Chương 6. LẬP TRÌNH

6.1. Ngôn ngữ lập trình

6.2. Xử lý ngôn ngữ

6.3. Thiết kế chương trình

6.4. Kiểm tra và gỡ rối

6.5. Lập tài liệu

6.1. Ngôn ngữ lập trình

- Ngôn ngữ máy:
  - Là tập hợp gồm nhiều lệnh máy
- Lệnh máy
  - Là một chuỗi các bit 0 và 1
  - Chỉ thực hiện một số tác vụ đơn giản như các phép tính số học và các hoạt động đọc ghi vùng nhớ/ thanh ghi
  - Một lệnh máy bao gồm 2 phần: mã lệnh và toán hạng
  - Chỉ có hai cấu trúc điều khiển cơ bản để thực hiện các lệnh: tuần tự và nhảy

- Ngôn ngữ lập trình:
  - Chương trình (*program*): Một đoạn mã lệnh yêu cầu máy tính thực hiện một công việc cụ thể nào đó
  - Lập trình (*programming*): Viết chương trình, bằng cách sử dụng một ngôn ngữ lập trình
  - Ngôn ngữ lập trình: Một hình thức ngôn ngữ giúp con người biểu diễn ý tưởng của mình dưới dạng chương trình, nhờ đó máy tính có thể thực hiện được ý tưởng này

Ví dụ

1010 0001 0000 0000 0000 0001 0000 0101 0000 0101 0000 0000 1010 0011 0000 0000…..

(64 kí số 0, 1)

⇒Ngôn ngữ máy, con người hầu như không thể đọc hiểu được (rất khó viết và hầu như không thể đọc)

Ngôn ngữ lập trình cấp thấp và cấp cao

- Ngôn ngữ lập trình mức thấp nhất: ngôn ngữ máy.
- Ngôn ngữ lập trình cấp cao: ngôn ngữ nhiều kiểu diễu dữ liệu và nhiều cấu trúc điều khiển hơn so với những gì được cung cấp bởi ngôn ngữ máy; đồng thời cách biểu diễn các phát biểu cũng gần với ngôn ngữ tự nhiên hơn

Một số dạng ngôn ngữ lập trình cấp cao

- Đa mục đích: Basic, C (C++, C#), Java, Fortran, Pascal
- Lập trình đệ quy
- Lập trình khai báo và lập trình thủ tục
  - Khai báo: C, Pascal
  - Thủ tục: Prolog, Lisp
- Lập trình logic
- Ngôn ngữ lập trình hàm
- Lập trình hướng đối tượng

Ví dụ

1010 0001 0000 0000 0000 0001 0000 0101 0000 0101 0000 0000 1010 0011 0000 0000…..

(64 kí số 0, 1)

⇒Ngôn ngữ máy, con người hầu như không thể đọc hiểu được (rất khó viết và hầu như không thể đọc)

```
Mov ax, [100]
Add ax, 5
Mov [100], ax
```

⇒ Hợp ngữ, con người có thể đọc được, nhưng chỉ bao gồm các cấu trúc tuần tự và nhảy của ngôn ngữ máy ⇒ khó biểu diễn ý tưởng (khó đọc và khó viết)

⇒ a := a + 5 (a chứa tại địa chỉ [100])

Ví dụ (tt)

a := 3 +1;

For i:=1 to 3 do read(a);

⇒ Ngôn ngữ Pascal: cung cấp nhiều cấu trúc dữ liệu hơn, có thể biểu diễn ý tưởng tốt hơn, tuy nhiên vẫn tương đối khó đọc so với ngôn ngữ tự nhiên (dễ viết và tương đối dễ đọc)

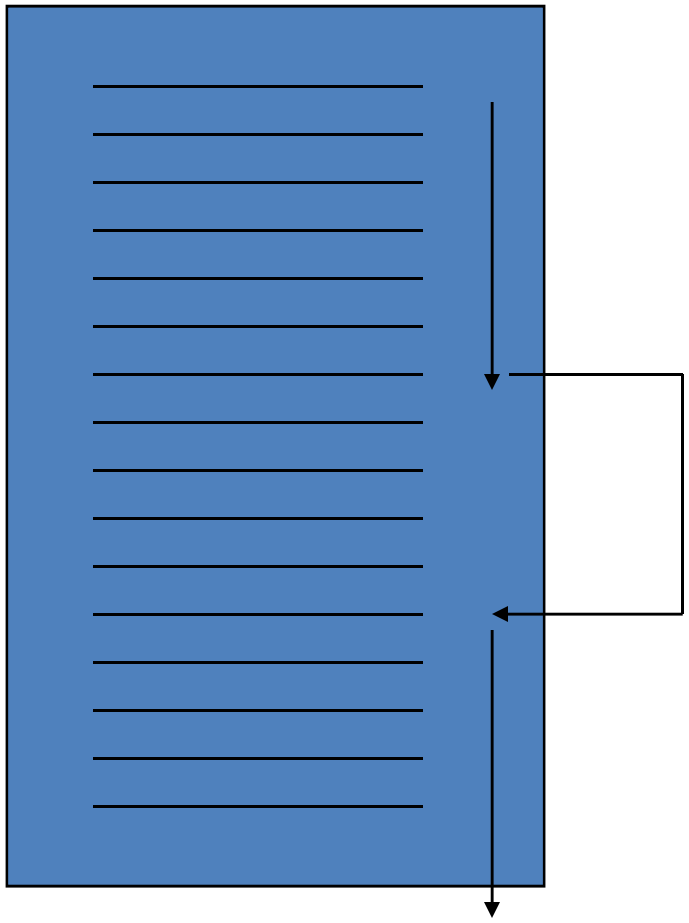
Select ID from Table Where ID > 5

⇒ Ngôn ngữ SQL: rất gần với ngôn ngữ tự nhiên, tuy nhiên không cung cấp nhiều cấu trúc điều khiển (dễ đọc, khó viết)

