







PHÁT TRIỂN VẬN HÀNH BẢO TRÌ PHẦN MỀM

ThS. NGUYỄN THỊ THANH TRÚC

Nội dung (Chương 4)

- 
-  **HIỂU CHƯƠNG TRÌNH**
 -  **NGƯỜI BẢO TRÌ VÀ CÁC NHU CẦU THÔNG TIN**
 -  **MÔ HÌNH QUI TRÌNH NẮM BẮT THÔNG TIN**
 -  **REVERSE ENGINEERING**
 -  **Thảo luận và làm bài tập**

Chương 4: ***CÁC TÁC VỤ YÊU CẦU BẢO TRÌ***

4.1 HIỂU CHƯƠNG TRÌNH

4.2 NGƯỜI BẢO TRÌ VÀ CÁC NHU CẦU THÔNG TIN

4.3 MÔ HÌNH QUI TRÌNH NẮM BẮT THÔNG TIN

4.4 REVERSE ENGINEERING

Chương 4: CÁC TÁC VỤ YÊU CẦU BẢO TRÌ

1. HIỂU CHƯƠNG TRÌNH

- Mục tiêu của nắm bắt chương trình
 - ✓ Phạm vi vấn đề
 - ✓ Hiệu quả thực thi
 - ✓ Mối liên hệ Nhân – Quả (Cause-Effect)
 - ✓ Mối liên hệ sản phẩm – Môi trường
 - ✓ Đặc trưng Quyết định – Hỗ trợ

2. NGƯỜI BẢO TRÌ VÀ CÁC NHU CẦU THÔNG TIN

- Managers
- Analysts
- Designers
- Programmers

3. MÔ HÌNH QUI TRÌNH NẮM BẮT THÔNG TIN

- Chiến lược nắm bắt chương trình
 - ✓ Top-Down Model III
 - ✓ Bottom-Up / Chunking Model
 - ✓ Opportunistic Model

4. REVERSE ENGINEERING

- Định nghĩa
- Mục đích và mục tiêu của reverse engineering
- Các mức của reverse engineering
- Kỹ thuật hỗ trợ
- Các lợi điểm

4.1 HIỂU CHƯƠNG TRÌNH

- ❑ Mục tiêu của năm bắt chương trình
 - Phạm vi vấn đề
 - Hiệu quả thực thi
 - Mối liên hệ Nhân – Quả (Cause-Effect)
 - Mối liên hệ sản phẩm – Môi trường
 - Đặc trưng Quyết định – Hỗ trợ

Phạm vi vấn đề

- ❑ **Nắm bắt được kiến thức phạm vi khá quan trọng. Tác động đến vấn đề vùng phạm vi chuyên biệt, cụ thể.**
 - Vd: môi trường điều trị bệnh nhân ...
- ❑ **Trong hệ thống lớn**
 - Ví dụ chăm sóc sức khỏe, viễn thông, tài chính được phân nhỏ thành vấn đề nhỏ, thành phần nhỏ hơn, được quản lý thành đơn vị chương trình như mô đun, thủ tục, hàm.
 - Ví dụ Trình biên dịch bao gồm thành phần parser, phân tích, phát sinh code, mỗi thành phần được phân rã thành phần nhỏ hơn .
- ❑ **Tác động đến sự thay đổi, ước tính nguồn tài nguyên đòi hỏi cho tác vụ bảo trì, kiến thức phạm vi vấn đề nói chung và vấn đề nhỏ cụ thể là cần thiết tác động trực tiếp nhân sự bảo trì trong việc chọn lựa thuật toán phù hợp, phương pháp luận, và công cụ.**
- ❑ **Việc chọn lựa nhân sự với mức độ chuyên gia và kỹ năng phù hợp là khía cạnh khác. Thông tin bao gồm từ nguồn khác nhau – tài liệu hệ thống, end-users, và chương trình nguồn**

Hiệu quả thực thi

- ❑ Ở mức cao trừu tượng, nhân sự bảo trì cần phải nắm (dự đoán) kết quả chương trình sẽ được phát sinh kết quả gì từ **đầu vào** được cho mà không cần biết đơn vị chương trình được xây dựng để có **kết quả tổng thể** và kết quả được cho như thế nào.
- ❑ Ở mức thấp, họ cần biết kết quả mỗi đơn vị chương trình sẽ được tạo và thực thi.
- ❑ Kiến thức data flow, control flow, và thuật toán có thể thuận tiện hoàn thành thực thi mục tiêu này.
- ❑ **Ví dụ** người lập trình muốn biết ở mức trừu tượng, đầu ra của qui trình hoàn tất biên dịch và ở mức thấp, đầu ra từ parser. Trong khi, thông tin này sẽ giúp cho người bảo trì xác định những thay đổi đã thực thi có đạt hiệu quả như mong đợi hay không

Mối liên hệ Cause-Effect

Trong chương trình lớn và phức tạp, kiến thức của mối liên hệ này là quan trọng:

- ❑ Cho phép nhân sự bảo trì đưa ra lý do làm thế nào thành phần của sản phẩm phần mềm **tương tác** trong khi thực thi.
- ❑ Cho phép người lập trình dự đoán phạm vi một thay đổi và bất kỳ **hệ quả** phát sinh từ thay đổi.
- ❑ Mối liên hệ cause-effect có thể được sử dụng để **lưu vết** luồng thông tin qua chương trình. Tại điểm mà nơi có những sự gián đoạn **bất thường** của luồng thông tin này mang dấu hiệu nguồn **phát sinh bug** chương trình

Ví dụ: A string reversing program

```
MODULE StringReversing:
FROM InOut IMPORT WriteString, Write, Read, EOL, WriteLn;
FROM StacksLibrary IMPORT StackType, Create, IsEmpty,
    Pop, Push;
VAR
    Stack: StackType;
    Char, Response: CHAR;
BEGIN
    REPEAT
        Create (Stack);
        WriteString ("Enter string to be reversed");
        WriteLn;
```

Ví dụ: A string reversing program (tt)

```
Read (Char);                                <--Segment A
WHILE Char # EOL DO 4
    Push (Stack, Char);
    Read (Char);
END (* While *)
WriteLn;
WriteString ("Reversed string is: ");
WHILE NOT IsEmpty (Stack) DO <---Segment B
    Pop (Stack, Char); 4
    Write (Char);
END (* While *)
WriteLn;
WriteString ("Process another string (Y or N)? ");
Read (Response);
UNTIL CAP (Response) # 'Y'
END StringReversing.
```

Mối liên hệ sản phẩm và môi trường

- ❑ Sản phẩm là hệ thống phần mềm. Môi trường là toàn bộ tất cả điều kiện và ảnh hưởng mà hành động từ bên ngoài sản phẩm.
 - Ví dụ: qui định nghiệp vụ, qui định chính phủ, mẫu công việc, platform điều hành của phần mềm và phần cứng
- ❑ Nó cần thiết cho nhân sự bảo trì để biết không chỉ mở rộng mối liên hệ.
- ❑ Kiến thức này dùng để dự đoán những thay đổi trong những thành phần này sẽ tác động như thế nào với sản phẩm nói chung và dưới chương trình cụ thể nói riêng

