CHƯƠNG 5

CÀI ĐẶT PHẦN MỀM

Cài đặt là một công đoạn trong việc phát triển phần mềm và nó được xem là một hệ quả tất yếu của thiết kế. Tuy vậy, phong cách lập trình và các đặc trưng của ngôn ngữ lập trình có ảnh hưởng lớn đến chất lượng của phần mềm. Một chương trình được cài đặt tốt đem lại cho ta thuận lợi trong việc bảo trì sau này.

5.1. PHONG CÁCH CÀI ĐẶT CHƯƠNG TRÌNH

Sau khi sinh ra chương trình đích, chức năng của mỗi module phải rõ ràng, không cần tham khảo tới đặc tả thiết kế - nói cách khác, chương trình phải dễ hiểu. Phong cách lập trình bao hàm một triết lý về lập trình nhấn mạnh tới tính đơn giản và rõ ràng. Viết một chương trình máy tính là viết một dãy các câu lệnh trong ngôn ngữ hiện có. Cách thức mỗi mệnh đề này diễn tả trong chừng mực nào đó sẽ xác định ra tính dễ hiểu của toàn bộ chương trình...

Các yếu tố của phong cách bao gồm tài liệu bên trong, phương pháp khai báo dữ liệu, cách tiếp cận đến việc xây dựng câu lệnh, các kỹ thuật vào/ra...

5.1.1. Tài liệu chương trình

Tài liệu chương trình được hiểu là tài liệu bên trong của chương trình gốc. Nó bắt đầu với việc chọn lựa các tên gọi định danh, tiếp đến là vị trí và thành phần của việc chú thích, và kết luận với cách tổ chức trực quan của chương trình.

Việc lựa chọn các tên gọi định danh có nghĩa chính là điều chủ chốt cho việc hiểu chương trình. Những ngôn ngữ giới hạn tên biến hay nhãn chỉ có trong vài ký tự nên tự nó đã mang nghĩa mơ hồ. Nhưng ý nghĩa thông thường phải được áp dụng khi tên gọi đã được chọn, các tên gọi dài không cần thiết đôi lúc có thể đưa ra tiềm năng lỗi. Các nghiên cứu đã chỉ ra rằng cho dù một chương trình nhỏ thì một tên gọi có nghĩa cũng làm tăng tính dễ hiểu. Theo ngôn từ của mô hình cú pháp/ngữ nghĩa, tên có ý nghĩa làm "đơn giản hoá việc chuyển đổi từ cú pháp chương trình sang cấu trúc ngữ nghĩa bên trong".

Khả năng diễn tả những lời chú thích theo ngôn ngữ tự nhiên như một phần của bản in chương trình gốc đều được mọi ngôn ngữ lập trình cung cấp. Tuy nhiên, một số vấn đề nảy sinh:

- Bao nhiêu chú thích là "đủ"?
- Nên đặt chú thích vào đâu?
- Chú thích có che mờ luồng logic không?
- Chú thích có làm lạc hướng độc giả không?

- Liệu có chú thích "không bảo trì" không, và do đó không tin cậy được?

Tuy vậy, một điều là rõ ràng: phần mềm phải chứa tài liệu bên trong. Lời chú thích cung cấp cho người phát triển một ý nghĩa truyền thông với các độc giả khác về chương trình gốc. Lời chú thích có thể cung cấp một hướng dẫn rõ rệt, dễ hiểu trong khâu bảo trì của công nghệ phần mềm.

Có nhiều hướng dẫn đã được đề nghị cho việc viết lời chú thích. Các chú thích mở đầu và chú thích chức năng là hai phạm trù đòi hỏi cách tiếp cận có hơi khác. Lời chú thích mở đầu nên xuất hiện ở ngay đầu của mọi module. Định dạng cho lời chú thích như thế là:

1. Một phát biểu về mục đích chỉ rõ chức năng module.
2. Mô tả giao diện bao gồm:
    a) Mẫu lời gọi,
    b) Mô tả về mọi đối số,
    c) Danh sách tất cả các module thuộc cấp.
3. Thảo luận về dữ liệu thích hợp như các biến quan trọng và những hạn chế và giới hạn về cách dùng chúng, và các thông tin quan trọng khác.
4. Lịch sử phát triển bao gồm:
    a) Tên người thiết kế module (tác giả),
    b) Tên người xét duyệt (kiểm toán) và ngày tháng,
    c) Ngày tháng sửa đổi và mô tả sửa đổi,

Các chú thích mô tả được nhúng vào bên trong thân của chương trình gốc và được dùng để mô tả cho các hàm xử lý. Lời chú thích nên đưa ra một điều gì đó phụ trợ, không chỉ là lời diễn giải chương trình. Bên cạnh đó, lời chú thích mô tả nên:

- Mô tả các khối chương trình, thay vì chú thích cho từng dòng.
- Dùng dòng trống hay thụt cấp để cho lời chú thích có thể được phân biệt với chương trình
- Phải đúng đắn; một lời chú thích không đúng hay gây ra hiểu sai thì còn tồi tệ hơn là không có chú thích nào cả.

Với những tên gọi tượng trưng đúng đắn và việc chú thích tốt, việc làm tài liệu bên trong thích hợp sẽ được đảm bảo.

Khi một thiết kế thủ tục chi tiết được biểu diễn bằng cách dùng một ngôn ngữ thiết kế chương trình thì tài liệu thiết kế có thể được nhúng trực tiếp vào trong văn bản chương trình gốc như những câu chú thích. Kỹ thuật này đặc biệt có ích khi việc làm tài liệu được thực hiện trong hợp ngữ và giúp đảm bảo rằng cả chương trình và thiết kế sẽ được bảo trì khi những thay đổi được thực hiện cho cả hai.

Việc viết thụt cấp ở chương trình gốc chỉ ra kết cấu và khối logic của chương trình sao cho những thuộc tính này là thấy được so với lề bên trái. Giống như việc chú thích, cách tiếp cận tốt nhất tới việc thụt cấp là nên để mở cho tranh luận. Việc thụt cấp thủ công có thể trở nên phức tạp khi có sự sửa đổi chương trình và kinh nghiệm chỉ ra rằng khi đã tích luỹ đủ hiểu biết thì sẽ tăng cường được việc để lề cho khớp. Có lẽ cách tiếp cận tốt nhất là dùng bộ định dạng chương trình tự động (như công cụ CASE) sẽ đặt đúng việc thụt cấp cho chương trình gốc. Nó xoá bỏ đi gánh nặng của việc làm thụt cấp cho người lập trình, và có thể cải thiện khuôn dạng chương trình với tương đối ít nổ lực.

5.1.2. Khai báo dữ liệu

Độ phức tạp và việc tổ chức cấu trúc dữ liệu được xác định trong bước thiết kế nhưng phong cách khai báo dữ liệu thì được thiết lập khi chương trình được sinh ra. Thứ tự khai báo dữ liệu nên được chuẩn hoá cho dù ngôn ngữ lập trình không có yêu cầu bắt buộc. Điều này tạo điều kiện thuận lợi cho việc kiểm thử, gỡ rối và bảo trì. Thậm chí, khi có nhiều định danh được khai báo trong câu lệnh thì việc sắp xếp theo trật tự chữ cái cho các tên gọi đó cũng có giá trị.

Nếu thiết kế có mô tả trước cấu trúc dữ liệu phức tạp thì nên dùng chú thích để giải thích các điểm đặc thù trong cài đặt ở ngôn ngữ lập trình.

5.1.3. Xây dựng câu lệnh

Mặc dầu việc xây dựng luồng logic phần mềm được thiết lập ở thiết kế nhưng việc xây dựng câu lệnh nằm ở bước lập trình. Thực tế đã chứng minh, việc xây dựng các câu lệnh của chương trình nên tuân theo phong cách lập trình cấu trúc. Các câu lệnh nên đơn giản và trực tiếp, không bị xoắn vào nhau để đảm bảo hiệu quả.

Trong thể hiện chương trình, cách xây dựng câu lệnh đơn và việc thụt cấp chương trình minh hoạ cho đặc trưng logic và chức năng của giai đoạn này, và nên tuân theo các chỉ dẫn:

+ Tránh dùng các phép kiểm tra điều kiện phức tạp,
+ Khử bỏ các phép kiểm tra điều kiện phủ định,
+ Tránh lồng nhau giữa các điều kiện hay chu trình,
+ Dùng các dấu ngoặc để làm sáng tỏ các biểu thức,
+ Dùng các dấu cách và các ký hiệu dễ đọc để làm sáng tỏ nội dung câu lệnh,...

5.1.4. Vào và ra

Phong cách vào/ra được thiết lập khi phân tích và thiết kế phần mềm nhưng cách thức cài đặt vào/ra lại ảnh hưởng lớn đến người sử dụng hệ thống. Phong cách vào/ra sẽ thay đổi theo mức độ tương tác con người.

Với vào/ra theo lô thì cách tổ chức cái vào logic, kiểm tra lỗi vào/ra có nghĩa, phục hồi lỗi vào/ra tốt và định dạng báo cáo ra hợp lý là những đặc trưng mong muốn. Với vào/ra tương tác thì một sơ đồ đưa vào có hướng dẫn, đơn giản, việc kiểm tra lỗi kỹ lưỡng và có thể phục hồi, sự nhất quán định dạng vào/ra lại là các mối quan tâm chủ yếu.

Khi cài đặt vào/ra, cần thoả mãn các tiêu chí cơ bản sau:

+ Làm hợp lệ mọi cái vào,
+ Kiểm tra sự tin cậy của các tổ hợp dữ liệu vào quan trọng,
+ Giữ cho định dạng dữ liệu vào đơn giản,
+ Dùng các chỉ báo cuối dữ liệu thay vì yêu cầu người sử dụng xác định số các khoản mục vào,
+ Đặt nhãn cho các dữ liệu vào,
+ Giữ các định dạng dữ liệu vào thống nhất,...

5.2. NỀN TẢNG CỦA NGÔN NGỮ LẬP TRÌNH

5.2.1. Kiểu dữ liệu, định nghĩa kiểu dữ liệu và kiểm tra kiểu dữ liệu

Kiểu dữ liệu là loại dữ liệu được định nghĩa từ trước của ngôn ngữ và mỗi ngôn ngữ hỗ trợ một số kiểu dữ liệu. Tất cả các ngôn ngữ đều hỗ trợ biến, hằng số dùng trong dữ liệu số và dữ liệu ký tự. Kiểu dữ liệu được hỗ trợ chung là: số nguyên, số thực và xâu ký tự.

Một số ít ngôn ngữ hỗ trợ các kiểu dữ liệu khác như: Logical, Boolean, Pointer, Object, Bit, Date,... hoặc kiểu dữ liệu tự định nghĩa.

Kiểu Boolean sinh ra giá trị nhị phân True, False dựa trên so sánh logic. Pointer là địa chỉ của chương trình khác hoặc cấu trúc dữ liệu mà được dùng để tham chiếu đến trong chương trình. Object được xây dựng để đóng gói dữ liệu và phương thức. Kiểu dữ liệu Date định nghĩa ngày tháng năm trong một khuôn dạng hợp lệ - thay cho việc phải viết các chương trình để xử lý kiểu Date, ta có thể sử dụng các thủ tục có sẵn của ngôn ngữ.

Các cấu trúc dữ liệu như mảng, bảng, danh sách tuyến tính,... là loại thứ ba của cấu trúc dữ liệu của ngôn ngữ. Các ngôn ngữ có thể hỗ trợ hoặc không hỗ trợ kiểu này. Tuy nhiên, các kiểu dữ liệu đơn giản như mảng, danh sách tuyến tính,... thường được hầu hết các ngôn ngữ hỗ trợ.

Cuối cùng, kiểu dữ liệu tự định nghĩa là kiểu dữ liệu do lập trình viên định nghĩa và chỉ có giá trị trong một chương trình hoặc ứng dụng nhất định. Kiểu dữ liệu tự định nghĩa có thể dùng để định nghĩa các kiểu dữ liệu khi ngôn ngữ không hỗ trợ kiểu dữ liệu đó.

Kiểm tra kiểu dữ liệu là việc ngôn ngữ kiểm tra sự phù hợp của kiểu dữ liệu được định nghĩa trong các phép toán học và các toán tử logic. Có bốn mức kiểm tra kiểu, từ không kiểm tra kiểu đến kiểm tra chặt, mức độ chặt chẽ của kiểm tra phụ thuộc vào dạng ứng dụng. Nói chung các tiến trình càng cần sự chính xác, nhất quán và ổn định thì càng đòi hỏi mức độ kiểm tra kiểu chặt chẽ hơn. Trong lập trình hướng đối tượng, kiểm tra kiểu càng quan trọng bởi tính đa hình cho phép nhiều module thực hiện cùng chức năng trên nhiều kiểu dữ liệu khác nhau, cho nên kiểm tra kiểu chặt chẽ sẽ làm giảm khả năng chương trình gặp lỗi.

+ Không kiểm tra kiểu (typeless checking) nghĩa là không tiến hành sự kiểm tra kiểu một cách tường minh.

Ví dụ: Trong các ngôn ngữ không kiểu như Basic hoặc Cobol, các kí tự được phép gán bởi integer, nhưng có thể gây ra lỗi nếu trường này được tham chiếu như là một số nguyên.

Không có gì bảo đảm việc không gặp lỗi khi ta thao tác trên các trường không kiểu. Các ngôn ngữ hoặc chương trình dịch có cách xử lý trường không kiểu không thống nhất.

+ Mức kiểm tra kiểu tiếp theo là ép kiểu tự động (automatic type coercion), trong đó nhiều kiểu dữ liệu được phép dùng chung, nhưng không phải tất cả và có thể dẫn đến lỗi chuyển đổi các kiểu không tương thích. Mức kiểm tra kiểu này còn có tên kiểm tra kiểu dạng hỗn hợp (mixed mode type checking), những kiểu dữ liệu khác nhau nhưng thuộc cùng một phân loại được chuyển sang một kiểu đích đối với toán tử kiểu hỗn hợp.

