Chapter 10

Software Engineering

OBJECTIVES

After reading this chapter, the reader should be able to:

- Understand the software life cycle.
- Describe the development process models.
- Understand the concept of modularity in software engineering.
- Understand the importance of quality in software engineering.
- Understand the role of documentation in software engineering.

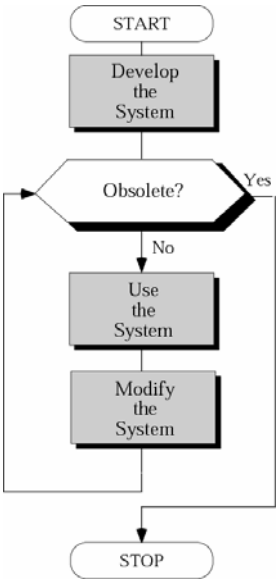
10.1

SOFTWARE LIFE CYCLE

Brooks/Cole Thomson Learning

Figure 10-1

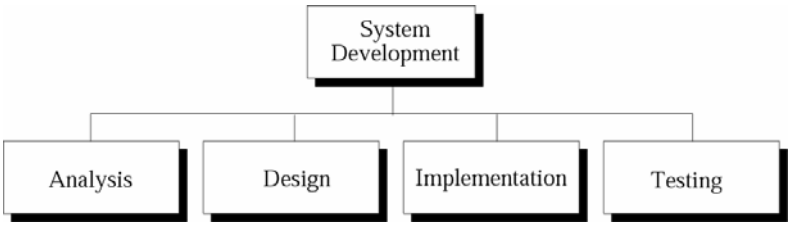
System life cycle



Brooks/Cole Thomson Learning

Figure 10-2

System development phases



Brooks/Cole Thomson Learning

System Development: 4 Phases

- Analysis phase:
  - Requirements that the packages should accomplish.
  - 4 steps:
    - Define the user
      - Accounting package: For any firm
      - Customized banking package: For a specific bank
    - Define the needs
      - The best answers come from the users.
    - Define the requirements
      - Based on the users' needs. Define system requirements.
    - Define the methods.

Brooks/Cole Thomson Learning

System Development: 4 Phases

- Design Phase:
  - How the systems will accomplish what was defined in analysis phase.
  - OS, files and databases are determined.
  - Modularity
    - Divide the package into small modules.
    - Link through a main program.
  - Tools
    - Like structure chart.

System Development: 4 Phases

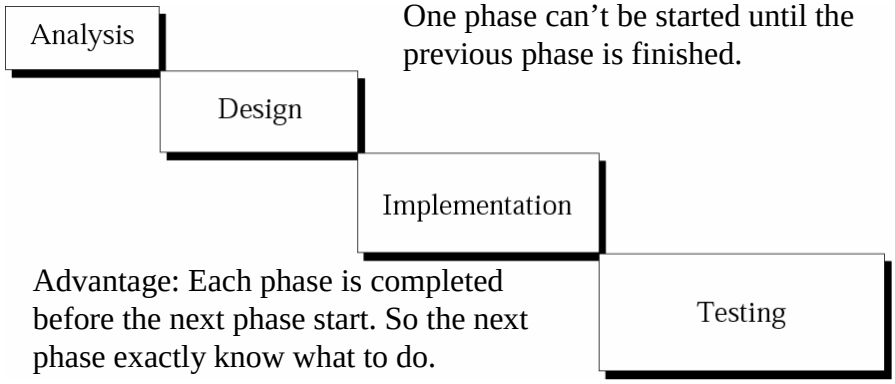
- Implementation Phase:
  - Create actual programs.
  - Tools:
    - Flowchart, and Pseudocode.
  - Coding
- Test Phase
  - Specialists: test engineers.
  - Two kind of tests
    - Black box testing
    - White box testing

System Development: 4 Phases

- Black box testing:
  - Test without knowing how it works.
  - Test according to package requirements.
- White box testing:
  - Know everything about the program
  - Programmer's responsibility.
    - Testing plans in design stages
    - Eye toward tests for flowcharts and pseudocodes.

Figure 10-3

Waterfall model



One phase can't be started until the previous phase is finished.

Advantage: Each phase is completed before the next phase start. So the next phase exactly know what to do.

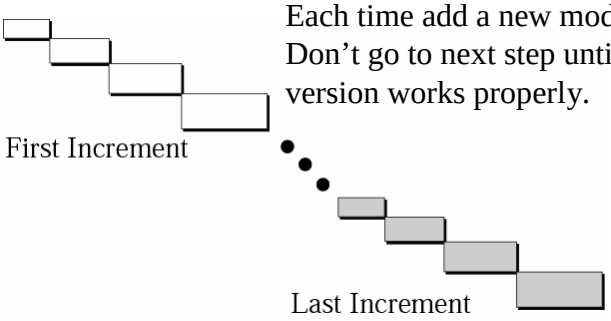
Disadvantage: When locating a problem, the entire process should be investigated.

