



Chapter Twelve



Systems Design and Development

After reading this chapter, you should be able to:

- Describe the process of designing, programming, and debugging a computer program
- Explain why there are many different programming languages and give examples of several
- Explain why computer languages are built into applications, operating systems and utilities

After reading this chapter, you should be able to:

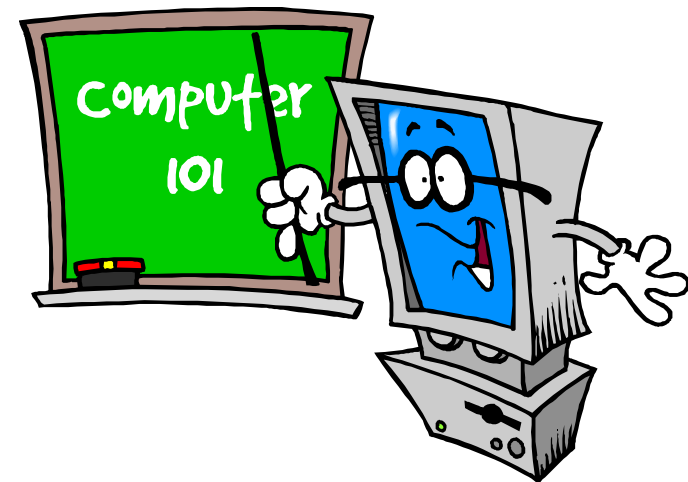
- Outline the steps in the life cycle of an information system and explain the purpose of program maintenance
- Explain the relationship between computer programming and computer science
- Describe the problems faced by software engineers in trying to produce reliable large systems

Chapter Outline

- How People Make Programs
- The Languages of Computers
- Programs in Perspective: Systems Analysis and the System Life Cycle
- The Science of Computing
- The State of Software

How People Make Programs

- Problem solving process involves:
 - Understanding the problem
 - Devising a plan for solving the problem
 - Carry out the plan
 - Evaluating the solution



How People Make Programs



Programs are solutions to problems.

Programming is a specialized form of problem solving.

Four-Step Process to Programming



Define the problem:
make sure you know
what the problem is.

**Devise, refine, and
test the algorithm:**
break the problem
into smaller,
solvable parts
(stepwise
refinement).

Four-Step Process to Programming

Write the program: begin with pseudocode and eventually use a computer language.

Test and debug the program: to make sure it solves the initial problem that was defined.

From Idea to Algorithm

Stepwise Refinement: complex problems will need to be broken into smaller problems.

Control Structures: logical structures that control the order in which the instructions the computer is to follow.

Testing the Algorithm: check the logic.

Stepwise Refinement

A complex problem like writing a computer game needs to be broken into three parts: a beginning, a middle, and an end.

Further refinement adds more detail to each part.

For example:

- begin the game
- repeat player's turn until the player guesses right answer or seven turns are completed
- end the game

Control Structures

Sequence:

instructions are followed in the order given.

*Display instructions
pick a number between 1 and 100
set counter to 0*

Selection:

instructions are based on logical decisions.

*if guess < number, then
say guess is too small;
else say guess is too big*

Control Structures

Repetition:

instructions are repeated until some condition is satisfied.

```
repeat turn until number is  
guessed or counter = 7  
    input guess from user  
    add 1 to counter  
end repeat
```

From Algorithm to Program

If the logic of the algorithm tests accurately, it can then be written into a program.

Writing a program is called **coding**.



From Algorithm to Program

The program will have three parts:

Program heading contains the name of the program and data files.

Declarations and definitions of variables and other programmer-defined items.

Body contains the instructions the computer will follow.

Into the Computer

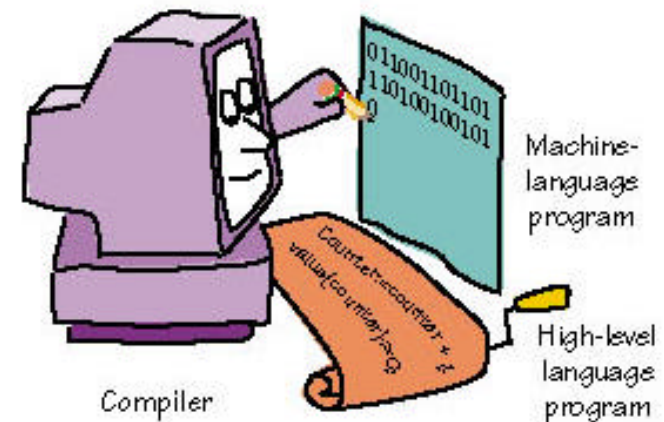
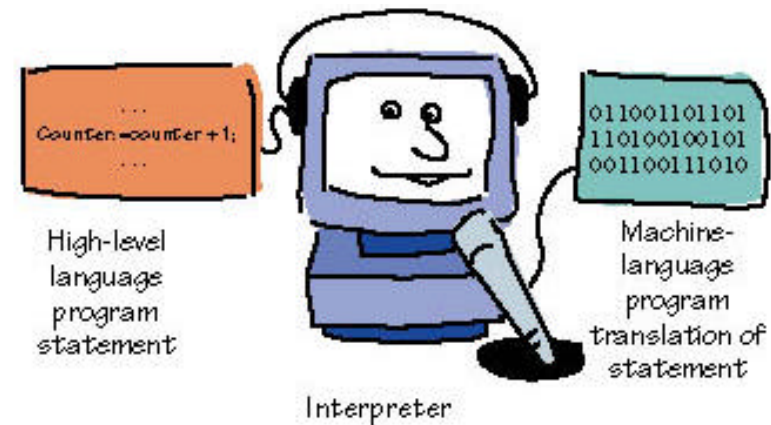
Next, the program needs to be entered into the computer using a **text editor**.