Đặc trưng Quyết định – Hỗ trợ

- ❑ Thuộc tính của sản phẩm phần mềm như độ phức tạp và khả năng dễ bảo trì là
 - ví dụ hướng dẫn trong kỹ thuật và qui trình ra quyết định như phân tích, ra quyết định ngân sách, cấp phát nhân lực.
- ❑ Đo độ phức tạp của hệ thống xác định thành phần hệ thống đòi hỏi nhiều tài nguyên cho kiểm thử
- ❑ Reverse engineering được dùng để nghiên cứu hiểu để trích chọn các loại thông tin.
- ❑ Có nhiều yếu tố tác động mở rộng mà nhân sự bảo trì có thể yêu cầu danh mục kiến thức đã nêu về hệ thống.
 - Bao gồm chiến lược nắm bắt thông tin, sự thông thạo phạm vi, chất lượng sưu liệu, báo cáo thuyết trình và tổ chức, thực nghiệm chương trình, và vấn đề thực thi, công cụ hỗ trợ

Một tiếp cận phân tích chương trình mới dựa trên trên thông tin hỗ trợ

- ❑ Theo kinh nghiệm phát triển, với các hệ thống lớn được sự hỗ trợ bởi luồng hệ thống, sơ đồ cấu trúc mô đun, luồng dữ liệu và tham chiếu, ngoài công cụ tự động còn cần bởi bảo trì bằng tay và kỹ năng của người bảo trì:
 - Nắm bắt **thủ tục**, **biến toàn cục**, mối liên hệ lỗi và mở rộng chức năng cục bộ, mô đun
 - **Độ phức tạp** của chương trình
 - Ngữ nghĩa các **vòng lặp**, mối liên hệ **input/output**
 - **Vấn đề gì** là quan trọng khi phân tích chương trình
 - ➔ **What-How-Why** cho một đối tượng (object) của chương trình cho việc hiểu chương trình

(WHAT) Đối tượng trong chương trình là gì?

- ❑ Trước khi cố gắng phân tích được chương trình theo tiếp cận **W-H-W** (What, How, Why), chúng ta nên suy nghĩa các **đối tượng** (object):
 - Lớp dữ liệu, cấu trúc, bảng, cờ, chuỗi và các biến thể hiện kiến thức phạm vi
 - Tên của các chức năng hay chu trình (routines) hay qui trình cho chúng ta gắn kết với chức năng của chúng
 - Thành phần liên quan được cung cấp bởi thư viện và môi trường chương trình
- ❑ Rõ ràng **WHW**(**W**hat,**H**ow,**W**hy) sẽ giúp người bảo trì phần mềm hiểu một cách hiệu quả. Hiển nhiên mô tả hình thức là khó trong khi công cụ tự động là không thể. Hiểu chương trình sẽ thực hiện tiếp diễn cùng với kiến thức phạm vi tốt của người bảo trì

Ví dụ 1

```
static void print_url(String spec) {  
    try {  
        System.out.println(spec);  
        URL url = new URL(spec);  
        String proto = url.getProtocol();  
        String host = url.getHost();  
        String file = url.getFile();  
        String ref = url.getRef();  
        System.out.println(' ',  
            proto+' '+proto+' ',host+' '+host+' ',  
            file+' '+file+' ',ref+' '+ref);  
    } catch (Exception e) {  
        System.out.println(e);  
    }  
}
```

Ví dụ 1 *this following is a php program array01.php3*

```
/*--Create an associate array based on data of file */
$fname = "report" ;
$MAXLN = 256 ;
/*--- Check file -----*/
if (file_exists($fname)) {
    $fd = fopen($fname,"r");
} else {
    print "Cann't open file $fname";
    return;
}
$i = 0;
$assAry = "";
while ($buf=fgets($fd,$MAXLN)) {
    $element = explode("\t",$buf);
    $esize = count($element);
    $valstr[] = "" ;
    $j = 0 ;
    while ($j < $esize -1 ) {
        $valstr[$j++] = $element[$j];
    }
    $assAry[$element[0]] = $valstr ;
}
fclose($fd);
while (list($key,$vall)=each($assAry)){
    echo "-----";
    echo "$key => $vall[0]<br> ";
}
}
```

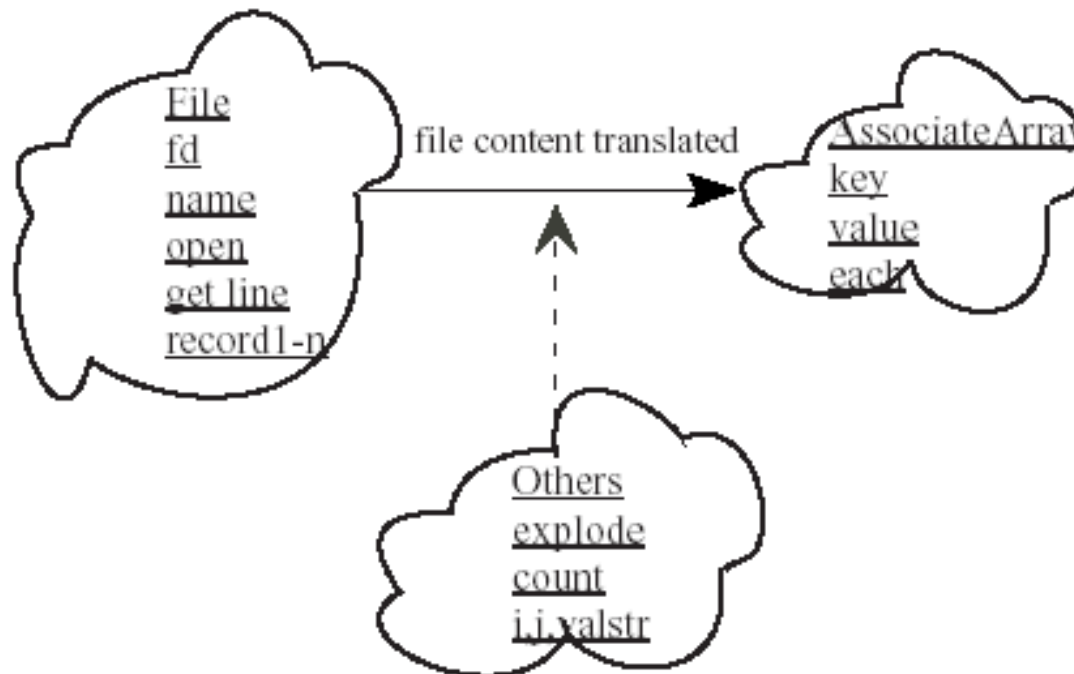
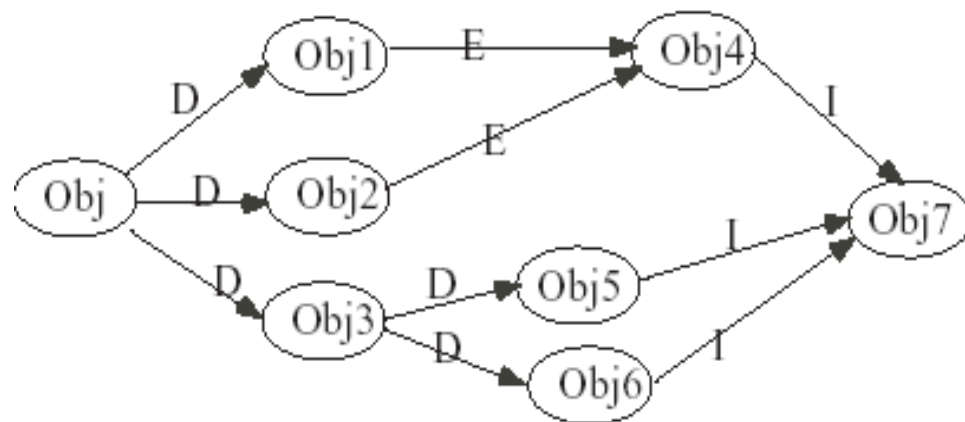



