Bài giảng
KỸ NGHỆ PHẦN MỀM

Nguyễn Việt Hà
Bộ môn Công nghệ phần mềm

MỤC LỤC

DANH MỤC HÌNH

DANH MỤC BẢNG BIỂU

Chương 1

Phần mềm và kỹ nghệ phần mềm

1.1 Tầm quan trọng và sự tiến hóa của phần mềm

Máy tính khác với các máy móc thông thường ở điểm nó có thể thực hiện các nhiệm vụ rất khác nhau bằng cách sử dụng các phần mềm khác nhau. Tức là phần mềm tạo ra sự khác biệt giữa các máy tính và cũng quyết định năng lực của máy tính.

Cho đến những năm 1990, xu hướng của ngành công nghiệp máy tính là phát triển phần cứng nhằm giảm giá thành hệ thống và tăng năng lực xử lý cũng như lưu trữ dữ liệu. Do nhu cầu phần mềm tăng lên nhanh chóng, thách thức hay mục tiêu của ngành công nghiệp máy tính hiện nay là sự cải thiện chất lượng và giảm giá thành của phần mềm.

Có thể nói *khả năng của phần cứng biểu thị cho tiềm năng của hệ thống còn phần mềm là một cơ chế giúp chúng ta khai thác tiềm năng này.*

Chúng ta hãy xem xét tầm quan trọng của phần mềm trên khía cạnh sự tiến hóa và phạm vi ứng dụng của chúng.

1.1.1 Tiến hóa của phần mềm

Sự tiến hóa của phần mềm gắn liền với sự tiến hóa của phần cứng và có thể chia làm 4 giai đoạn:

a. Những năm đầu (từ 1950 đến 1960):

- Giai đoạn này phần cứng thay đổi liên tục, số lượng máy tính rất ít và phần lớn mỗi máy đều được đặt hàng chuyên dụng cho một ứng dụng đặc biệt.

- Phương thức chính là xử lý theo lô (batch), tức là "gói" các chương trình có sử dụng kết quả của nhau lại thành một khối để tăng tốc độ thực hiện.

- Thời kỳ này lập trình máy tính được coi là nghệ thuật "theo bản năng", chưa có phương pháp hệ thống. Việc phát triển phần mềm chưa được quản lý.

- Môi trường lập trình có tính chất cá nhân; thiết kế, tiến trình phần mềm không tường minh, thường không có tài liệu. Sản xuất có tính đơn chiếc, theo đơn đặt hàng. Người lập trình thường là người sử dụng và kiêm cả việc bảo trì và sửa lỗi.

b. Thời kỳ trải rộng từ những năm 1960 đến giữa những năm 1970:

- Các hệ thống đa nhiệm, đa người sử dụng (ví dụ: Multics, Unix,...) xuất hiện dẫn đến khái niệm mới về tương tác người máy. Kỹ thuật này mở ra thế giới mới cho các ứng dụng và đòi hỏi mức độ tinh vi hơn cho cả phần mềm và phần cứng.

- Nhiều hệ thống thời gian thực với các đặc trưng thu thập, phân tích và biến đổi dữ liệu từ nhiều nguồn khác nhau và phản ứng (xử lý, tạo output) trong một khoảng thời gian nhất định xuất hiện.

- Tiến bộ lưu trữ trực tuyến làm xuất hiện thế hệ đầu tiên của hệ quản trị CSDL.

- Số lượng các hệ thống dựa trên máy tính phát triển, nhu cầu phân phối mở rộng, thư viện phần mềm phát triển, quy mô phần mềm ngày càng lớn làm nẩy sinh nhu cầu sửa chữa khi gặp lỗi, cần sửa đổi khi người dùng có yêu cầu hay phải thích nghi với những thay đổi của môi trường phần mềm (phần cứng, hệ điều hành, chương trình dịch mới). Công việc bảo trì phần mềm dần dần tiêu tốn nhiều công sức và tài nguyên đến mức báo động.

c. Thời kỳ từ giữa những năm 1970 đến đầu những năm 1990:

- Hệ thống phân tán (bao gồm nhiều máy tính, mỗi máy thực hiện một chức năng và liên lạc với các máy khác) xuất hiện làm tăng quy mô và độ phức tạp của phần mềm ứng dụng trên chúng.

- Mạng toàn cục và cục bộ, liên lạc số giải thông cao phát triển mạnh làm tăng nhu cầu thâm nhập dữ liệu trực tuyến, nảy sinh yêu cầu lớn phát triển phần mềm quản lý dữ liệu.

- Công nghệ chế tạo các bộ vi xử lý tiến bộ nhanh khiến cho máy tính cá nhân, máy trạm để bàn, và các thiết bị nhúng (dùng cho điều khiển trong robot, ô tô, thiết bị y tế, đồ điện gia dụng,...) phát triển mạnh khiến cho nhu cầu về phần mềm tăng nhanh.

- Thị trường phần cứng đi vào ổn định, chi phí cho phần mềm tăng nhanh và có khuynh hướng vượt chi phí mua phần cứng.

d. Thời kỳ sau 1990:

- Kỹ nghệ hướng đối tượng là cách tiếp cận mới đang nhanh chóng thay thế nhiều cách tiếp cận phát triển phần mềm truyền thống trong các lĩnh vực ứng dụng.

- Sự phát triển của Internet làm cho người dùng máy tính tăng lên nhanh chóng, nhu cầu phần mềm ngày càng lớn, quy mô và độ phức tạp của những hệ thống phần mềm mới cũng tăng đáng kể.

- Phần mềm trí tuệ nhân tạo ứng dụng các thuật toán phi số như hệ chuyên gia, mạng nơ ron nhân tạo được chuyển từ phòng thí nghiệm ra ứng dụng thực tế mở ra khả năng xử lý thông tin và nhận dạng kiểu con người.

1.1.2 Sự ứng dụng của phần mềm

Chúng ta có thể chia phần mềm theo miền ứng dụng thành 7 loại như sau:

a. Phần mềm hệ thống

- Là một tập hợp các chương trình được viết để phục vụ cho các chương trình khác

- Xử lý các cấu trúc thông tin phức tạp nhưng xác định (trình biên dịch, trình soạn thảo, tiện ích quản lý tệp)

- Đặc trưng bởi tương tác chủ yếu với phần cứng máy tính

- Phục vụ nhiều người dùng

- Cấu trúc dữ liệu phức tạp và nhiều giao diện ngoài

b. Phần mềm thời gian thực

Phần mềm điều phối, phân tích hoặc kiểm soát các sự kiện thế giới thực ngay khi chúng xuất hiện được gọi là phần mềm thời gian thực. Điển hình là các phần mềm điều khiển các thiết bị tự động. Phần mềm thời gian thực bao gồm các thành tố:

- Thành phần thu thập dữ liệu để thu và định dạng thông tin từ môi trường ngoài

- Thành phần phân tích để biến đổi thông tin theo yêu cầu của ứng dụng

- Thành phần kiểm soát hoặc đưa ra đáp ứng môi trường ngoài

- Thành phần điều phối để điều hòa các thành phần khác sao cho có thể duy trì việc đáp ứng thời gian thực

Hệ thống thời gian thực phải đáp ứng những ràng buộc thời gian chặt chẽ.

c. Phần mềm nghiệp vụ

Là các phần mềm phục vụ các hoạt động kinh doanh hay các nghiệp vụ của tổ chức, doanh nghiệp. Đây có thể coi là lĩnh vực ứng dụng phần mềm lớn nhất. Điển hình là các hệ thống thông tin quản lý gắn chặt với CSDL, các ứng dụng tương tác như xử lý giao tác cho các điểm bán hàng.

d. Phần mềm khoa học và công nghệ

- Được đặc trưng bởi các thuật toán (tính toán trên ma trận số, mô phỏng...).

- Thường đòi hỏi phần cứng có năng lực tính toán cao.

e. Phần mềm nhúng

- Nằm trong bộ nhớ chỉ đọc và được dùng để điều khiển các sản phẩm và hệ thống cho người dùng và thị trường công nghiệp.

- Có các đặc trưng của phần mềm thời gian thực và phần mềm hệ thống.

f. Phần mềm máy tính cá nhân

- Bùng nổ từ khi xuất hiện máy tính cá nhân, giải quyết các bài toán nghiệp vụ nhỏ như xử lý văn bản, trang tính, đồ họa, quản trị CSDL nhỏ...

- Yếu tố giao diện người-máy rất được chú trọng.

g. Phần mềm trí tuệ nhân tạo

- Dùng các thuật toán phi số để giải quyết các vấn đề phức tạp mà tính toán hay phân tích trực tiếp không quản lý nổi

- Các ứng dụng chính là: hệ chuyên gia (hệ cơ sở tri thức), nhận dạng (hình ảnh và tiếng nói), chứng minh định lý và chơi trò chơi, mô phỏng.

Ngoài ra, chúng ta còn có thể kể đến một dạng phần mềm đặc biệt là phần mềm phục vụ kỹ nghệ phần mềm. Đó là các phần mềm như chương trình dịch, phần mềm gỡ rối, các công cụ hỗ trợ phân tích thiết kế (CASE)... Các phần mềm này có thể xuất hiện dưới dạng phần mềm máy tính cá nhân, phần mềm hệ thống hoặc là phần mềm

nghiệp vụ.

1.2 Khó khăn, thách thức đối với phát triển phần mềm

Từ những năm 60, nhiều dự án phần mềm lớn không thành công như các dự án OS 360 (tiêu tốn một số tiền và thời gian gấp nhiều lần dự kiến) và TSS 360 (không đạt các chỉ tiêu kỹ thuật, hầu như không hoạt động) của IBM. Do đó, việc phát triển phần mềm dần dần đã được nhận thức là một lĩnh vực đầy khó khăn và chứa nhiều rủi ro. Chúng ta sẽ xem xét các khó khăn và thách thức trên các khía cạnh đặc trưng, qui mô và nhu cầu của phần mềm.

1.2.1 Phần mềm và phần mềm tốt

Phần mềm thông thường được định nghĩa bao gồm:

- các lệnh máy tính nhằm thực hiện các chức năng xác định
- các cấu trúc dữ liệu cho phép chương trình thao tác với dữ liệu
- các tài liệu giúp cho người dùng có thể vận hành được phần mềm

Bốn thuộc tính chủ chốt mà một hệ phần mềm tốt phải có là:

- *Có thể bảo trì được:* phần mềm tuổi thọ dài phải được viết và được lập tư liệu sao cho việc thay đổi có thể tiến hành được mà không quá tốn kém. Đây được coi là đặc tính chủ chốt nhất của một phần mềm tốt. Để có thể bảo trì được, phần mềm phải có một thiết kế tốt có tính modun hóa cao, được viết bằng ngôn ngữ bậc cao và được lập tài liệu (tài liệu phân tích, thiết kế, chú thích mã nguồn, hướng dẫn người dùng...) đầy đủ.

- *Đáng tin cậy:* phần mềm phải thực hiện được điều mà người tiêu dùng mong mỏi và không thất bại nhiều hơn những điều đã được đặc tả. Điều này có nghĩa là phần mềm phải thỏa mãn được nhu cầu của người dùng. Để đạt được yếu tố đáng tin cậy, trước tiên người phát triển cần phải hiểu một cách đúng đắn yêu cầu của người dùng và sau đó cần thỏa mãn được các yêu cầu này bằng các thiết kế và cài đặt tốt.

- *Có hiệu quả:* phần mềm khi hoạt động phải không lãng phí tài nguyên hệ thống như bộ nhớ, bộ xử lý. Nếu phần mềm chạy quá chậm hay đòi hỏi quá nhiều bộ nhớ... thì dù có được cài đặt rất nhiều chức năng cũng sẽ không được đưa vào sử dụng. Tuy nhiên, ngoại trừ các phần mềm nhúng hay thời gian thực đặc biệt, người ta thường không cực đại hóa mức độ hiệu quả vì rằng việc đó có thể phải dùng đến các kỹ thuật đặc thù và cài đặt bằng ngôn ngữ máy khiến cho chi phí tăng cao và phần mềm rất khó thay đổi (tính bảo trì kém).

- *Dễ sử dụng:* giao diện người sử dụng phải phù hợp với khả năng và kiến thức của người dùng, có các tài liệu hướng dẫn và các tiện ích trợ giúp. Đối tượng chính

của các phần mềm nghiệp vụ thường là người không am hiểu về máy tính, họ sẽ xa lánh các phần mềm khó học, khó sử dụng.

Có thể thấy rõ, việc tối ưu hóa đồng thời các thuộc tính này là rất khó khăn. Các thuộc tính có thể mẫu thuẫn lẫn nhau, ví dụ như tính hiệu quả và tính dễ sử dụng, tính bảo trì. Quan hệ giữa chi phí cải tiến và hiệu quả đối với từng thuộc tính không phải là tuyến tính. Nhiều khi một cải thiện nhỏ trong bất kỳ thuộc tính nào cũng có thể là rất đắt.

Một khó khăn khác của việc phát triển phần mềm là rất khó định lượng các thuộc tính của phần mềm. Chúng ta thiếu các độ đo và các chuẩn về chất lượng phần mềm.

Vấn đề giá cả phải được tính đến khi xây dựng một phần mềm. Chúng ta sẽ xây dựng được một phần mềm dù phức tạp đến đâu nếu không hạn chế về thời gian và chi phí. Điều quan trọng là chúng ta phải *xây dựng một phần mềm tốt với một giá cả hợp lý và theo một lịch biểu được định trước.*

1.2.2 Đặc trưng phát triển và vận hành phần mềm

Chúng ta có thể thấy khó khăn hàng đầu của việc phát triển phần mềm là do tính chất phần mềm là hệ thống logic, không phải là hệ thống vật lý. Do đó nó có đặc trưng khác biệt đáng kể với các đặc trưng của phần cứng.

Dưới đây là 3 yếu tố chính tạo ra sự phức tạp trong quá trình phát triển cũng như sử dụng, bảo trì phần mềm.

a. Phần mềm không được chế tạo theo nghĩa cổ điển

Phần mềm cũng được được thiết kế, phát triển như phần cứng, nhưng nó không định hình trước. Chỉ khi phát triển xong người ta có sản phẩm cụ thể và hiểu được nó có hiệu quả hay không. Tức là ở các bước trung gian, chúng ta rất khó kiểm soát chất lượng của phần mềm.

Giá thành của phần cứng chủ yếu bị chi phối bởi giá thành nguyên vật liệu và chúng ta tương đối dễ kiểm soát. Trong khi đó, giá thành phần mềm chủ yếu tập chung vào chi phí nhân công. Quá trình phát triển phần mềm phụ thuộc vào con người (hiểu biết, khả năng vận dụng, kinh nghiệm và cách thức quản lý) và được tiến hành phát triển trong điều kiện môi trường (kỹ thuật, xã hội) đa dạng và không ngừng thay đổi. Do đó chúng ta rất khó ước lượng được chi phí cũng như hiệu quả của phần mềm.

b. Phần mềm không hỏng đi nhưng thoái hóa theo thời gian

Phần mềm không cảm ứng đối với những tác động của môi trường vốn gây cho phần cứng bị mòn cũ đi, nhưng nó cũng thoái hóa theo thời gian.

Thực tế, phần mềm trải qua thời gian sử dụng cần phải được thay đổi (bảo trì) để đáp ứng nhu cầu luôn thay đổi của tổ chức sử dụng nó. Mỗi khi thay đổi, sẽ xuất hiện thêm một số khiếm khuyết mới không thể tránh làm cho số lỗi tiềm ẩn trong phần mềm tăng lên. Dần dần, phần mềm bị thoái hóa do tỷ lệ sai hỏng ngày càng tăng lên đến mức gây ra những thiệt hại không thể chấp nhận được.

Việc bảo trì phần mềm phức tạp hơn nhiều và có bản chất khác hẳn so với bảo trì phần cứng do sự phức tạp của hệ thống phần mềm và sự không có sẵn phần thay thế

cho bộ phận bị lỗi. Chúng ta không thay thế bộ phận bị lỗi bằng cái có sẵn mà thực tế phải tạo ra một môđun mới. Do đó, thông thường chỉ có nhà sản xuất phần mềm mới bảo trì (sửa chữa) được hỏng hóc. Sẽ rất khó ước lượng được chi phí cho bảo trì phần mềm.

c. Phần lớn phần mềm đều được xây dựng từ đầu, ít khi được lắp ráp từ thành phần có sẵn

- Phần mềm không có danh mục các thành phần cố định như phần cứng.
- Phần mềm thường được đặt hàng theo một đơn vị hoàn chỉnh, theo yêu cầu riêng của khách hàng.
- Phần mềm ít khi có thể lắp ráp theo một khuôn mẫu có sẵn. Yêu cầu với phần mềm thay đổi theo môi trường cụ thể mà ở đó nó được xây dựng. Môi trường của phần mềm (gồm phần cứng, phần mềm nền, con người và tổ chức) không thể định dạng từ trước và lại thay đổi thường xuyên.

Những yếu tố này dẫn đến chi phí cho phần mềm cao và rất khó đảm bảo được lịch biểu cho phát triển phần mềm.

1.2.3 Nhu cầu và độ phức tạp

Tuy ngành công nghiệp máy tính đã bước sang giai đoạn phát triển thứ tư nhưng các thách thức đối với phát triển phần mềm máy tính không ngừng gia tăng vì những nguyên nhân sau:

- Khả năng xây dựng các chương trình mới không giữ được cùng nhịp với nhu cầu về phần mềm tăng lên nhanh chóng, đặc biệt khi Internet phát triển và số lượng người dùng tăng cao. Ngày nay, sản xuất phần mềm đã trở thành một ngành công nghiệp không lồ tuy vậy năng suất không cao, không đáp ứng được đòi hỏi của xã hội và điều này ảnh hưởng lớn đến giá thành và chất lượng phần mềm. Ngoài ra, còn tồn tại rất nhiều chương trình được thiết kế và lập tài liệu sơ sài khiến cho việc bảo trì rất khó khăn và kém tài nguyên. Phát triển các phần mềm mới dễ bảo trì để thay thế các hệ thống cũ trở thành nhu cầu cấp bách.

- Cùng với sự phát triển của phần cứng, quy mô và độ phức tạp của các phần mềm mới ngày càng tăng. Một số phần mềm hiện đại có kích thước được tính bằng đơn vị triệu dòng lệnh (HĐH Unix, Windows...). Một vấn đề khó khăn trong sản xuất phần mềm lớn là độ phức tạp tăng vọt, các kinh nghiệm sản xuất sản phẩm nhỏ không ứng dụng được cho môi trường làm việc theo nhóm và phát triển sản phẩm lớn.

- Sự tinh vi và năng lực của phần cứng đã vượt xa khả năng xây dựng phần mềm để có thể sử dụng được các tiềm năng của nó.

Tất cả các khó khăn và thách thức nêu trên đã dẫn đến việc chấp nhận thực hành kỹ nghệ phần mềm để có thể tạo nhanh các phần mềm có nhất lượng ngày một cao, có quy mô và số lượng ngày một lớn và có những tính năng tương ứng với tiềm năng phần cứng.

1.3 Kỹ nghệ phần mềm

1.3.1 Định nghĩa

Một định nghĩa ban đầu về kỹ nghệ phần mềm do Fritz Bauer nêu ra là:

Việc thiết lập và sử dụng các nguyên lý công nghệ đúng đắn để thu được phần mềm một cách kinh tế vừa tin cậy vừa làm việc hiệu quả trên các máy thực.

Kỹ nghệ phần mềm là một quá trình gồm một loạt các bước chứa đựng 3 yếu tố chủ chốt:

- Phương pháp
- Công cụ
- Thủ tục

Các yếu tố này giúp người quản lý kiểm soát được tiến trình phát triển phần mềm, cung cấp cho người kỹ sư phần mềm một nền tảng để xây dựng phần mềm chất lượng cao theo một cách thức hiệu quả, trong những giới hạn nhất định.

a. Các phương pháp

Chỉ ra cách làm về mặt kỹ thuật để xây dựng phần mềm, được sử dụng trong các bước: lập kế hoạch, ước lượng dự án, phân tích yêu cầu hệ thống và phần mềm, thiết kế cấu trúc dữ liệu, kiến trúc chương trình và thủ tục thuật toán, mã hóa kiểm thử và bảo trì.

Các phương pháp cho kỹ nghệ phần mềm thường đưa ra các ký pháp đồ họa hay hướng ngôn ngữ đặc biệt, cách thức thực hiện và một tập các tiêu chuẩn về chất lượng của sản phẩm phần mềm.

b. Các công cụ

Cung cấp sự hỗ trợ tự động hay bán tự động để phát triển phần mềm theo từng phương pháp khác nhau. Khi các công cụ được tích hợp đến mức các thông tin do chúng tạo ra có thể được dùng cho các công cụ khác thì hệ thống hỗ trợ phát triển phần mềm đã được thiết lập và còn được gọi là kỹ nghệ phần mềm có máy tính hỗ trợ (CASE - Computer Aided Software Engineering).

c. Các thủ tục

Các thủ tục là chất keo dán các phương pháp và công cụ lại với nhau làm cho chúng được sử dụng hợp lý và đúng hạn trong quá trình phát triển phần mềm. Thủ tục bao gồm:

- Xác định ra trình tự các phương pháp sẽ được áp dụng cho mỗi dự án.

- Tạo sản phẩm cần bàn giao (tài liệu báo cáo, bản mẫu,...) cần cho việc kiểm soát để đảm bảo chất lượng và điều hòa thay đổi.

- Xác định những cột mốc mà tại đó có các sản phẩm nhất định được bàn giao để cho người quản lý phần mềm nắm được tiến độ và kiểm soát được kết quả.

Sau đây, chúng ta sẽ xem xét một số cách tiếp cận (còn gọi là mô hình hay khuôn cảnh) cơ bản trong tiến trình phát triển phần mềm.

1.3.2 Mô hình vòng đời cổ điển

Dưới đây mô tả kỹ nghệ phần mềm được tiến hành theo mô hình vòng đời cổ điển, đôi khi còn được gọi là mô hình thác nước (hình 1.1). Mô hình này yêu cầu tiếp cận một cách hệ thống, tuần tự và chặt chẽ (xong bước này mới chuyển sang bước sau) đối với việc phát triển phần mềm, bắt đầu ở mức phân tích hệ thống và tiến dần xuống phân tích, thiết kế, mã hóa, kiểm thử và bảo trì:

a. Kỹ nghệ và phân tích hệ thống

Kỹ nghệ và phân tích hệ thống bao gồm việc thu thập yêu cầu ở mức hệ thống với một lượng nhỏ thiết kế và phân tích ở mức đỉnh. Mục đích của bước này là xác định khái quát về phạm vi, yêu cầu cũng như tính khả thi của phần mềm.

b. Phân tích yêu cầu phần mềm

- Phân tích yêu cầu được tập trung việc thu thập và phân tích các thông tin cần cho phần mềm, các chức năng cần phải thực hiện, hiệu năng cần có và các giao diện cho người sử dụng.

- Kết quả của phân tích là tư liệu về yêu cầu cho hệ thống và phần mềm (đặc tả yêu cầu) để khách hàng duyệt lại và dùng làm tài liệu cho người phát triển.

c. Thiết kế

- Là quá trình chuyển hóa các yêu cầu phần mềm thành các mô tả thiết kế

- Thiết kế gồm nhiều bước, thường tập trung vào 4 công việc chính: thiết kế kiến trúc phần mềm, thiết kế cấu trúc dữ liệu, thiết kế chi tiết các thủ tục, thiết kế giao diện và tương tác.