Once typed into a text editor, it can be saved to disk.

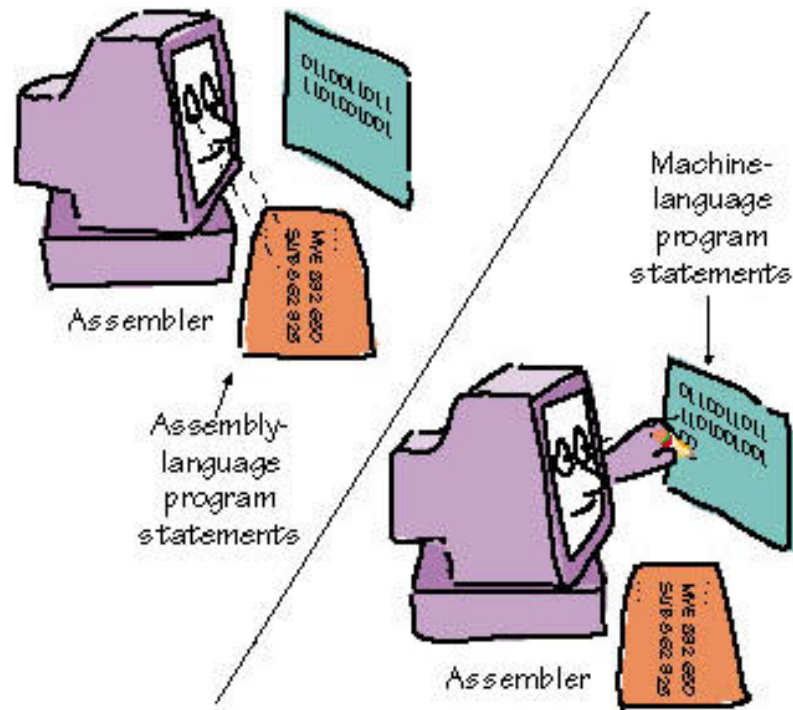
To run or execute the program, it must be **translated** into machine language by means of an interpreter or compiler.

Interpreters and Compilers

- **Interpreter:** each instruction is translated individually
- **Compiler:** the entire program is translated into machine language.



The Languages of Computers



Every computer has a native language - a **machine language**.

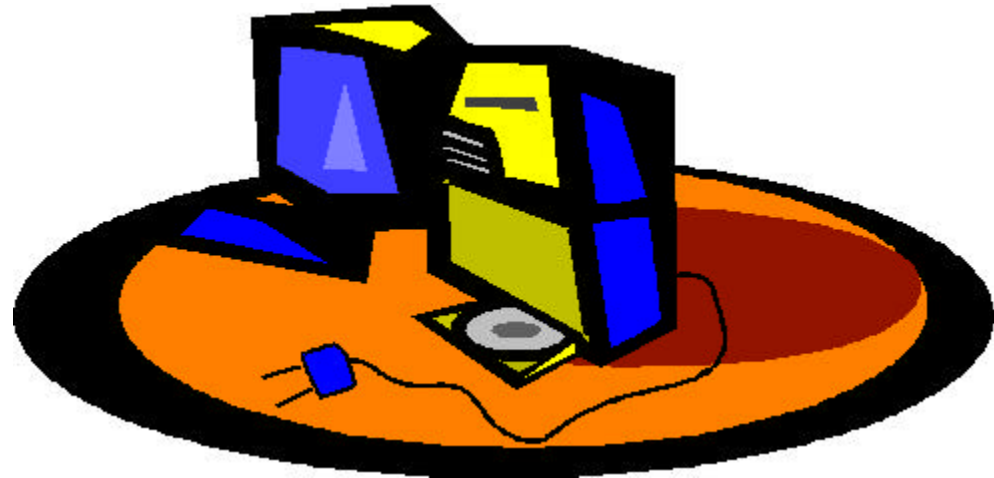
Machine language and Assembly language:

- Difficult to learn and use.
- Examples include machine and assembly languages.

The Languages of Computers

High-level Languages:

- English-like vocabulary.
- Transportable
- Examples include C++, LOGO, and BASIC.



Multilingual Machines

Well known high-level programming languages include:

FORTRAN (Formula Translation): the first commercial high-level language.