Figure 1: Program Layout



D:Decomposition
E:Evolution
I: Integration

Figure 2: Object View

Table 1: Object Element List

FUNCTIONS	VARIABLES	CONSTANT
array01.php3 file_exists fopen, fgets explode,count each list	fname fd,buf element,esize key,vall, valstr i,j,assAry	MAXLN(256) TAB

Các bước thực hiện: Thuật toán chung(GA)

- ❑ **GA1.** Liệt kê tất cả các đối tượng với mức khác nhau
- ❑ **GA2.** Sắp xếp đối tượng quan trọng dựa trên tài liệu khác nhau và kiến thức của người bảo trì .
 - Câu Trả lời: đối tượng là gì (WHAT)
- ❑ **GA3.** Nếu đối tượng được chọn làm bùng nổ đối tượng khác thì được bỏ vào danh sách.
 - Câu trả lời: mối liên hệ Chuồng bồ câu của các đối tượng được chọn
- ❑ **GA4.** Nếu có mối liên hệ từ và đến đối tượng khác, tạo mối liên hệ đến đối tượng mới sau đó lặp lại từ GA1

Bài tập đọc thêm

❑ **Làm thế nào để đọc - hiểu một project có hàng nghìn dòng**? Nội dung sau chính xác ko? Đề xuất theo kinh nghiệm riêng của bạn?

1. Kinh nghiệm: Viết nhiều code, sử dụng nhiều loại phần mềm
2. Kiến thức nền tảng và phải phong phú (sẽ là điểm chính)
3. Hiểu được 'chuẩn viết code của thế giới'
4. Khả năng về tiếng anh

❑ **Đọc thêm tài liệu tham khảo trên course**

- Kinh nghiệm đọc code của lập trình viên
- Khi bạn đọc hiểu code chính là lúc bạn đang rewriting code
- Bảo mật nhập môn (bảo mật cơ bản cho developer)
- Code dạo ký sự

Kết luận tiếp cận

1. Hiểu chương trình là một qui trình khó liên quan đến lập trình viên và người bảo trì, có kiến thức chương trình từ góc nhìn khác nhau
2. Công cụ nên được cung cấp nhiều có thể để hỗ trợ khám phá thông tin cho việc hiểu chương trình. Tự động hoá thì thích hợp hơn nhưng tự động hoàn toàn thì hiển nhiên không khả thi
3. Phương pháp phân tích chương trình sẽ đạt được đối với chương trình và kiểu ứng dụng
4. Restructuring hay refactoring chương trình có thể là tác vụ quan trọng trong qui trình bảo trì

Bài tập:

- Tìm hiểu các kỹ thuật refactoring
- Chọn chương trình nguồn mở (open source code) từ sourceforge.net đọc hiểu và phân tích chương trình (modules, functions,...) nêu khái quát hỗ trợ của nó.

4.2 NGƯỜI BẢO TRÌ VÀ CÁC NHU CẦU THÔNG TIN

- ☐ Managers
- ☐ Analysts
- ☐ Designers
- ☐ Programmers

Managers

- ❑ Trách nhiệm quản lý là thực hiện ra quyết định, kiến thức hỗ trợ quyết định trong thực hiện quyết định
- ❑ Mức độ hiểu biết đòi hỏi sẽ phụ thuộc vào quyết định thực thi
- ❑ Ước tính chi phí, thời gian cải thiện chính, kiến thức của độ lớn chương trình (thuật ngữ line of code, điểm chức năng (function point)
- ❑ Ước tính này để xác định xem có kinh tế để thay thế hệ thống cho khách hàng
- ❑ Không cần biết kiến trúc thiết kế của hệ thống hay thực thi chương trình ở mức thấp chi tiết để thực thi nhiệm vụ công việc của người quản lý
- ❑ *"Nobody can seriously have believed that executives could read programs"* Weinberg

Analysts

- ❑ Hiểu **phạm vi** vấn đề (vd tài chính hay health care) để chịu trách nhiệm xác định yêu cầu chức năng và phi chức năng, và thiết lập mối liên hệ giữa hệ thống và môi trường.
- ❑ Trong suốt bảo trì, xem xét môi trường thay đổi thế nào (ví dụ định, hệ thống điều hành mới).
- ❑ Như vậy, trước khi thực hiện thay đổi, người phân tích cần có cái nhìn **tổng thể hệ thống**, bức tranh tổng thể tương tác giữa các đơn vị chức năng chính.
- ❑ Xác định mối gắn kết thay đổi trên hiệu năng của hệ thống
- ❑ Giống nhà quản lý, không cần cái nhìn cục bộ - bức tranh cục bộ những phần của hệ thống và chúng thực thi như thế nào.
- ❑ Sử dụng mô hình vật lý như sơ đồ ngữ cảnh để triển khai và thể hiện thành phần chính và chúng liên hệ với môi trường, như vậy giúp nhà phân tích thu được hiểu biết tốt về hệ thống mà không cần lãng phí xem chi tiết thiết kế mức thấp và code.

Designers

- ❑ **Thiết kế kiến trúc (Architectural design)** kết quả trong sản phẩm thành phần chức năng, cấu trúc dữ liệu mức khái niệm và tương tác giữa các thành phần khác nhau
- ❑ **Thiết kế chi tiết (Detailed design)** kết quả trong thuật toán chi tiết, thể hiện dữ liệu, cấu trúc dữ liệu, giao diện giữa các thủ tục và chu trình.
- ❑ **Khi bảo trì, người thiết kế:**
 - trích rút thông tin và xác định cải tiến có thể được cung cấp bởi kiến trúc, cấu trúc dữ liệu, luồng dữ liệu và luồng kiểm soát của hệ thống hiện tại,
 - thông qua chương trình nguồn để lấy ý tưởng độ lớn công việc, vùng phạm vi của hệ thống bị tác động, và kiến thức và kỹ năng đòi hỏi bởi nhóm lập trình
- ❑ Dùng khái niệm che dấu thông tin, mô đun, phân rã chương trình, dữ liệu trừu tượng, hướng đối tượng, lý thuyết thiết kế tốt, sơ đồ luồng dữ liệu, sơ đồ luồng kiểm soát, sơ đồ cấu trúc, qui trình phân cấp đầu vào/đầu ra có thể giúp người thiết kế thu được hiểu biết tốt về hệ thống trước khi thiết kế thay đổi

Programmers & thông tin giúp cho họ

1. Quyết định restructure hay rewrite phân đoạn chương trình cụ thể hay không
2. Dự đoán dễ dàng bất kỳ tác động khi thực hiện thay đổi tác động những phần khác của hệ thống
3. Đưa ra những giả thiết vị trí và nguyên nhân gây ra lỗi
4. Xác định tính khả thi của những thay đổi đề xuất và cho thông báo cấp quản lý bất kỳ những vấn đề thấy trước.