- Lập tư liệu thiết kế (là một phần của cấu hình phần mềm) để phê duyệt

d. Mã hóa

Biểu diễn thiết kế bằng một hay một số ngôn ngữ lập trình và dịch thành mã máy thực hiện được.

e. Kiểm thử

Tiến trình kiểm thử bao gồm việc

i) phát hiện và sửa lỗi phần logic bên trong chương trình hay còn gọi là lỗi lập trình,

ii) kiểm tra xem phần mềm có hoạt động như mong muốn không, tức là phát hiện và sửa lỗi về chức năng như thiếu hụt, sai sót về chức năng; và kiểm tra xem phần mềm có đảm bảo tính hiệu quả trong thực hiện hay không.

f. Bảo trì

Bao gồm các công việc sửa các lỗi phát sinh khi áp dụng chương trình hoặc thích ứng nó với thay đổi trong môi trường bên ngoài (hệ điều hành mới, thiết bị ngoại vi mới, yêu cầu người dùng) hoặc yêu cầu bổ sung chức năng hay nâng cao hiệu năng cần có.

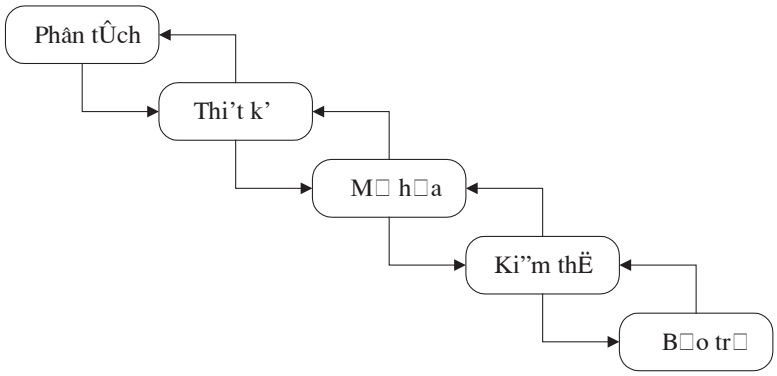
Một số các vấn đề có thể gặp phải khi dùng mô hình vòng đời cổ điển là:

1. Các dự án thực hiếm khi tuân theo dòng chảy tuần tự mà mô hình đề nghị. Bao giờ việc lặp lại cũng xuất hiện và tạo ra các vấn đề trong việc áp dụng mô hình này.

2. Khách hàng thường khó phát biểu mọi yêu cầu một cách tường minh từ đầu. Vòng đời cổ điển đòi hỏi điều này và thường khó thích hợp với sự bất trắc tự nhiên tồn tại vào lúc đầu của nhiều dự án.

3. Đòi hỏi khách hàng phải kiên nhẫn. Bản làm việc được của chương trình chỉ có được vào lúc cuối của thời gian dự án. Một sai sót nhỏ trong phân tích/thiết kế nếu đến khi có chương trình làm việc mới phát hiện ra, có thể sẽ là một thảm họa.

Tuy vậy, mô hình vòng đời cổ điển có một vị trí quan trọng trong công việc về kỹ nghệ phần mềm. Nó đưa ra một tiêu bản trong đó có thể bố trí các phương pháp cho phân tích, thiết kế, mã hóa, kiểm thử và bảo trì. Vòng đời cổ điển vẫn còn là một mô hình được sử dụng rộng rãi, nhất là đối với các dự án vừa và nhỏ.



Hình 1.1: Mô hình vòng đời cổ điển.

1.3.3 Mô hình làm bản mẫu

Cách tiếp cận làm bản mẫu cho kỹ nghệ phần mềm là cách tiếp cận tốt nhất khi:

- Mục tiêu tổng quát cho phần mềm đã xác định, nhưng chưa xác định được input và output.

- Người phát triển không chắc về hiệu quả của thuật toán, về thích nghi hệ điều hành hay giao diện người máy cần có.

Khi đã có bản mẫu, người phát triển có thể dùng chương trình đã có hay các công cụ phần mềm trợ giúp để sinh ra chương trình làm việc.

Làm bản mẫu là tạo ra một mô hình cho phần mềm cần xây dựng. Mô hình có thể có 3 dạng:

1. Bản mẫu trên giấy hay trên máy tính mô tả giao diện người-máy làm người dùng hiểu được cách các tương tác xuất hiện.

2. Bản mẫu cài đặt chỉ một tập con chức năng của phần mềm mong đợi.

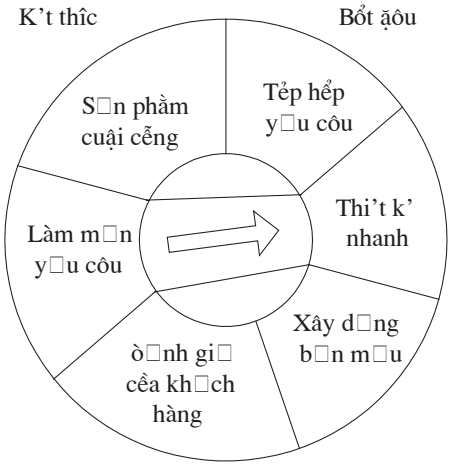
3. Bản mẫu là một chương trình có thể thực hiện một phần hay tất cả chức năng mong muốn nhưng ở mức sơ lược và cần cải tiến thêm các tính năng khác tùy theo khả năng phát triển.

Trước hết người phát triển và khách hàng gặp nhau và xác định mục tiêu tổng thể cho phần mềm, xác định các yêu cầu đã biết, các miền cần khảo sát thêm. Tiếp theo là giai đoạn thiết kế nhanh, tập trung vào việc biểu diễn các khía cạnh của phần mềm thấy được đối với người dùng (input và output), và xây dựng một bản mẫu. Người dùng đánh giá và làm mịn các yêu cầu cho phần mềm. Tiến trình này lặp đi lặp lại cho đến khi bản mẫu thoả mãn yêu cầu của khách hàng, đồng thời giúp người phát triển hiểu kỹ hơn nhu cầu nào cần phải thực hiện (hình 1.2).

Một biến thể của mô hình này là mô hình thăm dò, trong đó các yêu cầu được cập nhật liên tục và bản mẫu được tiến hóa liên tục để trở thành sản phẩm cuối cùng.

Mô hình làm bản mẫu có một số vấn đề như:

- Do sự hoàn thiện dần (tiến hóa) của bản mẫu, phần mềm nhiều khi có tính cấu trúc không cao, dẫn đến khó kiểm soát, khó bảo trì.
- Khách hàng nhiều khi thất vọng với việc phát triển phần mềm do họ nhầm tưởng bản mẫu là sản phẩm cuối cùng hướng tới người sử dụng. Khách hàng cũng có thể không dành nhiều công sức vào đánh giá bản mẫu.



Hình 1.2: Mô hình làm bản mẫu.

1.3.4 Mô hình xoắn ốc

Mô hình xoắn ốc được Boehm đưa ra năm 1988. Mô hình này đưa thêm vào việc phân tích yếu tố rủi ro. Quá trình phát triển được chia thành nhiều bước lặp lại, mỗi bước bắt đầu bằng việc phân tích rủi ro rồi tạo bản mẫu, cải tạo và phát triển bản mẫu, duyệt lại, và cứ thế tiếp tục (hình 1.3).

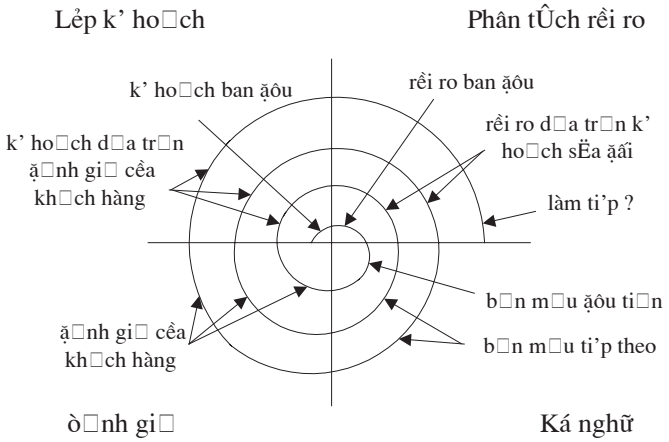
Nội dung một bước gồm bốn hoạt động chính:

- Lập kế hoạch: xác định mục tiêu, các giải pháp và ràng buộc
- Phân tích rủi ro: phân tích các phương án và xác định/giải quyết rủi ro
- Kỹ nghệ: phát triển sản phẩm "mức tiếp theo"
- Đánh giá: đánh giá của khách hàng về kết quả của kỹ nghệ

Với mỗi lần lặp xoắn ốc (bắt đầu từ tâm), các phiên bản được hoàn thiện dần. Nếu phân tích rủi ro chỉ ra rằng yêu cầu không chắc chắn thì bản mẫu có thể được sử dụng trong giai đoạn kỹ nghệ; các mô hình và các mô phỏng khác cũng được dùng để làm rõ hơn vấn đề và làm mịn yêu cầu.

Tại một vòng xoắn ốc, phân tích rủi ro phải đi đến quyết định "tiến hành tiếp hay dừng". Nếu rủi ro quá lớn thì có thể đình chỉ dự án.

Mô hình xoắn ốc cũng có một số vấn đề như khó thuyết phục những khách hàng lớn rằng cách tiếp cận tiến hóa là kiểm soát được. Nó đòi hỏi tri thức chuyên gia đánh giá rủi ro chính xác và dựa trên tri thức chuyên gia này mà đạt được thành công. Mô hình xoắn ốc đòi hỏi năng lực quản lý cao, nếu không quản lý tốt thì rất dễ rơi vào trạng thái sửa đổi cục bộ không có kế hoạch của mô hình làm bản mẫu (thăm dò). Và mô hình này còn tương đối mới và còn chưa được sử dụng rộng rãi như vòng đời hoặc làm bản mẫu. Cần phải có thêm một số năm nữa trước khi người ta có thể xác định được tính hiệu quả của mô hình này với sự chắc chắn hoàn toàn.



Hình 1.3: Mô hình xoắn ốc.

1.3.5 Kỹ thuật thế hệ thứ tư

Thuật ngữ kỹ thuật thế hệ thứ tư (4GT - fourth generation technology) bao gồm một phạm vi rộng các công cụ phần mềm có các điểm chung:

1. Cho phép người phát triển xác định một số đặc trưng của phần mềm ở mức cao.
2. Tự động sinh ra mã chương trình gốc theo nhu cầu của người phát triển.

Hiển nhiên là phần mềm được biểu diễn ở mức trừu tượng càng cao thì chương trình có thể được xây dựng càng nhanh hơn. Mô hình 4GT đối với kỹ nghệ phần mềm tập trung vào khả năng xác định phần mềm đối với một máy ở mức độ gần với ngôn ngữ tự nhiên hay dùng một ký pháp đem lại chức năng có ý nghĩa.

Hiện tại, một môi trường phát triển phần mềm hỗ trợ cho khuôn cảnh 4GT bao gồm một số hay tất cả các công cụ sau:

1. ngôn ngữ phi thủ tục để truy vấn CSDL

2. bộ sinh báo cáo

3. bộ thao tác dữ liệu

4. bộ tương tác và xác định màn hình

5. bộ sinh chương trình

6. khả năng đồ họa mức cao

7. khả năng làm trang tính

8. khả năng tạo tài liệu

Mỗi một trong những công cụ này đã tồn tại, nhưng chỉ cho vài lĩnh vực ứng dụng đặc thù. Ví dụ: các tính năng macro trong các phần mềm bảng tính, cơ sở dữ liệu, khả năng tự sinh mã trong các công cụ thiết kế giao diện "kéo - thả"...

Với những ứng dụng nhỏ, có thể chuyển trực tiếp từ bước thu thập yêu cầu sang cài đặt bằng công cụ 4GT. Tuy nhiên với những hệ thống lớn, cần phải có một chiến lược thiết kế. Việc dùng 4GT thiếu thiết kế (với các dự án lớn) sẽ gây ra những khó khăn như chất lượng kém, khó bảo trì khiến cho người dùng khó chấp nhận.

Vẫn còn nhiều tranh cãi xung quanh việc dùng khuôn cảnh 4GT:

- Người ủng hộ cho là 4GT làm giảm đáng kể thời gian phát triển phần mềm và làm tăng rất nhiều hiệu suất của người xây dựng phần mềm.

- Những người phản đối cho là các công cụ 4GT hiện tại không phải tất cả đều dễ dùng hơn các ngôn ngữ lập trình, rằng chương trình gốc do các công cụ này tạo ra là không hiệu quả, và rằng việc bảo trì các hệ thống phần mềm lớn được phát triển bằng cách dùng 4GT lại mở ra vấn đề mới.

Có thể tóm tắt hiện trạng của cách tiếp cận 4GT như sau:

1. Lĩnh vực ứng dụng hiện tại cho 4GT mới chỉ giới hạn vào các ứng dụng hệ thông tin nghiệp vụ, đặc biệt, việc phân tích thông tin và làm báo cáo là nhân tố chủ chốt cho các cơ sở dữ liệu lớn. Tuy nhiên, cũng đã xuất hiện các công cụ CASE mới hỗ trợ cho việc dùng 4GT để tự động sinh ra khung chương trình.

2. Đối với các ứng dụng vừa và nhỏ: thời gian cần cho việc tạo ra phần mềm được giảm đáng kể và khối lượng phân tích/thiết kế cũng được rút bớt.

3. Đối với ứng dụng lớn: các hoạt động phân tích, thiết kế và kiểm thử chiếm phần lớn thời gian và việc loại bỏ bớt lập trình bằng cách dùng 4GT nhiều khi đem lại hiệu quả không đáng kể so với tính rườm rà, kém hiệu quả của phần mềm xây dựng bằng phương pháp này.

Tóm lại, 4GT đã trở thành một phần quan trọng của việc phát triển phần mềm nghiệp vụ và rất có thể sẽ được sử dụng rộng rãi trong các miền ứng dụng khác trong thời gian tới.

1.3.6 Mô hình lập trình cực đoan

Lập trình cực đoan (XP - eXtreme Programming) do Kent Beck đề xuất là một phương pháp tiếp cận mới cho phát triển phần mềm. XP đưa ra nhiều hướng dẫn mới, đôi khi trái ngược lại với các cách thức phát triển phần mềm được đề xuất từ trước đến nay.

Hai khái niệm độc đáo mới và quan trọng hàng đầu trong XP là "tạo các ca thử nghiệm trước tiên" và "lập trình đôi".

a) Tạo các ca thử nghiệm trước tiên

Thông thường, thử nghiệm (và trước đó là tạo ca thử nghiệm) được tiến hành vào giai đoạn cuối của quá trình phát triển, khi bạn đã có mã nguồn và chuyển sang kiểm chứng tính đúng đắn của nó. Nhiều trường hợp việc kiểm thử không được coi trọng và chỉ được tiến hành khi bạn còn thời gian và kinh phí.

XP thay đổi quan niệm này bằng cách đặt cho kiểm thử một tầm quan trọng ngang bằng (có thể là lớn hơn) việc viết mã. Các ca kiểm thử được thiết kế trước khi viết mã và phải được thực hiện thành công mỗi khi chương trình đích được tạo ra.

Tạo ca thử nghiệm trước đem lại nhiều lợi thế. Thứ nhất, nó giúp bạn xác định một cách rõ ràng giao diện của modun. Hơn thế, để tạo được ca thử nghiệm, bạn cần phải hiểu rõ chức năng của nó. Tức là, XP yêu cầu bạn phải hiểu một cách rõ ràng các yêu cầu của modun trước khi bạn bắt tay vào phát triển nó.

b) Lập trình đôi

XP đưa ra khái niệm mang tính cách mạng (và trái ngược lại quan niệm từ trước đến nay) là mã nguồn của một môđun phải được viết bởi 2 lập trình viên dùng chung một máy tính.

Giá trị của lập trình đôi là trong khi một người viết mã thì người thứ hai nghĩ về nó. Người thứ hai này sẽ có trong đầu một bức tranh toàn thể về vấn đề cần giải quyết, chứ không chỉ là giải pháp của đoạn mã lúc đó. Điều này sẽ gián tiếp đảm bảo một chất lượng tốt hơn và dẫn tới một giải pháp mang tính tổng thể hơn. Đồng thời, điều này giúp cho họ theo được các chỉ dẫn của XP, đặc biệt là việc "tạo ca thử nghiệm trước". Nếu chỉ một người lập trình, họ sẽ rất dễ từ bỏ việc này, nhưng với hai người lập trình cùng làm việc thì họ có thể thay đổi nhau và giữ được các nguyên tắc của XP.

1.3.7 Tổ hợp các mô hình

Chúng ta đã xem xét các mô hình kỹ nghệ phần mềm như là các cách tiếp cận khác nhau tới kỹ nghệ phần mềm chứ không phải là các cách tiếp cận bổ sung cho nhau. Tuy nhiên trong nhiều trường hợp chúng ta có thể và cũng nên tổ hợp các khuôn cảnh để đạt được sức mạnh của từng khuôn cảnh cho một dự án riêng lẻ.

Ví dụ, khuôn cảnh xoắn ốc thực hiện điều này một cách trực tiếp, tổ hợp cả làm bản mẫu và các yếu tố của vòng đời cổ điển trong một cách tiếp cận tiến hóa tới kỹ nghệ phần mềm. Các kỹ thuật thế hệ thứ tư có thể được dùng để cài đặt bản mẫu hay cài đặt hệ thống sản xuất trong bước mã hóa của vòng đời cổ điển. Chúng ta có thể làm bản mẫu trong bước phân tích của mô hình vòng đời cổ điển.

Kết luận ở đây là chúng ta không nên bị lệ thuộc với bất cứ khuôn cảnh cụ thể nào. Tính chất và qui mô của phần mềm cần phát triển sẽ là yếu tố quyết định tới chọn khuôn cảnh. Mỗi cách tiếp cận đều có ưu điểm riêng và bằng cách tổ hợp khéo léo các cách tiếp cận thì chúng ta sẽ có một phương pháp hỗn hợp ưu việt hơn các phương pháp được dùng độc lập.

1.3.8 Tính khả thị của quá trình kỹ nghệ

Do đặc điểm là các phần tử lôgic nên quá trình phát triển phần mềm rất khó kiểm soát. Người ta tìm cách khắc phục vấn đề này bằng cách làm cho quá trình phát triển trở nên "nhìn thấy được", tức là ở mỗi bước (hoạt động) trong tiến trình phát triển phải tạo ra một sản phẩm hay tài liệu tương ứng.

Người quản lý dự án và cả khách hàng sẽ tiến hành xét duyệt các tài liệu này. Các tài liệu sẽ trở nên rất hữu ích cho công đoạn kiểm thử và nâng cấp phần mềm.

Ví dụ, đối với hoạt động phân tích chúng ta có các tài liệu như: báo cáo nghiên cứu khả thi, mô hình hệ thống, phác họa yêu cầu, đặc tả yêu cầu...

Chúng ta hãy so sánh tính khả thị của các khuôn cảnh đã biết:

- vòng đời cổ điển có tính khả thị cao do các bước phát triển tường minh, mô hình xoắn ốc cũng có tính khả thị tốt.

- Đối với mô hình làm bản mẫu, nếu tần số sửa chữa là lớn thì tính khả thị kém và việc tạo ra tài liệu là không hiệu quả.

- 4GT thì mới chỉ dùng với những ứng dụng nghiệp vụ đặc thù nên khó phát biểu gì về tính khả thị của nó.

Việc xây dựng tài liệu cũng có những vấn đề như:

- tạo ra các chi phí phụ làm chậm tiến trình phát triển

- khi phát hiện vấn đề về thiết kế, nhiều khi do không muốn thay đổi các tài liệu đã được xét duyệt, người phát triển có xu hướng dùng các giải pháp cục bộ không hiệu quả.

Các mô hình phát triển truyền thống thường chú trọng tới khâu lập tài liệu để nâng cao tính khả thị. Ngược lại, mô hình lập trình cực đoan (XP) lại không khuyến khích việc tạo nhiều tài liệu.

1.3.9 Vấn đề giảm kích cỡ của phần mềm

Như chúng ta đã biết, phần mềm hiện nay càng lớn, càng phức tạp. Một mặt, năng lực của nhóm lập trình không phải là tuyến tính so với năng lực của từng cá nhân. Độ phức tạp cũng tăng theo cấp số nhân, kéo theo chi phí cũng tăng theo cấp số nhân so với kích cỡ của chương trình cần phát triển.

Do đó, việc tìm cách giảm kích cỡ, độ phức tạp của chương trình là ưu tiên hàng đầu của kỹ nghệ phần mềm. Tại các bước phân tích thiết kế, giảm kích cỡ được thực hiện thông qua áp dụng chiến lược "chia để trị". Tức là chúng ta chia phần mềm thành các modun con có tính độc lập cao. Độ phức tạp của từng modun sẽ nhỏ hơn nhiều so với cả hệ thống, các modun con cũng có thể được phát triển song song.

Tại giai đoạn mã hóa, giảm kích cỡ có thể thực hiện được thông qua các phương thức như:

- dùng lại: dùng lại các thư viện đã phát triển, các thư viện thương mại...

- tự sinh mã: sử dụng các công cụ tự động hỗ trợ kỹ nghệ phần mềm (visual

modeling tools, GUI builders, CASE tools...)

- kỹ thuật hướng đối tượng: kỹ thuật hướng đối tượng hỗ trợ phát triển modun có tính dùng lại cao nhờ có cơ chế che dấu thông tin và khả năng kế thừa

- dùng các ngôn ngữ bậc cao (các ngôn ngữ có cấu trúc và năng lực biểu diễn cao)

Chúng ta xem xét ví dụ về việc lựa chọn ngôn ngữ. Việc chọn ngôn ngữ phụ thuộc nhiều vào miền ứng dụng, các ràng buộc về hiệu năng của phần mềm, tuy nhiên năng lực biểu diễn của ngôn ngữ cũng là một yếu tố quan trọng. Bảng 1.1 đưa ra một thống kê liên quan đến năng lực biểu diễn của ngôn ngữ: số dòng lệnh/đơn vị chức năng. VB không phải là một ngôn ngữ có cấu trúc cao nhưng được sử dụng rộng rãi trong các ứng dụng vừa và nhỏ cho môi trường Windows. Ngoài tính dễ học, dễ dùng, một trong những nguyên nhân giúp VB lan rộng chính là năng lực biểu diễn cao.

Bảng 1.1: Năng lực biểu diễn của ngôn ngữ

Ngôn ngữ	LOC/FP
Assembly	320
C	128
FORTRAN 77	105
COBOL 85	91
Ada 83	71
C++	56
Ada 95	55
Java	55
Visual Basic	35

1.4 Cái nhìn chung về kỹ nghệ phần mềm

Tiến trình phát triển kỹ nghệ phần mềm chứa ba giai đoạn chính bất kể mô hình kỹ nghệ phần mềm được chọn lựa. Ba giai đoạn này là xác định, phát triển và bảo trì, được gặp phải trong mọi dự án phát triển phần mềm, bất kể tới miền ứng dụng, kích cỡ và độ phức tạp.

Giai đoạn xác định tập trung vào khái niệm *cái gì*. Tức là trong khi xác định, người phát triển phần mềm cố gắng tập trung vào xác định thông tin nào cần được xử lý, chức năng và hiệu năng nào là cần có, giao diện nào cần được thiết lập, ràng buộc thiết kế nào hiện có và tiêu chuẩn hợp lệ nào cần có để xác định ra một hệ thống thành công. Yêu cầu chủ chốt của hệ thống và phần mềm cũng được xác định. Mặc dầu các phương pháp được áp dụng trong giai đoạn xác định thay đổi tùy theo mô hình kỹ nghệ phần mềm (hay tổ hợp các mô hình) được áp dụng, có ba bước riêng vẫn xuất hiện dưới dạng:

1. Phân tích hệ thống: Phân tích hệ thống xác định ra vai trò của từng phần tử trong một hệ thống dựa trên máy tính, tức là vạch ra vai trò mà phần mềm (cần phát triển)

sẽ giữ.