COBOL (Common Business Oriented Language): developed for business data processing problems.

Multilingual Machines

Well known high-level programming languages include:

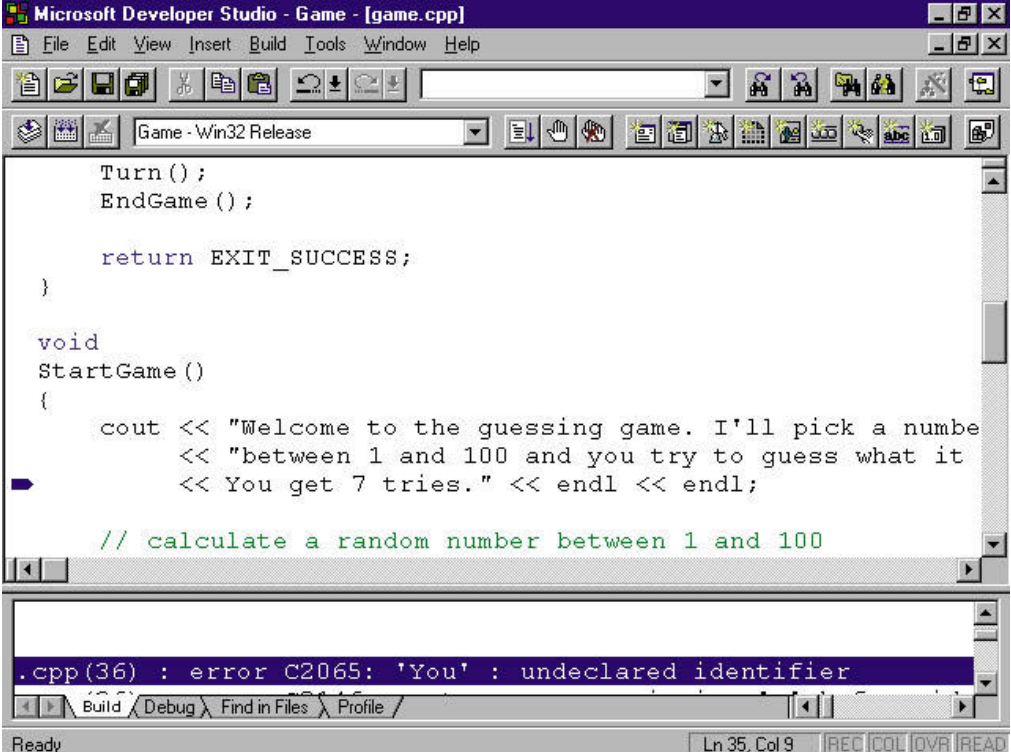
LISP (List Processing): developed to process non-numeric data like characters, words, and other symbols.

BASIC (Beginners All-purpose Symbolic Instruction Code): developed as an easy-to-learn language for beginners.

Multilingual Machines

Pascal: designed to encourage structured programming.

C: developed as a tool for programming operating systems such as UNIX.

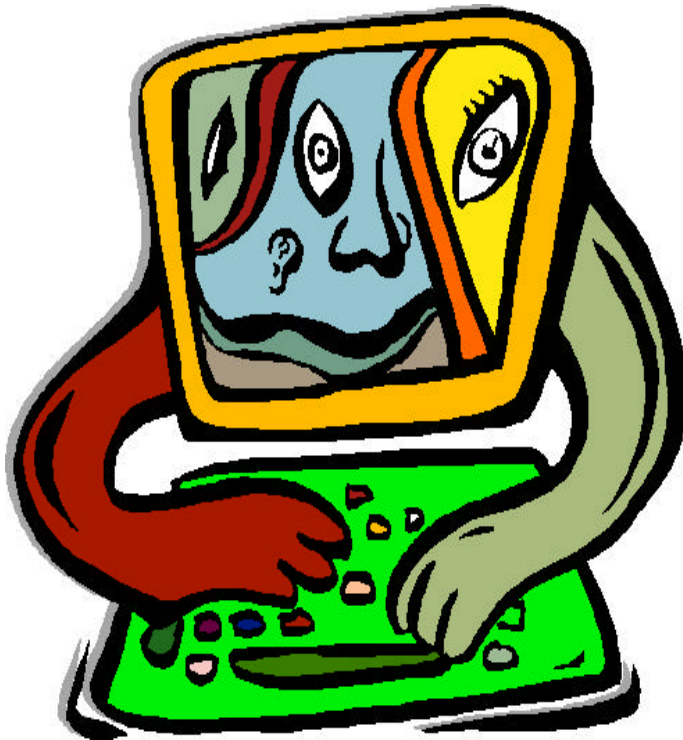


The screenshot shows the Microsoft Developer Studio interface. The main window displays a C++ program named 'game.cpp'. The code includes functions 'Turn()', 'EndGame()', and 'StartGame()'. The 'StartGame()' function contains a 'cout' statement and a comment. A compilation error is shown in the output window at the bottom, indicating an undeclared identifier 'You'.

```
Turn();  
EndGame();  
  
return EXIT_SUCCESS;  
}  
  
void  
StartGame()  
{  
    cout << "Welcome to the guessing game. I'll pick a number  
    << "between 1 and 100 and you try to guess what it  
    << "You get 7 tries." << endl << endl;  
    // calculate a random number between 1 and 100  
}
```