Ví dụ programmer cần biết:

- ❑ Chức năng của thành phần đơn lẻ của hệ thống và mối tương quan.
- ❑ Mỗi khối lệnh làm gì, kết quả thực hiện (control flow),
- ❑ Tác động qua lại trên đối tượng dữ liệu (data flow)
- ❑ và mục đích tập các câu lệnh (functions)

Thảo luận

- ❑ **Exercise 6.1** Mục tiêu đạt được của bạn là gì khi cố gắng hiểu chương trình
- ❑ **Exercise 6.2** Tại sao hiểu chương trình là quan trọng?
- ❑ **Exercise 6.3** Giả sử bạn là lập trình viên, bạn được yêu cầu như sau (i) cung cấp tiện ích quản lý thông điệp cho hệ thống vận hành quản lý thông tin (MIS), và (ii) tích hợp hệ thống MIS vào gói văn phòng tự động. Những thông tin về MIS bạn cần làm gì, có tác động đến thay đổi không? Chỉ ra lý do.

4.3 MÔ HÌNH QUI TRÌNH NẮM BẮT THÔNG TIN

- ❑ Mô hình qui trình nắm bắt thông tin
 - reading about the program
 - reading its source code
 - running it
- ❑ Chiến lược nắm bắt chương trình
 - Top-Down Model (Brook's model)
 - Bottom-Up / Chunking Model
 - Opportunistic Model
- ❑ Bài tập: đọc tìm hiểu các mô hình trên trong tài liệu ebook chính

Mô hình qui trình nắm bắt thông tin

□ *Read about the program*

- Sơ lược hệ thống – tài liệu đặc tả và thiết kế
- Sơ đồ cấu trúc và dữ liệu và control flow
- Phát triển tổng quan và hiểu biết tổng thể hệ thống.
- Giai đoạn này có thể bỏ qua nếu tài liệu hệ thống không chính xác, không cập nhật và không tồn tại.

Mô hình qui trình nắm bắt thông tin

❑ *Read the source code*

○ Cái nhìn toàn cục

- ✓ top-level understanding of the system
- ✓ xác định phạm vi bất kỳ tác động thay đổi có thể có ở phần khác của hệ thống

○ Xem xét mức cục bộ

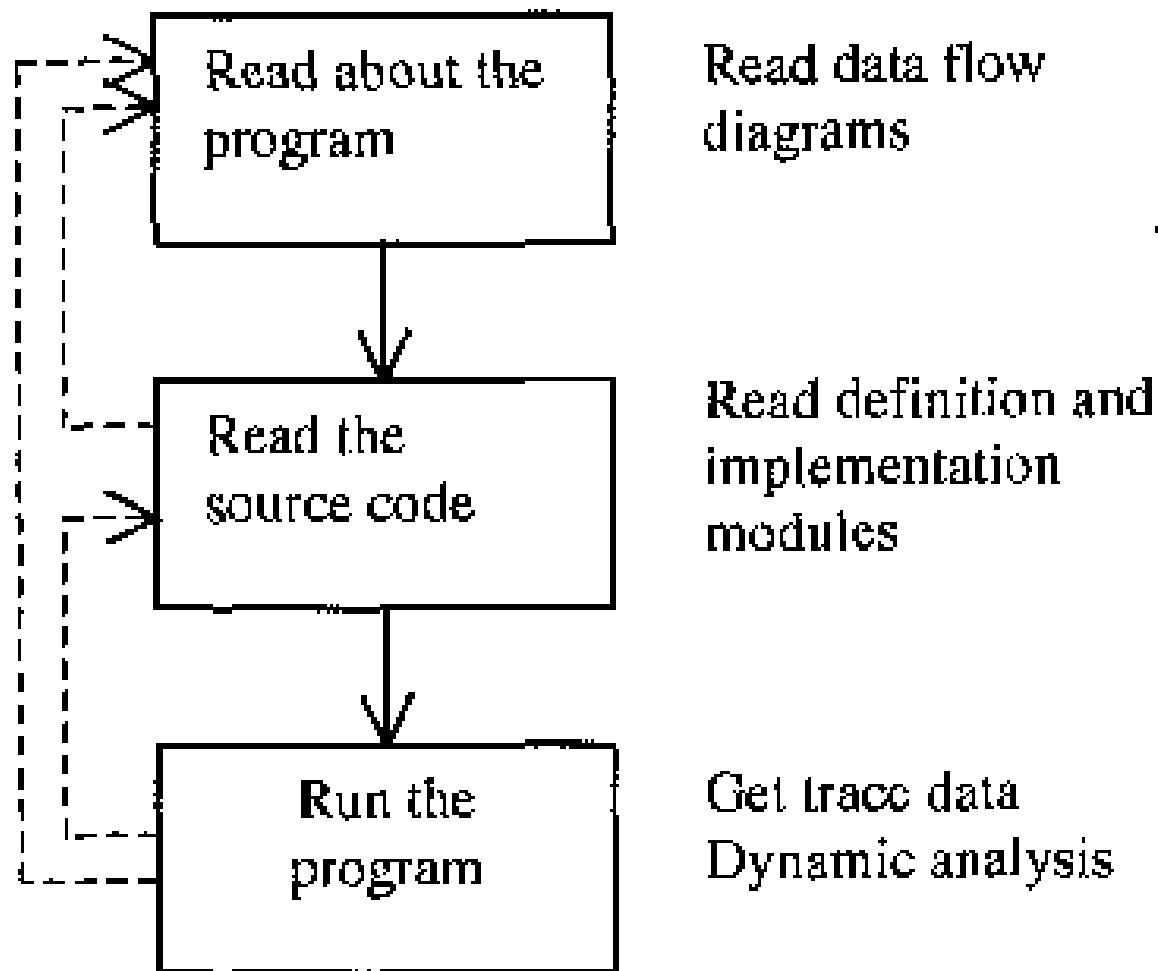
- ✓ tập trung vào phần cụ thể của hệ thống.
- ✓ Thông tin về cấu trúc hệ thống, loại dữ liệu, mẫu thuật toán

Mô hình qui trình nắm bắt thông tin

□ *Run program*

- to study the dynamic behavior (Ex: trace data)
- reveal some characteristics of the system which are difficult to obtain by just reading the source code.

Mô hình quy trình nắm bắt thông tin



Hình 6.2

Các bước nắm bắt thông tin chương trình

- ❑ Người lập trình có cách để suy nghĩ, giải quyết vấn đề, chọn lựa kỹ thuật và công cụ. Tuy nhiên có ba bước cơ bản để hiểu chương trình:
 - Bước 1: Đọc chương trình
 - Bước 2: Đọc chương trình nguồn (source code)
 - Bước 3: Chạy chương trình (Run)
- ❑ Thảo luận exercise 6.4: Mô hình qui trình nắm bắt thông tin Hình 6.2 (như 3 bước trên) có khác biệt và tương tự với những cách mà bạn đã sử dụng. Nêu rõ lý do?