2. Lập kế hoạch dự án phần mềm: Một khi vai trò của phần mềm đã được thiết lập, rủi ro đã được phân tích, tài nguyên đã được cấp phát, chi phí đã được ước lượng thì phải xác định cụ thể các công việc cần thực hiện và lập lịch thực hiện chúng.

3. Phân tích yêu cầu: Trong bước phân tích hệ thống chúng ta chỉ xác định được vai trò chung của phần mềm. Sau khi đã chính thức quyết đinh phát triển phần mềm, chúng ta cần phải phân tích để xác định chi tiết lĩnh vực thông tin, các chức năng cũng như các ràng buộc khi vận hành của phần mềm. Phân tích yêu cầu là khâu kỹ thuật quan trọng đầu tiên để đảm bảo chất lượng (tính đáng tin cậy) của phần mềm. Nếu xác định sai yêu cầu thì các bước kỹ thuật khác có tốt đến đâu thì phần mềm cũng sẽ không được đưa vào sử dụng.

Giai đoạn phát triển tập trung vào khái niệm *thế nào*. Tức là, trong giai đoạn này người phát triển phần mềm từng bước xác định cách cấu trúc dữ liệu và kiến trúc phần mềm cần xây dựng, cách các chi tiết thủ tục được cài đặt, cách dịch thiết kế vào ngôn ngữ lập trình (hay ngôn ngữ phi thủ tục) và cách thực hiện kiểm thử. Phương pháp được áp dụng trong giai đoạn phát triển sẽ thay đổi tùy theo mô hình nhưng có ba bước đặc thù bao giờ cũng xuất hiện dưới dạng:

1. Thiết kế phần mềm: Là quá trình "dịch" các yêu cầu phần mềm thành một tập các biểu diễn (dựa trên đồ họa, bảng, hay ngôn ngữ), mô tả cho cấu trúc dữ liệu, kiến trúc, thủ tục thuật toán và đặc trưng giao diện.

2. Mã hóa: Các biểu diễn thiết kế phải được biểu diễn bởi một (hay một vài) ngôn ngữ nhân tạo (ngôn ngữ lập trình qui ước hay ngôn ngữ phi thủ tục được dùng trong khuôn cảnh 4GT) mà sẽ tạo ra kết quả là các lệnh thực hiện được trên máy tính.

3. Kiểm thử phần mềm: Một khi phần mềm đã được cài đặt dưới dạng máy thực hiện được, cần phải kiểm thử nó để phát hiện các lỗi phân tích, thiết kế, cài đặt và đánh giá tính hiệu quả.

Giai đoạn bảo trì tập trung vào những *thay đổi* gắn với việc sửa lỗi, thích ứng khi môi trường phần mềm tiến hóa và sự nâng cao gây ra bởi sự thay đổi yêu cầu của người dùng. Giai đoạn bảo trì áp dụng lại các bước của giai đoạn xác định và phát triển, nhưng là việc thực hiện trong hoàn cảnh phần mềm hiện có. Có ba kiểu thay đổi gặp phải trong giai đoạn bảo trì:

1. Sửa đổi: Cho dù có các hoạt động bảo đảm chất lượng tốt nhất, vẫn có thể là khách hàng sẽ phát hiện ra khiếm khuyết trong phần mềm. Bảo trì sửa đổi làm thay đổi phần mềm để sửa các khiếm khuyết (lỗi lập trình, thuật toán, thiết kế...).

2. Thích nghi: Qua thời gian, môi trường ban đầu (như CPU, hệ điều hành, ngoại vi) để phát triển phần mềm có thể sẽ thay đổi. Bảo trì thích nghi thực hiện việc sửa đổi phần mềm để thích hợp với những thay đổi môi trường ngoài.

3. Nâng cao: Khi phần mềm được dùng, khách hàng/người dùng sẽ nhận ra những chức năng phụ sẽ có lợi. Bảo trì hoàn thiện mở rộng phần mềm ra ngoài các yêu cầu chức năng gốc của nó.

Tổng kết

Phần mềm đã trở thành phần tử chủ chốt của các hệ thống máy tính. Phát triển phần mềm đã tiến hóa từ xây dựng một công cụ xử lý thông tin thành một ngành công nghiệp.

Phần mềm là phần tử lôgíc cho nên việc kiểm soát nó khó hơn nhiều so với phần tử vật lý. Khó có thể tối ưu hóa đồng thời các tính năng cần có của phần mềm. Ví dụ, các tính năng như giao diện đồ họa dễ sử dụng và sự hoạt động hiệu quả, tích kiệm tài nguyên hệ thống trong hầu hết các trường hợp là loại trừ lẫn nhau. Thách thức lớn đối với việc phát triển phần mềm là chúng ta phải xây dựng phần mềm tốt theo một lịch trình và kinh phí định trước.

Kỹ nghệ phần mềm là một bộ môn tích hợp cả các phương pháp, công cụ và thủ tục để phát triển phần mềm máy tính. Có một số mô hình khác nhau cho kỹ nghệ phần mềm, mỗi mô hình đều có những mặt mạnh và điểm yếu, nhưng nói chung tất cả đều có một dãy các giai đoạn tổng quát là: xác định, phát triển và bảo trì.

Chương 2

Phân tích và đặc tả yêu cầu

2.1 Đại cương về phân tích và đặc tả

Phân tích và định rõ yêu cầu là bước kỹ thuật đầu tiên trong tiến trình kỹ nghệ phần mềm. Công việc ở bước này là tìm hiểu xem chúng ta phải phát triển cái gì, chứ không phải là phát triển như thế nào. Đích cuối cùng của khâu phân tích là tạo ra *đặc tả yêu cầu*, là tài liệu ràng buộc giữa khách hàng và người phát triển và là cơ sở của hợp đồng.

Hoạt động phân tích là hoạt động phối hợp giữa khách hàng và người phân tích (bên phát triển). Khách hàng phát biểu yêu cầu và người phân tích hiểu, cụ thể hóa và biểu diễn lại yêu cầu.

Hoạt động phân tích giữ một vai trò đặc biệt quan trọng trong phát triển phần mềm, giúp cho đảm bảo chất lượng của phần mềm (phần mềm đáng tin cậy). Phần mềm đáng tin cậy có nghĩa là phải thực hiện được chính xác, đầy đủ yêu cầu của người sử dụng. Nếu phân tích không tốt dẫn đến hiểu lầm yêu cầu thì việc sửa chữa sẽ trở nên rất tốn kém. Chi phí để sửa chữa sai sót về yêu cầu sẽ tăng lên gấp bội nếu như sai sót đó được phát hiện muộn, ví dụ như ở bước thiết kế hay mã hóa.

Việc phân tích, nắm bắt yêu cầu thường gặp các khó khăn như

- các yêu cầu thường mang tính đặc thù của tổ chức đặt hàng nó, do đó nó thường khó hiểu, khó định nghĩa và không có chuẩn biểu diễn

- các hệ thống thông tin lớn có nhiều người sử dụng thì các yêu cầu thường rất đa dạng và có các mức ưu tiên khác nhau, thậm chí mâu thuẫn lẫn nhau

- người đặt hàng nhiều khi là các nhà quản lý, không phải là người dùng thực sự do đó việc phát biểu yêu cầu thường không chính xác

Trong phân tích cần phân biệt giữa *yêu cầu* và *mục tiêu* của hệ thống. Yêu cầu là một đòi hỏi mà chúng ta có thể kiểm tra được còn mục tiêu là cái trừu tượng hơn mà chúng ta hướng tới.

Ví dụ, *giao diện của hệ thống phải thân thiện với người sử dụng* là một mục tiêu và nó tương đối không khách quan và khó kiểm tra. Có nghĩa là với một phát biểu chung chung như vậy thì khách hàng và nhà phát triển khó định ra được một ranh giới rõ ràng để nói rằng phần mềm đã thỏa mãn được đòi hỏi đó. Với một mục tiêu như vậy, một

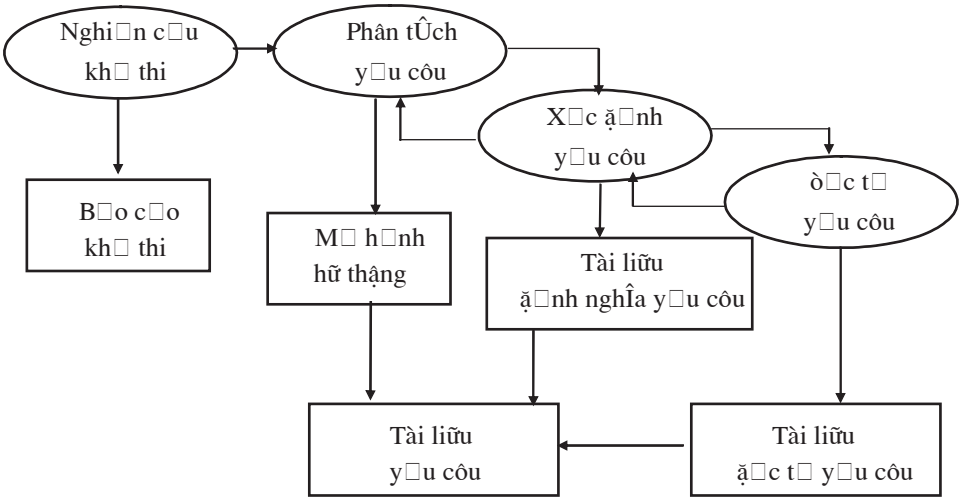
yêu cầu cho nhà phát triển có thể là *giao diện đồ họa mà các lệnh phải được chọn bằng menu*.

Mục đích của giai đoạn phân tích là xác định rõ các yêu cầu của phần mềm cần phát triển.

Tài liệu yêu cầu nên dễ hiểu với người dùng, đồng thời phải chặt chẽ để làm cơ sở cho hợp đồng và để cho người phát triển dựa vào đó để xây dựng phần mềm. Do đó yêu cầu thường được mô tả ở nhiều mức chi tiết khác nhau phục vụ cho các đối tượng đọc khác nhau. Các mức đó có thể là:

- Định nghĩa (xác định) yêu cầu: mô tả một cách dễ hiểu, vắn tắt về yêu cầu, hướng vào đối tượng người đọc là người sử dụng, người quản lý...
- Đặc tả yêu cầu: mô tả chi tiết về các yêu cầu, các ràng buộc của hệ thống, phải chính xác sao cho người đọc không hiểu nhầm yêu cầu, hướng vào đối tượng người đọc là các kỹ sư phần mềm (người phát triển), kỹ sư hệ thống (sẽ làm việc bảo trì)...

Các tài liệu yêu cầu cần được thẩm định để đảm bảo thỏa mãn nhu cầu người dùng. Đây là công việc bắt buộc để đảm bảo chất lượng phần mềm. Đôi khi việc xác định đầy đủ yêu cầu trước khi phát triển hệ thống là không thực tế và khi đó việc xây dựng các bản mẫu để nắm bắt yêu cầu là cần thiết.



Hình 2.1: Quá trình hình thành các yêu cầu.

2.2 Nghiên cứu khả thi

Đây là giai đoạn có tầm quan trọng đặc biệt, vì nó liên quan đến việc lựa chọn giải pháp. Trong giai đoạn này người phân tích phải làm rõ được các điểm mạnh và điểm yếu của hệ thống cũ, đánh giá được mức độ, tầm quan trọng của từng vấn đề, định ra các vấn đề cần phải giải quyết (ví dụ: những dịch vụ mới, thời hạn đáp ứng, hiệu quả

kinh tế). Sau đó người phân tích phải định ra một vài giải pháp có thể (sơ bộ) và so sánh cân nhắc các điểm tốt và không tốt của các giải pháp đó (như tính năng của hệ thống, giá cả cài đặt, bảo trì, việc đào tạo người sử dụng...). Đó là việc tìm ra một điểm cân bằng giữa nhu cầu và khả năng đáp ứng.

Mọi dự án đều khả thi khi nguồn tài nguyên vô hạn và thời gian vô hạn. Nhưng việc xây dựng hệ thống lại phải làm với sự hạn hẹp về tài nguyên và khó (nếu không phải là không hiện thực) bảo đảm đúng ngày bàn giao. Phân tích khả thi và rủi ro có liên quan với nhau theo nhiều cách. Nếu rủi ro của dự án là lớn thì tính khả thi của việc chế tạo phần mềm có chất lượng sẽ bị giảm đi.

Trong giai đoạn nghiên cứu khả thi, chúng ta tập trung vào bốn lĩnh vực quan tâm chính:

1. Khả thi về kinh tế: chi phí phát triển cần phải cân xứng với lợi ích mà hệ thống được xây dựng đem lại. Tính khả thi về kinh tế thể hiện trên các nội dung sau:

- Khả năng tài chính của tổ chức cho phép thực hiện dự án.

- Lợi ích mà dự án phát triển HTTT mang lại đủ bù đắp chi phí phải bỏ ra xây dựng nó.

- Tổ chức chấp nhận được những chi phí thường xuyên khi hệ thống hoạt động

Một thuật ngữ hay dùng để chỉ tài liệu nghiên cứu khả thi về kinh tế là luận chứng kinh tế. Luận chứng kinh tế nói chung được coi như nền tảng cho hầu hết các hệ thống (các ngoại lệ là hệ thống quốc phòng, hệ thống luật, các hệ thống phục vụ cho các nghiên cứu đặc biệt). Luận chứng kinh tế bao gồm:

- các mối quan tâm, nhất là phân tích chi phí/lợi ích

- chiến lược phát triển dài hạn của công ty

- sự ảnh hưởng tới các sản phẩm lợi nhuận khác

- chi phí cho tài nguyên cần cho việc xây dựng và phát triển thị trường tiềm năng

2. Khả thi về kỹ thuật: khảo cứu về chức năng, hiệu suất và ràng buộc có thể ảnh hưởng tới khả năng đạt tới một hệ thống chấp nhận được. Nói cách khác, khả thi kỹ thuật là xem xét khả năng kỹ thuật hiện tại có đủ đảm bảo thực hiện giải pháp công nghệ dự định áp dụng hay không.

Khả thi kỹ thuật thường là lĩnh vực khó thâm nhập nhất tại giai đoạn phân tích. Điều thực chất là tiến trình phân tích và xác định nhu cầu cần được tiến hành song song với việc xác nhận tính khả thi kỹ thuật. Các xem xét thường được gắn với tính khả thi kỹ thuật bao gồm:

Rủi ro xây dựng: liệu các phần tử hệ thống có thể được thiết kế sao cho đạt được chức năng và hiệu suất cần thiết thỏa mãn những ràng buộc trong khi phân tích không?

Có sẵn tài nguyên: có sẵn các nhân viên cho việc xây dựng phần tử hệ thống đang xét không? Các tài nguyên cần thiết khác (phần cứng và phần mềm) có sẵn cho việc xây dựng hệ thống không ?

Công nghệ: công nghệ liên quan đã đạt tới trạng thái sẵn sàng hỗ trợ cho hệ thống chưa?

3. Khả thi về pháp lý: nghiên cứu và đưa ra phán quyết về có hay không sự xâm

phạm, vi phạm pháp luật hay khó khăn pháp lý từ việc xây dựng và vận hành hệ thống. Tính khả thi pháp lý bao gồm một phạm vi rộng các mối quan tâm kể cả hợp đồng, nghĩa vụ pháp lý, sự vi phạm và vô số các bẫy pháp lý khác mà thường là các nhân viên kỹ thuật không biết tới. Trong nước, vấn đề khả thi về pháp lý vẫn chưa được coi trọng một cách đúng mức mặc dù đã có một số luật liên quan đến CNTT và bảo hộ bản quyền.

4. Tính khả thi về hoạt động: đánh giá tính khả thi của việc vận hành hệ thống. Trong mỗi phương án người ta cần xem xét hệ thống có thể vận hành trôi chảy hay không trong khuôn khổ tổ chức và điều kiện quản lý mà tổ chức đó (người dùng, khách hàng) có.

Mức độ các phương án được xem xét tới trong nghiên cứu khả thi thường bị giới hạn bởi các ràng buộc về chi phí và thời gian.

2.3 Nền tảng của phân tích yêu cầu

2.3.1 Các nguyên lý phân tích

Trên hai thập kỉ qua, người ta đã xây dựng ra một số phương pháp phân tích và đặc tả phần mềm. Những người nghiên cứu đã xác định ra các vấn đề và nguyên nhân của chúng, và đã xây dựng ra các qui tắc và thủ tục để vượt qua chúng. Mỗi phương pháp đều có kí pháp và quan điểm riêng. Tuy nhiên, tất cả các phương pháp này đều có quan hệ với một tập hợp các nguyên lý cơ bản:

1. Miền thông tin của vấn đề phải được biểu diễn lại và hiểu rõ.

2. Các mô hình mô tả cho thông tin, chức năng và hành vi hệ thống cần phải được xây dựng.

3. Các mô hình (và vấn đề) phải được phân hoạch theo cách để lộ ra các chi tiết theo kiểu phân tầng (hay cấp bậc).

4. Tiến trình phân tích phải đi từ thông tin bản chất hướng tới chi tiết cài đặt.

Bằng cách áp dụng những nguyên lý này, người phân tích tiếp cận tới vấn đề một cách hệ thống. Miền thông tin cần được xem xét sao cho người ta có thể hiểu rõ chức năng một cách đầy đủ. Các mô hình được dùng để cho việc trao đổi thông tin được dễ dàng theo một cách ngắn gọn. Việc phân hoạch vấn đề được sử dụng để làm giảm độ phức tạp. Những cách nhìn nhận cả từ góc độ bản chất và góc độ cài đặt về phần mềm đều cần thiết để bao hàm được các ràng buộc logic do yêu cầu xử lý áp đặt nên cùng các ràng buộc vật lý do các phần tử hệ thống khác áp đặt nên.

2.3.2 Mô hình hóa

Chúng ta tạo ra các mô hình để thu được hiểu biết rõ hơn về thực thể thực tế cần xây dựng. Khi thực thể là một vật vật lý (như toà nhà, máy bay, máy móc) thì ta có thể xây dựng một mô hình giống hệt về hình dạng, nhưng nhỏ hơn về qui mô. Tuy nhiên, khi

thực thể cần xây dựng là phần mềm, thì mô hình của chúng ta phải mang dạng khác. Nó phải có khả năng mô hình hóa thông tin mà phần mềm biến đổi, các chức năng (và chức năng con) làm cho phép biến đổi đó thực hiện được, và hành vi của hệ thống khi phép biến đổi xảy ra.

Trong khi phân tích các yêu cầu phần mềm, chúng ta tạo ra các mô hình về hệ thống cần xây dựng. Các mô hình tập trung vào điều hệ thống phải thực hiện, không chú ý đến cách thức nó thực hiện. Trong nhiều trường hợp, các mô hình chúng ta tạo ra có dùng kí pháp đồ hoạ mô tả cho thông tin, xử lý, hành vi hệ thống, và các đặc trưng khác thông qua các biểu tượng phân biệt và dễ nhận diện. Những phần khác của mô hình có thể thuần túy văn bản. Thông tin mô tả có thể được cung cấp bằng cách dùng một ngôn ngữ tự nhiên hay một ngôn ngữ chuyên dụng cho mô tả yêu cầu.

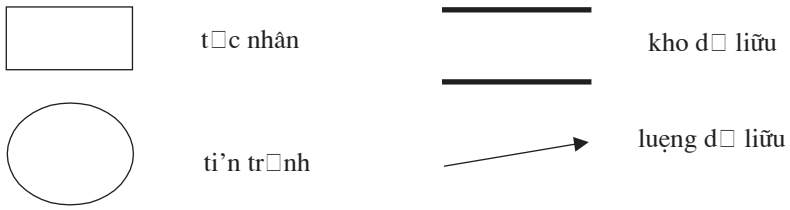
Các mô hình được tạo ra trong khi phân tích yêu cầu còn đóng một số vai trò quan trọng:

- Mô hình trợ giúp cho người phân tích trong việc hiểu về thông tin, chức năng và hành vi của hệ thống, do đó làm cho nhiệm vụ phân tích yêu cầu được dễ dàng và hệ thống hơn.
- Mô hình trở thành tiêu điểm cho việc xem xét và do đó, trở thành phần mấu chốt cho việc xác định tính đầy đủ, nhất quán và chính xác của đặc tả.
- Mô hình trở thành nền tảng cho thiết kế, cung cấp cho người thiết kế một cách biểu diễn chủ yếu về phần mềm có thể được "ánh xạ" vào hoàn cảnh cài đặt.

Dưới đây là một số mô hình (phương pháp) hay được dùng trong phân tích:

1. Biểu đồ luồng dữ liệu

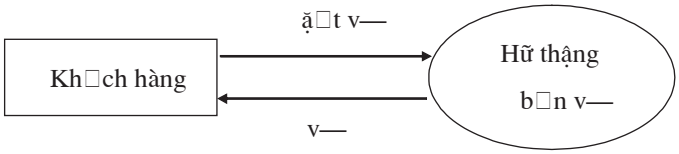
Khi thông tin đi qua phần mềm nó bị thay đổi bởi một loạt các phép biến đổi. Biểu đồ luồng dữ liệu (Data Flow Diagram - DFD) là một kỹ thuật vẽ ra luồng dữ liệu di chuyển trong hệ thống và những phép biến đổi được áp dụng lên dữ liệu. Ký pháp cơ bản của biểu đồ luồng dữ liệu được minh họa trên hình 2.2.



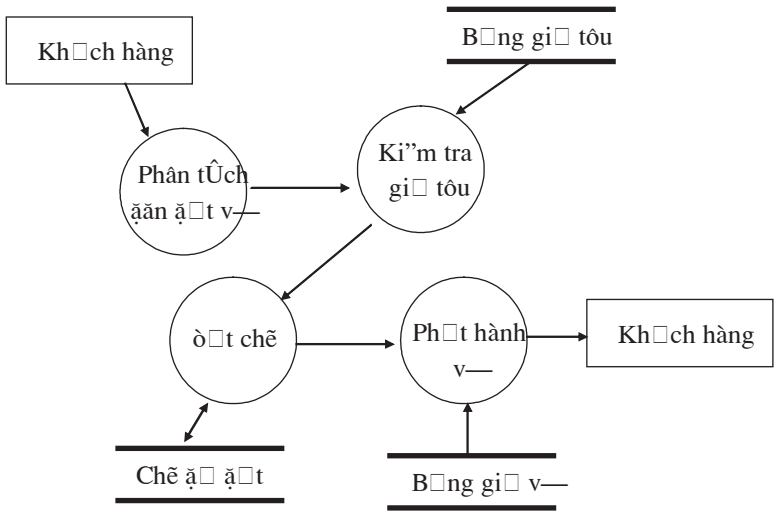
Hình 2.2: Ký pháp DFD.

Biểu đồ luồng dữ liệu có thể được dùng để biểu diễn cho một hệ thống hay phần mềm ở bất kì mức trừu tượng nào. Trong thực tế, DFD có thể được phân hoạch thành nhiều mức biểu diễn cho chi tiết chức năng và luồng thông tin ngày càng tăng. Do đó phương pháp dùng DFD còn được gọi là phân tích có cấu trúc. Một DFD mức 0, cũng còn được gọi là biểu đồ nền tảng hay biểu đồ ngữ cảnh hệ thống, biểu diễn cho toàn bộ phần tử phần mềm như một hình tròn với dữ liệu vào và ra được chỉ ra bởi các mũi tên tới và đi tương ứng. Một DFD mức 1 cụ thể hóa của DFD mức 0 và có thể chứa nhiều

hình tròn (chức năng) với các mũi tên (luồng dữ liệu) nối lẫn nhau. Mỗi một trong các tiến trình được biểu diễn ở mức 1 đều là chức năng con của toàn bộ hệ thống được mô tả trong biểu đồ ngữ cảnh. Hình 2.3 minh họa một DFD cho hệ thống bán vé tầu.