.cpp(36) : error C2065: 'You' : undeclared identifier

Multilingual Machines



Ada: is a massive language developed for the US Government

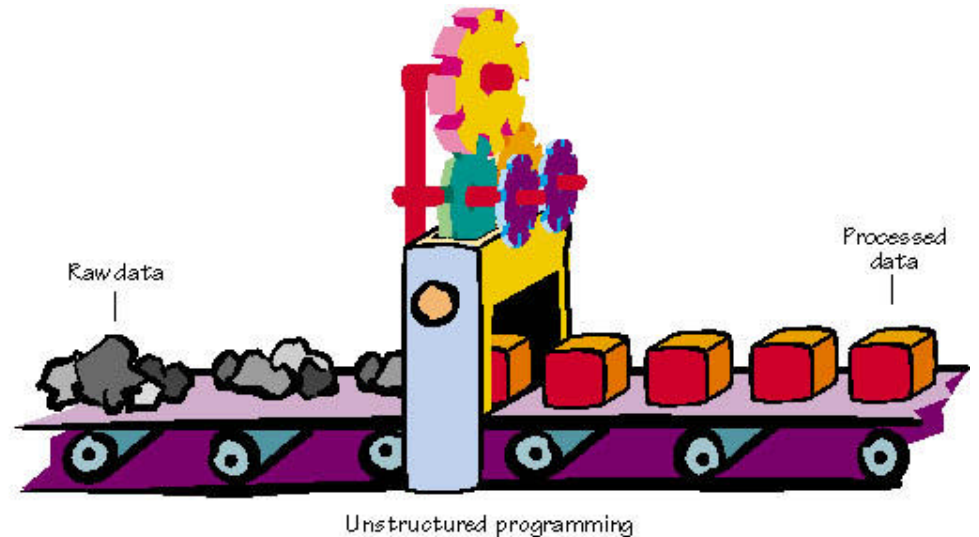
PROLOG: designed for working with logical relationships between facts

LOGO: is a dialect of LISP specially designed for children.

Programming Methodologies

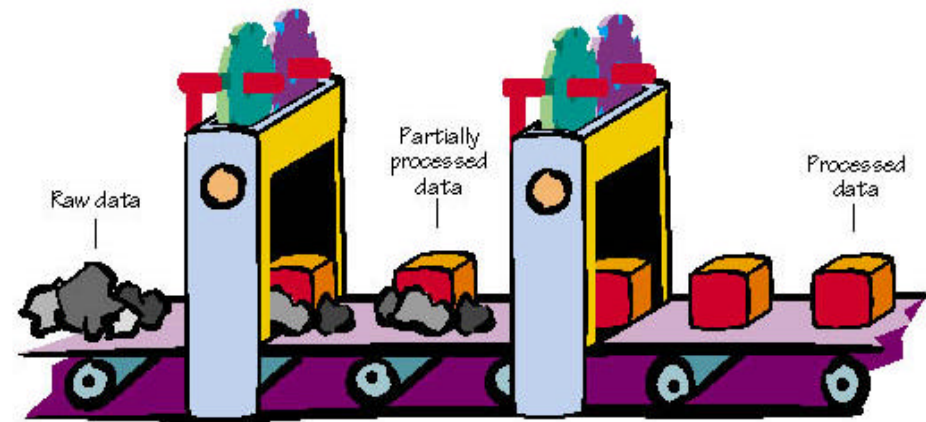
Structured

Programming: a technique to make the programming process easier and more productive by writing many small programs.

