CHƯƠNG 4

THIẾT KẾ PHẦN MỀM

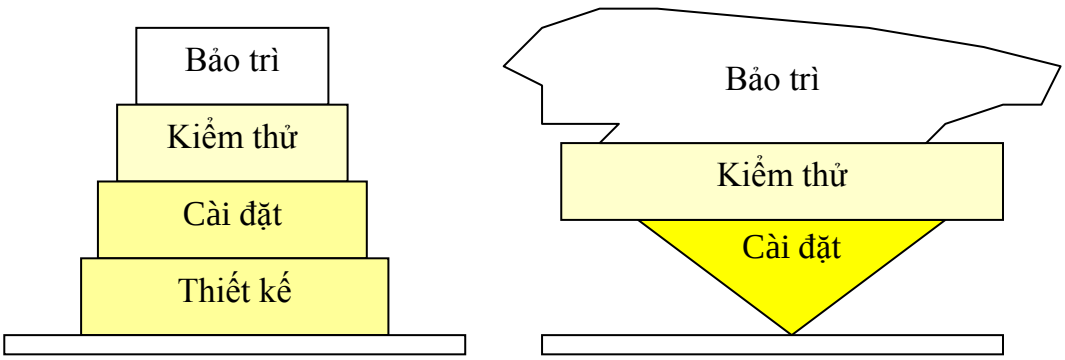
Xây dựng ứng dụng phần mềm là một dây chuyền các chuyển đổi, mà ở đó phân tích nhằm xác định ứng dụng sẽ thực hiện cái gì (what) còn thiết kế nhằm để trả lời câu hỏi phần mềm cụ thể sẽ như thế nào (how)? Tức là xác định cách thức thực hiện những gì đã được đặt ra ở phần phân tích.

Trong ba giai đoạn: thiết kế, cài đặt và bảo trì thì thiết kế là giai đoạn quan trọng nhất, chịu trách nhiệm đến 80% đối với sự thành công của một sản phẩm. Cài đặt là việc thực thi những gì đã thiết kế. Nếu trong quá trình cài đặt có xuất hiện vấn đề thì phải quay lại sửa bản thiết kế. Quá trình thiết kế tốt là cơ sở để quản lý và giảm chi phí cho công việc bảo trì phần mềm sau này.

4.1. ĐẶC ĐIỂM CỦA QUÁ TRÌNH THIẾT KẾ PHẦN MỀM

Nhiệm vụ của thiết kế là chuyển đổi những yêu cầu của hệ thống (kết quả của quá trình phân tích) sang dạng biểu diễn của hệ thống phần mềm. Nghĩa là xây dựng các mô tả văn bản (thiết kế chi tiết) nêu rõ mối quan hệ giữa tiền điều kiện và hậu điều kiện cho tất cả các chức năng (quá trình) của hệ thống. Tiền điều kiện xác định những cái sẽ nhận giá trị chân lý đúng trước khi một quá trình thực hiện, còn hậu điều kiện xác định những điều sẽ nhận giá trị đúng khi chấp nhận tiền điều kiện và khi quá trình đó kết thúc thành công.

Tầm quan trọng của thiết kế được thể hiện qua hình sau:

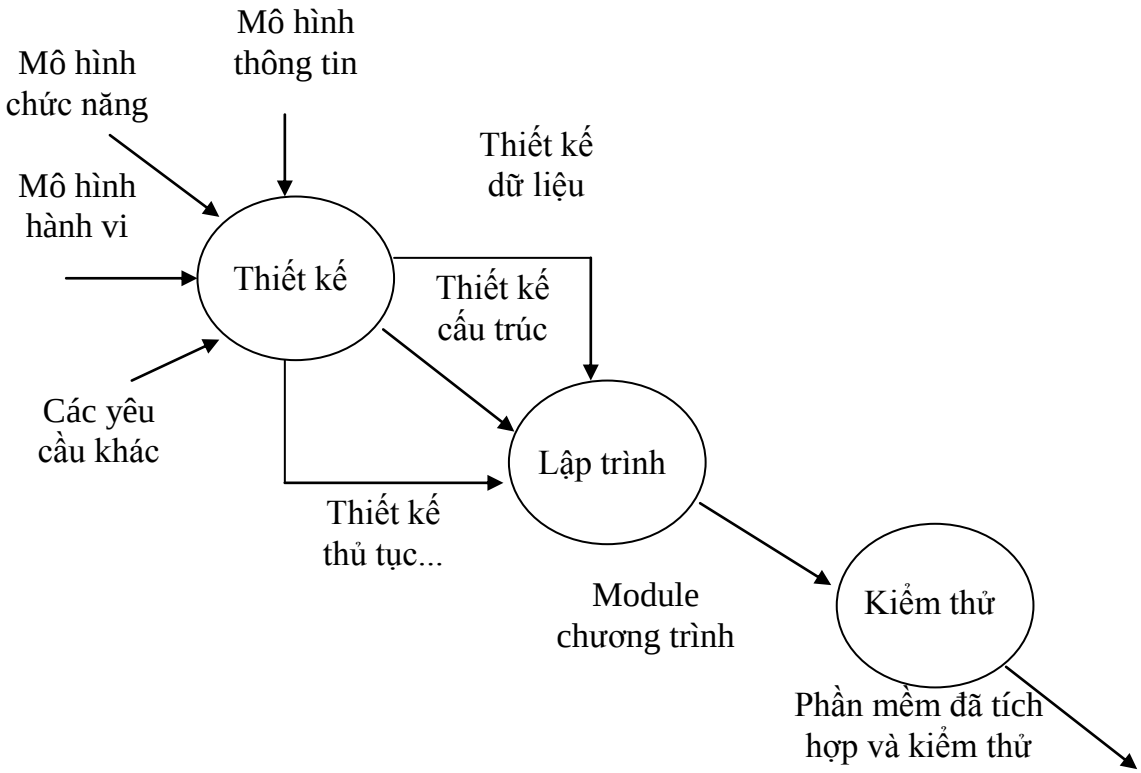


Có thiết kế Không có thiết kế

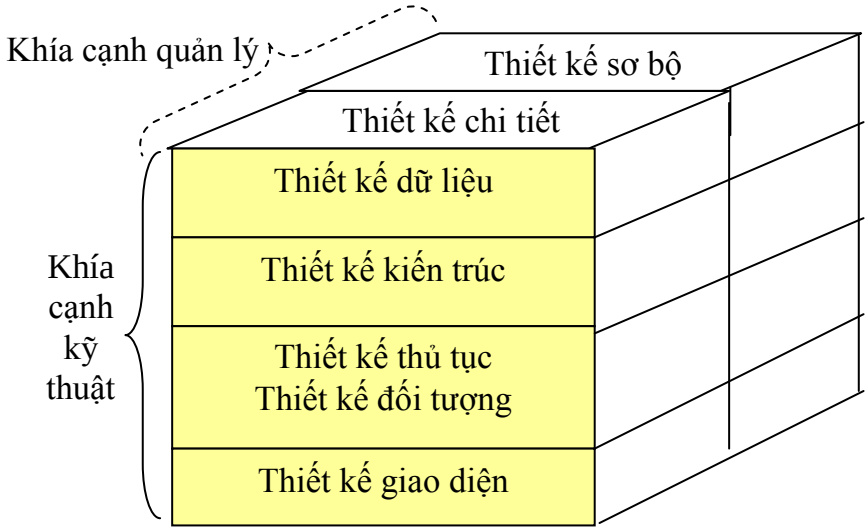
Như vậy, thiết kế là một thực tế về một quyết định chọn lựa, xây dựng một đặc tả về hành vi nhìn thấy được từ bên ngoài và bổ sung các chi tiết cần thiết cho việc cài

đặt trên hệ thống máy tính bao gồm cả chi tiết về tổ chức quản lý dữ liệu, công việc và tương tác với con người. Thiết kế phải nhờ vào các kinh nghiệm và phải học tập những cái có sẵn từ các hệ thống khác; không thể chỉ đọc sách là đủ. Bản thiết kế tốt là chìa khóa cho sự thành công của hệ thống.

Mối liên quan của thiết kế phần mềm với công nghệ phần mềm được thể hiện qua sơ đồ sau:



Thiết kế phần mềm là hoạt động được xác lập dựa trên hai mặt: quản lý và kỹ thuật, chúng đan xen với nhau. Mối quan hệ giữa hai khía cạnh kỹ thuật và quản lý được thể hiện qua sơ đồ:



- Trong quan điểm quản lý: thiết kế phần mềm được tiến hành 2 bước:

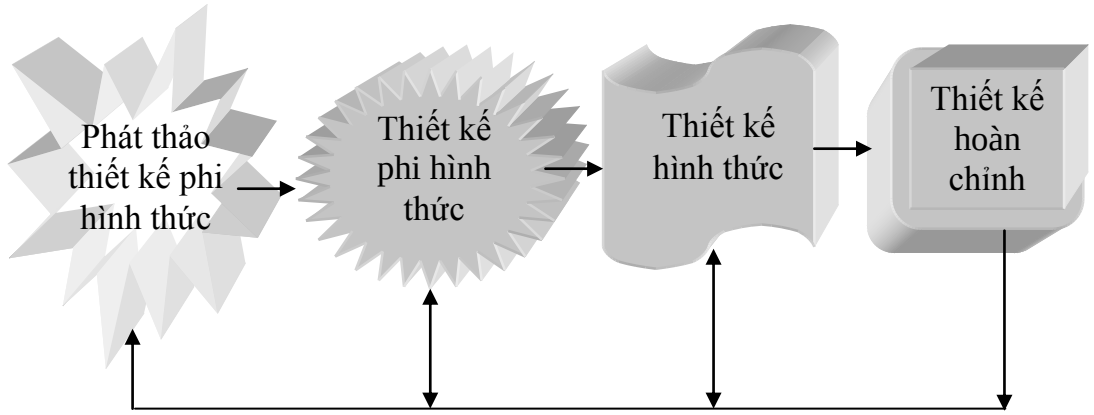
+ Thiết kế sơ bộ: quan tâm đến việc dịch các yêu cầu thành các kiến trúc dữ liệu và phần mềm.

+ Thiết kế chi tiết: tập trung vào việc làm mịn biểu diễn kiến trúc để dẫn đến cấu trúc dữ liệu chi tiết và biểu diễn thuật toán cho phần mềm.

