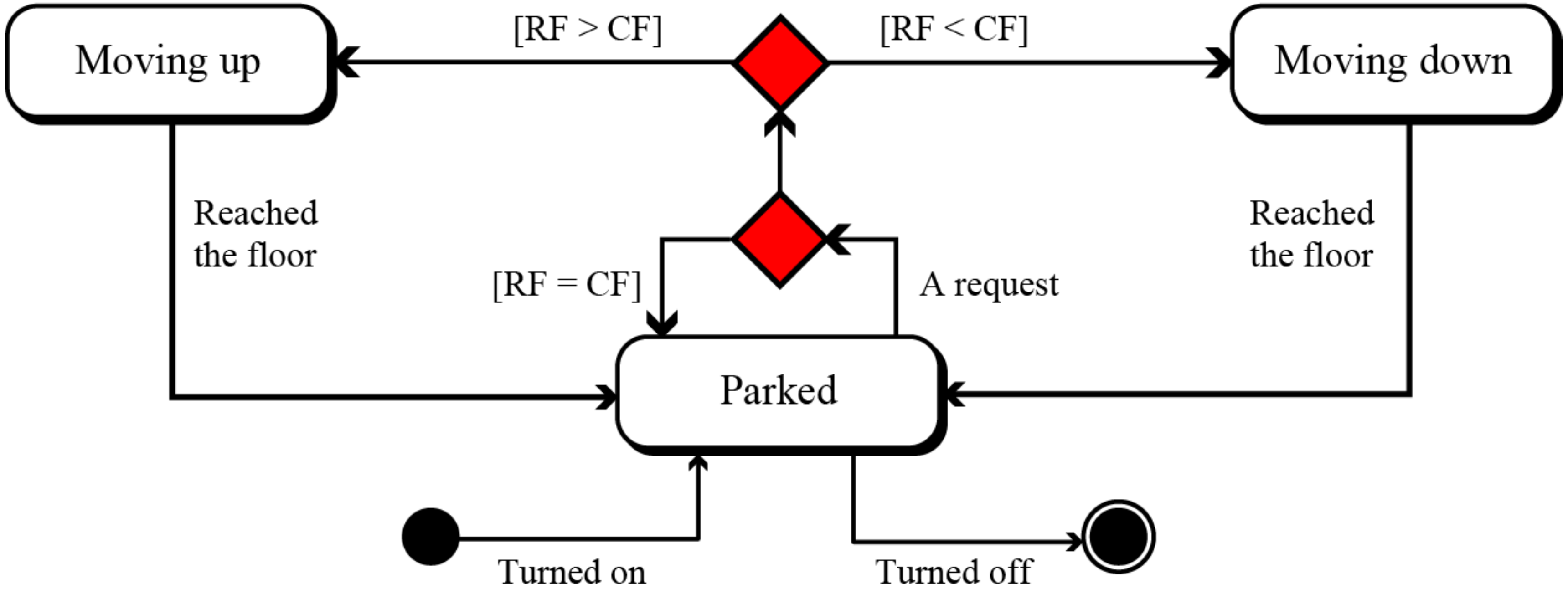


Figure 10.5 Shows a state diagram for this old-style elevator

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.
- Use case diagrams
  - A use-case diagram gives the user's view of a system: it shows how users communicate with the system.
  - A use-case diagram uses four components: system, use cases, actors and relationships. A system, shown by a rectangle, performs a function.

Use case diagrams

- The action (Function) in the system are shown by use cases, which are denoted by rounded rectangles.
- An actor is someone or something that uses the system, which represented by stick figures