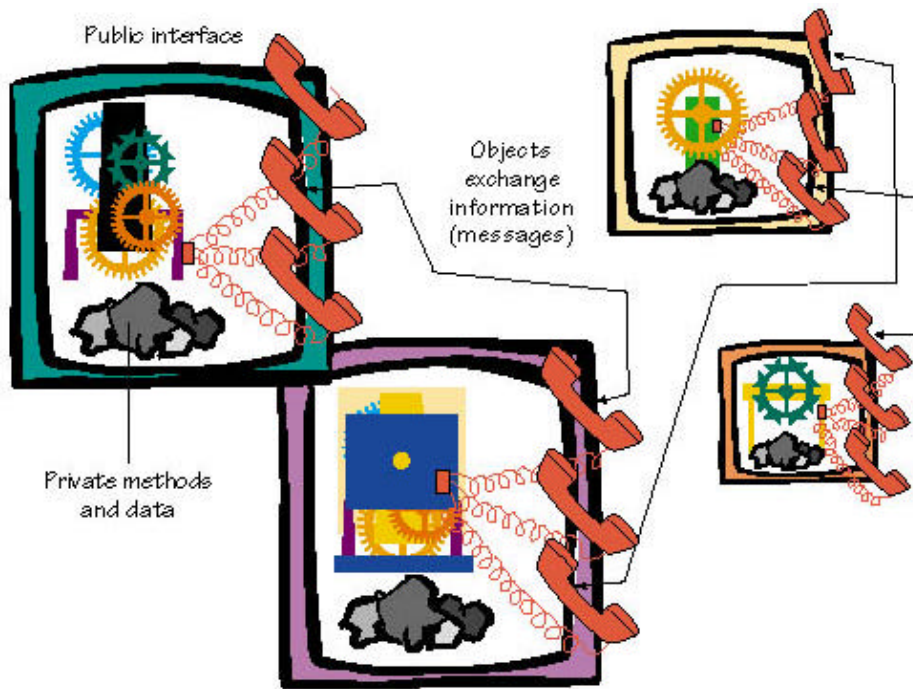


Programming Methodologies

- A program is well structured if it is:
 - made up of logically cohesive modules
 - arranged in a hierarchy
 - straightforward and readable

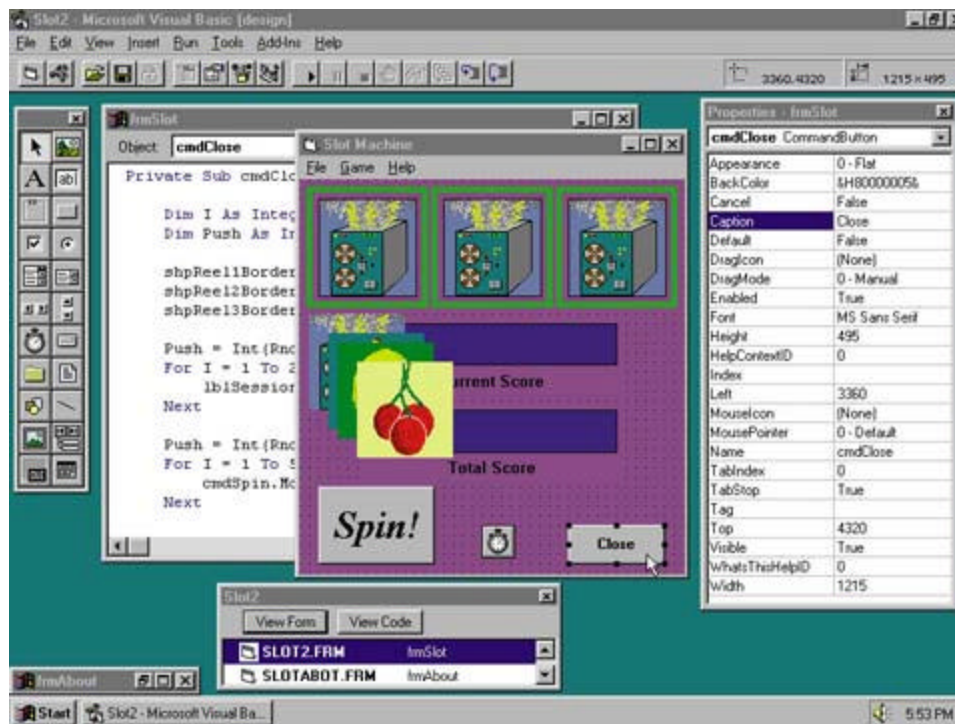


Programming Methodologies



Object-oriented Programming: the program is a collection of interactive objects that contain both data and instructions.