- Đối với khía cạnh kỹ thuật, xuất hiện một số hoạt động thiết kế như:
  + Thiết kế dữ liệu
  + Thiết kế kiến trúc
  + Thiết kế thủ tục
  + Thiết kế đối tượng
  +Thiết kế giao diện

Trong tiến trình thiết kế, mô hình để biểu diễn công việc thiết kế là đồ thị. Các đỉnh của đồ thị dùng để biểu diễn các thực thể (các tiến trình, các chức năng, các kiểu...) và các cạnh là các mối liên hệ giữa chúng. Quá trình thiết kế thường được mô tả bằng nhiều mức khác nhau của cách tiếp cận trừu tượng hóa, nhằm tách các bộ phận cấu thành của bài toán nhằm nâng cao độ chắc chắn, độ tin cậy của hệ thống.

Tiến trình thiết kế được chỉ ra ở sơ đồ sau:



4.2. CÁC HOẠT ĐỘNG CỦA QUÁ TRÌNH THIẾT KẾ PHẦN MỀM

Như trên đã nói, phân tích nhằm xác định ứng dụng sẽ thực hiện cái gì còn thiết kế nhằm để trả lời câu hỏi phần mềm cụ thể sẽ như thế nào? Trong thực tế, không có sự tách biệt giữa hai giai đoạn này mà là hai hoạt động được tiến hành song song và chúng bổ sung lẫn nhau. Các hoạt động phải thực hiện trong quá trình này gồm: định danh, suy diễn, tổng hợp lại, xem xét lại và tạo tài tiệu.

Định danh: chính là tìm kiếm những gì quan trọng có liên quan, như việc tìm các thực thể, các đối tượng, các mối quan hệ, các chức năng, các tiến trình và các ràng buộc của hệ thống.

Thiết kế là việc tham chiếu các yêu cầu logic sẽ làm việc như thế nào trong môi trường tính toán đích. Điều này có nghĩa là chúng ta định danh cấu trúc thiết kế hệ thống, nó là phương hướng thiết kế nền tảng. Các phương hướng có thể bao gồm:

- Xử lý theo lô, trực tuyến hoặc thời gian thực.
- Các hàm chức năng nào được kết nối với nhau và ứng dụng sẽ làm việc trong môi trường sản phẩm như thế nào.
- Các giao diện người sử dụng chung như điều khiển menu, cửa sổ – biểu tượng – menu – con trỏ, điều khiển lệnh,...
- Kiểu thao tác, người sử dụng là chuyên gia, tập sự hay bình thường.

Suy diễn: Xác định các chi tiết của những gì đã được định danh. Về mặt thể hiện, một yêu cầu có thể được cung cấp bản kê khai thống nhất của khách hàng cho báo cáo đặc biệt.

Trong quá trình xem xét, ta sẽ tìm câu trả lời cho các câu hỏi như:
Thông tin nào có thể đảm bảo chắc chắn về người sử dụng? Nó có đang tồn tại?
Điều gì đặc biệt đối với người sử dụng?
Kiểu của các Query người sử dụng đang hỏi?
Các dạng câu hỏi nào người sử dụng muốn hỏi mà họ không thể trả lời?
Kiểu dữ liệu phân tích nào người sử dụng cần?
Các form (như màn hình hiển thị hay giấy) được đưa ra?
Các người sử dụng đặt câu hỏi ở đâu (vật lý)?
Dữ liệu đang ở đâu (tập trung/ phân tán/ nữa tập trung)? Và nó có thể ở đâu?

Mỗi yêu cầu từ quá trình phân tích được khai triển thành chi tiết lớn hơn trong thiết kế và được tham chiếu tới phần cứng, phần mềm thuộc cấu trúc thiết kế hệ thống.

Ở đây, các vấn đề liên quan đến cần giải quyết như:

- Cơ sở dữ liệu thiết kế có thể được thiết kế để cung cấp như thế nào, về mặt thể hiện là thời gian đáp ứng tốt nhất với hiệu quả cao nhất?
- Các chương trình có thể được đóng gói như thế nào để đáp ứng các ràng buộc tiến trình?
- Các hoạt động xem xét khác được quyết định bao gồm các công việc thông thường chung cho các tiến trình sử dụng thông thường. Về mặt thể hiện, tiến trình hiển thị sẽ được thực hiện như thế nào? Các lập trình viên sẽ viết giao diện hiển thị riêng hay sẽ có các module chung cho các thao tác hiển thị? Phần thân và các chi tiết của hệ thống các chương trình tiện ích được tất cả các lập trình viên sử dụng.
- Hoạt động xem xét chính cuối cùng là kiểm tra các ràng buộc của ứng dụng. Chúng ta chắc chắn rằng mỗi ràng buộc đều được bảo toàn trong thiết kế và quá trình đó nằm trong các giới hạn quy định.

Tổng hợp lại: xây dựng một khung nhìn thống nhất của ứng dụng, điều hoà các phần không thích hợp và biểu diễn các yêu cầu trên form đồ thị. Sự biểu diễn có thể là thủ công hoặc tự động, sử dụng các công cụ tính toán cơ sở.

Trong thiết kế, tổng hợp lại phải xây dựng một thiết kế vật lý của ứng dụng, điều chỉnh các phần không phù hợp và biểu diễn các yêu cầu chi tiết hơn. Chúng ta có thể thêm các hàm chức năng vào ứng dụng trong môi trường đặc biệt.

Xem xét lại: là việc thực hiện điều khiển chất lượng. Tại cuối mỗi giai đoạn, phân tích lại tính khả thi, thời biểu và bố trí nhân sự. Xem xét lại chúng khi cần thiết dựa vào sự hoàn thiện hơn, các khái niệm hiện tại của hệ thống mới.

Như thế, xem xét lại chính là thực hiện điều chỉnh chất lượng. Tại cuối giai đoạn này dẫn hướng thiết kế xuyên suốt, so sánh thiết kế với các yêu cầu logic, sự hoàn thiện logic và sự đúng đắn. Phân tích lại thời biểu và bố trí nhân sự để tiến hành cài đặt, kiểm duyệt, thay đổi, đào tạo và sự chuyển công tác, xem xét lại chúng cho phù hợp yêu cầu.

Tạo tài liệu: tạo các chương trình hữu ích đặc biệt và một tài liệu thiết kế toàn thể. Tài liệu thiết kế mô tả cơ sở dữ liệu, cấu trúc của ứng dụng, các ràng buộc, cũng như các đồ thị và văn bản thiết kế. Các module của chương trình bao gồm các chi tiết của tiến trình, tất cả các giao diện thiết kế và các thông tin đặc biệt để phát triển ứng dụng.

Tuy nhiên, các hoạt động trên trong phân tích và thiết kế có những khác biệt cơ bản, chúng được chỉ ra ở bảng sau:

Hoạt động	**Phân tích**	**Thiết kế**
Định danh	Tìm các thứ yếu trong ứng dụng . Nội dung này không giới hạn bao gồm các thực thể, các đối tượng, các quan hệ, các hàm chức năng, các ràng buộc, các phần tử dữ liệu, các trình điều khiển, các yêu cầu chuẩn hoá,...	Làm tinh hệ thống khái niệm và cung cấp nó cho các yêu cầu chức năng. Định danh các thoả hiệp của các yêu cầu nào mà có thể cần thiết để làm việc trong giới hạn của môi trường cài đặt. Xác định các chuẩn chung và các quy định cho môi trường cài đặt sao cho tất cả các công việc còn lại phải tôn trọng.
Suy diễn	Xác định các chi tiết chức năng của những gì đã được định danh. Người sử dụng cung cấp các khái niệm cho các định nghĩa và mô tả cả các thủ tục, công thức và tiến trình. Việc xem xét tỉ mỉ này phụ thuộc vào phần cứng, phần mềm hay vùng làm việc	Đối với mỗi hàm chức năng, tham chiếu chúng tới môi trường phần cứng và phần mềm. Định danh các module có thể sử dụng lại. Đúc kết chi tiết các thông báo tiến trình và các module truyền thông ngoài.
Tổng hợp	Mở rộng khung nhìn ứng dụng thống nhất. Mở rộng và lập tài liệu biểu diễn ứng dụng. Đồ thị, bảng biểu, và các kỹ thuật khác hay được sử dụng để biểu diễn.	Mở rộng sự tham chiếu thống nhất của ứng dụng tới môi trường phần cứng và phần mềm. Xác định biểu đồ và đóng gói định vị cho tất cả dữ liệu và các tiến trình. Đồ thị, bảng biểu, và các kỹ thuật khác hay được sử dụng để biểu diễn.

Hoạt động	Phân tích	Thiết kế
Xem xét lại	Xem xét lại qua toàn bộ việc phân tích với các thành viên dự án và người sử dụng. Xem xét lại thời biểu và giá cả cần thiết.	Xem xét qua toàn bộ các thành phần thiết kế, sơ đồ kiểm tra, sơ đồ thay đổi và thiết kế cơ sở dữ liệu với các thành viên dự án, các đặc điểm chương trình với các lập trình viên, các thao tác hiển thị với người sử dụng. Xem xét lại thời biểu và giá cả cần thiết.
Tài liệu	Đúc kết các form và đồ thị, cung cấp văn bản tài liệu cho tất cả các hoạt động phân tích.	Đúc kết các form và đồ thị, cung cấp văn bản tài liệu cho tất cả các hoạt động thiết kế.

Để ý rằng, chúng ta bàn luận về phân tích và thiết kế như là việc tham chiếu đơn giản của "cái gì" và "như thế nào", nhưng nó không phải là tham chiếu 1-1. Ở đây, chúng ta cần tới sự thoả hiệp các yêu cầu phân tích trong thiết kế. Thoả hiệp các yêu cầu nghĩa là chúng có thể phải được cấu trúc lại, thao tác, phân nhỏ hay thay đổi để phù hợp với giới hạn của môi trường. Việc liên kết giữa phân tích và thiết kế chương trình lỏng hay chặt phụ thuộc vào phương pháp luận và môi trường cài đặt. Về mặt thể hiện, dữ liệu sẽ khác nhau nếu chúng ta sử dụng các chuẩn dữ liệu khác nhau. Mức độ chi tiết các yêu cầu sẽ khác nhau nếu ta sử dụng ngôn ngữ cài đặt khác nhau.

4.3. NỀN TẢNG THIẾT KẾ

Mặc dầu có nhiều phương pháp thiết kế phần mềm nhưng trong quá trình thiết kế, chúng ta đều sử dụng một số khái niệm làm nền tảng. Chúng được gọi là nền tảng thiết kế.

4.3.1. Trừu tượng (abstraction)

Khái niệm trừu tượng là sự cho phép tập trung vào vấn đề ở mức tổng quát nào đó, không xét tới các chi tiết mức thấp hơn không liên quan. Việc dùng trừu tượng hoá cho phép ta làm việc với khái niệm và thuật ngữ quen thuộc trong môi trường vấn đề mà không phải biến đổi chúng thành một cấu trúc không quen thuộc.