Cơ chế thực hiện chương trình

- Cơ chế tuần tự
- Cơ chế nhảy

=> Lặp và rẽ nhánh



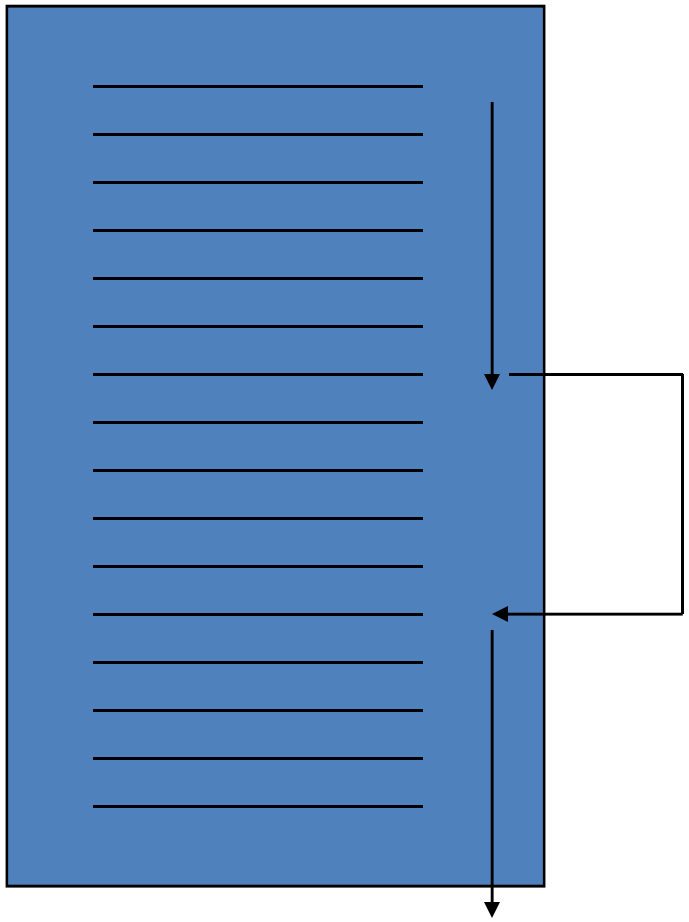
Cấu trúc điều khiển

- Cấu trúc điều khiển: Một cấu trúc ngôn ngữ quy định thứ tự thực hiện các lệnh
- Ngôn ngữ máy: Tuần tự và nhảy
- Ngôn ngữ cấp cao
  - Rẽ nhánh
  - Lặp

Cấu trúc tuần tự và nhảy

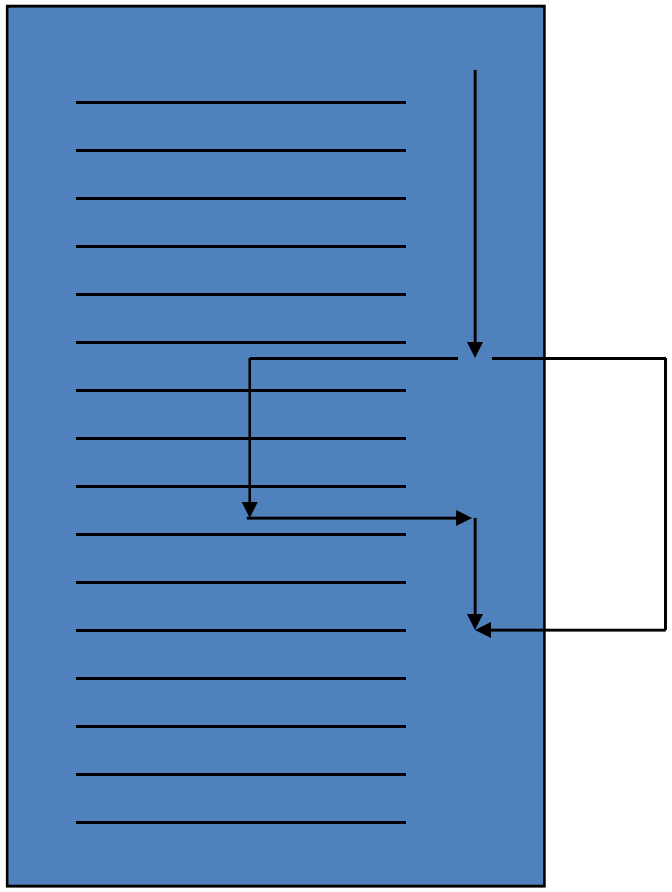
```
    A = 1;
    Goto Lable1;
    A = A*2;
Label1:
    A = A + 3
```

A=?



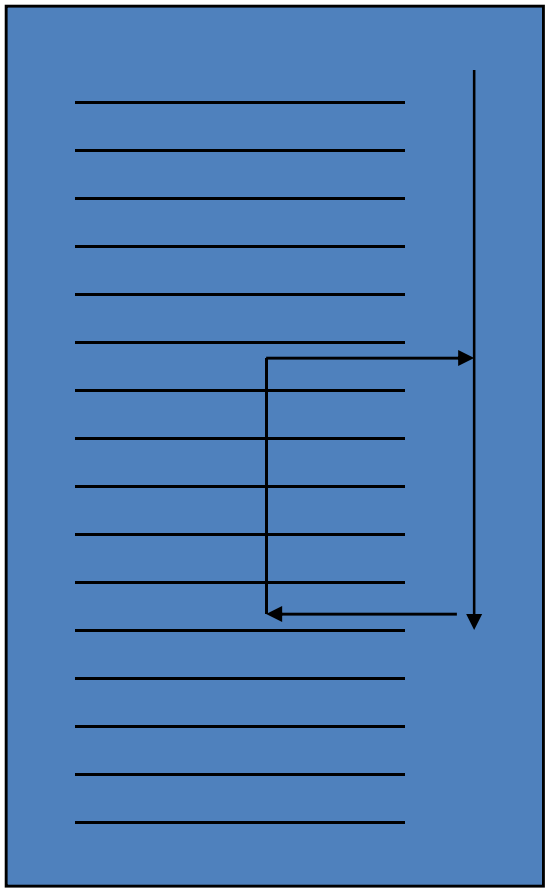
Cấu trúc rẽ nhánh

- if (x < y) {
  printf ("x is smaller");
  }
  else {
  printf ("x is greater")
  }

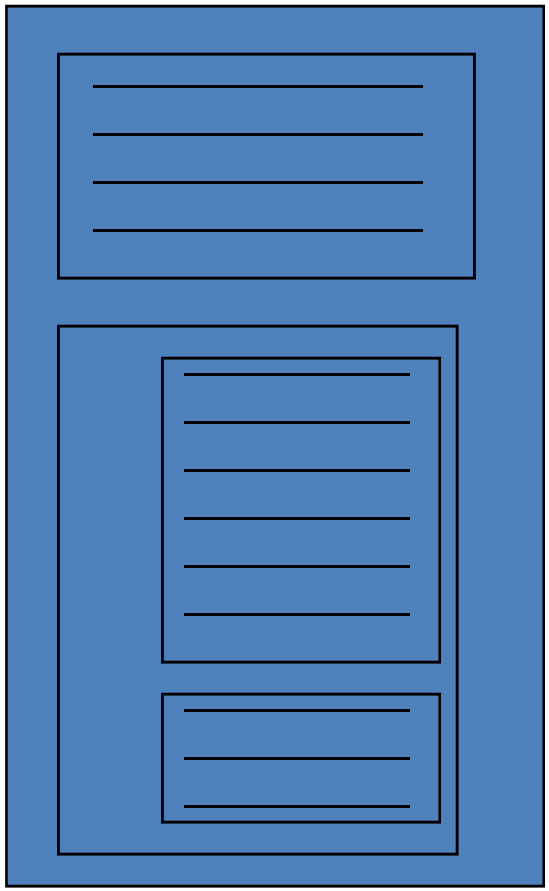


Cấu trúc lặp

- i = 1
  while (i < 5) do {
    printf(i);
    i = i + 1;
  }



Cấu trúc khối



Hàm và chương trình con

- Các phần chương trình nhỏ, có tên và có thể được gọi bởi tên ở các phần khác của chương trình
- Thực hiện một công việc chuyên nhiệm
- Cho phép chương trình được thiết kế thành nhiều thành phần nhỏ
- Có thể định nghĩa biến cục bộ riêng
- Hàm trả về kết quả khi được gọi

Ví dụ

```
int max(int a, int b) {
    if (a < b)
        return a;
    else
        return b;
}
void main() {
    int a;
    a = max(1,2)
}
```