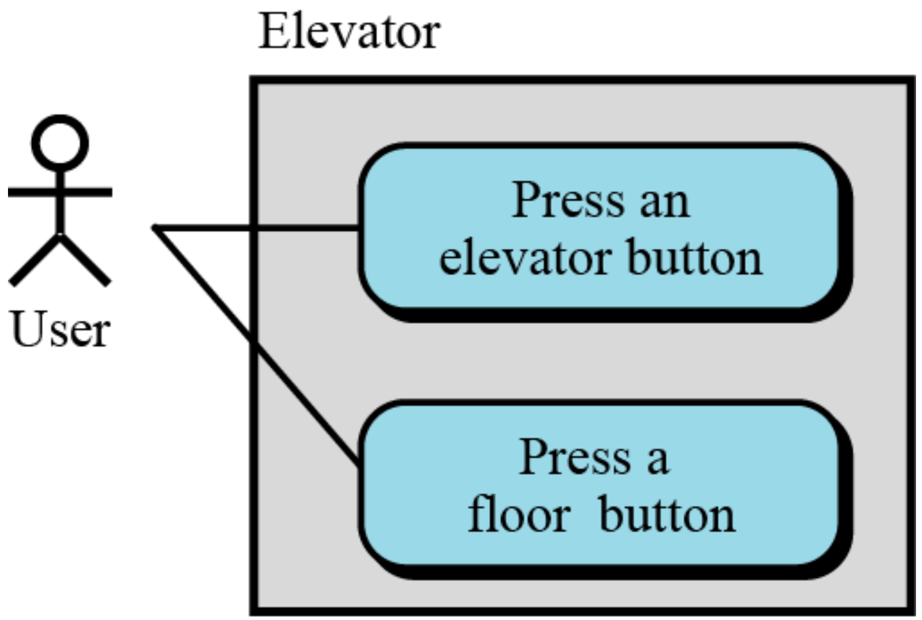


Figure 10.6 shows the use case diagram for the old-style elevator

Class diagrams

- The next step in analysis is to create a class diagram for the system. For example, we can create a class diagram for our old-style elevator. To do so, we need to think about the entities involved in the system.

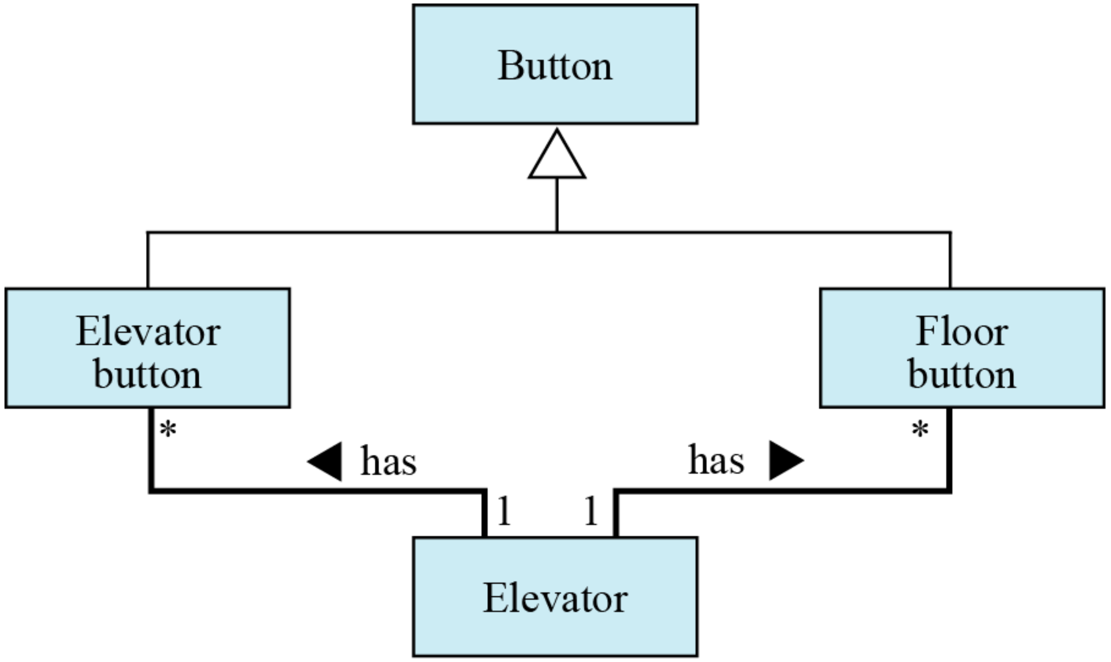


Figure 10.7 An example of a class diagram

State chart

- After the class diagram is finalized, a state chart can be prepared for each class in the class diagram. A state chart in object-oriented analysis plays the same role as the state diagram in procedure-oriented analysis. This means that for the class diagram of Figure 10.7, we need to have a four-state chart.