Khi xét vấn đề cho việc tìm ra giải pháp module, chúng ta có thể đặt ra nhiều mức độ trừu tượng. Tại mức trừu tượng cao nhất: phát biểu bằng ngôn ngữ môi trường của vấn đề. Tại mức trừu tượng thấp hơn, thường lấy khuynh hướng thủ tục; tại mức thấp nhất, giải pháp được phát biểu theo cách có thể cài đặt trực tiếp.

Trong mỗi bước của tiến trình đều là sự làm mịn cho một mức trừu tượng của giải pháp. Khi chuyển qua các mức trừu tượng khác nhau, chúng ta làm việc để tạo ra các trừu tượng thủ tục, trừu tượng dữ liệu và trừu tượng điều khiển.

- Trừu tượng thủ tục: là một dãy các lệnh có tên, có một chức năng xác định và giới hạn.

- Trừu tượng dữ liệu: là tập hợp các dữ liệu có tên mô tả cho một sự vật dữ liệu. Đối tượng dữ liệu này thực chất là một tập hợp nhiều mẫu thông tin khác nhau và ta có thể tham khảo tới bằng cách nói tên của trừu tượng dữ liệu. Trừu tượng dữ liệu làm cho người thiết kế có thể xác định một sự vật dữ liệu trong hoàn cảnh các thao tác (thủ tục) có thể áp dụng vào nó.

- Trừu tượng điều khiển: áp dụng cho cơ chế điều khiển chương trình mà không xác định các chi tiết bên trong.

4.3.2. Làm mịn (Refinement)

Làm mịn là chiến lược thiết kế trên xuống. Kiến trúc của một chương trình được phát triển bằng cách các mức làm mịn liên tiếp các thủ tục. Trong mỗi bước, một hay nhiều lệnh của chương trình đã cho được phân rã thành những lệnh chi tiết hơn. Việc phân rã hay làm mịn liên tiếp các đặc tả này kết thúc khi tất cả các lệnh đã được diễn đạt dưới dạng bất kỳ ngôn ngữ lập trình hay ngôn ngữ máy tính nền tảng nào. Khi các nhiệm vụ đã được làm mịn thì dữ liệu cũng phải được làm mịn, được phân rã hay cấu trúc lại.

Cần chú ý là mọi bước làm mịn đều kéo theo những quyết định thiết kế nào đó. Người lập trình cần nhận biết các tiêu chuẩn nền tảng cho quyết định thiết kế và sự tồn tại của các giải pháp khác.

4.3.3. Tính module

Phần mềm được chia thành các phần có tên riêng biệt và định địa chỉ được, gọi là các module. Các module được tích hợp để thỏa mãn yêu cầu của vấn đề.

Gọi C(x) là hàm xác định độ phức tạp cảm nhận được của vấn đề x, và E(x) là hàm xác định nổ lực cần có để giải quyết vấn đề x. Với hai vấn đề p, q ta có

Nếu C(p)>C(q) thì tổng quát ta suy ra E(p)>E(q) vì phải mất nhiều nổ lực hơn để giải quyết vấn đề khó hơn.

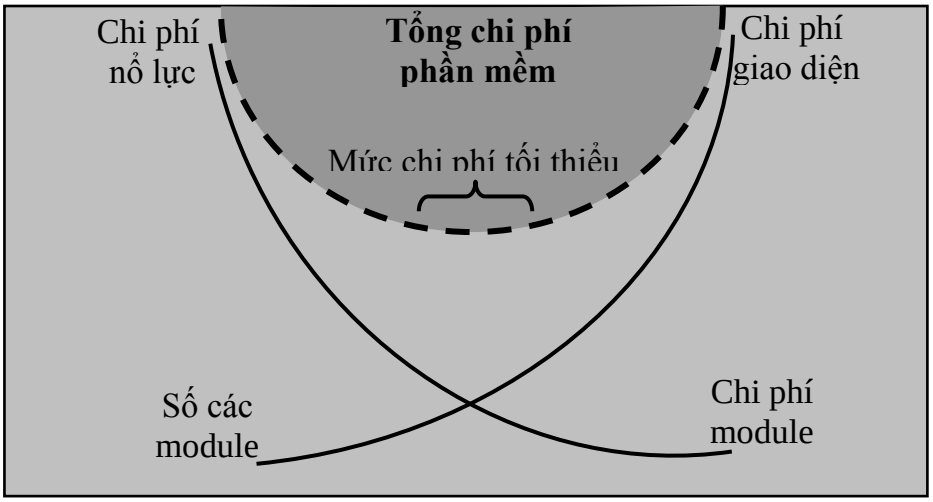
Một đặc trưng được chỉ qua thực nghiệm là: C(p+q)>C(p)+C(q)

Như thế, độ phức tạp cảm nhận của vấn đề tổ hợp p và q là lớn hơn độ phức tạp cảm nhận được khi tách biệt từng vấn đề p và q để xem xét.

Kết hợp ta có E(p+q)>E(p)+E(q). Điều này dẫn đến kết luận chia để trị cho việc giải quyết các vấn đề phức tạp. Đây là một luận cứ cho tính module.

Theo lập luận trên, liệu chúng ta chia phần mềm một cách không xác định thì nổ lực cần để phát triển nó sẽ trở thành nhỏ đến mức có thể bỏ qua được? Không may, câu trả lời là không vì lúc này sẽ xuất hiện các các lực khác như chi phí kết nối các module, chi phí xây dựng giao diện,... làm tăng chi phí nổ lực để giải quyết vấn đề.

Miền chi phí tối thiểu được chỉ ra ở sơ đồ sau:



4.3.4. Kiến trúc phần mềm

Kiến trúc phần mềm được suy dẫn ra qua tiến trình phân hoạch đặt mối quan hệ giữa các phần tử của giải pháp phần mềm với các bộ phận của thế giới thực được xác định không tường minh trong phân tích yêu cầu.

Kiến trúc phần mềm gồm có hai đặc trưng quan trọng

i. Cấu trúc phân cấp của các thành phần thủ tục (module): cấp bậc cây điều khiển.

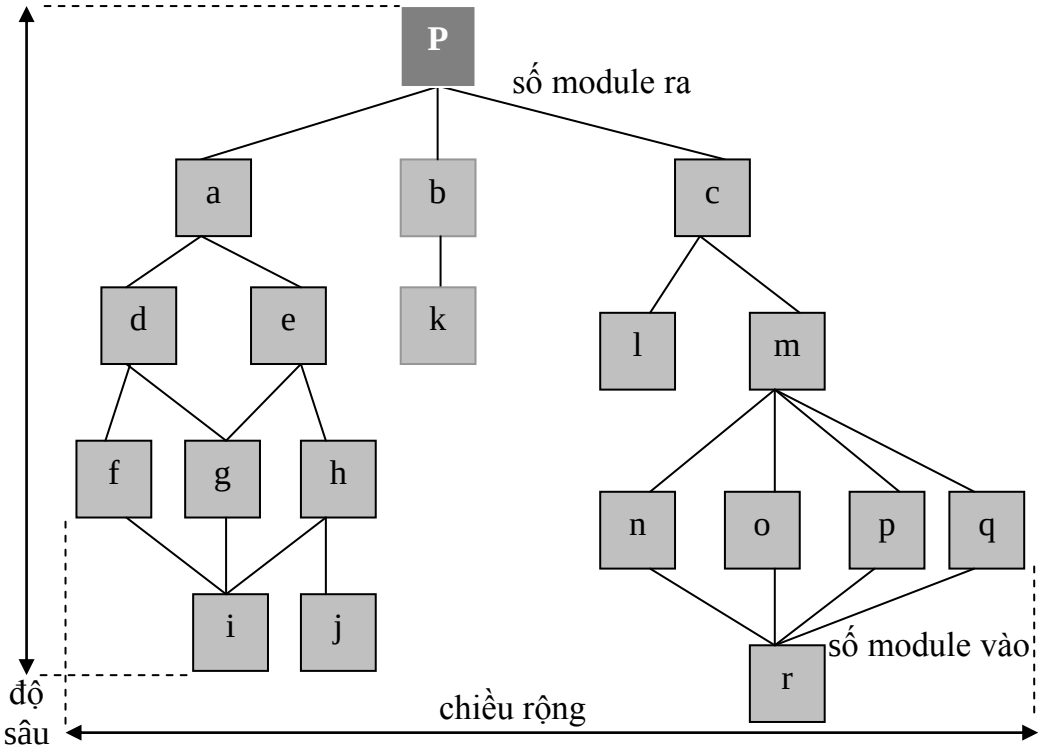
ii. Cấu trúc dữ liệu.

1. Cấp bậc điều khiển

Cấp bậc điều khiển còn gọi là cấu trúc chương trình. Nó biểu thị cho cách tổ chức của các thành phần module.

Một số chỉ số trên cây điều khiển được quan tâm:
+ Chiều rộng: độ trải rộng toàn bộ của điều khiển.
+ Độ sâu: chỉ báo về số mới điều khiển
+ Số module ra: là độ đo số các module trực tiếp bị điều khiển của một module khác.
+ Số module vào: số module trực tiếp điều khiển một module đã cho.
+ Thượng cấp: module điều khiển một module khác.
+ Thuộc cấp: một module bị module khác điều khiển.
+ Tính thấy được.
+ Tính nối được.
+ Tính cố kết,...