DFD m□c 0



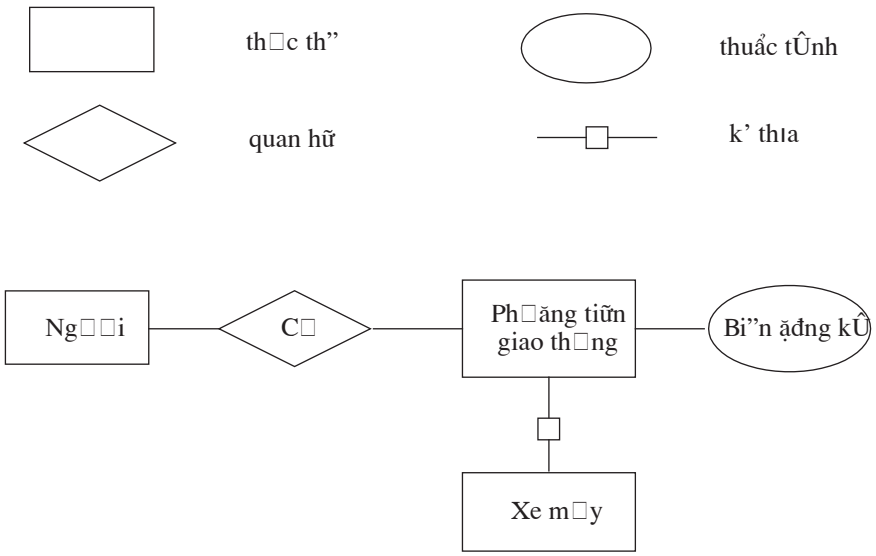
DFD m□c 1

Hình 2.3: Biểu đồ luồng dữ liệu của một hệ thống bán vé tầu.

2. Biểu đồ thực thể quan hệ

Ký pháp nền tảng cho mô hình hóa dữ liệu là biểu đồ thực thể - quan hệ (Entity - Relation Diagram). Tất cả đều xác định một tập các thành phần chủ yếu cho biểu đồ E-R: thực thể, thuộc tính, quan hệ và nhiều chỉ báo kiểu khác nhau. Mục đích chính của biểu đồ E-R là biểu diễn dữ liệu và mối quan hệ của các dữ liệu (thực thể).

Ký pháp của biểu đồ E-R cũng tương đối đơn giản. Các thực thể được biểu diễn bằng các hình chữ nhật có nhãn. Mối quan hệ được chỉ ra bằng hình thoi. Các mối nối giữa sự vật dữ liệu và mối quan hệ được thiết lập bằng cách dùng nhiều đường nối đặc biệt (hình 2.4).



Hình 2.4: Mô hình thực thể quan hệ người - phương tiện giao thông.

2.3.3 Người phân tích

Người phân tích đóng vai trò đặc biệt quan trọng trong tiến trình phân tích. Ngoài kinh nghiệm, một người phân tích tốt cần có các khả năng sau:

- Khả năng hiểu thấu các khái niệm trừu tượng, có khả năng tổ chức lại thành các phân tích logic và tổng hợp các giải pháp dựa trên từng dải phân chia.

- Khả năng rút ra các sự kiện thích đáng từ các nguồn xung khắc và lẫn lộn.

- Khả năng hiểu được môi trường người dùng/khách hàng.

- Khả năng áp dụng các phần tử hệ thống phần cứng và/hoặc phần mềm vào môi trường người sử dụng/khách hàng.

- Khả năng giao tiếp tốt ở dạng viết và nói.

- Khả năng trừu tượng hóa/tổng hợp vấn đề từ các sự kiện riêng lẻ.

2.4 Xác định và đặc tả yêu cầu

2.4.1 Xác định yêu cầu

Xác định yêu cầu là mô tả trừu tượng về các dịch vụ mà hệ thống cần cung cấp và các ràng buộc mà hệ thống cần tuân thủ khi vận hành. Nó chỉ mô tả các hành vi bên ngoài của hệ thống mà không liên quan tới các chi tiết thiết kế. Yêu cầu nên được viết sao cho có thể hiểu mà không cần một kiến thức chuyên môn đặc biệt nào.

Các yêu cầu được chia thành hai loại:

1) Các yêu cầu chức năng: các dịch vụ mà hệ thống cần cung cấp

2) Các yêu cầu phi chức năng: các ràng buộc mà hệ thống cần tuân thủ.

Các yêu cầu phi chức năng có thể chia làm 3 kiểu:

i) Yêu cầu sản phẩm: các yêu cầu về tốc độ, bộ nhớ, độ tin cậy, về tính khả chuyển và tái sử dụng...

ii) Yêu cầu về quá trình: yêu cầu đối với quá trình xây dựng sản phẩm như các chuẩn phải tuân theo, các phương pháp thiết kế, ngôn ngữ lập trình...

iii) Yêu cầu khác: các yêu cầu không thuộc hai nhóm trên như về tính pháp lý, về chi phí, về thành viên nhóm phát triển...

Các yêu cầu phi chức năng thường rất đặc thù với từng khách hàng và do đó khó phân tích và đặc tả một cách đầy đủ và chính xác.

Về nguyên tắc, yêu cầu của hệ thống phải vừa đầy đủ vừa không được mâu thuẫn nhau. Đối với các hệ thống lớn phức tạp thì chúng ta khó đạt được hai yếu tố này trong các bước phân tích đầu. Trong các bước duyệt lại yêu cầu cần phải bổ sung, chỉnh lý tư liệu yêu cầu.

2.4.2 Đặc tả yêu cầu

Tài liệu xác định yêu cầu là mô tả hướng khách hàng và được viết bởi ngôn ngữ của khách hàng. Khi đó có thể dùng ngôn ngữ tự nhiên và các khái niệm trừu tượng.

Tài liệu dặc tả yêu cầu (đặc tả chức năng) là mô tả hướng người phát triển, là cơ sở của hợp đồng làm phần mềm. Nó không được phép mơ hồ, nếu không sẽ dẫn đến sự hiểu nhầm bởi khách hàng hoặc người phát triển.

Với một yêu cầu mơ hồ thì người phát triển sẽ thực hiện nó một cách rẻ nhất còn khách hàng thì không muốn vậy. Do đó khách hàng có thể đòi hỏi sửa đổi chức năng phần mềm khi nó đã gần hoàn thiện khiến cho chi phí tăng và chậm thời điểm bàn giao.

Chi phí cho sửa các sai sót trong phát biểu yêu cầu là rất lớn, đặc biệt là khi các sai sót này được phát hiện khi đã bắt đầu xây dựng hệ thống. Theo một số thống kê thì 85% mã phải viết lại do thay đổi yêu cầu và 12% lỗi phát hiện trong 3 năm đầu sử dụng là do đặc tả yêu cầu không chính xác.

Do đó, việc *đặc tả chính xác yêu cầu* là mối quan tâm được đặt lên hàng đầu. Có hai phương pháp đặc tả là

1. Đặc tả phi hình thức: là cách đặc tả bằng ngôn ngữ tự nhiên

2. Đặc tả hình thức: là cách đặc tả bằng các ngôn ngữ nhân tạo (ngôn ngữ đặc tả), các công thức và biểu đồ

Đặc tả phi hình thức (ngôn ngữ tự nhiên) thuận tiện cho việc xác định yêu cầu nhưng nhiều khi không thích hợp với đặc tả yêu cầu vì:

i) Không phải lúc nào người đọc và người viết đặc tả bằng ngôn ngữ tự nhiên cũng hiểu các từ như nhau.

ii) Ngôn ngữ tự nhiên quá mềm dẻo do đó các yêu cầu liên quan đến nhau có thể được biểu diễn bằng các hình thức hoàn toàn khác nhau và người phát triển không nhận ra các mối liên quan này.

iii) Các yêu cầu khó được phân hoạch một cách hữu hiệu do đó hiệu quả của việc

đổi thay chỉ có thể xác định được bằng cách kiểm tra tất cả các yêu cầu chứ không phải một nhóm các yêu cầu liên quan.

Các ngôn ngữ đặc tả (đặc tả hình thức) khắc phục được các hạn chế trên, tuy nhiên đa số khách hàng lại không thông thạo các ngôn ngữ này. Thêm nữa mỗi ngôn ngữ đặc tả hình thức thường chỉ phục vụ cho một nhóm lĩnh vực riêng biệt và việc đặc tả hình thức là một công việc tốn kém thời gian.

Một cách tiếp cận là bên cạnh các đặc tả hình thức người ta viết các chú giải bằng ngôn ngữ tự nhiên để giúp khách hành dễ hiểu.

2.4.3 Thẩm định yêu cầu

Một khi các yêu cầu đã được thiết lập thì cần thẩm định xem chúng có thỏa mãn các nhu cầu của khách hàng hay không. Nếu thẩm định không được tiến hành chặt chẽ thì các sai sót có thể lan truyền sang các giai đoạn thiết kế và mã hóa khiến cho việc sửa đổi hệ thống trở nên rất tốn kém.

Mục tiêu của thẩm định là kiểm tra xem yêu cầu mà người phân tích xác định có thỏa mãn 4 yếu tố sau không:

1. Thỏa mãn nhu cầu của người dùng: cần phải thỏa mãn được các nhu cầu bản chất của người dùng (khách hàng).

2. Các yêu cầu không được mâu thuẫn nhau: với các hệ thống lớn các yêu cầu rất đa dạng và có khả năng sẽ mâu thuân nhau. Khi đó người phân tích phải loại bớt các yêu cầu (không chủ chốt) để sau cho các yêu cầu được mô tả trong tài liệu yêu cầu không mâu thuẫn nhau.

3. Các yêu cầu phải đầy đủ: cần chứa mọi chức năng và ràng buộc mà người dùng đã nhắm đến.

4. Các yêu cầu phải hiện thực: yêu cầu phải hiện thực về các khía cạnh kỹ thuật, kinh tế và pháp lý.

2.5 Làm bản mẫu trong quá trình phân tích

Đối với các hệ thống phức tạp, nhiều khi chúng ta không nắm chắc được yêu cầu của khách hàng, chúng ta cũng khó đánh giá được tính khả thi cũng như hiệu quả của hệ thống. Một cách tiếp cận đối với trường hợp này là xây dựng bản mẫu. Bản mẫu vừa được dùng để phân tích yêu cầu vừa có thể tiến hóa thành sản phẩm cuối cùng.

Bản mẫu phần mềm hoàn toàn khác với bản mẫu phần cứng. Khi phát triển các hệ thống phần cứng, thì thực tế người ta phát triển một bản mẫu hệ thống để thẩm định thiết kế hệ thống. Một bản mẫu hệ thống điện tử có thể được thực hiện và được thử nghiệm bằng cách dùng các thành phần chưa được lắp ráp vào vỏ trước khi đầu tư vào các mạch tích hợp chuyên dụng đắt tiền để thực hiện một đời sản phẩm mới của hệ thống. Bản mẫu phần mềm không phải nhằm vào việc thẩm định thiết kế (thiết kế của nó thường là hoàn toàn khác với hệ thống được phát triển cuối cùng), mà là để thẩm

định yêu cầu của người sử dụng.

2.5.1 Các bước làm bản mẫu

Xây dựng bản mẫu bao gồm 6 bước sau:

Bước 1: Đánh giá yêu cầu phần mềm và xác định liệu phần mềm cần xây dựng có xứng đáng để làm bản mẫu không

Không phải tất cả các phần mềm đều có thể đưa tới làm bản mẫu. Ta có thể xác định một số nhân tố làm bản mẫu: lĩnh vực ứng dụng, độ phức tạp ứng dụng, đặc trưng khách hàng và đặc trưng dự án.

Để đảm bảo tính tương tác giữa khách hàng với bản mẫu, chúng ta cần đảm bảo các điều kiện:

1. Khách hàng phải cam kết dùng tài nguyên để đánh giá và làm mịn bản mẫu (chi tiết hóa yêu cầu)

2. Khách hàng phải có khả năng đưa ra những quyết định về yêu cầu một cách kịp thời

Bước 2: Với một dự án chấp thuận được, người phân tích xây dựng một cách biểu diễn vắn tắt cho yêu cầu.

Trước khi có thể bắt đầu xây dựng một bản mẫu, người phân tích phải biểu diễn miền thông tin và các lĩnh vực, hành vi và chức năng của vấn đề rồi xây dựng một cách tiếp cận hợp lý tới việc phân hoạch. Có thể ứng dụng các nguyên lý phân tích nền tảng (phân tích top-down) và các phương pháp phân tích yêu cầu.

Bước 3: Sau khi đã duyệt xét mô hình yêu cầu, phải tạo ra một đặc tả thiết kế vắn tắt cho bản mẫu

Việc thiết kế phải xuất hiện trước khi bắt đầu làm bản mẫu. Tuy nhiên thiết kế tập trung chủ yếu vào các vấn đề thiết kế dữ liệu và kiến trúc mức đỉnh chứ không tập trung vào thiết kế thủ tục chi tiết.

Bước 4: Phần mềm bản mẫu được tạo ra, kiểm thử và làm mịn.

Bản mẫu nên được xây dựng một cách nhanh chóng và với một chi phí nhỏ. Một cách lý tưởng, bản mẫu nên được lắp ráp từ các khối chức năng (thư viện...) đã có. Có thể dùng các ngôn ngữ bậc cao hay các công cụ tự động để xây dựng bản mẫu.

Bước 5: Khách hàng đánh giá và làm mịn yêu cầu

Bước này là cốt lõi của cách tiếp cận làm bản mẫu. Chính ở đây mà khách hàng có thể xem xét cách biểu diễn được cài đặt cho yêu cầu phần mềm, gợi ý những thay đổi làm cho phần mềm đáp ứng tốt hơn với các nhu cầu thực tế.

Bước 6: Lặp lại các bước 4 và 5 cho tới khi tất cả các yêu cầu đã được hình thức hóa đầy đủ hay cho tới khi bản mẫu đã tiến hóa thành một phần mềm hoàn thiện.

2.5.2 Lợi ích và hạn chế của phát triển bản mẫu

Phát triển bản mẫu đem lại các lợi ích sau:

1. Sự hiểu lầm giữa những người phát triển phần mềm và những người sử dụng phần mềm có thể được nhận thấy rõ khi các chức năng của hệ thống được trình diễn.

2. Sự thiếu hụt các dịch vụ người dùng có thể được phát hiện.

3. Sự khó sử dụng hoặc sự nhầm lẫn các dịch vụ người dùng có thể được thấy rõ và được sửa lại.

4. Đội ngũ phát triển phần mềm có thể tìm ra được các yêu cầu không đầy đủ hoặc không kiên định khi họ phát triển bản mẫu.

5. Một hệ thống hoạt động được (mặc dầu là hạn chế) là bằng chứng thuyết minh cho tính khả thi và tính hữu ích của ứng dụng cho các nhà quản lý.

6. Bản mẫu đó được dùng làm cơ sở cho việc viết đặc tả một sản phẩm.

Mặc dù mục tiêu chủ yếu của việc tạo bản mẫu là để phát triển, thẩm định các yêu cầu phần mềm, nó cũng có các lợi ích khác như:

1. Dùng để huấn luyện người sử dụng ngay từ trước khi hệ thống được phân phối.

2. Dùng trong quá trình thử nghiệm hệ thống. Điều đó nghĩa là cùng các trường hợp thử như nhau vừa dùng cho thử bản mẫu vừa cho thử hệ thống. Kết quả khác nhau có nghĩa là có sai sót.

Tạo bản mẫu là một kỹ thuật giảm bớt rủi ro. Một rủi ro lớn trong việc phát triển phần mềm là các sai sót mà đến giai đoạn cuối mới phát hiện và việc chỉnh sửa vào thời điểm đó là rất tốn kém. Kinh nghiệm cho thấy việc tạo bản mẫu sẽ giảm bớt số các vấn đề của đặc tả yêu cầu và giá cả tổng cộng của việc phát triển có thể là thấp hơn nếu ta phát triển bản mẫu.

Hạn chế của cách tiếp cận tạo bản mẫu là:

1. Để đơn giản hóa việc tạo bản mẫu người ta có thể bỏ qua các yêu cầu phức tạp. Sự thật hẳn là không thể tạo bản mẫu một vài phần quan trọng nhất của hệ thống như các đặc điểm về sự an toàn - nguy kịch.

2. Các yêu cầu phi chức năng như độ tin cậy, độ an toàn hay hiệu năng thường không được biểu thị đầy đủ trong bản mẫu.

3. Do tính chưa hoàn thiện về chức năng/hiệu năng, người dùng có thể không dùng bản mẫu y như cách dùng một hệ thống thực đang hoạt động. Do đó, chất lượng đánh giá của khách hàng nhiều khi không cao.

2.6 Định dạng đặc tả yêu cầu

Kết quả của bước phân tích là tạo ra bản đặc tả yêu cầu phần mềm (Software Requirement Specification - SRS). Đặc tả yêu cầu phải chỉ rõ được phạm vi của sản phẩm, các chức

năng cần có, đối tượng người sử dụng và các ràng buộc khi vận hành sản phẩm. Có nhiều chuẩn khác nhau trong xây dựng tài liệu, dưới đây là một định dạng RSR thông dụng (theo chuẩn IEEE 830-1984).

1 Giới thiệu

1.1 Mục đích

Mục này giới thiệu mục đích của tài liệu yêu cầu. Thường chỉ đơn giản là định nghĩa "đây là tài liệu yêu cầu về phần mềm XYZ".

1.2 Phạm vi

Nêu pham vi đề cập của tài liệu yêu cầu.

1.3 Định nghĩa

Định nghĩa các khái niệm, các từ viết tắt, các chuẩn được sử dụng trong tài liệu yêu cầu.

1.4 Tài liệu tham khảo

Nêu danh mục các tài liệu tham khảo dùng để tạo ra bản đặc tả yêu cầu.

1.5 Mô tả chung về tài liệu

Mô tả khái quát cấu trúc tài liệu, gồm có các chương, mục, phục lục chính nào.

2 Mô tả chung

2.1 Tổng quan về sản phẩm

Mô tả khái quát về sản phẩm.

2.2 Chức năng sản phẩm

Khái quát về chức năng sản phẩm.

2.3 Đối tượng người dùng

Mô tả về đối tượng người dùng.

2.4 Ràng buộc tổng thể

Khái quát về các ràng buộc của phần mềm: ràng buộc phần cứng, môi trường sử dụng, yêu cầu kết nối với các hệ thống khác...

2.5 Giả thiết và sự lệ thuộc

Mô tả các giả thiết khi áp dụng tài liệu: ví dụ như tên phần cứng, phần mềm, hệ điều hành cụ thể.

3 Yêu cầu chi tiết

Mô tả các yêu cầu

3.1 Yêu cầu chức năng

Mô tả chi tiết về các yêu cầu chức năng.

3.1.1 Yêu cầu chức năng 1

3.1.1.1 Giới thiệu

3.1.1.2 Dữ liệu vào

3.1.1.3 Xử lý

3.1.1.4. Kết quả

3.1.2 Yêu cầu chức năng 2

...

3.1.n Yêu cầu chức năng n

3.2 Yêu cầu giao diện ngoài

Mô tả các giao diện của phần mềm với môi trường bên ngoài.

3.2.1 Giao diện người dùng

3.2.2 Giao diện phần cứng

3.2.3 Giao diện phần mềm

3.2.4 Giao diện truyền thông

3.3 Yêu cầu hiệu suất

Mô tả về hiệu suất, ví dụ như thời gian phản hồi với sự kiện, số giao dịch được thực hiện/giây,...

3.4 Ràng buộc thiết kế

Mô tả các ràng buộc thiết kế, ví dụ các ràng buộc về ngôn ngữ, về công nghệ, về cơ sở dữ liệu và về chuẩn giao tiếp.

3.5 Thuộc tính

Mô tả các thuộc tính của phần mềm.

3.5.1 Tính bảo mật, an toàn và khả năng phục hồi

Mức độ bảo mật dữ liệu, cách thức truy cập vào hệ thống. Độ an toàn của hệ thống đối với các trường hợp bất thường như mất điện... Khả năng phục hồi của hệ thống sau khi gặp sự cố.

3.5.2 Tính bảo trì

Các chức năng, giao diện đòi hỏi phải dễ sửa đổi (dễ bảo trì).

3.6 Các yêu cầu khác

Các yêu cầu khác liên quan đến sản phẩm.

Tổng kết

Phân tích và định rõ yêu cầu là bước kỹ thuật đầu tiên trong tiến trình kỹ nghệ phần mềm. Tại bước này các phát biểu chung về phạm vi phần mềm được làm mịn thành một bản đặc tả cụ thể để trở thành nền tảng cho mọi hoạt động kỹ nghệ phần mềm sau đó.

Việc phân tích phải tập trung vào các miền thông tin, chức năng và hành vi của vấn đề. Để hiểu rõ yêu cầu, người ta tạo ra mô hình, phân hoạch vấn đề và tạo ra những biểu diễn mô tả cho bản chất của yêu cầu rồi sau đó đi vào các chi tiết.

Trong nhiều trường hợp, không thể nào đặc tả được đầy đủ mọi vấn đề tại giai đoạn đầu. Việc làm bản mẫu thường giúp chỉ ra cách tiếp cận khác để từ đó có thể làm mịn thêm yêu cầu. Để tiến hành đúng đắn việc làm bản mẫu, có thể cần tới các công cụ và kỹ thuật đặc biệt.

Kết quả của việc phân tích là tạo ra bản đặc tả các yêu cầu phần mềm. Đặc tả cần được xét duyệt để đảm bảo rằng người phát triển và khách hàng có cùng nhận biết về hệ thống cần phát triển.

Chương 3

Thiết kế phần mềm

3.1 Khái niệm về thiết kế phần mềm

3.1.1 Khái niệm

Có thể định nghĩa thiết kế là một quá trình áp dụng nhiều kỹ thuật và các nguyên lý để tạo ra mô hình của một thiết bị, một tiến trình hay một hệ thống đủ chi tiết mà theo đó có thể chế tạo ra sản phẩm vật lý tương ứng với nó.

Bản chất thiết kế phần mềm là một quá trình chuyển hóa các yêu cầu phần mềm thành một biểu diễn thiết kế. Từ những mô tả quan niệm về toàn bộ phần mềm, việc làm mịn (chi tiết hóa) liên tục dẫn tới một biểu diễn thiết kế rất gần với cách biểu diễn của chương trình nguồn để có thể ánh xạ vào một ngôn ngữ lập trình cụ thể.

Mục tiêu thiết kế là để tạo ra một mô hình biểu diễn của một thực thể mà sau này sẽ được xây dựng.

Mô hình chung của một thiết kế phần mềm là một đồ thị có hướng, các nút biểu diễn các thực thể có trong thiết kế, các liên kết biểu diễn các mỗi quan hệ giữa các thực thể đó.