Ví dụ, trong Fortran, trộn lẫn số thực và số nguyên trong toán tử toán học dẫn đến các kết quả không thể dự đoán được bởi vì kiểu đích (target type) được quyết định bởi việc định nghĩa trường kết quả. Nếu trường kết quả được định nghĩa là thực, kết quả tính toán là số thực. Nếu trường kết quả được định nghĩa là integer, tiến trình sẽ làm tròn câu trả lời (số thực) và đưa ra kết quả là integer.

+ Kiểm tra kiểu giả chặt (Pseudostrong type checking) là mức thứ ba của kiểm tra kiểu, nó cho phép thao tác các đối tượng dữ liệu thuộc cùng một kiểu dữ liệu, nhưng phép kiểm tra kiểu này chỉ áp dụng khi chúng được định nghĩa trong cùng một module. Pascal là ngôn ngữ có kiểm tra kiểu giả chặt, nó hỗ trợ kiểm tra kiểu chặt chẽ trong module, nhưng không hỗ trợ chéo giữa các module. Cho nên, dữ liệu truyền từ một module sang module khác có thể chuyển sang kiểu dữ liệu khác mà không bị bắt lỗi.

+ Ở mức cao nhất của kiểm tra kiểu của ngôn ngữ, kiểm tra kiểu chặt chẽ chỉ cho phép thao tác trên những đối tượng dữ liệu có cùng kiểu đã xác định từ trước, bất kể nó nằm trong cùng hoặc khác module. Nếu trong module có kiểu dữ liệu không hợp lệ, ứng dụng sẽ dừng và đưa ra một thông báo lỗi. Ada là ngôn ngữ cung cấp kiểm tra kiểu chặt chẽ.

5.2.2. Chương trình con

Sự tinh tế của ngôn ngữ thể hiện ở mức độ hỗ trợ module hoá và quản lý bộ nhớ. Module hoá là cách thức tạo ra chương trình con và hàm. Các ngôn ngữ khác nhau ở cách hỗ trợ chương trình con và dữ liệu của nó. Trước hết, khả năng định nghĩa chương trình con, hàm là quan trọng để có được các đặc trưng chương trình mong muốn. Thứ hai, dữ liệu trong các module được quản lý như thế nào? Dữ liệu có thể là cục bộ hoặc tổng thể. Khả năng có được dữ liệu cục bộ là quan trọng trong việc che giấu thông tin và giảm thiểu việc liên kết. Phạm vi dữ liệu tổng thể cần được giới hạn để đảm bảo chất lượng của chương trình trong việc giấu thông tin và sự liên kết.

Trong các ngôn ngữ, chương trình con được gọi thông qua tên của nó. Tuỳ chọn cho xử lý việc gọi bao gồm cả việc truyền dữ liệu bằng biến, bằng tên, bằng địa chỉ, hoặc bằng giá trị. Truyền giá trị đòi hỏi sự định nghĩa dữ liệu cục bộ trong khi truyền dữ liệu bằng tên hoặc bằng địa chỉ được sử dụng với hoặc dữ liệu cục bộ hoặc dữ liệu tổng thể.

Nói chung, khi sử dụng chương trình con, module chính gọi chương trình con làm những việc của nó và trả lại kết quả cho module chính. Khả năng hỗ trợ xử lý chương trình con đòi hỏi một hoặc nhiều hơn một mục vào hoặc điểm thoát. Xử lý Exit và Return cũng quan trọng khi chuyển quyền điều khiển giữa các module. Trong các trường hợp, càng nhiều cơ hội để vào và thoát khỏi module đã xác định trước, thì

lập trình viên càng cần sự thành thạo, đảm bảo khả năng xử lý thành thạo, đảm bảo khả năng xử lý hoàn hảo. Theo các nhà lập trình cấu trúc, một module được thiết kế tốt nên có một điểm vào và một điểm ra. Module một vào và một ra ít gây lỗi hơn so với các module có nhiều mục vào, điểm ra.

5.2.3. Cấu trúc điều khiển

Về bản chất, một chương trình máy tính là một bản mã hoá thuật toán. Ở đây, các đối tượng chịu thao tác được mô tả và kiến trúc thông qua cấu trúc dữ liệu còn các thao tác được mô tả thông qua các cấu trúc điều khiển. Như vậy, cấu trúc điều khiển của ngôn ngữ là yếu tố quyết định thao tác gì và thao tác như thế nào trên dữ liệu đã mô tả. Chúng cung cấp các khả năng xử lý: tuần tự, lặp và cách thức lựa chọn các cấu trúc dữ liệu.

Sự tuần tự có hai dạng: giữa các dòng lệnh và trong dòng lệnh. Lập trình viên điều khiển sự tuần tự giữa các dòng lệnh (between-command sequencing) như là một trật tự của các lệnh, còn sự tuần tự trong dòng lệnh đó chính là thứ tự ưu tiên của các phép toán -operator precendence- dùng trong thao tác dữ liệu, nó được các ngôn ngữ quy định sẵn. Với hai khối lệnh A, B tuân theo phương thức xử lý tuần tự thì với R là số lần thực hiện của khối lệnh ta có $R_A=R_B=1$. Cấu trúc tuần tự trong các ngôn ngữ lập trình thường tuân theo trật tự từ trái sang phải và từ trên xuống dưới.

Cấu trúc lựa chọn trong ngôn ngữ lập trình thường được mô tả dưới các từ khoá If hoặc Case. Với biểu thức điều kiện lựa chọn E và các khối lệnh lựa chọn $A_1,A_2,...,A_n$, theo ký hiệu trên ta có $1=R_E>=R_{A1}+...+R_{An}$.

Cấu trúc lặp trong ngôn ngữ lập trình được hỗ trợ bởi các dạng: lặp biết trước số lần lặp (For), lặp với kiểm tra điều kiện lặp trước - lính canh đặt trước (While.....do), và lặp với kiểm tra điều kiện lặp sau (Do.....while).

Lặp biết trước số lần lặp được đánh dấu bởi các biểu thức đếm được đầu (D) đến cuối (C). Với khối lệnh A trong thân vòng lặp, ta có $R_C=R_D=1$ và $R_A=C-D+1$ nếu C>=D, ngược lại thì $R_A=0$ nếu C<D.

Lặp với kiểm tra điều kiện lặp trước ứng với biểu thức điều kiện lặp E thì lúc này, khối lệnh A trong thân vòng lặp tuân theo: $1<=R_E=R_A+1$.

Còn lặp với kiểm tra điều kiện lặp sau ứng với biểu thức điều kiện lặp E thì khối lệnh A trong thân vòng lặp tuân theo: $1<=R_E=R_A$.

Sự tương đương của các chương trình trong việc mã hoá bởi các cấu trúc điều khiển đã được chỉ ra ở định lý Boehm&Jaccopini như sau: *Mọi chương trình P được thể hiện bằng sơ đồ khối đều tồn tại một chương trình Q tương đương mạnh với nó nhưng chỉ dùng hai cấu trúc điều khiển để mô tả đó là cấu trúc tuần tự và cấu trúc lặp với điều kiện lặp xét trước.*

Ngoài việc cung cấp các cấu trúc điều khiển, các ngôn ngữ còn hỗ trợ các phương thức như: *Exits, Return, Fail,...* để thoát khỏi module hiện tại trở về module gọi hoặc tới module khác.

Bên cạnh các cấu trúc điều khiển đã đề cập ở trên, đệ quy là một thuộc tính của module. Chúng xuất hiện khi module gọi chính chúng hoặc các module gọi lẫn nhau. Trong một số ngôn ngữ lập trình, sự đệ quy không được hỗ trợ một cách tường minh, nhưng nó lại được coi là sức mạnh chính của một số ngôn ngữ khác- ví dụ như ngôn ngữ Prolog. Ở các chương trình sử dụng đệ quy, đòi hỏi khả năng duy trì hàng đợi hoặc stack của chương trình.

5.2.4. Vào và ra dữ liệu

Có bốn dạng thông tin vào/ra (I/O) là: lệnh vào/ra cụ thể, hướng bản ghi, hướng tập hợp, và hướng mảng.

Vào/ra hướng bản ghi đọc hoặc ghi các bản ghi vật lý, bản ghi này có thể chứa đựng một hoặc nhiều bản ghi logic. Các bản ghi (hoặc là bộ trong đại số quan hệ) sẽ nhóm các trường dữ liệu có quan hệ với nhau. Vào/ra hướng bản ghi đòi hỏi đóng mở file, đọc ghi các bản ghi và quản lý người sử dụng tất cả các công việc xử lý file. Ví dụ: Cobol, Fortrans, Assembler, Ada là các ngôn ngữ hướng bản ghi.

Hướng tập hợp giả sử rằng tất cả các bản ghi (hoặc các bộ) được coi như nhau. Ngôn ngữ điều khiển mọi file và mọi tiến trình đọc ghi theo sự lựa chọn mà người sử dụng định nghĩa. Ở cuối thủ tục, tập các bản ghi (là kết quả của thủ tục) được lưu trữ trong bộ nhớ phục vụ cho việc in ấn, hiển thị. Ví dụ SQL là ngôn ngữ hướng tập hợp.

Vào/ra hướng mảng là đọc và ghi chuỗi các trường được giả thiết là kiểu mảng, người sử dụng có nhiệm vụ định nghĩa và thao tác kiểu dữ liệu của mảng. Ngôn ngữ chỉ đơn giản đọc và ghi cho đến cuối mảng dữ liệu. Pascal là ngôn ngữ hướng mảng. Vào/ra trực tiếp danh sách (list-directed I/O) là một biến thể của vào/ra hướng mảng. Fortrans sử dụng vào/ra trực tiếp danh sách để định nghĩa danh sách các tên biến, mỗi tên biến được truy cập trực tiếp khi chúng được đọc. Nó đọc cho đến khi danh sách đầy rồi xử lý cho đến khi lệnh đọc được thực hiện lại. Các mục dữ liệu không được định dạng cụ thể, mà khuôn dạng ngầm chỉ trong tên biến.

5.2.5. Quản lý bộ nhớ

Sự tinh tế của ngôn ngữ còn thể hiện ở mức độ lập trình viên kiểm soát điều khiển việc quản lý bộ nhớ. Quản lý bộ nhớ là khả năng chương trình phân bổ bộ nhớ máy tính khi cần. Đây là tuỳ chọn nhưng chúng được sử dụng nhiều khi xử lý danh sách biến và các ứng dụng thời gian thực quản lý tài nguyên nhiều người sử dụng. Các ngôn ngữ có độ tinh tế thấp sử dụng bộ nhớ tĩnh: chương trình nhận lượng bộ nhớ lớn nhất tại thời điểm khởi tạo. Nếu chương trình cần nhiều bộ nhớ hơn lượng được cấp phát thì chương trình sẽ bị treo, ngôn ngữ điều khiển nhiệm vụ (job control language) sẽ cấp phát lượng bộ nhớ thiếu đó để chương trình chạy lại. Các ngôn ngữ tinh tế hơn sử dụng khả năng cấp phát bộ nhớ động, tức là chỉ cấp phát bộ nhớ khi nào cần thiết.

5.2.6. Quản lý lỗi

Quản lý lỗi là mức chương trình được cài đặt để phát hiện và quản lý lỗi mà không phải dừng chương trình. Khả năng này sẽ làm tăng độ phức tạp và mở rộng phạm vi hữu ích của ngôn ngữ. Ví dụ Cobol cho phép ta chặn đứng lỗi dữ liệu như

tràn, chia cho 0, nhưng lại không chặn được lỗi như định nghĩa dữ liệu không hợp lệ, đọc quá cuối file,.... Ngược lại Smalltalk cho phép chặn được bất kỳ lỗi nào.

Tóm lại, ngôn ngữ lập trình khác nhau ở mức độ chúng hỗ trợ các cách khác nhau cho điều khiển dữ liệu, xử lý vào/ra, thao tác toán học, chương trình con, và quản lý bộ nhớ. Ngôn ngữ hỗ trợ ít là ngôn ngữ đơn giản. Cấu trúc ngôn ngữ càng phức tạp thì phạm vi bao quát của nó càng lớn.

5.3. CÁC ĐẶC TRƯNG CỦA NGÔN NGỮ CÀI ĐẶT

Các đặc trưng được đánh giá ở đây gồm: đồng nhất (uniformity), sáng sủa (ambiguity), cô đọng (compactness), địa phương – cục bộ (locality), tuyến tính (linearity), dễ lập trình, dịch hiệu quả, khả chuyển. Tính sẵn có của công cụ trợ giúp, các bộ sinh mã và tính sẵn dùng của công cụ trợ giúp kiểm tra cũng được thêm vào nhằm làm tăng tính hấp dẫn của ngôn ngữ.

Tính đồng nhất là cách sử dụng ký hiệu nhất quán trong cả ngôn ngữ. Một ví dụ của sự không nhất quán trong Focus là việc sử dụng dấu ngoặc đơn cho tiêu đề bản báo cáo do người sử dụng tạo ra và dấu ngoặc kép của trang bản báo cáo. Ngôn ngữ không nhất quán cản trở người sử dụng học và dễ gây lỗi.

Tính sáng sủa đề cập đến mức độ con người và chương trình dịch bất đồng trong việc dịch các câu lệnh của ngôn ngữ. Lý tưởng nhất là ý nghĩa của con người tương tự với sự biên dịch của trình dịch và chương trình dịch ra giống sự nhận thức của con người. Thật không may, tính sáng sủa có những vấn đề cố hữu của mình, như các ứng dụng trí tuệ nhân tạo (ứng dụng suy luận trong cả tiến trình), khi thêm luật, cơ chế mới vào, sự thông dịch của dữ liệu, luật đó có lẽ cũng thay đổi.