DESIGN PHASE

- 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.
- 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 procedures or modules.

Structure charts

- A common tool for illustrating the relations between modules in procedure-oriented design is a structure chart. For example, the elevator system whose state diagram is shown in Figure 10.5 can be designed as a set of modules shown in the structure chart in Figure 10.8.

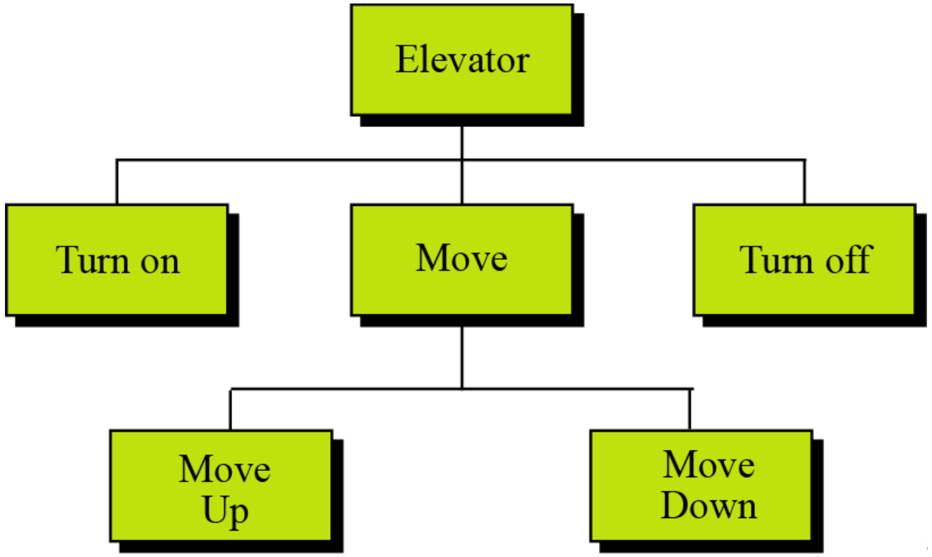


Figure 10.8 A structure chart

Modularity

- Modularity means breaking a large project into smaller parts that can be understood and handled easily.
- In other words, modularity means dividing a large task into small tasks that can communicate with each other.
- The structure chart discussed in the previous section shows the modularity in the elevator system. There are two main concerns when a system is divided into modules: coupling and cohesion

Modularity (Cont.)

- Coupling is a measure of how tightly two modules are bound to each other.
- The more tightly coupled, the less independent they are.
- Since the objective is to make modules as independent as possible, we want them to be loosely coupled.

Coupling between modules in a software system must be minimized.

Modularity (Cont.)

- Another issue in modularity is cohesion. Cohesion is a measure of how closely the modules in a system are related. We need to have maximum possible cohesion between modules in a software system.

Cohesion between modules in a software system must be maximized.

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. The object-oriented design phase lists details of these attributes and methods. Figure below shows an example of the details of our four classes used in the design of the old-style elevator.

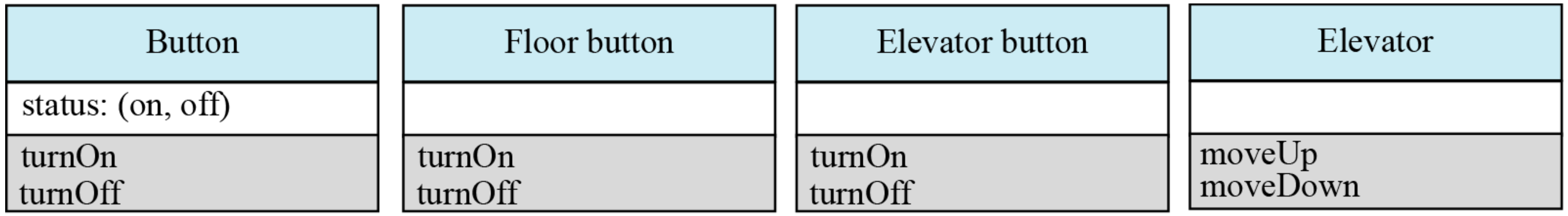


Figure 10.9 An example of classes with attributes and methods

IMPLEMENTATION PHASE

- 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 we need to mention in each case.

