MỤC LỤC

DANH MỤC BẢNG BIỂU

DANH MỤC HÌNH ẢNH

THUẬT NGỮ VIẾT TẮT

Chữ viết tắt	Chi tiết tiếng Anh	Nghĩa tiếng Việt
APSE	Ada Programming Support Environnement	Môi trường hỗ trợ ngôn ngữ lập trình Ada
ATM	Automated Teller Machine	Máy rút tiền tự động
CASE	Computer Aided Software Engineering	Công nghệ phần mềm được máy tính trợ giúp
CNTT		Công nghệ thông tin
COST	Commercial off-the-shelf	Gói phần mềm thương mại
CPM	Critical Path Method	Phương pháp đường găng
CSDL		Cơ sở dữ liệu
ERP	Entreprise Resource Planing	Quản trị tài nguyên doanh nghiệp
ICSE	Internetional Conference of SE	Hội thảo quốc tế về công nghệ phần mềm
IEEE	Institute Electrical and Electronic Engineers	Viện kỹ nghệ điện và điện tử
OOA	Object Oriented Analysis	Phân tích hướng đối tượng
OOD	Object Oriented Design	Thiết kế hướng đối tượng
OOP	Object Oriented Programme	Lập trình hướng đối tượng
SAP	Systems Applications and Products	Các hệ thống ứng dụng và sản xuất
SE	Software Engineering	Công nghệ phần mềm
RUP	Rational Unified Process	Tiến trình phần mềm hợp nhất
UML	Unified Modeling Language	Ngôn ngữ mô hình hóa thống nhất

LỜI NÓI ĐẦU

Sau gần 4 thập kỷ phát triển, công nghệ phần mềm (SE - Software Engineering) đến nay được xem là một phân ngành quan trọng của chuyên ngành công nghệ thông tin.

Xuất phát từ cuộc khủng hoảng phần mềm cuối những năm 60, các nhà quản lý và chuyên gia công nghệ thông tin đã nhận ra nhu cầu về cách tiếp cận có nguyên tắc hơn đối với việc phát triển phần mềm. Công nghệ phần mềm không đơn thuần là việc sinh ra một sản phẩm phần mềm, mà nó liên quan đến việc tạo ra một sản phẩm phần mềm một cách hiệu quả. Với những nguồn lực phần mềm không hạn chế, thì đa số các vấn đề trong phần mềm đều có thể giải quyết. Tuy nhiên, thách thức lớn nhất đối với các kỹ sư phần mềm là phải tạo ra một sản phẩm có chất lượng cao với chi phí thấp và thời gian phát triển tuân theo một lịch trình định trước.

Bài giảng này trình bày những vấn đề cơ bản sau của công nghệ phần mềm:

1. *Những vấn đề cơ bản của công nghệ phần mềm*: giới thiệu khái quát lịch sử hình thành và phát triển của công nghệ phần mềm, những khái niệm cơ bản liên quan đến công nghệ phần mềm.

2. *Tiến trình phát triển phần mềm*: tìm hiểu về quy trình và các cách thức cơ bản để phát triển phần mềm. Nêu những bước chính trong phát triển phần mềm: Khảo sát, phân tích hệ thống và yêu cầu phần mềm, thiết kế hệ thống và phát triển phần mềm, kiểm thử và bảo trì phần mềm.

3. *Công cụ hỗ trợ các hoạt động phát triển phần mềm*: Nhấn mạnh sự trợ giúp đắc lực của các công cụ phần mềm máy tính cho tiến trình phát triển phần mềm nhằm đạt được năng suất và chất lượng cao.

4. *Vấn đề quản lý dự án phần mềm*: Tiến trình phát triển dự án phần mềm và việc quản lý nó một cách hiệu quả.

Bài giảng này được biên soạn lần đầu để giảng dạy cho sinh viên chuyên ngành tin học và quản lý thông tin của Học viện Nông nghiệp Việt Nam. Nó cung cấp những kiến thức có tính nền tảng về vấn đề phát triển phần mềm cho những người hoạt động trong lĩnh vực tin học nói chung và cho sinh viên của khoa Công nghệ thông tin nói riêng.

Tác giả xin chân thành cảm ơn các đồng nghiệp của khoa Công nghệ thông tin, Học viện Nông nghiệp Việt Nam đã cho những ý kiến đóng góp để hoàn thiện tài liệu này.

Mặc dù đã rất cố gắng tham khảo nhiều tài liệu khác nhau, đúc rút từ thực tiễn giảng dạy và nghiên cứu khoa học, tổ chức biên soạn một cách công phu và kỹ càng, nhưng bài giảng không tránh khỏi nhiều thiếu sót. Tác giả chân thành mong đợi góp ý của các bạn đồng nghiệp, sinh viên và các độc giả để tài liệu ngày càng hoàn thiện hơn.

Mọi ý kiến đóng góp xin gửi về địa chỉ: Bộ môn Công nghệ phần mềm - Khoa Công nghệ thông tin – Học viện Nông nghiệp Việt Nam.

Tác giả

Phạm Thủy Vân

Chương 1: MỞ ĐẦU

Ngày nay, hầu hết hoạt động của mỗi quốc gia đều phụ thuộc vào các hệ thống máy tính phức tạp, phần mềm hoạt động trên các máy tính trở thành điều kiện sống còn của các máy tính đó. Cơ sở hạ tầng, các tiện ích dựa trên các hệ thống máy tính, các sản phẩm điện tử... đều bao gồm một máy tính và phần mềm điều khiển, sản xuất công nghiệp và dịch vụ phân phối dựa hoàn toàn vào máy tính. Do đó, việc sản xuất và bảo trì một cách hiệu quả với chi phí hợp lý là vấn đề chủ yếu ảnh hưởng tới sự phát triển của nền kinh tế quốc gia và quốc tế.

Sau một thời gian ngành sản xuất phần mềm rơi vào khủng hoảng, các nhà khoa học nhanh chóng nhận ra tầm quan trọng của việc đưa ra phương pháp luận cho phát triển phần mềm. Từ đó khái niệm công nghệ phần mềm ra đời. Công nghệ phần mềm là kỹ nghệ cơ bản tập trung chủ yếu vào việc phát triển phần mềm có chất lượng cao, hiệu quả với chi phí và thời gian phát triển hợp lý. Chương này giới thiệu về lịch sử hình thành và phát triển của ngành công nghệ phần mềm, một số khái niệm cơ bản có liên quan. Phần cuối chương đề cập đến một số vấn đề về đạo đức của các chuyên gia CNTT, từ đó giúp người học có ý thức hơn về vai trò và trách nhiệm của họ khi tham gia vào thị trường lao động trong lĩnh vực này.

1.1. LỊCH SỬ HÌNH THÀNH VÀ PHÁT TRIỂN

Trước thập kỷ 90 của thế kỷ XX, công nghệ thông tin tập trung nhiều cho phát triển phần cứng, nhằm giảm giá thành xử lý và tăng dung lượng lưu trữ dữ liệu. Từ thập kỷ 90 của thế kỷ XX trở lại đây, sự phát triển của công nghệ thông tin tập trung nhiều vào cải thiện chất lượng và giảm giá thành của các giải pháp dựa trên máy tính, tức là các phương pháp được cài đặt bằng phần mềm. Sự cấp bách này do nhu cầu phần mềm ngày càng tăng nhanh cả về số lượng, quy mô cũng như các tính năng.

1.1.1. Quá trình tiến hóa của phần mềm

Quá trình tiến hóa của phần mềm diễn ra qua các thời kỳ cũng tăng dần cùng với sự tiến bộ của phần cứng.

a) Những năm 1950s đến những năm 1960s

Trong giai đoạn này, phần cứng có năng lực hạn chế và thay đổi liên tục, phần mềm phần lớn mang tính chuyên dụng. Lập trình máy tính "theo bản năng" và được xem là một nghệ thuật, chưa có phương pháp mang tính hệ thống, phát triển phần mềm chưa được quản lý, phần lớn hệ thống xử lý theo lô.

Môi trường phát triển phần mềm mang tính cá nhân, tiến trình phát triển phần mềm không tường minh, thường không có tài liệu. Sản xuất phần mềm mang tính đơn chiếc, theo đơn đặt hàng, trong chương trình còn cho phép chấp nhận lỗi. Kết quả là, những người làm phần mềm có thể học được việc cài đặt một hệ thống dựa trên máy tính, nhưng không học được nhiều về kỹ nghệ làm ra các phần mềm một cách hiệu quả.

b) Những năm 1960s đến giữa những năm 1970s

- *Hệ thống phần mềm đa chương trình, đa người sử dụng phát triển*, dẫn đến khái niệm mới về *tương tác người - máy*. Kỹ thuật này mở ra nhiều ứng dụng mới và đòi hỏi mức độ tinh vi của cả phần mềm và phần cứng.

- *Hệ thống thời gian thực* ra đời, bao gồm việc thu thập, phân tích, biến đổi dữ liệu từ nhiều nguồn khác nhau để kiểm soát các tiến trình. Hệ thống phải đưa ra các yêu cầu đáp ứng trong phần nghìn giây, thay vì nhiều phút như trước.

- Tiến bộ lưu trữ trực tuyến làm xuất hiện các *hệ quản trị cơ sở dữ liệu thế hệ đầu.*

- 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. Vì thế nảy sinh nhu cầu sửa chữa chương trình khi gặp lỗi, hay người dùng yêu cầu chương trình phải thích nghi với những thay đổi của môi trường. Công việc bảo trì phần mềm làm phát sinh những bức xúc, tiêu tốn nhiều công sức và tài nguyên đến mức báo động.

c) Giữa những năm 1970s – 1990s

Thời kỳ này đặc trưng bằng việc phát triển mạng toàn cục và cục bộ, truyền thông tín hiệu số giải thông cao. Những sự kiện này đã làm tăng nhu cầu truy nhập dữ liệu, yêu cầu phát triển các phần mềm quản lý dữ liệu. Cùng với nó là sự phát triển các hệ thống phân tán làm tăng quy mô và độ phức tạp của phần mềm.

Sự tiến bộ nhanh và sử dụng phổ biến các bộ vi xử lý cho các thiết bị (ôtô, robot, lò vi sóng, thiết bị chuẩn đoán ...) trong công nghiệp, máy tính cá nhân ra đời và các máy trạm để bàn phát triển, làm cho nhu cầu về phần mềm tăng nhanh do phạm vi người dùng mở rộng. Phần cứng ngày càng ổn định, chi phí phần mềm có khuynh hướng tăng nhanh hơn chi phí mua máy. Từ đó nảy sinh nhu cầu tăng năng suất sản xuất phần mềm.

Phương pháp luận và phương pháp phát triển phần mềm hướng cấu trúc đạt đến mức độ hoàn thiện cao và cùng với nó là sự phát triển các công cụ trợ giúp công nghệ phần mềm bằng máy tính (CASE), đã làm tăng năng suất và chất lượng phần mềm một cách đáng kể.

d) Thời kỳ năm 1990 - nay

Trong thời kỳ này, cách tiếp cận công nghệ hướng đối tượng nhanh chóng thay thế các cách tiếp cận truyền thống trong việc phát triển nhiều lĩnh vực ứng dụng.

Các hệ thống thông minh như hệ chuyên gia và phần mềm trí tuệ nhân tạo chuyển từ phòng thí nghiệm ra thực tế. Phần mềm mạng nơron nhân tạo đã mở ra khả năng nhận dạng và có khả năng xử lý thông minh kiểu con người.

Sự phát triển của Internet làm cho người dùng máy tính tăng vọt, nhu cầu phần mềm ngày càng lớn, quy mô và độ phức tạp của những hệ thống phần mềm mới cũng tăng đáng kể. Các hệ thống dựa trên nền Web đang chiếm ưu thế trong các ứng dụng nghiệp vụ. Công nghệ hướng đối tượng (tiêu biểu như các hệ .NET) và phát triển phần mềm theo hướng tái sử dụng (Pattern và Frameworks) đang trở thành một xu hướng công nghệ. Tất cả các yếu tố trên tạo nên những thách thức mới cho việc phát triển phần mềm hiện nay.

1.1.2. Sự ra đời của công nghệ phần mềm

Nhìn lại sự tiến hóa của phần mềm ta thấy rằng trong những năm 60s – 70s của thế kỷ trước, nhiều vấn đề liên tục phát sinh, tạo ra những thách thức cho việc phát triển phần mềm như:

- Sự tăng quy mô của phần mềm (quy mô bài toán, phạm vi vấn đề, phạm vi ứng dụng ngày càng đa dạng phức tạp).

- Sự tăng chi phí làm phần mềm (cần nhiều lao động có kỹ năng).

- Sự kéo dài thời gian phát triển phần mềm (do phần mềm lớn).

- Sự phụ thuộc nhiều vào kinh nghiệm của người làm phần mềm.

- Chất lượng phần mềm không ổn định do phụ thuộc vào con người.

- Thiếu nghiêm trọng kỹ sư phần mềm (do nhu cầu tăng nhanh).

- Gánh nặng bảo trì nhiều hệ thống cũ để tiếp tục hoạt động.

Giải quyết các vấn đề nêu trên làm nảy sinh việc nghiên cứu các giải pháp cho chúng. Vào những năm 70s của thế kỷ XX, phát triển phần mềm được thừa nhận và bắt đầu trở thành một ngành công nghiệp do yêu cầu sử dụng mở rộng.

Năm 1975, sau hội nghị về Công nghệ phần mềm quốc tế (ICSE - Internetional Conference of SE), nhiều lý thuyết, phương pháp luận và kỹ thuật được đề nghị. Vào những năm 90s của thế kỷ XX, công cụ trợ giúp công nghệ phần mềm bằng máy tính (CASE) phát triển mạnh. Nhờ vậy, việc tự động hóa một số bước trong quá trình phát triển phần mềm đã phổ biến hơn. Nhiều phương pháp luận và kỹ thuật đã được đề nghị và áp dụng sau khủng hoảng. Tuy nhiên, tính ổn định của các sản phẩm phần mềm và kỹ thuật kiểm thử còn chưa được giải quyết trọn vẹn. Vì vậy, công nghệ phần mềm ra đời như một đòi hỏi tất yếu của phát triển phần mềm.

1.2. MỘT SỐ KHÁI NIỆM CƠ BẢN TRONG LĨNH VỰC CÔNG NGHỆ PHẦN MỀM

1.2.1. Khái niệm phần mềm

Rất nhiều người cho rằng phần mềm và các chương trình máy tính là hai khái niệm tương đương nhau. Tuy nhiên, có thể hiểu theo một định nghĩa rộng hơn, phần mềm không chỉ đơn thuần là những chương trình, mà nó còn kết hợp với tài liệu, cấu hình dữ liệu cần thiết để thực hiện những chức năng của chương trình một cách đúng đắn và hợp lý. Một hệ thống phần mềm thông thường bao gồm một số chương trình riêng biệt, các tệp cấu hình mà chúng được sử dụng để thiết lập các chương trình, tài liệu hệ thống, những tài liệu mô tả cấu trúc của hệ thống và tài liệu người dùng, những trang Web để người dùng có thể tải về những thông tin mới nhất của sản phẩm. Có hai kiểu phần mềm cơ bản:

Generic products (sản phẩm dùng chung): đó là những hệ thống độc lập được sản xuất bởi một tổ chức phát triển và bán cho thị trường mở, cho bất kỳ khách hàng nào có nhu cầu và khả năng mua chúng. Những ví dụ về kiểu sản phẩm này bao gồm những phần mềm chạy trên các máy tính cá nhân như: cơ sở dữ liệu, xử lý văn bản, các gói phần mềm đồ họa, các công cụ quản lý dự án…

Customised (or bespoke) products (sản phẩm đặt hàng): các hệ thống này được xây dựng cho một đối tượng khách hàng cụ thể. Trước khi bắt tay phát triển sản phẩm cần phải có một hợp đồng phát triển phần mềm được xây dựng cho một khách hàng. Các ví dụ về các hệ thống này là: các hệ thống điều khiển các thiết bị điện, các hệ thống được viết riêng để phục vụ cho nhu cầu sản xuất và kinh doanh của một đơn vị, các hệ thống điều khiển giao thông hàng không…

Sự khác nhau cơ bản giữa các loại phần mềm này là: đối với các phần mềm dùng chung, tổ chức phát triển phần mềm sẽ kiểm soát việc đặc tả phần mềm, còn với những phần mềm riêng,

việc đặc tả phần mềm thường được thực hiện và kiểm soát bởi tổ chức mua phần mềm, những người phát triển phần mềm phải làm việc theo đặc tả này.

Tuy nhiên, sự phân biệt này ngày càng trở nên không rõ ràng, ngày càng nhiều công ty phần mềm bắt đầu với một hệ thống phần mềm chung và biến đổi nó cho phù hợp với nhu cầu của từng khách hàng. Các hệ thống ERP hay SAP là một ví dụ rõ ràng nhất cho cách tiếp cận này.

1.2.2. Khái niệm công nghệ phần mềm

Công nghệ phần mềm – software engineering (một số tài liệu gọi là kỹ nghệ phần mềm) là một kỹ nghệ cơ bản liên quan tới tất cả các khía cạnh của sản xuất phần mềm, từ giai đoạn bắt đầu đặc tả hệ thống cho đến khi bảo trì hệ thống và sau khi đưa vào sử dụng. Trong định nghĩa này, có hai khái niệm quan trọng:

Những nguyên tắc công nghệ: là các nguyên tắc mà các kỹ sư phải sử dụng trong công việc. Họ áp dụng những lý thuyết, phương thức và công cụ thích hợp, sử dụng chúng một cách có chọn lọc và luôn luôn cố gắng phát hiện ra các giải pháp cho vấn đề đặt ra, thậm chí ngay cả khi không có các phương thức và cơ sở lý thuyết thích hợp. Các kỹ sư phần mềm phải làm việc trong những ràng buộc về tài chính và về tổ chức, do đó, những giải pháp mà họ đưa ra phải nằm trong những ràng buộc này.

Tất cả các khía cạnh của sản phẩm phần mềm: công nghệ phần mềm không chỉ liên quan tới các tiến trình kỹ thuật của việc phát triển phần mềm mà nó còn là những hoạt động, chẳng hạn như việc quản lý dự án phần mềm và việc phát triển các công cụ, các phương pháp, cơ sở lý thuyết cho việc phát triển phần mềm.

Nói chung, các kỹ sư phần mềm phải tuân theo một phương pháp và cách tiếp cận được xác định cho công việc của họ để có thể đưa ra được phần mềm chất lượng tốt. Hầu hết công nghệ được lựa chọn là thích hợp nhất, tuy nhiên trong một điều kiện cụ thể và tùy theo sự sáng tạo, việc phát triển phần mềm không tuân thủ những nguyên tắc ban đầu có thể sẽ hiệu quả hơn.

1.2.3. Sự khác nhau giữa công nghệ phần mềm và khoa học máy tính

Về cơ bản, khoa học máy tính liên quan tới những nguyên lý và phương pháp làm cơ sở cho máy tính và các hệ thống phần mềm, trong khi đó, công nghệ phần mềm liên quan tới những vấn đề thực tế trong sản xuất phần mềm. Những kiến thức về khoa học máy tính là cần thiết cho các kỹ sư phần mềm, giống như những hiểu biết về vật lý cho các kỹ sư điện. Một cách lý tưởng, tất cả các kỹ sư phần mềm cần phải lấy nền tảng kiến thức từ khoa học máy tính.

1.2.4. Tiến trình phần mềm

Một tiến trình phần mềm là một tập hợp các hoạt động để sản xuất ra phần mềm. Có 4 hoạt động cơ bản trong tiến trình phần mềm:

Đặc tả phần mềm: khách hàng và các kỹ sư phần mềm định nghĩa sản phẩm sẽ được sinh ra và những ràng buộc liên quan đến phần mềm cũng như quá trình phát triển phần mềm.

Phát triển phần mềm: phần mềm được thiết kế và lập trình.

Kiểm thử phần mềm: phần mềm được kiểm tra để đảm bảo rằng nó phù hợp với yêu cầu của khách hàng và không còn lỗi.

Cải tiến - bảo trì phần mềm: phần mềm được thay đổi để đáp ứng những thay đổi của khách hàng và thị trường.

Các kiểu hệ thống khác nhau cần những tiến trình phát triển khác nhau, ví dụ, các phần mềm thời gian thực điều khiển các thiết bị trong máy bay phải được đặc tả một cách hoàn chỉnh trước khi phát triển. Trong khi đó với các hệ thống thương mại, việc đặc tả và lập trình thường được phát triển cùng nhau. Kết quả là, những hoạt động chung này được tổ chức theo các cách khác nhau và được mô tả ở các mức độ chi tiết khác nhau. Tuy nhiên, việc sử dụng những tiến trình không hợp lý có thể làm giảm chất lượng hoặc giảm tính sử dụng của sản phẩm trong khi chi phí cho việc phát triển phần mềm cũng tăng theo.

1.2.5. Mô hình tiến trình phần mềm

Một mô hình tiến trình phần mềm là việc mô tả một tiến trình phát triển phần mềm. Các mô hình tiến trình có thể bao gồm việc mô tả các hoạt động trong tiến trình phần mềm, mô tả các sản phẩn phẩm mềm và vai trò của những người liên quan trong quá trình phát triển. Một vài ví dụ về các kiểu mô hình tiến trình là:

Workflow model: mô hình này chỉ ra thứ tự các hoạt động trong một tiến trình cùng với đầu vào, đầu ra và sự phụ thuộc. Các hoạt động trong mô hình này mô tả các hoạt động của con người.

Data-flow/Activity model: các mô hình này mô tả tiến trình như một tập hợp các hoạt động, mỗi hoạt động thực hiện việc biến đổi một số dữ liệu. Nó chỉ ra rằng dữ liệu đầu vào được xử lý như thế nào, chẳng hạn như một bản đặc tả được biến đổi để có sản phẩm đầu ra là một bản thiết kế. Những hoạt động ở đây có thể chỉ ra sự biến đổi được thực hiện bởi con người hoặc máy tính.

Role/Action model: mô hình này giới thiệu vai trò của những người liên quan trong tiến trình phần mềm và những hoạt động mà họ phải chịu trách nhiệm.

Hầu hết các mô hình tiến trình đều dựa trên một trong ba cách tiếp cận phát triển phần mềm dưới đây:

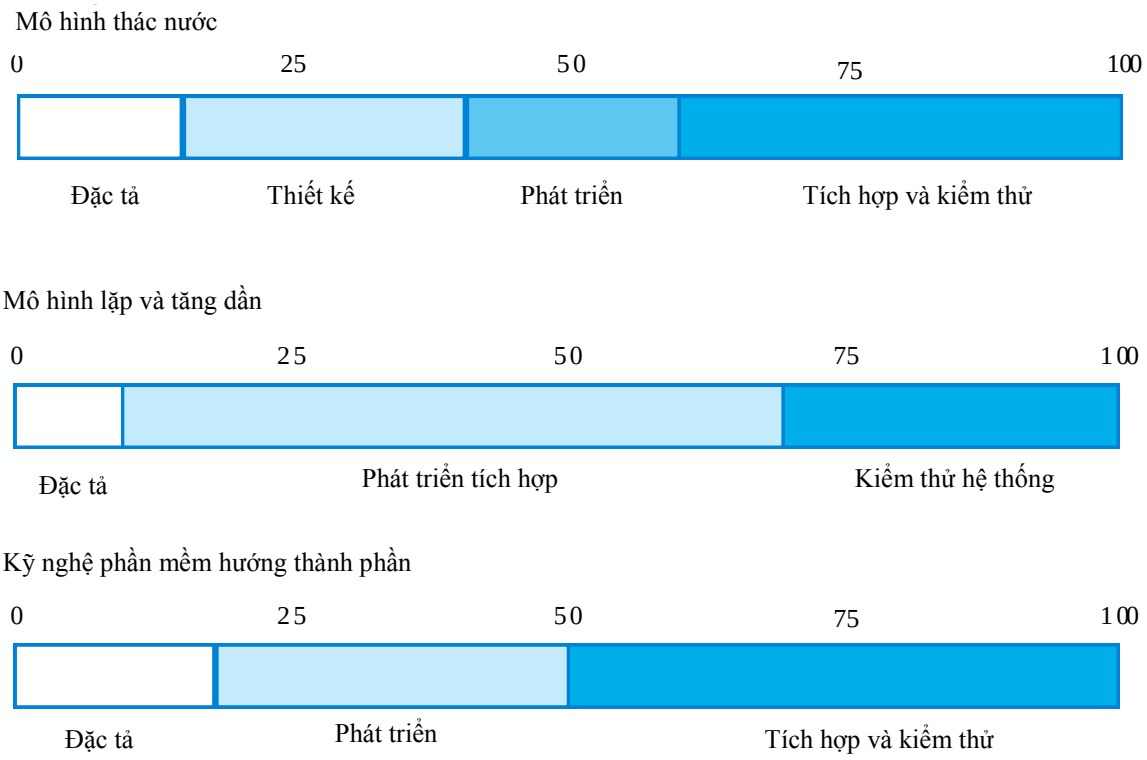
Cách tiếp cận thác nước: mô hình này lấy những hoạt động trên và biểu diễn chúng như những giai đoạn riêng rẽ, chẳng hạn như: đặc tả yêu cầu, thiết kế phần mềm, phát triển và kiểm thử… Việc phát triển phần mềm tuân theo tiến trình này.