Các thế hệ ngôn ngữ lập trình

- Thế hệ thứ nhất:
  - Xuất hiện vào thập niên 60
  - Tập lệnh gần giống như tập lệnh máy (machine code)
  - Đại diện tiêu biểu: Fortran
- Thế hệ thứ hai
  - Phát triển các cấu trúc dữ liệu từ thế hệ thứ nhất
  - Xuất hiện cấu trúc khối (block structure), các cấu trúc điều khiển (control structures) và các dạng cú pháp linh hoạt hơn
  - Chương trình đã có thể được thiết kế (design)
  - Đại diện tiêu biểu: Algol-60

Các thế hệ ngôn ngữ lập trình (tt)

- Thế hệ thứ ba:
  - Xuất hiện các kiểu dữ liệu do người sử dụng định nghĩa (user-defined data types)
  - Các dạng cấu trúc điều khiển tiếp tục được bổ sung hiệu quả hơn
  - Ngôn ngữ độc lập hơn với kiến trúc máy tính
  - Đại diện tiêu biểu: Pascal

Các thế hệ ngôn ngữ lập trình (tt)

- Thế hệ thứ tư: (Fourth Generation Languages – 4GL)
  - Dễ sử dụng hơn, đặc biệt dành cho những người không phải là chuyên gia
  - Cho phép đưa ra những giải pháp nhanh để xử lý dữ liệu
  - Xúc tích hơn
  - Gần với ngôn ngữ tự nhiên
  - Gần gũi với người sử dụng
  - Không có dạng thủ tục (non-procedural)
  - Đại diện tiêu biểu: Structured Query Language (SQL)

- Thế hệ thứ năm:
  - Các ngôn ngữ được chuyên dụng hoá, độc lập với kiến trúc máy tính, phục vụ các nhu cầu lập trình đặc trưng
  - Hỗ trợ nhiều cấu trúc điều khiển và có các dạng cú pháp tương đối dễ đọc

6.2. Xử lý ngôn ngữ

Định nghĩa của ngôn ngữ máy tính

- Máy tính chỉ có thể hiểu và thực thi được một chương trình khi các lệnh của chương trình được viết một cách tuyệt đối chính xác và rõ ràng về ngữ nghĩa
- Để viết được một chương trình như vậy, ngôn ngữ lập trình cũng phải được định nghĩa theo một hình thức rõ ràng và chính xác
- Ngôn ngữ dùng để định nghĩa ngôn ngữ lập trình là siêu ngôn ngữ (*meta-language*)

Dịch ngôn ngữ máy tính

- Máy tính chỉ có thể hiểu và thực thi được chương trình viết bằng ngôn ngữ máy
- Các ngôn ngữ lập trình giúp người sử dụng dễ thể hiện ý tưởng của mình
- Để máy tính thực hiện được một chương trình viết bằng ngôn ngữ lập trình cấp cao, chương trình đó cần phải được dịch sang ngôn ngữ máy

- Dịch (hoặc xử lý) ngôn ngữ máy tính là chuyển đổi một ngôn ngữ lập trình sang một dạng ngôn ngữ khác (thường là ngôn ngữ máy)
- Hai dạng dịch chính: biên dịch (*compiler*) và thông dịch (*interpreter*)

Biên dịch và thông dịch

- Hai dạng dịch ngôn ngữ cấp cao: biên dịch và thông dịch
- Chương trình biên dịch nhận một chương trình nguồn ở mức cao và tạo ra một chương trình đối tượng tương ứng ở mức thấp. Nếu có lỗi xảy ra trong lúc dịch, quá trình biên dịch sẽ dừng lại
- Chương trình biên dịch thường sử dụng các thư viện thời gian thực thi để hiện thực các tác vụ được mô tả trong chương trình nguồn.

Biên dịch và thông dịch (tt)

- Chương trình thông dịch không tạo ra chương trình đối tượng. Chương trình nguồn sẽ được dịch và chuyển sang mã thực thi theo từng lệnh một
- So sánh biên dịch và thông dịch:
  - Các quá trình xử lý mã nguồn (ví dụ như kiểm tra lỗi) được chương trình biên dịch xử lý trước khi chương trình đối tượng thực thi
  - Nếu một đoạn lệnh lặp lại nhiều lần, chương trình thông dịch sẽ phải dịch lại tất cả đoạn lệnh đó
  - Chương trình thông dịch phải tiếp tục được giữ lại trong bộ nhớ máy tính khi thực thi

Liên kết

- Một chương trình đối tượng thường bao gồm nhiều module. Các module thường có liên quan với nhau qua thao tác gọi và trả về địa chỉ hoặc dùng chung một số dữ liệu
- Chương trình đối tượng cũng có thể dùng đến một số module ngoài trong các thư viện thời gian thực thi
- Chương trình liên kết sẽ tạo các liên kết cần thiết đến các lời gọi chương trình và truy xuất dữ liệu từ các module ngoài

Các cơ chế liên kết

- ✓ Liên kết tĩnh (*static link '.lib'*) là hoạt động liên kết xảy ra tại thời điểm dịch, trước khi chương trình chạy, tất cả các vị trí chứa thông tin chưa hoàn chỉnh đều phải được hiệu chỉnh lại.
- ✓ Liên kết động (*dynamic link '.dll'*) là hoạt động liên kết xảy ra tại thời điểm chạy chương trình, cụ thể tại lần đầu tiên chạy lệnh chứa thông tin chưa hoàn chỉnh (hay mỗi lần chạy lại).