Brooks/Cole Thomson Learning

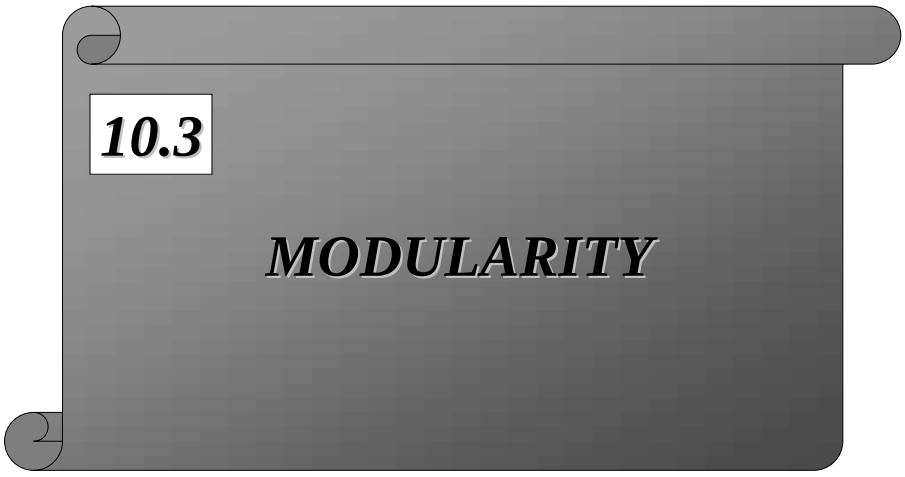
Figure 10-4

Incremental model

A simplified version→ Add details…
Begin with main module
Each time add a new module
Don't go to next step until the previous version works properly.



Brooks/Cole Thomson Learning



Modularity

- Definition: Break a large project into small parts that can be understood and handled.
- Two tools:
  - Structure chart: procedural programming
    - An example is given in later slide.
  - Class diagram: Object oriented programming
- Two measure for modularity
  - Coupling: Data relation
  - Cohesion: Process relation

A Simple Example for Structure Chart and Data Flow

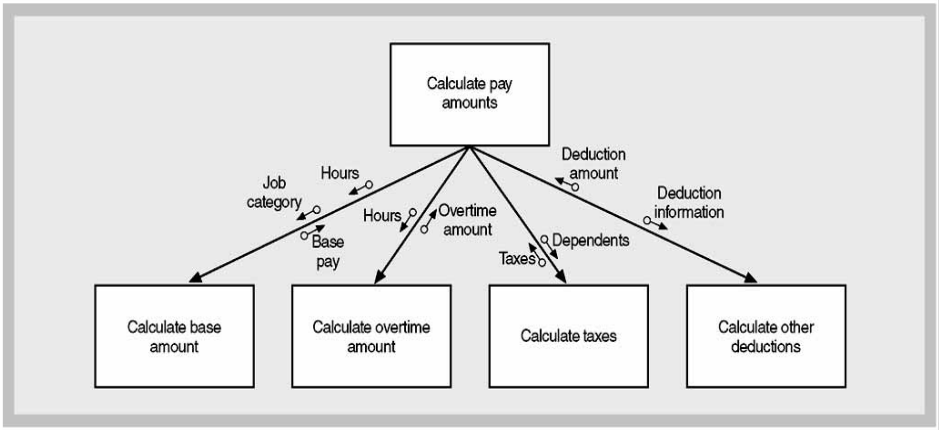


FIGURE 10-6

A simple structure chart for the *Calculate pay amounts* module.

Coupling

- Definition: Measure of how tightly two modules are bounded to each other.
  - Independent modules are prefer→ Loosely coupled
  - Code reusable
  - Less likely to create errors.
  - Easy to maintain
- Data coupling
  - Only passed min. required data to the called function.
- Stamp coupling
  - Send the composite object (like array) to the called function.
  - Send extra data might result in errors or side effects.

Coupling

– Control coupling
  - Pass the flag that direct the logic flow of a function.
  - Used to communicate the status.

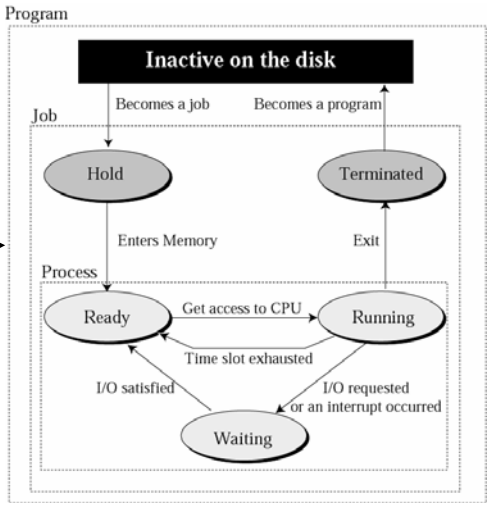
– Global coupling
  - Use global variables to communicate among functions.
  - When changing the program, not impossible to isolate the impact.
  - Functions are not easily reused in other programs.