Tính cô đọng của ngôn ngữ nằm ở sự ngắn gọn. Các đặc trưng của chương trình bao gồm sự kết cấu có cấu trúc, từ khoá và viết tắt, hàm có sẵn, đã đơn giản hoá việc lập trình. Tương phản với hai ngôn ngữ thế hệ bốn SQL và Focus là Cobol, ngôn ngữ thế hệ ba. Thực tế cho thấy 3 đến 5 dòng lệnh 4GLs tương đương với 50 đến 150 dòng lệnh trong ngôn ngữ Cobol. Thời gian học Focus ngắn hơn Cobol một phần là bởi tính cô động của ngôn ngữ.

Tính cô đọng bao hàm tính cục bộ trong việc cung cấp sự phân đoạn tự nhiên của mã lệnh, làm đơn giản hoá việc học, trực quan hoá từng phần của vấn đề và có thể mô phỏng các giải pháp. Tính cục bộ được cung cấp thông qua khối case, hoặc những cơ chế phân đoạn (chunks). Sự phân đoạn có lẽ được thực hiện thông qua thực thi đoạn mã trong ngôn ngữ Cobol, cấu trúc case trong ngôn ngữ Focus, hoặc định nghĩa đối tượng trong ngôn ngữ Smalltalk.

Tính tuyến tính đề cập đến mức độ có thể đọc mã một cách liên tiếp (tuần tự). Ngôn ngữ càng tuyến tính (tuần tự) thì càng dễ phân đoạn và hiểu đoạn mã. Tính tuyến tính đơn giản hoá việc hiểu và bảo trì. Trong ví dụ đoạn mã Cobol được chặt thành các đoạn và thực hiện.

Trong lựa chọn ngôn ngữ độ khó khi biên dịch cũng đóng một vai trò quan trọng. Nói chung, nhiều ngôn ngữ mô tả, ví dụ như SQL, đang được xem xét, cân nhắc trên cơ sở dễ dàng hơn khi dịch ra mã ngữ so với các ngôn ngữ thủ tục như Fortran. Mặc dù vậy, Prolog và các ngôn ngữ suy diễn khác tuy đơn giản trong việc mô tả và phát triển các luật đơn nhưng không tầm thường trong việc quyết định kết hợp các luật để tạo ra các tri thức đúng mới.

Tính hiệu quả của trình biên dịch nằm ở tính hiệu quả của mã assembler nhận được sau khi dịch. Tính hiệu quả đó thay đổi tuỳ theo ngôn ngữ và nhà sản xuất. Tính hiệu quả của trình biên dịch đặc biệt quan trọng khi lập trình một hệ thống máy bay hay các ứng dụng thường trú tương tác với các thành phần hệ thống như là một phần của hệ thống lớn.

Cùng với tính hiệu quả, tính khả chuyển của mã cũng rất quan trọng. Tính khả chuyển là khả năng đáp ứng của mã trên các cơ sở thực hiện khác nhau. Các cơ sở thực hiện bao gồm cả phần cứng, hệ điều hành, hay môi trường thực hiện phần mềm. Khi các ứng dụng dùng chung và phân tán càng phổ biến thì sự cần thiết đối với tính khả chuyển của ngôn ngữ sẽ càng tăng. Lý tưởng nhất, chương trình sẽ thực hiện được ở bất cứ nơi nào, trên bất cứ phần cứng hay hệ điều hành nào.

Tóm lại, nền tảng không đóng vai trò chính để phân biệt ngôn ngữ thì những tính đặc trưng của ngôn ngữ sẽ trở nên quan trọng trong việc lựa chọn ngôn ngữ.

5.4. PHÂN LỚP VÀ ĐÁNH GIÁ VỀ NGÔN NGỮ CÀI ĐẶT

5.4.1. Các lớp ngôn ngữ

Hiện nay có hàng trăm ngôn ngữ lập trình, tuy nhiên theo đánh giá thì người ta chia nó ra làm bốn thế hệ - từ thế hệ thứ nhất đến thế hệ thứ bốn.

+ Các ngôn ngữ thế hệ thứ nhất: là các chương trình được viết theo mã máy hoặc hợp ngữ. Các ngôn ngữ này phụ thuộc vào máy và có mức độ trừu tượng thấp. Ta chỉ nên dùng các ngôn ngữ này khi các ngôn ngữ cấp cao không thể đáp ứng được hay không hỗ trợ yêu cầu của ứng dụng.

+ Các ngôn ngữ thế hệ thứ hai: được phát triển từ cuối những năm 1950 đến đầu những năm 1960, như FORTRAN, COBOL, ALGOL, BASIC,... Nó được xem là nền tảng cho mọi ngôn ngữ lập trình hiện đại - thế hệ thứ ba. Các ngôn ngữ thế hệ thứ hai được đặt trưng bởi việc sử dụng rộng rãi thư viện phần mềm khổng lồ và nó cũng đã được chấp nhận rộng rãi.

+ Các ngôn ngữ thế hệ thứ ba: còn được gọi là ngôn ngữ lập trình hiện đại hay có cấu trúc. Nó được đặc trưng bởi khả năng cấu trúc dữ liệu và thủ tục mạnh. Các ngôn ngữ thuộc thế hệ này như: PASCAL, C, ADA, MODULA-2, C++, C-OBJECTIVE,...

+ Các ngôn ngữ thế hệ thứ tư: Trọng tâm của ngôn ngữ thế hệ thứ tư là nâng mức độ trừu tượng của chương trình lên cao. Các ngôn ngữ này cũng giống như mọi ngôn ngữ nhân tạo khác đều chứa một cú pháp phân biệt để biểu diễn điều khiển và cấu trúc dữ liệu, tuy nhiên nó biểu thị các cấu trúc này ở mức độ trừu tượng cao hơn bằng cách xoá bỏ yêu cầu xác định chi tiết thuật toán. Một số ngôn ngữ thuộc thế hệ thứ tư như ngôn ngữ vấn đáp, ngôn ngữ hỗ trợ quyết định, ngôn ngữ làm bản mẫu,...

5.4.2. So sánh, đánh giá về một số ngôn ngữ cài đặt

Ở đây, chúng ta đánh giá một số ngôn ngữ phổ biến được dùng trong các tổ chức kinh doanh ngày nay như: SQL, Focus, Basic, Cobol, Fortran, C, Pascal, Ada, Prolog, và Smalltalk. Những ngôn ngữ này đại diện cho những kiểu lập trình chủ yếu đã xét ở trên gồm: lập trình thủ tục (Basic, Cobol, Fortran, Pascal), hướng đối tượng (Smalltalk, Ada), xử lý khai báo (SQL, Prolog), các ngôn ngữ thế hệ thứ tư (Focus), và hệ chuyên gia (Prolog).

1. SQL- Structured Query Language

Được xem là chuẩn American National Standards Institute đối với ngôn ngữ hỏi đáp cơ sở dữ liệu, SQL là một ngôn ngữ khá thành công. Ưu điểm của SQL hầu hết không mang tính kỹ thuật: dễ dàng sử dụng, gọn gàng, đồng nhất, cục bộ, tuyến tính, tính khả chuyển và khả năng tự động của các công cụ. Sự đơn giản của ngôn ngữ được thể hiện ở thời gian học ngôn ngữ nhanh đối với những người lần đầu sử dụng ngôn ngữ - người mới học có thể viết câu hỏi trong vòng ít phút. Và thời gian để trở thành thành thạo ít hơn so với các ngôn ngữ cơ sở dữ liệu khác.

Nhiều môi trường hỗ trợ phân tích và thiết kế trên hệ cơ sở dữ liệu logic thông qua các quá trình chuẩn hoá. Các sản phẩm này cũng sinh ra lệnh SQL định nghĩa cơ sở dữ liệu như là kết quả thiết kế logic cơ sở dữ liệu.

2. Focus

Là ngôn ngữ thế hệ bốn bao gồm một Database Engine cùng ngôn ngữ hỏi đáp tương thích với SQL, bộ hiển thị, hệ hỗ trợ đồ hoạ, thiết kế, bảo trì và các tiến trình xử lý thông minh. Focus DB hỗ trợ các mô hình quan hệ, mô hình phân cấp và mô hình mạng, cung cấp một giao diện với nhiều khuôn dạng. Cũng như SQL, mặt mạnh chủ yếu của Focus liên quan tới những đặc trưng phi kỹ thuật của ngôn ngữ, đó là tính cô đọng, tính cục bộ, tính tuyến tính, không bị ràng buộc bởi mã chuyển đổi, tính khả chuyển và tính sẵn dùng của công cụ CASE cho việc phân tích thiết kế dữ liệu. Đôi khi Focus có thể nhập nhằng trong việc biên dịch sự phân cấp dữ liệu hay đa kết nối dữ liệu. Hàng loạt các version của Focus hỗ trợ các khả năng đa người sử dụng. Focus là một ngôn ngữ đã được ngầm định là không hỗ trợ những định nghĩa của người dùng hoặc những tài nguyên khác của người sử dụng.

3. Basic - Beginers All purpose Symbolic Interchange Code

Được đánh giá một ngôn ngữ mạnh, cơ bản, trong ngôn ngữ không có những kỹ thuật phức tạp nhưng có toàn bộ các thành phần sơ đẳng. Basic là một ngôn ngữ dễ

học, dễ viết, có tính thống nhất, chặt chẽ và các hệ thống trợ giúp kiểm tra tự động tốt. Các đặc trưng ngôn ngữ còn lại thay đổi tuỳ thuộc vào các phiên bản Basic khác nhau. Khả năng khả chuyển của Basic kém bởi các lệnh vào ra thường phải thay đổi để phù hợp với môi trường.

Basic hỗ trợ các thao tác lập trình chuẩn với một số giới hạn, cùng một số kiểu dữ liệu nhưng không có chức năng kiểm tra kiểu. Cấu trúc ngôn ngữ bao gồm các phép lặp, điều kiện và xử lý mảng, đọc/viết các file.

4. Cobol- Common Business Oriented Language

Là một ngôn ngữ được sử dụng nhiều trong lịch sử máy tính. Cobol được ví như một chiếc xe bus, lập trình Cobol mất nhiều thời gian, nhưng nó lại phù hợp với một số vấn đề thương mại. Như một ngôn ngữ đa mục đích, Cobol cung cấp tất cả các chức năng cơ bản.

Các tiến trình vào/ra của Cobol rất hiệu quả, có tính thống nhất cao và hỗ trợ hầu hết các loại dữ liệu. Ngôn ngữ Cobol không phù hợp cho những ứng dụng thời gian thực hay các ứng dụng đệ quy.

Trong các đặc trưng phi kỹ thuật, Cobol có tính sẵn dùng cao của công cụ trợ giúp, bộ sinh mã, và các chương trình kiểm tra. Như hầu hết các ngôn ngữ thông dụng khác Cobol là ngôn ngữ đầu tiên được phát triển hỗ trợ tự động. Đây là ngôn ngữ có tính tự động cao và được hỗ trợ bởi nhiều trình biên dịch. Trong các đặc trưng phi kỹ thuật khác, Cobol thường kém hơn SQL và Focus nhưng cũng tốt hơn nhiều các ngôn ngữ thủ tục khác.

5. Fortran - Formula Tranlastion

Là một ngôn ngữ của những năm 60. Điểm yếu của Fortran là trong lĩnh vực xử lý dữ liệu và hỗ trợ cấu trúc file. Fortran không được tích hợp với các phần mềm DBMS các giới hạn về tuần tự... Vì thế các quá trình vào ra của Fortran thường bị giới hạn nhiều so với các ngôn ngữ khác.

Điểm mạnh của Fortran là tính hiệu quả trong giải thuật sinh mã để thực hiện quá trình xử lý số. Chương trình dịch của Fortran thường được hỗ trợ bởi một thư viện các chương trình con chứa nhiều thuật toán ngắn được sử dụng thường xuyên, các quá trình thiết kế và xử lý toán học. Các chương trình con này được thiết kế để dễ dàng định nghĩa và sử dụng các biến tổng thể và các biến cục bộ. Sự xáo trộn các dạng dữ liệu trong Fortran là rất quan trọng bởi vì quá trình xử lý số sẽ cho kết quả khác nhau tuỳ thuộc vào định nghĩa những trường dữ liệu được xử lý.

6. C

C là một ngôn ngữ cấp cao được phát triển để thực hiện các xử lý cấp thấp. Một chương trình viết bằng C là một dãy các hàm và chúng được truy cập đến bởi một tên của chúng trong mã của chương trình.

C là một ngôn ngữ ngắn gọn, xúc tích và khó hiểu vì thế nó chỉ thực sự hiệu quả cho những người lập trình có nhiều kỹ năng và kinh nghiệm về lập trình và có thể sẽ không mang lại hiệu quả cao cho những người lập trình viên kém.

7. Pascal

Pascal là một ngôn ngữ được thiết kế rất rõ ràng và được dùng làm tài liệu giảng dạy cho sinh viên của ngành khoa học máy tính. Một chương trình viết bằng Pascal thường có một khuôn dạng rất thoải mái và Pascal lại có cấu trúc cú pháp tự nhiên cho nên Pascal trở thành ngôn ngữ rất dễ đọc.

Trong thời điểm hiện tại Pascal đã được cung cấp những tiến trình điều khiển thời gian thực. Tuy nhiên Pascal chuẩn không cung cấp những thư viện thông thường bởi vì hồi đó người ta đều cho rằng tất cả các module chương trình được viết thành một chương trình có nghĩa là mã của chương trình đó nằm trong khuôn khổ một chương trình đơn. Trong Pascal có một số điều khiển nhỏ thực hiện các tiến trình ngắt. Tiến trình vào ra được giới hạn hơn so với một số ngôn ngữ, không hỗ trợ truy cập ngẫu nhiên và rất hạn chế trong việc xử lý xâu.

8. Prolog - Programming in Logic

Là một ngôn ngữ được phát triển riêng cho lĩnh vực trí tuệ nhân tạo. Prolog được phát triển bởi một trường đại học ở Marseiller từ rất sớm (những năm 70) nhưng được phát triển rộng rãi ở Mỹ bởi David Warren. Prolog là một ngôn ngữ hướng mục đích, một ngôn ngữ đặc tả với cấu trúc là những mệnh đề và các luật.