Phát triển lặp và tăng dần: cách tiếp cận này đan xen các hoạt động đặc tả, phát triển và kiểm thử. Một hệ thống ban đầu được xây dựng rất nhanh từ những đặc tả rất trìu tượng, sau đó được làm mịn với đầu vào của khách hàng để sinh ra một hệ thống phù hợp với yêu cầu của khách hàng. Hệ thống này sau đó có thể được bàn giao tới khách hàng sử dụng hoặc tiếp tục phát triển để nâng cấp cải tiến sản phẩm.

Công nghệ phần mềm dựa trên thành phần (tái sử dụng): kỹ thuật này là việc kết hợp các thành phần có sẵn để xây dựng thành một ứng dụng mới. Tiến trình phát triển hệ thống chủ yếu dựa trên việc tích hợp các thành phần này lại với nhau hơn là việc phát triển chúng từ đầu.

1.2.6. Chi phí của công nghệ phần mềm

Chi phí của công nghệ phần mềm là việc xác định nguồn kinh phí cần thiết cho từng giai đoạn phát triển phần mềm. Không có câu trả lời đơn giản cho câu hỏi này vì nó còn phụ thuộc vào mô hình tiến trình được lựa chọn và kiểu phần mềm cần xây dựng. Ví dụ: phần mềm thời gian thực thường đòi hỏi việc kiểm thử đắt hơn so với các hệ thống dựa trên nền web. Hình vẽ 1.1 chỉ ra phần trăm chí phí cho từng giai đoạn ứng với mỗi mô hình tiến trình khác nhau.



1.

phần

m

h đó

v

năm

7

ống;

các c năng đến nay những năm 80s, 90s, ngoài các phương pháp hướng chức năng còn có thêm các phương pháp hướng đối tượng. Những cách tiếp cận khác nhau ngày nay được tích hợp trong một cách tiếp cận hợp nhất dựa trên ngôn ngữ UML.

Không có phương pháp lý tưởng, các phương pháp khác nhau thích hợp với những lĩnh vực ứng dụng khác nhau. Ví dụ: các phương pháp hướng đối tượng thường thích hợp cho các hệ thống tương tác nhưng lại không thích hợp cho những hệ thống có yêu cầu thời gian thực.

Tất cả các phương pháp đều dựa trên ý tưởng phát triển các mô hình hệ thống có thể biểu diễn bằng đồ họa và sử dụng các mô hình này như một bản đặc tả hoặc bản thiết kế. Các phương pháp có thể bao gồm một số thành phần khác nhau:

Bảng 1.1. Các thành phần mô hình hệ thống

Thành phần	*Mô tả*	*Ví dụ*

Mô hình hệ thống	Mô tả các mô hình hệ thống nên được phát triển và khái niệm được sử dụng trong mô hình này.	Mô hình đối tượng, mô hình data-flow, mô hình máy ảo và phân tầng…
Quy tắc (rules)	Các ràng buộc luôn luôn phải áp dụng trong mô hình hệ thống.	Mỗi thực thể trong một mô hình hệ thống phải có một tên duy nhất.
Gợi ý (recommendation)	Khám phá ra những điểm tốt trong thiết kế của phương thức này. Tuân theo những gợi ý này có thể giúp bạn xây dựng được một mô hình tổ chức hệ thống tốt.	Không có đối tượng nào có trên 7 đối tượng con nằm trong nó.
Quy trình hướng dẫn	Mô tả các hoạt động mà ta có thể tuân theo để phát triển các mô hình hệ thống và tổ chức các hoạt động này.	Các thuộc tính của đối tượng nên được đưa vào tài liệu trước khi xác định các phương thức cho đối tượng này.

1.2.8. CASE - Các công cụ trong công nghệ phần mềm

Computer-Aided software engineering bao gồm các kiểu chương trình khác nhau được sử dụng để hỗ trợ cho các hoạt động trong tiến trình phần mềm, chẳng hạn như phân tích yêu cầu, mô hình hóa hệ thống, gỡ lỗi và kiểm thử. Các công cụ thường được tích hợp để thực hiện một chức năng trọn vẹn hay một số chức năng riêng rẽ. Khi các công cụ được tích hợp tới mức thông tin do chúng tạo ra có thể được dùng cho các công cụ khác hoặc cho các giai đoạn tiếp theo của quá trình phát triển thì chúng được gọi là những bộ công cụ (workbenches), hay một hệ thống trợ giúp phát triển phần mềm và được gọi là môi trường phát triển (developmeny invironments). Các hệ thống này gọi chung là công nghệ phần mềm được máy tính trợ giúp (CASE).

1.2.9. Những thuộc tính phần mềm tốt

Cũng như các dịch vụ mà nó cung cấp, các sản phẩm phần mềm phải có một số thuộc tính khác nhau phản ánh chất lượng của phần mềm. Những thuộc tính này không liên quan trực tiếp đến những gì mà phần mềm thực hiện, thay vào đó, chúng phản ánh cách thực thi của phần mềm, cấu trúc và cách tổ chức chương trình nguồn, những tài liệu liên quan. Dưới đây là một số thuộc tính có thể lấy làm cơ sở để đánh giá chất lượng phần mềm.

Bảng 1.2. Các thuộc tính của phần mềm

Đặc tính sản phẩm	***Mô tả***
Tính bảo trì	Phần mềm nên được viết theo cách mà nó có thể đáp ứng được những thay đổi của khách hàng. Đây là một trong những thuộc tính quan trọng bởi sự thay đổi của phần mềm là không thể tránh khỏi.
Độ tin cậy	Độ tin cậy của phần mềm bao gồm một số đặc tính như: tin cậy, an ninh và an toàn. Phần mềm tin cậy không phải là nguyên nhân của những thiệt hại về kinh tế và vật lý ngay cả khi hệ thống đó bị lỗi.
Tính hiệu quả	Phần mềm nên được thiết kế trong đó tính đến cả khả năng của các

	nguồn tài nguyên như bộ nhớ và bộ vi xử lý. Tính hiệu quả ở đây bao gồm: thời gian trả lời, thời gian xử lý và việc sử dụng bộ nhớ.
Tính sử dụng	Phần mềm phải được người sử dụng chấp nhận, điều này có nghĩa là nó cần phải hiểu được, sử dụng được và tương thích với các hệ thống khác.

1.2.10. Những thách thức cơ bản của lĩnh vực phát triển phần mềm

Tính không đồng nhất: các hệ thống được yêu cầu để thực thi một cách phân tán trên các hệ thống khác nhau được kết nối trong mạng, bao gồm các máy tính khác nhau, các phần mềm hỗ trợ khác nhau, môi trường thực thi khác nhau. Ta thường xuyên phải tích hợp các phần mềm mới vào các hệ thống cũ được viết bằng các ngôn ngữ lập trình khác nhau. Tính không đồng nhất là một thách thức trong việc phát triển các kỹ thuật xây dựng các phần mềm độc lập và mềm dẻo để có thể thực thi trên các môi trường không đồng nhất.

Thách thức về việc bàn giao: Một trong những hạn chế của các kỹ thuật công nghệ phần mềm truyền thống là thời gian cho ra đời một sản phẩm có thể sử dụng được là khá dài. Tuy nhiên, ngày nay trong kinh doanh cần phải đáp ứng một cách nhanh chóng những thay đổi, điều đó có nghĩa là phải cung cấp được những sản phẩm phần mềm có thể biến đổi một cách nhanh chóng. Thách thức này là việc bàn giao sản phẩm trong thời gian ngắn, với những hệ thống lớn và phức tạp mà đôi khi không cần sự thỏa hiệp về chất lượng hệ thống.

Thách thức về độ tin cậy: khi các sản phẩm phần mềm không tách rời khỏi cuộc sống của chúng ta thì điều quan trọng là chúng ta phải có được những sản phẩm phần mềm đủ độ tin cậy. Điều này đặc biệt đúng trong các hệ thống truy cập từ xa thông qua các trang Web hoặc qua giao diện. Thách thức về độ tin cậy là phải phát triển những kỹ thuật để có thể chứng minh được rằng sản phẩm là đáng tin cậy đối với người sử dụng.

1.3. MỘT SỐ VẤN ĐỀ VỀ ĐẠO ĐỨC CỦA CÁC CHUYÊN GIA CNTT

Trong thực tế, mỗi lĩnh vực đều có những người làm việc có trách nhiệm, hiểu biết và được đào tạo tốt, họ được gọi là các chuyên gia. Đó có thể là các nhà phân tích thị trường, tư vấn tài chính hay các chuyên gia công nghệ thông tin. Trong lĩnh vực công nghệ thông tin có một số chuyên ngành như:

- Lập trình
- Phân tích hệ thống
- Công nghệ phần mềm
- Quản trị dữ liệu
- Quản trị mạng
- Quản lý thông tin.

Cũng có những người làm việc về công nghệ thông tin, nhưng họ không được đánh giá là những chuyên gia, vì họ không có các bằng cấp chứng nhận. Như vậy, về cơ bản, họ sẽ không phải chịu trách nhiệm trước pháp lý về những sai lầm mà họ mắc phải, vì họ không đáp ứng các định nghĩa của pháp luật là một nhà chuyên môn. Đây là điểm khác biệt của ngành công nghệ thông tin so với các lĩnh vực khác, ví dụ như y tế, pháp luật, giáo dục…

1.3.1. Những mối quan hệ cần phải quản lý của các chuyên gia công nghệ thông tin

Các chuyên gia công nghệ thông tin phải làm việc trong một môi trường có nhiều ràng buộc, nhiều mối quan hệ với các đối tượng khác nhau, bao gồm người sử dụng lao động, khách hàng, nhà cung cấp, các chuyên gia khác, người sử dụng công nghệ thông tin và cộng đồng nói chung. Những mối quan hệ này cần phải được xem xét, đánh giá và cũng cần đề ra những quy tắc đạo đức nhất định. Vì việc vi phạm những quy tắc đạo đức này có thể dẫn đến tổn thất nặng nề về tính mạng, sức khỏe và tài sản của những người sử dụng trực tiếp hoặc gián tiếp các hệ thống được phát triển.

a) Chuyên gia công nghệ thông tin và người sử dụng lao động

Mối quan hệ này tương đối phức tạp, đòi hỏi phải có sự nỗ lực từ cả hai phía để duy trì. Chuyên gia công nghệ thông tin và người sử dụng lao động cần phải thảo luận để đi đến sự thống nhất về một số điều khoản cơ bản trong hợp đồng. Những điều khoản này bao gồm: tên công việc, trách nhiệm công việc cụ thể, hiệu quả mong đợi, trang phục, nơi làm việc, giờ làm việc và lợi ích của công ty… Bên cạnh đó, có cả những quy định riêng của công ty, những quy tắc đạo đức, cách ứng xử trong tổ chức, bao gồm: bí mật công ty, quy định về việc nghỉ lễ, thời gian nghỉ do những lý do cá nhân… Rồi các vấn đề liên quan đến pháp luật như không được có những hành động vi phạm pháp luật, ví dụ như làm sai lệch nội dung báo cáo, tiết lộ thông tin dữ liệu của khách hàng… Tóm lại nó giống như tất cả các bản hợp đồng lao động trong những lĩnh vực khác.

Tuy nhiên, trong lĩnh vực công nghệ thông tin, do đặc thù nghề nghiệp, do tính chất từng công việc hoặc từng dự án nên có thể có thêm những quy định riêng. Ví dụ: ngôn ngữ lập trình được sử dụng, loại và số lượng tài liệu hướng dẫn được đưa ra, quy trình kiểm tra đánh giá…

Các chuyên gia CNTT phải thiết lập một chuỗi các quy định cho việc thi hành các chính sách liên quan đến vấn đề đạo đức của việc sử dụng CNTT. Các chuyên gia CNTT có kiến thức, kỹ năng và có quyền truy cập cả thông tin chuyên môn lẫn thông tin cá nhân một cách thoải mái, cũng như cho phép hoặc không cho phép người khác làm như vậy.

Việc vi phạm bản quyền phần mềm, các hành vi trái pháp luật trong việc sao chép phần mềm hoặc tạo điều kiện cho những người khác để họ có thể truy cập vào những phần mềm mà họ không có quyền là một trong những vấn đề nhạy cảm, dễ dẫn đến những vi phạm pháp luật và quy định của công ty.

b) Mối quan hệ giữa các chuyên gia CNTT và khách hàng

Các chuyên gia thường cung cấp các dịch vụ cho khách hàng, những người làm việc bên ngoài hoặc bên trong tổ chức của họ. Trong mối quan hệ giữa các chuyên gia CNTT và khách hàng, mỗi bên cần phải cung cấp những lợi ích cho phía bên kia.

Những chuyên gia CNTT cung cấp: phần mềm, phần cứng hoặc các dịch vụ với chi phí cố định trong khoảng thời gian đưa ra. Mối quan hệ này thông thường được xác định qua các điều khoản hợp đồng: ai làm gì, khi nào bắt đầu công việc, mất bao nhiêu thời gian, kinh phí phải trả như thế nào… Thông thường có một sự chênh lệch rất lớn về chuyên môn giữa các chuyên gia CNTT và khách hàng, nên họ phải thường xuyên trao đổi thông tin liên lạc với nhau để đảm bảo thành công.

Thông thường khách hàng là người quyết định về các vấn đề về dự án trên cơ sở thông tin, sự tư vấn và lựa chọn của các chuyên gia CNTT, khách hàng tin tưởng và giao phó chúng cho các chuyên gia vì họ tin rằng họ sẽ có được những dịch vụ tốt nhất từ phía nhà cung cấp. Còn các chuyên gia CNTT phải tin tưởng rằng khách hàng sẽ cung cấp những thông tin có liên quan, lắng nghe và hiểu những điều mà các chuyên gia tư vấn, giải thích, đặt câu hỏi để hiểu những tác động của các quyết định, từ đó sẽ có được sự lựa chọn đúng đắn nhất từ những giải pháp mà các chuyên gia gợi ý. Trách nhiệm trong các quyết định được chia sẻ cho cả khách hàng và các chuyên gia CNTT.

Một trong những vấn đề thuộc về đạo đức giữa các chuyên gia CNTT và khách hàng liên quan đến những tư vấn viên CNTT hoặc kiểm toán viên, người giới thiệu các sản phẩm và dịch vụ của một nhà cung cấp cho một vấn đề mà họ phát hiện ra.

Ví dụ: một công ty tư vấn CNTT có thể được thuê để đánh giá việc lập kế hoạch chiến lược về CNTT của một công ty. Sau vài tuần làm việc, công ty tư vấn có thể đánh giá chiến lược đang tồn tại rất thấp và họ sẽ đưa ra một chiến lược phát triển mới. Câu hỏi đặt ra ở đây là liệu việc đánh giá này có khách quan hay không, và làm thế nào khách hàng có thể tin tưởng được?

Trong thời gian làm việc với dự án, Các chuyên gia CNTT không thể cung cấp đầy đủ và báo cáo chính xác tình trạng của dự án nếu như họ thiếu thông tin, công cụ hoặc kinh nghiệm để thực hiện một đánh giá chính xác. Người quản lý dự án muốn che dấu thông tin hoặc thay đổi nó trước khi công khai với những người khác.

c) Mối quan hệ giữa các chuyên gia CNTT và nhà cung cấp

Thông thường, các chuyên gia CNTT có mối liên hệ với nhiều nhà cung cấp phần cứng, phần mềm và các dịch vụ khác nhau, họ cũng hiểu rằng việc xây dựng mối quan hệ tốt với các nhà cung cấp cũng sẽ giúp họ có được sự trao đổi thông tin có ích và có thể cả việc chia sẻ ý tưởng trong công việc. Những thông tin này có thể dẫn đến những cải tiến về mặt công nghệ và giá thành sản phẩm, cũng như là hiệu quả của việc sử dụng nó. Đây có thể là những khía cạnh mà các chuyên gia CNTT không xem xét đến.

Các chuyên gia CNTT cần phải phát triển tốt mối quan hệ với các nhà cung cấp bằng cách chia sẻ với họ một cách công bằng và không đưa ra những yêu cầu không hợp lý.

Mặt khác, các nhà cung cấp cũng luôn luôn cố gắng để duy trì các mối quan hệ tích cực với khách hàng nhằm tăng doanh thu của công ty. Đôi khi, để đạt được những mục tiêu này, họ đã có những hành vi vi phạm đạo đức. Ví dụ như vấn đề quà biếu và nhận hối lộ.

d) Mối quan hệ giữa chuyên gia CNTT và các chuyên gia khác

Các chuyên gia cảm nhận được mức độ trung thành của các thành viên khác đối với mình. Kết quả là họ có thể giúp đỡ nhau để có những vị trí mới nhưng đôi khi cũng có những hiện tượng công kích nhau, hạ thấp uy tín của nhau trước công chúng.

Một số vấn đề về đạo đức có thể nảy sinh giữa các thành viên trong cùng một lĩnh vực. Một trong những ví dụ điển hình là vấn đề thổi phồng sự việc, liên quan tới việc không thành thật, nói quá về khả năng của mình so với thực tế.

Một vấn đề về đạo đức nữa là chia sẻ thông tin của các doanh nghiệp, do đặc thù nghề nghiệp, các chuyên gia CNTT có thể truy cập vào CSDL của doanh nghiệp, những thông tin riêng

tư, bí mật về nhân viên, khách hàng, nhà cung cấp, các kế hoạch sản phẩm mới, chương trình khuyến mại, ngân sách… Do đó họ có thể có cơ hội thực hiện những hành vi vi phạm đạo đức nghề nghiệp thông qua việc tiết lộ thông tin bí mật của khách hàng.

e) Mối quan hệ giữa chuyên gia CNTT và người sử dụng

Là người sử dụng những sản phẩm phần mềm và phần cứng, được thiết kế, cài đặt dịch vụ và hỗ trợ sản phẩm bởi các chuyên gia CNTT. Người sử dụng CNTT sử dụng những phương tiện này để nâng hiệu quả hoạt động của tổ chức.

Các chuyên gia CNTT có trách nhiệm phải hiểu nhu cầu và khả năng của người sử dụng để có thể cung cấp những dịch vụ tốt nhất cho những yêu cầu của họ, tất nhiên là còn phải cần nhắc đến cả những ràng buộc về thời gian và ngân sách.

Một trách nhiệm nữa của các chuyên gia CNTT là thiết lập môi trường làm việc giúp người sử dụng tránh những nguy cơ vi phạm đạo đức. Ví dụ không khuyến khích sử dụng các phần mềm vi phạm bản quyền, giảm thiểu việc sử dụng không thích hợp các nguồn tài nguyên công nghệ thông tin, tránh việc chia sẻ thông tin không hợp lý.

f) Mối quan hệ giữa các chuyên gia CNTT và cộng đồng

Những quy định của pháp luật thiết lập các tiêu chuẩn an toàn cho các sản phẩm và dịch vụ để bảo vệ công chúng. Tuy nhiên, các văn bản pháp luật đôi khi cũng có những thiếu sót, không thể bảo vệ một cách tuyệt đối quyền lợi của cộng đồng. Trong lĩnh vực CNTT, các chuyên gia có thể hiểu một cách chi tiết và rõ ràng những tác động tốt xấu do hệ quả công việc của họ tới cộng đồng và có thể có những hành động nhằm hạn chế những rủi ro có thể xảy ra. Vì thế, xã hội mong chờ họ chính là những người mang lại lợi ích cho xã hội. Một trong những cách tiếp cận để đáp ứng sự kỳ vọng này là thiết lập và duy trì những tiêu chuẩn nghề nghiệp để bảo vệ công chúng. Rõ ràng, các hoạt động của các chuyên gia CNTT có thể ảnh hưởng tới xã hội.

Ví dụ: các chuyên gia CNTT có thể thiết kế và phát triển một hệ thống kiểm soát tình trạng xử lý hóa chất của nhà máy. Nếu hệ thống này thiết kế không đúng, hoặc thất bại sẽ mang đến những nguy hại cho người dân ở gần nhà máy.

Tóm lại, những chuyên gia CNTT có mối liên hệ với nhiều đối tượng khác nhau trong xã hội, rất nhiều người có thể bị ảnh hưởng bởi các hành động của họ. Đó chính là câu trả lời cho câu hỏi: "Có cần thiết lập một chuẩn mực về đạo đức cho các chuyên gia CNTT hay không?"

1.3.2. Những quy tắc đạo đức của các chuyên gia CNTT

Quy tắc đạo đức nghề nghiệp đề cập đến các nguyên tắc và giá trị cốt lõi cần thiết cho công việc của một nhóm nghề cụ thể. Có một số ngành nghề các học viên phải cam kết tuân thủ các quy tắc đạo đức điều chỉnh hành vi của họ. Ví dụ như trong lĩnh vực y học, các học viên trước khi ra trường phải đọc lời thề Hypocrat. Lĩnh vực thi hành luật, các luật sư khi hành nghề cũng phải cam kết tuân thủ các quy tắc đạo đức nghề nghiệp. Các quy tắc này được hội đồng luật sư toàn quốc, liên đoàn luật sư ban hành.

Tuy nhiên, về cơ bản pháp luật không cung cấp tài liệu hoàn thiện để hướng dẫn các hành vi phi đạo đức. Một hành động không vi phạm pháp luật chưa chắc đã là một hành động mang

tính đạo đức. Vì vậy cần phải thiết lập những nguyên tắc đạo đức cơ bản và những giá trị cốt lõi cho từng ngành nghề cụ thể.

Một quy tắc đạo đức đưa ra những nguyên tắc cơ bản và những giá trị cốt lõi cần thiết cho công việc của một nhóm nghề riêng biệt. Hầu hết những quy tắc đạo đức của các chuyên gia trong một tổ chức bao gồm hai thành phần cơ bản sau:

- Phác thảo về những nguyện vọng cơ bản mà một chuyên gia mong muốn.
- Danh sách những quy tắc cơ bản mà các thành viên trong tổ chức cần tuân thủ.

Về cơ bản, các quy tắc đạo đức hướng tới lợi ích cho cả cá nhân, nghề nghiệp và xã hội thông qua cải thiện việc ra quyết định mang tính đạo đức, khuyến khích nâng cao các chuẩn trong thực tiễn và ứng xử có đạo đức, đề cao sự thật và tôn trọng cộng đồng. Để đảm các quy tắc đạo đức không bị vi phạm thì cũng cần đưa ra các tiêu chuẩn để đánh giá. Mục tiêu cơ bản của việc xây dựng các quy tắc đạo đức cho mỗi tổ chức là:

- Cải thiện việc ra quyết định mang tính đạo đức: việc tuân thủ một quy tắc đạo đức nghề nghiệp là các cá nhân trong tổ chức có thể sử dụng một tập các giá trị cốt lõi và niềm tin như kim chỉ nam cho việc ra các quyết định mang tính đạo đức.

- Khuyến khích việc nâng cao ý thức đạo đức trong quá trình thực hiện nhiệm vụ của mình. Đồng thời cũng nhắc nhở các chuyên gia về trách nhiệm và nghĩa vụ của họ khi thực thi công việc, tránh bị những cám dỗ thỏa hiệp để đáp ứng những áp lực kinh doanh ngày càng lớn.

- Các quy tắc đạo đức cũng xác định các hành vi có thể chấp nhận và không thể chấp nhận để hướng dẫn các chuyên gia khi họ làm việc với các đối tác khác. Các quy tắc đạo đức cũng có thể đưa ra những quy định nghiêm ngặt đòi hỏi các cá nhân trong tổ chức phải tuân thủ. Nếu vi phạm họ có thể bị phạt hoặc mất quyền hành nghề. Điều này cũng có thể tồn tại trong lĩnh vực CNTT.

- Tăng cường sự tin tưởng và tôn trọng của cộng đồng. Các quy tắc đạo đức nói chung được xây dựng trên kỳ vọng rằng một chuyên gia trong lĩnh vực đó sẽ thực hiện công việc của họ theo các quy chuẩn đạo đức.

CÂU HỎI ÔN TẬP

1. Khủng hoảng phần mềm là gì? Vì sao khủng hoảng phần mềm lại dẫn tới sự ra đời của ngành công nghệ phần mềm?

2. Hãy trình bày những khái niệm cơ bản của lĩnh vực công nghệ phần mềm.

3. Các chuyên gia CNTT cần phải quản lý những mối quan hệ nào trong quá trình làm việc? Những mối quan hệ này ảnh hưởng như thế nào đối với việc tuân thủ các quy tắc đạo đức nghề nghiệp.