Mental Models

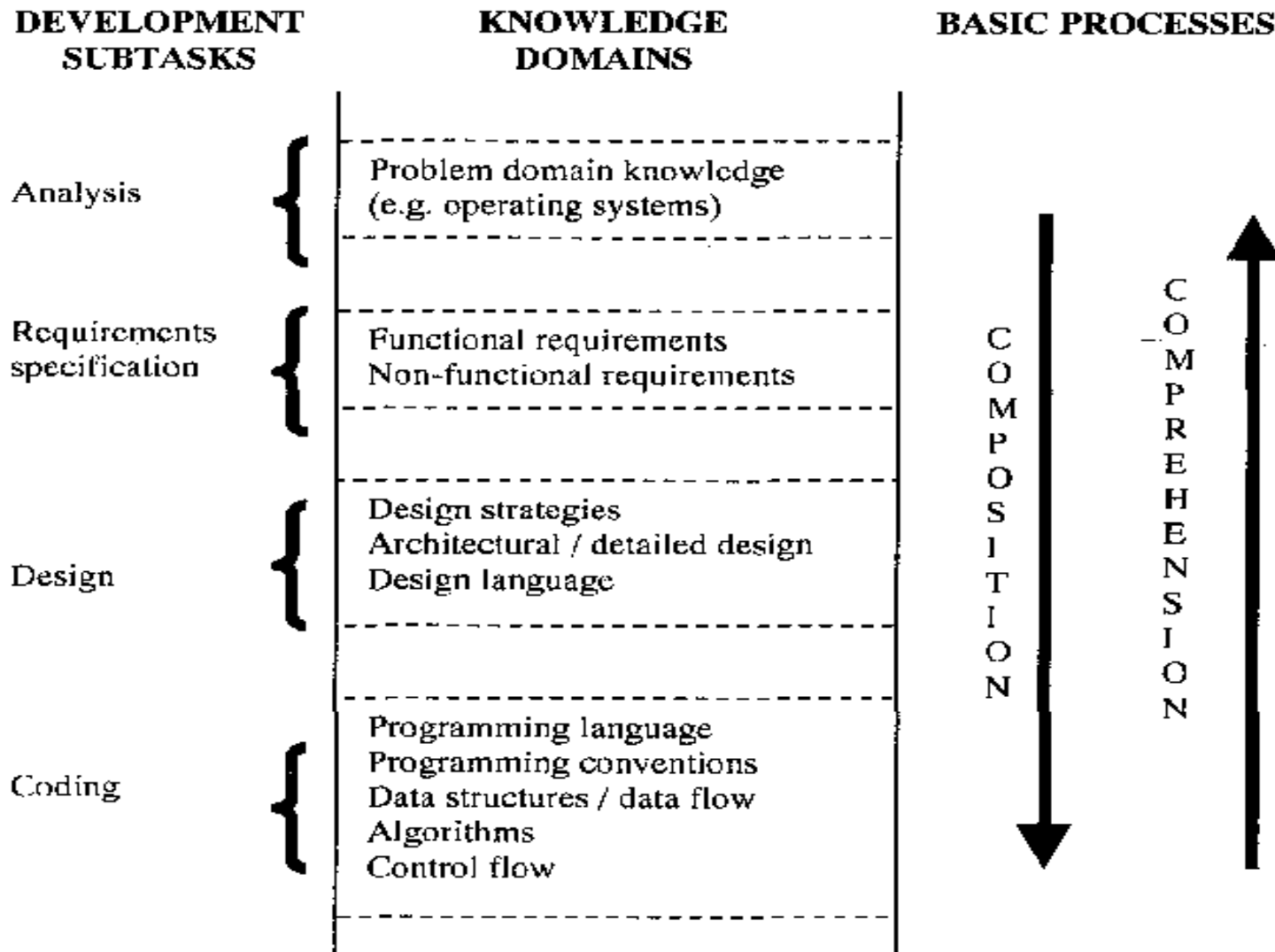
- ❑ Our understanding of a phenomenon depends on our ability to form a mental representation
- ❑ which serves as a working model of the phenomenon to be understood
- ❑ The content and formation of mental models hinges on cognitive structures and cognitive processes.
- ❑ The mental model is formed after observation, inference or interaction with the target system.

Chiến lược nắm bắt chương trình

Table 6.2 Program comprehension strategies and their differences

Model	Features		
	<i>Tenet of the model</i>	<i>Cognitive process</i>	<i>Cognitive structure</i>
Top-down	Program understanding is mapping from how the program works (programming domain) to what is to be done (problem domain)	Top-down reconstruction of knowledge domains and their mappings Reconstruction based on hypotheses creation, confirmation and refinement cycle	Problem and programming domain knowledge Potential intermediate domain knowledge Multiple layers of domain knowledge
Bottom-up	Recognition of recurring patterns in program code	Bottom-up chunking of recognised patterns to produce high-level semantic structures	Mapping between knowledge domains Hierarchical multi-layered arrangement of patterns
Opportunistic	Combination of both top-down and bottom-up strategies	Top-down and bottom-up cues are exploited as they become available An assimilation process is used to obtain information from source code and system documentation	Similar to top-down and bottom-up representations depending on the level of abstraction

Phạm vi kiến thức trong năm bắt thông tin



Các hướng dẫn cho chương trình

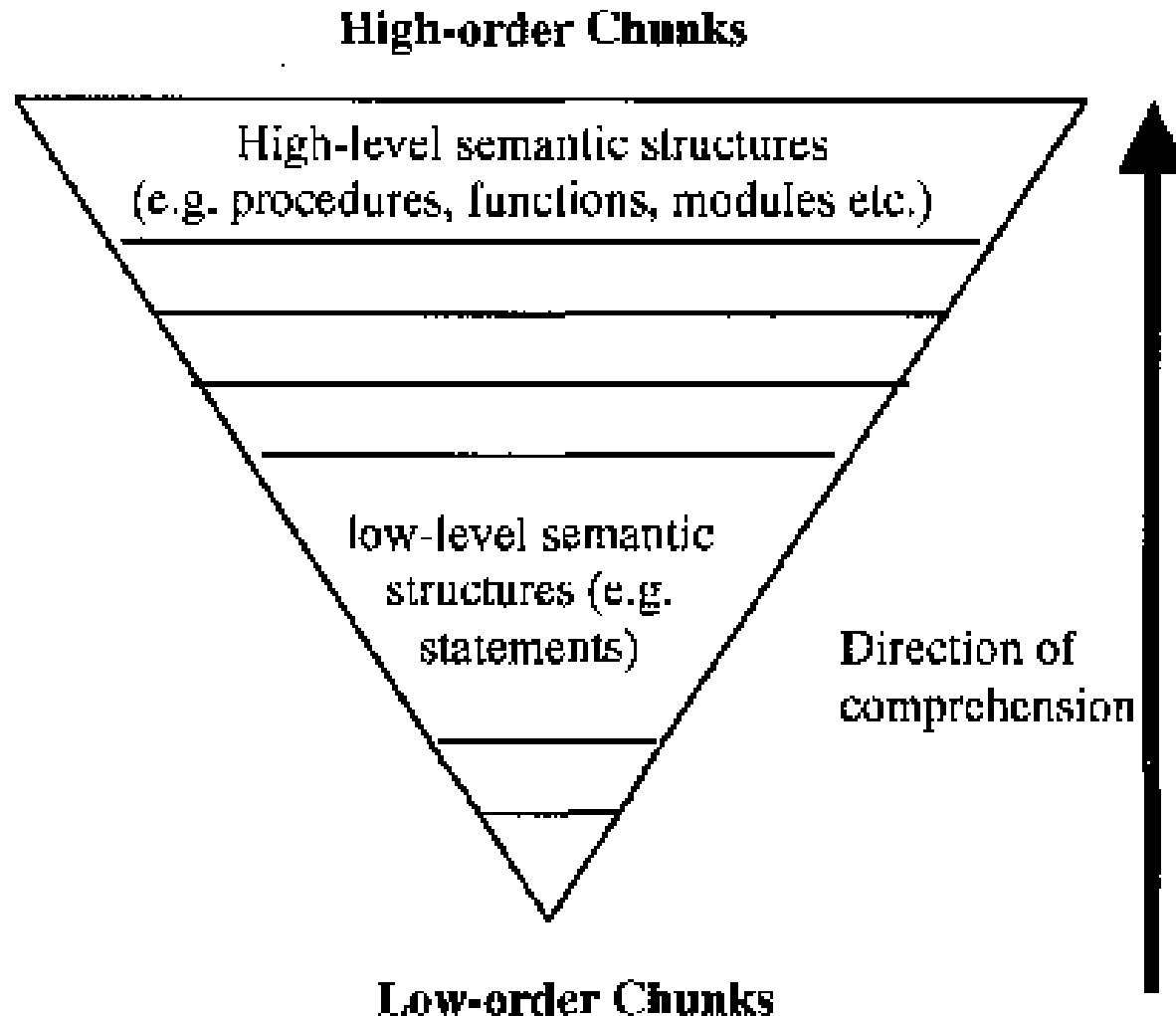
❑ *Internal to the program text*

1. Prologue comments, including data and variable dictionaries
2. Variable, structure, procedure and label names
3. Declarations or data divisions
4. Interline comments
5. Indentation or pretty-printing
6. Subroutine or module structure
7. I/O formats, headers, and device or channel assignments