Prolog mệnh đề là những thành phần cụ thể của thông tin thực. Prolog luật được định nghĩa từ mệnh đề được giả định để tạo thông tin.

9. Smalltalk

Smalltalk được phát triển như là môi trường điều hành và ngôn ngữ lập trình vào những năm 70 tại trung tâm nghiên cứu Xerox Palo Alto bởi nhóm Learning Research. Đó là một ngôn ngữ hướng đối tượng, coi mọi thứ như là đối tượng, thậm chí đối với thể hiện, các số nguyên. Smalltalk được tối ưu ở mức cao và do vậy, được sử dụng để thiết kế các ứng dụng có hiệu quả.

Smalltalk có đầy đủ các chức năng, là ngôn ngữ lập trình có thể làm được mọi việc không hạn chế. Điểm yếu chủ yếu của Smalltalk là không hỗ trợ các đối tượng liên tục như là file. Nhưng nếu file được coi là một đối tượng, thì nó có thể được xử lý trong Smalltalk .

Điểm mạnh của Smalltalk là nó được sử dụng trong các quá trình xử lý hướng sự kiện như trong điều khiển tiến trình, việc điều khiển hệ thống điều hoà nhiệt độ, hoặc là sự thông báo kịp thời nhu cầu sản xuất. Các ứng dụng loại này sử dụng các thông điệp không liên tục từ môi trường bên ngoài để điều khiển quá trình xử lý thực hiện bởi ứng dụng.

10. Ada

Ada, ngôn ngữ lập trình chính thức của Bộ Quốc phòng Mỹ với hàng trăm nghìn người sử dụng, có một lối tư duy khác về cách lập trình so với các ngôn ngữ khác.

Ada được thiết kế bởi một hội đồng và không tạo thành một ngôn ngữ hoàn thiện nhưng nó lại tốt hơn tất cả. Phiên bản hiện hành của Ada là dựa trên đối tượng

hơn là hướng đối tượng. Trong các ứng dụng dựa trên các đối tượng, các chương trình cùng hoạt động trên một tập hợp các đối tượng, mỗi tập hợp đại diện một thể hiện của một vài kiểu đối tượng. Tất cả các kiểu đối tượng là thành phần của mô hình phân cấp các kiểu mà chúng được kết nối thông qua quá trình xử lý hơn là việc kế thừa các quan hệ. Các lớp thường khó phân biệt với các kiểu bởi vì không có các đối tượng nhất quán như là file và nó không hỗ trợ kế thừa.

Khái niệm file trong Ada giống như trong Smalltalk được định nghĩa là một kiểu trong cấu trúc của ngôn ngữ và mọi quá trình xử lý hiện trên các kiểu. Giống như Smalltalk, sức mạnh của Ada là khả năng của nó trong việc hỗ trợ xử lý hướng sự kiện như tên lửa dẫn đường trong hệ thống phòng thủ quốc gia.

Phiên bản tương lai của Ada có thể đáp ứng cấu trúc thừa kế và xử lý đa lớp và sự liên kết động các đối tượng, xử lý thông điệp thực nhất quán và các đối tượng nhất quán cung cấp các cấu trúc dữ liệu đa dạng. Với sự mở rộng này ngôn ngữ Ada thích hợp với các ứng dụng ảo. Một sự lưu ý tương tự về sự khác nhau trong tư duy hướng đối tượng như đã chỉ ra trong phần Smalltalk: thiết kế hướng đối tượng và phát triển chương trình khác nhau về cơ bản hơn là sự phát triển các ứng dụng thủ tục thông thường với các ngôn ngữ như là Cobol.

5.4.3. Chọn ngôn ngữ cho ứng dụng

Khi đã làm việc trên ứng dụng ta không có sự lựa chọn về ngôn ngữ. Nhưng nếu ban đầu chọn sai ngôn ngữ thì chúng ta phải liên tục sửa đổi yêu cầu để phù hợp với những giới hạn của ngôn ngữ. Việc lựa chọn ngôn ngữ lập trình phải xem ngôn ngữ lập trình đó có phù hợp với kiểu ứng dụng hay không và xem nó có phù hợp với việc dùng để phát triển ứng dụng.

Dựa vào các đánh giá ở 5.4.2, ta có bảng thống kê về việc lựa chọn ngôn ngữ lập trình dựa trên các tiêu chí của ứng dụng như sau

Kiểu ứng dụng	**SQL**	**Focus**	**Basic**	**Cobol**	**Fortran**	**C**	**Pascal**	**Prolog**	**Ada**	**Smalltalk**
Lô	X	X	X	X						
Trực tuyến	X	X	X	X		X	X		X	X
Thời gian thực						X			X	X
Hỏi đáp CSDL	X	X				X			X	X
Hỗ trợ quyếtđịnh	X	X			X	X	X	X	X	X
Hệchuyên gia								X		
EIS		X				X			X	X

5.5. HIỆU QUẢ CỦA CHƯƠNG TRÌNH VÀ TẦM QUAN TRỌNG

Trước hết, tính hiệu quả nó là một yêu cầu hoàn thiện và nó được thiết lập trong phân tích yêu cầu phần mềm. Thứ hai là nó được thiết kế tốt, sau đó mới đến tính hiệu quả của chương trình đi đôi với tính đơn giản của chương trình. Cần lưu ý rằng không được bỏ qua tính rõ ràng, dễ đọc hay đúng đắn để có được sự cải thiện vô nghĩa về tính hiệu quả.

Có hai lý do cơ bản để nghiên cứu về việc tăng hiệu suất của chương trình. Thứ nhất là tầm quan trọng mang tính bản chất của nó trong nhiều ứng dụng: chi phí về thời gian và bộ nhớ. Lý do thứ hai là vấn đề giáo dục - đây là một mảnh đất tốt để đào tạo. Trong giới hạn ở đây, chúng ta chỉ bàn luận về các vấn đề cho việc tăng tính hiệu quả thực tế - có thể đo được - của chương trình.

5.6. MỘT SỐ VẤN ĐỀ TRONG CẢI TIẾN HIỆU SUẤT

Như đã đề cập ở trên, ta sẽ quan tâm đến các vấn đề để giảm thời gian chạy và chi phí bộ nhớ cho chương trình.

Để minh hoạ cho các luận cứ được nêu về các vấn đề trên, ở đây, ta sẽ chỉ ra các vấn đề quan tâm thông qua các bài toán ví dụ minh hoạ.

5.6.1. Tốc độ xử lý

Trong hầu hết các trường hợp, tốc độ của chương trình là quan trọng như các ứng dụng thời gian thực, ứng dụng về xử lý trên các cơ sở dữ liệu lớn,... Để một ứng dụng có tốc độ nhanh, người lập trình chúng phải quan tâm đến nhiều yếu tố như: thuật toán sử dụng, lựa chọn cấu trúc dữ liệu, tinh chế mã cho chương trình,...

5.6.1.1. Thuật toán sử dụng

1. Xác định lại bài toán

Yêu cầu: Trước khi bắt tay vào giải bài toán, hãy tìm hiểu kỹ các yêu cầu mà bài toán đặt ra và tận dụng mọi điều đã biết từ bài toán.

Bài toán minh hoạ: Cho một mảng số nguyên gồm 1.000.000 phần tử; các giá trị nằm trong khoảng từ 0..10 một cách ngẫu nhiên. Hãy sắp xếp để được một mảng có thứ tự giảm dần.

- Giải bài toán tổng quát: là một bài toán sắp xếp; dùng một đoạn chương trình sắp xếp có sẵn của hệ thống hay sử dụng một thuật toán sắp xếp có sẵn như Insert - Sort hay Quick - Sort chẳng hạn. Chi phí về độ phức tạp là $o(n^2)$ hay o(nlogn).

- Tuy nhiên, ta đã bỏ qua một tính chất của bài toán đó là các giá trị chỉ nằm trong khoảng 0..10. Sau khi nghiên cứu bài toán ta quyết định sử dụng thuật toán đếm cho việc sắp xếp bài toán.

  + Khởi tạo 10 biến nguyên với giá trị 0.

+ Vcới mỗi giá trị i trong mảng, tăng biến thứ i lên một đơn vị
+ Thực hiện rải giá trị cho mảng ứng với số lần là giá trị của biến thứ i;
Như thế, chi phí về độ phức tạp của bài toán là o(n).

2. Sức mạnh của thuật toán

Yêu cầu: Việc nghiên cứu thuật toán giúp ích rất nhiều cho các nhà lập trình. Các thuật toán có ảnh hưởng quan trọng đến các hệ thống phần mềm và đặc biệt chúng tăng nhanh tốc độ vận hành.

Bài toán minh hoạ: Quay mảng một chiều chứa N phần tử về bên trái I một vị trí. Với N = 8; I = 3 ta được mảng ABCDEFGH sẽ quay thành DEFGHABC.

- Thuật toán 1: Ta có thể giải bài toán bằng cách sao I phần tử đầu tiên của mảng sang một mảng đoạn; dịch chuyển N - I phần tử còn lại của mảng về bên trái I vị trí; sau đó sao I phần tử đầu tiên từ mảng tạm về cuối mảng. Trong trường hợp N và I lớn, như thế việc cần mảng tạm khá tốn bộ nhớ; xét trong trường hợp bộ nhớ của máy không dồi dào thì giải quyết như thế nào?

- Thuật toán 2: Ta cũng có thể viết 1 thủ tục con quay mảng X sang bên trái một vị trí (như thế sẽ giải quyết được vấn đề tốn bộ nhớ vì chỉ cần dùng một biến phụ) và sau đó thực hiện thủ tục trên I lần. Tuy nhiên, thủ tục trên tốn thời gian tỉ lệ với N và như thế chương trình sẽ tỉ lệ với thời gian I*N; do vậy khi N và I lớn thì đây là điều không thể thực hiện được.

- Thuật toán 3: Để ý rằng: khi quay mảng X gồm N phần tử về I vị trí, (giả sử quay sang trái), lúc này phần tử X[i+1] sẽ là phần tử X'[1] (ký hiệu X'[i] là phần tử thứ i của mảng X sau khi quay); X[2i+1] sẽ là phần tử X'[i+1],....và cứ thế tiếp tục. Do đó ta có thuật toán sau:
  + Dịch chuyển X[1] đến 1 biến tạm T
  + Dịch chuyển X[i+1] và X[1];
  + Dịch chuyển X[2+i+1] và X[i+1],....
  .....
  + Quá trình cứ thế tiếp tục; chỉ số được tính theo module của N tức khi vượt quá N sẽ chia lấy dư cho N.
  + Quá trình lặp cho đến khi gặp phần tử X[1] và lúc này dùng giá trị từ biến T và quá trình chấm dứt.

  Ví dụ: Với trường hợp N = 8, I = 3 $\Rightarrow$ mảng X = ABCDEFGH ta có như sau:
  X = ABCDEFGH; N = 8; I = 3; T = A
  + Dịch chuyển X[i+1]: X[4] $\rightarrow$ X[1]
  + Dịch chuyển X[2i+1]: X[7] $\rightarrow$ X[4]
  + Dịch chuyển X[3i+1]: X[10] $\Rightarrow$ X[2] $\rightarrow$ X[7]
  + Dịch chuyển X[4i+1]: X[13] $\Rightarrow$ X[5] $\rightarrow$ X[10] $\Rightarrow$ X[2]
  ....
  Quá trình được tóm tắt như sau:
  T = A; N = 8; I = 3; X = ABCDEFGH
  4 $\rightarrow$ 1..... DBCDEFGH

7 → 4.....DBCGEFGH
(10)2 → 7 .DBCGEFBH
5 → 2.....DECGEFBH
8 → 5.....DECGHFBH
(11)3 → 8..DECGHFBC
6 → 3(11)...DEFGHFBC
(9)1 → 6 ...DEFGHABC
T => Dừng

Đây là thuật toán có chi phí vùng nhớ không lớn và thời gian chạy chấp nhận được.

3. Các kỹ thuật thiết kế thuật toán và tinh chế thuật toán.

Yêu cầu: Thực hiện theo các nguyên tắc sau:

- Lưu trữ các trạng thái cần thiết để tránh tính lại,
- Tiền xử lý thông tin để đưa vào các cấu trúc dữ liệu,
- Sử dụng các thuật toán thích hợp,
- Chỉ ra được cận dưới của thuật toán,
- Sử dụng các kết quả được tích luỹ,...

Bài toán minh hoạ: Cho vector X chứa N số thực X[1], X[2], ...,X[N]. Gọi vector con của X là vector mà phần tử của nó là các phần tử liên tiếp trong X. Tổng của một vector được tính là tổng các phần tử của vector đó. Tính tổng lớn nhất trong các vector con, tức tìm L,U∈1..N để tổng X[i], i∈L..U là lớn nhất.

Để đơn giản, vectơ con có tổng lớn nhất được gọi tắt là vectơ lớn nhất. Ví dụ, nếu vectơ đầu vào có dạng:

31	-41	59	26	53	58	97	-93	3	84
		↑				↑			
		3				7			

Lúc này, kết quả của bài toán là tổng của vectơ X[3..7]. Bài toán rất đơn giản khi tất cả các số là số dương - khi đó kết quả chính là bản thân vectơ X. Vấn đề sẽ phức tạp hơn khi có thêm các số âm. Chúng ta nhận xét rằng nếu tất cả đều là số âm thì kết quả là bằng 0 (đó chính là tổng của vectơ rỗng).