4. Các tổ chức CNTT chuyên nghiệp có vai trò gì trong việc hỗ trợ các chuyên gia CNTT thực hành các quy tắc đạo đức nghề nghiệp?

Chương 2: TIẾN TRÌNH PHẦN MỀM

Tiến trình phần mềm là một tập hợp các hoạt động nhằm kiểm soát quá trình sản xuất phần mềm. Tiến trình phần mềm rất phức tạp, mang tính sáng tạo, dựa trên các quyết định và sự đánh giá của con người. Do nó mang tính sáng tạo và phê bình, nên những nỗ lực để có thể sản xuất phần mềm một cách tự động vẫn còn rất hạn chế. Các công cụ trợ giúp việc sản xuất phần mềm cũng chỉ có thể hỗ trợ một vài hoạt động trong các tiến trình này. Nhìn chung, việc tự động hóa trong quá trình sản xuất phần mềm vẫn còn là một điều xa vời.

Mặc dù có rất nhiều tiến trình phần mềm khác nhau, nhưng tựu chung lại, chúng đều có một số hoạt động cơ bản, chung cho tất cả các tiến trình:

- ***Đặc tả phần mềm:*** chức năng và các ràng buộc phải được xác định.

- ***Thiết kế và thực thi:*** phần mềm đáp ứng đúng bản đặc tả phải được tạo ra.

- ***Xác nhận – kiểm thử phần mềm:*** phần mềm phải được xác nhận để đảm bảo rằng nó thực hiện được những gì mà khách hàng mong muốn.

- ***Cải tiến – bảo trì phần mềm:*** phần mềm phải được cải tiến để đáp ứng được những yêu cầu thay đổi của khách hàng.

Mục tiêu chính của chương này là tóm tắt các hoạt động cơ bản của một tiến trình phần mềm và giới thiệu một số mô hình tiến trình phần mềm cơ bản được sử dụng trong phát triển phần mềm.

2.1. MÔ HÌNH TIẾN TRÌNH PHẦN MỀM

Mô hình tiến trình phần mềm là một cách trìu tượng hóa (mô hình hóa) một tiến trình phần mềm. Mỗi mô hình tiến trình đưa ra một tiến trình trong một tình huống cụ thể, sau đó sẽ cung cấp một phần thông tin về tiến trình để nhóm phát triển có thể sử dụng để phát triển sản phẩm phần mềm. Trong phần này, chúng ta cùng tìm hiểu một số mô hình tiến trình chung nhất, chúng có thể được xem như framework của tiến trình nhưng không có những mô tả chi tiết về các hoạt động cụ thể.

Những mô hình chung này không phải là những mô tả rõ ràng, cụ thể về các tiến trình phần mềm mà có thể được sử dụng để giải thích cho các cách tiếp cận khác nhau trong phát triển phần mềm. Có 3 mô hình cơ bản:

Mô hình thác nước: mô hình này lấy cơ sở là các hoạt động của tiến trình phần mềm: đặc tả, phát triển, kiểm thử và thay đổi. Các hoạt động này được coi như các giai đoạn tách biệt nhau.

Phát triển tiến hóa: cách tiếp cận này tiến hành xen kẽ các hoạt động đặc tả, phát triển, xác nhận và kiểm thử (validation). Một hệ thống ban đầu sẽ được nhanh chóng phát triển từ những đặc tả trìu tượng, phiên bản này sau đó được khách hàng xem xét đánh giá để sinh ra sản phẩm phù hợp với nhu cầu của khách hàng.

Công nghệ phần mềm hướng thành phần: cách tiếp cận này dựa trên một tập hợp các thành phần đã tồn tại và có thể tái sử dụng được. Tiến trình phát triển hệ thống tập trung vào việc tích hợp các thành phần có sẵn để xây dựng thành hệ thống chứ không chú trọng vào việc phát triển sản phẩm từ đầu.

Đây là 3 mô hình tiến trình chung được sử dụng rộng rãi trong thực tế phát triển phần mềm. Chúng thường không được áp dụng theo kiểu độc lập và duy nhất, mà thường sử dụng kết hợp với nhau, đặc biệt là khi phát triển các hệ thống lớn. Ví dụ như trong tiến trình phát triển phần mềm RUP, chúng ta sẽ nhận thấy nó kết hợp các thành phần của tất cả các mô hình này. Các hệ thống con bên trong một hệ thống lớn có thể được phát triển bằng các cách tiếp cận khác nhau. Tuy nhiên, chúng ta sẽ tìm hiểu từng mô hình một cách độc lập và chúng ta cần hiểu rõ để có thể kết hợp chúng với nhau trong thực tế.

Ngoài những mô hình tiến trình chung này, ở một số tổ chức, người ta còn có thể lựa chọn một cách tiếp cận khác, đó là việc sử dụng các ngôn ngữ hình thức để đặc tả phần mềm, sau đó biến đổi bản đặc tả này thành mã nguồn có thể thực thi được.

Một ví dụ điển hình của tiến trình phát triển hình thức là tiến trình Cleanroom, bắt nguồn từ IBM. Trong tiến trình Cleanroom, mỗi thành phần của phần mềm được đặc tả bằng ngôn ngữ hình thức (B, Z, OCL…) và sau đó, bản đặc tả này được biến đổi thành mã nguồn.

Việc sử dụng những ngôn ngữ hình thức là rất cần thiết đối với các hệ thống an toàn quan trọng, những hệ thống đòi hỏi độ tin cậy, an toàn cao. Cách tiếp cận hình thức có thể chứng minh được bằng các công cụ toán học để đảm bảo rằng những yêu cầu đưa ra là đúng, đầy đủ và thống nhất. Tuy nhiên, những tiến trình dựa trên việc biến đổi hình thức không được sử dụng rộng rãi, vì chúng đòi hỏi phải có những chuyên gia, trong thực tế, với những hệ thống cơ bản, tiến trình này thường không hiệu quả về mặt kinh phí cũng như chất lượng so với những tiến trình khác.

2.1.1. Mô hình thác nước

Mô hình tiến trình phát triển phần mềm được ra đời đầu tiên là mô hình thác nước, như trong hình minh họa 2.1 (Royce, 1970). Những giai đoạn cơ bản trong mô hình này được ánh xạ vào các hoạt động phát triển cơ bản:

Phân tích và xác định yêu cầu: các dịch vụ của hệ thống, các mục tiêu được đưa ra thông qua việc trao đổi và thống kê với người sử dụng hệ thống. Sau đó chúng được định nghĩa một cách chi tiết và được xem là một bản đặc tả hệ thống.

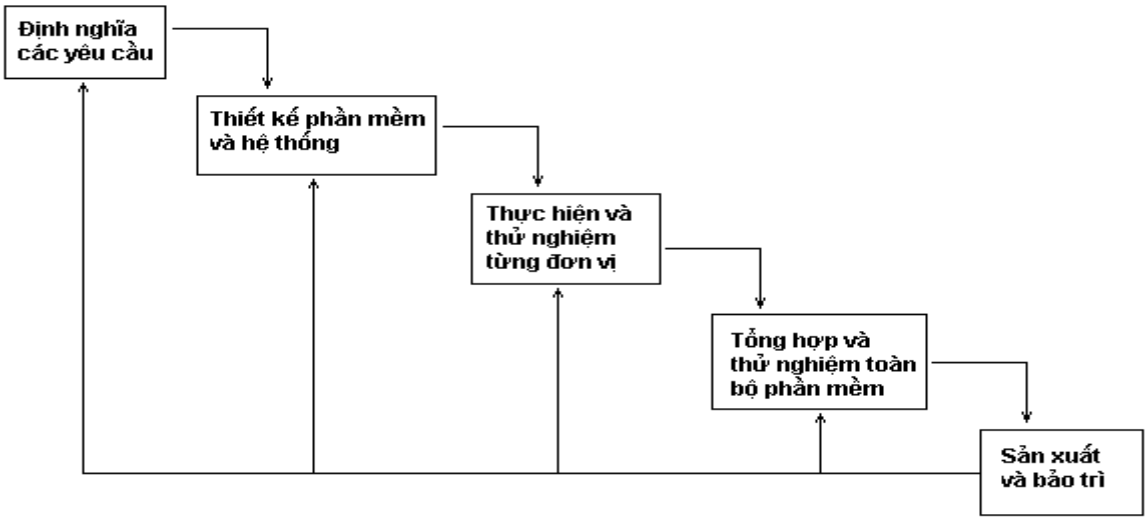
Thiết kế phần mềm và hệ thống: quá trình thiết kế sẽ tiến hành phân vùng hệ thống cho các yêu cầu phần cứng hoặc phần mềm, nó sinh ra kiến trúc hệ thống. Thiết kế phần mềm liên quan đến việc xác định, mô tả các thành phần trìu tượng của hệ thống và mối quan hệ giữa chúng.

Thực hiện và kiểm thử đơn vị: trong bước này, bản thiết kế phần mềm sẽ được cụ thể hóa bằng các ngôn ngữ lập trình để phát triển thành chương trình hoặc các đơn vị chương trình có thể cài đặt trong các hệ thống máy tính. Việc kiểm thử đơn vị liên quan tới việc xác nhận rằng mỗi đơn vị chương trình đã được phát triển theo đúng đặc tả.

Tích hợp và kiểm thử hệ thống: các đơn vị chương trình riêng rẽ hoặc các chương trình được tích hợp lại với nhau và được kiểm thử như một hệ thống hoàn chỉnh để đảm bảo rằng yêu cầu phần mềm đã được thỏa mãn. Sau khi kiểm thử, hệ thống phần mềm được bàn giao tới khách hàng.

Triển khai cài đặt và bảo trì: thông thường, đây là một giai đoạn khá dài. Phần mềm được cài đặt và đưa vào sử dụng. Việc bảo trì phần mềm liên quan tới việc sửa những lỗi không

được phát hiện sớm trong quy trình làm phần mềm, nâng cấp các đơn vị của hệ thống và từ đó có thể bổ xung thêm các dịch vụ mới để đáp ứng được yêu cầu thay đổi.



Hình 2.1. Mô hình thác nước

Trong mô hình, kết quả của mỗi giai đoạn là một hoặc nhiều tài liệu và giai đoạn tiếp theo chỉ được bắt đầu khi giai đoạn trước đã kết thúc. Nhưng trong thực tế, các giai đoạn này có thể chồng lên nhau và sinh ra những thông tin phản hồi về giai đoạn trước đó. Ví dụ: trong quá trình thiết kế, những vấn đề về yêu cầu sẽ được xác định. Trong quá trình lập trình, những vấn đề về thiết kế có thể sẽ được bộc lộ… Tiến trình phần mềm không chỉ đơn giản là một đường thẳng mà nó là một thứ tự có lặp của các hoạt động phát triển.

Nhược điểm của mô hình thác nước:

Chi phí sản xuất và cải tiến các tài liệu, lặp lại các hoạt động phát triển phần mềm thường tốn kém và coi như làm mới một khối lượng công việc đáng kể. Tuy nhiên, sau một số bước lặp nhỏ, thông thường một số phần của tiến trình phát triển sẽ được cố định (đóng băng), chẳng hạn như việc đặc tả hoàn thành, sau đó tiếp tục với các giai đoạn phát triển tiếp theo. Một số vấn đề có thể được để lại giải quyết sau, hoặc bỏ qua.

Giai đoạn cuối cùng của vòng đời phần mềm là cài đặt và bảo trì. Sau khi đưa phần mềm vào sử dụng, lỗi và những điểm thiếu sót trong yêu cầu phần mềm gốc sẽ được phát hiện, chương trình và những lỗi thiết kế hiện ra hoặc chức năng mới cần được bổ sung. Để thực hiện những thay đổi này, ta cần phải lặp lại các giai đoạn trước đó trong tiến trình.

Ưu điểm của mô hình thác nước:

Dễ phân công công việc, kiến trúc hệ thống ổn định, tài liệu được sinh ra sau mỗi bước và phù hợp với các mô hình tiến trình công nghệ khác. Vấn đề cơ bản của nó là tính mềm dẻo và linh hoạt. Đôi khi, có những thay đổi hoặc những lỗi phần mềm yêu cầu nhóm phát triển phải làm lại gần như từ đầu.

Ứng dụng của mô hình thác nước:

Mô hình này chỉ nên được sử dụng khi các yêu cầu hệ thống đã được xác định một cách rõ ràng và đầy đủ ngay từ đầu. Ngoài ra nó còn có thể được ứng dụng trong các dự án công nghệ khác. Ngày nay, tiến trình phần mềm dựa trên cách tiếp cận này vẫn còn được sử dụng cho việc phát triển phần mềm, đặc biệt với những dự án phần mềm là một phần của dự án công nghệ lớn hoặc những hệ thống an toàn quan trọng.

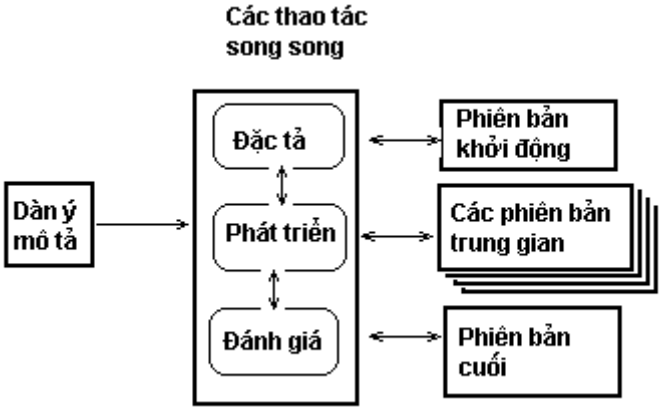
2.1.2. Phát triển tiến hóa

Phát triển tiến hóa dựa trên ý tưởng phát triển một phiên bản đầu tiên, chuyển tới người sử dụng và chờ những góp ý, sau đó phiên bản này tiếp tục được chỉnh sửa cho tới khi hệ thống hoàn chỉnh. Các hoạt động đặc tả, phát triển, kiểm định được lặp đi lặp lại nhiều lần hơn là những hoạt động riêng rẽ, những phản hồi được đáp ứng một cách nhanh chóng qua các giai đoạn.

Phát triển tiến hóa có hai mô hình:

Mô hình phát triển mở rộng: Khi mục tiêu của tiến trình là làm việc với khách hàng để tìm hiểu yêu cầu người dùng và đưa ra hệ thống cuối cùng thì việc phát triển phần mềm sẽ được bắt đầu bằng một phần của hệ thống mà nhóm phát triển đã hiểu cặn kẽ. Tiếp theo sẽ thêm những tính năng mới hoặc thay đổi các tính năng đã có theo yêu cầu của khách hàng.

Mô hình làm mẫu: mục tiêu của việc phát triển là xây dựng phần mềm phiên bản đầu tiên để hiểu yêu cầu của khách hàng, từ đó đưa ra được những yêu cầu hệ thống tốt nhất. Phần mềm khởi đầu tập trung vào những yêu cầu chưa rõ ràng. Thực chất của mô hình này là làm rõ yêu cầu của khách hàng để có được bản đặc tả chi tiết hơn trước khi bắt tay vào phát triển phần mềm. Sau giai đoạn này, nhóm phát triển có thể lựa chọn một mô hình hoàn toàn khác để xây dựng phần mềm (chẳng hạn như mô hình thác nước hoặc mô hình tiến hóa).



Hình 2.2. Mô hình phát triển phần mềm theo kiểu tiến hóa

Ưu điểm của mô hình:

Cách tiếp cận theo mô hình tiến hóa thường hiệu quả hơn mô hình thác nước vì thông thường nó đáp ứng được ngay lập tức nhu cầu của khách hàng. Ưu điểm của mô hình tiến hóa là có thể phát triển một cách tăng dần. Khi người sử dụng hiểu rõ hơn vấn đề của họ thì họ có thể phản ánh rõ hơn hệ thống phần mềm.

Nhược điểm của mô hình:

Tiến trình phần mềm không rõ ràng: người quản lý cần phải có những sản phẩm bàn giao thường xuyên để đánh giá tiến trình. Nếu hệ thống được phát triển một cách nhanh chóng, việc sử dụng kinh phí cho việc đưa ra các tài liệu cho từng phiên bản là tốn kém và không hiệu quả.

Hệ thống thường có tính cấu trúc yếu: những thay đổi thường xuyên sẽ làm phá vỡ tính cấu trúc của chương trình. Việc kết hợp chặt chẽ những thay đổi sẽ trở nên rất khó khăn và tốn kém.

Ứng dụng của mô hình:

Với những hệ thống vừa và nhỏ (dưới 500.000 dòng mã nguồn), đây là cách tiếp cận tốt nhất để phát triển. Vấn đề của phát triển tiến hóa chỉ xuất hiện với những hệ thống lớn, phức tạp và có thời gian phát triển dài, hay khi các nhóm khác nhau phát triển từng thành phần của hệ thống. Khi đó khó có thể đưa ra một kiến trúc hệ thống ổn định và khó có thể tích hợp các thành phần được thực hiện bởi các nhóm khác nhau.

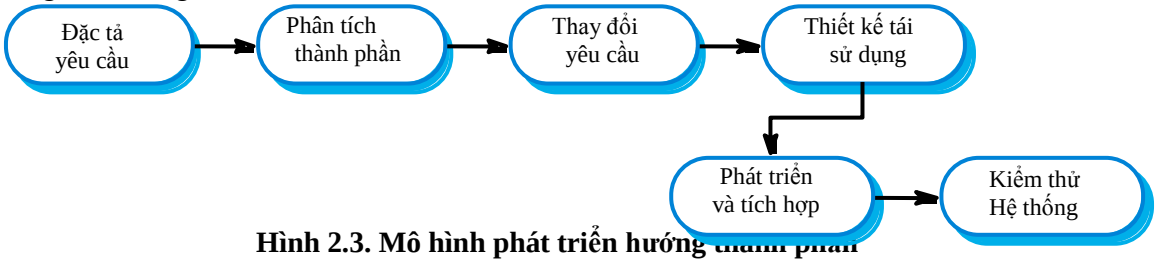
Với những hệ thống lớn cần phải sử dụng những tiến trình kết hợp để tận dụng được ưu điểm của cả mô hình thác nước và mô hình tiến hóa. Ví dụ như ban đầu ta có thể sử dụng mô hình mẫu để hiểu rõ ràng và chắc chắn yêu cầu phần mềm. Sau đó bạn có thể thực thi hệ thống lại bằng cách tiếp cận có cấu trúc hơn. Một số phần của hệ thống đã được hiểu rõ ràng có thể được đặc tả và phát triển dựa trên cách tiếp cận của mô hình thác nước. Những phần khác của hệ thống, chẳng hạn như giao diện người dùng, có thể sử dụng cách tiếp cận phát triển mở rộng.

2.1.3. Công nghệ phần mềm hướng thành phần

Trong phần lớn các dự án phần mềm, có một số thành phần có thể tái sử dụng. Điều này thường xảy ra một cách không chính thức khi nhóm phát triển làm việc trên những dự án mà người ta biết về thiết kế hoặc mã nguồn có những yêu cầu tương tự. Họ nhìn vào đó, thay đổi chúng cho phù hợp với hệ thống mới. Trong cách tiếp cận tiến hóa được mô tả ở phần trên, tái sử dụng thường được áp dụng đối với việc phát triển hệ thống một cách nhanh chóng.

Việc tái sử dụng một cách không chính thức thường không phân biệt tiến trình phát triển được sử dụng một cách rõ ràng. Tuy nhiên, trong vài năm gần đây, một cách tiếp cận mới được gọi là công nghệ phần mềm dựa trên thành phần (component), dựa vào việc tái sử dụng, đã nổi lên và được sử dụng một cách rộng rãi.

Cách tiếp cận tái sử dụng dựa trên việc sử dụng một lượng lớn các thành phần phần mềm có sẵn và một số framework tích hợp cho các thành phần này. Đôi khi, những thành phần này là những hệ thống, thực hiện một số chức năng nào đó. Mô hình phần mềm hướng thành phần được mô phỏng như trong hình vẽ 2.3.



Hình 2.3. Mô hình phát triển hướng thành phần

Trong khi những yêu cầu phần mềm đầu tiên được xây dựng và việc xác nhận phần mềm được tiến hành song song với các giai đoạn khác trong tiến trình. Những giai đoạn trung gian trong tiến trình dựa trên tái sử dụng có thể rất khác nhau. Những bước này có thể là:

Phân tích thành phần: bản đặc tả yêu cầu được đưa ra, việc tìm kiếm các thành phần có thể đáp ứng được những yêu cầu này. Thông thường, không có sự thích hợp tuyệt đối giữa các yêu cầu và thành phần có sẵn, mà các thành phần này chỉ cung cấp một phần chức năng được yêu cầu.

Cải tiến yêu cầu: trong giai đoạn này, những yêu cầu được phân tích và sử dụng những thông tin về các thành phần đã được khám phá. Những yêu cầu này sau đó thường được thay đổi để thích hợp với những thành phần đã có. Khi những thay đổi này không thể thực hiện được, hoạt động phân tích thành phần có thể được thực hiện lại để tìm ra một giải pháp thích hợp.

Thiết kế hệ thống tái sử dụng: trong giai đoạn này, framework của hệ thống được thiết kế hoặc những framework đã có sẵn được tái sử dụng. Người làm thiết kế có tính đến các thành phần tái sử dụng và tổ chức framework để phục vụ điều này. Một số phần mềm mới có thể được thiết kế nếu các thành phần có thể tái sử dụng không sẵn sàng.

Phát triển và tích hợp: phần mềm không thể mua được từ bên ngoài sẽ được nhóm phát triển, những thành phần và các hệ thống COST được tích hợp lại tạo thành một hệ thống mới. Việc tích hợp hệ thống trong mô hình này có thể là một phần của tiến trình phát triển.

Công nghệ phần mềm dựa trên việc tích hợp các thành phần có ưu điểm cơ bản là việc giảm bớt công sức phát triển, do đó giá thành sản phẩm cũng thấp hơn, ít rủi ro hơn. Thông thường, sản phẩm cũng sớm được bàn giao tới khách hàng. Tuy nhiên, việc thỏa hiệp giữa các yêu cầu là không thể tránh khỏi, điều này có thể dẫn tới tình trạng hệ thống sẽ không thực sự phù hợp với yêu cầu thực của khách hàng. Hơn nữa, một số điều khiển của hệ thống là không thích hợp khi các thành phần sử dụng các phiên bản khác nhau. Do đó, phương pháp này chỉ thích hợp cho việc phát triển hệ thống dựa trên tích hợp các dịch vụ Web của một nhóm các nhà cung cấp.

2.2. TIẾN TRÌNH LẶP

Sự thay đổi là điều không thể tránh khỏi trong những dự án phần mềm lớn. Những yêu cầu hệ thống thay đổi khi hệ thống không đáp ứng những áp lực từ bên ngoài, các kỹ thuật mới sẵn sàng, việc thiết kế và thực thi thay đổi. Điều đó có nghĩa là tiến trình phần mềm không phải là một tiến trình một lần, hơn thế nữa, các hoạt động trong tiến trình thường lặp đi lặp lại để đáp ứng những yêu cầu thay đổi. Vì thế việc phát triển lặp là cơ sở của sản xuất phần mềm. Phần này đề cập đến 2 mô hình tiến trình cơ bản được thiết kế riêng cho tiến trình lặp.

Mô hình gia tăng (incremental): việc đặc tả, thiết kế và thực thi phần mềm được chia thành một loạt các thành phần, mỗi thành phần được phát triển một lượt và tích hợp vào hệ thống chung sau khi hoàn thành.

Phát triển xoắn ốc: việc phát triển phần mềm tuân theo một vòng xoắn ốc, từ những yêu cầu sơ lược đầu tiên đến hệ thống được phát triển hoàn thiện cuối cùng.

Ưu điểm của các tiến trình lặp là việc đặc tả và phát triển hệ thống có thể thực hiện song song với nhau. Tuy nhiên, điều này cũng có thể dẫn đến một số xung đột, đặc biệt là trong trường

hợp có nhiều khách hàng cùng yêu cầu một sản phẩm. Khi đó, việc thỏa thuận xem thành phần nào sẽ được phát triển trước cũng là một vấn đề đáng được quan tâm.