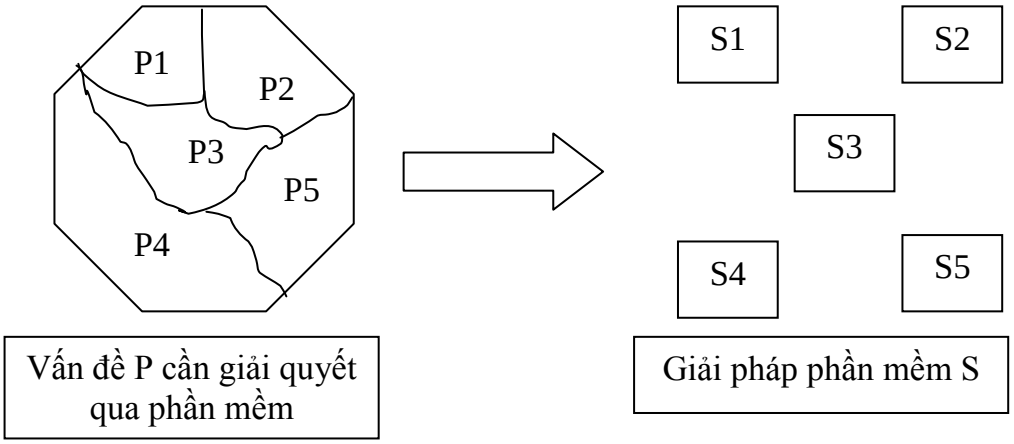
Cấu trúc chương trình và các chỉ số được minh hoạ như hình sau:



2. Cấu trúc dữ liệu

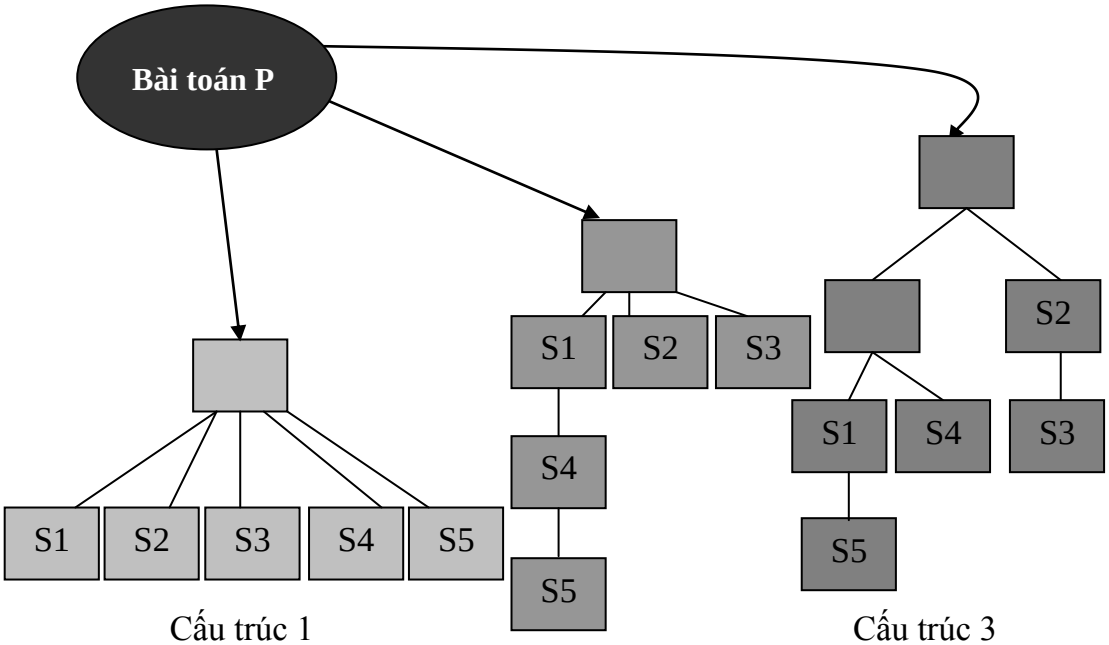
Cấu trúc dữ liệu biểu diễn cho mối quan hệ logic giữa các phần dữ liệu riêng lẻ, một số cấu trúc dữ liệu thường sử dụng như: khoảng mục vô hướng, vector tuần tự, danh sách móc nối, không gian n chiều, cây cấp bậc,...

Ta xét phần mềm P cần giải quyết qua phần mềm và giải pháp phần mềm S được chọn như hình sau:



Rõ ràng, khi giải quyết một vấn đề, chúng ta có nhiều giải pháp phần mềm. Mỗi giải pháp S_i ta có một cấu trúc khác nhau.

Xét bài toán P được cho như sau



Ta tháy, mỗi cấu trúc dựa trên nền tảng khác nhau về khái niệm thiết kế "tốt" và câu hỏi "cái nào tốt nhất" chúng ta rất khó để trả lời nếu không muốn nói là không trả lời được. Chi tiết của vần đề này được bàn chi tiết ở mục 4.6.

4.3.5. Che dấu thông tin

Che dấu thông tin là khái niệm các module nên được đặc trưng bởi những quyết định thiết kế mà ẩn kín với mọi module khác, thông tin chứa trong module này không thể thâm nhập tới được từ các module khác không cần đến những thông tin đó. Che dấu thông tin kéo theo việc xác định một tập module độc lập mà trao đổi giữa các module chỉ là các thông tin thật sự cần thiết cho việc vận hành phần mềm.

Che dấu thông tin là một tiêu chuẩn thiết kế đối với hệ thống module vì những lợi ích mà nó mang lại. Khi có sai sót xảy ra, sự thay đổi sẽ ít có khả năng lan truyền sang những vị trí khác bên trong phần mềm.

4.3.6. Thiết kế module

Sự trừu tượng hóa và che dấu thông tin được dùng để thiết kế module. Bên trong cấu trúc chương trình, các module có thể được phân loại:

+ Module tuần tự: được tham khảo và thực hiện không bị ngắt

+ Module tăng trưởng: có thể bị ngắt trước khi hoàn tất và sau đó chạy lại tại thời điểm ngắt.

+ Module song song: thực hiện đồng thời với module khác.

Với mỗi module cần có:

+ tính độc lập chức năng,

+ tính cố kết,

+ tính gắn nối,...

4.4. CHẤT LƯỢNG THIẾT KẾ

Như đã đề cập ở trên, rất khó để có thể xác định được thế nào là thiết kế tốt, nó phụ thuộc vào ứng dụng và vào yêu cầu dự án. Một thiết kế tốt phải là một thiết kế mà nó cho phép sản sinh ra mã hữu hiệu; nó có thể là một thiết kế tối thiểu mà theo đó là việc thực hiện là càng chặt càng tốt; hoặc nó là thiết kế bảo dưỡng được tốt nhất hay chỉ là tiêu chuẩn tốt cho người dùng. Một thiết kế bảo dưỡng tốt có thể thích nghi với việc cải biên các chức năng và việc thêm các chức năng mới. Do đó, thiết kế như thế phải là dễ hiểu và việc sửa đổi chỉ có hiệu ứng cục bộ. Các thành phần của thiết kế là kết dính theo nghĩa là tất cả các phần trong thành phần đó phải có một quan hệ logic chặt chẽ. Các thành phần ấy phải là được nối ghép lỏng lẻo. Sự nối ghép là một độ đo của tính độc lập của các thành phần. Nối ghép càng lỏng lẻo thì càng dễ thích nghi.

Thực tế, đã có một vài công việc được tiến hành để thiết lập độ đo chất lượng thiết kế dùng để xem một thiết kế có là tốt hay không.

4.4.1. Sự kết dính

Sự kết dính của một thành phần là độ đo về tính khớp lại với nhau. Một thành phần hẳn thực hiện một chức năng logic hoặc thực hiện một thực thể logic. Tất cả các phần của thành phần đó đều tham gia vào việc thực hiện đó. Nếu một thành phần không trực tiếp tham gia vào việc chức năng logic đó thì mức độ kết dính của nó là thấp.

Theo một số chuyên gia thì có bảy mức kết dính theo thứ tự tăng dần sau đây:

a) Kết dính gom góp: Các phần của thành phần không liên quan với nhau, song lại bị bó vào một thành phần.

b) Hội hợp logic: Các thành phần cùng thực hiện các chức năng tương tự chẳng hạn như vào, xử lý lỗi,... là được đặt vào cùng một thành phần.

c) Kết dính theo thời điểm: Tất cả các thành phần cùng hoạt hoá một lúc, chẳng hạn như khởi sự và kết thúc, là được bó lại với nhau.

d) Kết dính thủ tục: Các phần tử trong thành phần được ghép lại trong một dãy điều khiển.

e) Kết dính truyền thông: Tất cả các phần tử của thành phần cùng thao tác trên một dữ liệu vào và đưa cùng một dữ liệu ra.

f) Kết dính tuần tự: Trong một thành phần, ra của phần tử này là vào của phần tử khác.

g) Kết dính chức năng: Mỗi phần của thành phần đều là cần thiết để thi hành cùng một chức năng nào đó.

4.4.2. Sự ghép nối

Ghép nối liên quan đến kết dính. Nó chỉ ra độ ghép nối giữa các đơn vị của chương trình. Hệ thống có ghép nối cao sẽ có độ ghép nối mạnh giữa các đơn vị, các đơn vị phụ thuộc lẫn nhau. Hệ thống nối ghép lỏng lẻo làm cho các đơn vị là độc lập hoặc là tương đối độc lập với nhau.

Các module là được ghép nối chặt chẽ nếu chúng dùng các biến chung và nếu chúng trao đổi các thông tin điều khiển (ghép nối chung nhau và ghép nối điều khiển).

Ghép nối lỏng lẻo đạt được khi đảm bảo rằng các thông tin biểu diễn là được giữ trong thành phần này và giao diện dữ liệu của nó với các đơn vị khác lại thông qua danh sách tham số của nó.

Tiêu chuẩn của các ghép nối trong chương trình được đánh giá như sau:

- Một phép nối chuẩn là một phép gọi tới một module khác bằng tên.
- Một phép nối điều khiển xấu là một phép chuyển tới bất kỳ một điểm vào thứ cấp nào được xác định trong một module khác.
- Một phép nối dữ liệu xấu là một sự liên hệ tới dữ liệu xác định trong một module khác nhưng không phải bởi một phép truyền tường minh qua một phép nối chuẩn.
- Một ghép cặp (couple) tính toán là một mục dữ liệu dùng chung được module nhận dùng làm cơ sở tính toán hay lập chỉ mục, nhưng không phải để lập quyết định.

4.4.3. Sự hiểu được

Sự hiểu được liên quan tới một số các đặc trưng thành phần sau đây:

a) Tính kết dính: Có thể hiểu được thành phần đó mà không cần tham khảo đến một thành phần nào khác hay không?

b) Đặt tên: Phải chăng là mọi tên được dùng trong thành phần đó đều có nghĩa? Tên có nghĩa là những tên phản ánh của thực thể trong thế giới thực được mô hình bởi thành phần đó.

c) Soạn tư liệu: Thành phần có được soạn thảo tư liệu sao cho ánh xạ giữa các thực thể của thế giới thực và thành phần đó là rõ ràng?

d) Độ phức tạp: độ phức tạp của các thuật toán được dùng để thực hiện thành phần đó là như thế nào? Từ phức tạp ở đây được dùng theo nghĩa không hình thức. Độ phức tạp cao ám chỉ nhiều quan hệ giữa các thành phần khác nhau của thành phần thiết kế đó và một cấu trúc logic phức tạp mà nó dính líu đến độ sâu lồng nhau của phát biểu. Các thành phần phức tạp là khó hiểu, vì thế người thiết kế nên làm cho thiết kế thành phần càng đơn giản càng tốt.