- Thuật toán 1: Một chương trình có thể viết ngay được là xét tất cả các cặp số nguyên L và U thoả mãn 1<= L<= U<= N; đối với mỗi cặp như vậy ta tính tổng của vectơ con X[L..U] và so sánh tổng này với giá trị lớn nhất hiện có. Ta có giã mã cho thuật toán 1 như sau:

```
MaxSoFar := 0;
For L := 1 to N do
    For U := L to N do
       Begin
            Sum := 0;
            For I := L to U do
```

```
                    Sum := Sum + X[I];
                    /* Sum chứa tổng của X[L..U] */
                MaxSoFar := Max(MaxSoFar, Sum);
            End;
```

Chương trình này ngắn và dễ hiểu, tuy nhiên điều không may là nó chạy rất chậm. Độ phức tạp của chương trình là $o(n^3)$.

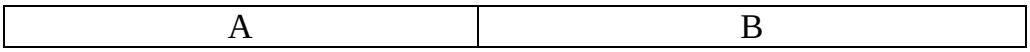
- Thuật toán 2: Đối với thuật toán 1, đa số người lập trình cho rằng có thể viết chương trình chạy nhanh hơn. Có hai cách như vậy. Các cách này đều có độ phức tạp $o(n^2)$. Thuật toán thứ nhất tính nhanh các tổng của vectơ con bằng cách sử dụng hệ thức: Tổng của X[L..U] = Tổng của X[L..U-1] + X[U]; Ta có thuật toán 2 như sau:

```
MaxSoFar := 0.0;
For L := 1 to N do
    Begin
        Sum := 0.0;
        For U := L to N do
            Begin
                Sum := Sum + X[U];
                /* Sum chứa tổng của X[L..U] */
                MaxSoFar := Max(MaxSoFar, Sum);
            End;
    End;
```

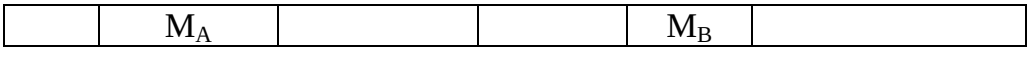
Các lệnh trong vòng lặp thứ nhất thực hiện N lần. Với mỗi lần thực hiện các lệnh trong vòng lặp thứ nhất, các lệnh trong vòng lặp thứ hai thực hiện nhiều nhất là N lần. Vậy ta có độ phức tạp là $o(n^2)$.

- Thuật toán 3: Thuật toán chia để trị: "Để giải bài toán kích thước N, chúng ta giải một cách đệ quy hai bài toán con kích thước khoảng N/2, kết hợp lời giải của chúng để tạo ra lời giải của toàn bộ bài toán".

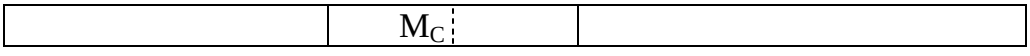
Trong trường hợp này bài toán của ta là xử lý vectơ độ dài N, do đó một cách tự nhiên là chia vectơ này thành hai vectơ con có độ dài gần bằng nhau. Chúng ta gọi hai vectơ này là A và B.



Sau đó, bằng đệ quy chúng ta tìm vectơ con lớn nhất trong A và B gọi là MA và MB.



Để ý rằng kết quả bài toán là giá trị lớn nhất trong hai tổng của vectơ MA và MB. Kết quả của bài toán có thể là tổng của vectơ MC chứa đồng thời các thành phần của A và B. Ta gọi là vectơ con như vậy là vectơ vượt biên.



Như vậy thuật toán chi để trị sẽ tính MA,MB bằng đệ quy và tính MC bằng phương pháp khác, kết quả bài toán này là giá trị lớn nhất trong ba tổng của ba vectơ này. Các mô tả trên là gần đủ để viết chương trình. Chúng ta còn phải mô tả cách quản lý các vectơ nhỏ và cách tính vectơ MC. Phần đầu tiên rất dễ: Đối với vectơ chỉ chứa một phần tử, vectơ con lớn nhất hoặc là chính nó hoặc là vectơ rỗng trong trường hợp phần tử của vectơ đó là số âm, và vectơ con lớn nhất của vectơ rỗng cũng là vectơ rỗng. Để tính MC, chúng ta nhận xét rằng thành phần của vectơ MC nằm trong vectơ A là vectơ con lớn nhất trong tất cả các vectơ con của vectơ A, bắt đầu từ biên của A và B. Tương tự như thế đối với thành phần của vectơ MC nằm trong vectơ B. Kết hợp tất cả các yếu tố này, chúng ta có thuật toán 3, được gọi bởi lệnh:

```
Answer := MaxSum(1,N);
Recursive Function MaxSum(L,U)
   Begin
      if L > U then  return 0;  /* vectơ rỗng */
      if L = U then  return Max(0.0, X[L]); /* vectơ một phần tử */
      M := (L+U) div 2
                                   /* A là vectơ X[L..M], B là vectơ X[M+1..U]*/
       /* Tìm giá trị lớn nhất của tổng các thành phần bên trái (trong vectơ A)
                                                   của vectơ vượt biên */
      Sum := 0;
      MaxToLeft := 0;
      For I := M downto L do
            Begin
                  Sum := Sum + X[I]
                  MaxToLeft := Max(MaxToLeft, Sum)
            End;

      /* Tìm giá trị lớn nhất của tổng các thành phần bên phải (trong vectơ B)
                                                   của vectơ vượt biên */
      Sum := 0;
      MaxToRight := 0;
      for I := M +1 to U do
            Begin
                  Sum := Sum + X[I]
                  MaxToRight := Max(MaxToRight, Sum)
            End;
      MaxCrossing := MaxToLeft + MaxToRight;
      MaxInA := MaxSum(L,M);
      MaxInB := MaxSum(M+1,U);
      Return Max(MaxCrossing, MaxInA, MaxInB);
  End;
```

Thuật toán thực hiện o(n) công việc trong mỗi mức đệ quy, và có tất cả là o(logn)mức đệ quy. Nên chương trình này giải quyết bài toán với độ phức tạp o(nlogn).

- Thuật toán 4: Thuật toán quét: Giả sử rằng chúng ta đã giải bài toán cho vectơ X[1..I-1]; làm thế nào để mở rộng kết quả này cho bài toán với vectơ X[1..I]? Lý luận tương tự như trong thuật toán "chia để trị": tổng lớn nhất trong vectơ X[1..I-1] (gọi là MaxSoFar), hoặc tổng lớn nhất trong tất cả các tổng của vectơ con kết thúc tại I (gọi là MaxEndingHere).

	MaxSoFar		MaxEndingHere

Nếu chúng ta tính MaxEndingHere bằng cách tương tự như trong thuật toán 3, thì ta chỉ có một thuật toán bình phương (có độ phức tạp $o(n^2)$). Để làm nhanh hơn, chúng ta nhận xét điều như sau: vectơ con lớn nhất kết thúc tại vị trí I là vectơ con lớn nhất kết thúc tại vị trí I-1 được bổ sung thêm phần tử X[I] ở cuối hoặc là vectơ rỗng trong trường hợp tổng của vectơ nhận được là số âm. Ta có thuật toán 4 như sau:

```
MaxSoFar = 0;
MaxEndingHere = 0;
For I := 1 to N do
  Begin
                    /* Bất biến: MaxEndingHere và MaxSoFar là đúng đối với
                                                              X[1..I-1]*/
      MaxEndingHere := Max(MaxEndingHere + X[I],0);
      MaxSoFar := Max(MaxSoFar, MaxEndingHere);
  End;
```

Chương trình này có thời gian chạy là o(n). Vì vậy thuật toán này được gọi là thuật toán tuyến tính.

Như vậy, khi xây dựng ứng dụng, việc sử dụng các thuật toán phù hợp làm giảm thời gian chạy chương trình một cách đáng kể.

5.6.1.2. Lựa chọn cấu trúc dữ liệu

Song song với thuật toán, việc chọn lựa cấu trúc dữ liệu ảnh hưởng lớn đến hiệu suất chương trình và nó tác động đến bản thân thuật toán bởi cấu trúc dữ liệu gắn bó mật thiết với thuật toán.

Việc chọn đúng đắn cấu trúc dữ liệu làm giảm không gian bộ nhớ, giảm thời gian chạy, tăng tính chuyển đặc và dễ bảo trì, đặc biệt là các cấu trúc dữ liệu cao cấp, mặc dầu chúng không thường được dùng nhưng khi cần thiết thì không thể thiếu chúng được.

Ta sẽ gặp lại việc chọn lựa cấu trúc dữ liệu trong phần 5.6.2. ở sau, trong phần xét về không gian bộ nhớ chương trình.

5.6.1.3. Tinh chế mã

Thông thường, để tăng tính hiệu quả của chương trình, người ta thường bàn về các tiếp cận bậc cao như: định nghĩa bài toán, cấu trúc hệ thống, thiết kế thuật toán và chọn cấu trúc dữ liệu.

Tuy nhiên, các tiếp cận bậc thấp như tinh chế mã mà nó thường được thực hiện ở những phần tốn kém của chương trình để cải tiến hiệu suất. Mặc dù đây là phương pháp không phải lúc nào cũng cần thiết nhưng đôi lúc nó tạo ra khác biệt lớn trong hiệu suất của chương trình.

Các phương pháp thường dùng của tinh chế mã.
+ Tính trước các giá trị,
+ Thay tương đương,
+ Dùng biến trung gian thích hợp, không tính lại các hằng trong vòng lặp.

Bài toán: Cho một chuỗi gồm 1 triệu ký tự. Hãy phân loại mỗi ký tự theo 4 kiểu sau: kiểu chữ in, kiểu chữ hoa, kiểu số hay là các kiểu "khác".

- Lời giải mà ta thường làm: là thực hiện các so sánh đối với mỗi ký tự. Như vậy, trong bảng mã ASCII, để xác định mỗi ký tự thuộc loại nào phải mất rất nhiều lần so sánh; và đây chính là điểm "nóng" của chương trình.

- Tinh chế mã: Ở đây, nếu ta xem mỗi ký tự như là một chỉ số của mảng mà thành phần của nó là các kiểu ký tự. Như vậy, kiểu ký tự C là mảng [C] và để xác định kiểu của một ký tự, ta chỉ cần truy cập đến một mảng đơn giản thay vì phải thực hiện các chuỗi so sánh phức tạp.

Như vậy, khi thực hiện tinh chế mã, cần xác định ở đâu là điểm "nóng" của chương trình và hãy tập trung vào điểm nóng. Hơn nữa, ta đã biết rất nhiều phương pháp để cải tiến hiệu suất của chương trình và hãy dùng đến phương pháp này sau cùng và đôi khi, phương pháp này còn được dùng để làm giảm không gian chiếm bởi chương trình.

5.6.2. Không gian bộ nhớ

Trong những ngày đầu của kỹ thuật máy tính, các nhà lập trình bị hạn chế bởi những bộ nhớ nhỏ; Ngày nay vấn đề này không còn là điểm "nóng" nữa. Tuy vậy, khi thiết kế chương trình không phải lúc nào ta cũng có đủ bộ nhớ để sử dụng bởi nhiều lý do khác nhau.

5.6.2.1. Không gian dữ liệu

Nguyên tắc để làm giảm không gian lưu trữ dữ liệu.
+ Đảm bảo tính đơn giản,
+ Trong một số trường hợp đừng lưu trữ, hãy tính lại khi cần thiết,
+ Đặc biệt, việc nghiên cứu kỹ các cấu trúc dữ liệu, (thường là cấu trúc dữ liệu thưa thớt) sẽ làm giảm nhiều không gian cần thiết để lưu trữ các thông tin cho trước,
+ Nén dữ liệu sau đó giải nén khi dùng,
+ Sử dụng các nguyên tắc cấp phát bộ nhớ: chẳng hạn như cấp phát bộ nhớ động,...

Xét bài toán: Trên một bản đồ chứa 2.000 điểm (bảng đồ quân sự) được đánh số từ 1 đến 2.000. Một vị trí trên bảng đồ được xác định bằng cặp tọa độ (x,y) với x là

số nguyên nằm trong khoảng 1...200; y là số nguyên nằm trong khoảng 1...150. Chương trình dùng cặp (x,y) để xác định điểm nào (nếu có) được chọn trong 2000 điểm đã cho.

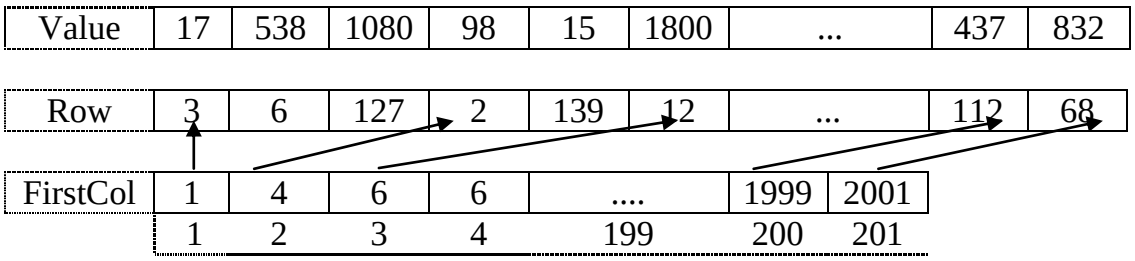
- Không gian lưu trữ 1: Một cách hiển nhiên để lưu trữ bản đồ trên là dùng mảng 2 chiều 200 x 150 số nguyên; ứng với mỗi x.y sẽ chứa một giá trị trong khoảng từ 1...2.000 hay sẽ chứa giá trị 0.

...									
6	538								
5				965					
4						1121			
3	17								
2		98						162	
1	1	2	3	4	5	6	7	8	...

Việc dùng bảng này, thời gian truy cập nhanh; nhưng nó chiếm đến 200 x 150 = 30.000 ô nhớ; và giả sử để lưu trữ dữ liệu (ở đây là 1...2.000) cần 2 byte thì ta cần 60.000B bộ nhớ.

Tuy nhiên, trong mảng trên thì đa số là giá trị 0 (giá trị không dùng). Do vậy, nếu ta dùng cái nào chỉ lưu trữ các giá trị cần thiết (ở đây là 2.000 giá trị) thì việc chiếm bộ nhớ sẽ giảm đáng kể.