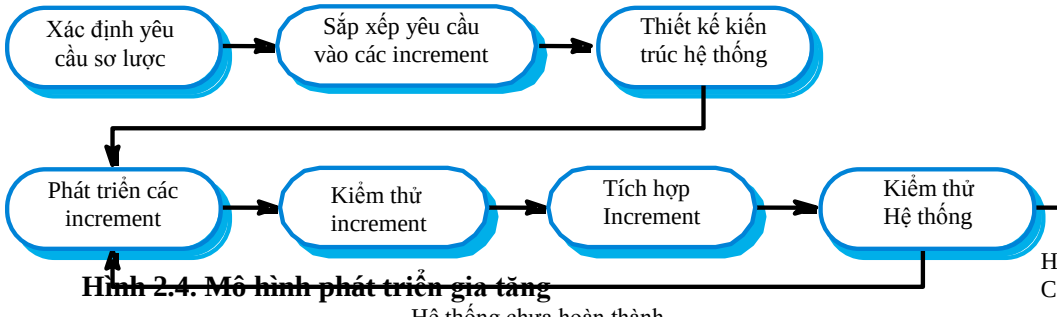
2.2.1. Mô hình gia tăng

Trong mô hình thác nước, những yêu cầu của khách hàng phải được xác định gần như hoàn chỉnh ngay từ đầu, trước khi tiến hành thiết kế, ngay cả việc thiết kế cũng phải được hoàn thành trước khi bắt tay vào xây dựng mã nguồn. Những thay đổi yêu cầu buộc ta phải thực hiện lại hầu hết các bước, từ đặc tả yêu cầu đến thiết kế và thực thi. Tuy nhiên, việc phân chia giữa thiết kế và thực thi cần phải được kiểm soát thông qua các hệ thống được tài liệu hóa một cách rõ ràng và dễ thay đổi. Ngược lại, trong cách tiếp cận tiến hóa, nó cho phép các yêu cầu và các quyết định thiết kế được trì hoãn nhưng cũng vì thế mà phần mềm trở nên kém tính cấu trúc, khó hiểu và khó bảo trì.

Mô hình gia tăng (hình 2.4) là một cách tiếp cận kết hợp những ưu điểm của cả hai mô hình: mô hình thác nước và mô hình tiến hóa. Trong tiến trình phát triển tăng dần, khách hàng xác định một cách sơ lược các dịch vụ được cung cấp bởi hệ thống. Họ sẽ xác định thành phần nào quan trọng nhất, thành phần nào ít quan trọng hơn với họ. Một số thành phần (increments) được xác định cung cấp một hệ thống con các chức năng hệ thống. Việc xác định các dịch vụ được đưa vào các thành phần phụ thuộc vào việc những dịch vụ nào được ưu tiên phát triển trước.

Khi một thành phần của hệ thống được xác định, những yêu cầu cho những dịch vụ được triển khai đầu tiên trong thành phần sẽ được xác định chi tiết trước khi phát triển. Trong quá trình phát triển, việc phân tích yêu cầu cho các thành phần tiếp theo sẽ được thực hiện, nhưng những yêu cầu thay đổi cho thành phần hiện tại sẽ không được chấp nhận.

Khi một thành phần hoàn thành và được bàn giao, khách hàng có thể đưa vào khai thác, điều này có nghĩa là các thành phần đã hoàn thành là một phần của hệ thống. Việc sử dụng chúng có thể cho ta những kinh nghiệm nhất định để phát triển các thành phần tiếp theo cũng như cho phiên bản cuối cùng của thành phần hiện tại. Khi một thành phần mới được hoàn thành, chúng được tích hợp với các thành phần đã có, vì thế các chức năng của hệ thống cũng dần được hoàn thiện trước khi được bàn giao chính thức.



Hình 2.4. Mô hình phát triển gia tăng

Ưu điểm của mô hình:

Khách hàng không phải đợi cho đến khi hệ thống hoàn thiện, họ có thể thu được lợi nhuận ngay khi phần mềm đang được triển khai. Thành phần đầu tiên sẽ làm hài lòng hầu hết những yêu cầu quan trọng, vì thế họ có thể sử dụng hệ thống ngay lập tức.

Khách hàng có thể sử dụng sớm các thành phần như là các phiên bản mẫu và có được những kinh nghiệm cần thiết để diễn tả các yêu cầu của mình trong các thành phần tiếp theo. Do đó, thường có ít rủi ro hơn khi dự án gặp lỗi. Mặc dù một số vấn đề có thể không được tính đến

trong một số thành phần, vẫn có khả năng một số thành phần khác được bàn giao thành công tới khách hàng.

Khi các dịch vụ ưu tiên cao nhất được phát triển trước, các thành phần sau đó được tích hợp với chúng, do đó những thành phần quan trọng nhất sẽ được kiểm thử nhiều lần nhất. Điều này có nghĩa là khách hàng sẽ tránh được những lỗi phần mềm không kiểm soát được ở những phần quan trọng nhất của hệ thống.

Nhược điểm của mô hình:

Tuy nhiên, cũng có những vấn đề đối với việc bàn giao gia tăng. Các thành phần phải tương đối nhỏ (dưới 20,000 dòng mã nguồn) và mỗi thành phần bao gồm một số chức năng của hệ thống. Do đó, đôi khi khó kết hợp những yêu cầu của khách hàng với các thành phần có kích thước hợp lý. Hơn nữa, hầu hết các hệ thống đều yêu cầu một tập hợp các chức năng cơ bản được sử dụng trong các phần khác nhau, khi các yêu cầu không được xác định một cách chi tiết cho đến khi thành phần được thực thi thì nhóm phát triển khó có thể xác định được các chức năng chung mà mọi thành phần đều cần đến.

2.2.2. Mô hình xoắn ốc

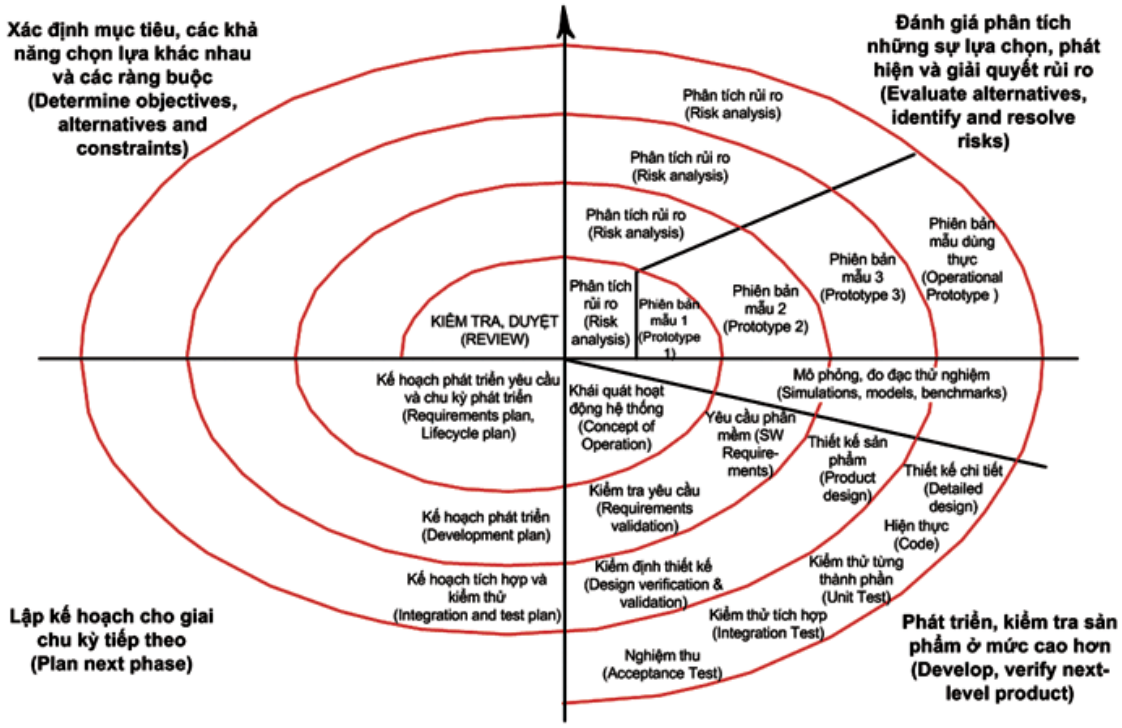
Mô hình xoắn ốc của tiến trình phần mềm được đưa ra bởi Boehm (1988). Trong mô hình này, các hoạt động không phải là các hoạt động tuần tự, mà tiến trình phát triển phần mềm vòng theo một đường xoắn ốc. Mỗi vòng lặp trong hình xoắn là một giai đoạn trong tiến trình phần mềm (hình 2.5). Vòng trong cùng thường là các hoạt động nghiên cứu tính khả thi, vòng tiếp theo là xác định yêu cầu, rồi đến thiết kế, thực thi… Mỗi vòng được chia thành 4 phần:

Thiết lập mục tiêu: các mục tiêu cụ thể của một giai đoạn trong dự án được xác định. Các ràng buộc đối với tiến trình và sản phẩm được xác định, bản kế hoạch quản lý chi tiết được đưa ra. Những rủi ro đối với dự án được xác định. Các chiến lược xen kẽ, phụ thuộc vào những rủi ro này có thể được lập kế hoạch.

Đánh giá và giảm thiểu rủi ro: với mỗi rủi ro đã được xác định, người quản lý dự án sẽ đưa ra một bản phân tích chi tiết. Các bước được thực hiện để giảm thiểu rủi ro. Ví dụ: nếu có một rủi ro là yêu cầu không hợp lý, nhóm phát triển có thể xây dựng một phần mềm mẫu để thử nghiệm.

Phát triển và kiểm thử (validation): sau khi đã giải quyết rủi ro, nhóm phát triển sẽ lựa chọn một mô hình thích hợp. Ví dụ: nếu những rủi ro về giao diện người dùng được coi là nhiều hơn, mô hình phát triển thích hợp có thể là mô hình mẫu (prototype). Nếu rủi ro về tính an toàn được xem xét, ta có thể sử dụng mô hình phát triển biến đổi hình thức. Mô hình thác nước có thể thích hợp nhất nếu những rủi ro được xác định là sẽ nảy sinh khi tích hợp các hệ thống con.

Lập kế hoạch: dự án phát triển phần mềm sẽ được xem xét và quyết định về những vấn đề sẽ được thực hiện trong vòng xoắn tiếp theo. Nếu quyết định tiếp tục thực hiện dự án, sẽ xây dựng kế hoạch cho vòng xoắn tiếp theo.



Hình 2.5. Mô hình xoắn ốc

Ưu điểm của mô hình xoắn ốc:

Ưu điểm lớn nhất của mô hình xoắn ốc là khả năng phân tích đánh giá rủi ro được đẩy lên như một phần thiết yếu trong mỗi vòng lặp (spiral) để tăng mức độ tin cậy của dự án. Mô hình này cũng kết hợp được những tính chất tốt nhất của mô hình thác nước và mô hình tiến hóa. Hơn nữa, cho phép thay đổi mô hình thực thi tùy theo điều kiện thực tế dự án tại mỗi vòng lặp.

Đây chính là mô hình tổng quát nhất, tất cả các mô hình khác đều có thể xem là một hiện thực của mô hình tổng quát này, hay cũng có thể xem nó là mô hình tổng hợp các mô hình khác. Đặc biệt, nó được ứng dụng không chỉ trong phát triển phần mềm mà còn trong phát triển phần cứng.

Nhược điểm của mô hình xoắn ốc:

Mô hình này tuy có nhiều ưu điểm nhưng nó khá phức tạp khi triển khai, do đó không phù hợp cho những dự án nhỏ với ít rủi ro. Nếu đội dự án không có kỹ năng phân tích rủi ro tốt thì việc áp dụng mô hình này cũng không hiệu quả.

Do đó, mô hình này chỉ thích hợp với những dự án lớn có nhiều rủi ro hay sự thành công của dự án không có được sự đảm bảo nhất định, những dự án đòi hỏi nhiều tính toán, xử lý như các hệ thống hỗ trợ quyết định. Bên cạnh đó, đội ngũ thực hiện dự án phải có khả năng phân tích rủi ro tốt.

2.3. CÁC HOẠT ĐỘNG TRONG TIẾN TRÌNH

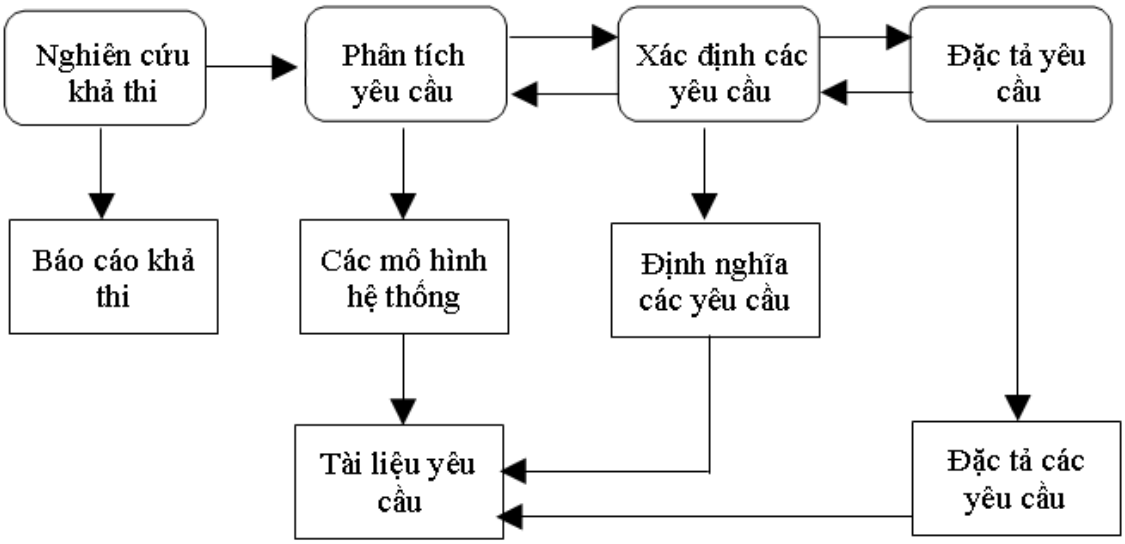
Bốn hoạt động cơ bản: đặc tả, phát triển, kiểm thử và cải tiến được tổ chức một cách khác nhau trong các tiến trình phát triển khác nhau. Trong mô hình thác nước, các hoạt động này được tổ chức một cách tuần tự, trong khi đó, với mô hình phát triển tiến hóa, các hoạt động này lại

được tổ chức một cách xen kẽ. Tóm lại, các hoạt động này được thực hiện như thế nào phụ thuộc vào kiểu phần mềm, tình hình nhân sự và cấu trúc của tổ chức. Ở đây người ta không xem xét tính đúng sai trong cách tổ chức, mà mục tiêu của phần này là cung cấp cho bạn những hướng dẫn cơ bản để tổ chức các hoạt động một cách hiệu quả nhất.

2.3.1. Đặc tả phần mềm

Đặc tả phần mềm hoặc kỹ nghệ yêu cầu là một tiến trình để hiểu và xác định những dịch vụ cần có trong hệ thống cũng như xác định những ràng buộc đối với việc phát triển và các chức năng của hệ thống. Công nghệ yêu cầu là một trong những giai đoạn rất quan trọng trong tiến trình phần mềm, vì mỗi lỗi trong giai đoạn này sẽ không tránh khỏi việc dẫn đến những sai lầm trong các giai đoạn tiếp theo.

Tiến trình kỹ nghệ yêu cầu được chỉ ra trong hình 2.6. Tiến trình này sẽ sinh ra các tài liệu yêu cầu, đó là các bản đặc tả cho hệ thống. Các yêu cầu thường được trình bày ở 2 mức độ chi tiết khác nhau trong tài liệu. Khách hàng và người dùng cuối cần những mô tả yêu cầu ở mức cao, còn người phát triển hệ thống lại cần những đặc tả chi tiết hơn.



Hình 2-6. Các giai đoạn phân tích yêu cầu

Nghiên cứu tính khả thi: người ta sẽ ước lượng xem liệu những yêu cầu của người dùng có được thỏa mãn khi sử dụng các kỹ thuật phần cứng và phần mềm hiện có. Nghiên cứu sẽ xem xét liệu dự án có được kinh phí hợp lý, phù hợp với mục tiêu kinh doanh của tổ chức. Việc nghiên cứu tính khả thi được tiến hành nhanh và ít tốn kém. Kết quả đưa ra sẽ quyết định những công việc thực hiện tiếp theo và được phân tích một cách chi tiết hơn.

Phân tích và suy luận ra các yêu cầu: bước này sẽ chỉ ra những yêu cầu của hệ thống thông qua việc quan sát các hệ thống đang tồn tại, thảo luận với người dùng tiềm năng và người mua, người phân tích nhiệm vụ… Công việc này có thể liên quan tới việc phát triển một hoặc nhiều mô hình hệ thống và các phần mềm mẫu (prototype). Những phân tích này giúp người làm phân tích hiểu rõ hơn cách thức phá triển hệ thống.

Đặc tả yêu cầu: đây là công việc chuyển từ những thông tin thu thập được trong quá trình phân tích thành tài liệu chứa những yêu cầu. Có 2 kiểu yêu cầu trong tài liệu này: yêu cầu người

dùng cuối hoặc yêu cầu khách hàng, có mức trìu tượng cao hơn và yêu cầu hệ thống, mô tả chi tiết hơn các chức năng của hệ thống.

Xác nhận yêu cầu (validation): công việc này kiểm tra các yêu cầu xem nó có mang tính hiện thực, thống nhất và đầy đủ không. Trong quá trình đặc tả, những lỗi trong tài liệu yêu cầu là không thể tránh khỏi. Những lỗi này cần phải được khắc phục ngay lập tức để tránh dẫn đến lỗi trong các bước tiếp theo.

Tất nhiên, các hoạt động trong tiến trình xác định yêu cầu không được thực hiện một cách đơn giản theo một thứ tự đã định. Việc phân tích yêu cầu vẫn tiếp tục trong khi những yêu cầu mới xuất hiện. Các hoạt động nghiên cứu tính khả thi, phân tích và đặc tả yêu cầu được tiến hành xen kẽ nhau. Trong một số phương pháp phát triển phần mềm linh hoạt, chẳng hạn như phương pháp lập trình cực đại (Extreme Programming), các yêu cầu cũng được phát triển dần từng bước, thứ tự ưu tiên do người dùng xác định và người dùng cũng là một thành viên trong nhóm phát triển.

2.3.2. Thiết kế và thực thi phần mềm

Giai đoạn này liên quan tới việc chuyển những yêu cầu phần mềm thành những hệ thống có thể thực thi được. Thông thường nó liên quan đến việc thiết kế và lập trình. Nếu phát triển phần mềm theo phương pháp tiếp cận tiến hóa, thì việc thực thi phần mềm còn liên quan tới việc làm mịn các yêu cầu.

Thiết kế phần mềm là việc mô tả cấu trúc của phần mềm được thực thi, dữ liệu của hệ thống, giao diện giữa các thành phần, và đôi khi, đó là thuật toán sẽ được sử dụng. Người làm thiết kế cũng không phải ngay lập tức đưa ra được một bản thiết kế hoàn chỉnh, mà thông thường nó cũng phải được chỉnh sửa lặp đi lặp lại nhiều lần.

Tiến trình thiết kế có thể liên quan tới việc phát triển một vài mô hình của hệ thống ở những mức độ trìu tượng khác nhau. Khi thiết kế được phân tích, lỗi và các điểm thiếu sót ở các giai đoạn trước sẽ được bộc lộ, điều này cũng giúp cho việc chỉnh sửa các bản thiết kế trước đó.

Hình 2.7 chỉ ra một tiến trình để có thể đưa ra một bản thiết kế hoàn chỉnh. Sơ đồ này gợi ý rằng các bước trong tiến trình thiết kế là tuần tự. Nhưng trong thực tế, các bước trong tiến trình thiết kế cũng được thực hiện xen kẽ nhau. Những phản hồi của bước kế tiếp sẽ làm cơ sở để hoàn thiện những công việc đã thực hiện ở những bước trước đó. Một bản đặc tả sau mỗi bước là đầu ra của các hoạt động thiết kế.

Thiết kế kiến trúc: các hệ thống con tạo nên hệ thống chính và mối quan hệ giữa chúng được xác định và tài liệu hóa.

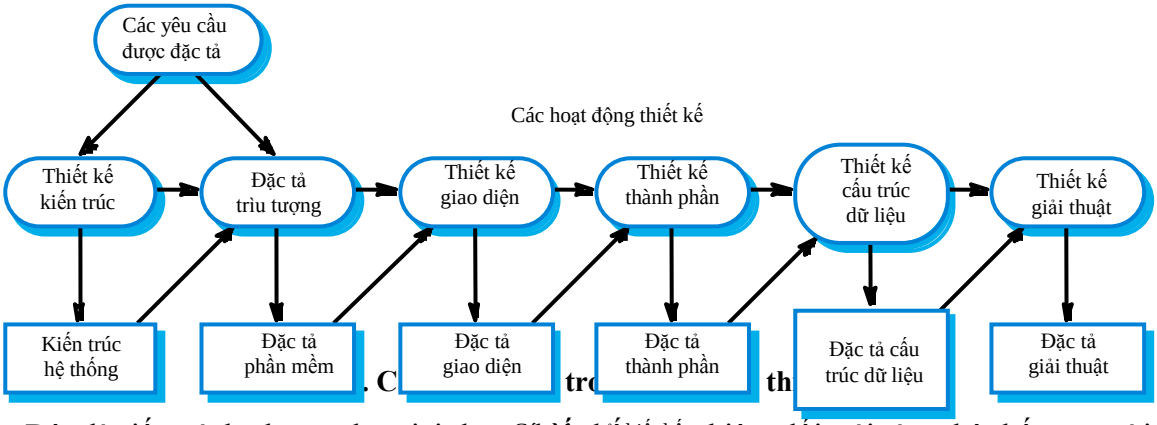
Đặc tả trìu tượng: với mỗi hệ thống con, người ta phải xây dựng một bản mô tả trìu tượng về các dịch vụ và những ràng buộc của nó.

Thiết kế giao diện: với mỗi hệ thống con, giao diện của nó với các hệ thống con khác được thiết kế và tài liệu hóa. Giao diện phải không nhập nhằng và cho phép các hệ thống khác sử dụng mà không cần biết các hoạt động bên trong.

Thiết kế thành phần: các dịch vụ được bố trí vào các thành phần và giao diện của các thành phần này được thiết kế.

Thiết kế cấu trúc dữ liệu: cấu trúc dữ liệu sử dụng trong hệ thống được xác định và thiết kế một cách chi tiết.

Thiết kế giải thuật: giải thuật được sử dụng trong các dịch vụ được xác định và thiết kế chi tiết.



. C tr th

Đây là tiến trình chung cho giai đoạn thiết kế, tuy nhiên, đối với từng hệ thống người ta lại có thể có những biến đổi cho phù hợp. Ví dụ: hai giai đoạn cuối trong tiến trình thiết kế là thiết kế cấu trúc dữ liệu và thiết kế giải thuật, người ta có thể thực hiện khi bắt tay vào lập trình. Nếu sử dụng cách tiếp cận mở rộng trong thiết kế, giao diện của hệ thống có thể thiết kế sau khi xác định rõ các cấu trúc dữ liệu. Giai đoạn đặc tả trìu tượng có thể bị bỏ qua, mặc dù nó rất quan trọng đối với các hệ thống đòi hỏi tính an toàn cao.

Khi phương pháp phát triển phần mềm linh hoạt được sử dụng, đầu ra của tiến trình thiết kế sẽ không phải là những tài liệu đặc tả riêng biệt, nhưng nó sẽ được đưa vào trong mã nguồn của chương trình. Sau khi thiết kế kiến trúc, các giai đoạn thiết kế sau được thực hiện theo kiểu gia tăng (incremental). Mỗi increment được xem như một đoạn mã nguồn của chương trình hơn là một mô hình thiết kế.

Ngược lại, với các tiếp cận theo phương thức cấu trúc, việc thiết kế chủ yếu dựa trên những mô hình đồ họa và trong một số trường hợp, người ta có thể sinh ra mã nguồn tự động từ các mô hình này. Phương pháp cấu trúc được phát triển từ những năm 70s để phục vụ cho việc thiết kế hướng chức năng, sau đó là hướng đối tượng và xuất hiện một số ngôn ngữ trợ giúp quá trình thiết kế (UML). Phương pháp cấu trúc bao gồm một mô hình tiến trình thiết kế, các khái niệm thiết kế, định dạng báo cáo, các quy tắc và hướng dẫn thiết kế. Các phương pháp cấu trúc cung cấp một phần hoặc toàn bộ các mô hình sau của hệ thống:

Mô hình đối tượng chỉ ra các lớp đối tượng được sử dụng trong hệ thống và mối liên hệ giữa chúng.

Mô hình tuần tự chỉ ra các đối tượng tương tác thế nào với hệ thống.

Mô hình trạng thái chỉ ra các trạng thái của hệ thống và quá trình biến đổi từ trạng thái này sang trạng thái khác.

Mô hình cấu trúc chỉ ra các thành phần của hệ thống và việc kết hợp các thành phần này lại với nhau.

Mô hình luồng dữ liệu chỉ ra cách thức hệ thống biến đổi dữ liệu trong quá trình xử lý. Thông thường mô hình này không sử dụng trong phương pháp hướng đối tượng nhưng nó vẫn

được sử dụng một cách tuần tự khi thiết kế các hệ thống kinh doanh và các hệ thống thời gian thực.