Đa số công việc về đo chất lượng thiết kế được tập trung vào cố gắng đo độ phức tạp của thành phần và từ đó thu được một vài độ đo về sự dễ hiểu của thành phần. Độ phức tạp phản ánh độ dễ hiểu, nhưng cũng có một số các nhân tố khác ảnh hưởng đến độ dễ hiểu, chẳng hạn như tổ chức dữ liệu và kiểu cách mô tả thiết kế. Các số đo độ phức tạp có thể chỉ cung cấp một chỉ số cho độ dễ hiểu của một thành phần.

Sự thừa kế trong một thiết kế hướng đối tượng phản ánh độ dễ hiểu. Nếu sự thừa kế được dùng để gắn các chi tiết thiết kế thì thiết kế sẽ dễ hiểu hơn. Mặc khác nếu sử dụng sự thừa kế đòi hỏi người đọc thiết kế phải phải nhìn nhiều lớp đối tượng khác nhau trong tôn ti thừa kế thì độ dễ hiểu của thiết kế là được rút gọn.

4.4.4. Sự thích nghi được

Nếu một thiết kế là nhằm được bảo trì thì nó phải là sẵn sàng thích nghi được. Dĩ nhiên điều đó suy ra rằng các thành phần của chúng nên được ghép nối lỏng lẻo. Tuy nhiên sự thích nghi được lại có nghĩa là thiết kế đó phải được soạn thảo tư liệu tốt, tư liệu thành phần phải là dễ hiểu và kiên định với sự thực hiện, và nghĩa là sự thực hiện phải được viết ra trong cách dễ đọc.

Một thiết kế dễ thích nghi hẳn là có mức nhìn thấy được cao. Nó có một quan hệ rõ ràng giữa các mức khác nhau của thiết kế. Người đọc thiết kế có thể tìm được các biểu diễn liên quan sao cho lược đồ cấu trúc biểu diễn sự vận chuyển của biểu đồ dòng dữ liệu.

Cần phải dễ dàng kết hợp chặt chẽ các biến đổi về thiết kế trong toàn bộ tư liệu thiết kế.

Nếu không như vậy thì các thay đổi thiết kế có thể sẽ không được đưa vào trong các mô tả liên quan. Tư liệu thiết kế đó có thể trở nên không kiên định. Các biến đổi tiếp sau là khó thực hiện (thành phần thiết kế này là ít thích nghi được) vì rằng sự cải biên đó không thể dựa vào tính kiên định của của tư liệu thiết kế.

Để có độ thích nghi tối ưu thì mỗi thành phần phải là tự chứa. Một thành phần có thể là ghép nối lỏng lẻo theo nghĩa chỉ là hợp tác với các thành phần khác thông qua việc chuyển các thông báo. Điều này không giống như là tự chứa vì rằng thành phần đó có thể dựa trên các thành phần khác chẳng hạn như các chức năng hệ thống hoặc các chức năng xử lý sai. Sự thích nghi với một thành phần có thể dính líu với sự thay đổi các phần của thành phần đó mà nó dựa trên các chức năng ngoại nên đặc tả của các chức năng ngoại này cũng xét đến sự cải biên đó.

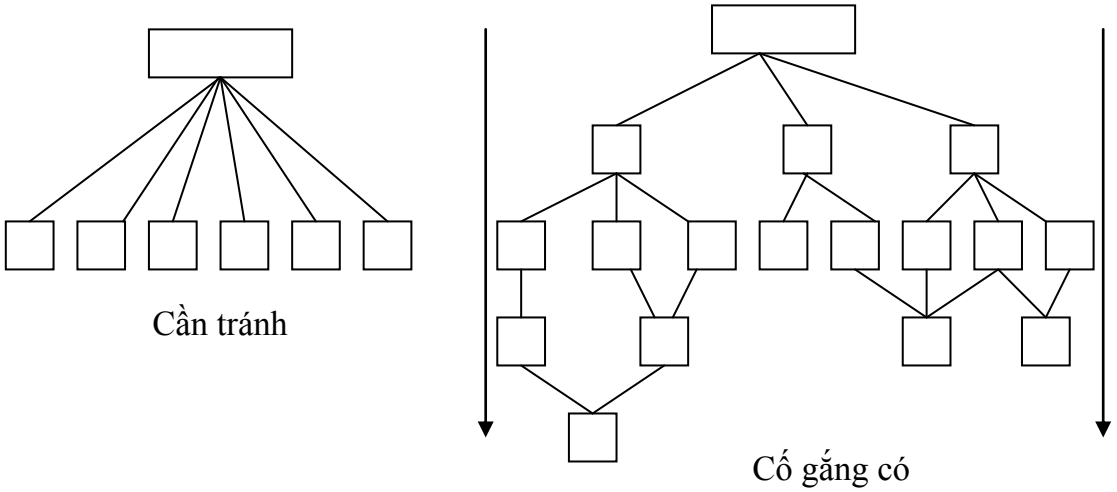
Muốn tự chứa một cách hoàn toàn thì một thành phần không nên dùng các thành phần khác được xác định ngoại lai. Tuy nhiên, điều đó lại mâu thuẫn với kinh nghiệm nói rằng các thành phần hiện có nên là dùng lại được. Vậy là cần có một cân bằng giữa tính ưu việt của sự dùng lại các thành phần và sự mất mát tính thích nghi được của thành phần.

4.4.5. Một số yêu cầu thiết kế

+ Linh hoạt đối với những yêu cầu thay đổi không định trước.
+ Dễ thử nghiệm.
+ Tính sáng sủa, dễ đọc
+ Kích thước module nhỏ
+ Tính độc lập module (tính mở/đóng giữa các module), sử dụng yếu tố "hộp đen"
+ Phải quan hệ chặt chẽ giữa thiết kế và yêu cầu.
+ Mỗi module hoàn toàn độc lập, nó thực hiện một chức năng duy nhất và thực hiện trọn vẹn chức năng đó.
+ Mọi thứ trong module ràng buộc với nhau qua việc xử lý nối tiếp nhau trên cùng một dòng dữ liệu.
+ Mọi thứ trong module được điều khiển bởi cùng một dữ liệu vào, hay cùng một phức hợp thiết bị, hay cùng thực hiện từng phần của cùng một kết xuất.

+ Module có thể hiểu được hoàn toàn dựa vào những tham biến truyền cho nó và nhận từ nó.

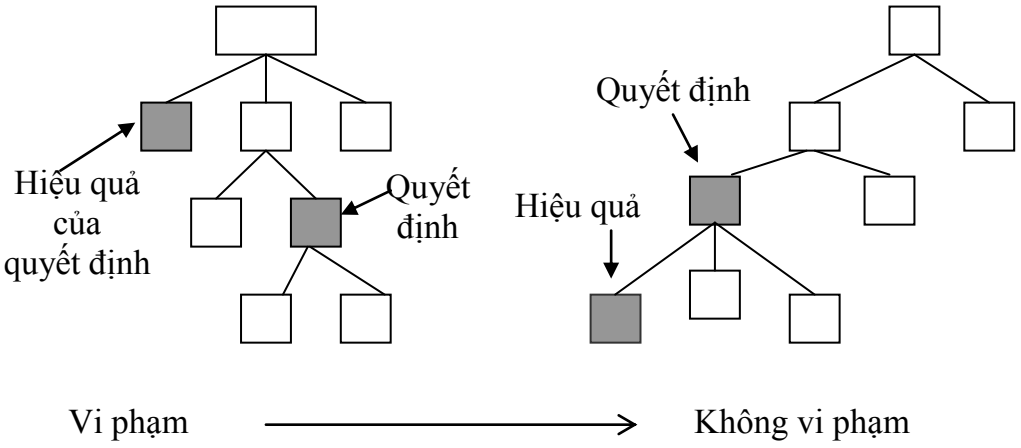
+ Khi thiết kế cố gắng giảm thiểu cấu trúc bằng việc tản ra nhiều và cố gắng co cụm khi chiều sâu tăng thêm.



+ Giữ phạm vi hiệu quả của một module bên trong phạm vi kiểm soát của module đó.

- Phạm vi hiệu quả của module m được định nghĩa là tất cả các module khác bị ảnh hưởng bởi một quyết định thực hiện trong module m.

- Phạm vi kiểm soát của module m là tất cả các module và mọi module thuộc cấp và thuộc cấp cuối cùng đối với module m.



+ Ước lượng giao diện module để giảm độ phức tạp, dư thừa và tăng tính nhất quán.

+ Xác định các module có chức năng dự kiến được.

+ Cố gắng giữ các module một đầu vào và một đầu ra, tránh các "mối nối bệnh hoạn".

4.5. CHIẾN LƯỢC THIẾT KẾ

Do các hệ phần mềm lớn là phức tạp nên người ta thường dùng các phương pháp tiếp cận khác nhau trong việc thiết kế các phần khác nhau của một hệ thống. Chẳng có một chiến lược tốt nhất nào cho các dự án. Hai chiến lược thiết kế hiện đang được dùng rộng rãi trong việc phát triển phần mềm đó là thiết kế hướng chức năng và thiết kế hướng đối tượng. Mỗi chiến lược thiết kế đều có ưu và nhược điểm riêng phụ thuộc vào ứng dụng phát triển và nhóm phát triển phần mềm.

Cách tiếp cận hướng chức năng hay hướng đối tượng là bổ sung và hỗ trợ cho nhau chứ không phải là đối kháng nhau. Kỹ sư phần mềm sẽ chọn cách tiếp cận thích hợp nhất cho từng giai đoạn thiết kế.

4.5.1. Thiết kế hướng chức năng

Thiết kế hướng chức năng là một cách tiếp cận thiết kế phần mềm trong đó bản thiết kế được phân giải thành một bộ các đơn thể được tác động lẫn nhau, mà một đơn thể có một chức năng được xác định rõ ràng. Các chức năng có các trạng thái cục bộ nhưng chúng chia sẻ với nhau trạng thái hệ thống, trạng thái này là tập trung và mọi chức năng đều có thể truy cập được.

Có người nghĩ rằng thiết kế hướng chức năng đã lỗi thời và nên được thay thế bởi cách tiếp cận hướng đối tượng. Thế nhưng, nhiều tổ chức đã phát triển các chuẩn và các phương pháp dựa trên sự phân giải chức năng. Nhiều phương pháp thiết kế kết hợp với các công cụ CASE đều là hướng chức năng và có nhiều hệ thống đã được phát triển bằng cách sử dụng phương pháp tiếp cận hướng chức năng. Các hệ thống đó sẽ phải được bảo trì cho một tương lai xa xôi. Bởi vậy thiết kế hướng chức năng vẫn sẽ còn được tiếp tục sử dụng rộng rãi. Đó là của thiên hạ. Còn người Việt Nam chúng ta, chúng ta chưa có tập quán dùng một phương pháp thiết kế nào, liệu chúng ta có nhất thiết phải đi theo trào lưu đó hay chúng ta nên đi thẳng vào phương pháp nào hữu hiệu nhất?