❑ *External to the program*

1. Users' manuals
2. Program logic manuals
3. Flowcharts
4. Cross-reference listings
5. Published descriptions of algorithms or techniques

Bottom-up



Điểm yếu của top-down và bottom-up

- ❑ Những điểm yếu chính của cả chiến lược nắm bắt thông tin bằng top-down and bottom-up:
 - Thiếu xem xét chú ý đến đóng góp những yếu tố như công cụ hỗ trợ sẵn để hiểu chương trình;
 - Những sự kiện mà qui trình hiểu chương trình hiếm khi tham dự như vai trò các mô hình được định nghĩa tốt. Trái lại người lập trình hướng đến bất kỳ mối liên gắn kết có trước mà được xảy ra như cách tình cờ cơ hội.

Kỹ thuật đọc hiểu

- ❑ **Exercise 6.5:** Liệt kê những loại khác nhau của chiến lược hiểu chương trình, phân biệt giữa chúng
- ❑ **Exercise 6.6:** Những chiến lược gì bạn đã dùng và trong những hoàn cảnh nào?

Các yếu tố tác động đến đọc hiểu

- ❑ **Phạm vi kiến thức:** Chuyên gia, Vấn đề, Ứng dụng, hệ thống
- ❑ Thực nghiệm chương trình, vấn đề thực thi: Độ phân rã, tính môđun, tính che dấu thông tin, Thuật toán, Chương trình, cách đặt tên, ghi chú
- ❑ **Tài liệu:** bên ngoài, bên trong tổ chức
- ❑ Tổ chức/ thuyết trình:
- ❑ Công cụ hỗ trợ nắm bắt thông tin: công cụ phân tích tĩnh/ động

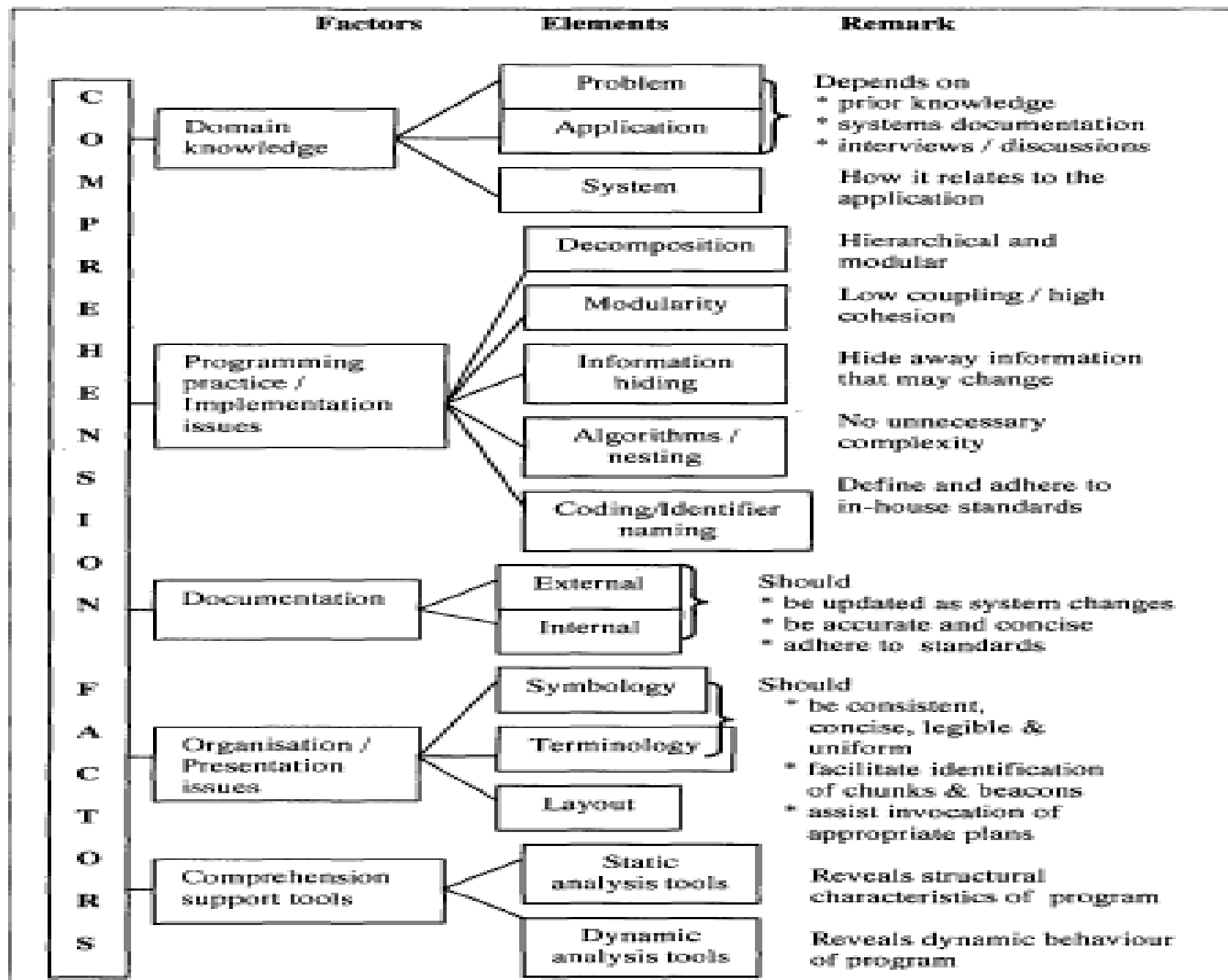


Figure 6.5 Taxonomy of program comprehension-related factors

Chuyên gia

- *“Experts differ from novices in both their breadth and their organisation of knowledge: experts store information in larger chunks organised in terms of underlying abstractions. This organisation apparently facilitates quick recognition of problem types and recall of associated solution strategies.”*

Petre

- Người lập trình càng có kinh nghiệm phạm vi ứng dụng với ngôn ngữ lập trình, càng dễ và nhanh chóng hiểu chương trình và cũng như toàn bộ hệ thống hiệu quả

Vấn đề thực thi

- ❑ Kiểu/ cách thức đặt tên
- ❑ Ghi chú chương trình
- ❑ Cơ chế phân rã
 - Phân rã mô đun
 - Lập trình có cấu trúc
- ❑ Đề xuất document standard và coding standard, phong cách lập trình => viết bằng văn bản thực thi cho nhóm dự án (**Bài tập**)

Tài liệu

- ❑ Tài liệu hệ thống rất hữu ích và quan trọng bởi nó không chỉ đầu mối liên lạc tác giả gốc của hệ thống thông tin.
- ❑ Đó là phần báo cáo trang trọng trong công nghiệp phần mềm: từ dự án khác, phòng ban, và công ty khác. Như vậy, khi người bảo trì cần truy xuất vào hệ thống tài liệu để có thể hiểu chức năng, thiết kế, thực thi và vấn đề liên quan đến bảo trì thành công.
- ❑ Đôi khi, tài liệu hệ thống **không chính xác**, quá lỗi thời **chưa cập nhật**. Trong trường hợp như vậy, người bảo trì phải thường xuyên xem sơ liệu nội bộ với chính chương trình nguồn – ghi chú chương trình.