Choice of language

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

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.

Software quality factors

- Software quality can be divided into three broad measures: operability, maintainability and transferability. Each of these measures can be further broken down as shown in Figure below.

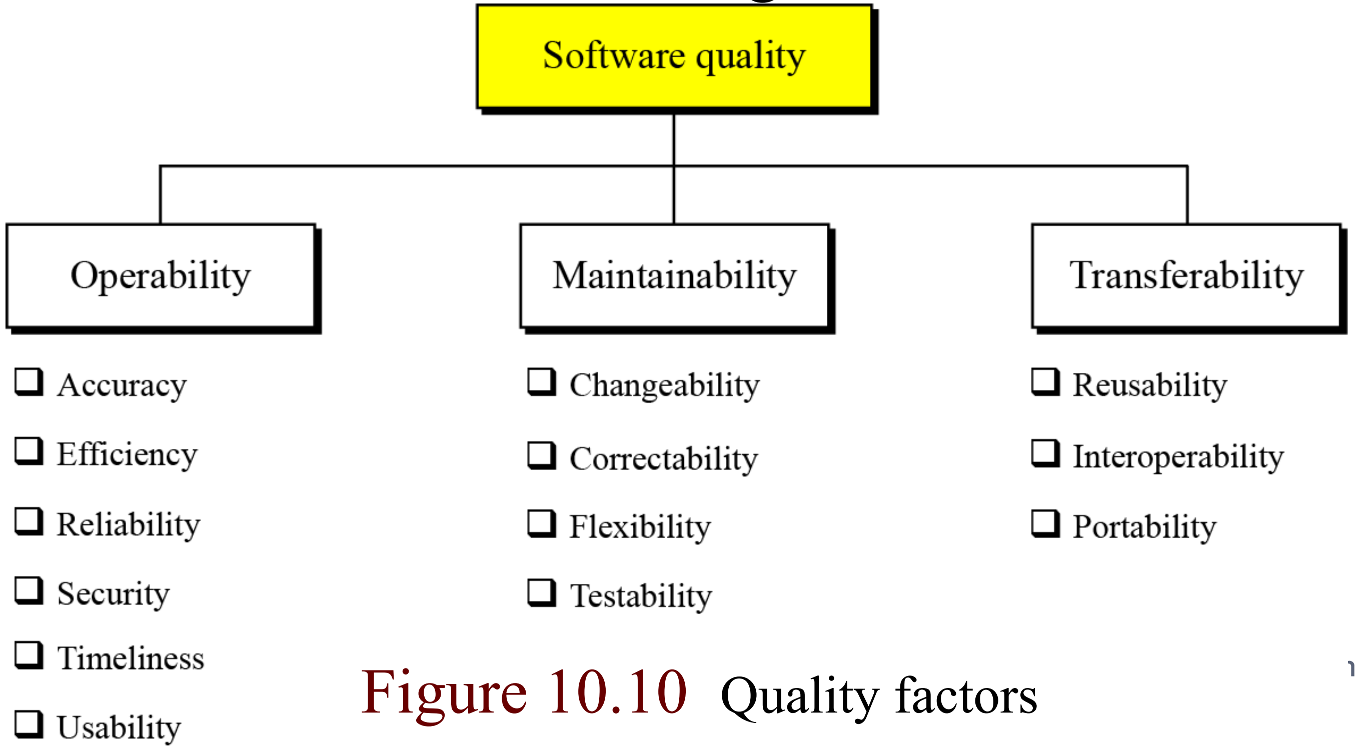


Figure 10.10 Quality factors

Software quality factors

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

Software quality factors (Cont.)

- 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.