Người ta dùng các biểu đồ dòng dữ liệu - mà nó mô tả việc xử lý dữ liệu logic, các lược đồ cấu trúc - nó chỉ ra cấu trúc của phần mềm và các mô tả PDL (page description language) - nó mô tả thiết kế chi tiết. Khái niệm dòng dữ liệu đã bị cải biên làm cho nó thích hợp hơn việc sử dụng một hệ thống vẽ biểu đồ tự động và sử dụng một dạng lược đồ cấu trúc có kèm thêm các thông tin điều khiển.

Chiến lược thiết kế hướng chức năng dựa trên việc phân giải hệ thống thành một bộ các chức năng có tương tác nhau với trạng thái hệ thống tập trung dùng chung cho các chức năng đó. Các chức năng này có thể có các thông tin trạng thái cục bộ nhưng chỉ dùng cho quá trình thực hiện chức năng đó mà thôi.

Thiết kế hướng chức năng gắn với các chi tiết của một thuật toán của chức năng đó nhưng các thông tin trạng thái hệ thống là không bị che dấu. Điều này có thể gây ra một vấn đề vì rằng một chức năng có thể thay đổi trạng thái theo một cách mà các chức năng khác không ngờ tới. Việc thay đổi một chức năng và cách nó sử dụng trạng thái hệ thống có thể gây ra những tương tác bất ngờ đối với các chức năng khác.

Do đó cách tiếp cận chức năng để thiết kế là thắng lợi nhất khi mà khối lượng thông tin trạng thái hệ thống là được làm nhỏ nhất và thông tin dùng chung nhau là rõ ràng.

4.5.2. Thiết kế hướng đối tượng

Hệ thống được nhìn nhận như một bộ các đối tượng (chứ không phải là bộ các chức năng). Hệ thống được phân tán, mỗi đối tượng có những thông tin trạng thái riêng của nó. Đối tượng là một bộ các thuộc tính xác định trạng thái của đối tượng đó và các phép toán của nó. Nó được thừa kế từ một vài lớp đối tượng lớp cao hơn, sao cho dễ định nghĩa nó chỉ cần nêu đủ các khác nhau giữa nó và các lớp cao hơn nó.

Che dấu thông tin là chiến lược thiết kế dấu càng nhiều thông tin trong các thành phần càng hay. Cái đó ngầm hiểu rằng việc kết hợp điều khiển logic và cấu trúc dữ liệu được thực hiện trong thiết kế càng chậm càng tốt. Liên lạc thông qua các thông tin trạng thái dùng chung (các biến tổng thể) là ít nhất, nhờ vậy khả năng hiểu là được tăng lên. Thiết kế là tương đối dễ thay đổi vì sự thay đổi một thành phần không thể không dự kiến các hiệu ứng phụ trên các thành phần khác.

Thiết kế hướng đối tượng là dựa trên việc che dấu thông tin, nhìn hệ phần mềm như là một bộ các đối tượng tương tác với nhau chứ không phải là một bộ các chức năng như cách tiếp cận chức năng. Các đối tượng này có một trạng thái được che dấu và các phép toán trên các trạng thái đó. Thiết kế biểu thị các dịch vụ được yêu cầu và được cung cấp bởi các đối tượng có tương tác với nó.

Thiết kế hướng đối tượng có ba đặc trưng

i) Vùng dữ liệu dùng chung là bị loại bỏ. Các đối tượng liên lạc với nhau bằng cách trao đổi thông báo chứ không phải bằng các biến dùng chung.

ii) Các đối tượng là các thực thể độc lập mà chúng sẵn sàng được thay đổi vì rằng tất cả các trạng thái và các thông tin biểu diễn là chỉ ảnh hưởng trong phạm vi chính đối tượng đó thôi. Các thay đổi về biểu diễn thông tin có thể được thực hiện không cần sự tham khảo tới các đối tượng hệ thống khác.

iii) Các đối tượng có thể phân tán và có thể hành động tuần tự hoặc song song. Không cần có quyết định về tính song song ngay từ một giai đoạn sớm của quá trình thiết kế.

Các ưu điểm

i) Dễ bảo trì vì các đối tượng là độc lập. Các đối tượng có thể hiểu và cải biên như là một thực thể độc lập. Thay đổi trong thực hiện một đối tượng hoặc thêm các dịch vụ sẽ không làm ảnh hưởng tới các đối tượng hệ thống khác.

ii) Các đối tượng là các thành phần dùng lại được thích hợp (do tính độc lập của chúng). Một thiết kế có thể dùng lại được các đối tượng đã được thiết kế trong các bản thiết kế trước đó.

iii) Đối với một vài lớp hệ thống, có một phản ánh rõ ràng giữa các thực thể có thực (chẳng hạn như các thành phần phần cứng) với các đối tượng điều khiển nó trong hệ thống. Điều này cải thiện được tính dễ hiểu của thiết kế.

Nhược điểm:

Sự nhận minh các đối tượng hệ thống thích hợp là khó khăn. Cách nhìn tự nhiên nhiều hệ thống là cách nhìn chức năng và việc thích nghi với cách nhìn hướng đối tượng đôi khi là khó khăn.

Phương pháp thiết kế hướng đối tượng vẫn còn là tương đối chưa chín muồi và đang thay đổi mau chóng.

Ở đây, cần phân biệt hai khái niệm là thiết kế hướng đối tượng và lập trình (cài đặt) hướng đối tượng:

+ Thiết kế hướng đối tượng là một chiến lược thiết kế nó không phụ thuộc vào một ngôn ngữ thực hiện cụ thể nào. Các ngôn ngữ lập trình hướng đối tượng và các khả năng bao gói đối tượng làm cho thiết kế hướng đối tượng được thực hiện một cách đơn giản hơn. Tuy nhiên một thiết kế hướng đối tượng cũng có thể được thực hiện trong một ngôn ngữ kiểu như Pascal hoặc C mà không có các đặc điểm như vậy.

+ Việc chấp nhận thiết kế hướng đối tượng như là một chiến lược hữu hiệu đã dẫn đến sự phát triển phương pháp thiết kế hướng đối tượng. Như Ada không phải là ngôn ngữ lập trình hướng đối tượng vì nó không trợ giúp sự thừa kế của các lớp, nhưng lại có thể thực hiện các đối tượng trong Ada bằng cách sử dụng các gói hoặc các nhiệm vụ, do đó Ada được dùng để mô tả các thiết kế hướng đối tượng.

+ Thiết kế hướng đối tượng là một chiến lược thiết kế, nó không phụ thuộc vào ngôn ngữ để thực hiện. Các ngôn ngữ lập trình hướng đối tượng và khả năng bao gói dữ liệu làm cho dễ thực hiện một thiết kế hướng đối tượng hơn. Tuy nhiên cũng có thể thực hiện một thiết kế hướng đối tượng trong một ngôn ngữ kiểu như Pascal hoặc C.

4.6. THIẾT KẾ KIẾN TRÚC ỨNG DỤNG VÀ CÁC MÔ HÌNH CHO THIẾT KẾ ỨNG DỤNG

4.6.1. Thiết kế kiến trúc ứng dụng

Như đã nói ở trên, kiến trúc của phần mềm ứng dụng được suy dẫn ra qua tiến trình phân hoạch đặt mối quan hệ giữa các phần tử của giải pháp phần mềm với các bộ phận của thế giới thực được xác định không tường minh trong phân tích yêu cầu. Các hệ thống lớn có thể được phân rã thành các phân hệ cung cấp các dịch vụ. Mỗi phân hệ có các module và có một giao diện xác định để giao tiếp với các phân hệ khác. Nó là một hệ thống có quyền riêng của mình cho phép các hoạt động không phụ thuộc vào các dịch vụ của các phân hệ khác.

Quy trình thiết kế khởi đầu để xác định các phân hệ của hệ thống và thiết lập một khuôn khổ điều khiển và truyền thông giữa các phân hệ được gọi là thiết kế kiến trúc cho ứng dụng. Thiết kế kiến trúc ứng dụng luôn được tiến hành trước khi có các đặc tả chi tiết về hệ thống.

Việc phân rã kiến trúc hệ thống là cần thiết cho việc cấu trúc và tổ chức đặc tả và chứa các hoạt động sau:

- Cấu trúc hệ thống: hệ thống được cấu trúc thành một số các phân hệ, mỗi phân hệ là một đơn vị phần mềm độc lập. Các liên kết thông tin giữa các phân hệ được xác định.

- Mô hình hoá điều khiển: một mô hình chung về các quan hệ điều khiển giữa các thành phần của hệ thống được thiết lập.
- Phân rã mô hình: mỗi phân hệ đã xác định sẽ được phân rã thành các module. Kiến trúc sư sẽ phải quyết định các kiểu module và các kết nối.

Kết quả của thiết kế kiến trúc ứng dụng là tài liệu thiết kế kiến trúc. Nó bao gồm một số các biểu diễn đồ hoạ về các mô hình hệ thống kèm theo các giải thích bằng văn bản. Nó mô tả tại sao hệ thống được phân rã thành các phân hệ và các phân hệ lại được phân rã thành các module.

Giai đoạn đầu tiên của thiết kế kiến trúc là việc phân rã hệ thống thành nhiều phân hệ tương tác nhau. Tại mức trừu tượng nhất, một thiết kế kiến trúc có thể được coi như là một sơ đồ khối trong đó mỗi khối đại diện cho một phân hệ. Các hộp nằm trong một khối được coi là phân hệ con của phân hệ. Các mũi tên đại diện cho các điều khiển hoặc các dữ liệu giao tiếp giữa các phân hệ. Sau đó, một số mô hình đặc trưng hơn được phát triển để biểu diễn các dữ liệu chia sẻ, các giao diện và phân phối dữ liệu giữa các phân hệ trong ứng dụng.

4.6.2. Các mô hình thiết kế ứng dụng

4.6.2.1. Mô hình kho dữ liệu

Các phân hệ cần trao đổi thông tin, và nó có thể tiến hành theo hai cách:

i. Mỗi phân hệ duy trì một cơ sở dữ liệu riêng của mình. Dữ liệu được trao đổi giữa các phân hệ bằng cách chuyển đổi các thông báo.

ii. Mọi dữ liệu được lưu trữ tại một cơ sở dữ liệu trung tâm có thể được truy cập bởi mọi phân hệ. Mô hình này gọi là mô hình kho dữ liệu.