Programming Methodologies

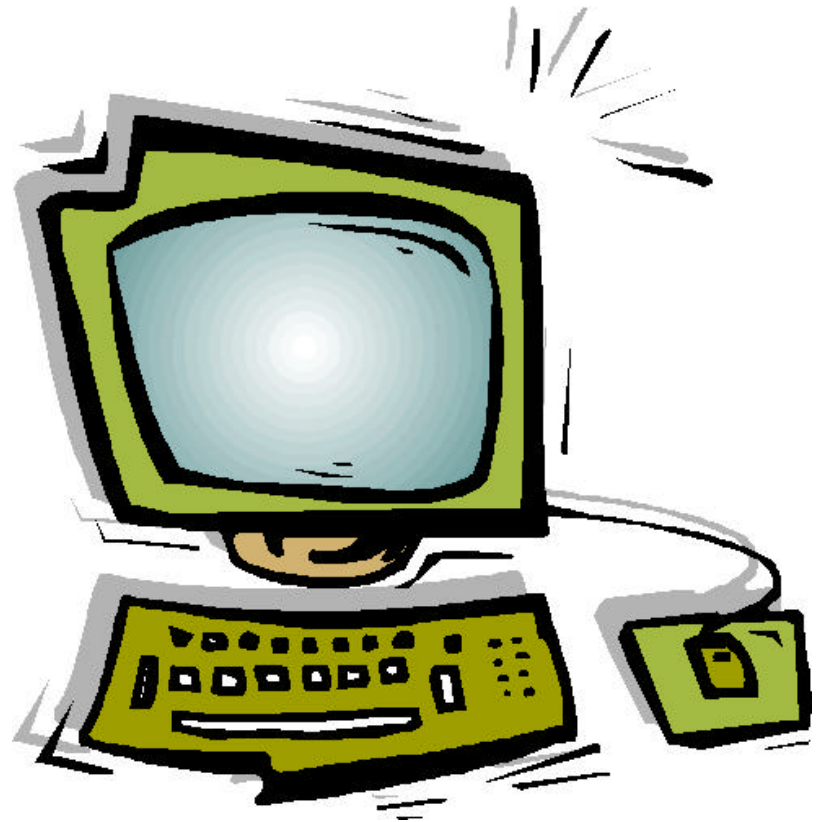


- **Visual Programming:** programmers write programs by drawing pictures and pointing to objects on the screen.

Languages for Users

Languages designed to meet the needs of most computer users include:

Macro or scripting languages: used in automating a repetitive task.



Languages for Users

Fourth-generation languages (4GLs):
easier to use and more like natural language. Characteristics include:

- English-like phrases
- Non-procedural
- Increases productivity

Querying a database with a *query language* is one example.

Languages for Users

- **Component Software:** users construct small custom applications from software components.
 - Plug-ins for Netscape Navigator and Internet Explorer
 - JavaScript

The Future of Programming

Trends:

- Natural languages and artificial intelligence will provide users with programming tools that will understand the language of the user.
- The distinction between user and programmer will begin to fade. Users won't need to master complicated programming languages to construct applications.

The Future of Programming

- Programs will program themselves based on simple descriptions provided by the user.



Systems Analysis and the Systems Life Cycle

Programs are only part of larger
information systems - collections of
people, machines, data, and methods
organized to accomplish specific
functions and to solve a problem.

Systems Analysis and the Systems Life Cycle

System Life Cycle - a sequence of steps or phases it passes through between the time the system is conceived and the time it is phased out.

Systems analyst - a computer professional primarily responsible for developing and managing a system as it progresses through these phases.

The Systems Development Life Cycle

The systems development life cycle is a sequence of steps followed by a project team.

Investigation: “Why is there a problem?”

Analysis: “What is the problem?”

Design: “How can the problem be solved?”



The Systems Development Life Cycle

Development: teams of programmers and others begin developing the various parts of the system.

Implementation: the system is put to work.

Maintenance: ongoing upgrades.

Retirement: phasing out the current system.

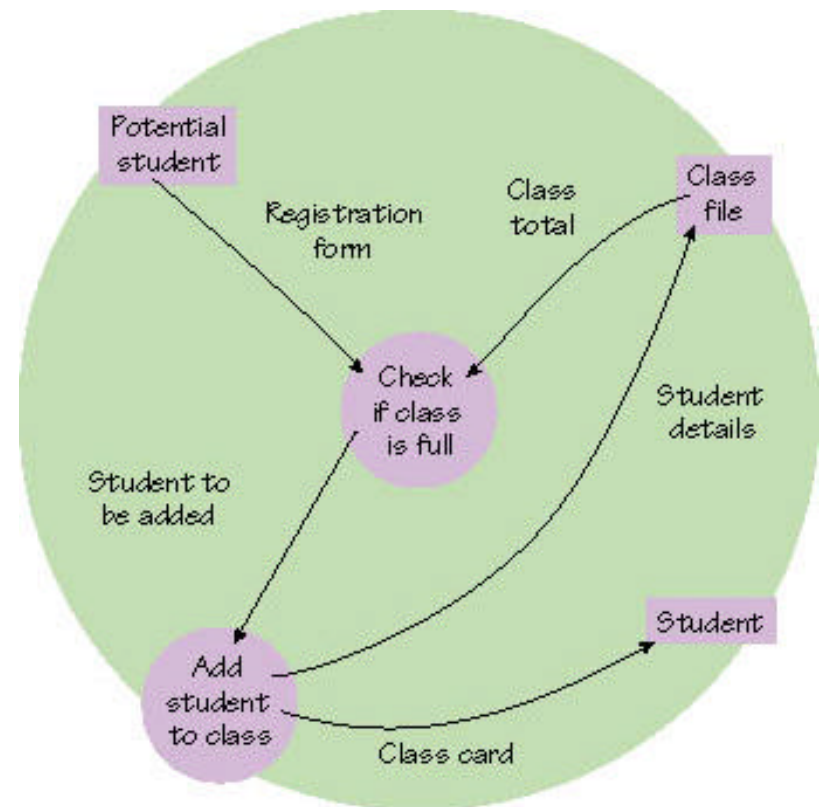
Investigation

Systems investigation involves defining the problem - identifying the information needs of the organization, examining the current system, needs of organization, and studying feasibility of changing systems.