- Không gian lưu trữ 2: Thay vì dùng một mảng 2 chiều ở trên, ta sử dụng 3 mảng 1 chiều như sau:



Mảng value dùng để chứa các giá trị (ở đây là 1...2.000) theo từng cột, như vậy cần 2.000 ô nhớ; Mảng Row là mảng để chứa hàng tương ứng với giá trị ở mảng value; như thế cần 2.000 ô nhớ. Mảng FirstCol là mảng để chứa số cột hiện có - tương ứng với giá trị ở mảng value; như thế cần 200 ô nhớ; thêm 1 ô nhớ đầu tiên để đánh dấu, nên tổng cộng cần 201 ô.

Các điểm trong cột I được biểu diễn bằng các phần tử trong mảng Row và Value giữa các vị trí FirstinCol [I] và FirstinCol [I+1]-1. Ở đây, FirstinCol [201] được xác định (mặc dù chúng ta chỉ có 200 cột) để biểu thức I+1 hợp lệ. Theo hình trên, chúng ta có 3 điểm trong cột thứ nhất điểm 17 ở vị trí (1,3), điểm 538 ở vị trí (1,6,), điểm 1053 ở vị trí (1,127). Có hai điểm trong cột 2 (điểm 98 ở vị trí (2,2), điểm 15 ở vị trí (2,139)), cột 3 không có điểm nào, và có 2 điểm trong cột 200. Để xác định điểm nào (trong số 2000 điểm) được lưu giữ tại vị trí (I,J), chúng ta dùng giải mã sau:

```
For K := FirstinCol[I]  to  FirstinCol[I+1]  do
    If   Row[K] = J  then
            /* Tìm thấy ở vị trí */
         Return Value[K]
            /* Không có điểm nào tại vị trí (I,J) */
    Else
         Return 0;
```

Phương pháp này dùng ít không gian hơn nhiều so với phương pháp trước đó. Ở đây chúng ta dùng hai mảng 2000 phần tử và một mảng 201 phần tử (như vậy tất cả là 4201 từ 16 bit thay vì 30000 từ trong phương pháp trước đó). Mặc dù nó hơi chậm hơn (một lần truy nhập phải mất 150 lần so sánh trong trường hợp tồi nhất, nhưng trung bình chỉ cần 6 lần), nhưng chương trình chạy tốt và người dùng không gặp phải vấn đề gì.

Lời giải này minh hoạ một số điểm tổng quát về cấu trúc dữ liệu. Vấn đề ở đây rất cổ điển: việc biểu diễn thưa thớt (tức là mảng trong đó hầu hết các phần tử có cùng một giá trị, thường là giá trị 0). Lời giải trên có ý tưởng rất đơn giản và dễ cài đặt bằng mảng. Chú ý rằng ở đây không có mảng LastinCol đi cùng với mảng FirstinCol, bởi vì chúng ta sử dụng một điều là điểm cuối cùng trong một cột chính là điểm đứng trước điểm đầu tiên của cột tiếp theo. Đay là một ví dụ tầm thường của nguyên tắc tính lại thay vì lưu trữ. Tương tự, chúng ta không có mảng Col đi cùng với mảng Row vì chúng ta chỉ truy nhập mảng Row thông qua mảng FirstinCol, do đó chúng ta luôn biết cột hiện thời là cột nào.

Nhiều kỹ thuật sử dụng cấu trúc dữ liệu khác cũng có thể làm giảm không gian. Trong thực tế, chúng ta tiết kiệm không gian bằng cách thay thế mảng 3 chiều thành mảng 2 chiều. Nếu chúng ta sử dụng một khoá được lưu trữ như là chỉ số của mảng, thì chúng ta không cần phải lưu trữ bản thân khoá này; thay vào đó, chúng ta chỉ cần lưu trữ các thuộc tính thích hợp của nó, chẳng hạn như số lần xuất hiện của nó. Thêm vào đó, các ứng dụng của kỹ thuật đánh chỉ số bằng khoá đã được vận dụng. Trong ví dụ về ma trận thưa thớt trên đây, việc đánh chỉ số bằng khoá thông qua mảng FirstinCol cho phép chúng ta giải quyết vấn đề mà không cần dùng đến mảng Col.

5.6.2.2. Không gian chương trình

Trong một số chương trình, đôi lúc thì kích thước của chính bản thân nó là vấn đề. Hãy định nghĩa các chương trình con hay sử dụng các bộ thông dịch chuyên dụng để làm cho chương trình đơn giản, trong sáng hơn làm cho nó rõ ràng hơn và dễ bảo trì.

5.6.3. Lựa chọn hệ thống và phần cứng

Nên lựa chọn các ngôn ngữ lập trình phù hợp với ứng dụng của bạn. Đôi lúc cần hãy thay thế các chương trình con viết trên ngôn ngữ khác để có tốc độ lớn hơn.

Trong xu thế phát triển của phần cứng hiện nay, cần phải tận dụng thế mạnh của phần cứng để có hiệu suất của chương trình cao, mặc dù điều này làm hạn chế tính phổ cập của nó nhưng hiện nay yêu cầu về phần cứng cao là chấp nhận được.

5.7. CÔNG CỤ TRỢ GIÚP VÀ PHÂN LOẠI

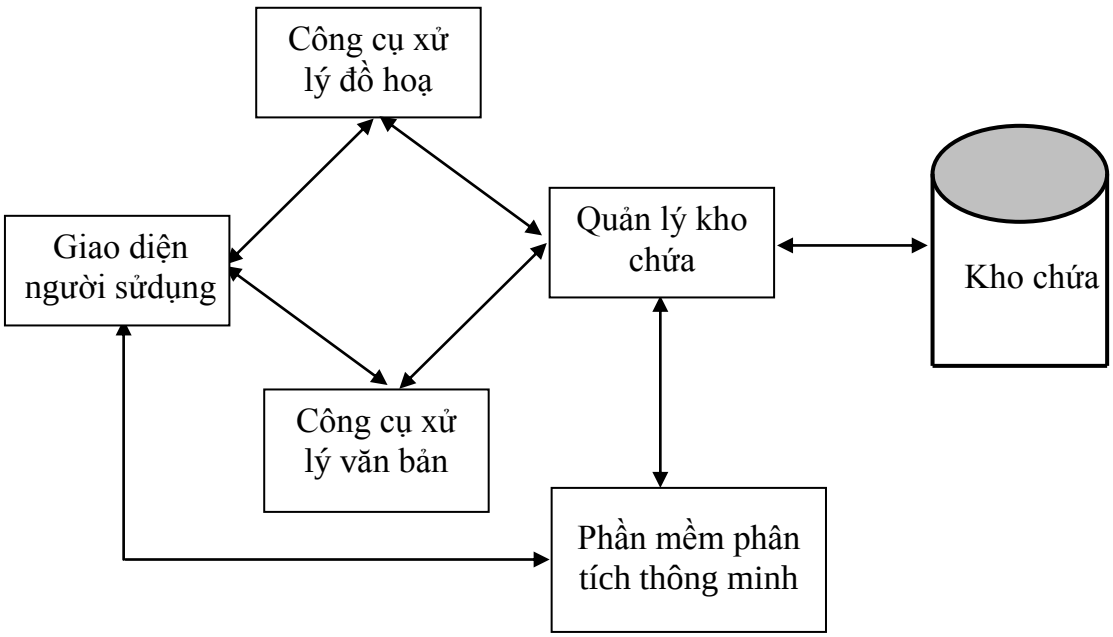
Mọi người đều biết sự đắt đỏ và các khó khăn của việc phát triển phần mềm. Việc cung cấp cho các kỹ sư phần mềm các công cụ trợ giúp, nó cung cấp khả năng tự động phát triển chương trình là rất hiệu quả về mặt chi phí. Trong thời đại nhiều máy móc như hiện nay, những trợ giúp phát triển thường có sẵn: môi trường CASE, sinh mã và những trợ giúp kiểm tra như gỡ rối, các chương trình biên dịch tăng, môi trường thực hiện theo kiểu cửa sổ và cả tốc độ phát triển mã hoạt động. Bất kỳ ngôn ngữ nào đều có sự trợ giúp tự động,

Như thế, một môi trường phát triển phần mềm là một bộ các công cụ phần cứng và phần mềm chúng được kết lại để trợ giúp cho việc sản sinh ra một hệ thống phần mềm trong một miền ứng dụng chuyên biệt.

5.7.1. Công cụ CASE

CASE có các loại ICASE, Upper CASE và Lower CASE. ICASE nghĩa là "Intergrated" CASE hay là CASE tích hợp, "Upper" nghĩa là công cụ ý tưởng hay chỉ là thiết kế logic, "Lower" nghĩa là công cụ chỉ hỗ trợ lập trình. Trong những công cụ trên, tài liệu này tập trung vào các công cụ CASE hỗ trợ tối thiểu một giai đoạn phân tích và có thể hỗ trợ các giai đoạn khác; chúng được gọi chung là "CASE". Chúng ta cũng xác định những giai đoạn hiện nay đang được hỗ trợ (dĩ nhiên là các giai đoạn này có thể thay đổi khi phát hành).

Môi trường CASE chuẩn bao gồm một kho chứa, các công cụ đồ hoạ, phần mềm soạn thảo văn bản, phần mềm giao diện kho chứa, phần mềm đánh giá, và giao diện người sử dụng.



Một kho chứa là một từ điển dữ liệu hỗ trợ định nghĩa về các kiểu đối tượng khác nhau và quan hệ giữa các đối tượng đó. Các công cụ đồ hoạ hỗ trợ cho việc phát triển dạng sơ đồ và đánh giá sự hoàn chỉnh của sơ đồ dựa vào các luật đã được xác định trước. Phần mềm văn bản cho phép định dạng tên, nội dung, và chi tiết các phần

tử trong kho chứa. Phần mềm giao diện là bộ biên dịch xác định dạng dữ liệu được dùng (đồ hoạ hoặc văn bản). Phần mềm đánh giá là trí tuệ của CASE. Phần mềm này phân tích các đầu vào của sơ đồ hoặc kho chứa và xác định xem chúng có cú pháp hoàn chỉnh hay không (ví dụ có thoả mãn các định nghĩa của kiểu dữ liệu thành phần không), và chúng có tương thích với các đối tượng đang tồn tại khác trong ứng dụng hay không. Giao diện người dùng cung cấp các màn hình và các báo cáo để xử lý tương tác và gián tiếp.

CASE lý tưởng phải cung cấp đầy đủ hỗ trợ tự động cho toàn bộ chu kỳ tồn tại của dự án, bắt đầu bằng việc phân tích ở mức xí nghiệp, duy trì công việc và kết thúc. CASE lý tưởng do đó trở thành tiêu điểm cho mọi công việc nằm trong công nghệ phần mềm, và công việc của kỹ sư hệ thống tập trung vào khía cạnh logic của thiết kế. Công cụ CASE lý tưởng cũng có thể cung cấp về mặt kỹ thuật, dữ liệu và thực hiện thiết kế của tổ chức, lập dự án và kiểm tra nhóm làm việc trong các ứng dụng, chuẩn hoá dữ liệu, sơ đồ xây dựng cơ sở dữ liệu, xây dựng mã đơn giản bằng ngôn ngữ người sử dụng lựa chọn, tự động kiểm tra mã sinh ra đề phòng các lỗi logic và đánh giá sự hoàn chỉnh và đúng đắn trong quá trình làm việc một cách thông minh. Công cụ CASE thực hiện sự tiên tiến phải nhận ra các thành phần đã có trong kho chứa sử dụng lại việc phân tích, thiết kế và mã nguồn.

Kho chứa của CASE xác định đồng thời cái gì được hỗ trợ và trong phạm vi nào đó, có thể là hỗ trợ được bao nhiêu. Kho chứa là một siêu từ điển có thể nhập và lưu trữ siêu dữ liệu. Chẳng hạn, một phần tử dữ liệu trong một ứng dụng là dữ liệu, và các thuộc tính của nó tạo thành dữ liệu trao đổi có thể lưu trữ trong từ điển. Các thuộc tính của một phần tử bao gồm kiểu dữ liệu, kích thước, dung lượng, tần suất thay đổi và tiêu chuẩn soạn thảo. Một kho chứa CASE hoạt động như là một hệ quản trị cơ sở dữ liệu phục vụ cho hoạt động kỹ sư, cung cấp khả năng mở rộng lưu giữ dữ liệu trao đổi, và duy trì tất cả các thành phần và mối liên hệ qua lại giữa chúng.

Sự thông minh trong CASE thể hiện ở hai dạng chính: thông minh của giao diện và thông minh của bản thân sản phẩm CASE. Giao diện phải cung cấp cả kiểu thao tác cho người mới tập sự và cho người đã có kinh nghiệm. Nó phải cho phép khả năng một công việc được lưu giữ và tiếp tục lại. Công cụ này có thể được thay đổi theo yêu cầu cá nhân của người sử dụng. Ví dụ, nếu tôi muốn in màu vàng trên nền xanh, tôi sẽ gọi một lược đồ dòng dữ liệu, và tôi phải được phép thay đổi giá trị ngầm định để dùng các thuật ngữ và dạng của mình.

Các dạng biến đổi các giá trị đưa vào phải được phản ánh thông qua các tập sơ đồ. Điều này có nghĩa là nếu người dùng đưa các thực thể và các thuộc tính vào một kho chứa, khi anh ta di chuyển để triển khai một lược đồ đồ hoạ quan hệ thực thể, thông tin trong kho chứa phải được phản ánh trên lược đồ.

Trí tuệ của sản phẩm CASE bao gồm việc phân tích bên trong và giữa các kiểu lược đồ và đầu vào kho chứa. Trong trường hợp lý tưởng, yêu cầu của ứng dụng A nếu xung đột với mục tiêu của xí nghiệp Z, sẽ phải được bật cờ báo để xem ý kiến của người quản lý.