Hoạt động thiết kế là một loại hoạt động đặc biệt:

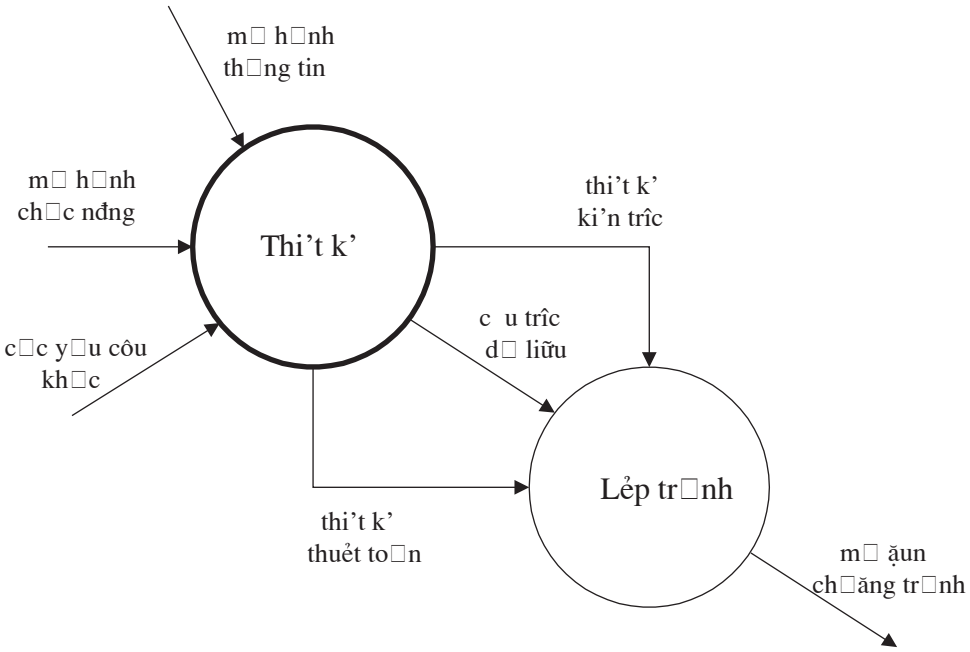
- Là một quá trình sáng tạo, đòi hỏi có kinh nghiệm và sự nhanh nhạy và sáng tạo

- Cần phải được thực hành và học bằng kinh nghiệm, bằng khảo sát các hệ đang tồn tại, chỉ học bằng sách vở là không đủ.

3.1.2 Tầm quan trọng

Tầm quan trọng của thiết kế phần mềm có thể được phát biểu bằng một từ "chất lượng". Thiết kế là nơi chất lượng phần mềm được nuôi dưỡng trong quá trình phát triển: cung cấp cách biểu diễn phần mềm có thể được xác nhận về chất lượng, là cách duy nhất mà chúng ta có thể chuyển hóa một cách chính xác các yêu cầu của khách hàng thành sản phẩm hay hệ thống phần mềm cuối cùng.

Thiết kế phần mềm là công cụ giao tiếp làm cơ sở để có thể mô tả một cách đầy



Hình 3.1: Vai trò của thiết kế phần mềm trong quá trình kỹ nghệ.

đủ các dịch vụ của hệ thống, để quản lý các rủi ro và lựa chọn giải pháp thích hợp.

Thiết kế phần mềm phục vụ như một nền tảng cho mọi bước kỹ nghệ phần mềm và bảo trì:

- Không có thiết kế có nguy cơ sản sinh một hệ thống không ổn định - một hệ thống sẽ thất bại. Một hệ thống phần mềm rất khó xác định được chất lượng chừng nào chưa đến bước kiểm thử. Thiết kế tốt là bước quan trọng đầu tiên để đảm bảo chất lượng phần mềm.

3.1.3 Quá trình thiết kế

Thiết kế phần mềm là quá trình chuyển các đặc tả yêu cầu dịch vụ thông tin của hệ thống thành đặc tả hệ thống phần mềm. Thiết kế phần mềm trải qua một số giai đoạn chính sau:

1. Nghiên cứu để hiểu ra vấn đề. Không hiểu rõ vấn đề thì không thể có được thiết kế hữu hiệu.

2. Chọn một (hay một số) giải pháp thiết kế và xác định các đặc điểm thô của nó. Chọn giải pháp phụ thuộc vào kinh nghiệm của người thiết kế, vào các cấu kiện dùng lại được và vào sự đơn giản của các giải pháp kéo theo. Nếu các nhân tố khác là tương tự thì nên chọn giải pháp đơn giản nhất.

3. Mô tả trừu tượng cho mỗi nội dung trong giải pháp. Trước khi tạo ra các tư liệu chính thức người thiết kế cần phải xây dựng một mô tả ban đầu sơ khai rồi chi tiết hóa nó. Các sai sót và khiếm khuyết trong mỗi mức thiết kế trước đó được phát hiện và phải được chỉnh sửa trước khi lập tư liệu thiết kế.

Kết quả của mỗi hoạt động thiết kế là một đặc tả thiết kế. Đặc tả này có thể là một đặc tả trừu tượng, hình thức và được tạo ra để làm rõ các yêu cầu, nó cũng có thể là một đặc tả về một phần nào đó của hệ thống phải được thực hiện như thế nào. Khi quá trình thiết kế tiến triển thì các chi tiết được bổ sung vào đặc tả đó. Các kết quả cuối cùng là các đặc tả về các thuật toán và các cấu trúc dữ liệu được dùng làm cơ sở cho việc thực hiện hệ thống.

Các hoạt động thiết kế chính trong một hệ thống phần mềm lớn:

Các nội dung chính của thiết kế là:

- Thiết kế kiến trúc: Xác định hệ tổng thể phần mềm bao gồm các hệ con và các quan hệ giữa chúng và ghi thành tài liệu

- Đặc tả trừu tượng: các đặc tả trừu tượng cho mỗi hệ con về các dịch vụ mà nó cung cấp cũng như các ràng buộc chúng phải tuân thủ.

- Thiết kế giao diện: giao diện của từng hệ con với các hệ con khác được thiết kế và ghi thành tài liệu; đặc tả giao diện không được mơ hồ và cho phép sử dụng hệ con đó mà không cần biết về thiết kế nội tại của nó.

- Thiết kế các thành phần: các dịch vụ mà một hệ con cung cấp được phân chia cho các thành phần hợp thành của nó.

- Thiết kế cấu trúc dữ liệu: thiết kế chi tiết và đặc tả các cấu trúc dữ liệu (các mô hình về thế giới thực cần xử lý) được dùng trong việc thực hiện hệ thống.

- Thiết kế thuật toán: các thuật toán được dùng cho các dịch vụ được thiết kế chi tiết và được đặc tả.

Quá trình này được lặp lại cho đến khi các thành phần hợp thành của mỗi hệ con được xác định đều có thể ánh xạ trực tiếp vào các thành phần ngôn ngữ lập trình, chẳng hạn như các gói, các thủ tục và các hàm.

3.1.4 Cơ sở của thiết kế

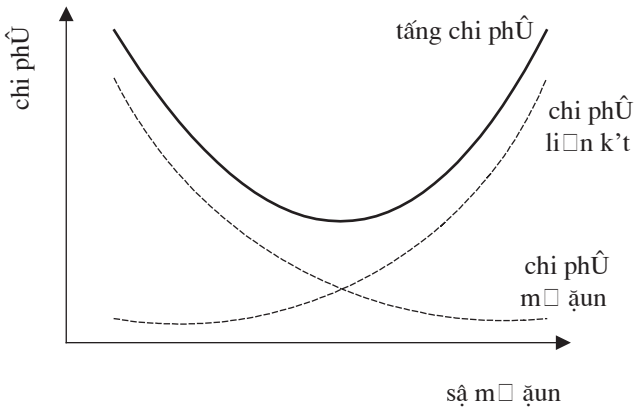
Phần mềm được chia thành các thành phần có tên riêng biệt và xác định được địa chỉ, gọi là các mô đun, được tích hợp để thỏa mãn yêu cầu của vấn đề.

Người ta nói rằng: tính môđun là thuộc tính riêng của phần mềm cho phép một chương trình trở nên quản lý được theo cách thông minh. Người đọc không thể nào hiểu thấu phần mềm nguyên khối (như một chương trình lớn chỉ gồm một môđun). Điều này dẫn đến kết luận "chia để trị" sẽ dễ giải quyết một vấn đề phức tạp hơn khi chia nó thành những phần quản lý được.

Với cùng một tập hợp các yêu cầu, nhiều môđun hơn có nghĩa là kích cỡ từng môđun nhỏ; độ phức tạp giảm và chi phí cho phát triển môđun giảm. Nhưng khi số các mô đun tăng lên thì nỗ lực liên kết chúng bằng việc làm giao diện cho các môđun cũng tăng lên. Đặc trưng này dẫn đến đường cong tổng chi phí (nỗ lực) như trong hình 3.2.

Chúng ta nên mô đun hóa nhưng cần phải duy trì chi phí trong vùng lân cận của chi phí tối thiểu. Môđun hóa còn chưa đủ hay quá mức đều nên tránh. Một gợi ý cho

kích cỡ của các môđun cơ sở là mỗi môđun đảm nhận một chức năng cơ bản.



Hình 3.2: Tính môđun và chi phí phần mềm.

3.1.5 Mô tả thiết kế

Một bản thiết kế phần mềm là một mô hình mô tả một đối tượng của thế giới thực có nhiều thành phần và các mối quan hệ giữa chúng với nhau.

Việc mô tả thiết kế cần đảm bảo thực hiện được các yêu cầu:

- làm cơ sở cho việc triển khai chương trình
- làm phương tiện giao tiếp giữa các nhóm thiết kế các hệ con
- cung cấp đủ thông tin cho những người bảo trì hệ thống

Thiết kế thường được mô tả ở hai mức: thiết kế mức cao (high level design) và thiết kế chi tiết (low level design). Thiết kế mức cao hay thiết kế kiến trúc chỉ ra:

- mô hình tổng thể của hệ thống
- cách thức hệ thống được phân rã thành các môđun
- mối quan hệ (gọi nhau) giữa các môđun
- cách thức trao đổi thông tin giữa các môđun (giao diện, các dữ liệu dùng chung, các thông tin trạng thái)

Tuy nhiên thiết kế mức cao không chỉ ra được thứ tự thực hiện, số lần thực hiện của môđun, cũng như các trạng thái và hoạt động bên trong của mỗi môđun.

Nội dung của các môđun được thể hiện ở mức thiết kế chi tiết. Các cấu trúc cơ sở của thiết kế chi tiết hay còn gọi là thiết kế thuật toán là:

- cấu trúc tuần tự
- cấu trúc rẽ nhánh
- cấu trúc lặp

Mọi thuật toán đều có thể mô tả dựa trên 3 cấu trúc trên. Có ba loại hình mô tả thường được sử dụng trong thiết kế:

- Dạng văn bản phi hình thức: Mô tả bằng ngôn ngữ tự nhiên các thông tin không

thể hình thức hóa được như các thông tin phi chức năng. Bên cạnh các cách mô tả khác, mô tả văn bản thường được bổ sung để làm cho thiết kế được đầy đủ và dễ hiểu hơn.

- Các biểu đồ: Các biểu đồ được dùng để thể hiện các mối quan hệ giữa các thành phần lập lên hệ thống và là mô hình mô tả thế giới thực. Việc mô tả đồ thị của các thiết kế là rất có lợi vì tính trực quan và cho một bức tranh tổng thể về hệ thống. Trong thời gian gần đây, người ta đã xây dựng được một ngôn ngữ đồ thị dành riêng cho các thiết kế phần mềm với tên gọi: ngôn ngữ mô hình hóa thống nhất (Unified Modeling Model - UML). Tại mức thiết kế chi tiết, có một số các dạng biểu đồ hay được sử dụng là flow chart, JSP, Nassi-Shneiderman diagrams.

- Giả mã (pseudo code): Hiện nay, giả mã là công cụ được ưa chuộng để mô tả thiết kế ở mức chi tiết. Các ngôn ngữ này thuận tiện cho việc mô tả chính xác thiết kế, tuy nhiên lại thiếu tính trực quan.

Dưới đây là một ví dụ sử dụng giả mã:

```
£˚c´#iy˚# g˚E?# w„Y#

     Eq r#P L Y„fi#
        ƒ˚E?# |:˚A|
     #fir#
        ƒ˚E?# |:rA|
     #"iEq
     ƒ˚E?# "„Y#

 #"i £˚c´#iy˚#
```

Nói chung thì cả ba loại biểu diễn trên đây đều được sử dụng trong thiết kế hệ thống. Thiết kế kiến trúc thường được mô tả bằng đồ thị (structure chart)và được bổ sung văn bản phi hình thức, thiết kế dữ liệu lôgic thường được mô tả bằng các bảng, các thiết kế giao diện, thiết kế cấu trúc dữ liệu chi tiết, thiết kế thuật toán thường được mô tả bằng pseudo code.

3.1.6 Chất lượng thiết kế

Không có cách nào hay để xác định được thế nào là thiết kế tốt. Tiêu chuẩn dễ bảo trì là tiêu chuẩn tốt cho người dùng. Một thiết kế dễ bảo trì có thể thích nghi với việc cải biên các chức năng và việc thêm các chức năng mới. Một thiết kế như thế phải dễ hiểu và việc sửa đổi chỉ có hiệu ứng cục bộ. Các thành phần thiết kế phải là kết dính (cohesive) theo nghĩa là tất cả các bộ phận trong thành phần phải có một quan hệ logic chặt chẽ, các thành phần ghép nối (coupling) với nhau là lỏng lẻo. Ghép nối càng lỏng lẻo thì càng dễ thích nghi, nghĩa là càng dễ sửa đổi để phù hợp với hoàn cảnh mới.

Để xem một thiết kế có là tốt hay không, người ta tiến hành thiết lập một số độ đo chất lượng thiết kế:

1) Sự kết dính (Cohesion)

Sự kết dính của một môđun là độ đo về tính khớp lại với nhau của các phần trong môđun đó. Nếu một môđun chỉ thực hiện một chức năng logic hoặc là một thực thể logic, tức là tất cả các bộ phận của môđun đó đều tham gia vào việc thực hiện một công việc thì độ kết dính là cao. Nếu một hoặc nhiều bộ phận không tham gia trực tiếp vào việc chức năng logic đó thì mức độ kết dính của nó là thấp. Thiết kế là tốt khi độ kết dính cao. Khi đó chúng ta sẽ dễ dàng hiểu được từng môđun và việc sửa chữa một môđun sẽ không (ít) ảnh hưởng tới các môđun khác.

Constantine và Yourdon định ra 7 mức kết dính theo thứ tự tăng dần sau đây:

a. Kết dính gom góp: các công việc không liên quan với nhau, song lại bị bó vào một môđun.

b. Kết dính logic: các thành phần cùng thực hiện các chức năng tương tự về logic chẳng hạn như vào/ra, xử lý lỗi,... được đặt vào cùng một mô đun.

c. Kết dính thời điểm: tất cả các thành phần cùng hoạt hóa một lúc, chẳng hạn như các thao tác khởi tạo được bó lại với nhau.

d. Kết dính thủ tục: các phần tử trong môđun được ghép lại trong một dãy điều khiển.

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

f. Kết dính tuần tự: trong một môđun, đầu ra của phần tử này là đầu vào của phần tử khác.

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

Các lớp kết dính này không được định nghĩa chặt chẽ và cũng không phải luôn luôn xác định được.

Một đối tượng kết dính nếu nó thể hiện như một thực thể đơn: tất cả các phép toán trên thực thể đó đều nằm trong thực thể đó. Vậy có thể xác định một lớp kết dính nữa là:

h. Kết dính đối tượng: mỗi phép toán đều liên quan đến thay đổi, kiểm tra và sử dụng thuộc tính của một đối tượng, là cơ sở cung cấp các dịch vụ của đối tượng.

2) Sự ghép nối (Coupling)

Ghép nối là độ đo sự nối ghép với nhau giữa các đơn vị (môđun) của hệ thống. Hệ thống có nối ghép cao thì các môđun phụ thuộc lẫn nhau lớn. Hệ thống nối ghép lỏng lẻo thì các môđun là độc lập hoặc là tương đối độc lập với nhau và chúng ta sẽ dễ bảo trì nó.

Các mô đun 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 bảo đảm rằng các thông tin cục bộ được che dấu trong các môđun và các môđun trao đổi thông tin thông qua danh sách tham số (giao diện) xác

định. Có thể chia ghép nối thành các mức từ chặt chẽ đến lỏng lẻo như sau:

a. Ghép nối nội dung: hai hay nhiều môđun dùng lẫn dữ liệu của nhau, đây là mức xấu nhất, thường xẩy ra đối với các ngôn ngữ mức thấp dùng các dữ liệu toàn cục hay lạm dụng lệnh GOTO.

b. Ghép nối chung: một số môđun dùng các biến chung, nếu xẩy ra lỗi thao tác dữ liệu, sẽ khó xác định được lỗi đó do môđun nào gây ra.

c. Ghép nối điều khiển: một môđun truyền các thông tin điều khiển để điều khiển hoạt động của một môđun khác.

d. Ghép nối dư thừa: môđun nhận thông tin thừa không liên quan trực tiếp đến chức năng của nó, điều này sẽ làm giảm khả năng thích nghi của môđun đó.

e. Ghép nối dữ liệu: Các môđun trao đổi thông tin thông qua tham số và giá trị trả lại.

f. Ghép nối không có trao đổi thông tin: môđun thực hiện một chức năng độc lập và hoàn toàn không nhận tham số và không có giá trị trả lại.

Ưu việt của thiết kế hướng đối tượng là do bản chất che dấu thông tin của đối tượng dẫn tới việc tạo ra các hệ ghép nối lỏng lẻo.

Việc thừa kế trong hệ thống hướng đối tượng lại dẫn tới một dạng khác của ghép nối, ghép nối giữa đối tượng mức cao và đối tượng kế thừa nó.

3) Sự hiểu được (Understandability)

Sự hiểu được của thiết kế liên quan tới một số đặc trưng sau đây:

a. Tính kết dính: có thể hiểu được thành phần đó mà không cần tham khảo tới 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 tên 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 trong 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 như thế nào?

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 cấu trúc if-then-elsse. 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 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.

4) Sự thích nghi được (Adaptability)

Một thiết kế dễ bảo trì thì nó phải sẵn sàng thích nghi được, nghĩa là các thành phần của chúng nên được ghép nối lỏng lẻo. Một thành phần có thể là ghép nối lỏng lẻo theo nghĩa là chỉ hợp tác với các thành phần khác thông qua việc truyền các thông báo. Sự thích nghi được còn có nghĩa là thiết kế phải được soạn thảo tư liệu tốt, dễ hiểu và nhất quán.

Để có độ thích nghi thì hệ thống còn cần phải phải tự chứa. Muốn là tự chứa một cách hoàn toàn thì một hệ thống 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 hệ thống.

Một trong những ưu việt chính của kế thừa trong thiết kế hướng đối tượng là các thành phần này có thể sẵn sàng thích nghi được. Cơ cấu thích nghi được này không dựa trên việc cải biên thành phần đã có mà dựa trên việc tạo ra một thành phần mới thừa kế các thuộc tính và các chức năng của thành phần đó. Chúng ta chỉ cần thêm các thuộc tính và chức năng cần thiết cho thành phần mới. Các thành phần khác dựa trên thành phần cơ bản đó sẽ không bị ảnh hưởng gì.

3.2 Thiết kế hướng chức năng

3.2.1 Cách tiếp cận 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 tác động lẫn nhau, mà mỗi 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.

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 công cụ CASE đều là hướng chức năng. Vô khối các hệ thống đã được phát triển bằng cách sử dụng phương pháp tiếp cận hướng chức năng. Các hệ thống đó sẽ 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.

Trong thiết kế hướng chức năng, người ta dùng các biểu đồ luồng dữ liệu (mô tả việc xử lý dữ liệu), các lược đồ cấu trúc (nó chỉ ra cấu trúc của phần mềm), và các mô tả thiết kế chi tiết.

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. Việc thay đổi một chức năng và cách nó sử dụng trạng thái của hệ thống có thể gây ra những tương tác bất ngờ đối với các chức năng khác.

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

3.2.2 Biểu đồ luồng dữ liệu

Biểu đồ luồng dữ liệu chỉ ra cách thức biến đổi dữ liệu vào thành dữ liệu ra thông qua một dãy các phép biến đổi. Bước thứ nhất của thiết kế hướng chức năng là phát triển một biểu đồ luồng dữ liệu hệ thống. Biểu đồ này không nhất thiết bao gồm các thông tin điều khiển nhưng nên lập tư liệu các phép biến đổi dữ liệu.

Biểu đồ luồng dữ liệu là một phần hợp nhất của một số các phương pháp thiết kế và các công cụ CASE thường trợ giúp cho việc tạo ra biểu đồ luồng dữ liệu.

3.2.3 Lược đồ cấu trúc

Lược đồ cấu trúc chỉ ra cấu trúc các thành phần theo thứ bậc của hệ thống. Nó chỉ ra rằng các phần tử của một biểu đồ luồng dữ liệu có thể được thực hiện như thế nào với tư cách là một thứ bậc của các đơn vị chương trình. Lược đồ cấu trúc có thể được dùng như là một mô tả chương trình nhìn thấy được với các thông tin xác định các sự lựa chọn và các vòng lặp. Lược đồ cấu trúc được dùng để trình bày một tổ chức tĩnh của thiết kế.

3.2.4 Các từ điển dữ liệu

Từ điển dữ liệu vừa có ích cho việc bảo trì hệ thống vừa có ích trong quá trình thiết kế. Với mỗi khái niệm thiết kế, cần có một từ khóa mô tả ứng với từ khóa (entry) của từ điển dữ liệu cung cấp thông tin về khái niệm đó (kiểu, chức năng của dữ liệu...). Đôi khi người ta gọi cái này là một mô tả ngắn của chức năng thành phần.

Các từ điển dữ liệu dùng để nối các mô tả thiết kế kiểu biểu đồ và các mô tả thiết kế kiểu văn bản. Một vài bộ công cụ CASE cung cấp một phép nối tự động biểu đồ luồng dữ liệu và từ điển dữ liệu.

3.3 Thiết kế hướng đối tượng

3.3.1 Cách tiếp cận hướng đối tượng

Thiết kế hướng đối tượng dựa trên chiến lược che dấu thông tin cấu trúc vào bên trong các thành phần. 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 được nâng lên. Thiết kế là tương đối dễ thay đổi vì sự thay đổi cấu trúc một thành phần có thể không cần quan tâm tới các hiệu ứng phụ trên các thành phần khác.

Việc che dấu thông tin trong thiết kế hướng đối tượng là dựa trên sự nhìn hệ phần mềm như là một bộ các đối tượng tương tác với nhau chứ không phải là bộ các chức năng như cách tiếp cận chức năng. Các đối tượng có một trạng thái riêng được che dấu

và các phép toán trên trạng thái đó. Thiết kế biểu thị các dịch vụ được yêu cầu cùng với những hỗ trợ mà các đối tượng có tương tác với nó cung cấp.

3.3.2 Ba đặc trưng của thiết kế hướng đối tượng

Thiết kế hướng đối tượng bao gồm các đặc trưng chính sau:

1. Không có vùng dữ liệu dùng chung. Các đối tượng liên lạc với nhau bằng cách trao đổi thông báo.

2. Các đối tượng là các thực thể độc lập, dễ thay đổi vì rằng tất cả các trạng thái và các thông tin biểu diễn 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 khác.

3. Các đối tượng có thể phân tán và có thể hoạt động tuần tự hoặc song song. Đây là một trong những lý do khiến cho thiết kế hướng đối tượng được sử dụng rộng rãi trong các hệ thống nhúng.

3.3.3 Cơ sở của thiết kế hướng đối tượng

Cơ sở của thiết kế hướng đối tượng là các lớp. Lớp là một trừu tượng mô tả cho một nhóm sự vật. Đối tượng của một lớp là một thực thể (cụ thể hóa) của lớp đó.

Thiết kế của một lớp bao gồm:

- cấu trúc dữ liệu (thuộc tính)

- hàm, thủ tục (chức năng)

- giao diện (cung cấp khả năng trao đổi dữ liệu đối với các lớp khác, về bản chất là các chức năng của đối tượng)

Việc cài đặt các giao diện là một yếu tố quan trọng để đảm bao che dấu cấu trúc dữ liệu. Tức là thiết kế nội tại của đối tượng độc lập với giao diện do đó chúng ta có thể sửa đổi thiết kế mà không sợ ảnh hưởng tới các đối tượng khác.