Tổ chức/ thuyết minh chương trình

- ❑ Thuyết minh chương được cải tiến có thể cải thiện khả năng hiểu chương trình:
- ❑ Thuận tiện biểu thức rõ ràng và chính xác mô hình chương trình và truyền thông của các mô hình này đối với người đọc chương trình
- ❑ Nhấn mạnh đến luồng kiểm soát, cấu trúc phân cấp chương trình và tính logic và tổng hợp của người lập trình – mục đích gạch dưới cấu trúc và cải thiện tính dễ nhìn của chương trình nguồn qua cách sử dụng **ngắt dòng**, **khoảng trắng**, **khối** và **tô bóng**

Công cụ hỗ trợ nắm bắt thông tin

- ❑ Có công cụ được sử dụng để tổ chức và thể hiện chương trình nguồn theo cách thực hiện càng rõ ràng càng dễ đọc và như vậy càng dễ hiểu.
- ❑ 'Book Paradigm' là pretty-printer, static analyser và browser.
- ❑ Nhiều công cụ đọc hiểu được thiết kế phục vụ trợ giúp cho người đọc hiểu, tăng tốc độ, qui trình hiểu. Tuy nhiên, đầu ra của công cụ này không cung cấp sự giải thích chức năng của mỗi thành phần. Ở đây mô tả Book Paradigm và một số đặc trưng của nó

Ví dụ

{ Contents }	
{ This contains a }	
{ listing of the }	
{ chapters }	
Chapter 1.....	5
Chapter 2.....	15
Chapter 3.....	23

{ Chapter 1 }	
{ This contains }	
{ the following }	
{ programs }	
{ SearchPatRecord }	
{ }	
{ }	

{ Index }	
Clinic	3
DateOfBirth	6
GPNmae	4, 20
PatientAddress ...	2, 15
PatientID	7
PatientName	2, 5
NextOfKin	8

Ví dụ Hình 6.10

```
FROM BasicIO IMPORT WriteReal, WriteString, WriteLn, ReadInt,
    WriteInt;
CONST max = 20;
VAR a : ARRAY [1..max] OF INTEGER; number: INTEGER; total:
    INTEGER;
BEGIN
    WriteString("Type in 20 numbers"); number := 0;
    WHILE number <> 20 DO number := number + 1; ReadInt (afnumber)];
    END;
    WriteString ("The 20 numbers in reverse are "); WriteLn; number := 20;
    REPEAT WriteInt (a[number]),max); WriteLn; number := number - 1
    UNTIL number = 0;
    number := 20 total := 0;
    WHILE number <> 0 DO total := total + a[number]; DEC (number); END;
    WriteString ("The average for the 20 numbers is ");
    WriteReal (FLOAT (total) / FLOAT (max), 12);
END AddNumbers
```

Bài tập thảo luận

- ❑ **Exercise 6.7** Liệt kê và giải thích các yếu tố chính tác động đến việc hiểu một chương trình
- ❑ **Exercise 6.8** Bạn có thể cải thiện khả năng đọc hiểu chương trình Hình 6.10 bằng những cách xử lý nào?
- ❑ **Exercise 6.9** Liệt kê tất cả công cụ bảo trì đã có trong hệ thống của bạn. Có găng thử 3 trong những công cụ này, với mỗi loại chức năng chính của chúng là gì và làm thế nào nó cải thiện khả năng đọc.
- ❑ **Exercise 6.10** Tại sao quan trọng đối với người bảo trì thu được hiểu biết tốt chiến lược nắm bắt chương trình khác nhau và vấn đề dựa trên kinh nghiệm

4.4 REVERSE ENGINEERING

- ❑ Định nghĩa
- ❑ Mục đích và mục tiêu của reverse engineering
- ❑ Các mức của reverse engineering
- ❑ Kỹ thuật hỗ trợ
- ❑ Các lợi ích

Định nghĩa

- ❑ **Abstraction** – là mô hình tóm tắt chi tiết chủ đề được tái thể hiện
- ❑ **Forward engineering** – tiếp cận công nghệ phần mềm truyền thống bắt đầu với phân tích yêu cầu và tiến hành thực thi hệ thống.
- ❑ **Reengineering** – Quy trình kiểm tra và thông báo nơi hệ thống thay đổi lần đầu bởi reverse engineering và sau đó forward engineering.
- ❑ **Restructuring** – chuyển đổi một hệ thống từ hình thức này sang hình thức khác.
- ❑ **Reverse engineering** – Quy trình phân tích một hệ thống: nhận diện thành phần hệ thống và mối liên hệ bên trong và tạo thể hiện hệ thống trong hình thức khác ở mức trừu tượng cao hơn.