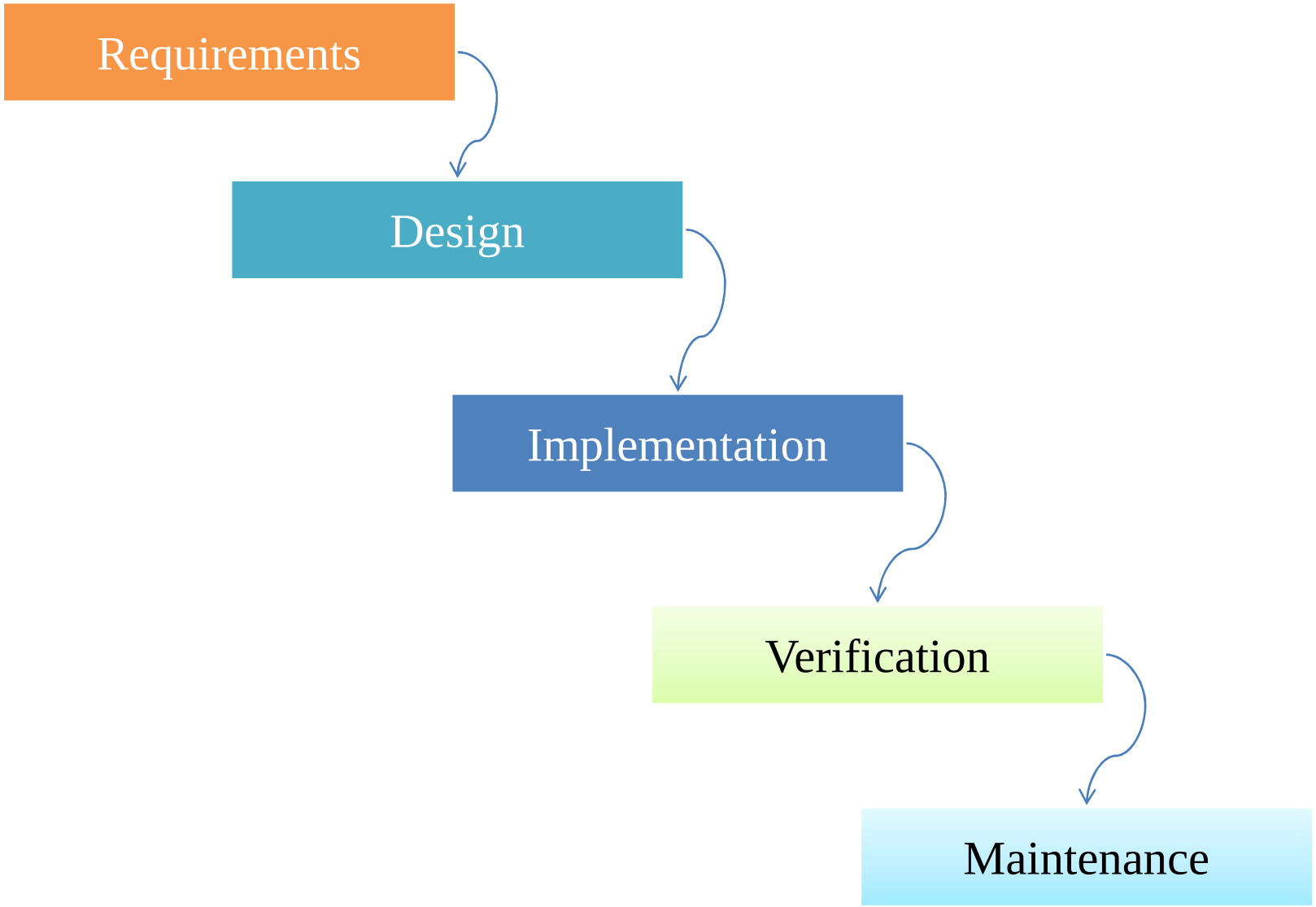
➢ Liên kết động có nhiều ưu điểm hơn liên kết tĩnh và hầu hết các hệ thống hiện nay (Windows, Linux) đều sử dụng chủ yếu cơ chế liên kết động.

6.3. Thiết kế ứng dụng

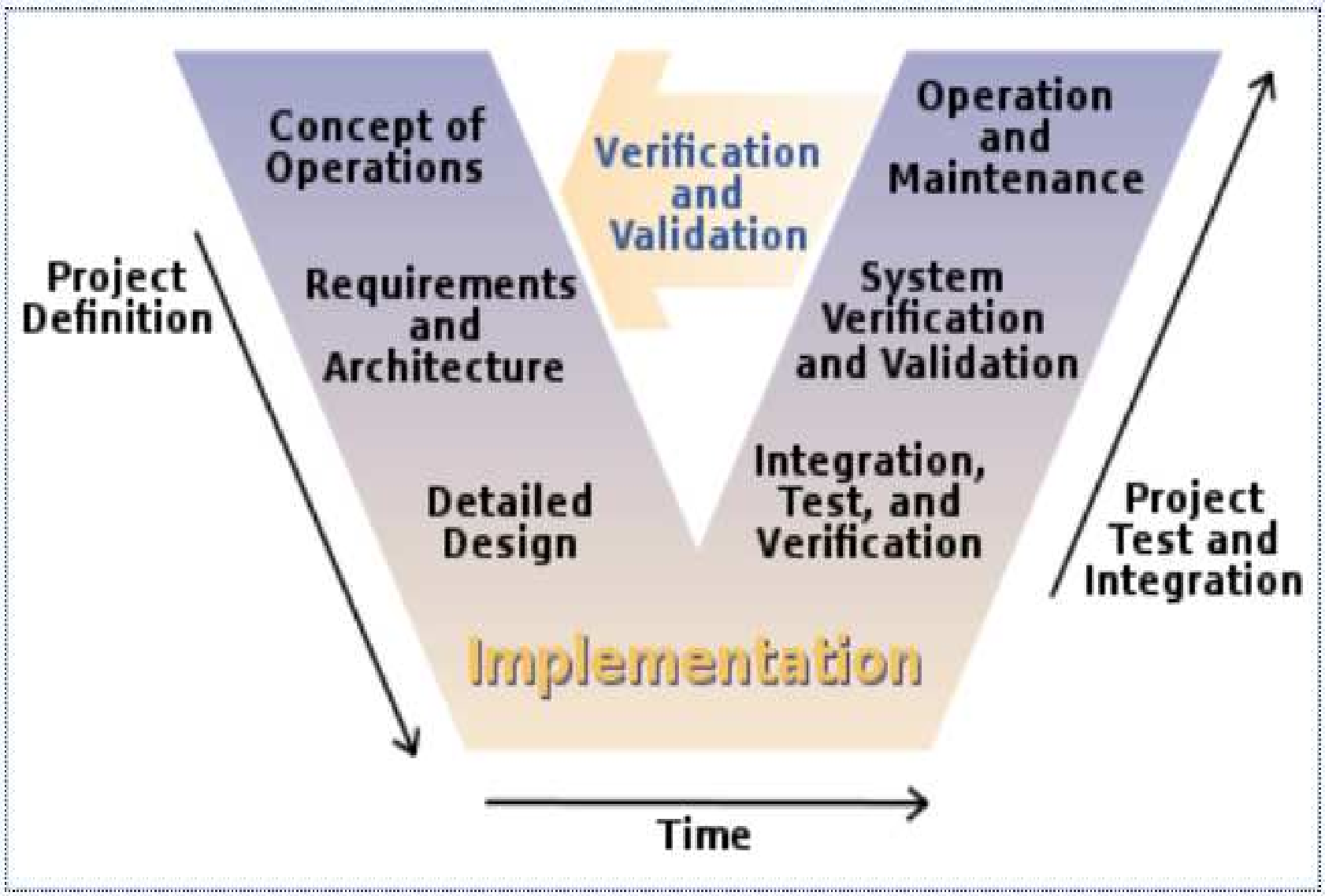
- Phần mềm phục vụ nhu cầu cho người dùng hiện nay khá phức tạp, khá lớn

  => không thể viết mã nguồn chương trình ngay khi được đặt hàng
- Từ bài toán cần giải quyết đến khi có được chương trình giải quyết bài toán đó, người ta phải thực hiện nhiều công việc khác nhau
- Thuật ngữ "qui trình phát triển phần mềm" (*Software Development Process*) để miêu tả cụ thể, chi tiết trình tự các công việc cần phải thực hiện để xây dựng được chương trình từ bài toán cần giải quyết.