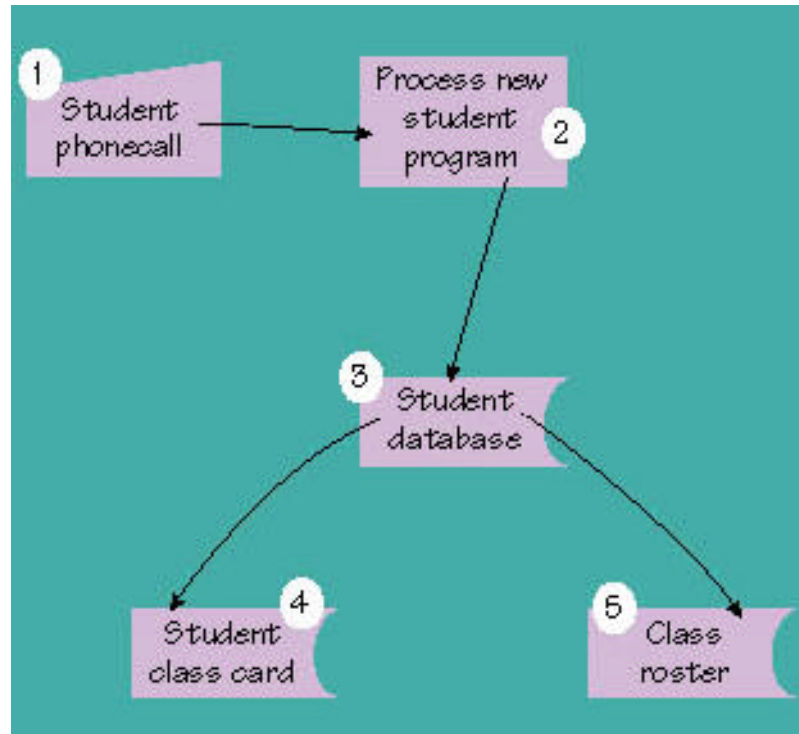
This phase produces a **feasibility study**.

Analysis

The systems analyst gathers documents, interviews users, observes the system in action, and generally gathers and analyses data to understand the current system.



Design

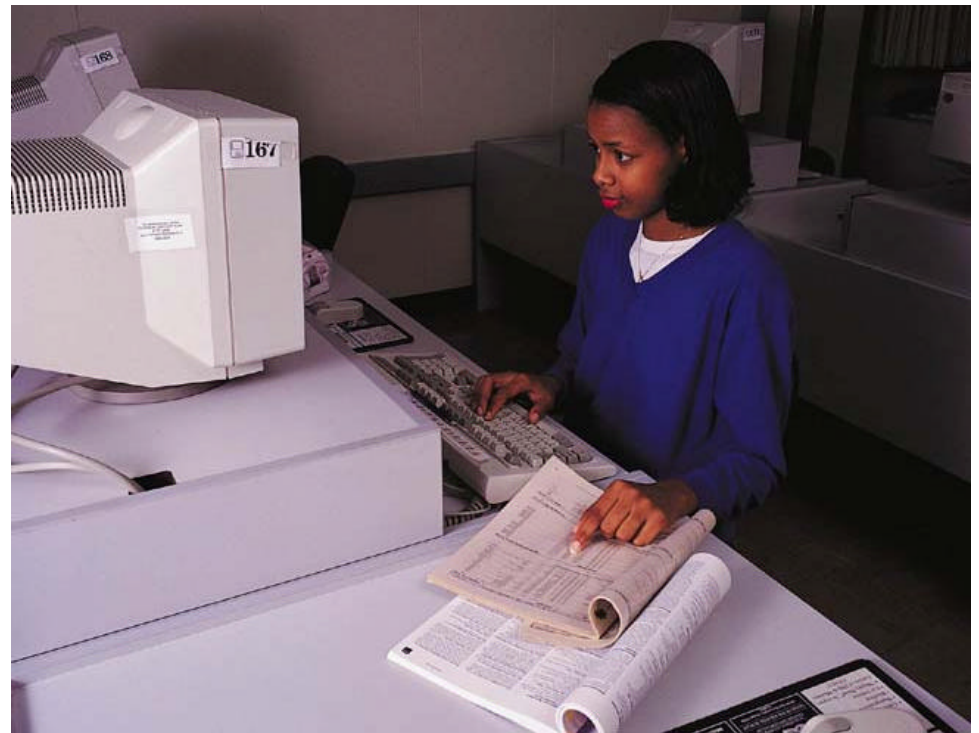


In the design phase, the systems analyst develops the system specifications that describe ***how*** exactly the system requirements will be met.

Design

The systems analyst considers:

- user interface design.
- database design.
- process design.