Tính trừu tượng

- ☐ *Trừu tượng chức năng*
- ☐ *Trừu tượng dữ liệu*
- ☐ *Trừu tượng qui trình*

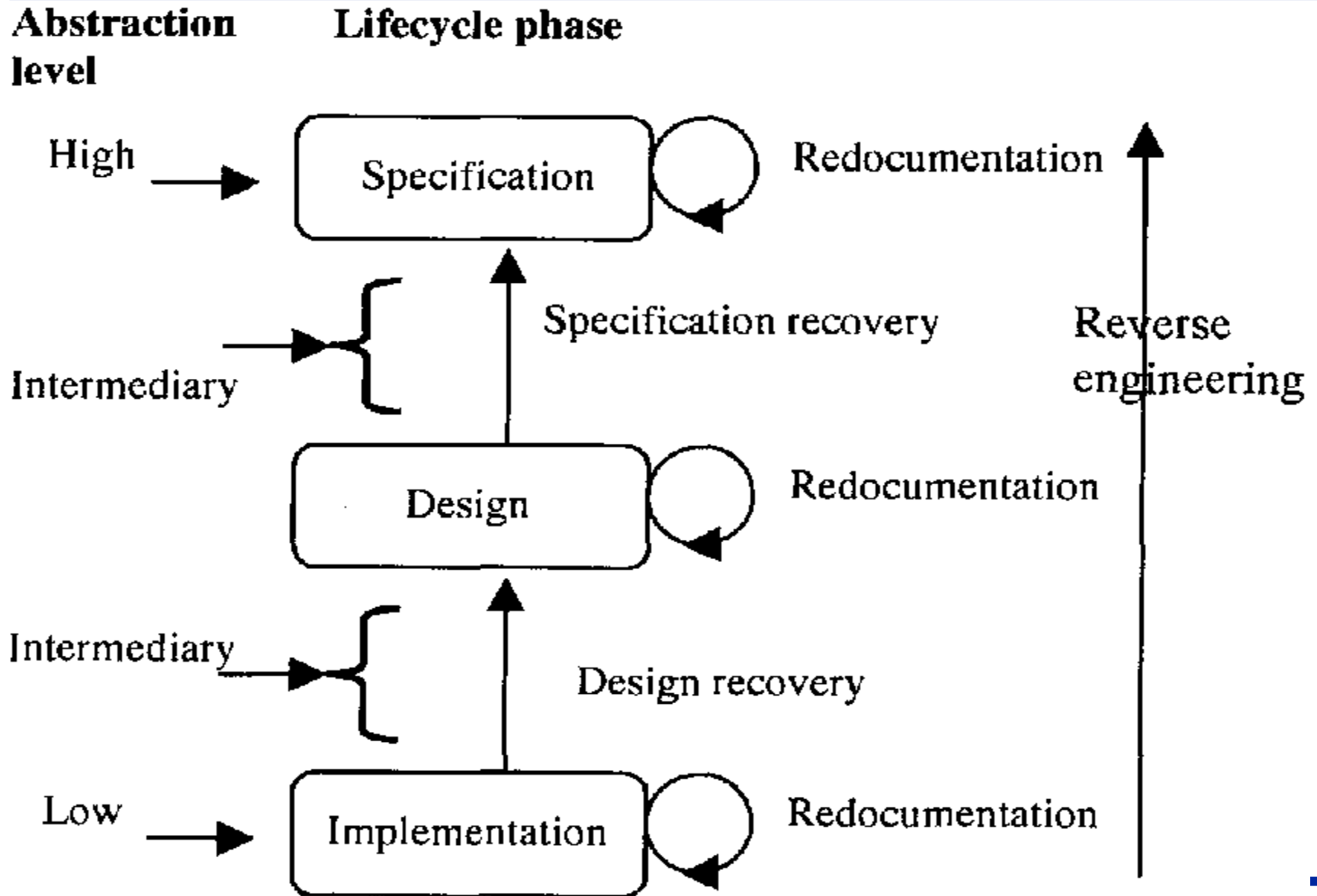
Mục đích và mục tiêu của reverse engineering

- ☐ *Khôi phục thông tin mất mát :*
- ☐ *To facilitate migration between platforms:*
- ☐ *Cải thiện và cung cấp sưu liệu :*
- ☐ *To provide alternative views:*
- ☐ *Trích rút thành phần có thể dùng lại :*
- ☐ *Đối phó với độ phức tạp :*
- ☐ *Dò hiệu ứng phụ:*
- ☐ *Giảm nỗ lực bảo trì :*

Tóm tắt mục tiêu và lợi ích

Objectives	Benefits
1. To recover lost information	1. Maintenance
2. To facilitate migration between platforms	(a) enhances understanding, which assists identification of errors
3. To improve and/or provide documentation	(b) facilitates identification and extraction of components affected by adaptive and perfective changes
4. To provide alternative views	(c) provides documentation or alternative views of the system
5. To extract reusable components	
6. To cope with complexity	2. Reuse: Supports identification and extraction of reusable components
7. To detect side effects	
8. To reduce maintenance effort	3. Improved quality of system

Các mức của reverse engineering



Các mức của reverse engineering

- ☐ *Redocumentation*
- ☐ ***Khôi phục thiết kế***
- ☐ ***Khôi phục đặc tả***
- ☐ ***Những điều kiện cho Reverse Engineering***

❑ **Forward Engineering**

❑ **Restructuring**

- *Control-flow-driven restructuring:*
- *Efficiency-driven restructuring:*
- *Adaption-driven restructuring:*

❑ **Reengineering**

```
IF ExamScore >= 75 THEN
    Grade := 'A'
ELSIF ExamScore >= 60 THEN
    Grade := 'B'
ELSIF ExamScore >= 50 THEN
    Grade := 'C'
ELSIF ExamScore >= 40 THEN
    Grade := 'D'
ELSE
    Grade := 'F'
ENDIF
```

```
CASE ExamScore OF
    75..100 : Grade := 'A'
    60..74 : Grade := 'B'
    50..59 : Grade := 'C'
    40..49 : Grade := 'D'
ELSE
    Grade := 'F'
ENDCASE
```

Bài tập

Part 1: Question 60 - Single Correct (2.5 Points)

Consider the following two pieces of codes and choose the best answer

Code 1:

```
switch (x)
```

```
{
```

```
    case 1:
```

```
        cout <<"x is 1";
```

```
        break;
```

```
    case 2:
```

```
        cout <<"x is 2";
```

```
        break;
```

```
    default:
```

```
        cout <<"value of x unknown";
```

```
}
```

Code 2

If (x==1)

{

 Cout << "x is 1";

}

Else if (x==2)

{

 Cout << "x is 2";

}

Else

{

 Cout << "value of x unknown";

}


- A Both of the above code fragments have the same behaviour
 - B Both of the above code fragments produce different effects
 - C The first code produces more results than second
 - D The second code produces more results than first
-

Các lợi ích

❑ **Bảo trì**

- Corrective Change
- Adaptive Change/Perfective change
- Preventive Change

❑ **Tính sử dụng lại phần mềm**

❑ **Reverse Engineering và kỹ thuật kết hợp trong thực nghiệm**

Case study 7.8

- ❑ Đọc hiểu case study 7.8
- ❑ Phân tích vấn đề hiện trạng đã nêu:
 - Vấn đề tự động hóa :
 - Vấn đề đặt tên :
 - ???
- ❑ Thực hiện bài tập theo sau:

Bài tập (1)

- ❑ **Exercise 7.1** Giải thích khác nhau giữa những loại khác nhau kỹ thuật reverse engineering và cho ví dụ thích hợp.
- ❑ **Exercise 7.2** Thực hiện khôi phục đặc tả và thiết kế trên tất cả hay các phần của hệ thống phần mềm mà bạn không quen (hệ thống nên có ít nhất 2K dòng code độ lớn).
 - Những kỹ thuật bạn dùng nhận diện đặc tả và thiết kế là gì và tại sao?
 - Những hình thức thể hiện bạn xem là phù hợp cho những tác vụ này là gì? Chỉ ra lý do.
 - Bài học kinh nghiệm mà bạn đã học được trong các công việc này là gì?

Bài tập (2)

- ❑ **Exercise 7.3** Một ngân hàng có substantial investment trong hệ thống phần mềm viết bằng Cobol ít nhất 1 triệu dòng code và chạy trên 20 năm. Nó được dùng cơ bản mỗi ngày để thực thi thao tác khác nhau như quản lý tài khoản khách hàng và loans. Sau vài năm cập nhật, cả dự định và không hoạch định – hệ thống trở nên quá đắt tiền để bảo trì. Kết quả là, ngân hàng muốn vài lời khuyên ở các bước tiếp để làm.
- Giả sử bạn được thuê làm việc như nhân viên tư vấn bảo trì.
 - Bạn sẽ cho Ngân hàng lời khuyên gì?
 - Chỉ ra lý do cho bất kỳ đề nghị mà bạn đã thực hiện

Yêu cầu thực hiện tuần tiếp theo

- ☐ Viết lại các báo cáo cho các thảo luận trên lớp và các bài tập
- ☐ Tiếp tục chuẩn bị công việc cho nhóm
- ☐ Mỗi nhóm tự chuẩn bị tìm hiểu và thử nghiệm một trong các công cụ hỗ trợ qui trình bảo trì → hướng dẫn sử dụng và demo trước lớp
- ☐ Chủ động Đăng ký thuyết trình nhóm với lớp trưởng, để điều phối nhóm thuyết trình ở các buổi học tiếp theo.