CASE lý tưởng phải cho phép người sử dụng phân tách và tích hợp các ứng dụng khác nhau một cách dễ dàng. Ví dụ, một công ty có thể muốn tư liệu hoá tất cả ứng dụng đã sử dụng và bắt đầu quản lý điện tử chúng. Khi người sử dụng xác định một ứng dụng mới có thể muốn tích hợp nó với một ứng dụng cũ, họ phải được phép để tạo ra một định nghĩa thứ ba để làm sáng tỏ sự chồng chéo, dư thừa, không chắc chắn và các vấn đề khác mà cặp tích hợp gặp phải.

Theo quy luật 40-20-40 của việc triển khai hệ thống, 40% thời gian của dự án được dùng để phân tích và thiết kế, 20% dành cho lập trình và 40% còn lại để chạy thử. Hướng hiện tại của các nhà bán sản phẩm là giảm bớt mã, do đó giảm đi 20% thời gian phát triển. Nhưng CASE lý tưởng sẽ còn có thể giảm được 40% thời gian chạy thử. Các công cụ kiểm tra CASE không cần thiết bằng các vấn đề khác (chẳng hạn cho sản phẩm làm việc tự do lỗi).

Trong một số sản phẩm ở thập kỷ 90, các nhà sản xuất sẽ cung cấp hỗ trợ kiểm tra trong các môi trường CASE của họ. Một cách lý tưởng, những hỗ trợ như vậy sẽ bao gồm cả thử nghiệm hộp đen và hộp trắng cùng với can thiệp của con người, nhưng không bắt buộc. Thử nghiệm hộp đen dùng cho kiểm tra sự kiểm tra chính xác của đầu ra dựa trên đầu vào; trong khi đó thử nghiệm hộp trắng dùng cho các sơ đồ logic trong chương trình. Phần mềm thông minh sẽ phân tích kiểu thực hiện và xác định chiến lược thử nghiệm thích hợp nhất. Thêm vào đó, nó sẽ phát triển dữ liệu thử nghiệm dựa trên các yêu cầu logic, dẫn dắt thử nghiệm và duy trì các kết quả thử nghiệm. Các kết quả thử nghiệm sẽ được tổng hợp thông qua chạy thử, các giai đoạn thử nghiệm, phiên bản phần mềm, và ngay cả các môi trường phần cứng. Khi lỗi được tìm ra, phần mềm này sẽ quay lại chỉnh tìm nguồn gốc lỗi - có khả năng là thông qua các module.

Khi phần mềm tạo nên lỗi phải có khả năng sửa chúng; nhưng nếu gây ra lỗi là một người hay là một lỗi logic, phần mềm phải thông báo cho kỹ sư hệ thống yêu cầu sửa lỗi.

Các sản phẩm trong tương lai này sẽ thực sự khắc phục 40% thời gian của dự án do chúng đủ thông minh để xác định các thành phần có thể sử dụng lại được của ứng dụng. Việc sử dụng lại thiết kế sẽ tiết kiệm được nhiều nhất nhưng cũng là việc khó khăn nhất. Đầu tiên, các module mã sẽ được sử dụng lại, sau đó đến lượt thiết kế, cuối cùng là phân tích logic cũng được dùng lại. Việc tìm ra mã có thể sử dụng lại được các công cụ CASE cung cấp vào giữa những năm 90, các phần khác sẽ được cung cấp vào thế kỷ này.

Việc miêu tả tính chất của CASE lý tưởng tập trung vào việc xem CASE có thể làm gì tốt hơn hiện tại. Với vấn đề này, chúng ta sẽ nghiên cứu CASE mỗi khi nó hỗ trợ một phương pháp luận và phát triển trong các chương sau. Mặc dù CASE và trí tuệ nhân tạo đang còn rất non trẻ, nhưng sự phát triển được mô tả ở trên hiện tại đã có thể thực hiện được với hiện trạng kỹ thuật công nghệ thế giới. Kho chứa CASE sẽ trở thành một trung tâm cho mọi công việc hoạt động trong các tổ chức hệ thống thông tin. Trí tuệ của CASE sẽ là chính trí tuệ của con người.

Các cấp độ tổ hợp CASE có thể phân loại. Tại mức thấp nhất của phổ tích hợp là các công cụ đơn. Khi các công cụ riêng lẻ cung cấp các tiện ích về truyền dữ liệu mức độ tích hợp đã có đôi chút cải thiện. Các công cụ như vậy sản xuất các đầu ra theo

dạng chuẩn có thể tương thích với các công cụ khác có thể đọc format này. Trong một số trường hợp, các người xây dựng các công cụ CASE hoàn chỉnh làm việc với nhau thông qua "cầu nối" giữa các tools. Single- source intergration xuất hiện khi các nhà sản xuất công cụ CASE đơn lẻ tích hợp một số lượng công cụ khác nhau và bán chúng như là một gói phần mềm.

Mức độ tích hợp cao nhất có (IPSE) Intergrated Project Support Environment. Các chuẩn cho mỗi khối cấu thành được mô tả trên được tạo. Các nhà sản xuất công cụ CASE sử dụng các chuẩn IPSE này để xây dựng các công cụ tương thích với các nhà sản xuất khác theo chuẩn.

5.7.2. Phân loại các công cụ CASE

5.7.2.1. Các công cụ lập kế hoạch hệ thống tác nghiệp

Bằng cách mô hình hoá các yêu cầu thông tin chiến lược của tổ chức, công cụ lập kế hoạch hệ thống tác nghiệp (Business System Planning Tools) cung cấp một siêu mô hình mà từ đó hệ thống thông tin đặc trưng sẽ được suy ra. Các thông tin tác nghiệp được mô hình hoá khi nó chuyển từ các thực thể được tổ chức khác nhau trong công ty. Mục đích chính của các công cụ trong phân loại là giúp hiểu biết được thông tin di chuyển giữa các đơn vị tổ chức như thế nào.

Các công cụ như vậy cung cấp nội dung quan trọng khi các chiến lược hệ thống thông tin được cấu trúc và khi các hệ thống và phương pháp hiện tại không hợp với yêu cầu của tổ chức.

5.7.2.2 Các công cụ quản lý dự án

Nhiều nhà quản lý dự án phần mềm đang tiếp tục đánh giá, điều khiển và theo dõi các dự án phần mềm theo cách trước đây đã làm từ 1950. Mỉa mai thay, có một dãy rộng các công cụ quản lý dự án CASE có tác động sâu sắc lên chất lượng của quản lý dự án cho các cố gắng phát triển phần mềm cỡ lớn và nhỏ.

Hiện nay, phần lớn các công cụ quản lý dự án CASE định hướng vào một phần đặc trưng của quản lý dự án hơn là cung cấp hỗ trợ toàn bộ cho hoạt động quản lý. Bằng cách sử dụng một tập hợp công cụ CASE có chọn lọc, quản trị dự án có thể tạo ra các đánh giá hiệu quả về giá thành, nguồn lực, và thời gian của dự án phần mềm, xác định các cấu trúc công việc và thời biểu làm việc đồng thời theo dõi dự án. Hơn nữa người quản lý có thể sử dụng các công cụ để thu thập các metrics mà cuối cùng cung cấp các chỉ định về chất lượng và hiệu quả phát triển phần mềm.

Công cụ lập kế hoạch dự án: Các công cụ trong lớp này tập trung vào hai mảng chính: định lượng giá, nguồn lực dự án và lập biểu dự án. Các công cụ định lượng giá thành cho phép quản trị dự án ước lượng cỡ của dự án bằng cách dùng các độ đo gián tiếp (số dòng mã và số các chức năng) và mô tả toàn bộ các đặc tính dự án (ví dụ độ phức tạp, kinh nghiệm của đội ngũ). Các công cụ này tiếp theo ước lượng nguồn lực, thời gian dự án và gợi ý số lượng người. Nhiều công cụ trong số đó cho phép mô

phỏng tình huống để quản trị có thể đã định thời gian hoàn thành và kiểm tra giá thành và khả năng thực hiện.

Các công cụ lập biểu dự án cho phép nhà quản lý xác định mọi nhiệm vụ, tạo mạng các công việc, biểu diễn sự phụ thuộc công việc. Phần lớn các công cụ sử dụng phương pháp lập đoạn gắng để xác định thời lượng hoàn thành dự án.

Các công cụ theo dõi các yêu cầu: Khi hệ thống được phát triển lớn dần thì rất có nguy cơ rơi vào tình trạng đổ vỡ, hệ thống đã hoàn thiện không đáp ứng hoàn toàn các yêu cầu của khách hàng. Mục đích của các công cụ theo dõi yêu cầu là cung cấp một cách tiếp cận hệ thống để phân tách các yêu cầu, bắt đầu với các quy trình gội thầu -RFP (Request For Proposal)- của khách hàng hoặc các đặc tả.

Công cụ theo dõi yêu cầu đặc trưng bao gồm các định lượng text giao tác người-máy, cùng với hệ thống quản lý cơ sở dữ liệu lưu trữ và phân loại các yêu cầu hệ thống mà chuyển đến từ các RFP hoặc các đặc tả. Phân tích viên phân loại các yêu cầu được biểu diễn bởi các câu và đưa chúng vào cơ sở dữ liệu.

Các công cụ quản lý và độ đo: Các độ đo phần mềm cải thiện khả năng của nhà quản lý để điều khiển và phối hợp quá trình xử lý sản xuất phần mềm và khả năng của các cộng tác viên để cải thiện chất lượng phần mềm. Các công cụ đánh giá hoặc độ đo hiện nay tập trung vào các đặc trưng xử lý và chế tạo. Các công cụ định hướng quản lý thu thập các độ đo đặc trưng dự án. Các công cụ định lượng kỹ thuật xác định các độ đo kỹ thuật. Cung cấp các điểm quan trọng nhất về chất lượng thiết kế hoặc mã.

Các công cụ quản lý hỗ trợ cho nhà quản lý hệ thống thông tin cho phép ưu tiên các dự án có sự cạnh tranh về các nguồn tài nguyên hữu hạn. Do sử dụng các yêu cầu, độ ưu tiên, các ràng buộc được đặt trong các tổ chức và các lỗi, rũi ro của kỹ thuật và nghiệp vụ, những công cụ này sử dụng các kiến thức chuyên gia để đưa ra các gợi ý quyết định cho nhà quản lý.

5.6.2.3. Các công cụ hỗ trợ

Các công cụ tư liệu hoá cho phép cán bộ phát triển ứng dụng tự động hoá cập nhật tài liệu và in các báo cáo về ứng dụng.

5.6.2.4. Các công cụ phân tích và thiết kế

Các công cụ phân tích và thiết kế cho phép các kỹ sư phần mềm tạo các mô hình của hệ thống. Nó bao gồm biểu diễn cho dữ liệu, luồng điều khiển, nội dung dữ liệu (thông qua các đinh nghĩa từ điển các yêu cầu), quá trình xử lý, các đặc tả điều khiển, và các biểu diễn mô hình hoá khác.

Các công cụ phân tích và thiết kế hỗ trợ cả việc tạo mô hình cũng như đánh giá chất lượng. Bằng quá trình kiểm tra tính chắc chắn và giá trị của mô hình, các công cụ phân tích và thiết kế cung cấp cho công nghệ phần mềm khả năng giảm tối thiểu các lỗi có khả năng lan truyền tới chương trình ứng dụng.

Công cụ SA/SD: Phần lớn các công cụ phân tích và thiết kế sử dụng phương pháp phân tích và thiết kế cấu trúc. Nó cho phép tạo các mô hình của hệ thống phức tạp dần, bắt đầu từ mức độ yêu cầu và kết thúc với sơ đồ kiến trúc.

Công cụ PRO/SIM: Các công ty tạo mẫu và mô phỏng (Prototyping and simulation) cho khả năng dự đoán trước dáng điệu của hệ thống. Mặt khác, nó cho phép khách hàng trong thời gian ngắn nhất có thể quan sát được mô hình. Nhiều công cụ dạng này cho phép sản sinh mã.

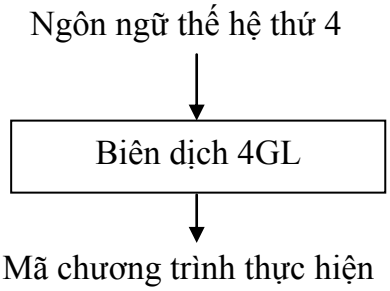
Các công cụ phát triển và thiết kế giao diện: Các nghiên cứu công nghiệp cho thấy rằng 50-80% mã của các ứng dụng tương tác là dành cho quản lý giao diện. Các công cụ phát triển và thiết kế giao diện thực sự là tập hợp các công cụ tạo các đơn vị chương trình như menu, button, windows,...Tuy nhiên, các công cụ trên đang được thay thế bởi các công cụ tạo mẫu giao diện cho phép tạo các màn hình theo chuẩn một cách nhanh chóng.

5.7.2.5.Các công cụ lập trình

Các công cụ lập trình bao gồm bộ dịch, soạn thảo, gỡ lỗi cho phép dùng phần lớn các ngôn ngữ lập trình truyền thống. Các công cụ này có liên quan nhiều tới môi trường lập trình hướng đối tượng, ngôn ngữ thế hệ bốn, sản sinh chương trình.