– Content coupling
  - One function refers directly to the data or statements in another one.
  - Break structure programming

Cohesion

- Def: Measure how closely the processes in a program are related.

One program may have many processes.



An Example to Fork a Process (Used in UNIX programming)

```
• void main()
{
 pid_t p=fork();
 if(p==0)
 {
    //child process
   Do something that a child process should do
  }
  else
  {
     //Parent process
   Do something that a parent process should do
 }
}
```


Cohesion

- Functional cohesion
  - Only one process
  - Each function should only do one thing
  - One thing should be done in one place.
- Sequential cohesion
  - The output of one process is input to another
  - Ex: Calculate the price
  - Extend item price → Sum item prices →Calculate the tax→ Calculate the total.

Cohesion

- Communicational cohesion
  - Process works on the same data
- The above three level are good structured programming principles.
- Procedural cohesion
  - Combines unrelated processes that are linked by control flows.
- Temporal cohesion
  - Unrelated processes that occurs together.
  - Init. and finalized of a job.
- Logical and coincidence cohesion
  - Seldom found today.

Brooks/Cole Thomson Learning
©Brooks/Cole, 2003

Quality Software

- How to measure the quality of a software?
- Definition of Quality software
  - Software that satisfies the user's explicit and implicit needs, is well documented, meets the operating standards, run efficiently.
  - Three broad measure
    - See later slide.

Figure 10-5

Quality factors

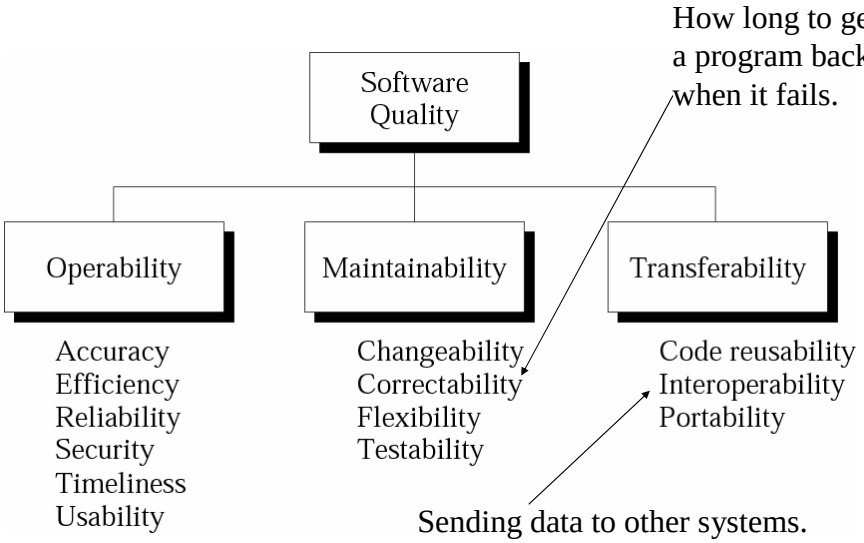
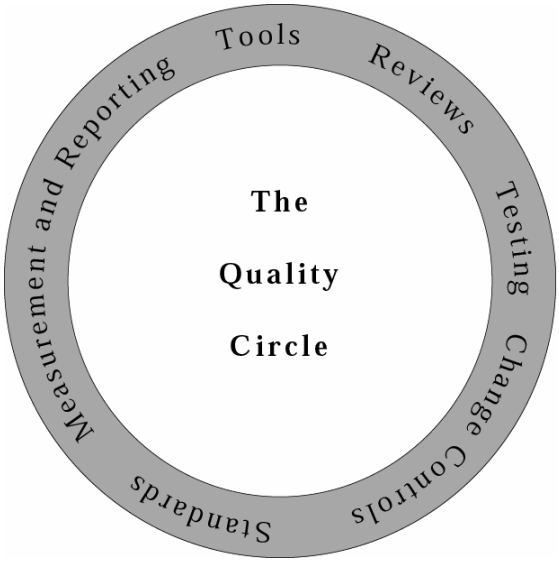


Figure 10-6

Quality circle



Brooks/Cole
Thomson Learning

10.5

DOCUMENTATION

Brooks/Cole
Thomson Learning

Documentation

- User documentation: manual
- System documentation
  - Analysis phase: information collected, source, methods
  - Design phase: Structure chart
  - Implementation phase
    - General documentation
    - Function documentation
  - Testing phase
- Documentation is an ongoing process until the package become obsolete.