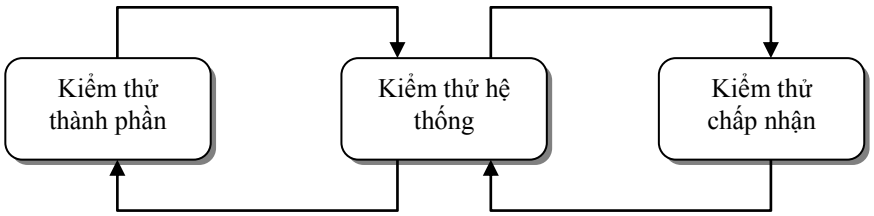
Lập trình là một hoạt động mang tính cá nhân, không có một tiến trình chung để tuân theo. Một số lập trình viên bắt đầu bằng những thành phần mà họ đã hiểu kỹ càng, sau đó mới làm việc với các thành phần mà họ chưa hiểu cặn kẽ. Cũng có cách tiếp cận khác, họ để lại những thành phần quen thuộc đến cuối, bởi họ đã biết phải thực thi chúng như thế nào. Một số lập trình viên lại thích xác định dữ liệu sớm trong tiến trình, và sau đó sử dụng những dữ liệu này để định hướng cho việc lập trình, một số lại để việc cấu trúc dữ liệu đến cuối…

Thông thường, các lập trình viên thực hiện việc kiểm thử phần mã nguồn mà họ đã phát triển. Thông thường nó liên quan đến những lỗi cần phải loại bỏ trong chương trình. Công việc này gọi là gỡ lỗi (debugging). Việc phát hiện và sửa lỗi là các tiến trình khác nhau. Kiểm thử liên quan tới việc phát hiện ra các lỗi còn tồn tại, gỡ lỗi liên quan tới việc xác định và sửa lỗi. Khi gỡ lỗi, Lập trình viên đưa ra giả thiết về cách thực thi mà họ có thể quan sát được, sau đó kiểm thử giả thiết này với hy vọng sẽ tìm ra được những lỗi gây ra những điểm bất thường ở đầu ra. Người lập trình cũng có thể viết những trường hợp kiểm thử (test case) mới để xác định vị trí của lỗi.

2.3.3. Thẩm định phần mềm

Thẩm định hay xác minh và thẩm định phần mềm (verification and validation) là việc chỉ ra rằng hệ thống phù hợp với đặc tả và đáp ứng được những mong đợi từ khách hàng. Nó liên quan tới việc kiểm tra các tiến trình, chẳng hạn như việc kiểm duyệt và xem xét tại mỗi giai đoạn trong tiến trình phần mềm, từ yêu cầu người dùng đến xây dựng chương trình.

Tuy nhiên, chi phí cơ bản của việc thẩm định được tính từ khi hệ thống chức năng được kiểm thử. Trừ những chương trình nhỏ, các hệ thống không nên được kiểm thử như một hệ thống đơn độc lập. Hình 2.8 chỉ ra một tiến trình kiểm thử ba giai đoạn: các thành phần của hệ thống được kiểm thử, hệ thống tích hợp được kiểm thử. Cuối cùng, hệ thống được kiểm thử với dữ liệu của khách hàng.



Hình 2.8. Tiến trình kiểm thử

Một cách lý tưởng, lỗi của các thành phần được phát hiện trong giai đoạn đầu của tiến trình và những vấn đề về giao diện sẽ được phát hiện khi hệ thống được tích hợp. Tuy nhiên, khi các lỗi được phát hiện, chương trình phải được sửa lỗi và điều này làm cho các giai đoạn khác trong quy trình kiểm thử phải được thực hiện lại. Những lỗi trong các thành phần chương trình có thể xuất hiện trong suốt quá trình kiểm thử. Tiến trình được lặp đi lặp lại kết hợp với những phản hồi từ các giai đoạn khác nhau.

Các giai đoạn trong tiến trình kiểm thử bao gồm:

Kiểm thử thành phần (đơn vị): các thành phần độc lập được kiểm thử để đảm bảo rằng chúng được thực thi đúng. Mỗi thành phần được kiểm thử độc lập, không có các thành phần hệ thống khác. Các thành phần (components) có thể là các thực thể đơn, chẳng hạn như các chức năng hoặc các lớp đối tượng, hoặc nó có thể là các nhóm đồng nhất của các thực thể này.

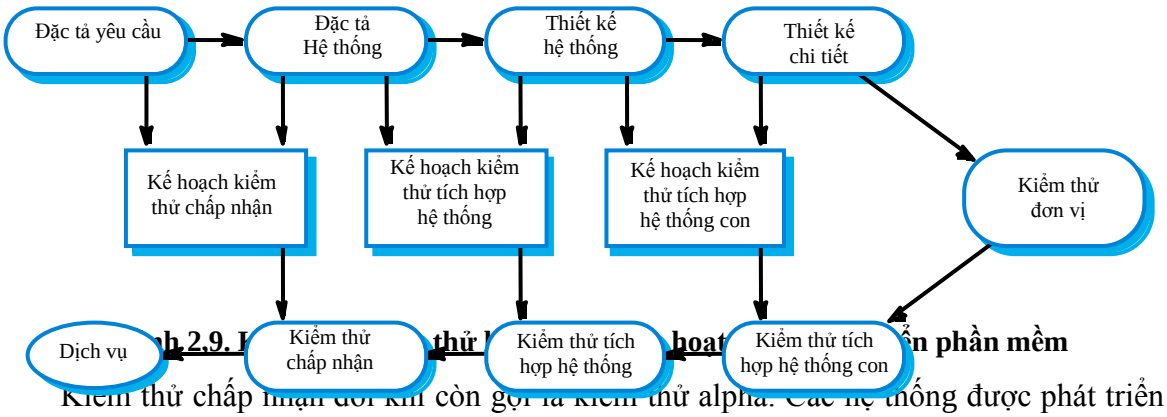
Kiểm thử hệ thống: các thành phần được tích hợp với nhau tạo thành hệ thống. Tiến trình này liên quan tới việc tìm ra các lỗi từ việc tương tác ngoài dự kiến giữa các thành phần và các vấn đề về giao diện giữa các thành phần. Nó cũng liên quan tới việc thẩm định hệ thống, xem hệ thống có đáp ứng được các yêu cầu chức năng và yêu cầu phi chức năng, kiểm tra những thuộc tính quan trọng của hệ thống. Đối với các hệ thống lớn, việc này có thể là một tiến trình bao gồm nhiều giai đoạn, khi các thành phần được tích hợp thành các hệ thống con, các hệ thống con này sau đó mới được tích hợp với nhau thành hệ thống hoàn chỉnh cuối cùng.

Kiểm thử chấp nhận: đây là giai đoạn cuối cùng trong tiến trình kiểm thử trước khi hệ thống được đưa vào sử dụng. Hệ thống được kiểm thử với dữ liệu do khách hàng cung cấp chứ không phải là những dữ liệu mô phỏng. Kiểm thử chấp nhận có thể phát hiện ra những lỗi hoặc những điểm còn thiếu sót khi xác định các yêu cầu hệ thống, bởi dữ liệu thực sẽ kiểm tra hệ thống khác so với dữ liệu kiểm thử. Kiểm thử hệ thống nói chung liên quan tới những vấn đề về yêu cầu, khi mà các chức năng của hệ thống không thực sự đáp ứng yêu cầu của người sử dụng hoặc tính hiệu năng của hệ thống không được chấp nhận.

Thông thường, việc phát triển thành phần và kiểm thử luân phiên nhau. Các lập trình viên thường kiểm thử phần mềm bằng dữ liệu của họ ngay sau khi lập trình xong. Đây là một cách tiếp cận nhạy cảm và hiệu quả về mặt kinh tế. Lập trình viên hiểu về thành phần mà họ đã xây dựng một cách rõ ràng nhất, vì thế họ là người tốt nhất để tạo ra các trường hợp kiểm thử.

Nếu hệ thống phát triển theo cách tiếp cận gia tăng, mỗi thành phần nên được kiểm thử ngay khi phát triển, việc kiểm thử này dựa trên những yêu cầu đối với từng thành phần. Trong phương pháp lập trình cực đại (XP), Nhóm phát triển sẽ xây dựng các trường hợp kiểm thử ngay khi làm đặc tả, trước khi tiến hành lập trình. Điều này giúp cho kiểm thử viên và người lập trình hiểu rõ hơn yêu cầu, đảm bảo rằng không có sự chậm trễ khi tạo ra các trường hợp kiểm thử.

Những giai đoạn kiểm thử cuối cùng liên quan tới công việc của một số lớn các lập trình viên, vì thế cần phải có kế hoạch cụ thể. Hình 2.9 minh họa kế hoạch kiểm thử kết nối giữa kiểm thử và các hoạt động phát triển phần mềm.



h 2.9. K thử hoạt ển phần mềm

Kiểm thử chấp nhận đôi khi còn gọi là kiểm thử alpha. Các hệ thống được phát triển cho một khách hàng riêng lẻ. Tiến trình kiểm thử alpha tiếp tục cho đến khi hệ thống phát triển hoàn thiện và khách hàng chấp nhận sử dụng hệ thống.

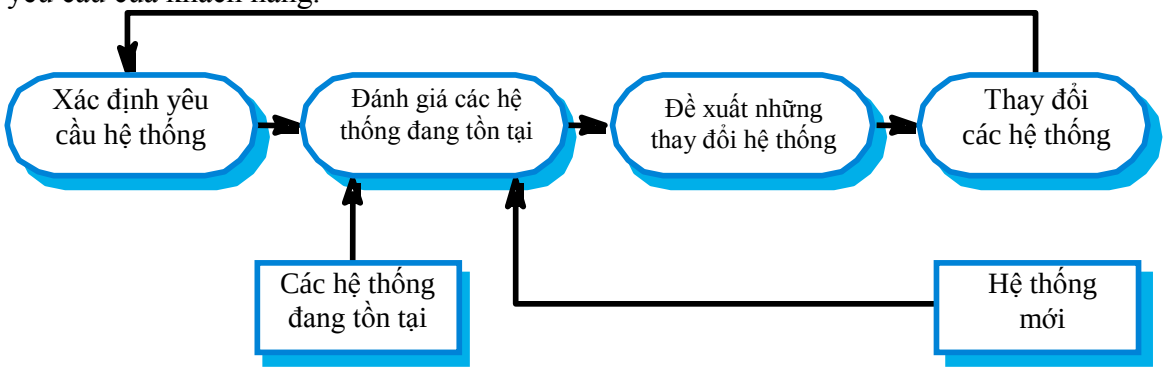
Khi đưa một hệ thống ra thị trường thành một sản phẩm phần mềm, tiến trình kiểm thử được gọi là kiểm thử beta. Kiểm thử beta liên quan tới việc triển khai hệ thống tới một số khách hàng tiềm năng và họ đồng ý sử dụng hệ thống. Những vấn đề còn tồn tại sẽ được báo cáo lại với nhóm phát triển hệ thống. Sau những phản hồi này, hệ thống có thể thay đổi và hoàn thiện lại trong các phiên bản tiếp theo để đưa ra thị trường.

2.3.4. Cải tiến phần mềm

Tính mềm dẻo, linh hoạt của phần mềm là một trong những nguyên nhân chính dẫn đến việc ngày càng nhiều phần mềm có thể hợp nhất với nhau thành những hệ thống phức tạp. Khi quyết định mua phần cứng mới, rất tốn kém để thay đổi thiết kế phần cứng. Tuy nhiên, đối với phần mềm, ta có thể thay đổi tại bất kỳ thời điểm nào, ngay cả khi phần mềm đã hoàn thành. Những thay đổi này đôi khi cũng rất tốn kém, nhưng dù sao nó cũng rẻ hơn là việc thay đổi phần cứng tương đương.

Từ trước, người ta luôn tách việc phát triển và bảo trì phần mềm thành 2 giai đoạn khác nhau. Người ta cho rằng, việc phát triển phần mềm là một hoạt động sáng tạo, từ các khái niệm ban đầu, người phát triển có thể xây dựng thành các hệ thống thực thi được. Và như thế, người ta nghĩ rằng việc bảo trì phần mềm là một việc nhàm chán và ít thú vị. Mặc dù chi phí cho bảo trì phần mềm lớn hơn gấp vài lần chi phí phát triển phần mềm, việc bảo trì vẫn được coi là một công việc ít thách thức hơn so với việc làm mới.

Việc phân biệt này ngày càng trở nên không hợp lý. Người ta cho rằng việc cải tiến phần mềm ngày nay cũng được coi là một giai đoạn trong tiến trình phát triển phần mềm. Hình 2.10 đã chỉ ra việc thay đổi được tiến hành một cách liên tục trong vòng đời phần mềm cho phù hợp với yêu cầu của khách hàng.



Hình 2.10. Tiến trình cải tiến sản phẩm phần mềm

2.4. RUP – TIẾN TRÌNH SẢN XUẤT PHẦN MỀM CỦA RATIONAL

RUP là một ví dụ về một mô hình tiến trình hiện đại được đưa ra từ việc kết hợp UML và tiến trình phát triển phần mềm hợp nhất của Rumbough. Tác giả đưa ra tiến trình này như việc minh họa cho một mô hình tiến trình lặp. Nó là sự kết hợp các thành phần của các tiến trình chung, hỗ trợ việc lặp đi lặp lại và minh họa một thực tế tốt trong đặc tả và thiết kế. Một tiến trình RUP bao gồm các phối cảnh sau:

- Một phối cảnh động chỉ ra các giai đoạn của mô hình theo thời gian.
- Một phối cảnh tĩnh chỉ ra quá trình hoạt động.
- Một phối cảnh chỉ ra những gợi ý thực tế tốt được sử dụng trong suốt tiến trình.

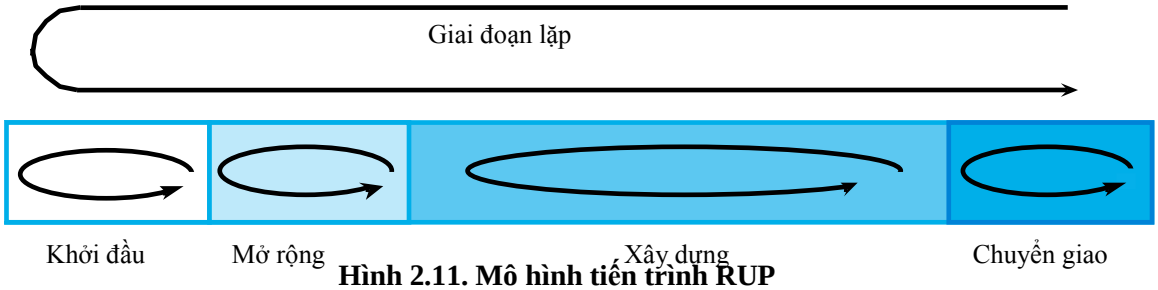
RUP là một mô hình được phân chia thành 4 giai đoạn trong tiến trình phần mềm, tuy nhiên, không giống như mô hình thác nước, các giai đoạn trong RUP gần với mục tiêu nghiệp vụ hơn là những vấn đề liên quan tới kỹ thuật. Hình 2.11 chỉ ra các giai đoạn trong RUP.

Giai đoạn khởi đầu: mục tiêu của giai đoạn khởi đầu là sinh ra một tình huống nghiệp vụ. Ta cần xác định tất cả các thực thể bên ngoài (con người và các hệ thống) sẽ ảnh hưởng tới hệ thống của bạn và định ra các mối tương tác này. Sau đó sử dụng những thông tin này để đánh giá những đóng góp của hệ thống cho mục tiêu nghiệp vụ. Nếu hiệu quả của nó không lớn, ta có thể kết thúc dự án sau bước này.

Giai đoạn mở rộng: mục tiêu của giai đoạn này là tăng cường những hiểu biết của lĩnh vực ứng dụng. Đưa ra một framework kiến trúc cho hệ thống, phát triển kế hoạch dự án và xác định những rủi ro quan trọng của dự án. Khi hoàn thành giai đoạn này, nên có một mô hình yêu cầu cho hệ thống (UML sử dụng các mô hình Use-case), một bản mô tả kiến trúc và một bản kế hoạch cho phần mềm.

Giai đoạn xây dựng: giai đoạn này chủ yếu liên quan tới việc thiết kế hệ thống, lập trình và kiểm thử. Một phần của hệ thống được phát triển song song và được tích hợp trong giai đoạn này. Khi kết thúc giai đoạn này, nên có một hệ thống phần mềm thực thi được và các tài liệu kèm theo để có thể sẵn sàng chuyển tới người sử dụng.

Giai đoạn chuyển giao: giai đoạn cuối cùng của RUP liên quan tới việc chuyển giao sản phẩm từ nhóm phát triển sang nhóm người sử dụng và phần mềm được thực thi trong môi trường thực. Giai đoạn này thường được bỏ qua trong hầu hết các mô hình tiến trình phần mềm khác, nhưng trong thực tế, đây lại là một giai đoạn tốn kém và đôi khi nó xuất hiện một số vấn đề. Khi hoàn thành giai đoạn này, cần xây dựng một tài liệu hệ thống phần mềm làm việc đúng trong môi trường chức năng.



Hình 2.11. Mô hình tiến trình RUP

Việc lặp lại trong RUP có thể tiến hành theo hai cách giống như trong hình 2.11. Mỗi giai đoạn có thể được diễn ra một cách lặp đi lặp lại với kết quả phát triển tăng dần. Hơn nữa, các giai đoạn có thể được diễn ra một cách tăng dần, từ giai đoạn chuyển giao đến giai đoạn khởi đầu giống như trong hình.

Viễn cảnh thực tế của RUP mô tả những kỹ thuật phần mềm tốt và được đề cử để sử dụng trong phát triển hệ thống. Sáu bước thực hành cơ bản:

- ***Phát triển phần mềm lặp đi lặp lại:*** kế hoạch thực thi các increment dựa trên sự lựa chọn của khách hàng và phát triển, chuyển giao những hệ thống có độ ưu tiên cao nhất trong giai đoạn đầu của tiến trình phát triển.

- ***Quản lý yêu cầu:*** tài liệu hóa những yêu cầu người dùng một cách rõ ràng và theo dõi những thay đổi của các yêu cầu này. Phân tích ảnh hưởng của những thay đổi trước khi chấp nhận chúng.

- ***Sử dụng kiến trúc hướng thành phần:*** cấu trúc kiến trúc hệ thống vào các thành phần như đã thảo luận ở phần đầu của chương này.

- ***Mô hình hóa phần mềm:*** sử dụng các lược đồ UML để mô tả các cấu trúc động và tĩnh của hệ thống.

- ***Xác minh chất lượng phần mềm:*** đảm bảo rằng phải đáp ứng được các chuẩn chất lượng của tổ chức.

- ***Kiểm soát sự thay đổi của phần mềm:*** quản lý những thay đổi của phần mềm sử dụng một hệ thống quản lý thay đổi và các công cụ, thủ tục quản lý cấu hình.

RUP không phải là một tiến trình phù hợp với tất cả các kiểu phát triển, nhưng nó đưa ra một tiến trình chung cho các thế hệ phần mềm mới. Sáng kiến quan trọng nhất là sự phân chia thành các giai đoạn và các workflow. Điều quan trọng là nhận ra rằng việc triển khai hệ thống là một phần trong tiến trình phát triển phần mềm, các giai đoạn là động và có những mục tiêu cụ thể. Luồng công việc (workflow) là tĩnh và là các hoạt động mang tính kỹ thuật mà không được kết hợp vào một giai đoạn nào đó sẽ được sử dụng trong suốt quá trình để đáp ứng mục tiêu của từng giai đoạn.

2.5. KỸ NGHỆ PHẦN MỀM CÓ MÁY TÍNH TRỢ GIÚP (CASE)

CASE là tên của những phần mềm được sử dụng để hỗ trợ các hoạt động trong tiến trình sản xuất phần mềm, chẳng hạn như kỹ nghệ yêu cầu, thiết kế, phát triển chương trình và kiểm thử. Các công cụ CASE bao gồm: bộ soạn thảo thiết kế, từ điển dữ liệu, trình biên dịch, gỡ lỗi, các công cụ xây dựng hệ thống…

Kỹ thuật CASE giúp tự động hóa một số hoạt động trong tiến trình sản xuất phần mềm. Ví dụ về các hoạt động có thể tự động hóa bằng việc sử dụng CASE:

- Xây dựng các mô hình đồ họa hệ thống như là một phần của đặc tả yêu cầu hoặc thiết kế phần mềm.

- Hiểu một bản thiết kế sử dụng từ điển dữ liệu lưu trữ những thông tin về các thực thể và những mối quan hệ trong thiết kế.

- Bộ sinh giao diện người dùng từ việc mô tả giao diện đồ họa được tạo ra một cách tương tác bởi người sử dụng.

- Gỡ lỗi chương trình qua việc cung cấp thông tin về một chương trình đang được thực thi.

- Trình biên dịch tự động sinh ra các chương trình từ những phiên bản cũ của một ngôn ngữ lập trình, chẳng hạn như COBOL sang một phiên bản mới hơn.

Kỹ thuật CASE ngày nay hỗ trợ cho hầu hết các hoạt động trong tiến trình phần mềm. Điều này về cơ bản cũng có thể kiểm soát được chất lượng và năng suất lao động, mặc dù nó không được hoàn hảo như lời tiên đoán của những người tạo ra CASE. Những cải tiến từ việc sử dụng CASE bị hạn chế bởi những yếu tố sau:

- Công nghệ phần mềm là một hoạt động sáng tạo – điều này không dễ dàng để có thể tự động hóa.

- Công nghệ phần mềm là một hoạt động làm việc theo nhóm, phần lớn thời gian phải làm việc chung với các cá nhân khác – CASE không thực sự hỗ trợ yếu tố này.

- Các kỹ thuật CASE ngày nay có những tiến bộ rất đáng khích lệ và các công cụ CASE cũng như các mô trường hỗ trợ được cung cấp bởi nhiều nhà cung cấp khác nhau. Tuy nhiên, hầu hết chúng đều tập trung vào các công cụ hỗ trợ trong quá trình làm đặc tả.

CÂU HỎI ÔN TẬP

1. Tiến trình phần mềm là gì? Có những kiểu mô hình tiến trình chung nào?
2. Với những kiểu dựa án nào ta nên sử dụng mô hình thác nước. Hãy giải thích tại sao?
3. Mô hình phát triển xoắn ốc phù hợp với kiểu dự án nào? Hãy giải thích?
4. Hãy nêu những ưu điểm và nhược điểm của mô hình phát triển tiến hóa.
5. Hãy nêu các hoạt động chung trong tiến trình phần mềm.
6. Hãy so sánh mô hình tiến trình phát triển phần mềm RUP với mô hình phát triển tiến hóa.
7. CASE là gì? Nêu tầm quan trọng của CASE trong phát triển phần mềm.

Chương 3: QUẢN LÝ DỰ ÁN PHẦN MỀM

Cuối những năm 1960 và đầu những năm 1970, ngành sản xuất phần mềm rơi vào thời kỳ khủng hoảng, rất nhiều dự án phần mềm lớn thất bại do những bất cập trong quản lý dự án: phần mềm chuyển giao không đúng thời hạn, không đáp ứng đủ độ tin cậy, chi phí vượt nhiều lần so với dự đoán, sản phẩm không đáp ứng được yêu cầu của người sử dụng. Theo thống kê của Standish Group (2006):

- Có tới 50% trong số các dự án phần mềm thất bại.

- Chỉ có 16,2% dự án là hoàn thành đúng hạn và nằm trong giới hạn ngân sách, đáp ứng tất cả tính năng và đặc tính như cam kết ban đầu.

- Có 52,7% dự án được hoàn thành và đi vào hoạt động nhưng không hoàn thành đúng hạn và bội chi, thêm nữa không đáp ứng đầy đủ tính năng và đặc tính như thiết kế ban đầu.

- Có 31,1% dự án thất bại trước khi được hoàn thành.

- Hơn 83,8% dự án thất bại hoặc không đáp ứng những yêu cầu ban đầu.

Năm 1995, các công ty ở Mỹ đã phải chi 81 tỷ USD cho những dự án bị hủy bỏ, 59 tỷ USD đầu tư thêm cho các dự án không đúng kế hoạch.

Ở Việt Nam, dự án "Hệ thống điện tử xử lý thông tin tại SeaGame 22 của VN" dự kiến kinh phí là 15 tỷ VND, nhưng mới đến tháng 6/2003 đã chi tới 90 tỷ VND. Những dữ liệu trên cho thấy: các dự án phần mềm cần được quản lý để tránh những thất bại, mặc dù quản lý tốt chưa đủ yếu tố để đảm bảo thành công nhưng quản lý kém chắc chắn sẽ dẫn đến sự thất bại của dự án [3].

Chương này giới thiệu một số khái niệm cơ bản trong quản lý dự án nói chung và các dự án CNTT nói riêng. Nội dung chính của chương đề cập đến các hoạt động trong quản lý dự án theo phương pháp truyền thống và một hoạt động không thể thiếu trong quản lý dự án là phân tích và quản lý rủi ro. Đây là một trong những yếu tố quyết định tới sự thành công của dự án.