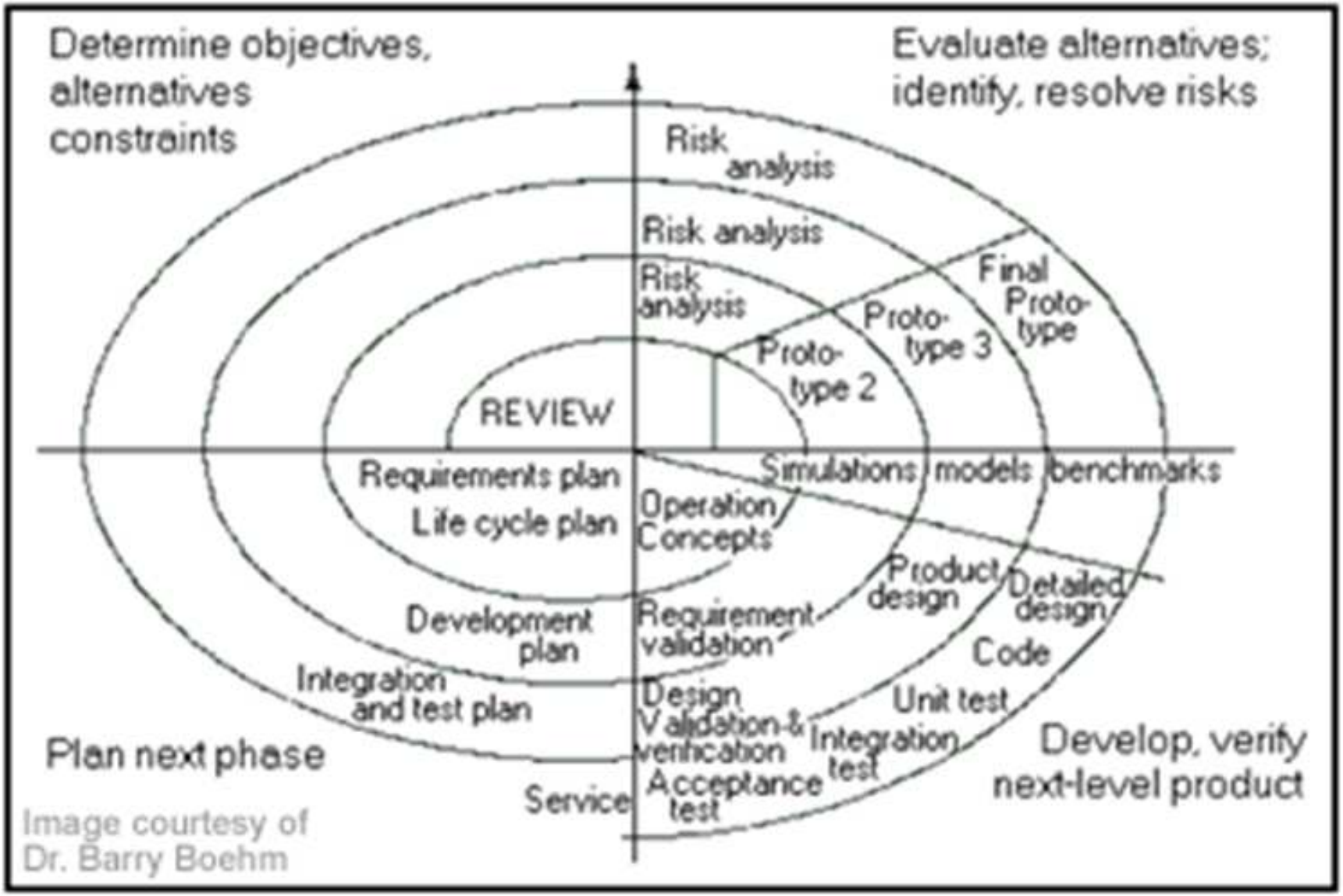
Waterfall model



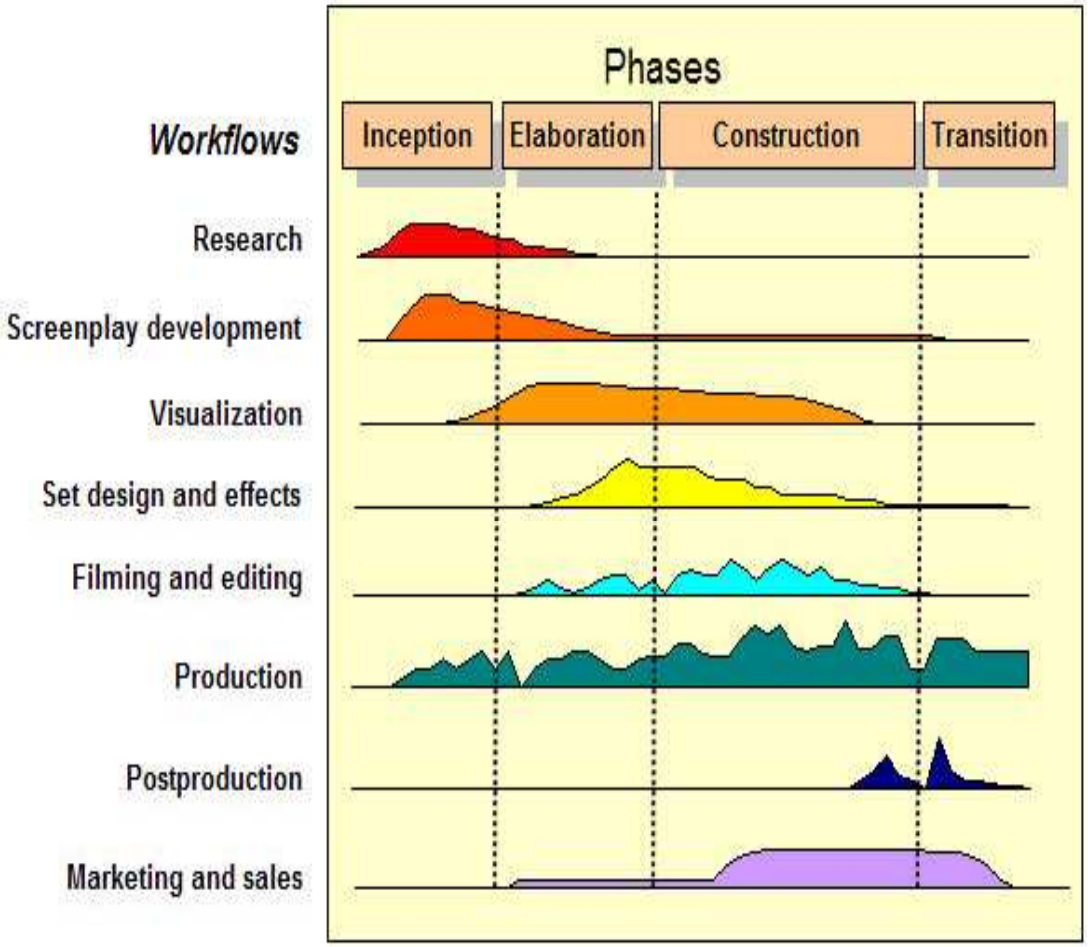
V-model


Concept of Operations
Verification and Validation
Operation and Maintenance
Project Definition
Requirements and Architecture
System Verification and Validation
Detailed Design
Integration, Test, and Verification
Project Test and Integration
Implementation
Time