Mô hình kho dữ liệu phù hợp cho các ứng dụng khi dữ liệu được tạo bởi một phân hệ và được sử dụng bởi các phân hệ khác. Đây là cách hữu hiệu để chia sẻ một số lượng lớn dữ liệu mà không cần chuyển đổi dữ liệu tường minh từ một phân hệ này tới các phân hệ khác.

Phân hệ phải chấp nhận mô hình này nếu muốn tham gia hệ thống. Sẽ rất khó tích hợp một phân hệ mới nếu nó không phù hợp với tiêu chuẩn của kho dữ liệu. Phân hệ tạo dữ liệu không cần liên quan đến việc dữ liệu được phân hệ khác sử dụng như thế nào. Việc phát triển mô hình sẽ khó khăn khi một số lượng lớn dữ liệu đã có theo tiêu chuẩn cũ. Việc chuyển đổi dữ liệu sẽ rất tốn kém. Các hoạt động như lưu trữ, bảo mật, điều khiển truy nhập và khôi phục được tập trung hoá.

Tuy nhiên, các phân hệ có thể có các yêu cầu khác nhau về mức độ bảo mật, khôi phục và chiến lược lưu trữ. Mô hình này bắt buộc các phân hệ phải chấp nhận một chính sách chung. Mọi việc sẽ đơn giản nếu phân hệ mới cần tích hợp tương thích với dữ liệu cũ. Tuy nhiên sẽ khó khăn nếu phân phối dữ liệu tới nhiều máy khác nhau. Việc này sẽ phát sinh khả năng dư thừa dữ liệu, không toàn vẹn.

4.6.2.2. Mô hình khách - phục vụ

Mô hình khách - phục vụ là một mô hình hệ thống phân tán biểu diễn việc phân tán các dữ liệu và xử lý trên nhiều máy tính khác nhau. Các thành phần chính là:

- Một tập các server độc lập phục vụ cho các phân hệ.
- Một tập các khách hàng yêu cầu các dịch vụ. Chúng có thể là các phân hệ, hay là các thể hiện khác nhau của cùng một chương trình.
- Một mạng cho phép các khách hàng có thể truy nhập được các dịch vụ.

Khách hàng phải biết được định danh của các dịch vụ, còn các dịch vụ không cần biết các định danh của khách hàng.

Ưu điểm quan trọng nhất của mô hình này là sự phân tán rất rõ ràng. Mô hình này dễ dàng thêm một server và tích hợp dần dần khi có nhu cầu mà không ảnh hưởng tới các thành phần cũ. Sự thiếu vắng của mô hình chia sẻ dữ liệu ở đây có nghĩa là sẽ khó dự đoán được các vấn đề khi tích hợp dữ liệu vào hệ thống cũ. Mỗi server phải có trách nhiệm với bản thân mình về lưu trữ, khôi phục,...Không có một trung tâm nên khách hàng phải tự biết và tìm server, đây là vấn đề khó khăn đối với các mạng lớn như WAN, Internet.

4.6.2.3. Mô hình máy trừu tượng

Mô hình máy trừu tượng đôi khi gọi là mô hình lớp, mô hình hoá giao diện của các phần mềm. Nó tổ chức một hệ thống thành một dãy các lớp cung cấp các dịch vụ khác nhau. Mỗi lớp xác định một máy trừu tượng, ngôn ngữ máy của lớp này sẽ làm cơ sở để thực hiện cho lớp kế tiếp. Nhược điểm là hệ thống được cấu trúc theo cách này tương đối phức tạp. Việc thực hiện cũng gặp vấn đề do nhiều lớp của cùng một giao tiếp cần phải có.

4.6.2.4. Mô hình điều khiển

Là mô hình mà để vận hành, hệ thống phải được điều khiển làm việc đồng bộ và đúng. Mô hình cấu trúc không có các thông tin điều khiển mà các luồng điều khiển được chỉ ra ở mô hình điều khiển. Hai cách tiếp cận chung có thể xác định là:

- Điều khiển tập trung,
- Điều khiển trên cơ sở sự kiện

Mô hình điều khiển bổ sung cho mô hình cấu trúc. Mỗi mô hình cấu trúc đã nói trên đều có thể dùng mô hình điều khiển tập trung hoặc mô hình điều khiển trên cơ sở sự kiện.

1. Mô hình điều khiển tập trung

Một phân hệ được thiết kế như bộ điều khiển hệ thống có trách nhiệm quản lý việc thực hiện các phân hệ khác. Các mô hình điều khiển tập trung phân lớp theo hai loại phụ thuộc việc điều khiển được tiến hành tuần tự hay song song.

+ Mô hình gọi - trả lời: Mô hình này phù hợp với các mô hình thủ tục top - down.

+ Mô hình quản lý: Mô hình này thích hợp với các hệ thống đồng thời. Một cấu thành hệ thống được thiết kế như là một bộ quản trị và điều khiển việc khởi động, kết thúc và phối hợp các phân hệ khác.

2. Mô hình hệ thống điều khiển bởi sự kiện

Mô hình hệ thống điều khiển bởi sự kiện có nhiều kiểu khác nhau của hệ thống hướng sự kiện, như:

- Mô hình phát tin: Trong mô hình này, về nguyên tắc, một sự kiện được thông báo cho các phân hệ. Các phân hệ được thiết kế điều khiển sự kiện này sẽ tự quyết việc trả lời. Mô hình này hiệu quả với các phân hệ được phân bố trên các máy tính khác nhau trên mạng. Ưu điểm của nó là việc phát triển tương đối đơn giản. Một phân hệ mới xử lý một lớp sự kiện mới có thể được tích hợp khi ghi nhận các sự kiện này vào bộ điều khiển sự kiện. Mỗi phân hệ có thể kích hoạt mọi phân hệ khác không cần biết tên và vị trí của các phân hệ đó. Phân bố trong suốt với các phân hệ. Nhược điểm của mô hình này là phân hệ không biết sự kiện có được xử lý hay không và khi nào được xử lý. Rất có thể hai phân hệ khác nhau cùng sinh một sự kiện và có thể gây xung đột.
- Mô hình điều khiển ngắt: Có một hệ thống bên ngoài được sử dụng riêng cho việc theo dõi các ngắt bên ngoài và được chuyển tới các phân hệ tương ứng. Mô hình này phù hợp với các hệ thống thời gian. Ưu điểm của nó là cho phép đáp ứng nhanh nhất với các sự kiện. Nhược điểm là việc lập trình phức tạp.

4.6.2.5. Mô hình đối tượng

Hệ thống được phân thành các đối tượng giao tiếp với nhau. Phân tích hướng đối tượng chỉ ra các lớp đối tượng liên quan, các thuộc tính và các hoạt động của chúng. Ưu điểm của nó là tính bao đóng cho phép dấu các thực hiện của các đối tượng và cho phép dùng lại mã. Tuy nhiên nhược điểm là để sử dụng các dịch vụ, các đối tượng phải gọi tường minh các tên và giao diện của các đối tượng khác. Sự thay đổi giao diện sẽ làm ảnh hưởng tới các đối tượng khác.

4.6.2.6. Mô hình luồng dữ liệu

Hệ thống được phân hoá thành các module chức năng. Chúng nhận các dữ liệu chuyển hoá chúng rồi lại đưa ra kết quả. Trong mô hình luồng dữ liệu, các bộ biến đổi xử lý dữ liệu đầu vào và tạo dữ liệu ra. Dữ liệu được chảy tuần tự theo luồng từ bộ biến đổi này sang bộ khác. Mỗi bước của quy trình giống như một phép biến đổi.

Mô hình này có ưu điểm:

- Nó hỗ trợ việc sử dụng lại các biến đổi.
- Nó phù hợp với suy nghĩ của mọi người quan niệm về dữ liệu được xử lý theo luồng có đầu vào và đầu ra.
- Thêm các xử lý khác vào hệ thống đơn giản.
- Dễ thực hiện xử lý song song hoặc tuần tự.

Nhược điểm của mô hình này là:

- Cần phải có một định dạng chung cho các dữ liệu để có thể xử lý bởi mọi bộ biến đổi.
- Các hệ thống tương tác khó được viết theo mô hình luồng dữ liệu. Trong khi các giao diện vào và ra theo dạng văn bản có thể dùng mô hình luồng dữ liệu thì các giao diện đồ hoạ có các dạng vào ra phức tạp hơn dựa trên sự kiện khó có thể áp dụng mô hình luồng dữ liệu.

4.7. THIẾT KẾ GIAO DIỆN NGƯỜI SỬ DỤNG

Khi các hệ thống tin học hoá ngày càng đi vào đời sống con người thì vấn đề thiết kế giao diện càng trở nên quan trọng trong việc phát triển phần mềm. Có nhiều câu hỏi được đặt ra trong quá trình thiết kế giao diện, như:

+ Ai là người dùng?

+ Người dùng học cách tương tác với hệ thống mới dựa trên máy tính như thế nào?

+ Người dùng diễn giải thông tin do hệ thống tạo ra như thế nào?

+ Người dùng trông đợi gì ở hệ thống?...

Để có được hệ thống thân thiện với người sử dụng, có nhiều nhân tố cần được quan tâm trong vấn đề thiết kế giao diện.

4.7.1. Nhân tố con người

Nhân tố con người: được dựa vào các yếu tố

- Nền tảng về cảm nhận của con người,
- Mức độ kỹ năng và hành vi con người,
- Nhiệm vụ và nhân tố con người: được phân loại dựa vào mục đích của ứng dụng, cụ thể:
  - Nhiệm vụ trao đổi: các hoạt động làm cho thông tin được truyền từ nơi sản xuất đến nơi tiêu thụ.
  - Nhiệm vụ đối thoại: các hoạt động làm cho người sử dụng định hướng và điều khiển tương tác với hệ thống dựa trên máy tính.
  - Nhiệm vụ nhận biết: các hoạt động được thực hiện một khi đã thu được thông tin, các hoạt động liên kết với chức năng của hệ thống.
  - Nhiệm vụ điều khiển: các hoạt động cho phép người sử dụng kiểm soát thông tin, nhận biết và ra lệnh cho tiến trình thông qua đó các nhiệm vụ tổng quát khác xuất hiện.

4.7.2. Phong cách tương tác người - máy

Phong cách tương tác người - máyquan hệ chặt chẽ với lịch sử tiến hóa của máy tính, có một số loại giao diện như:

- Giao diện chỉ lệnh vào hỏi,
- Giao diện đơn (menu đơn giản),
- Giao diện trỏ và nhặt (hướng cửa sổ),...

Hiện nay, giao diện thông dụng nhất cho người sử dụng là giao diện hướng cửa sổ (X-Window).