3.1. CÁC KHÁI NIỆM CƠ BẢN

3.1.1. Khái niệm dự án

Theo quan điểm chung, dự án là một lĩnh vực hoạt động đặc thù, một nhiệm vụ cần được thực hiện theo một phương pháp riêng, trong khuôn khổ nguồn lực riêng, kế hoạch tiến độ cụ thể nhằm tạo ra một sản phẩm mới. Do đó, dự án có tính cụ thể, mục tiêu rõ ràng, xác định để tạo ra một sản phẩm mới.

Theo PMBOOK® Guide 2000, p.4, dự án là "một nỗ lực tạm thời được cam kết để tạo ra một sản phẩm hoặc dịch vụ duy nhất". Theo định nghĩa này, hoạt động của dự án tập trung vào 2 khía cạnh:

- Nỗ lực tạm thời: Mọi dự án đều có điểm bắt đầu và kết thúc cụ thể. Dự án chỉ kết thúc khi đã đạt được mục tiêu dự án hoặc dự án thất bại.

- Sản phẩm và dịch vụ là duy nhất: Điều này thể hiện sự khác biệt so với các sản phẩm và dịch vụ đã có hoặc kết quả của dự án khác.

Tóm lại, dự án là một chuỗi các công việc (nhiệm vụ, hoạt động), được một số người thực hiện nhằm đạt được mục tiêu đề ra trong điều kiện ràng buộc về phạm vi, thời gian và ngân sách.

3.1.2. Các đặc trưng của dự án

- Sản phẩm đơn chiếc, duy nhất;

- Chỉ làm một lần;

- Người tham gia thường có chuyên môn khác nhau, được tập hợp lại;

- Có lịch trình chặt chẽ, chi phí xác định;

- Kết quả không chắc chắn: sản phẩm đáp ứng yêu cầu và thỏa mãn ràng buộc nghĩa là dự án thành công, ngược lại là dự án thất bại.

Các đặc trưng này đặt ra nhiều vấn đề cần nghiên cứu và giải quyết trong hoạt động triển khai dự án. Mặt khác, một dự án thất bại thường do các nguyên nhân sau:

- Quản lý dự án kém: người quản lý thiếu kiến thức và kinh nghiệm quản lý dự án.

- Không lường hết được phạm vi, độ phức tạp của công việc. Do đó, dự kiến nguồn lực (người, kinh phí, thời gian) thực hiện không chính xác.

- Thiếu thông tin khi thực hiện, nên không xử lý kịp thời các vấn đề nảy sinh.

- Các lý do khác, trong đó nhiều yếu tố bất ngờ từ môi trường như công nghệ thay đổi, người cung cấp nguồn lực vi phạm hợp đồng.

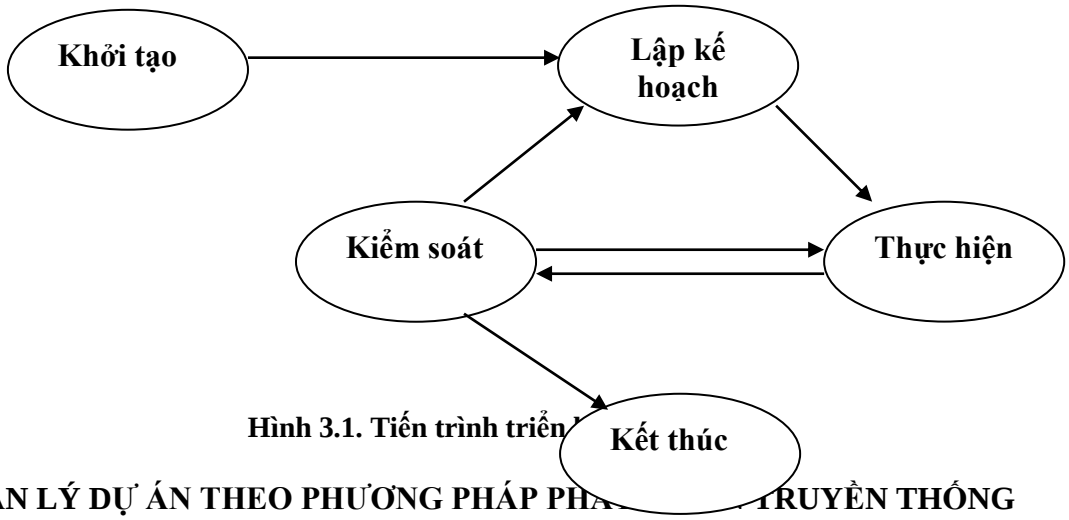
Ba lý do sau cùng đều liên quan tới lý do đầu tiên. Vì thế có thể nói rằng, quản lý dự án kém là nguyên nhân chủ yếu dẫn đến dự án thất bại.

3.1.3. Quản lý dự án

Quản lý dự án là *"ứng dụng kiến thức, kỹ năng, công cụ và kỹ thuật vào các hoạt động dự án để thỏa mãn các yêu cầu của dự án"* (PMBOK ® Guide, 2000, p.6).

Xét theo khía cạnh khác, quản lý dự án là một quá trình lập kế hoạch, điều phối thời gian, nguồn lực và giám sát quá trình phát triển của dự án nhằm đảm bảo cho dự án hoàn thành đúng thời hạn, trong phạm vi ngân sách được duyệt và đạt được các yêu cầu đã định về kỹ thuật, chất lượng của sản phẩm, dịch vụ, bằng các phương pháp và điều kiện tốt nhất cho phép.

Công việc quản lý dự án trải ra trong suốt vòng đời của dự án: từ lúc xác định (hình thành) dự án, lập kế hoạch, thực hiện dự án đến kết thúc dự án. Bức tranh tổng quát về tiến trình triển khai một dự án được minh họa trong hình 3.1.



Hình 3.1. Tiến trình triển k

3.2. QUẢN LÝ DỰ ÁN THEO PHƯƠNG PHÁP PHÁ[illegible] TRUYỀN THỐNG

3.2.1. Các hoạt động quản lý dự án

Rất khó để có thể đưa ra một bảng mô tả công việc chuẩn cho người làm quản lý dự án phần mềm. Công việc này rất khác nhau, phụ thuộc vào tổ chức và phần mềm sẽ được xây dựng. Tuy nhiên, những hoạt động phổ biến của quản lý dự án thường bao gồm:

- Viết kế hoạch đề xuất;
- Ước lượng các chi phí nguồn lực;
- Lựa chọn, đánh giá nguồn lực;
- Lập kế hoạch dự án và lịch làm việc;
- Triển khai và kiểm soát tiến trình dự án;
- Viết báo cáo và trình bày;
- Tổng kết, đánh giá.

Viết kế hoạch đề xuất: Giai đoạn đầu của dự án liên quan đến việc viết kế hoạch đề xuất để có thể giành được hợp đồng thực hiện công việc. Bản đề xuất nêu các mục tiêu của dự án và cách thức thực hiện dự án. Thông thường, bao gồm việc ước lượng chi phí và thời gian thực hiện, đồng thời cũng phải chứng minh tại sao dự án nên được thực hiện bởi một tổ chức hoặc một nhóm xác định. Việc viết kế hoạch đề xuất là nhiệm vụ quan trọng, có khả năng quyết định xem dự án có được phê duyệt hay không hoặc tổ chức có được chấp nhận để thực hiện dự án hay không. Nói chung, không có những hướng dẫn cụ thể cho việc viết kế hoạch đề xuất, mà phụ thuộc nhiều vào kỹ năng của người viết.

Lập kế hoạch dự án và lịch làm việc: Xác định các hoạt động, các mốc quan trọng và thời gian bàn giao sản phẩm của dự án. Lập kế hoạch giúp cho quá trình phát triển phần mềm đáp ứng được mục tiêu của dự án. Ước lượng chi phí ước lượng tài nguyên cần thiết để thực hiện dự án.

Lựa chọn, đánh giá nguồn lực: Người quản lý dự án cần phải lựa chọn nhân sự thực hiện dự án. Một cách lý tưởng là người quản lý lựa chọn được những nhân viên có kinh nghiệm tham gia vào dự án của mình. Nhưng công việc này thường không đạt được như mong muốn, do một số lý do sau:

- Ngân sách của dự án có thể không cho phép sử dụng những nhân viên có mức lương cao, vì thế phải lựa chọn những nhân viên có ít kinh nghiệm với mức lương thấp hơn.

- Những nhân viên có nhiều kinh nghiệm không phải lúc nào cũng sẵn sàng tham gia dự án. Có thể xảy ra tình trạng không thể xây dựng được một đội ngũ nhân viên mới phù hợp. Bên trong tổ chức, những người có kinh nghiệm thích hợp có thể đã tham gia vào các dự án khác.

- Một tổ chức có thể phát triển kỹ năng của nhân viên cho dự án phần mềm. Những nhân viên không có kinh nghiệm có thể được phân công để học và tiếp thu những kinh nghiệm cần thiết.

- Người quản lý phải làm việc trong những ràng buộc này, đặc biệt khi không còn thời gian để đào tạo nhân viên. Tuy nhiên, phải có một số thành viên của dự án có kinh nghiệm về kiểu hệ thống đang được phát triển. Với những kinh nghiệm này, các vấn đề mà dự án gặp phải có thể sẽ được giải quyết dễ dàng hơn.

Triển khai và kiểm soát dự án: Hoạt động giám sát là một hoạt động liên tục, người quản lý theo dõi xem tiến trình dự án có phù hợp với kế hoạch và kinh phí đã xác định trong giai đoạn lập kế hoạch hay không. Trong hầu hết các tổ chức, một cơ chế đặc biệt được dùng để giám sát dự án. Một người quản lý nhiều kinh nghiệm có thể đưa ra các vấn đề để thảo luận với nhân viên của dự án. Các cuộc thảo luận này thường xoay quanh những khó khăn gặp phải trong quá trình thực hiện dự án, từ đó dự đoán những vấn đề tiềm ẩn sẽ phát sinh. Ví dụ: thảo luận hàng ngày với nhân viên dự án để khám phá ra một vấn đề đặc thù trong việc tìm ra vài lỗi phần mềm, còn hơn là ngồi đợi cho đến khi nghe báo cáo tình trạng mục tiêu không hoàn thành. Người quản lý phần mềm có thể tham khảo những chuyên gia giầu kinh nghiệm để quyết định xem vấn đề sẽ được giải quyết như thế nào.

Trong thời gian thực hiện dự án, cũng cần phải thường xuyên xem xét tiến trình và công nghệ được sử dụng để phát triển dự án, kiểm tra xem dự án và mục tiêu của tổ chức vẫn được giữ vững hay không. Việc xem xét này có thể dẫn đến việc hủy bỏ dự án. Ví dụ: đối với một dự án lớn, đòi hỏi thời gian phát triển vài năm, trong thời gian này, các mục tiêu của tổ chức chắc chắn sẽ có sự thay đổi. Những thay đổi này có thể dẫn đến tình trạng phần mềm không còn mang tính cấp bách hoặc những yêu cầu cơ bản không còn thích hợp. Khi đó người quản lý có thể quyết định dừng việc phát triển phần mềm hoặc thay đổi dự án cho phù hợp với những thay đổi của tổ chức.

Viết báo cáo và trình bày: Người quản lý dự án thường xuyên phải viết các báo cáo cho cả khách hàng và tổ chức đấu thầu dự án. Các tài liệu phải chặt chẽ và ngắn gọn để tóm tắt những thông tin quan trọng từ những báo cáo chi tiết về dự án. Họ phải có khả năng giới thiệu những thông tin này trong quá trình xem xét tiến trình dự án.

Kết thúc dự án: Cần tiến hành một số công việc để hoàn tất những gì có liên quan đến dự án và đánh giá rút kinh nghiệm.

3.2.2. Lập kế hoạch dự án

Hiệu quả của việc quản lý một dự án phần mềm phụ thuộc vào việc lập kế hoạch dự án. Người quản lý phải có khả năng dự đoán trước các vấn đề có thể phát sinh và so sánh các giải pháp để giải quyết vấn đề. Bản kế hoạch được đưa ra từ giai đoạn đầu thực hiện dự án nên nó được sử dụng làm công cụ dẫn đường cho dự án. Kế hoạch khởi đầu này nên là kế hoạch tốt nhất,

có thể đưa ra những thông tin có giá trị liên quan đến tiến trình của dự án. Bản kế hoạch khởi đầu cần đạt được các mục tiêu sau:

- Có thể sắp xếp thời gian cho hầu hết các hoạt động quản lý dự án.

- Hoạt động liên tục từ thiết kế ban đầu cho đến khi bàn giao hệ thống. Kế hoạch phải được xem xét lại một cách thường xuyên khi có những thông tin mới xuất hiện.

- Có nhiều kiểu kế hoạch khác nhau có thể được phát triển để từ đó đưa ra kế hoạch cơ bản cho dự án phần mềm liên quan đến thời gian biểu và ngân sách.

Bảng 3.1. Các kiểu kế hoạch cần cho dự án

Kế hoạch	Mô tả
Kế hoạch chất lượng	Mô tả các thủ tục và các chuẩn sẽ được sử dụng trong dự án
Kế hoạch kiểm thử	Mô tả cách tiếp cận, nguồn tài nguyên và lịch trình được sử dụng để kiểm thử dự án
Kế hoạch quản lý cấu hình	Mô tả các thủ tục quản lý cấu hình và cấu trúc được sử dụng
Kế hoạch bảo trì	Dự đoán những yêu cầu bảo trì của hệ thống, chi phí cho bảo trì và yêu cầu cần đạt được
Kế hoạch phát triển nhân lực	Mô tả cách thức để phát triển kỹ năng và kinh nghiệm của những thành viên tham gia dự án

a) Tiến trình lập kế hoạch dự án

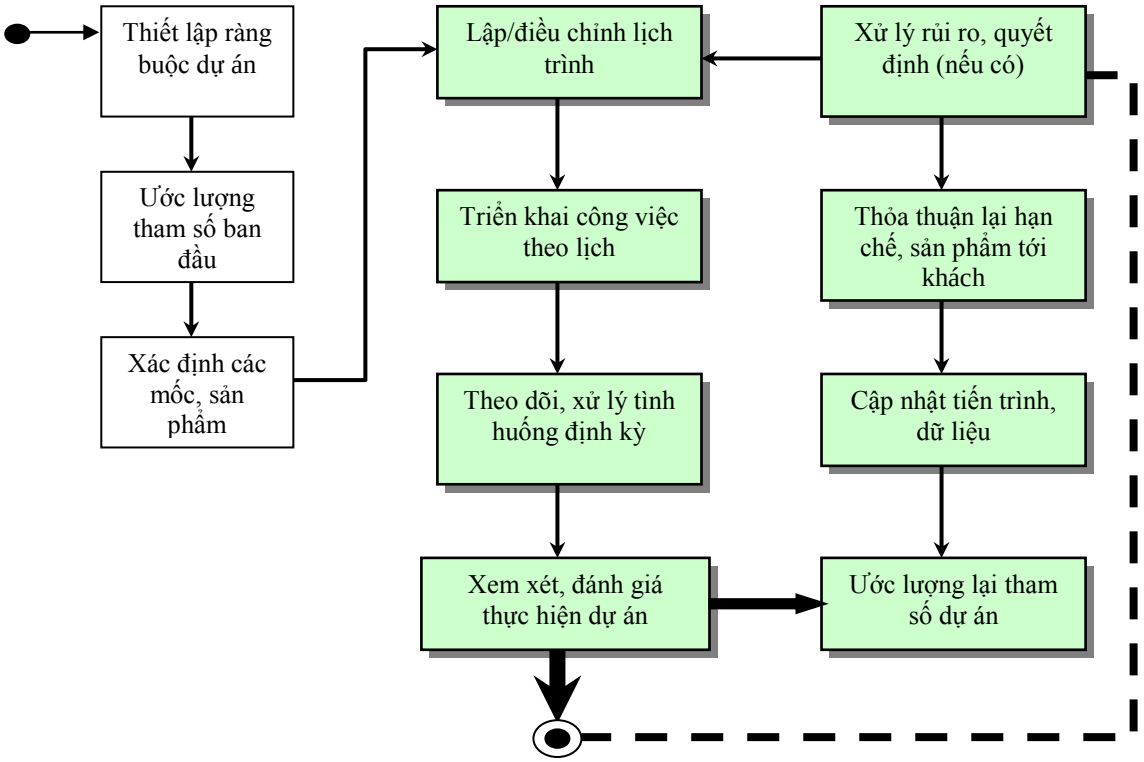
Biểu đồ hình 3.2 mô tả tiến trình lập kế hoạch dự án cho phát triển phần mềm. Hình vẽ này đã chỉ ra rằng việc lập kế hoạch là một tiến trình liên tục, bắt đầu từ khi triển khai dự án cho đến khi dự án kết thúc. Đây là một thông tin có giá trị trong suốt thời gian thực hiện dự án, vì thế kế hoạch dự án cũng cần phải được xem xét lại một cách thường xuyên. Mục tiêu của tổ chức là vấn đề quan trọng của dự án, những mục tiêu này thường xuyên thay đổi, nên kế hoạch dự án có thay đổi là cần thiết.

Trong thời gian đầu của tiến trình lập kế hoạch, ta cần xem xét đến các ràng buộc có ảnh hưởng đến dự án như thời hạn giao nộp sản phẩm, những nhân viên có thể tham gia vào dự án và nguồn ngân sách có thể giành cho dự án... Từ đó ước lượng các tham số của dự án, chẳng hạn như: cấu trúc, kích thước, sự phân tán của các chức năng. Tiếp theo người quản lý dự án cần phải xác định được các mốc quan trọng và thời gian giao nộp sản phẩm. Tiến trình sau đó rơi vào một vòng lặp. Người quản lý phải đưa ra một thời gian biểu ước lượng cho dự án và xác định các hoạt động trong thời gian biểu đã đưa ra. Sau đó thỉnh thoảng (khoảng 2-3 tuần) nên xem xét lại tiến trình và ghi lại những điểm không phù hợp với thời gian biểu đã được lập. Vì những ước tính ban đầu chỉ là tương đối nên kế hoạch này thường phải điều chỉnh.

Khi có nhiều thông tin có ích hơn, cần xem xét lại những tham số ban đầu về dự án và thời gian biểu của dự án. Nếu dự án bị chậm tiến độ, người quản lý có thể thỏa thuận lại với khách hàng về những ràng buộc và thời gian bàn giao sản phẩm. Nếu sự thỏa thuận không thành công và thời gian biểu không đáp ứng được, các chuyên gia kỹ thuật có thể giúp nhà quản lý khắc phục khó khăn. Mục tiêu của việc xem xét lại kế hoạch là nhằm giúp tìm ra những giải pháp tối ưu để khắc phục những vấn đề nảy sinh.

Người quản lý dự án cũng cần phải chuẩn bị tâm lý là không bao giờ mọi thứ đều diễn ra như ý muốn và luôn luôn có những vấn đề phát sinh trong quá trình thực hiện dự án. Công việc

và thời gian biểu ban đầu nên bi quan hơn là lạc quan. Khi xây dựng kế hoạch, nên xem xét đủ các yếu tố ngẫu nhiên để tránh việc phải đàm phán lại mỗi khi một vòng lặp được hoàn thành.



Hình 3.2. Tiến trình lập kế hoạch và thực hiện dự án

b) Cấu trúc bản kế hoạch dự án

Kế hoạch dự án thiết lập ra những tài nguyên có thể sử dụng cho dự án, phân chia các công việc và lịch trình thực hiện công việc. Kế hoạch dự án có thể là một tài liệu duy nhất, bao gồm nhiều loại khác nhau như đã giới thiệu ở trên. Trong trường hợp khác, kế hoạch dự án chỉ liên quan đến tiến trình phát triển phần mềm. Sự thay đổi chi tiết trong kế hoạch dự án phụ thuộc vào kiểu dự án và tổ chức thực hiện dự án. Tuy nhiên, hầu hết các bản kế hoạch dự án đều bao gồm các thành phần sau:

Giới thiệu: Mô tả tóm tắt những mục tiêu của dự án và đưa ra những ràng buộc (thời gian, ngân sách...) có ảnh hưởng tới việc quản lý dự án.

Tổ chức dự án: giới thiệu tóm tắt về tổ chức phát triển dự án, những người tham gia và vai trò của họ trong dự án.

Phân tích rủi ro: Nêu lên những rủi ro có thể xảy ra và khả năng xảy ra rủi ro này là cao hay thấp, đồng thời đưa ra những chiến lược để giảm được tối đa những rủi ro.

Các nguồn tài nguyên phần cứng và phần mềm cần cho dự án: phần này liệt kê những tài nguyên phần cứng và phần mềm cần để phát triển dự án. Nếu phải mua phần cứng, thì phải ước lượng giá thành của sản phẩm và thời gian có được sản phẩm.

Phân chia công việc: Chia dự án thành các hoạt động và xác định những mốc thời gian quan trọng, thời điểm hoàn thành các công việc được giao.

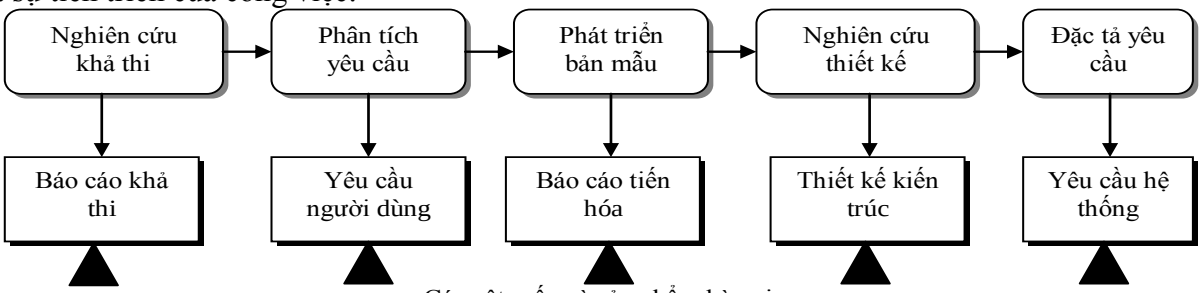
Thời gian biểu dự án: Chỉ ra sự phụ thuộc giữa các hoạt động của dự án, ước lượng thời gian cần để đạt được các mốc quan trọng, cũng như nhân sự để tham gia vào công việc.

Cơ chế kiểm tra giám sát và báo cáo: Chỉ ra việc quản lý các báo cáo, khi nào cần đưa ra báo cáo và cơ chế nào được sử dụng để giám sát các báo cáo đó.

Kế hoạch dự án cần được định kỳ lặp lại trong quá trình thực hiện dự án. Một vài thành phần như lịch trình dự án sẽ thay đổi thường xuyên, những phần khác tương đối ổn định. Tài liệu phải được tổ chức cho phép dễ dàng sửa đổi và thay thế những phần còn được sử dụng tiếp.

c) Các mốc quan trọng và các sản phẩm bàn giao

Vì phần mềm là một sản phẩm vô hình nên thông tin về sản phẩm là những báo cáo và các tài liệu mô tả giai đoạn phần mềm đang được phát triển. Thiếu những thông tin này, người quản lý không nắm được tiến trình của dự án để có những điều chỉnh phù hợp. Các hoạt động trong một dự án cần phải được tổ chức theo những quy trình rõ ràng, cụ thể cho việc quản lý để đánh giá sự tiến triển của công việc.



Các cột mốc và sản phẩm bàn giao

Khi lập kê hoạch cho một dự án, ta cần đưa ra những môc thời gian quan trọng, là thời điểm kết thúc một hoạt động trong tiến trình công việc. Với mỗi mốc thời gian quan trọng, cần đưa ra một đầu ra cụ thể, chẳng hạn như là một tài liệu, tài liệu này không cần quá chi tiết mà đơn giản là tóm tắt được công việc đã hoàn thành tại mốc này.

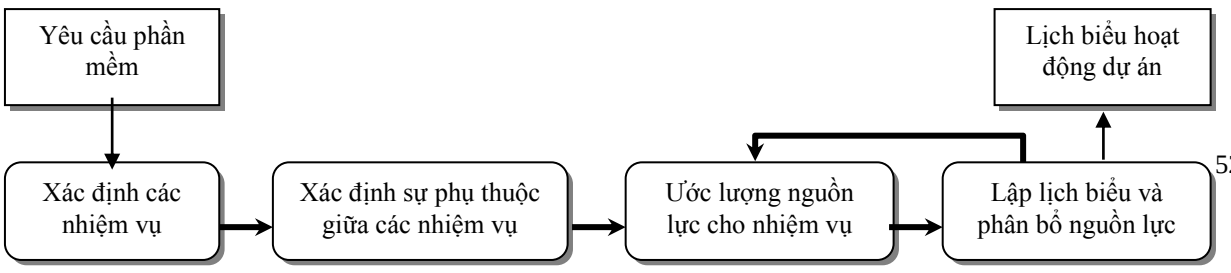
Thời hạn bàn giao sản phẩm là việc đưa kết quả dự án tới khách hàng. Nó thường được bàn giao tại những giai đoạn chính chẳng hạn như đặc tả hoặc thiết kế. Thời hạn bàn giao là một mốc quan trọng, nhưng mốc quan trọng lại không phải là thời hạn bàn giao. Những mốc quan trọng có thể là những kết quả bên trong dự án, được người quản lý sử dụng để kiểm tra tiến độ dự án nhưng không giao nộp cho khách hàng. Để thiết lập các mốc, tiến trình phầm mềm cần được phân ra thành những hoạt động cơ bản với các đầu ra cụ thể đi kèm như hình 3.3.