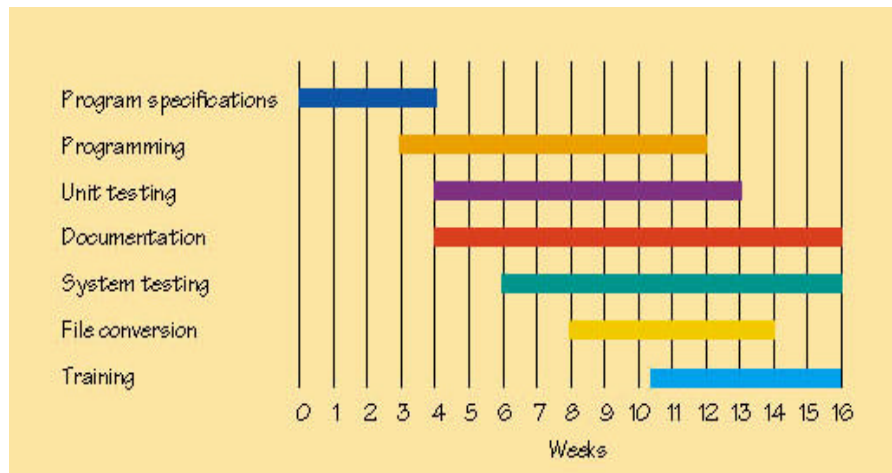


Development

The development phase is a process of turning the design specifications into a real working system.

The initial testing of the system is known as **alpha testing** and potential users do **beta testing** after the bugs are worked out.

Development



Development includes a complex mix of scheduling; hardware, software, and communications purchasing; documentation; and programming.

Implementation

This phase may involve extensive training and technical user support.

Implementation includes user education and training, equipment replacement, file conversion, and careful monitoring of the new system for problems.

Maintenance



The maintenance phase involves monitoring, evaluating, repairing, and enhancing the system throughout the life cycle.

Retirement

At some point in the life of a system, on-going maintenance is not enough. The needs of an organization change, users' expectations change, and there is always new technology available.



The Science of Computing

- **Computer theory** - applies concepts of theoretical mathematics to computational problems.
- **Algorithms** - logical underpinnings of computer programs.
- **Data structures** - define the logical structure of data

The Science of Computing

- **Programming concepts and languages** - have evolved through generations.
- **Computer architecture** - deals with the way hardware and software work together.

The Science of Computing

- **Management information systems (MIS)** - is part computer science, part business.
 - MIS specialist focus on developing systems in timely, reliable and useful information to manager in business.
 - MIS applies theoretical concepts of computer science to real-world problems.

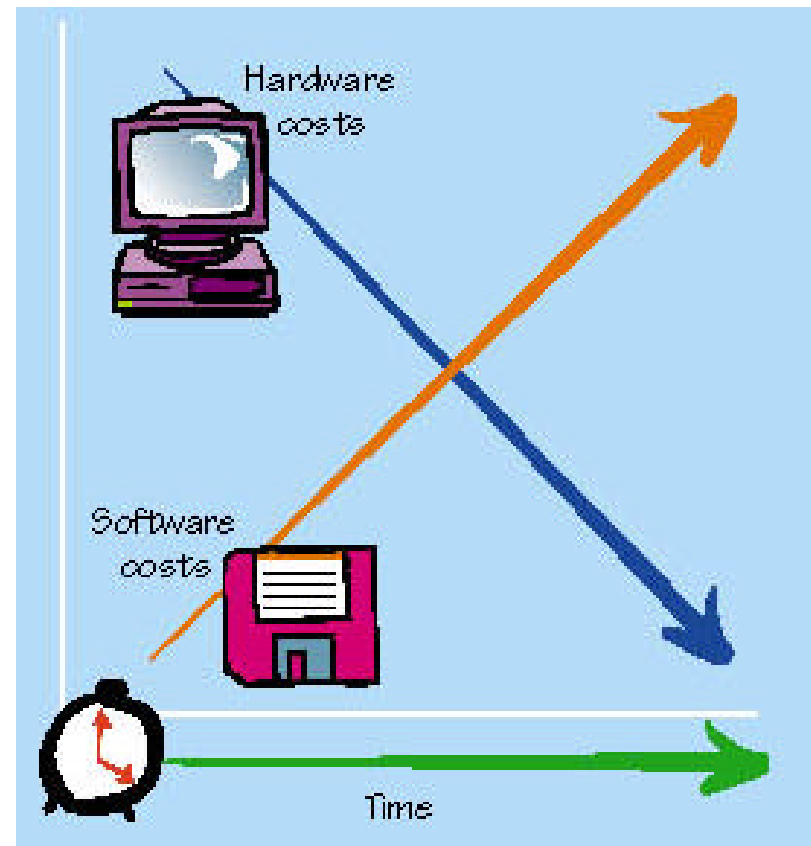
The Science of Computing

- **Software engineering** - is a relatively new branch of computer science that attempts to apply engineering principles and techniques to the less-than-concrete world of computer software.

The State of Software

The problems faced by software engineers affect all of us.

Two inherent problems in software development are **cost** and **reliability**.



Software Problems

Cost:

The cost of hardware has dropped but the cost of developing software has continued to rise.

Reliability:

Software errors include errors of omission, syntax, logic, clerical, capacity, and judgement.

Software Solutions

Responding to the cost and reliability issues, computer scientists are working to improve:

- Programming Techniques
- Programming Environment
- Program Verification
- Clean Room Programming
- Human Management