Các đối tượng trao đổi với nhau bằng cách truyền các thông báo. Tức là một đối tượng yêu cầu một đối tượng khác thực hiện một chức năng nào đó. Thông báo bao gồm: tên đối tượng, tên phương thức, và các tham số.

Vòng đời của một đối tượng khi hệ thống hoạt động như sau:

- khởi tạo: hệ thống tạo ra đối tượng bằng cách xác lập vùng dữ liệu đồng thời tự động thực hiện các chức năng liên quan đến khởi tạo đối tượng.

- hoạt động: đối tượng nhận các thông báo và thực hiện các chức năng được yêu cầu.

- phá hủy: hệ thống giải phóng vùng nhớ đã được cấp phát sau khi thực hiện tự động các thao tác cần thiết để hủy đối tượng.

Nhờ có chức năng khởi tạo và phá hủy được gọi tự động chúng ta có thể tự động hóa được một số công việc và tránh được nhiều sai sót trong lập trình như quên khởi tạo dữ liệu, quên cấp phát hay quên giải phóng vùng nhớ động.

Sự kế thừa

Kế thừa là một khái niệm quan trọng trong thiết kế hướng đối tượng. Một lớp có thể được định nghĩa dựa trên sự kế thừa một hoặc nhiều lớp đã được định nghĩa. Kế thừa ở đây bao gồm

- kế thừa cấu trúc dữ liệu
- kế thừa chức năng

Khả năng kế thừa giúp cho rút gọn được chương trình và nâng cao tính tái sử dụng. Một chiến lược chung là trước tiên tạo ra các lớp trừu tượng (để có thể dùng chung) và đối với các bài toán cụ thể tạo ra các lớp kế thừa bằng cách thêm các thông tin đặc thù.

3.3.4 Các bước thiết kế

Thiết kế hướng đối tượng bao gồm các bước chính sau:

1. Xác định lớp đối tượng.
2. Xác định thuộc tính cho lớp: các biến của lớp
3. Xác định hành vi (chức năng): các hàm
4. Xác định tương tác giữa các lớp đối tượng: giao diện (thông báo).
5. Áp dụng tính kế thừa: xây dựng các lớp trừu tượng có các thuộc tính chung, đây là một khâu đặc trưng của thiết kế hướng đối tượng.

Ví dụ, giả sử cần xây dựng các lớp hình tròn, elíp và đa giác. Có thể thấy elip và hình tròn có một số các thuộc tính chung như tọa độ tâm, chúng ta có thể xây dựng lớp hình nón chứa các thuộc tính chung này. Giữa hình nón và đa giác lại có thể tìm ra các thuộc tính chung như mầu nền, mầu biên..., và có thể xây dựng lớp trừu tượng hình hình học chứa các thuộc tính này.

Phương pháp xác định đối tượng

Xác định đối tượng là một trong nhưng công đoạn thiết kế quan trọng, phụ thuộc nhiều vào kinh nghiệm và bài toán cụ thể. Có một số phương pháp được đề xuất, một trong các phương pháp này là phân tích từ vựng của "câu yêu cầu". Cụ thể là danh từ trong câu yêu cầu sẽ được coi là đối tượng và động từ sẽ được coi là chức năng.

Ví dụ: Với yêu cầu *Phần mềm Email cung cấp cho user khả năng gửi thư, đọc thư và soạn thảo thư điện tử.*, chúng ta có thể sơ bộ tạo ra các đối tượng *phần mềm, user, email* và các chức năng *gửi, nhận, soạn thảo*.

3.3.5 Ưu nhược điểm của thiết kế hướng đối tượng

Thiết kế hướng đối tượng có các ưu điểm chính sau:

- 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.

- Các đối tượng là các thành phần dùng lại được thích hợp do tính độc lập của chúng và khả năng kế thừa cao.
- Có một vài lớp hệ thống thể hiện phản ánh quan hệ 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 đạt được tính dễ hiểu của thiết kế.

Nhược điểm chính của thiết kế hướng đối tượng là sự khó nhận ra các đối tượng của một hệ thống. Cách nhìn tự nhiên đối với nhiều hệ thống là cách nhìn chức năng.

3.3.6 Quan hệ giữa thiết kế và lập trình hướng đối tượng

Thiết kế hướng đối tượng là một chiến lược thiết kế, không phụ thuộc vào ngôn ngữ thực hiện cụ thể nào. Các ngôn ngữ lập trình hướng đối tượng có các khả năng bao gói đối tượng, và kế thừa làm cho việc thực hiện thiết kế hướng đối tượng an toàn hơn và đơn giản hơn.

Một thiết kế hướng đối tượng cũng có thể được thực hiện trong một ngôn ngữ thủ tục kiểu như PASCAL hoặc C (không có các đặc điểm bao gói như vậy). 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 (tasks), do đó Ada cũng được dùng để mô tả các thiết kế hướng đối tượng.

Tuy nhiên, cũng phải nhấn mạnh rằng chúng ta có thể mô tả thiết kế hướng đối tượng trên các ngôn ngữ truyền thống nhưng chúng ta không thể kiểm tra được sự tuân thủ *tư tưởng hướng đối tượng* trên các ngôn ngữ này, nghĩa là người phát triển vẫn có thể truy cập đến cấu trúc dữ liệu vật lý của đối tượng và việc đó làm vô nghĩa khái niệm che dấu thông tin.

Việc chấp nhận thiết kế hướng đối tượng như là một chiến lược hữu hiệu dẫn đến sự phát triển rộng rãi các phương pháp thiết kế hướng đối tượng và các ngôn ngữ lập trình hướng đối tượng.

3.3.7 Quan hệ giữa thiết kế hướng đối tượng và hướng chức năng

Có nhiều quan niệm khác nhau về quan hệ giữa thiết kế hướng đối tượng và thiết kế hướng chức năng. Có người cho rằng, hai chiến lược thiết kế này hỗ trợ lẫn nhau, cụ thể là

- DFD dưa ra mô hình về các thuộc tính và chức năng
- luồng giao tác đưa ra hướng dẫn về tương tác giữa các đối tượng (thông báo)
- Mô hình E - R đưa ra hướng dẫn xây dựng đối tượng

Thêm nữa, thiết kế nội tại của lớp đối tượng có nhiều điểm tương đồng với thiết kế hướng chức năng.

Một quan điểm khác cho rằng thiết kế hướng đối tượng và thiết kế hướng chức năng là hai cách tiếp cận hoàn toàn khác nhau, các khái niệm như che dấu thông tin, kế thừa là đặc trưng quan trọng và bản chất của thiết kế hướng đối tượng và nếu không dứt bỏ cách nhìn thiết kế hướng chức năng thì không thể khai thác hiệu quả các đặc trưng này.

3.4 Thiết kế giao diện người sử dụng

Thiết kế hệ thống máy tính bao gồm một phổ rất rộng các công việc từ thiết kế phần cứng cho đến thiết kế giao diện người sử dụng. Giao diện của hệ thống thường là tiêu chuẩn so sánh để phán xét về hệ thống. Giao diện được thiết kế kém sẽ gây ra những nhầm lẫn cho người sử dụng, khiến cho họ không sử dụng được các chức năng cần thiết và trong trường hợp xấu có thể thực hiện các thao tác nguy hiểm như phá hủy thông tin cần thiết.

Tầm quan trọng của giao diện còn được xem xét trên hai yếu tố:

- khía cạnh nghiệp vụ: người dùng thông qua giao diện để tương tác với hệ thống, đây là khâu nghiệp vụ thủ công duy nhất do đó nếu được thiết kế tốt sẽ nâng cao tốc độ xử lý công việc và dẫn tới hiệu quả kinh tế cao.

- khía cạnh thương mại: đối với các sản phẩm bán hàng loạt, giao diện được thiết kế tốt (dễ sử dụng, đẹp) sẽ gây ấn tượng với khách hàng và là yếu tố chính khi khách hàng chọn mua sản phẩm.

Ngoài các yếu tố hiệu quả công việc, đẹp, dễ học dễ sử dụng, một thiết kế giao diện hiện đại nên có tính độc lập cao với khối chương trình xử lý dữ liệu.

Đối với nhiều hệ thống, giao diện là bộ phận có tầm quan trọng chủ chốt và có yêu cầu sửa đổi thường xuyên. Do đó, để tiện cho việc sửa đổi, giao diện nên được thiết kế có tính môđun hóa cao và nên có độ độc lập tối đa với khối chương trình xử lý dữ liệu. Điều này cũng dẫn đến khả năng chúng ta có thể xây dựng nhiều giao diện khác nhau cho các đối tượng sử dụng khác nhau hay chạy trên các hệ thống khác nhau.

Có hai dòng giao diện chính là:

- giao diện dòng lệnh: là loại giao diện đơn giản nhất, thường được thiết kế gắn chặt với chương trình và có tính di chuyển cao (tương đương với chương trình). Giao diện dòng lệnh phù hợp với các ứng dụng thuần túy xử lý dữ liệu, nhất là đối với các chương trình mà đầu ra là đầu vào của chương trình khác. Giao diện dòng lệnh gọn nhẹ, dễ xây dựng nhưng thường khó học, khó sử dụng và chỉ phù hợp với người dùng chuyên nghiệp trong các ứng dụng đặc thù.

- giao diện đồ họa: sử dụng các cửa sổ, menu, icon... cho phép người dùng có thể truy cập song song đến nhiều thông tin khác nhau; người dùng thường tương tác bằng cách phối hợp cả bàn phím và con chuột; giao diện đồ họa dễ học, dễ sử dụng và trở nên rất thông dụng và có độ chuẩn hóa cao.

Nhìn trên khía cạnh độc lập với khối chương trình xử lý, có một số cách thức xây dựng giao diện khác nhau:

- giao diện đồ họa (GUI) truyền thống: là giao diện đồ họa được thiết kế có độ liên

kết cao với chương trình (được xây dựng cùng ngôn ngữ, cùng bộ công cụ...), hầu hết các chương trình trên máy tính cá nhân sử dụng loại giao diện này.

- X protocol: giao diện đồ họa sử dụng giao thức X protocol, phổ biến trên các máy Unix/Linux. Loại giao diện này có ưu điểm là có thể hoạt động độc lập với khối chương trình còn lại, tức là ta có thể chạy giao diện trên một máy tính trong khi đó phần xử lý bên trong lại hoạt động trên một máy khác. Đáng tiếc là phương thức này vẫn chưa phổ biến trên các máy tính cá nhân (chạy hệ điều hành MS Windows).

- client/server: một cách tiếp cận để hướng tới tính độc lập và khả chuyển của giao diện là xây dựng giao diện như là một chương trình client, tương tác với khối chương trình xử lý (server) thông qua các giao thức trao đổi thông tin trên mạng (TCP/IP).

- web based: một trong các cách thức xây dựng giao diện phổ biến hiện nay là dựa trên nền web, sử dụng các trình duyệt web để trao dổi thông tin với server. Tuy có một số nhược điểm về an toàn thông tin và tốc độ nhưng với tính độc lập hoàn toàn với phần xử lý, độ chuẩn hóa cao và khả năng sẵn có trên hầu hết các thiết bị nối mạng, phương thức này đang được ứng dụng rộng rãi.

Thiết kế giao diện khác với thiết kế các chức năng khác của phần mềm ở điểm hướng tới người sử dụng, cần người sử dụng đánh giá. Các công đoạn thiết kế khác như thiết kế dữ liệu, thiết kế thuật toán che dấu hoạt động kỹ thuật chi tiết khỏi khách hàng. Ngược lại, khách hàng (người dùng tiềm ẩn) nên tham gia vào quá trình thiết kế giao diện. Kinh nghiệm và khả năng của họ cần phải được tính đến khi thiết kế giao diện.

3.4.1 Một số vấn đề thiết kế

Trong thiết kế giao diện, cần chú ý tới một số vấn đề sau:

1. Thời gian phản hồi

Chúng ta cần quan tâm tới hai loại thời gian là

- Thời gian đáp ứng trung bình: là thời gian trung bình mà hệ thống phản hồi đối với một yêu cầu của người dùng. Thời gian để sinh ra "kết quả thực sự" của yêu cầu sẽ phụ thuộc vào bản chất yêu cầu, thuật toán, tốc độ của máy tính, tuy nhiên chúng ta cần quan tâm khía cạnh tâm lý là nếu người dùng đợi quá lâu mà không nhận được thông tin gì thì họ sẽ nghĩ là có vấn đề và có thể sẽ tiến hành các thao tác ngoài mong đợi như lặp lại thao tác hay dừng hệ thống.
- Độ biến thiên của thời gian: độ biến thiên của thời gian cũng là đại lượng cần quan tâm. Nếu độ biến thiên lớn, ví dụ một thao tác thường được đáp ứng trong 1 giây mà có trường hợp phải mất 5 giây mới hoàn thành thì cũng có thể làm cho người dùng đưa ra các thao tác sai.

2. Các tiện ích

Một giao diện tốt cần có các tiện ích để trợ giúp người sử dụng. Có các loại tiện ích sau

- Tích hợp: là tiện ích được tích hợp vào giao diện như nút **Help** cung cấp các thuyết minh về thao tác.
- Phụ thêm: là các tiện ích phụ thêm như các tài liệu trực tuyến.
- Macro: một số chương trình còn cho phép người dùng tự động hóa một số thao tác bằng các lệnh kiểu macro.

3. Thông báo

Các thông báo do hệ thống đưa ra cần

- Có nghĩa: mọi thông báo cần có nghĩa đối với người dùng.
- Ngắn gọn: các thông báo cần ngắn gọn đi vào bản chất vấn đề, đặc biệt là đối với kiểu giao diện dòng lệnh.
- Có tính xây dựng: thông báo nên có tính xây dựng như đưa ra các nguyên nhân và các hướng khắc phục.

3.4.2 Một số hướng dẫn thiết kế

Dưới đây là một số yếu tố mà giao diện tốt nên có:

- Hướng người dùng: đối tượng người dùng phải rõ ràng, giao diện nên được thiết kế có tính đến năng lực, thói quen... của loại đối tượng đó.
- Có khả năng tùy biến cao: giao diện nên có khả năng tùy biến cao để phục vụ cho các cá nhân có cách sử dụng khác nhau, các môi trường hoạt động khác nhau. Các phần mềm trên hệ UNIX với giao diện theo chuẩn X protocol thường được thiết kế có độ tùy biến rất cao.
- Nhất quán: các biểu tượng, thông báo, cách thức nhập dữ liệu phải nhất quán và nên tuân theo các chuẩn thông thường.
- An toàn: nên có chế độ xác nhận lại đối với các thao tác nguy hiểm (như xóa dữ liệu) và nên có khả năng phục hồi trạng thái cũ (undo).
- Dễ học, dễ sử dụng: giao diện luôn cần được thiết kế hướng tới tính dễ học, dễ sử dụng, tức là không đòi hỏi người dùng phải có các năng lực đặc biệt. Ví dụ như không cần nhớ nhiều thao tác, không đòi hỏi phải thao tác nhanh, các thông tin trên màn hình dễ đọc... Một cách tốt nhất để xây dựng giao diện dễ học dễ sử dụng là tuân theo các chuẩn giao diện thông dụng.

Tổng kết

Thiết kế là cái lõi của kỹ nghệ phần mềm. Trong khi thiết kế người ta sẽ phát triển, xét duyệt và làm tư liệu cho việc làm mịn dần các chi tiết thủ tục, cấu trúc chương trình, cấu trúc dữ liệu. Thông qua thiết kế và xét duyệt, chúng ta có thể thẩm định được chất lượng phần mềm.

Tính môđun (trong cả chương trình và dữ liệu) và khái niệm trừu tượng làm cho người thiết kế có khả năng đơn giản hóa và dùng lại các thành phần phần mềm. Việc làm mịn đưa ra một cơ chế để biểu diễn các tầng kế tiếp của chi tiết chức năng. Cấu trúc chương trình và dữ liệu đóng góp cho một quan điểm tổng thể về kiến trúc phần mềm, trong khi thủ tục lại đưa ra những chi tiết cần thiết cho việc cài đặt thuật toán. Che dấu thông tin và độc lập chức năng đưa ra những trực cảm để đạt tới tính môđun có hiệu quả.

Thiết kế phần mềm có thể được xem xét hoặc theo cách nhìn kỹ thuật hoặc theo cách nhìn quản lý dự án. Theo quan điểm kỹ thuật, thiết kế bao gồm 4 hoạt động: thiết kế dữ liệu, thiết kế kiến trúc, thiết kế thủ tục và thiết kế giao diện. Theo quan điểm quản lý, thiết kế tiến hóa từ thiết kế sơ bộ sang thiết kế chi tiết.

Ký pháp thiết kế, đi kèm với các khái niệm lập trình có cấu trúc làm cho người thiết kế biểu diễn được chi tiết thủ tục theo cách thức làm thuận tiện cho việc dịch sang mã chương trình. Chúng ta có thể sử dụng các ký pháp đồ họa, bảng và ngôn ngữ mô tả.

Còn nhiều phương pháp thiết kế phần mềm quan trọng như thiết kế hướng chức năng, hướng đối tượng. Những phương pháp này, được kết hợp với những nền tảng đã trình bày ở trên tạo nên cơ sở cho một cách nhìn đầy đủ về thiết kế phần mềm.

Chương 4

Lập trình

4.1 Ngôn ngữ lập trình

Ngôn ngữ lập trình là phương tiện để liên lạc giữa con người và máy tính. Tiến trình lập trình - sự liên lạc thông qua ngôn ngữ lập trình - là một hoạt động con người. Lập trình là bước cốt lõi trong tiến trình kỹ nghệ phần mềm.

4.1.1 Đặc trưng của ngôn ngữ lập trình

Cách nhìn kỹ nghệ phần mềm về các đặc trưng của ngôn ngữ lập trình tập trung vào nhu cầu xác định dự án phát triển phần mềm riêng. Mặc dầu người ta vẫn cần các yêu cầu riêng cho chương trình gốc, có thể thiết lập được một tập hợp tổng quát những đặc trưng kỹ nghệ:

(1) dễ dịch thiết kế sang chương trình,

(2) có trình biên dịch hiệu quả,

(3) khả chuyển chương trình gốc,

(4) có sẵn công cụ phát triển,

(5) dễ bảo trì.

Bước lập trình bắt đầu sau khi thiết kế chi tiết đã được xác định, xét duyệt và sửa đổi nếu cần. Về lý thuyết, việc sinh chương trình gốc từ một đặc tả chi tiết nên là trực tiếp. Dễ dịch thiết kế sang chương trình đưa ra một chỉ dẫn về việc một ngôn ngữ lập trình phản xạ gần gũi đến mức nào cho một biểu diễn thiết kế. Một ngôn ngữ cài đặt trực tiếp cho các kết cấu có cấu trúc, các cấu trúc dữ liệu phức tạp, vào/ra đặc biệt, khả năng thao tác bit, và các kết cấu hướng sự vật sẽ làm cho việc dịch từ thiết kế sang chương trình gốc dễ hơn nhiều (nếu các thuộc tính này được xác định trong thiết kế).

Mặc dầu những tiến bộ nhanh chóng trong tốc độ xử lý và mật độ nhớ đã bắt đầu làm giảm nhẹ nhu cầu chương trình siêu hiệu quả, nhiều ứng dụng vẫn còn đòi hỏi các chương trình chạy nhanh, gọn (yêu cầu bộ nhớ thấp). Các ngôn ngữ với trình biên dịch tối ưu có thể là hấp dẫn nếu hiệu năng phần mềm là yêu cầu chủ chốt.

Tính khả chuyển chương trình gốc là một đặc trưng ngôn ngữ lập trình có thể được

hiểu theo ba cách khác nhau:

- Chương trình gốc có thể được chuyển từ bộ xử lý này sang bộ xử lý khác và từ trình biên dịch nọ sang trình biên dịch kia với rất ít hoặc không phải sửa đổi gì.

- Chương trình gốc vẫn không thay đổi ngay cả khi môi trường của nó thay đổi (như việc cài đặt bản mới của hệ điều hành).

- Chương trình gốc có thể được tích hợp vào trong các bộ trình phần mềm khác nhau với ít hay không cần thay đổi gì vì các đặc trưng của ngôn ngữ lập trình.

Trong số ba cách hiểu về tính khả chuyển này thì cách thứ nhất là thông dụng nhất. Việc đưa ra các chuẩn (do tổ chức tiêu chuẩn quốc tế IFO hay Viện tiêu chuẩn quốc gia Mĩ - ANSI) góp phần làm nâng cao tính khả chuyển.

Tính sẵn có của công cụ phát triển có thể làm ngắn bớt thời gian cần để sinh ra chương trình gốc và có thể cải thiện chất lượng của chương trình. Nhiều ngôn ngữ lập trình có thể cần tới một loạt công cụ kể cả trình biên dịch gỡ lỗi, trợ giúp định dạng chương trình gốc, các tiện nghi soạn thảo có sẵn, các công cụ kiểm soát chương trình gốc, thư viện chương trình con mở rộng trong nhiều lĩnh vực ứng dụng, các trình duyệt, trình biên dịch chéo cho phát triển bộ vi xử lý, khả năng bộ xử lý macro, công cụ kỹ nghệ ngược và những công cụ khác. Trong thực tế, khái niệm về môi trường phát triển phần mềm tốt (bao hàm cả các công cụ) đã được thừa nhận như nhân tố đóng góp chính cho kỹ nghệ phần mềm thành công.

Tính dễ bảo trì của chương trình gốc có tầm quạn trọng chủ chốt cho tất cả các nỗ lực phát triển phần mềm không tầm thường. Việc bảo trì không thể được tiến hành chừng nào người ta còn chưa hiểu được phần mềm. Các yếu tố của cấu hình phần mềm (như tài liệu thiết kế) đưa ra một nền tảng cho việc hiểu biết, nhưng cuối cùng thì chương trình gốc vẫn phải được đọc và sửa đổi theo những thay đổi trong thiết kế.

Tính dễ dịch thiết kế sang chương trình là một yếu tố quan trọng để dễ bảo trì chương trình gốc. Bên cạnh đó, các đặc trưng tự làm tài liệu của ngôn ngữ (như chiều dài được phép của tên gọi, định dạng nhãn, định nghĩa kiểu, cấu trúc dữ liệu) có ảnh hưởng mạnh đến tính dễ bảo trì.

4.1.2 Lựa chọn ngôn ngữ lập trình

Các đặc trưng của ngôn ngữ lập trình sẽ quyết định miền ứng dụng của ngôn ngữ. Miền ứng dụng là yếu tố chính để chúng ta lựa chọn ngôn ngữ cho một dự án phần mềm.

C thường là một ngôn ngữ hay được chọn cho việc phát triển phần mềm hệ thống. Trong các ứng dụng thời gian thực chúng ta hay gặp các ngôn ngữ như Ada, C, C++ và cả hợp ngữ do tính hiệu quả của chúng. Các ngôn ngữ này và Java cũng được dùng cho phát triển phần mềm nhúng.

Trong lĩnh vực khoa học kỹ thuật thì FORTRAN với khả năng tính toán với độ chính xác cao và thư viện toán học phong phú vẫn còn là ngôn ngữ thống trị. Tuy vậy, PASCAL và C cũng được dùng rộng rãi.

COBOL là ngôn ngữ cho ứng dụng kinh doanh và khai thác CSDL lớn nhưng các ngôn ngữ thế hệ thứ tư đã dần dần chiếm ưu thế.

BASIC vẫn đang tiến hóa (Visual Basic) và được đông đảo người dùng máy tính cá nhân ủng hộ mặc dù ngôn ngữ này rất hiếm khi được những người phát triển hệ thống dùng.

Các ứng dụng trí tuệ nhân tạo thường dùng các ngôn ngữ như LISP, PROLOG hay OPS5, tuy vậy nhiều ngôn ngữ lập trình (vạn năng) khác cũng được dùng.