- Transferability refers to the ability to move data and/or a system from one platform to another and to reuse code.

TESTING PHASE

- 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: glass-box and black-box.

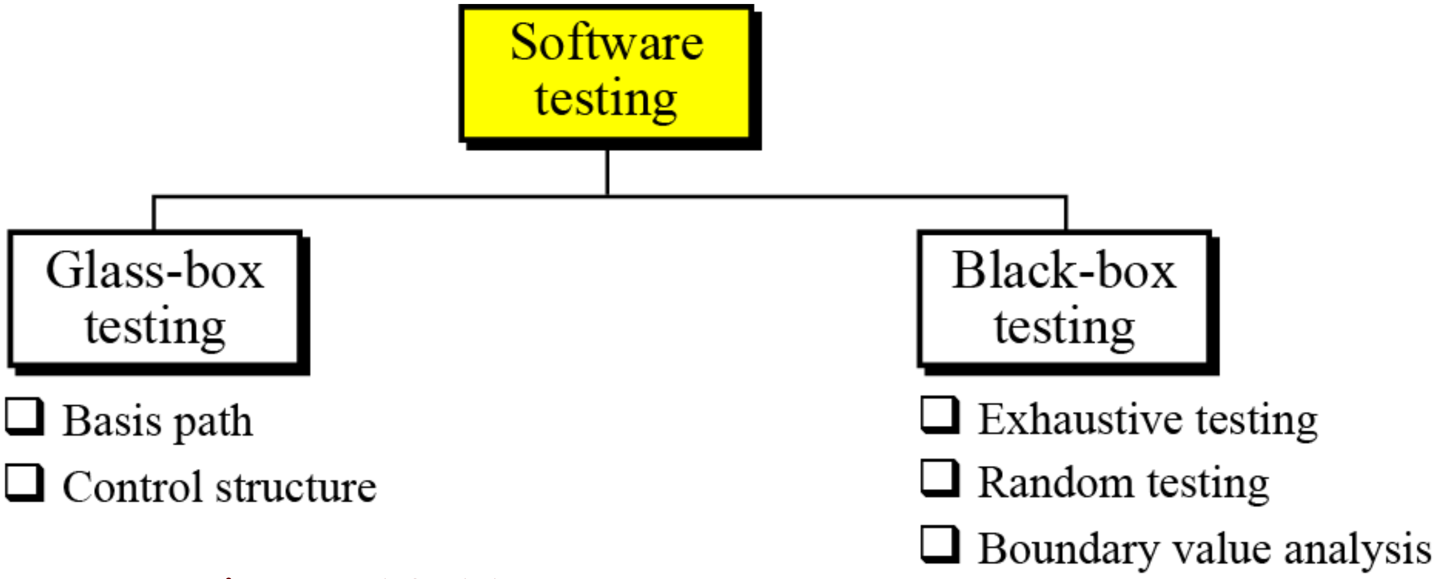


Figure 10.11 Software testing

Glass-box testing

- Glass-box testing (or white-box testing) is based on knowing the internal structure of the software.
- The testing goal is to determine whether all components of the software do what they are designed for.
- 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.

Glass-box testing (Cont.)

- Glass-box testing that uses the structure of the software is required to guarantee that at least the following four criteria are met:
  - All independent paths in every module are tested at least once.
  - All the decision constructs (two-way and multiway) are tested on each branch.
  - Each loop construct is tested.
  - All data structures are tested.
- Several testing methodologies have been designed in the past. We briefly discuss two of them: basis path testing and control structure testing.