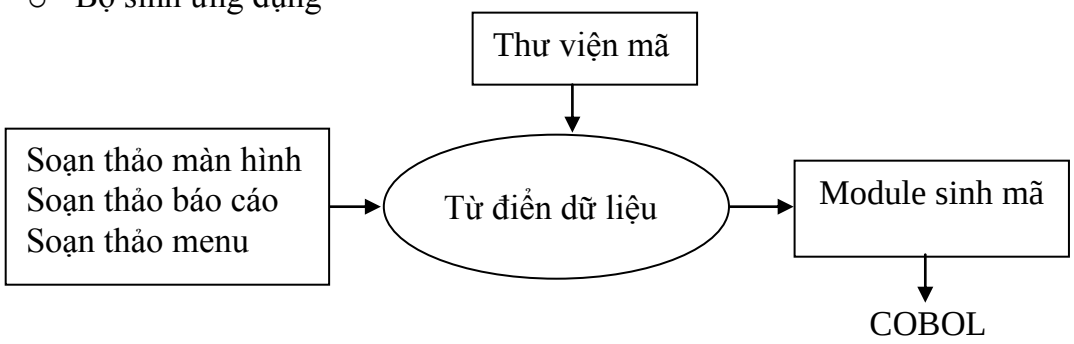
Các công cụ mã hoá truyền thống: Đã có thời các công cụ mã hoá quy ước là: Complier, Editor, Debugger. Pressman đã nói về vấn đề này như sau: "Khi trong tay bạn chỉ có cái búa, thì mọi vấn đề sẽ giống như cái đinh". Hầu như suốt 30 năm, lập trình viên chỉ có các công cụ trên trong tay nên mọi vấn đề về công nghệ phần mềm đều quy về vấn đề mã hoá.

Các công cụ mã hoá thế hệ 4: Hệ thống hỏi đáp cơ sở dữ liệu, sinh mã và ngôn ngữ thế hệ 4 đã làm thay đổi cách lập trình.

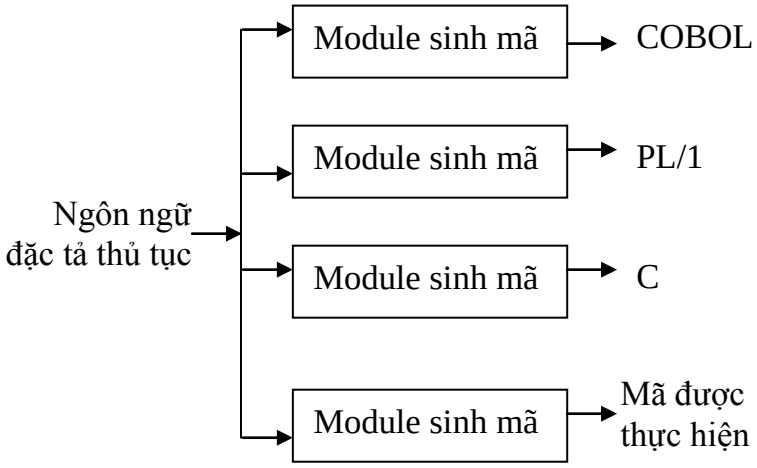


Các công cụ lúc này được phân làm một số loại, ví dụ như:

- Bộ sinh ứng dụng



- Bộ sinh mã



*Các công cụ lập trình hướng đối tượng***:** Lập trình hướng đối tượng là một trong các công nghệ "nóng nhất" trong công nghệ phần mềm. Vì lý do này, các nhà chế tạo CASE đang cung cấp ồ ạt các công cụ phát triển phần mềm hướng đối tượng tới thị trường.

Các ngôn ngữ phổ biến hiện nay là C++, Eiffel, Smalltalk, Objective-C. Môi trường hướng đối tượng đặc trưng thường kết hợp với các cung cấp giao diện thế hệ 3 (menu, mouse, multitasking,...) cùng với các chức năng đặc biệt như "browser" - một chức năng cho phép kỹ sư phần mềm kiểm tra tất cả các đối tượng được chứa trong thư viện.

5.7.2.6. Các công cụ tích hợp và kiểm tra

Có ba hạng công cụ kiểm tra được sử dụng nhiều nhất. Nhiều công cụ phải trải hết 2 hoặc 3 phân loại nói trên.

*Các công cụ phân tích tĩnh***:** Các công cụ kiểm tra tĩnh giúp các kỹ sư phần mềm trong việc rút ra các trường hợp kiểm tra. Ba kiểu khác nhau của các công cụ kiểm tra tĩnh được dùng trong công nghiệp: các công cụ kiểm tra dựa trên mã, các ngôn ngữ kiểm tra đặc tả và các công cụ kiểm tra dựa trên yêu cầu.

Code-Based testing tools nhận mã nguồn như là đầu vào và tiến hành một số phân tích. Căn cứ vào mô tả của đầu vào chương trình và thiết kế thủ tục như là chỉ dẫn các công cụ kiểm tra tĩnh suy ra các trường hợp kiểm tra sử dụng đường dẫn, các kiểm tra điều kiện và các tiêu chuẩn luồng dữ liệu.

Requirements-based testing tools phân tách các yêu cầu người dùng và khuyên một số trường hợp kiểm tra (hoặc lớp kiểm tra) mà sẽ thử các yêu cầu. Để thực hiện tốt thì các công cụ trong phân hạng này phải truy nhập tới các đặc tả hình thức của phần mềm.

Specialized testing languages cho phép các kỹ sư phần mềm viết các đặc tả kiểm tra chi tiết mà mô tả mỗi trường hợp kiểm tra và logic cho sự thực hiện nó. Trong phần lớn các trường hợp, các công cụ kiểm tra tĩnh sẽ tư liệu hoá và phân loại các phép kiểm tra. Và nó sẽ so sánh kết quả thực với kết quả dự tính.

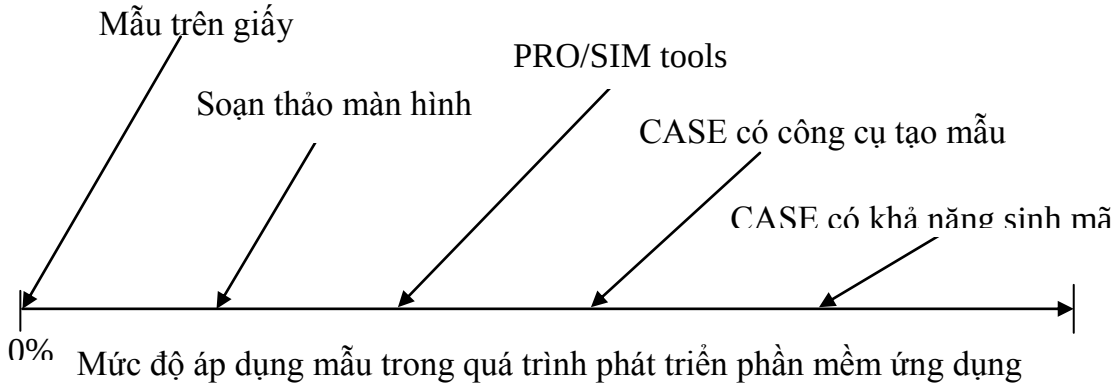
Các công cụ phân tích động: Các công cụ kiểm tra động tương tác với quá trình thực hiện chương trình, kiểm tra đường thử, kiểm tra xác nhận về giá trị các biến, các công cụ động có thể là loại can thiệp hoặc không can thiệp. Công cụ can thiệp (intrusive) thay đổi phần mềm để kiểm tra. Công cụ không can thiệp (nointrusive) sử dụng một xử lý phần cứng tách rời chạy song song với xử lý chưa có chương trình đang kiểm tra. Phần lớn các công cụ thuộc phân loại phân tích động tạo các báo cáo chỉ rõ số lượng các khối, câu lệnh đã được thực hiện và các thời gian thực hiện trung bình cho các khối lệnh. Công cụ kiểm tra động có thể được dùng để nối tiếp với công cụ kiểm tra tĩnh. Các kiểm tra tĩnh tạo ra các trường hợp test sau đó được quản lý bởi các công cụ động.

Công cụ quản lý test: Công cụ quản lý test dùng để điều khiển và phối hợp các kiểm tra phần mềm cho mỗi bước kiểm tra chính. Các công cụ trong phân loại này quản lý và phối hợp các kiểm tra regression. Tiến hành các so sánh output thực và biểu kiến. Nhiều công cụ quản lý test cũng phục vụ như là các bộ điều khiển test.

Một bộ điều khiển test đọc một hoặc nhiều trường hợp test từ file test, định dạng các dữ liệu test để phù hợp nhu cầu phần mềm, sau đó gọi phần mềm để test. Cuối cùng, nhà quản lý test đôi khi làm việc với công cụ theo dõi yêu cầu để cung cấp các phân tích coverage yêu cầu cho kiểm tra.

5.7.2.7. Các công cụ tạo mẫu

Tạo mẫu được dùng rộng rãi như là sự tiến hoá của công nghệ phần mềm. Các công cụ tạo mẫu được phân bổ theo hình sau:



Mức thấp nhất, các công cụ tạo "mẫu trên giấy". Công cụ chạy trên máy có thể tạo các hình ảnh thực dùng để minh hoạ các chức năng, dáng điệu hệ thống. Các hình ảnh này không thể thực hiện được. Trong một số trường hợp, bộ vẽ màn hình có thể sinh ra các mã để tạo màn hình. Các công cụ CASE phức tạp hơn cho phép tạo các thiết kế dữ liệu, cả màn hình hiển thị và báo biểu.

5.7.2.8. Các công cụ bảo trì

Các công cụ bảo trì có thể được phân lại theo các chức năng sau:

+ Thiết kế ngược với các công cụ đặc trưng.
+ Các công cụ phân tích và cấu trúc lại mã.
+ Các công cụ kiến tạo lại hệ thống trực tuyến.

Các công cụ thiết kế ngược: Các công cụ thiết kế ngược tiến hành tạo lại các phân tích ban đầu trên cơ sở các chương trình đã tồn tại. Các công cụ này cũng có thể phân ra loại tĩnh và động.

Một công cụ thiết kế ngược dùng mã nguồn như đầu vào và phân tích lấy ra kiến trúc chương trình, cấu trúc điều khiển, luồng logic, cấu trúc dữ liệu, luồng dữ liệu. Các công cụ thiết kế phụ thuộc (Dependency analysis tools) tiến hành phần lớn các chức năng trên, ngoài ra nó còn xây dựng sơ đồ graphic biểu diễn sự phụ thuộc chỉ sự liên kết giữa các cấu trúc dữ liệu, khối chương trình, và các đặc tính khác của chương trình.

Các công cụ thiết kế ngược quan sát phần mềm khi nó chạy và sử dụng các thông tin nhận được để xây dựng mô hình dáng điệu của chương trình. Mặc dù chúng ít được dùng nhưng cần thiết cho việc bảo trì các chương trình thời gian thực hoặc được nhúng trong hệ thống khác.

Các công cụ thiết kế: Các công cụ thiết kế lại có thể chia làm hai nhóm: công cụ cấu trúc lại mã, công cụ thiết kế lại dữ liệu.

Các công cụ cấu trúc mã lại nhận mã nguồn phi cấu trúc như là đầu vào, tạo thiết kế ngược, sau đó cấu trúc lại mã mới. Mặc dù các công cụ như vậy có thể hữu ích, chúng chỉ tập trung trong việc thiết kế thủ tục của chương trình.

Các công cụ thiết kế lại dữ liệu làm việc tại cực kia của thiết kế. Các công cụ như vậy truy nhập định nghĩa dữ liệu hoặc cơ sở dữ liệu được mô tả trong ngôn ngữ lập trình hoặc ngôn ngữ mô tả cơ sở dữ liệu. Sau đó nó chuyển sang dạng biểu diễn graphic có thể phân tích được bởi kỹ sư phần mềm.

Tương tác với các công cụ thiết kế lại, kỹ sư phần mềm có thể thay đổi cấu trúc của cơ sở dữ liệu, chuẩn hoá dữ liệu, sau đó tự động sinh mã mới. Các công cụ có thể sử dụng hệ chuyên gia.

5.7.3. Một số công cụ được cung cấp tự động cho việc sinh mã

Các tiện ích CASE được xây dựng trên việc sinh mã hay các giao diện phục vụ cho sinh mã, cho phép trộn lẫn các mã khác nhau để phù hợp với môi trường phát triển và ngôn ngữ sinh ra.

Một số CASE thông dụng hiện nay như:

Sản phẩm	Kỹ thuật
ADW-Construction Workbench của Knowledgeware Inc., Atlanta, CA	Xây dựng các mã giả bằng các modul, nó dùng cho việc sinh mã cho MsDOS, MVS
Cdevelopment Environment, OOSD/C++ của S/Cubed Inc. Stamford, CT	Sinh mã Cobol cho các Mainframe, AS/400, OS/2. Sinh mã C cho MsDOS, OS/2
IEW của Texas Instruments Dallas, TX	Sinh mã Cobol gắn với SQL, sinh mã C Cho MVS, MsDOS, OS/2. Giao diện cho Telon và sinh ra các mã khác.

Sản phẩm	Kỹ thuật
NeXTStep 3.0 của NeXT Computer Redwood City, CA	Môi trường cơ sở dữ liệu hướng đối tượng
ObjectMaker của Mark V Systems	Sinh mã C hay C++ cho MsDOS, VMS, Unix, AIX
Software Through Pictures của Intergrated Development	Sinh mã C hay C++ cho Unix, AIX
Systems Architect của Popkin Software & System Inc. New York, NY	Sinh mã C cho MsDOS, OS/2
Teamwork,Ensemble của Cadre Technologies Providence, RI.	Sinh mã C hay C++ cho Unix, OS/2, AIX
Visible Analyst Workbench của Visible Systems Corp. Newton, MA.	Sinh mã C cho MsDOS

Những tiện ích trên cung cấp một cách tự động bao gồm các tiện ích sinh mã, chương trình biên dịch lớn dần và môi trường tạo ra chương trình. Toàn bộ chúng chính là tiện ích Lower CASE hay Back-End CASE.

Câu hỏi

1. Thế nào là một phong cách cài đặt chương trình tốt.
2. Tài liệu trong của chương trình đem lại những lợi ích gì?
3. Các nền tảng của một ngôn ngữ có thể lập trình được? Hãy chỉ rõ sự tương đương giữa bản thiết kế và cài đặt để đảm bảo được sự đúng đắn của chương trình.
4. Với hồ sơ thiết kế hệ thống trong chương 4, bạn chọn công cụ cài đặc thế nào, vì sao?
5. Trong cài đặc ứng dụng, để nâng cao hiệu suất của hệ thống, bạn đã làm gì? Hãy nêu và phân tích sự cải tiến đó.