3.2.3. Lập lịch dự án

a) Tiến trình lập lịch

Nhìn chung, việc đánh giá thời gian biểu của một dự án là một công việc rất khó khăn, trừ những dự án tương tự với các dự án đã thực hiện trước đó, việc xác định thời gian biểu của dự án thường không chắc chắn và phụ thuộc vào rất nhiều yếu tố khác nhau, ví dụ như phương pháp thiết kế, mô hình phát triển hoặc ngôn ngữ thực hiện.

Tiến trình lập lịch trình dự án được thể hiện trong hình vẽ 3.4, nó liên quan đến việc chia một dự án thành các hoạt động riêng rẽ và ước lượng thời gian để hoàn thành các hoạt động này. Tuy nhiên, có một số hoạt động được tiến hành song song, vì thế cần phải tính toán cả phương án lựa chọn nhân sự cho từng hoạt động sao cho dự án có hiệu quả nhất, tránh tình trạng dự án bị chậm tiến độ do một công việc quan trọng chưa hoàn thành.



Hình 3.4. Tiến trình lập lịch cho hoạt động dự án

Sau khi đã xác định được công việc, cần ước lượng thời gian thực hiện mỗi công việc. Nguyên tắc là ước lượng thời gian tiến hành cho từng công việc nhỏ, từ đó ước lượng cho những công việc được gộp lại. Việc ước lượng thời gian có thể theo kinh nghiệm. Trong phương pháp PERT (*Program Evaluation and Review Technicques*), ước lượng thời gian công việc dựa trên ba ước lượng:

- Ước lượng khả dĩ (ML – *Most likely*) là thời gian cần thiết để hoàn thành công việc trong điều kiện "bình thường".

- Ước lượng lạc quan (Mo – *Most Optimistic*) là thời gian cần thiết để hoàn thành công việc trong điều kiện "lý tưởng".

- Ước lượng bi quan (MP – *Most Pessimistic*) là thời gian cần thiết để hoàn thành công việc trong điều kiện "tồi nhất".

Ước lượng cuối cùng thời gian thực hiện công việc tính theo: $\mathbf{t}_{cv} = (MO + 4ML + MP)/6$

Các hoạt động của dự án thường diễn ra ít nhất một tuần, vì vậy, việc phân chia công việc hiệu quả nhất là mỗi công việc nên thực hiện trong vòng từ tám đến mười tuần, nếu quá thời gian này, nên chia thành những công việc nhỏ hơn.

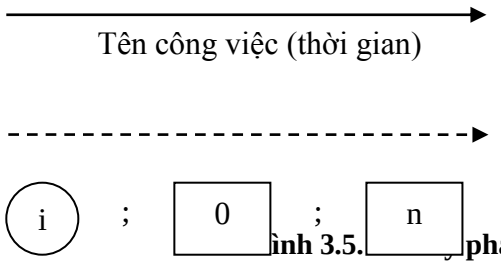
Người quản lý cũng phải đánh giá các tài nguyên cần thiết để hoàn thành mỗi công việc. Tài nguyên có thể là con người, dung lượng đĩa cần thiết trên máy chủ, chi phí cần thiết cho những phần cứng đặc biệt, chi phí đi lại cần thiết cho các nhân viên dự án… Việc dự toán tài nguyên cho các công việc được tiến hành dựa trên kinh nghiệm hay số liệu thống kê của các dự án đã thực hiện.

Một quy tắc tốt là, khi đánh giá, nếu không có gì sai thì tăng số liệu ước lượng lên để bao gồm cả những vấn đề đã thấy trước. Yếu tố bất ngờ bao gồm những vấn đề không thấy trước cũng có thể bổ sung thêm vào sự đánh giá. Yếu tố bất ngờ này phụ thuộc vào kiểu dự án, các tham số tiến trình (thời hạn cuối, các chuẩn…). Ta nên thêm 30% vào đánh giá ban đầu cho những vấn đề đã biết trước và thêm 20% nữa để bao gồm những vấn đề chưa nghĩ tới.

Lịch của dự án thường được đưa ra dưới dạng một tập các đồ thị biểu diễn phân chia công việc, sự phụ thuộc giữa chúng và vị trí của mỗi nhân viên trong việc thực hiện công việc được chỉ ra. Các công cụ phần mềm thường được sử dụng để trợ giúp công việc quản lý như phần mềm Microsoft Project 2000. Đây là công cụ trợ giúp việc sinh tự động các biểu đồ, các tham số, lập báo cáo.

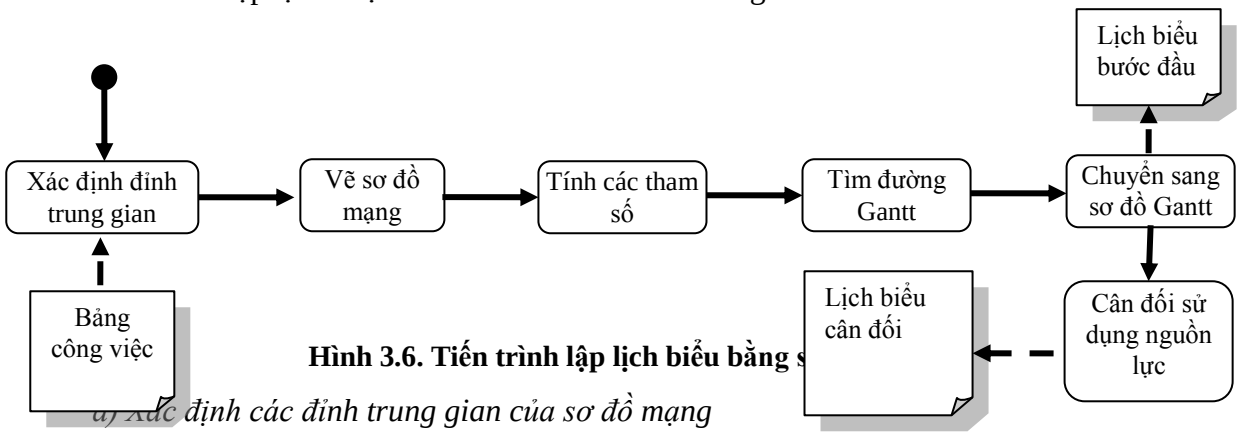
3.2.4. Phương pháp và công cụ lập lịch

Phương pháp đường găng (CMP – Critical Path Method) thường được sử dụng trong hoạt động lập lịch. Phương pháp này sử dụng một sơ đồ mạng các công việc, nó là một đồ thị có hướng, mỗi cung biểu diễn một công việc, mỗi đỉnh biểu diễn một sự kiện là điểm bắt đầu hay kết thúc của một số các công việc. Có một đỉnh bắt đầu cho dự án là đỉnh mà tại đó các công việc đi ra và một đỉnh kết thúc, các công việc cuối cùng của dự án đi vào. Trên mỗi cung có ghi tên công việc và thời gian thực hiện. Hình 3.5 là các kí hiệu biểu diễn các thành phần của sơ đồ mạng.



ình 3.5. pháp

Tiến trình lập lịch được mô tả như sơ đồ mô tả trong hình 3.6.



Hình 3.6. Tiến trình lập lịch biểu bằng s

a) Xác định các đỉnh trung gian của sơ đồ mạng

Xuất phát từ bảng liệt kê công việc của dự án (bảng 3.2), ta tiến hành đánh dấu các công việc ở cột thứ ba (cột "đi sau công việc") như sau:

Bước 1: Duyệt lần lượt từ trên xuống, đánh dấu khoanh tròn cho các chữ (công việc) là duy nhất (cặp 2/cặp 3) trên dòng chưa được đánh dấu và chưa bị xóa. Các công việc được khoanh ở một dòng sẽ xác định một đỉnh trung gian ngay sau chúng trong sơ đồ mạng.

Bước 2: Xóa tên tất cả các công việc đã được khoanh mà có mặt trong các dòng khác chứa hơn 2/3 … công việc và quay về bước 1.

Bước 3: Nếu các dòng chứa 1 công việc đã được khoanh hay đã bị xóa hết, thì xét đến các dòng chứa 2/(3) công việc chưa được khoanh và chưa bị xóa. Lặp lại bước 1 và 2 cho đến khi không còn công việc nào trong cột còn chưa khoanh hay chưa bị xóa.

Áp dụng thuật toán trên cho bảng công việc, ví dụ ta có kết quả như bảng sau:

Bảng 3-2. Bảng liệt kê các công việc của dự án và đánh dấu

Công việc	Thời gian t_{cv}	Đi sau công việc	Công việc	Thời gian t_{cv}	Đi sau công việc
a	1	_	k	2	(g), i
b	5	_	m	3	(i)

c	6	_	l	3	i
d	4	_	n	2	k
e	4	a	o	1	l, n
f	3	b	p	2	~~g~~, i,
g	3	c	q	3	~~g~~, i, l, n
h	4	d	r	2	o, p
I	2	e, f	s	1	r, q

b) Vẽ sơ đồ mạng

1. Vẽ đỉnh **0.**

2. Vẽ các công việc đi ra từ đỉnh này (dựa vào bảng mô tả công việc) xem các công việc nào là những công việc đi sau các công việc kết thúc ở đỉnh đang xét. Trong trường hợp đỉnh 0 đó là các công việc không đi sau công việc nào (như hình a, b, c, d).

3. Sau mỗi công việc vừa được vẽ ta sẽ thêm vào một đỉnh trung gian nếu nó đã được khoanh riêng rẽ trong bảng công việc. Nếu nó được khoanh cùng với các công việc khác mà chúng cũng đã được vẽ trong mạng thì chụm chúng lại và thêm vào đó một đỉnh. Ngược lại thì giữ nguyên hiện trạng.

4. Xét tiếp một đỉnh vừa được thêm vào và quay lại bước 2.

5. Khi vẽ hết các công việc của bảng, tất cả các công việc chưa có đỉnh ngay sau nó thì chụm chúng lại đỉnh kết thúc (**n**) được thêm vào cuối cùng.

6. Xét các công việc bị xóa trong cột 3 của bảng công việc:

- Nếu ở một dòng có các công việc đều bị xóa thì thêm một đỉnh giả và các công việc giả đi từ đỉnh là kết thúc của công việc bị xóa đến đỉnh này.

- Nếu ở một dòng vừa có công việc bị xóa, vừa có công việc được khoanh thì thêm một công việc giả từ đỉnh là kết thúc của các công việc được xóa (kể cả đỉnh giả) đến đỉnh là kết thúc của các công việc được khoanh.

7. Đánh số các đỉnh theo thứ tự tăng dần và đảm bảo đỉnh ở đầu công việc luôn nhỏ hơn đỉnh cuối công việc.

Kết quả vẽ mạng công việc trong bảng 3.2 được thể hiện trong hình 3.7a.

c) Tính các tham số thời gian của mạng

Trên mạng, với mỗi công việc, ta ghi thêm vào thời gian thực hiện chúng (bên cạnh). Tiếp đó tiến hành tính các tham số như sau:

- *Thời gian sớm nhất của một đỉnh:* $\mathbf{t_s(j)}$

Quá trình tính xuôi lần lượt từ đỉnh nhỏ đến đỉnh lớn, bắt đầu tử đỉnh 0:

- $\mathbf{t_s(0)} = 0$

- đỉnh **j** (j = 1, 2… n):

$$t_s(j) = \textbf{Max}\ \{t_s(\text{đỉnh đầu cv}) + t_{cv}\}$$

(mọi cv đi vào đỉnh j)

- *Thời gian muộn nhất của một đỉnh:* $t_m(i)$

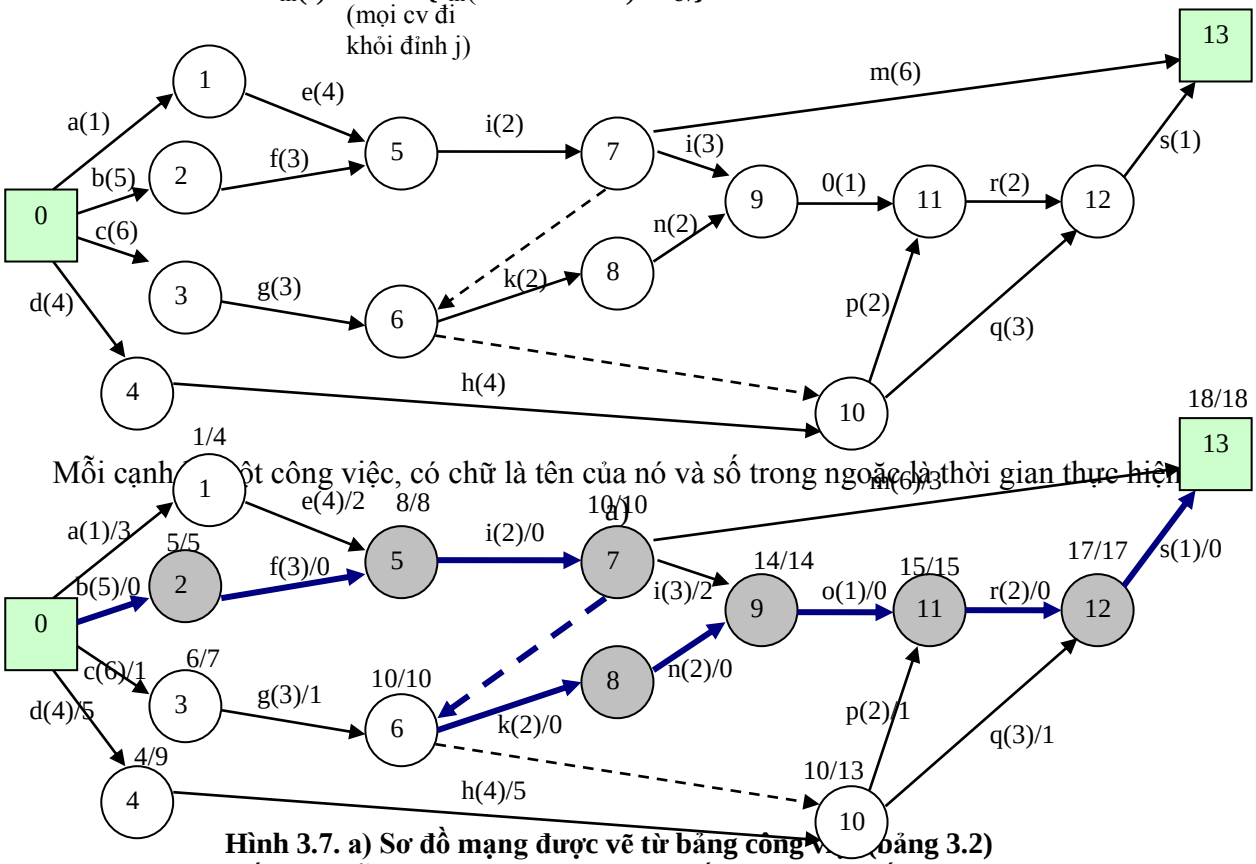
Quá trình tính ngược lần lượt từ đỉnh lớn đến đỉnh nhỏ, bắt đầu từ đỉnh **n:**

- $t_m(n) = t_s(n)$

- đỉnh **i** (i = n-1, …, 0):

$$t_m(i) = \textbf{Min}\ \{t_m(\text{đỉnh đầu cv}) - t_{cv}\}$$

(mọi cv đi khỏi đỉnh j)



Mỗi cạnh là một công việc, có chữ là tên của nó và số trong ngoặc là thời gian thực hiện

Hình 3.7. a) Sơ đồ mạng được vẽ từ bảng công việc (bảng 3.2)

2 số trên mỗi đỉnh là thời gian sớm nhất và muộn nhất của đỉnh

2 số trên mỗi cạnh là thời gian thực hiện và dự trữ của công việc

b)

$$t_{df}(cv) = t_m(\text{đỉnh cuối cv}) - t_s(\text{đỉnh đầu cv}) - t_{cv}$$

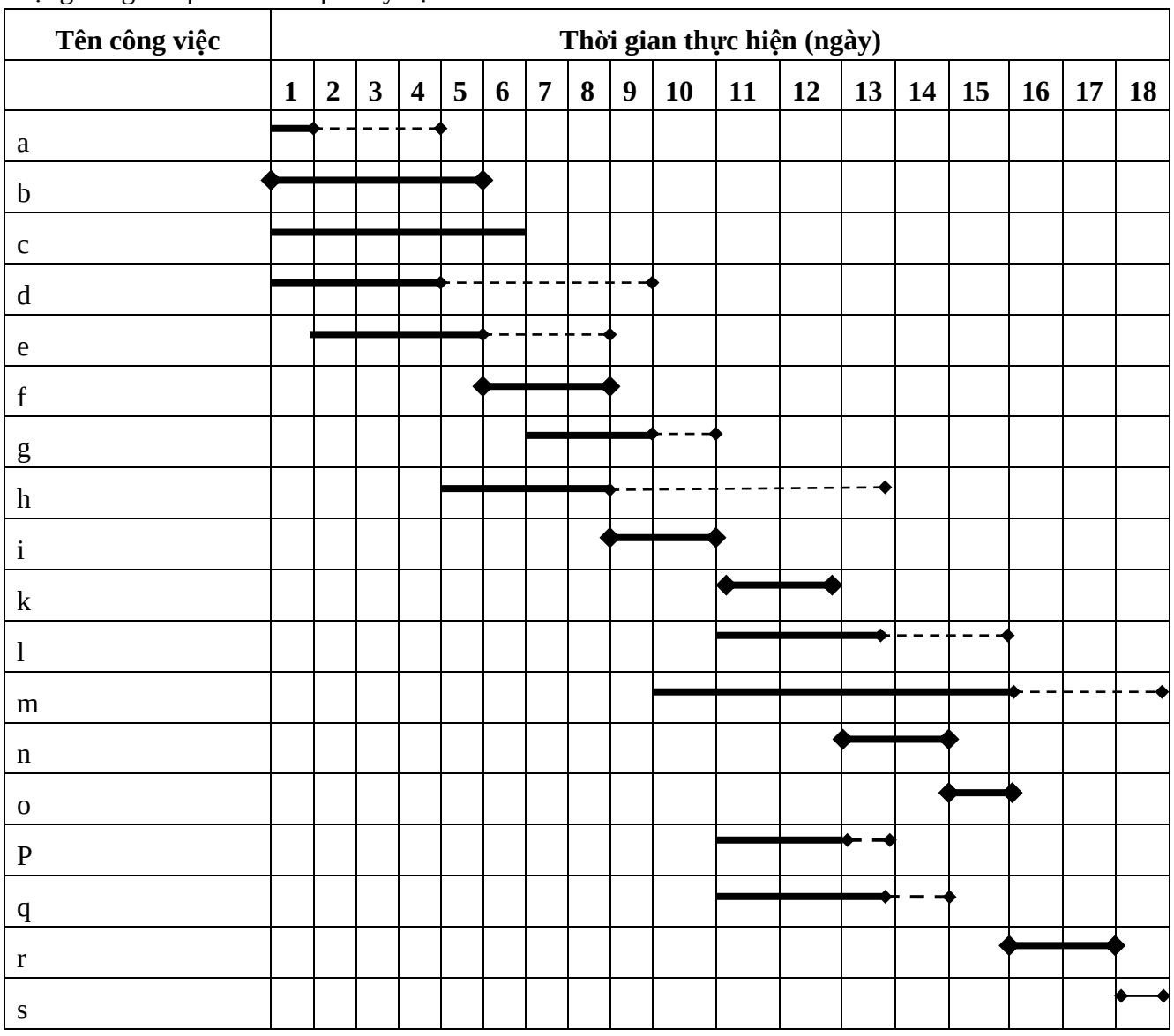
d) Tìm công việc găng và đường Gantt

Những *công việc Gantt* (nét đậm trên hình 3.7) là những công việc có thời gian dự phòng bằng 0 ($t_{df}(cv) = 0$). Khi đó đường Gantt là đường gồm các công việc găng đi từ đỉnh bắt đầu đến đỉnh kết thúc. Độ dài của đường này (18 ngày như hình 3.7b) là độ dài ngắn nhất cần thiết để thực hiện dự án (vì chưa kể đến các điều kiện ràng buộc về các nguồn lực khác). Nếu kéo dài thời gian thực hiện của bất kỳ một công việc nào trên đường găng đều dẫn đến việc kéo dài thời gian thực hiện dự án. Vì thế, các công việc Gantt được đặc biệt chú ý khi theo dõi và giám sát dự án. Trong trường hợp cần thiết, có thể chia nhỏ hơn những công việc Gantt (nếu có thể), vẽ lại mạng công việc và tính toán lại, hy vọng rằng có thể rút ngắn hơn thời gian thực hiện dự án.

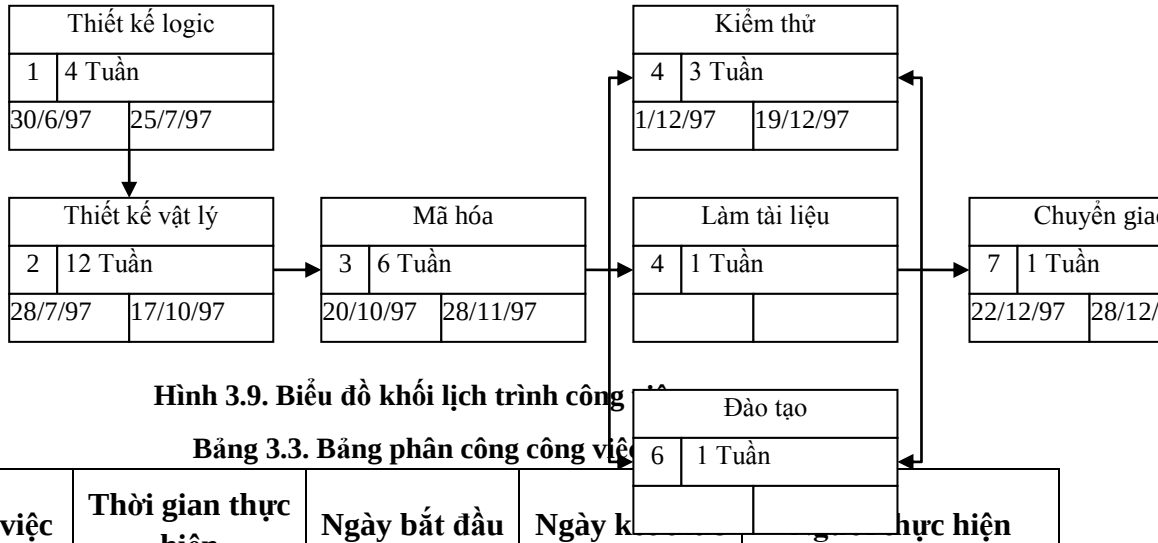
e) Chuyển sang sơ đồ Gantt

Sau khi đã tính toán các tham số của công việc và các đỉnh của mạng công việc, ta có thể chuyển nó sang biểu đồ Gantt (hình 3.8) để dễ quan sát và theo dõi trong quản lý dự án. Trong biểu đồ Gantt, mỗi công việc biểu diễn bằng một khối chữ nhật có độ dài bằng thời gian thực hiện công việc. Điểm bắt đầu của nó được đặt tại thời điểm bằng thời gian bắt đầu sớm nhất của đỉnh mà công việc đó đi ra. Đường nét đứt là thời gian dự phòng của công việc. Những công việc găng có hai đầu phình to. Nhìn trên biểu đồ ta có thể thấy rõ ngày bắt đầu và ngày kết thúc của mỗi công việc cũng như khả năng kéo dài của chúng.

Ngoài sơ đồ mạng của công việc và biểu đồ Gantt, người ta còn dùng biểu đồ công việc dạng khối (hình 3.9) hay dạng lịch trình (bảng 3.3) để trợ giúp hoạt động quản lý dự án. Các sơ đồ và bảng biểu dữ liệu này sau mỗi chu kỳ quản lý lại được cập nhật lại, phục vụ cho chu kỳ tiếp theo. Tất cả các sơ đồ, biểu đồ và các tính toán đi theo nó vừa nêu ở trên có thể tạo ra một cách tự động bằng các phần mềm quản lý dự án.