Spiral model



Unified Software Development Process



A movie project plan, structured similarly to a software development project

The Unified Software Development Process

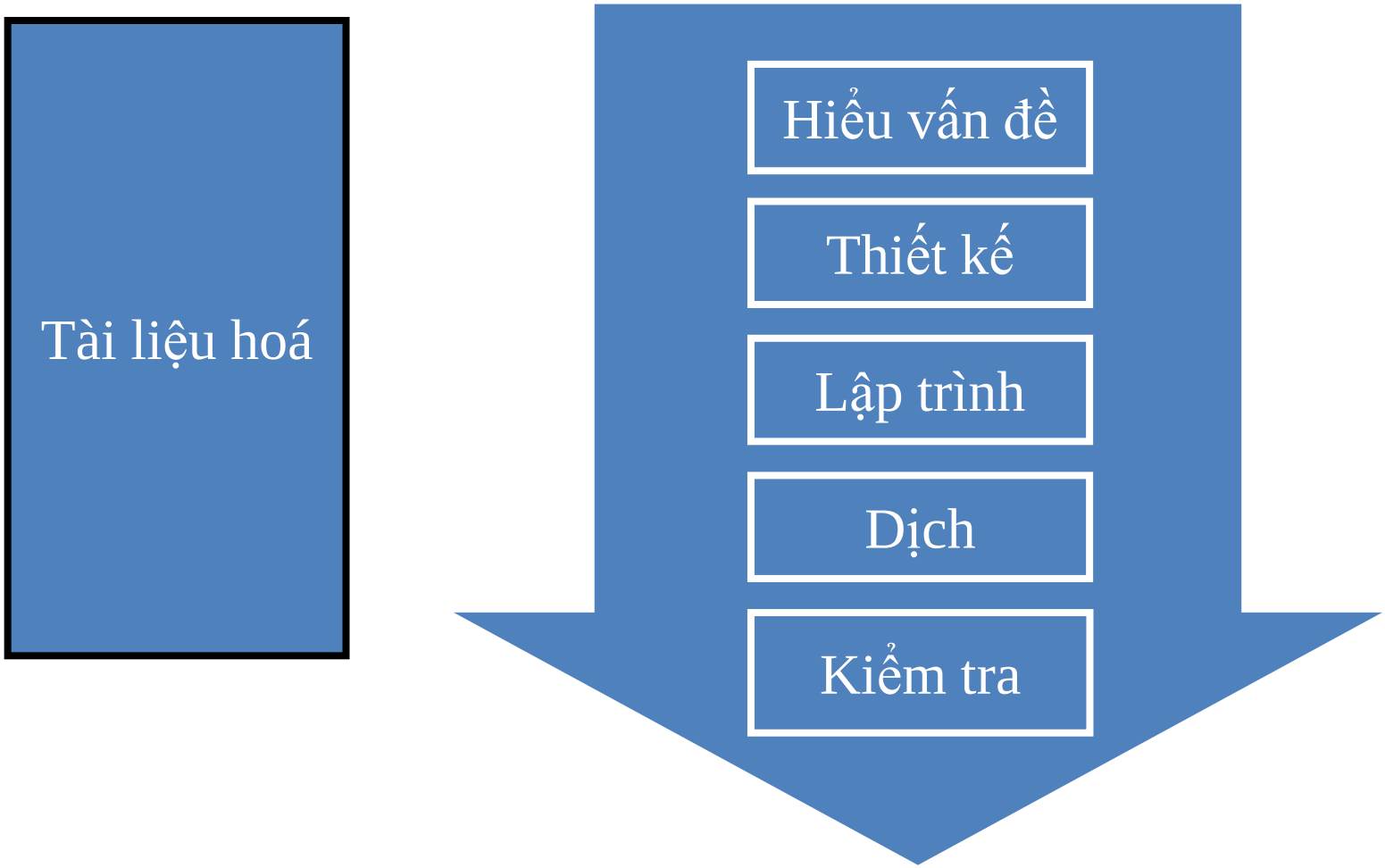
- Use-Case Model
  - Use-Case Diagram
- Analysis Model
  - describe "Realization of a Use-Case" by a Collaboration Diagram and a Flow of Event Description
- Design Model
  - Class Diagram , Sequence Diagram, and Statechart Diagram
- Deployment Model
  - Deployment Diagram
- Implementation Model
  - Component Diagram
- Test Model
  - Test Case

Unified Software Development Process

- Để xây dựng 1 chương trình, qui trình phát triển phần mềm hợp nhất (*Unified Software Development Process*) sẽ xác định rõ ràng các thông tin sau :
  - bao nhiêu loại người (*role*) sẽ tham gia thực hiện, thí dụ như kiến trúc sư phần mềm, phân tích viên, kỹ sư thiết kế, lập trình viên, kiểm lỗi viên,...
  - mỗi loại người sẽ phải thực hiện các công việc gì cụ thể, thí dụ lập trình viên A phải viết code cho bao nhiêu hàm, các hàm đó cụ thể là gì ?
  - mỗi công việc sẽ được thực hiện khi nào ?
  - mỗi công việc sẽ được thực hiện bằng cách nào ?
  - kết quả mỗi công việc sẽ được miêu tả theo định dạng nào,
  - bằng ngôn ngữ miêu tả nào ?

Luồng công việc (workflow)

- Thường để phát triển 1 chương trình, ta cần thực hiện các luồng công việc chức năng sau đây



Hiểu vấn đề

- Xây dựng một ngữ cảnh chính xác cho vấn đề:
  - Xác định giả định nào có thể dùng và giả định nào không thể
- Một số bài toán giải được nhưng khi phân tích dưới quan điểm thiết kế chương trình có thể sẽ trở nên phức tạp

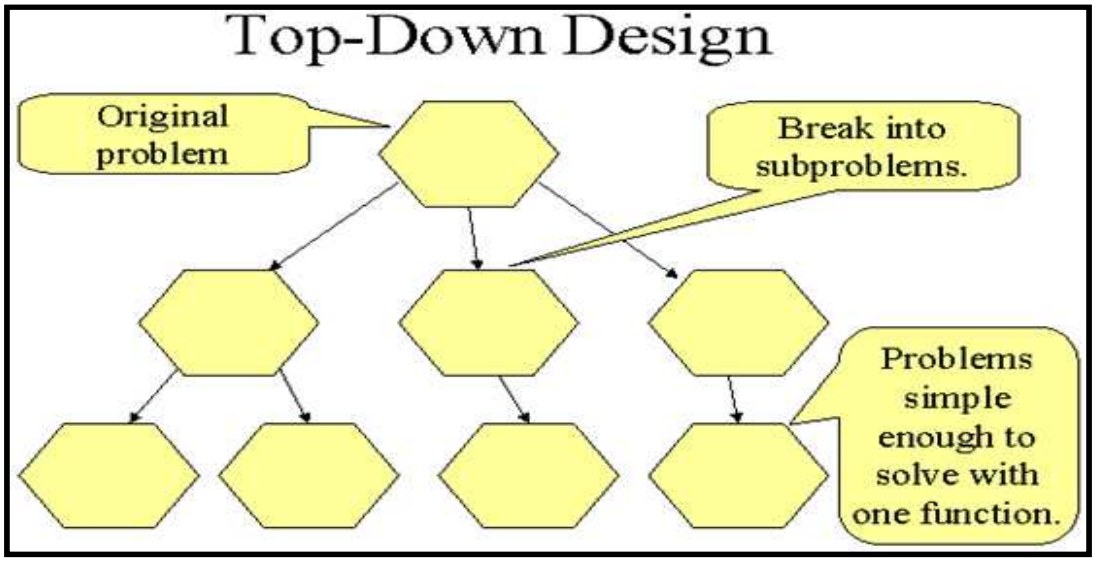
Ví dụ

- Viết một chương trình đọc vào hai số và in ra thương của hai số đó
  - Sử dụng định dạng nào cho thông tin nhập: hai số nhập vào là hai số nguyên, số hữu tỷ hay số thực. Nếu là số thực thì độ chính xác thập phân chấp nhận được là bao nhiêu
  - Chương trình sẽ xử lý như thế nào khi số chia nhập vào là 0