Xu hướng phát triển phần mềm hướng đối tượng xuyên suốt phần lớn các miền ứng dụng đã mở ra nhiều ngôn ngữ mới và các dị bản ngôn ngữ qui ước. Các ngôn ngữ lập trình hướng đối tượng được dùng rộng rãi nhất là Smalltalk, C++, Java. Ngoài ra còn có Eiffel, Object- PASCAL, Flavos và nhiều ngôn ngữ khác.

Với đặc trưng hướng đối tượng, tính hiệu quả thực hiện cũng như có nhiều công cụ và thư viện, C++ hiện đang được sử dụng rộng rãi trong lĩnh vực phát triển các ứng dụng nghiệp vụ. Java cũng là một ngôn ngữ hướng đối tượng đang được sử dụng rộng rãi cho phát triển các dịch vụ Web và phần mềm nhúng vì các lý do độ an toàn cao, tính trong sáng, tính khả chuyển và hướng thành phần. Theo một số thống kê thì tốc độ phát triển một ứng dụng mới bằng Java cao hơn đến 2 lần so với các ngôn ngữ truyền thống như C hay thậm chí C++.

Các ngôn ngữ biên dịch (script) với những câu lệnh và thư viện mạnh hiện đang rất được chú ý. ASP, JavaScript, PERL... đang được sử dụng rộng rãi trong lập trình Web.

4.1.3 Ngôn ngữ lập trình và và sự ảnh hưởng tới kỹ nghệ phần mềm

Nói chung, chất lượng của thiết kế phần mềm được thiết lập theo cách độc lập với các đặc trưng ngôn ngữ lập trình. Tuy nhiên thuộc tính ngôn ngữ đóng một vai trò trong chất lượng của thiết kế được cài đặt và ảnh hưởng tới cách thiết kế được xác định. Ví dụ như khả năng xây dựng mô đun và bao gói chương trình. Thiết kế dữ liệu cũng có thể bị ảnh hưởng bởi các đặc trưng ngôn ngữ. Các ngôn ngữ lập trình như Ada, C++, Smalltalk đều hỗ trợ cho khái niệm về kiểu dữ liệu trừu tượng - một công cụ quan trọng trong thiết kế và đặc tả dữ liệu. Các ngôn ngữ thông dụng khác, như PASCAL, cho phép định nghĩa các kiểu dữ liệu do người dùng xác định và việc cài đặt trực tiếp danh sách móc nối và những cấu trúc dữ liệu khác. Các tính năng này cung cấp cho người thiết kế phạm vi rộng hơn trong các bước thiết kế sơ bộ và chi tiết.

Các đặc trưng của ngôn ngữ cũng ảnh hưởng tới kiểm thử phần mềm. Các ngôn ngữ trực tiếp hỗ trợ cho các kết cấu có cấu trúc có khuynh hướng giảm bớt độ phức tạp của chương trình, do đó có thể làm cho nó dễ dàng kiểm thử. Các ngôn ngữ hỗ trợ cho việc đặc tả các chương trình con và thủ tục ngoài (như FORTRAN) thường làm cho việc kiểm thử tích hợp ít sinh lỗi hơn.

4.2 Phong cách lập trình

Phong cách lập trình bao hàm một triết lý về lập trình nhấn mạnh tới tính dễ hiểu của chương trình nguồn. Các yếu tố của phong cách bao gồm: tài liệu bên trong chương trình, phương pháp khai báo dữ liệu, cách xây dựng câu lệnh và các kỹ thuật vào/ra.

4.2.1 Tài liệu chương trình

Tài liệu bên trong của chương trình gốc bắt đầu với việc chọn lựa các tên gọi định danh (biến và nhãn), tiếp tục với 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 là điều chủ chốt cho việc hiểu chương trình. Những ngôn ngữ giới hạn độ dài tên biến hay nhãn làm các tên mang nghĩa mơ hồ. 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 hóa việc chuyển đổi từ cú pháp chương trình sang cấu trúc ngữ nghĩa bên trong".

Một điều rõ ràng là: 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 để hiểu trong pha cuối cùng của kỹ nghệ phần mềm - bảo trì.

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 modul. Định dạng cho lời chú thích như thế là:

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

Các chú thích chức năng được nhúng vào bên trong thân của chương trình gốc và được dùng để mô tả cho các khối chương trình.

4.2.2 Khai báo dữ liệu

Thứ tự khai báo dữ liệu nên được chuẩn hóa cho dù ngôn ngữ lập trình không có yêu cầu bắt buộc nào về điều đó.

Các tên biến ngoài việc có nghĩa còn nên mang thông tin về kiểu của chúng. Ví dụ, nên thống nhất các tên biến cho kiểu số nguyên, kiểu số thực... Cần phải chú giải về mục đích đối với các biến quan trọng, đặc biệt là các biến tổng thể.

Các cấu trúc dữ liệu nên được chú giải đầy đủ về cấu trúc và chức năng, và các đặc thù về sử dụng. Đặc biệt là đối với các cấu trúc phức tạp như danh sách móc nối trong C hay Pascal.

4.2.3 Xây dựng câu lệnh

Việc xây dựng luồng logic phần mềm được thiết lập trong khi thiết kế. Việc xây dựng từng câu lệnh tuy nhiên lại là một phần của bước lập trình. Việc xây dựng câu lệnh nên tuân theo một qui tắc quan trọng hơn cả: mỗi câu lệnh nên đơn giản và trực tiếp.

Nhiều ngôn ngữ lập trình cho phép nhiều câu lệnh trên một dòng. Khía cạnh tiết kiệm không gian của tính năng này khó mà biện minh bởi tính khó đọc nảy sinh. Cấu trúc chu trình và các phép toán điều kiện được chứa trong đoạn trên đều bị che lấp bởi cách xây dựng nhiều câu lệnh trên một dòng.

Cách xây dựng câu lệnh đơn và việc tụt lề minh họa cho các đặc trưng logic và chức năng của đoạn này. Các câu lệnh chương trình gốc riêng lẻ có thể được đơn giản hóa bởi:

- tránh dùng các phép kiểm tra điều kiện phức tạp
- khử bỏ các phép kiểm tra điều kiện phủ định
- tránh lồng nhau nhiều giữa các điều kiện hay chu trình
- dùng dấu ngoặc để làm sáng tỏ các biểu thức logic hay số học
- dùng dấu cách và/hoặc các ký hiệu dễ đọc để làm sáng tỏ nội dung câu lệnh
- chỉ dùng các tính năng chuẩn của ngôn ngữ

Để hướng tới chương trình dễ hiểu luôn nên đặt ra câu hỏi: Liệu có thể hiểu được điều này nếu ta không là người lập trình cho nó không?

4.2.4 Vào/ra

Vào ra của các mô đun nên tuân thủ theo một số hướng dẫn sau:

- Làm hợp lệ mọi cái vào.
- Kiểm tra sự tin cậy của các tổ hợp khoản mục vào quan trọng.
- Giữ cho định dạng cái vào đơn giản.
- Dùng các chỉ báo cuối dữ liệu thay vì yêu cầu người dùng xác định "số các khoản mục".
- Giữ cho định dạng cái vào thống nhất khi một ngôn ngữ lập trình có các yêu cầu định dạng nghiêm ngặt.

4.3 Lập trình tránh lỗi

Tránh lỗi và phát triển phần mềm vô lỗi dựa trên các yếu tố sau:

i) Sản phẩm của một đặc tả hệ thống chính xác.

ii) Chấp nhận một cách tiếp cận thiết kế phần mềm dựa trên việc bao gói dữ liệu và che dấu thông tin.

iii) Tăng cường duyệt lại trong quá trình phát triển và thẩm định hệ thống phần mềm.

iv) Chấp nhận triết lý chất lượng tổ chức: chất lượng là bánh lái của quá trình phần mềm.

v) Việc lập kế hoạch cẩn thận cho việc thử nghiệm hệ thống để tìm ra các lỗi chưa được phát hiện trong quá trình duyệt lại và để định lượng độ tin cậy của hệ thống.

Có hai cách tiếp cận chính hỗ trợ tránh lỗi là:

Lập trình có cấu trúc:

Thuật ngữ này được đặt ra từ cuối những năm 60 và có nghĩa là lập trình mà không dùng lệnh goto, lập trình chỉ dùng các vòng lặp while và các phát biểu if để xây dựng điều khiển và trong thiết kế thì dùng cách tiếp cận trên - xuống. Việc thừa nhận lập trình có cấu trúc là quan trọng bởi vì nó là bước đầu tiên bước từ cách tiếp cận không khuôn phép tới phát triển phần mềm.

Lập trình có cấu trúc buộc người lập trình phải nghĩ cẩn thận về chương trình của họ, và vì vậy nó ít tạo ra sai lầm trong khi phát triển. Lập trình có cấu trúc làm cho chương trình có thể được đọc một cách tuần tự và do đó dễ hiểu và dễ kiểm tra. Tuy nhiên nó chỉ là bước đầu tiên trong việc lập trình nhằm đạt độ tin cậy tốt.

Có một vài khái niệm khác cũng hay dẫn tới các lỗi phần mềm:

i) Các số thực dấu chấm động: các phép toán số thực được làm tròn khiến cho việc so sánh các kết quả số thực, nhất là so sánh bằng giữa hai số thực là không khả thi. Số thực dấu phẩy động có độ chính xác khác nhau khiến cho kết quả phép tính không theo mong muốn. Ví dụ, trong phép tính tính phân chúng ta cần cộng các giá trị nhỏ trước với nhau nếu không chúng sẽ bị làm tròn.

ii) Các con trỏ và bộ nhớ động: con trỏ là các cấu trúc bậc thấp khó quản lý và dễ gây ra các lỗi nghiệm trọng đối với hệ thống. Việc cấp phát và thu hồi bộ nhớ động phức tạp và là một trong các nguyên nhân chính gây lỗi phần mềm.

iii) Song song: lập trình song song đòi hỏi kỹ thuật cao và hiểu biết sâu sắc về hệ thống. Một trong các vấn đề phức tạp của song song là quản lý tương tranh.

iv) Đệ quy.

v) Các ngắt.

Các cấu trúc này có ích, nhưng người lập trình nên dùng chúng một cách cẩn thận.

Phân quyền truy cập dữ liệu:

Nguyên lý an ninh trong quân đội là các cá nhân chỉ được biết các thông tin có liên quan trực tiếp đến nhiện vụ của họ. Khi lập trình người ta cũng tuân theo một nguyên lý tương tự cho việc truy cập dữ liệu hệ thống. Mỗi thành phần chương trình chỉ được

phép truy cập đến dữ liệu nào cần thiết để thực hiện chức năng của nó. Ưu điểm của việc che dấu thông tin là các thông tin bị che dấu không thể bị sập đổ (thao tác trái phép) bởi các thành phần chương trình mà được xem rằng không dùng thông tin đó. Tiến hóa của sự phân quyền truy cập là che dấu thông tin, hay nói chính xác hơn là che dấu cấu trúc thông tin. Khi đó, chúng ta có thể thay đổi cấu trúc thông tin mà không phải thay đổi các thành phần khác có sử dụng thông tin đó.

4.3.1 Lập trình thứ lỗi

Đối với các hệ thống đòi hỏi độ tin cậy rất cao như hệ thống điều khiển bay thì cần phải có khả năng *dung thứ lỗi (fault tolerance)*, tức là khả năng đảm bảo cho hệ thống vẫn hoạt động chính xác ngay cả khi có thành phần sinh lỗi.

Có bốn hoạt động cần phải tiến hành nếu hệ thống là thứ lỗi:

i) Phát hiện lỗi.

ii) Định ra mức độ thiệt hại.

iii) Hồi phục sau khi gặp lỗi: Hệ thống phải hồi phục về trạng thái mà nó biết là an toàn. Cũng có thể là chỉnh lý trạng thái bị hủy hoại (hồi phục tiến), cũng có thể là lui về một trạng thái trước mà an toàn (hồi phục lùi).

vi) Chữa lỗi: Cải tiến hệ thống để cho lỗi đó không xuất hiện nữa. Tuy nhiên trong nhiều trường hợp phát hiện được đúng nguyên nhân gây lỗi là rất khó khăn vì nó xẩy ra bởi một tổ hợp của thông tin vào và trạng thái của hệ thống.

Thông thường, thứ lỗi được thực hiện bằng cách song song hóa các chức năng, kết hợp với bộ điều khiển thứ lỗi. Bộ điều khiển sẽ so sánh kết quả của các khối chương trình thực hiện cùng nhiệm vụ và sử dụng nguyên tắc đa số để chọn kết quả.

4.3.2 Lập trình phòng thủ

Lập trình phòng thủ là cách phát triển chương trình mà người lập trình giả định rằng các mâu thuẫn hoặc các lỗi chưa được phát hiện có thể tồn tại trong chương trình. Phải có phần mềm kiểm tra trạng thái hệ thống sau khi biến đổi và phải đảm bảo rằng sự biến đổi trạng thái là kiên định. Nếu phát hiện một mâu thuẫn thì việc biến đổi trạng thái là phải rút lại và trạng thái phải trở về trạng thái đúng đắn trước đó.

Nói chung một lỗi gây ra một sự sụp đổ trạng thái: các biến trạng thái được gán các trị không hợp luật. Ngôn ngữ lập trình như Ada cho phép phát hiện ra các lỗi đó ngay trong thời gian biên dịch. Tuy nhiên việc kiểm tra biên dịch chỉ hạn chế cho các giá trị tĩnh và một vài phép kiểm tra thời gian thực là không thể tránh được. Một cách để phát hiện lỗi trong chương trình Ada là dùng cơ chế xử lý bất thường kết hợp với đặc tả miền trị.

Hồi phục lỗi là một quá trình cải biên không gian trạng thái của hệ thống sao cho hiệu ứng của lỗi là nhỏ nhất và hệ thống có thể tiếp tục vận hành, có lẽ là trong một mức suy giảm. Hồi phục tiến liên quan đến việc cố gắng chỉnh lại trạng thái hệ thống. Hồi phục lùi liên quan đến việc lưu trạng thái của hệ thống ở một trạng thái đúng đã

biết.

Hồi phục tiến thường là một chuyên biệt ứng dụng. Có hai tình thế chung mà khi đó hồi phục tiến có thể thành công:

1) Khi dữ liệu mã bị sụp đổ: Việc sử dụng kỹ thuật mã hóa thích hợp bằng cách thêm các dữ liệu dư thừa vào dữ liệu cho phép sửa sai khi phát hiện lỗi.

2) Khi cấu trúc nối bị sụp đổ: Nếu các con trỏ tiến và lùi đã có trong cấu trúc dữ liệu thì cấu trúc đó có thể tái tạo nếu như còn đủ các con trỏ chưa bị sụp. Kỹ thuật này thường được dùng cho việc sửa chữa hệ thống tệp và cơ sở dữ liệu.

Hồi phục lùi là một kỹ thuật đơn giản liên quan đến việc duy trì các chi tiết của trạng thái an toàn và cất giữ trạng thái đó khi mà sai lầm đã bị phát hiện. Hầu hết các hệ quản trị cơ sở dữ liệu đều có bộ hồi phục lỗi. CSDL chỉ cập nhật dữ liệu một khi giao dịch đã hoàn tất và không phát hiện được vấn đề gì. Nếu giao dịch thất bại thì CSDL không được cập nhật.

Một kỹ thuật khác là thiết lập các điểm kiểm tra thường kỳ mà chúng là các bản sao của trạng thái hệ thống. Khi mà một lỗi được phát hiện thì trạng thái an toàn đó được tái lưu kho từ điểm kiểm tra gần nhất.

Trường hợp hệ thống dính líu tới nhiều quá trình hợp tác thì dãy các giao tiếp có thể là các điểm kiểm tra của các quá trình đó không đồng bộ và để hồi phục thì mỗi quá trình phải trở lại trạng thái ban đầu của nó.

4.4 Lập trình hướng hiệu quả thực hiện

4.4.1 Tính hiệu quả chương trình

Tính hiệu quả của chương trình gốc có liên hệ trực tiếp với tính hiệu quả của thuật toán được xác định trong thiết kế chi tiết. Tuy nhiên, phong cách lập trình có thể có một tác động đến tốc độ thực hiện và yêu cầu bộ nhớ. Tập hợp các hướng dẫn sau đây bao giờ cũng có thể áp dụng được khi thiết kế chi tiết được dịch thành chương trình:

- Đơn giản hóa các biểu thức số học và lôgic trước khi đi vào lập trình.

- Tính cẩn thận từng chu kỳ lồng nhau để xác định liệu các câu lệnh hay biểu thức có thể được chuyển ra ngoài hay không

- Khi có thể, hãy tránh dùng mảng nhiều chiều

- Khi có thể hãy tránh việc dùng con trỏ và danh sách phức tạp

- Dùng các phép toán số học "nhanh"

- Không trộn lẫn các kiểu dữ liệu, cho dù ngôn ngữ có cho phép điều đó

- Dùng các biểu thức số học và logic bất kì khi nào có thể được

Nhiều trình biên dịch có tính năng tối ưu tự động sinh ra chương trình hiệu quả bằng cách dồn nén các biểu thức lặp, thực hiện tính chu trình, dùng số học nhanh và áp dụng các thuật toán có hiệu quả liên quan khác. Với những ứng dụng trong đó tính hiệu quả có ý nghĩa quan trọng, những trình biên dịch như thế là công cụ lập trình không thể thiếu được.

4.4.2 Hiệu quả bộ nhớ

Tính hiệu quả bộ nhớ phải được tính vào đặc trưng *phân trang* của hệ điều hành. Nói chung, tính cục bộ của chương trình hay việc bảo trì lĩnh vực chức năng qua các kết cấu có cấu trúc là một phương pháp tuyệt vời làm giảm việc phân trang và do đó làm tăng tính hiệu quả.

Hạn chế bộ nhớ trong phát triển phần mềm nhúng là mối quan tâm rất thực tế, mặc dầu bộ nhớ giá thấp, mật độ cao vẫn đang tiến hóa nhanh chóng. Nếu yêu cầu hệ thống cần tới bộ nhớ tối thiểu (như sản phẩm giá thấp, khối lượng lớn) thì trình biên dịch ngôn ngữ cấp cao phải được trù tính cẩn thận với tính năng nén bộ nhớ, hay như một phương kế cuối cùng, có thể phải dùng tới hợp ngữ.

4.4.3 Hiệu quả vào/ra

Các thiết bị vào ra thường có tốc độ chậm hơn rất nhiều so với khả năng tính toán của máy tính và tốc độ truy cập bộ nhớ trong. Việc tối ưu vào ra có thể làm tăng đáng kể tốc độ thực hiện. Một số hướng dẫn đơn giản để tăng cường hiệu quả vào/ra:

- Số các yêu cầu vào/ra nên giữ mức tối thiểu
- Mọi việc vào/ra nên qua bộ đệm để làm giảm phí tổn liên lạc.
- Với bộ nhớ phụ (như đĩa) nên lựa chọn và dùng phương pháp thâm nhập đơn giản nhất chấp nhận được.
- Nên xếp khối vào/ra với các thiết bị bộ nhớ phụ.
- Việc vào/ra với thiết bị cuối hay máy in nên nhận diện các tính năng của thiết bị có thể cải tiến chất lượng hay tốc độ.

Tổng kết

Bước lập trình là một tiến trình dịch (chuyển hóa) thiết kế chi tiết thành chương trình mà cuối cùng được biến đổi thành các lệnh mã máy thực hiện được.

Các đặc trưng của ngôn ngữ lập trình có ảnh hưởng lớn đến quá trình xây dựng, kiểm thử cũng như bảo trì phần mềm.

Phong cách lập trình quyết định tính dễ hiểu của chương trình gốc. Các yếu tố của phong cách bao gồm việc làm tài liệu bên trong, phương pháp khai báo dữ liệu, thủ tục xây dựng câu lệnh, và kỹ thuật lập trình vào/ra.

Lập trình cần hướng tới hiệu quả thực hiện, tức là tích kiệm tài nguyên phần cứng (mức độ sử dụng CPU, bộ nhớ...). Mặc dầu tính hiệu quả có thể là yêu cầu cực kì quan trọng, chúng ta nên nhớ rằng một chương trình hoạt động hiệu quả mà lại không dễ hiểu dẫn đến khó bảo trì thì giá trị của nó cũng bị hạn chế.

Chương 5

Xác minh và thẩm định

5.1 Đại cương

Xác minh và thẩm định là sự kiểm tra việc phát triển phần mềm. Là công việc xuyên suốt quá trình phát triển phần mềm, chứ không chỉ ở khâu kiểm thử khi đã có mã nguồn.

Xác minh (verification) là sự kiểm tra xem sản phẩm có đúng với đặc tả không, chú trọng vào việc phát hiện lỗi lập trình.

Thẩm định (validation) là sự kiểm tra xem sản phẩm có đáp ứng được nhu cầu người dùng không, có hoạt động hiệu quả không, tức là chú trọng vào việc phát hiện lỗi phân tích, lỗi thiết kế.

Tóm lại, mục đích của thẩm định và xác minh là

- phát hiện và sửa lỗi phần mềm
- đánh giá tính dùng được của phần mềm

Có hai khái niệm là thẩm định/xác minh tĩnh và thẩm định/xác minh động.

Thẩm định và xác minh tĩnh là sự kiểm tra mà không thực hiện chương trình, ví dụ như xét duyệt thiết kế, xét duyệt yêu cầu, nghiên cứu mã nguồn, sử dụng các biến đổi hình thức (suy luận) để kiểm tra tính đúng đắn của chương trình. Thẩm định và xác minh tĩnh được tiến hành ở mọi khâu trong vòng đời phần mềm. Về lý thuyết, có thể phát hiện được hầu hết các lỗi lập trình, tuy nhiên phương pháp này không thể đánh giá được tính hiệu quả của chương trình.

Thẩm định và xác minh động là sự kiểm tra thông qua việc thực hiện chương trình, được tiến hành sau khi đã phát triển chương trình (mã nguồn). Hiện vẫn là kỹ thuật kiểm tra chính.

Cả hai hướng nêu trên đều rất quan trọng và chúng bổ khuyết lẫn nhau. Trong chương này, chúng ta đi sâu vào tìm hiểu về thẩm định và xác minh động, gọi là sự thử nghiệm (kiểm thử) chương trình.

Có hai loại thử nghiệm (động) là:

- thử nghiệm (tìm) khuyết tật: được thiết kế để phát hiện khuyết tật của hệ thống (đặc biệt là lỗi lập trình).

t thử nghiệm thống kê: sử dụng các dữ liệu (thao tác) phổ biến của người dùng (dựa trên sự thống kê) để đánh giá tính dùng được của hệ thống.

Thử nghiệm cần phải có

t tính lặp lại: thử nghiệm phải lặp lại được để phát hiện thêm lỗi và kiểm tra xem lỗi đã được sửa hay chưa. Bất cứ khi nào sửa mã chương trình chúng ta phải thử nghiệm lại (kể cả đối với các thử nghiệm đã thành công).

t tính hệ thống: việc thử nghiệm phải tiến hành một cách có hệ thống để đảm bảo kiểm thử được mọi trường hợp, nếu tiến hành thử nghiệm một cách ngẫu nhiên thì không đảm bảo được điều này.

t được lập tài liệu: để kiểm soát xem cái nào đã được thực hiện, kết quả như thế nào...

5.2 Khái niệm về phép thử

Một phép thử được gọi là thành công nếu nó phát hiện ra khiếm khuyết của phần mềm. Chú ý là phép thử chỉ chứng minh được sự tồn tại của lỗi trong hệ thống chứ không chứng minh được hệ thống không có lỗi. Một phép thử (ca thử nghiệm) bao gồm

- tên của mô đun thử nghiệm
- dữ liệu vào
- dữ liệu ra mong muốn (đúng)
- dữ liệu ra thực tế (khi đã tiến hành thử nghiệm)

Các ca thử nghiệm nên được thiết kế khi chúng ta tạo các tài liệu phân tích và thiết kế, chứ không phải là khi đã viết xong mã nguồn.