4.7.3. Thiết kế giao diện người - máy

Thiết kế giao diện người - máy là một nhiệm vụ quan trọng trong thiết kế phần mềm. Hướng dẫn thiết kế giao diện người - máy gồm các vấn đề liên quan:

1. *Mô hình thiết kế giao diện được thể hiện qua bốn mô hình:*
   + Mô hình thiết kế,
   + Mô hình người sử dụng: được phân loại cho người mới học, người ít hiểu biết và người hiểu biết.
   + Mô hình của người dùng hay cảm nhận hệ thống: tức là hình ảnh của hệ thống mà người sử dụng mang trong đầu.
   + Hình ảnh hệ thống: cách biểu lộ bên ngoài của hệ thống dựa trên máy tính với mọi thông tin hỗ trợ.

2. *Phân tích và mô hình hóa nhiệm vụ: gồm các bước*
   - Thiết lập các mục tiêu và ý đồ cho nhiệm vụ
   - Ánh xạ từng mục tiêu, ý đồ thành dãy các hành động xác định
   - Xác định dãy các hành động khi nó sẽ được thực hiện tại mức giao diện
   - Chỉ ra trạng thái của hệ thống, tức là giao diện giống thế nào vào lúc hành động trong dãy đó được thực hiện
   - Xác định các cơ chế điều khiển, như thiết bị và hành động sẵn có cho người dùng để thay đổi trạng thái hệ thống.
   - Chỉ ra cách thức cơ chế điều khiển này ảnh hưởng đến trạng thái hệ thống.
   - Chỉ ra cách thức người dùng diễn giải trạng thái của hệ thống từ thông tin được cung cấp qua giao diện.

3. *Vấn đề thiết kế: có bốn vấn đề*
   - Thời gian hệ thống đáp ứng,
   - Tiện nghi giúp đỡ người dùng,
   - Giải quyết thông tin lỗi,
   - Gán nhãn chỉ lệnh.

a. Thời gian hệ thống đáp ứng: chứa hai đặc trưng quan trọng là độ dài và độ biến thiên. Cần chú ý nếu thời gian đáp ứng quá lâu thì người sử dụng nhàm chán còn nếu sự đáp ứng quá nhanh thì người sử dụng dễ mắc sai lầm do vội vã.

b. Tiện nghi giúp đỡ: phải trả lời các câu hỏi
   + Trợ giúp có sẵn với mọi chức năng vào mọi lúc không?
   + Người sử dụng sẽ yêu cầu trợ giúp thế nào?
   + Trợ giúp sẽ được trình bày như thế nào?
   + Người sử dụng sẽ trở về tương tác thông thường thế nào?
   + Thông tin trợ giúp được cấu trúc thế nào?

c. Giải quyết thông báo lỗi: thể hiện ở
   + Thông báo nên mô tả vấn đề mà người dùng có thể hiểu được

+ Thông báo nên đưa ra những lời khuyên có tính xây dựng để khôi phục từ lỗi

+ Thông báo nên chỉ ra bất kỳ hậu quả lỗi tiêu cực nào để người dùng có thể kiểm tra. Ví dụ: không có file dữ liệu.

+ Thông báo nên đi kèm tín hiệu nghe, thấy được

+ Không đưa ra các thông báo hàm ý trách móc người dùng

d. Gán nhãn chỉ lệnh:

+ Mọi tùy chọn đơn có tương ứng với một chỉ thị không?

+ Các lệnh sẽ có dạng nào; các từ gõ vào

+ Việc học và nhớ lệnh khó đến đâu? Có thể làm gì khi quên mất chỉ lệnh.

+ Liệu chỉ lệnh có phù hợp với người dùng hay không?

4. *Công cụ cài đặt:* các công cụ sẵn có để thiết kế giao diện.

5. *Tiến hóa thiết kế*

Khi có một mô hình thiết kế giao diện được tạo ra thì áp dụng các tiêu chuẩn đánh giá để thiết kế giao diện. Cuộc xét duyệt gồm các vấn đề:

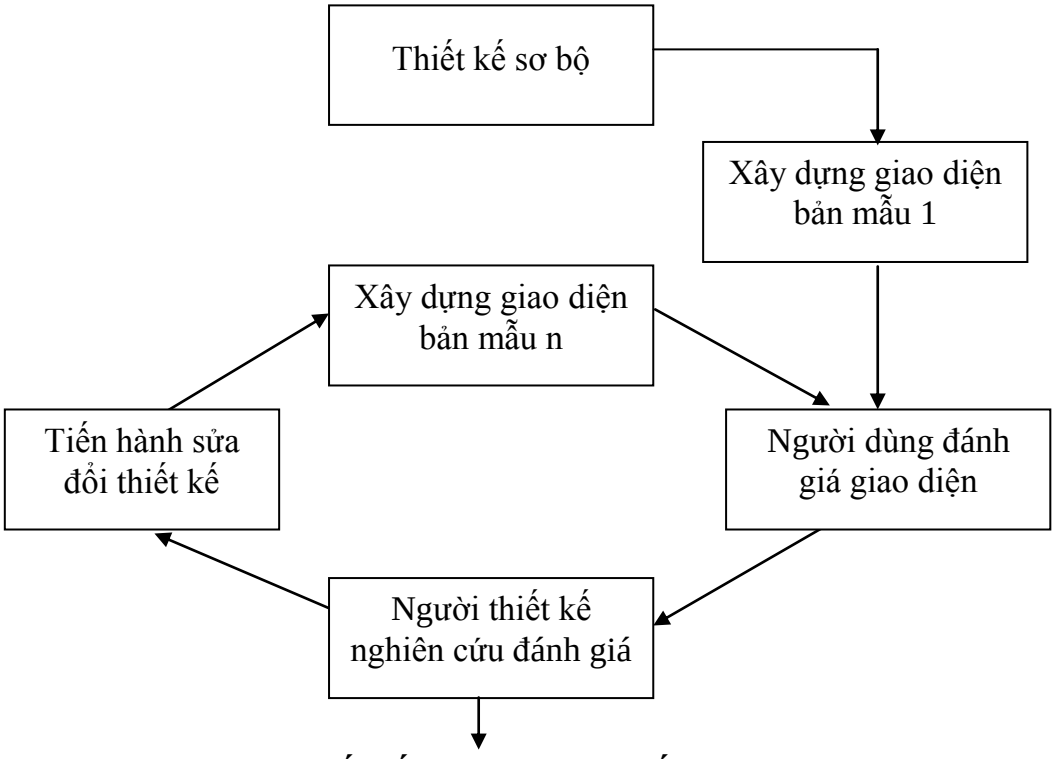
+ Độ dài và độ phức tạp của đặc tả viết về hệ thống và giao diện của nó cung cấp một chỉ dẫn về khối lượng học tập người dùng hệ thống cần học.

+ Số các chỉ lệnh được xác định và số trung bình các đối số trên chỉ lệnh đưa ra một chỉ dẫn về thời gian tương tác và hiệu quả tổng thể của hệ thống.

+ Số các hành động, chỉ lệnh và trạng thái hệ thống được mô hình thiết kế nêu ra cũng chỉ ra khối lượng cần nhớ trên người dùng hệ thống.

+ Phong cách tương tác, tiện nghi giúp đỡ và giao thức xử lý lỗi đưa ra một chỉ dẫn chung về độ phức tạp của giao diện và mức độ người dùng sẽ chấp nhận nó.

Quy trình đánh giá thiết kế giao diện được mô tả như sau:



4.7.4. Hướng dẫn thiết kế giao diện

a. *Hướng dẫn về tương tác chung:* giao diện phải
   - Nhất quán,
   - Cho thông tin phản hồi có nghĩa,
   - Yêu cầu kiểm chứng mọi hành động phá hủy không tầm thường,
   - Cho phép dễ dàng lần ngược nhiều hành động ,
   - Tìm kiếm tính hiệu quả trong đối thoại, vận động và ý nghĩa,
   - Dung thứ cho sai lầm,
   - Phân loại các hoạt động theo chức năng và tổ chức màn hình hài hòa theo vùng,
   - Cung cấp tiện nghi trợ giúp làm ngữ cảnh,
   - Dùng các động từ đơn giản hay cụm từ ngắn để đặt tên chỉ lệnh,...

b. *Hướng dẫn về hiển thị thông tin*
   - Chỉ hiển thị thông tin có liên quan đến ngữ cảnh hiện tại,
   - Đừng chôn vùi người dùng dưới dữ liệu, hãy dùng định dạng trình bày cho phép hấp thu nhanh chóng thông tin,
   - Dùng nhãn nhất quán, cách viết tắt chuẩn và màu sắc dự kiến trước được,
   - Cho phép người dùng duy trì ngữ cảnh trực quan,
   - Đưa ra các thông báo lỗi có nghĩa,
   - Dùng chữ hoa, chữ thường, thụt cấp và gộp nhóm văn bản để trợ giúp cho việc hiểu,
   - Sử dụng cửa sổ để đóng khung các kiểu thông tin khác nhau,
   - Dùng cách hiển thị "tương tự" để biểu diễn thông tin dễ được hấp thụ hơn với dạng biểu diễn này,
   - Xem xét vùng hiển thị có sẵn trên màn hình và dùng nó một cách có hiệu quả,...

c. *Hướng dẫn về vào dữ liệu*
   - Tối thiểu số hành động đưa vào mà người sử dụng thực hiện,
   - Duy trì sự nhất quán giữa hiển thị thông tin và cái vào dữ liệu,
   - Cho phép người dùng làm phù hợp cái vào,
   - Tương tác nên mềm dẻo và hài hòa với mode đưa vào ưa thích,
   - Khử kích hợp các lệnh không phù hợp hiện tại,
   - Để cho người dùng kiểm soát luồng tương tác,
   - Cung cấp trợ giúp cho mọi hành động đưa vào,...

Giao diện người-máy là cánh cửa đi vào phần mềm ứng dụng. Để giao diện đáp ứng được yêu cầu thì người thiết kế nó cần phải hiểu biết về nhân tố con người và công nghệ giao diện. Thêm vào đó, kiểu cách giao diện, công nghệ phần cứng và phần mềm sẵn có, và bản thân ứng dụng cũng có ảnh hưởng đến kết quả cuối cùng.

Câu hỏi

1. Tại sao phải có thiết kế ứng dụng?
2. Nêu các hoạt động của quá trình thiết kế phần mềm.
3. Thế nào là một thiết kế phần mềm tốt.
4. Với tài liệu yêu cầu đã có ở chương 3, bạn hãy xây dựng tài liệu thiết kế cho ứng dụng. Bạn đã lựa chọ mô hình thiết kế ứng dụng nào, vì sao?
5. Vì sao phải thiết kế giao diện ứng dụng? Chỉ rõ các vấn đề quan trọng khi thiết kế giao diện ứng dụng.