Thiết kế chương trình

- Yêu cầu đối với một chương trình máy tính: không xảy ra lỗi vào bất kỳ lúc nào trong thời gian thực thi
  - Kết quả xuất bị lỗi
  - Bị treo
  - Xử lý dữ liệu bất thường
- Thiết kế chương trình tốt giúp giảm thiểu tối đa lỗi chương trình

- Thiết kế chương trình có cấu trúc
  - Phân tích từ trên xuống (*top-down*)
  - Tinh chế từng bước (*stepwise refinement*)

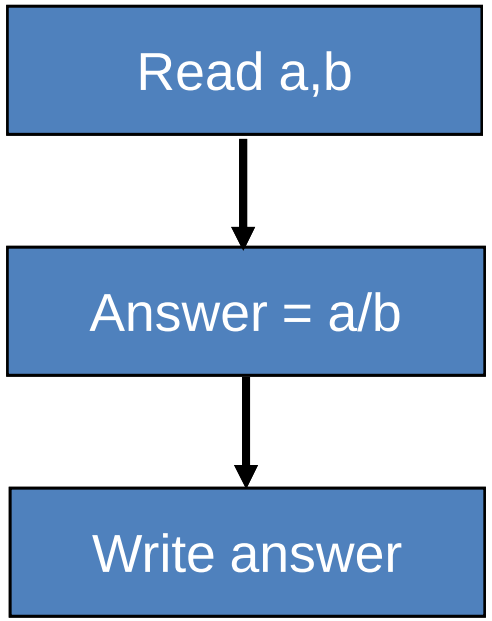


Thiết kế lời giải

- Phân tích từ trên xuống, tinh chế từng bước:
  - Lời giải tổng quát được đưa ra trước, bao gồm các module chính của chương trình
  - Mỗi module được phân tích thành nhiều module nhỏ hơn

Sử dụng biểu đồ dòng chảy và mã giả

- Mã giả (*pseudo code*):
  - read a,b
  - Answer = a/b
  - Write answer



Lập trình cấu trúc (Structured Programming)

- Là phương pháp thiết kế chương trình phổ biến hiện nay, bao gồm các nguyên lý cơ bản sau:
  - Sử dụng các cấu trúc điều khiển hạn chế trong các dạng sau: tuần tự, lựa chọn và lặp
  - Module hoá chương trình
  - Phân tích từ trên xuống, phân rã từng bước
  - Trình bày chương trình theo một định dạng rõ ràng
  - Sử dụng chú thích
  - Ưu tiên lựa chọn giải pháp đơn giản

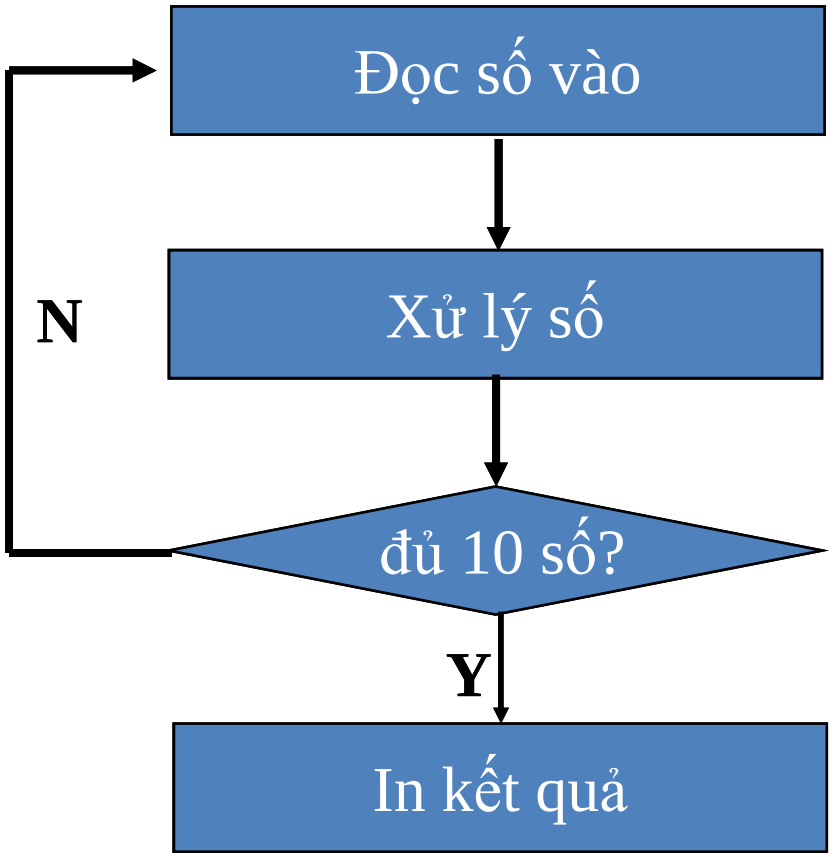
Case Study

- Hiểu vấn đề: **Đọc** mười số nguyên dương hoặc âm từ bàn phím và **tính tổng** các số dương và âm riêng biệt. **In** ra hai tổng tính được.