5.3 Thử nghiệm chức năng và thử nghiệm cấu trúc

Có hai kỹ thuật thử nghiệm tìm khuyết tật chính là thử nghiệm chức năng và thử nghiệm cấu trúc.

5.3.1 Thử nghiệm chức năng

Thử ngiệm chức năng (functional testing) còn gọi là thử nghiệm hộp đen (black box testing) là sự thử nghiệm sử dụng các ca thử nghiệm được thiết kế dựa trên đặc tả yêu cầu, tài liệu người dùng nhằm mục đích phát hiện ra các khiếm khuyết. Thử nghiệm chức năng nhìn nhận mô đun được thử nghiệm như là một hộp đen, và chỉ quan tâm đến chức năng (hành vi) của mô đun, tức là kiểm tra xem có hoạt động đúng với đặc tả hay không.

Các ca kiểm thử bao gồm các trường hợp bình thường và không bình thường (dữ liệu không hợp lệ...) của mô đun.

Thông thường, không thể thử nghiệm với mọi dữ liệu, chiến lược chung khi thiết kế dữ liệu thử nghiệm là *phân hoạch (dữ liệu) tương đương*. Phân hoạch tương đương chia miền dữ liệu vào ra thành các vùng, mà mỗi vùng chứa các dữ liệu có cùng hành vi. Do đó, đối với mỗi vùng dữ liệu chỉ cần xây dựng một ca thử nghiệm. Thêm vào đó là các ca sử dụng đối với biên giới của các vùng. Theo kinh nghiệm, các sai sót về lập trình thường sảy ra đối với các dữ liệu biên.

Ví dụ, đối với hàm tính trị tuyệt đối của số nguyên, có thể chia miền đối số thành 2 vùng:

- vùng dữ liệu = 0

- vùng dữ liệu ¥ 0

Do đó các dữ liệu đầu vào để kiếm thử có thể là 100, -20, và 0.

Ngoài các ca thử nghiệm trên, thông thường còn cần kiểm tra với các dữ liệu đặc thù như:

- biên của số trong máy tính (ví dụ -32768, 32767)

- 0, số âm, số thập phân

- không có input

- input ngẫu nhiên

- input sai kiểu...

Thử nghiệm chức năng có thể giúp chúng ta

- phát hiện sự thiếu sót chức năng

- phát hiện khiếm khuyết

- sai sót về giao diện giữa các mô đun

- sự không hiệu quả của chương trình

- lỗi khởi tạo, lỗi kết thúc

Tuy nhiên thử nghiệm chức năng chỉ dựa trên đặc tả nên không thể kiểm thử được các trường hợp không được khai báo trong đặc tả, không đảm bảo thử hết được các khối mã nguồn của mô đun.

Thử nghiệm chức năng cũng không phát hiện được các đoạn mã yếu (có khả năng sinh lỗi với một trạng thái đặc biệt nào đó của hệ thống), và trong nhiều trường hợp việc đảm bảo xây dựng đầy đủ các ca thử nghiệm là khó khăn.

Ví dụ, hàm tính trị tuyệt đối sau có thể thoát được thử nghiệm chức năng tuy có lỗi.

```
E"? „br»E"? "d
z
    Eq »"“Kd °#?y°" "B
    #fir# »"¥Kd °#?y°" t"B
§
```

5.3.2 Thử nghiệm cấu trúc

Thử nghiệm cấu trúc (structural testing) là sự thử nghiệm dựa trên phân tích chương trình. Kỹ thuật chính ở đây là xác định đường đi (path) của chương trình (điều khiển) từ input đến output. Mục đích của thử nghiệm cấu trúc là kiểm tra tất cả các đường đi có thể. Tức là đảm bảo mọi lệnh đều được thực hiện ít nhất một lần trong một ca thử nghiệm nào đó. Thử nghiệm cấu trúc chú trọng vào phân tích các cấu trúc rẽ nhánh và các vòng lặp.

Ví dụ:

```
E"? Y„P»E"? P8 E"? fl8 E"? Zd
z
    Eq »P"fld z
            Eq »P"Zd °#?y°" PB
            #fir# °#?y°" ZB
    §
     #fir# z
              Eq »fl " Zd °#?y°" flB
              #fir# °#?y°" ZB
     §
§
```

Trong ví dụ trên có 4 đường đi có thể do đó cần ít nhất 4 ca thử nghiệm để thử nghiệm tất cả các đường đi này.

Thử nghiệm cấu trúc xem xét chương trình ở mức độ chi tiết và phù hợp khi kiểm tra các mô đun nhỏ. Tuy nhiên thử nghiệm cấu trúc có thể không đầy đủ vì kiểm thử hết các lệnh không chứng tỏ là chúng ta đã kiểm thử hết các trường hợp có thể. Có khả năng tồn tại các tổ hợp lệnh khác nhau gây lỗi. Ngoài ra, chúng ta không thể kiểm thử hết các đường đi đối với các vòng lặp lớn.

Tóm lại, thử nghiệm chức năng và thử nghiệm cấu trúc đều rất quan trọng và chúng bổ khuyết lẫn nhau.

5.4 Quá trình thử nghiệm

Quá trình thử nghiệm có thể chia làm các giai đoạn như sau:

- t thử nghiệm đơn vị: là bước thử nghiệm đối với từng chức năng (hàm) nhằm mục đích chính là phát hiện lỗi lập trình, thường sử dụng nhiều thử nghiệm cấu trúc.
- t thử nghiệm mô đun: thử nghiệm mô đun (liên kết một số hàm)
- t thử nghiệm hệ con: nếu hệ thống bao gồm một số hệ con độc lập thì đây là bước tiến hành thử nghiệm với từng hệ con riêng biệt

- thử nghiệm hệ thống (tích hợp): thử nghiệm sự hoạt động tổng thể hệ thống, kiểm tra tính đúng đắn của giao diện, tính đúng đắn với đặc tả, và tính dùng được. Chủ yếu sử dụng thử nghiệm chức năng.
- thử nghiệm nghiệm thu (alpha): thử nghiệm được tiến hành bởi một nhóm nhỏ người sử dụng dưới sự hướng dẫn của người phát triển, sử dụng các dữ liệu thực, thẩm định tính dùng được của hệ thống.
- thử nghiệm beta: là mở rộng của thử nghiệm alpha, được tiến hành với một số lớn người sử dụng không có sự hướng dẫn của người phát triển, kiểm tra tính ổn định, điểm tốt và không tốt của hệ thống.

Các bước thử nghiệm ban đầu nặng về kiểm tra lỗi lập trình (xác minh), các bước thử nghiệm cuối thiên về kiểm tra tính dùng được của hệ thống (thẩm định).

Ngoài ra còn một bước hay một khái niệm thử nghiệm khác được gọi là thử nghiệm quay lui. Thử nghiệm quay lui được tiến hành mỗi khi chúng ta sửa mã chương trình:

- khi sửa lỗi

- khi nâng cấp chương trình

5.4.1 Thử nghiệm gây áp lực

Đối với một số hệ thống quan trọng, người ta còn tiến hành thử nghiệm gây áp lực (stress testing). Đây là loại (bước) thử nghiệm được tiến hành khi đã có phiên bản làm việc, nhằm tìm hiểu hoạt động của hệ thống trong các trường hợp tải trọng lớn (dữ liệu lớn, số người sử dụng lớn, tài nguyên hạn chế...). Mục đích của thử nghiệm áp lực là

- tìm hiểu giới hạn chịu tải của hệ thống

- tìm hiểu về đặc trưng của hệ thống khi đạt và vượt giới hạn chịu tải (khi bị sụp đổ)

Ngoài ra thử nghiệm áp lực còn nhằm xác định các trạng thái đặc biệt như tổ hợp một số điều kiện dẫn đến sự sụp đổ của hệ thống; tính an toàn của dữ liệu, của dịch vụ khi hệ thống sụp đổ.

5.5 Chiến lược thử nghiệm

Khi thử nghiệm hệ con và thử nghiệm hệ thống (tích hợp), có các chiến lược thử nghiệm chính là thử nghiệm dưới lên (bottom-up testing) và thử nghiệm trên xuống (top-down testing).

5.5.1 Thử nghiệm dưới lên

Thử nghiệm dưới lên tiến hành thử nghiệm với các mô đun ở mức độ thấp trước. Mô đun thượng cấp (mô đun gọi) được thay thế bằng chương trình kiểm thử có nhiện vụ

đọc dữ liệu kiểm thử, gọi mô đun cần kiểm thử và kiểm tra kết quả.

Nhược điểm của thử nghiệm dưới lên là

- phát hiện chậm các lỗi thiết kế

- chậm có phiên bản thực hiện được của hệ thống

5.5.2 Thử ngiệm trên xuống

Thử nghiệm trên xuống tiến hành thử nghiệm với các mô đun ở mức cao trước, các mô đun mức thấp được tạm thời phát triển với các chức năng hạn chế, có giao diện giống như trong đặc tả. Mô đun mức thấp có thể chỉ đơn giản là trả lại kết quả với một vài đầu vào định trước.

Thử nghiệm trên xuống có ưu điểm là

- phát hiện sớm các lỗi thiết kế, do đó có thể thiết kế, cài đặt lại với giá rẻ

- có phiên bản hoạt động sớm (với tính năng hạn chế) do đó có thể sớm tiến hành thẩm định

Nhược điểm của kiểm thử trên xuống là các chức năng của mô đun cấp thấp nhiều khi rất phức tạp do đó khó có thể mô phỏng được, dẫn đến không kiểm thử đầy đủ chức năng hoặc phải đình chỉ kiểm thử cho đến khi các mô đun cấp thấp xây dựng xong.

Chương 6

Quản lý dự án phát triển phần mềm

6.1 Đại cương

Quản lý dự án là tầng đầu tiên trong phát triển phần mềm. Chúng ta gọi là tầng quản lý vì nó là bước kỹ thuật cơ sở kéo dài suốt vòng đời phần mềm. Mục tiêu của việc quản lý dự án phát triển phần mềm là đảm bảo cho dự án

- đúng thời hạn
- không vượt dự toán
- đầy đủ các chức năng đã định
- thỏa mãn yêu cầu của khách hàng (tạo ra sản phẩm tốt)

Quản lý dự án bao gồm các pha công việc sau

- thiết lập: viết đề án
- ước lượng chi phí
- phân tích rủi ro
- lập kế hoạch
- chọn người
- theo dõi và kiểm soát dự án
- viết báo cáo và trình diễn sản phẩm

Tiến hành quản lý dự án là *người quản lý dự án*, có các nhiệm vụ và quyền hạn như sau

- Thời gian
  - tạo lập kế hoạch, điều chỉnh kế hoạch
  - kiểm tra/đối chiếu các tiến trình con với kế hoạch
  - giữ một độ mềm dẻo nhất định trong kế hoạch
  - phối thuộc các tiến trình con
- Tài nguyên: thêm tiền, thêm thiết bị, thêm người...
- Sản phẩm: thêm bớt chức năng của sản phẩm...
- Rủi ro: phân tích và tìm phương pháp xử lý, chấp nhận một số rủi ro

Ngoài ra, người quản lý dự án còn cần phải quan tâm đến sự phối thuộc với các dự án khác và thông tin cho người quản lý cấp trên... Phương pháp tiếp cận của người quản lý dự án

t hiểu rõ mục tiêu (tìm cách định lượng các mục tiêu bất cứ khi nào có thể)

t hiểu rõ các ràng buộc (chi phí, lịch biểu, tính năng...)

t lập kế hoạch để đạt dược mục tiêu trong các ràng buộc

t giám sát và điều chỉnh kế hoạch

t tạo môi trường làm việc ổn định, năng động cho nhóm

Quản lý tồi sẽ dẫn đến sự chậm trễ của dự án, tính năng yếu kém và tăng chi phí phát triển. Một ví dụ kinh điển về quản lý tồi là dự án hệ điều hành OS360 của IBM bị chậm 2 năm so với kế hoạch.

6.2 Độ đo phần mềm

Để quản lý chúng ta cần định lượng được đối tượng quản lý cần quản lý, ở đây là phần mềm và qui trình phát triển. Chúng ta cần đo kích cỡ phần mềm, chất lượng phần mềm, năng suất phần mềm...

6.2.1 Đo kích cỡ phần mềm

Có hai phương pháp phổ biến để đo kích cỡ phần mềm là đo số dòng lệnh (LOC - Lines Of Code) và đo điểm chức năng (FP - Function Points). Độ đo LOC tương đối trực quan, tuy nhiên phụ thuộc rất nhiều vào ngôn ngữ lập trình cụ thể. Từ kích cỡ của phần mềm (LOC), chúng ta có thể tính một số giá trị như

- Hiệu năng = KLOC/người-tháng

- Chất lượng = số khiếm khuyết/KLOC

- Chi phí = giá thành/KLOC

Các thông số của các dự án đã phát triển trong quá khứ sẽ được dùng dể phục vụ cho ước lượng cho các phần mềm sẽ phát triển.

Điểm chức năng FP được tính dựa trên đặc tả yêu cầu và độc lập với ngôn ngữ phát triển. Tuy nhiên nó lại có sự phụ thuộc vào các tham số được thiết lập dựa trên kinh nghiệm. Mô hình cơ sở của tính điểm chức năng là

b£ L „ƒ`l „QUl „-†l „[ul „…b8

trong đó

- I : số Input

- O: số Output

- E: số yêu cầu

- L: số tệp truy cập

- F: số giao diện ngoại lai (devices, systems)

6.2.2 Độ đo dựa trên thống kê

Người ta còn thiết lập một số độ đo phần mềm dựa trên thống kê như sau:

- Độ tin cậy MTBF - Mean Time Between Failure: thời gian chạy liên tục của hệ thống

- Thời gian khôi phục hệ thống MTTR - Mean Time To Repair

- Tính sẵn có : ...Tb/»: ...Tb l : .....‡d

6.3 Ước lượng

Công việc đầu tiên của người quản lý dự án là ước lượng - ước lượng về kích cỡ, chi phí, thời gian tiến hành dự án. Việc này thông thường được tiến hành bằng cách phân rã phần mềm cần phát triển thành các khối nhỏ và áp dụng các kinh nghiệm (các thông số như kích cỡ, chi phí, năng lực nhân viên...) đối với các phần mềm đã phát triển để ước lượng, đánh giá công việc.

Một mô hình ước lượng hay được dùng là mô hình COCOMO - Constructive Cost Model ước lượng chi phí từ số dòng lệnh. Dùng mô hình này ta sẽ có thể ước lượng các thông số sau:

- Nỗ lực phát triển † L „u^{b}

- Thời gian phát triển ... L ´†^{i}

- Số người tham gia w L †/...

Trong đó „8b8´8i là các tham số tùy thuộc vào từng loại dự án (xem bảng 6.1). Điểm đáng chú ý ở đây là từ nỗ lực phát triển chúng ta suy ra thời gian và số người tham gia vào dự án.

Bảng 6.1: COCOMO - Các tham số cơ sở

	a	b	c	d
organic	3.2	1.05	2.5	0.38
semi-detached	3.0	1.12	2.5	0.35
embeded	2.8	1.2	2.5	0.32

Các bước tiến hành của COCOMO như sau:

- thiết lập kiểu dự án (organic: đơn giản, semi-detached: trung bình, embeded: phức tạp)

- xác lập các mô đun và ước lượng dòng lệnh

- tính lại số dòng lệnh trên cơ sở tái sử dụng

- tính nỗ lực phát triển E cho từng mô đun

- tính lại E dựa trên độ khó của dự án (mức độ tin cậy, kích cỡ CSDL, yêu cầu về tốc độ, bộ nhớ,...)

- Tính thời gian và số người tham gia

Ví dụ: Phần mềm với 33.3 nghìn dòng lệnh và các tham số „8b8´8i lần lượt là 3.0, 1.12, 2.5, 0.35, ta tính được:

† L –AK) ––A^fAfQ L ƒ..Q người-tháng

... L QA..) † ^KA-... L ƒ [A. tháng

w L † / [7 ƒƒ người

Cần nhớ rằng đo phần mềm là công việc rất khó khăn do

t hầu hết các thông số đều không đo được một cách trực quan

t rất khó thẩm định được các thông số

t không có mô hình tổng quát

t các kỹ thuật đo còn đang thay đổi

Chúng ta không thể kiểm soát được quá trình sản xuất phần mềm nếu không ước lượng (đo) nó. Một mô hình ước lượng nghèo nàn vẫn hơn là không có mô hình nào và phải liên tục ước lượng lại khi dự án tiến triển.

6.4 Quản lý nhân sự

Chi phí (trả công) con người là phần chính của chi phí xây dựng phần mềm. Ngoài ra, năng lực của người phát triển phần mềm lại rất biến thiên, kéo theo sự phức tạp trong tính toán chi phí.

Phát triển phần mềm được tiến hành theo nhóm. Kích thước tốt của nhóm là từ 3 đến 8 người. Phần mềm lớn thường được xây dựng bởi nhiều nhóm nhỏ. Một nhóm phát triển có thể gồm các loại thành viên sau:

t người phát triển

t chuyên gia về miền ứng dụng

t người thiết kế giao diện

t thủ thư phần mềm (quản lý cấu hình phần mềm)

t người kiểm thử

Một nhóm phát triển cần có người quản lý, và người có vai trò lãnh đạo về mặt kĩ thuật.

Một đặc trưng của làm việc theo nhóm là sự trao đổi thông tin (giao tiếp) giữa các thành viên trong nhóm. Thời gian dùng cho việc giao tiếp có thể chiếm đến nửa tổng thời gian dành cho pháp triển phần mềm. Ngoài ra, thời gian không dùng cho phát triển sản phẩm cũng chiếm một phần lớn thời gian còn lại của người lập trình.

Một người có thể đồng thời làm việc cho nhiều nhóm (dự án) phần mềm khác nhau. Điều này làm cho việc tính toán giá thành phần mềm phức tạp.

Cần ghi nhớ, trong sản xuất phần mềm thì

- Năng lực của các thành viên là không đồng đều

- Người tốt (nhất) có thể sản xuất hơn 5 lần trung bình, người kém có thể không cho kết quả gì

- Một số công việc quá khó đối với mọi người

Không nên tăng số thành viên một cách vô ý thức, vì như thế chỉ làm tăng sự phức tạp giao tiếp giữa các thành viên, khiến công việc nhiều khi chậm lại. Một số việc (phức tạp, đặc thù) chỉ nên để một người làm.

6.5 Quản lý cấu hình

Quản lý cấu hình phần mềm (còn gọi là quản lý mã nguồn) là một công việc quan trọng trong sản xuất phần mềm. Mã nguồn (và dữ liệu) là sản phẩm chính của dự án phần mềm.

Quản lý cấu hình được tự động hóa thông qua các công cụ. Nhiệm vụ chính của công cụ quản lý là:

- lưu trữ mã nguồn
- tạo ra một điểm truy cập duy nhất (phiên bản thống nhất) cho người lập trình sửa đổi, thêm bớt mã nguồn.

Do đó chúng ta có thể dễ dàng:

- kiểm soát được tính thống nhất của mã nguồn
- kiểm soát được sự sửa đổi, lý do của sự sửa đổi, lý lịch các lần sửa đổi
- dễ dàng lưu trữ và truy cập tới các phiên bản khác nhau của phần mềm
- tối ưu hóa vùng đĩa cần thiết cho lưu trữ

Phương thức hoạt động của các công cụ này là:

- quản lý tập chung (mã nguồn, tư liệu, công cụ phát triển...)
- các tệp được tạo một lần duy nhất, các phiên bản sửa đổi chỉ ghi lại sai phân đối với bản gốc
- sử dụng phương pháp check out/check in khi sửa đổi tệp

Thông thường, người phát triển khi muốn sửa đổi mã nguồn sẽ thực hiện thao tác check out tệp đó. Khi tệp đã bị check out thì các người phát triển khác chỉ có thể mở tệp dưới dạng chỉ đọc. Khi kết thúc sửa đổi và ghi tệp vào CSDL, người sửa đổi tiến hành check in để thông báo kết thúc công việc sửa đổi, đồng thời có thể ghi lại các thông tin liên quan (lý do sửa đổi...) đến sự sửa đổi.

Dữ liệu được lưu trữ của dự án thông thường bao gồm:

- mã nguồn
- dữ liệu
- tư liệu
- công cụ phát triển (chương trình dịch...), thường cần để đảm bảo tương thích với các phiên bản cũ, và để đảm bảo chương trình được tạo lại (khi sửa lỗi...) đúng như cái đã phân phát cho khách hàng
- các ca kiểm thử

Một số các công cụ quản lý cấu hình phổ biến là RCS, CVS trên HĐH Solaris và SourceSafe của Microsoft.

6.6 Quản lý rủi ro

Quản lý rủi ro là một công việc đặc biệt quan trọng và khó khăn trong phát triển phần mềm. Có các nguyên nhân (rủi ro) sau dẫn đến chấm dứt dự án:

- chi phí phát triển quá cao
- quá chậm so với lịch biểu
- tính năng quá kém so với yêu cầu

Quản lý rủi ro bao gồm các công việc chính sau:

- dự doán rủi ro
- đánh giá khả năng xảy ra và thiệt hại
- tìm giải pháp khắc phục

Dưới đây là các rủi ro thường xẩy ra khi phát triển phần mềm và các phương pháp khắc phục chúng:

- thiếu người phát triển: sử dụng những người tốt nhất; xây dựng nhóm làm việc; đào tạo người mới
- kế hoạch, dự toán không sát thực tế: ước lượng bằng các phương pháp khác nhau; lọc, loại bỏ các yêu cầu không quan trọng.
- phát triển sai chức năng: chọn phương pháp phân tích tốt hơn; phân tích tính tổ chức/mô hình nghiệp vụ của khách hàng
- phát triển sai giao diện: phân tích thao tác người dùng; tạo kịch bản cách dùng; tạo bản mẫu.
- yêu cầu quá cao: lọc bớt yêu cầu; phân tích chi phí/lợi ích.
- thay đổi yêu cầu liên tục: áp dụng thiết kế che dấu thông tin; phát triển theo mô hình tiến hóa.

Tài liệu tham khảo

[1] Kent Beck, *Extreme Programming Explained*, Addison-Wasley, 2000.

[2] Bruce Eckel, *Thinking in Java*, 3rd ed., 2002.

[3] Mike Gancarz, *The Unix Philosophy*, Digital Press, 1994.

[4] Roger S. Pressman (dịch: Ngô Trung Việt), *Kỹ nghệ phần mềm*, Tập I,II,III, NXB Giáo dục, 1997.

[5] Walker Royce, *Software Project Management - A Unified Framework*, Addison-Wesley, 1998.

[6] Stephen R. Schach, *Classical and Object-Oriented Software Engineering with UML and C++*, 4th ed., McGraw-Hill, 1999.

[7] Ian Sommerville, *Software Engineering*, 6th ed., Addison-Wasley, 2001.

[8] Nguyễn Quốc Toản, *Bài giảng về Nhập môn Công trình học phần mềm*, Khoa Công nghệ, 1999.

[9] Lê Đức Trung, *Công nghệ phần mềm*, NXB Khoa học và Kỹ thuật, 2001.

[10] Ngô Trung Việt, Nguyễn Kim Ánh (biên soạn), *Nhập môn Công nghệ phần mềm*, NXB Khoa học và kỹ thuật, 2003.

[11] Nguyễn Văn Vỵ, *Phân tích thiết kế các hệ thống thông tin hiện đại*, NXB Thống kê, 2002.