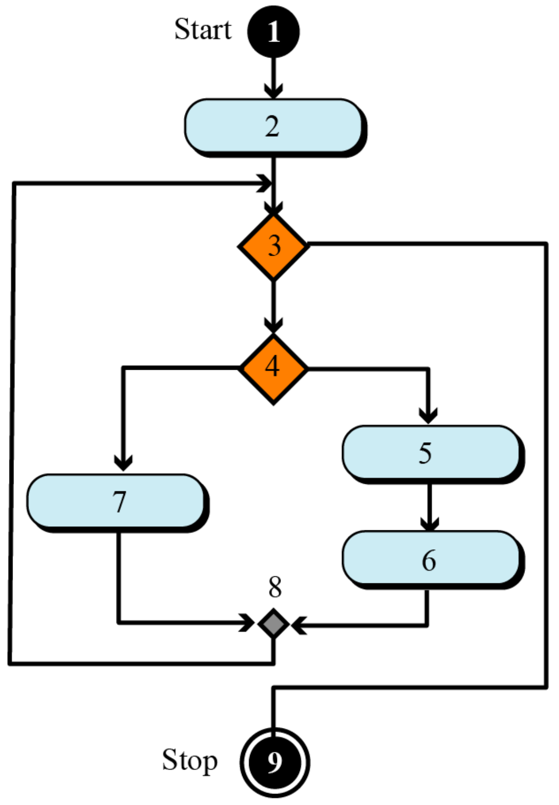
Basis path testing

- Basis path testing was proposed by Tom McCabe. This method creates a set of test cases that executes every statement in the software at least once.

Basis path testing is a method in which each statement in the software is executed at least once.

Example 10.1

To give the idea of basis path testing and finding the independent paths in part of a program, assume that a system is made up of only one program and that the program is only a single loop with the UML diagram shown in Figure 10.12.



Independent path:

Path1: (1, 2, 3, 9)
Path2: (1, 2, 3, 4, 5, 6, 8, 3, 9)
Path3: (1, 2, 3, 4, 7, 8, 3, 9)

Figure 10.12 An example of basis path testing

Control structure testing

- Control structure testing is more comprehensive than basis path testing and includes it. This method uses different categories of tests that are listed below.
  - Condition testing:
    - Applies to any condition expression in the module.
    - Is designed to check whether all conditions (including *simple conditions* and *compound conditions*)are set correctly.
  - Data flow testing:
    - Is based on the flow of data through the module. This type of testing selects test cases that involve checking the value of variables when they are used on the left side of the assignment statement.
  - Loop testing:
    - Uses test cases to check the validity of loops. All types of loops are carefully tested.

Black-box 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.

Black-box testing (Cont.)

- 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.
- 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.

Black-box testing (Cont.)

- 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 $x \geq 100$ is written as $x > 100$.

DOCUMENTATION

- For software to be used properly and maintained efficiently, documentation is needed. Usually, three separate sets of documentation are prepared for software: user documentation, system documentation and technical documentation.
- If the software has problems or it is modified after release, they must be documented too.
- Documentation only stops when the package becomes obsolete

Documentation is an ongoing process.

User documentation

- To run the software system properly, the users need documentation, traditionally called a *user guide*, that shows how to use the software step by step. User guides usually contains a tutorial section to guide the user through each feature of the software.

- A good user guide can be a very powerful marketing tool: the importance of user documentation in marketing cannot be over-emphasized. User guides should be written for both the novice and the expert users, and a software system with good user documentation will definitely increase sales.

System documentation

- System documentation defines the software itself. It should be written so that the software can be maintained and modified by people other than the original developers. System documentation should exist for all four phases of system development.
- In the analysis phase, the information collected should be carefully documented. In addition, the analysts should define the sources of information.

System documentation (Cont.)

- In the design phase, the tools used in the final copy must be documented. For example, the final copy of the chart should be documented with complete explanations.
- In the implementation phase, every module of the code should be documented. In addition, the code should be self-documenting as far as possible using comments and descriptive headers.
- Finally, the developers must carefully document the testing phase. Each type of test applied to the final product should be mentioned along with its results.

Technical documentation

- Technical documentation describes the installation and the servicing of the software system. Installation documentation defines how the software should be installed on each computer, for example, servers and clients. Service documentation defines how the system should be maintained and updated if necessary.

References

- Behrouz Forouzan and Firouz Mosharraf, “Foundations of computer science”, Second edition, chapter10, pp. 271-284
- Jeffrey A. Hoffer ,Joey F. George , and Joseph S. Valacich, Modern Systems Analysis and Design, Sixth Edition, Chapter 1.