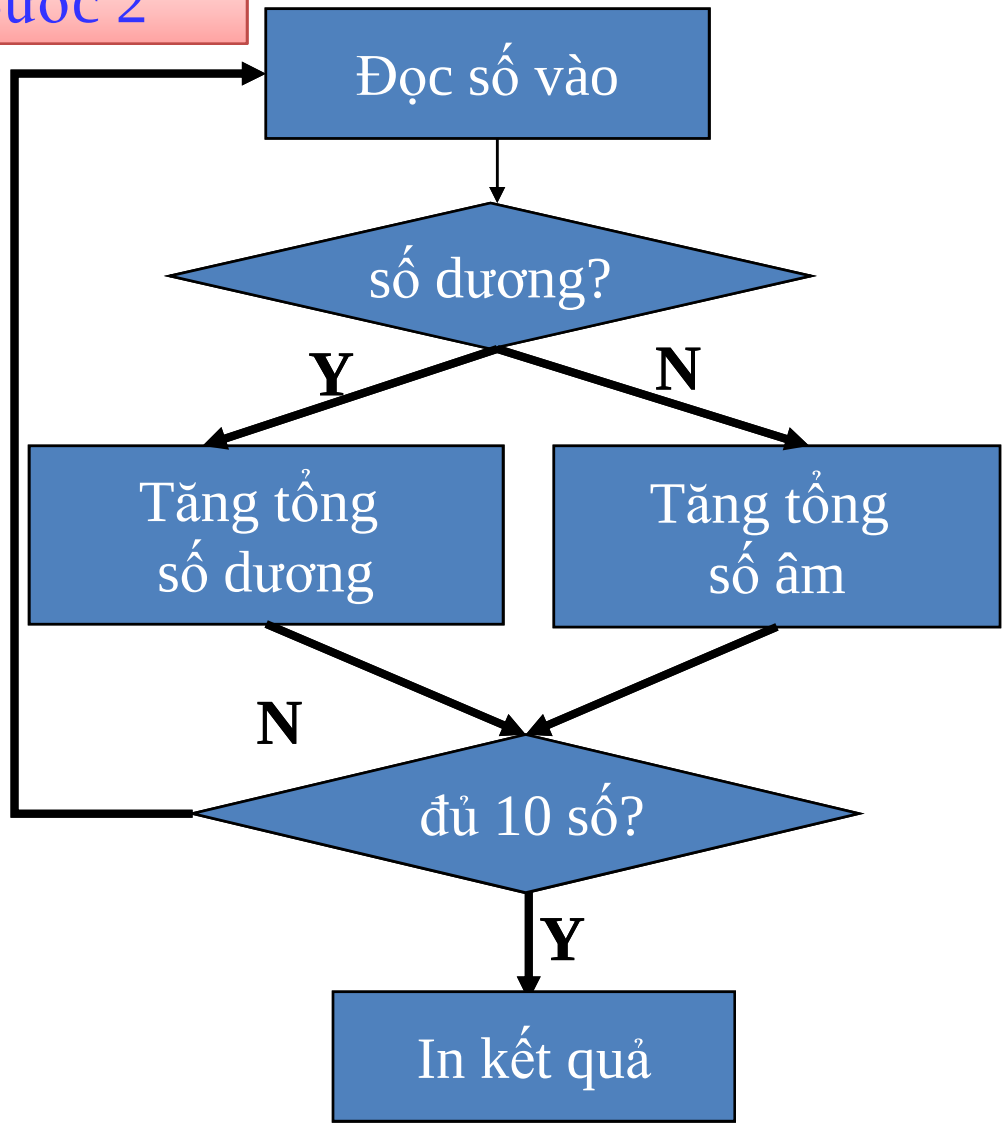
- Chương trình có thể phân rã thành các công việc chính như sau:
  - Đọc các số vào
  - Xử lý các số đọc vào (tính tổng số dương và số âm)
  - In kết quả

Tinh chế từng bước

Bước 1



Bước 2



Kiểm tra lời giải

- Sử dụng dữ liệu kiểm tra (*test data*) để kiểm tra kết quả xuất của chương trình và so sánh với lời giải mong đợi
- Khi bắt gặp lỗi, kiểm tra thiết kế và chỉnh sửa chương trình trước khi tiếp tục kiểm tra
- Nhiệm vụ của workflow là kiểm tra và thử nghiệm chương trình thực thi xem nó có lỗi không, nếu có thì lỗi cụ thể nằm ở lệnh nào, tại sao bị lỗi và sữa lỗi.
  - lặp kiểm thử từng hàm chức năng theo 1 thứ tự xác định.

Kiểm thử

- kiểm thử hộp đen (*black-box testing*) : kiểm thử thành phần theo góc nhìn từ ngoài xem hành vi của thành phần có thỏa mãn đặc tả sử dụng không ? Thí dụ ta thử gọi hàm cos(0) xem hàm có trả về 1 không, nếu hàm trả về giá trị khác 1, ta nói hàm cos bị lỗi.
- kiểm thử hộp trắng (*white-box testing)* : kiểm thử thành phần theo góc nhìn bên trong xem từng lệnh của thành phần có chạy đúng theo giải thuật thiết kế không ? Thường khi kiểm tra hộp đen 1 thành phần nào đó bị lỗi thì ta mới tiến hành kiểm thử hộp trắng để xác định chính xác các lệnh gây lỗi trong thành phần đó.

6.5. Tài liệu hoá chương trình

- Nhằm mục đích cung cấp cho người sử dụng tất cả thông tin cần thiết để hoàn toàn hiểu được mục đích của chương trình và cách xây dựng chương trình
- Một tài liệu của chương trình thường bao gồm các phần sau:
  - Loại chương trình
  - Người sẽ sử dụng chương trình
  - Xác định xem có cần thiết thay đổi mã nguồn của chương trình sau khi đã được kiểm tra và chấp nhận lần cuối hay không

Các yêu cầu với tài liệu chương trình

- Tài liệu dành cho lập trình viên: đặc tả thiết kế chương trình. Thông tin sơ lược về cách chương trình giao tiếp với người sử dụng
- Tài liệu dành cho người sử dụng: mô tả chi tiết các tính năng của chương trình và thường kèm theo các bản hướng dẫn mẫu (tutorial)
- Tài liệu dành cho nhà quản lý: mô tả tổng quan chương trình, chẳng hạn các tính năng, khả năng của chương trình, yêu cầu phần cứng v.v…

Các đề mục trong tài liệu chương trình

- Một tài liệu chương trình thường bao gồm các phần sau
  - Định danh (identification)
  - Đặc tả chung
  - Thông tin người dùng
  - Đặc tả chương trình

Định danh cho chương trình

- Tên chương trình
- Mô tả ngắn về các tính năng của chương trình
- Tác giả
- Ngày viết chương trình
- Ngôn ngữ sử dụng và phiên bản
- Yêu cầu phần cứng

Đặc tả chung

- Mô tả các hoạt động chính của chương trình trong điều kiện hoạt động bình thường
- Các file đặc tả
- Hạn chế và/ hoặc giới hạn của chương trình
- Các công thức, tài liệu tham khảo hoặc văn bản mô tả những chương trình con hoặc kỹ thuật phức tạp được sử dụng

Đặc tả chương trình

- Lược đồ cấu trúc, lược đồ dòng chảy (flowchart), các bảng quyết định
- Các bảng mã giả có chú thích
- Quá trình kiểm tra bao gồm các lớp dữ liệu kiểm tra cùng với kết quả mong đợi và nhật ký kiểm tra

Hướng dẫn sử dụng

- Chỉ dẫn cài đặt
- Giải thích chi tiết về hoạt động của chương trình
- Các bài hướng dẫn mẫu

The Ten Commandments of Computer Ethics

1. THOU SHALT NOT USE A COMPUTER TO HARM OTHER PEOPLE.
2. THOU SHALT NOT INTERFERE WITH OTHER PEOPLE'S COMPUTER WORK.
3. THOU SHALT NOT SNOOP AROUND IN OTHER PEOPLE'S COMPUTER FILES.
4. THOU SHALT NOT USE A COMPUTER TO STEAL.
5. THOU SHALT NOT USE A COMPUTER TO BEAR FALSE WITNESS.
6. THOU SHALT NOT COPY OR USE PROPRIETARY SOFTWARE FOR WHICH YOU HAVE NOT PAID.
7. THOU SHALT NOT USE OTHER PEOPLE'S COMPUTER RESOURCES WITHOUT AUTHORIZATION OR PROPER COMPENSATION.
8. THOU SHALT NOT APPROPRIATE OTHER PEOPLE'S INTELLECTUAL OUTPUT.
9. THOU SHALT THINK ABOUT THE SOCIAL CONSEQUENCES OF THE PROGRAM YOU ARE WRITING OR THE SYSTEM YOU ARE DESIGNING.
10. THOU SHALT ALWAYS USE A COMPUTER IN WAYS THAT INSURE CONSIDERATION AND RESPECT FOR YOUR FELLOW HUMANS.

Written by the The Computer Ethics Institute