Hình 3.8. Biểu đồ Gantt lịch trình dự án



Hình 3.9. Biểu đồ khối lịch trình công việ

Bảng 3.3. Bảng phân công công việ

TT	Tên công việc	Thời gian thực hiện	Ngày bắt đầu	Ngày k	hực hiện
1	Thiết kế logic	4	30/6/97	25/7/97	Trần Trung Dũng
2	Thiết kế vật lý	12	28/7/97	17/10/97	Hoàng Tùng
3	Mã hóa	6	20/10/97	28/11/97	Hoàng Lan
4	Kiểm thử	3	1/12/97	19/12/97	Vũ Trung Dũng
5	Làm tài liệu	1			Trần Quốc Cường
6	Đào tạo	1			Đào Xuân Hưng
7	Chuyển giao	1	22/12/97	28/12/97	Nguyễn Thành Trung

3.3. QUẢN LÝ RỦI RO ĐỐI VỚI DỰ ÁN PHÁT TRIỂN PHẦN MỀM

Quản lý rủi ro có thể coi là một trong những công việc chính của người quản lý dự án. Nó liên quan đến việc chống lại những rủi ro mà có thể ảnh hưởng tới lịch làm việc và chất lượng của phần mềm đang được phát triển, từ đó có hành động để tránh những rủi ro này. Kết quả của việc phân tích rủi ro nên đưa ra thành văn bản gắn liền với bản kế hoạch dự án cùng với sự phân tích về hậu quả và nguy cơ xảy ra rủi ro. Việc quản lý rủi ro một cách hiệu quả giúp ta dễ đối phó với những vấn đề phát sinh và không làm cho việc vượt quá ngân sách hoặc vượt quá thời hạn trở nên không kiểm soát nổi.

3.3.1. Khái niệm rủi ro

Một rủi ro là khả năng một vài tình huống bất lợi sẽ xảy ra, đối với dự án phần mềm, rủi ro có thể rơi vào các trường hợp sau:

- Rủi ro của dự án ảnh hưởng tới thời gian biểu hoặc tài nguyên. Ví dụ: mất một người thiết kế có kinh nghiệm.

- Rủi ro của sản phẩm ảnh hưởng tới chất lượng hoặc hiệu quả của phần mềm đã được phát triển. Ví dụ: lỗi của một thành phần mua về để thực hiện không như mong muốn.

- Rủi ro kinh doanh ảnh hưởng tới tổ chức phát triển phần mềm hoặc người mua sản phẩm phần mềm. Ví dụ: Đối thủ cạnh tranh giới thiệu sản phẩm mới có chức năng tương tự.

Tất nhiên những rủi ro này thường xếp chồng lên nhau, nếu một lập trình viên có kinh nghiệm rời bỏ dự án, thì cũng có thể coi đây là một rủi ro dự án vì thời hạn giao nộp sản phẩm có thể bị chậm lại. Nó cũng có thể là một rủi ro của sản phẩm bởi việc thay thế bằng một lập trình viên không có kinh nghiệm làm cho phần mềm có nhiều lỗi, đó cũng có thể là một rủi ro kinh doanh vì người lập trình có kinh nghiệm này sẽ không tham gia vào những dự án tiếp theo.

Sự ảnh hưởng của những rủi ro này còn phụ thuộc vào dự án và môi trường của tổ chức đang phát triển phần mềm. Tuy nhiên, có nhiều rủi ro thực chất chỉ là một và ta có thể phân loại theo bảng 3.4.

Bảng 3.4. Bảng phân loại rủi ro

Rủi ro	Ảnh hưởng	Mô tả
Sự luân chuyển cán bộ	Dự án	Nhân viên có kinh nghiệm sẽ rời khỏi dự án trước khi nó hoàn thành
Thay đổi quản lý	Dự án	Có thay đổi về sự ưu tiên trong quản lý của tổ chức
Phần cứng không sẵn sàng	Dự án	Trang thiết bị được dự trù là cần thiết cho dự án không được chuyển đến đúng thời hạn
Thay đổi yêu cầu	Dự án và sản phẩm	Số lượng những thay đổi đối với các yêu cầu lớn hơn dự đoán
Chậm tiến độ	Dự án và sản phẩm	Những yếu tố cơ bản không sẵn sàng như trong thời gian biểu
Sự đánh giá thấp dự án	Dự án và sản phẩm	Kích thước của hệ thống bị đánh giá thấp so với thực tế
Những công cụ CASE không đủ mạnh	Sản phẩm	Công cụ cung cấp cho dự án không đáp ứng được những yêu cầu ban đầu
Thay đổi công nghệ	Kinh doanh	Những công nghệ cơ bản để xây dựng hệ thống bị thay thế bởi những công nghệ mới

3.3.2. Tiến trình quản lý rủi ro

Quản lý rủi ro đặc biệt quan trọng đối với các dự án phần mềm bởi vì nó luôn gắn liền với tính không chắc chắn, điều mà bất kỳ dự án nào cũng phải đối mặt. Nó có thể bắt nguồn từ việc xác định thiếu các yêu cầu, khó ước lượng thời gian và tài nguyên cho việc phát triển phần mềm, phụ thuộc vào những kỹ năng cá nhân và những thay đổi yêu cầu xuất phát từ việc thay đổi nhu cầu của khách hàng. Người quản lý dự án phải đoán trước được những rủi ro này, hiểu được mức độ ảnh hưởng của nó tới dự án, tới sản phẩm và chiến lược kinh doanh của tổ chức, và cuối cùng là làm thế nào để tránh được những rủi ro. Họ cần phải đưa ra những kế hoạch liên tiếp, để khi có một rủi ro xảy ra sẽ có ngay những hành động để ứng phó. Thông thường, các bước để giải quyết vấn đề này bao gồm:

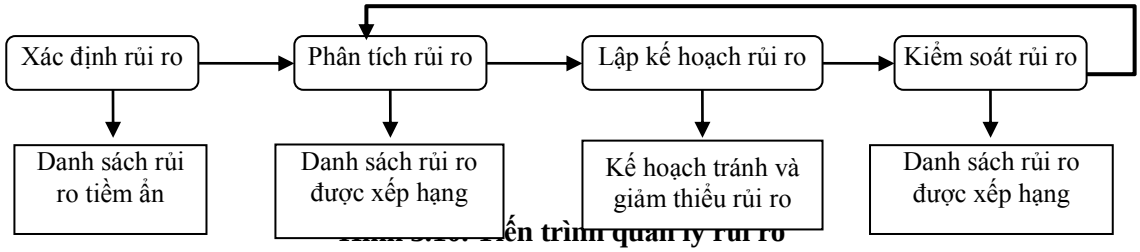
- Xác định rủi ro: Xác định những rủi ro của dự án, của sản phẩm và kinh doanh.

- Phân tích rủi ro: Đánh giá khả năng và hậu quả của những rủi ro này.

- Lập kế hoạch rủi ro: Lập kế hoạch để tránh hoặc giảm thiểu sự ảnh hưởng của rủi ro.

- Kiểm soát rủi ro: Kiểm soát rủi ro trong suốt dự án.

Tiến trình quản lý rủi ro cũng như việc lập kế hoạch dự án là một tiến trình liên tục, xuyên suốt dự án. Vì thế cần phải tài liệu hóa các hậu quả của tiến trình quản lý rủi ro trong một kế hoạch quản lý rủi ro. Tài liệu này bao gồm việc thảo luận về những rủi ro mà dự án sẽ phải đối mặt, bản phân tích rủi ro và những chiến lược đưa ra để kiểm soát những rủi ro này. Ở vị trí thích hợp, người quản lý cũng nên đưa ra kế hoạch của tiến trình quản lý rủi ro. Tiến trình quản lý rủi ro được minh họa trong hình 3.10.



Hình 3.10: Tiến trình quản lý rủi ro

a) Xác định rủi ro

Xác định rủi ro là giai đoạn đầu của tiến trình quản lý rủi ro. Mục đích của giai đoạn này là khám phá ra những rủi ro có thể đối với dự án. Việc xác định rủi ro có thể được thực hiện bằng cách thành lập một nhóm làm việc riêng, sử dụng cách tiếp cận brainstorming (tập hợp những thành viên quan trọng của dự án) hoặc đơn giản chỉ dựa trên kinh nghiệm. Để giúp cho quá trình xác định các rủi ro, người ta đã đưa ra một số loại rủi ro thường gặp:

- *Rủi ro về mặt công nghệ:* liên quan đến các công nghệ về phần cứng và phần mềm được sử dụng để phát triển dự án. Ví dụ: Cơ sở dữ liệu sử dụng trong hệ thống không thể xử lý nhiều giao dịch tại cùng một thời điểm; Các thành phần phần mềm được tái sử dụng có chứa những khuyết điểm làm hạn chế chức năng của hệ thống.

- *Rủi ro về nhân lực:* liên quan đến những nhân viên làm việc trong nhóm phát triển. Ví dụ: không thể thành lập một đội ngũ nhân viên có những kỹ năng theo yêu cầu; những nhân viên quan trọng bị ốm và không thể làm việc tại những thời điểm quan trọng; yêu cầu đào tạo huấn luyện nhân viên không được đáp ứng.

- *Rủi ro về mặt tổ chức:* xuất phát từ môi trường tổ chức, nơi phát triển dự án. Ví dụ: Tổ chức được cấu trúc lại và thay đổi người quản lý dự án; những vấn đề về tài chính của dự án gặp khó khăn làm giảm ngân sách giành cho dự án.

- *Rủi ro về công cụ:* xuất phát từ những công cụ hỗ trợ việc phát triển hệ thống. Chẳng hạn như: mã nguồn được sinh ra bởi công cụ CASE không hiệu quả; các công cụ CASE không thể tích hợp lại với nhau.

- *Rủi ro về yêu cầu phần mềm:* xuất phát từ việc các yêu cầu phần mềm của khách hàng có sự thay đổi. Đặc biệt khi việc thay đổi yêu cầu đòi hỏi phải thiết kế lại những công việc chính, yêu cầu không sử dụng được nữa, khách hàng hiểu sai ảnh hưởng của những thay đổi yêu cầu.

- *Rủi ro do ước lượng:* xuất phát từ việc ước lượng không chính xác về thời gian, tài nguyên... cần để phát triển dự án. Ví dụ: thời gian cần thiết để phát triển phần mềm quá ngắn, tỷ lệ tài nguyên cần cho việc sửa lỗi bị đánh giá thấp, kích thước của phần mềm bị đánh giá thấp…

Khi hoàn thành việc xác định những rủi ro, quản lý dự án nên đưa ra một danh sách những rủi ro có thể xảy ra và đánh giá khả năng xảy ra của những rủi ro này cũng như mức độ ảnh hưởng của nó tới sản phẩm, tới tiến trình và tới mục tiêu kinh doanh của tổ chức.

b) Phân tích rủi ro

Trong quá trình phân tích rủi ro, nhóm quản lý dự án phải xem xét một cách cụ thể từng trường hợp rủi ro đã được xác định và đánh giá khả năng xảy ra cũng như tính nghiêm trọng của mỗi rủi ro. Đây không phải là một công việc đơn giản, người thực hiện phải dựa vào sự phán xét và kinh nghiệm của bản thân, điều này giải thích tại sao lại đòi hỏi những người làm quản lý dự án phải có kinh nghiệm. Việc ước lượng rủi ro nói chung không phải là một con số chính xác, mà người ta có thể dựa vào một khoảng số liệu: Khả năng xảy ra có thể là rất thấp (<10%), thấp (10-25%), trung bình (25-50%), cao (50-75%) hoặc rất cao (>75%). Ảnh hưởng của rủi ro có thể là khủng khiếp, nghiêm trọng, có thể bỏ qua được hoặc không quan trọng.

Bảng 3.5. Bảng đánh giá một số tình huống rủi ro

Rủi ro	Khả năng xảy ra	Ảnh hưởng
Vấn đề tài chính của tổ chức gặp khủng hoảng và phải giảm ngân sách cho dự án	Thấp	Rất nghiêm trọng
Không thể thành lập một đội ngũ nhân viên có những kỹ năng theo yêu cầu	Cao	Rất nghiêm trọng
Những nhân viên quan trọng bị ốm và không thể làm việc tại những thời điểm quan trọng	Trung bình	Nghiêm trọng
Các thành phần phần mềm được sử dụng lại có chứa những khuyết điểm làm hạn chế khả năng của hệ thống	Trung bình	Nghiêm trọng
Việc thay đổi yêu cầu đòi hỏi phải thiết kế lại những công việc chính	Trung bình	Nghiêm trọng
Tổ chức được cấu trúc lại và thay đổi người quản lý dự án	Cao	Nghiêm trọng
Cơ sở dữ liệu sử dụng trong hệ thống không thể xử lý nhiều giao dịch tại cùng một thời điểm	Thấp	Khủng khiếp
Ước lượng: thời gian cần thiết để phát triển quá ngắn	Cao	Nghiêm trọng
Các công cụ CASE không thể tích hợp lại với nhau	Cao	Có thể bỏ qua
Khách hàng hiểu sai ảnh hưởng của những thay đổi yêu cầu	Trung bình	Có thể bỏ qua
Yêu cầu đào tạo huấn luyện nhân viên không được đáp ứng	Trung bình	Có thể bỏ qua
Kích thước của phần mềm bị đánh giá thấp	Cao	Có thể bỏ qua
Tỷ lệ của việc sửa lỗi bị đánh giá thấp	Trung bình	Có thể bỏ qua
Mã nguồn sinh ra bởi công cụ CASE không hiệu quả	Trung bình	Không quan trọng

Bảng 3.5 là các ví dụ minh họa cho các tình huống rủi ro được đề cập ở trên. Tất nhiên, cả tần suất và mức độ nghiêm trọng của mỗi rủi ro đều có sự thay đổi khi những thông tin về rủi ro trở thành có giá trị hơn và khi kế hoạch quản lý rủi ro được thực hiện, vì thế, việc phân tích này cũng cần phải cập nhật một cách thường xuyên. Khi một rủi ro đã được phân tích và xếp hạng, người quản lý nên xem xét xem rủi ro nào có ý nghĩa nhất. Việc đánh giá này phải dựa trên sự kết hợp giữa hai yếu tố: tần xuất và mức độ ảnh hưởng. Những lỗi khủng khiếp luôn luôn phải xem xét, những lỗi nghiêm trọng sẽ phải xem xét nhiều hơn những lỗi có tần xuất xảy ra trung bình hoặc thỉnh thoảng.

c) Lập kế hoạch phòng ngừa – hạn chế rủi ro

Bước lập kế hoạch để phòng ngừa và hạn chế rủi ro cần phải xem xét từng rủi ro chính đã được xác định và đưa ra các chiến lược quản lý. Các chiến lược quản lý rủi ro nhìn chung khá phức tạp, nó cần phải được xem xét và cân nhắc trong từng trường hợp cụ thể. Dưới đây là 5 chiến lược xử lý rủi ro:

- *Chấp nhận rủi ro:* Không làm gì cả. Chiến lược này được sử dụng khi xác suất xảy ra rủi ro và tác động của chúng là tối thiểu, nếu chúng xảy ra cũng dễ dàng xử lý.

- *Tránh rủi ro:* để tránh rủi ro, ta có thể bỏ đi phần dự án liên quan đến rủi ro, tức là làm thay đổi phạm vi dự án hoặc thay đổi phạm vi nghiệp vụ. Ở trường hợp này, những thay đổi cần được chủ dự án và khách hàng chấp nhận, hậu quả tất yếu là thu nhập và chi phí thường giảm đi.

- *Giám sát và chuẩn bị dự phòng:* chọn một chỉ số để xác định xem rủi ro đã đến hay chưa. Ví dụ: rủi ro liên quan đến thầu phụ, cần cập nhật trạng thái tiến độ của họ. Kế hoạch đáp ứng được chuẩn bị trước khi rủi ro xảy ra. Cách thường được thực hiện là lưu lại một phần kinh phí.

- *Chuyển rủi ro cho người khác:* Chuyển rủi ro cho người khác như mua bảo hiểm. Có nhiều phương pháp, như ký một hợp đồng dịch vụ có giá cố định. Khi đó đã chuyển rủi ro cho thầu phụ. Tuy nhiên, trường hợp này cũng có thể làm nảy sinh những rủi ro mới, vì thế cần phải tính toán một cách cụ thể. Phần quan trọng trong chiến lược này là xác định được các điều khoản hợp đồng hiệu quả và quản lý tốt các nhà thầu phụ.

- *Hạn chế rủi ro:* Hạn chế hay giảm tác động rủi ro bằng các biện pháp đầu tư hay nỗ lực nhiều hơn, bao gồm tất cả những gì mà đội dự án có thể làm để vượt qua được rủi ro từ môi trường của dự án.

d) Kiểm soát rủi ro

Là việc đánh giá mỗi rủi ro đã được xác định một cách thường xuyên để quyết định khả năng xảy ra của những rủi ro này là tăng hay giảm, đồng thời đánh giá lại mức độ ảnh hưởng của rủi ro. Những rủi ro quan trọng cần được thảo luận tại những cuộc họp quản lý tiến trình.

Tất nhiên, những công việc này thường không quan sát được một cách trực tiếp, vì thế bạn phải nhìn vào những yếu tố khác để xác định khả năng xảy ra và mức độ ảnh hưởng của rủi ro. Những nhân tố này có thể ảnh hưởng bởi kiểu rủi ro, bảng 3.6 đưa ra một vài ví dụ về các nhân tố có thể giúp bạn đánh giá được rủi ro dựa theo những kiểu rủi ro này.

Bảng 3.6. Các yếu tố rủi ro

Kiểu rủi ro	Chỉ dẫn
Công nghệ	Chậm bàn giao phần cứng hoặc phần mềm trợ giúp, rất nhiều vấn đề công nghệ gián tiếp.
Con người	Tinh thần làm việc của nhân viên thấp, quan hệ giữa những nhân viên trong đội lỏng lẻo, nhân viên có thể có những hoạt động ngoài dự án.
Tổ chức	Tin đồn nhảm về tổ chức, thiếu những hoạt động của người quản lý cấp cao.
Công cụ	Các thành viên trong đội không sẵn sàng sử dụng công cụ, phàn nàn về các công cụ CASE, yêu cầu những công cụ làm việc hiệu quả hơn.
Yêu cầu	Rất nhiều yêu cầu thay đổi, khách hàng phàn nàn.

Ước lượng	Không thực hiện được đúng như thời gian biểu, không sửa được các báo lỗi khiếm khuyết.

3.4. KẾT THÚC DỰ ÁN

Đây là thời điểm hoàn tất dự án. Kết thúc dự án diễn ra sau giai đoạn triển khai thực hiện. Giai đoạn bảo trì thường được coi là việc tiếp tục các dự án khác, các dự án này sẽ được quản lý riêng. Kết thúc dự án bao gồm các công việc sau:

- Đóng dự án: Để đánh dấu sự hoàn tất của một dự án, cần thực hiện một số hoạt động như đánh giá các thành viên, kiến nghị các lợi ích cho họ, hoàn thiện các tài liệu và chứng từ thanh toán. Cảm ơn những người đóng góp, tham gia và hỗ trợ trong quá trình công việc.

- Tổng kết sau dự án: Mục tiêu của hoạt động này xác định và đánh giá được mặt mạnh, mặt yếu của sản phẩm dự án, của quá trình phát triển sản phẩm, quá trình quản lý dự án. Từ đó rút ra những kinh nghiệm cần thiết cho các dự án sau.

- Kết thúc mọi hợp đồng: Ký kết các bản thanh lý hợp đồng với khách hàng và các nhà cung cấp.

3.5. CẤU TRÚC TÀI LIỆU QUẢN LÝ DỰ ÁN

1. GIỚI THIỆU CHUNG

1.1. Mô tả tóm tắt dự án

1.2. Các giả thiết và ràng buộc

2. TỔ CHỨC DỰ ÁN

2.1. Cấu trúc tổ chức

2.2. Các thành viên trong đội dự án

3. RỦI RO CỦA DỰ ÁN

3.1. Danh sách rủi ro

3.2. Đánh giá và quản lý rủi ro

4. CÔNG CỤ VÀ CƠ SỞ HẠ TẦNG THỰC HIỆN DỰ ÁN

4.1. Phần cứng

4.2. Phần mềm

4.3. Cơ sở hạ tầng khác

5. PHÂN CÔNG CÔNG VIỆC CỦA DỰ ÁN

5.1. Các phụ thuộc quan trọng

5.2. Các mốc thời gian quan trọng

5.3. Các thời điểm cần bàn giao sản phẩm

6. THỜI GIAN BIỂU CỦA DỰ ÁN

7. TRAO ĐỔI THÔNG TIN TRONG DỰ ÁN

7.1. Trao đổi thông tin giữa các thành viên trong đội dự án

7.2. Trao đổi thông tin với ban quản lý cấp cao

7.3. Trao đổi thông tin với khách hàng

7.4. Trao đổi thông tin với các đối tượng khác

CÂU HỎI ÔN TẬP

1. Hãy giải thích tại sao cần phải có nhóm chịu trách nhiệm quản lý dự án phần mềm.
2. Phân tích những khó khăn trong quản lý dự án phần mềm so với các dự án khác.
3. Nêu những công việc cơ bản của người làm quản lý dự án.
4. Phân tích tiến trình lập kế hoạch dự án. Kế hoạch dự án là sản phẩm làm một lần hay thường xuyên thay đổi trong suốt thời gian tồn tại của dự án?
5. Mốc thời gian quan trọng của dự án là gì? Một dự án có những mốc thời gian quan trọng nào? Mốc thời gian quan trọng và thời điểm bàn giao sản phẩm phần mềm có phải là một hay không?
6. Vai trò của quản lý rủi ro trong tiến trình phần mềm? Có những loại rủi ro nào?
7. Có bao nhiêu chiến lược quản lý rủi ro? Chiến lược nào nên tránh, chiến lược nào cần được ưu tiên?

Chương 4: XÁC ĐỊNH VÀ ĐẶC TẢ YÊU CẦU PHẦN MỀM

Yêu cầu đối với một hệ thống phần mềm là những mô tả về các dịch vụ được cung cấp bởi hệ thống, những ràng buộc đối với các chức năng và quá trình phát triển sản phẩm. Những yêu cầu này phản ánh nhu cầu của khách hàng đối với hệ thống. Qua đó sẽ giúp họ giải quyết những vấn đề của họ, chẳng hạn như hệ thống điều khiển một thiết bị, đặt hàng và tìm kiếm thông tin sản phẩm, quản lý tài chính, quản lý nhân sự... Tiến trình tìm kiếm, phân tích, tài liệu hóa các yêu cầu phần mềm, kiểm tra các dịch vụ và các ràng buộc gọi là kỹ nghệ yêu cầu (RE – Requirement Engineering).

Chương này giới thiệu các nội dung chính sau: định nghĩa về các kiểu yêu cầu phần mềm; các mức và các hình thức diễn tả yêu cầu phần mềm; tìm hiểu về kỹ nghệ yêu cầu và đặc tả tài liệu yêu cầu phần mềm.

4.1. TỔNG QUAN VỀ YÊU CẦU PHẦN MỀM

4.1.1. Khái niệm yêu cầu phần mềm

Trước khi tiến hành xây dựng một hệ thống phần mềm, nhóm phát triển cần phải hiểu được bản chất của vấn đề cần giải quyết. Đây thường là một công việc phức tạp, nhất là với những dự án mới hay một hệ thống đang tồn tại được dùng làm mô hình cho phần mềm muốn phát triển. Trên thực tế, người ta thường không phân biệt được hai khái niệm ***yêu cầu*** và ***nhu cầu***, đôi khi chúng được hiểu như nhau. Một tổ chức có thể quyết định rằng họ "*cần một hệ thống phần mềm trợ giúp công việc kế toán*". Nhưng việc đưa ra một nhu cầu đơn giản như thế cho nhóm phát triển phần mềm và hy vọng có được một hệ thống phần mềm chấp nhận được và dùng tốt là điều không thể. Thông tin của vấn đề cần được giải quyết phải được thu thập, phân tích, xác định nội dung vấn đề một cách rõ ràng, đầy đủ mới có được những yêu cầu chính xác cho hệ thống. Khi đó mới đưa ra được một giải pháp cho việc thiết kế và thực thi hệ thống đáp ứng được những yêu cầu đã đề ra.

Quá trình thiết lập các dịch vụ mà hệ thống phải cung cấp và các ràng buộc mà hệ thống phải tuân thủ khi hoạt động hay phát triển gọi là tìm hiểu và phân tích yêu cầu. Kết quả của công việc này là *bản đặc tả yêu cầu*, đây thường là tư liệu chính thức đầu tiên được tạo ra trong tiến trình phần mềm. Yêu cầu cho một hệ thống phần mềm mô tả những công việc mà hệ thống sẽ làm và những ràng buộc mà nó phải tuân thủ khi hoạt động. Yêu cầu có thể là ***yêu cầu chức năng*** (các chức năng, dịch vụ) hoặc ***yêu cầu phi chức năng*** (các ràng buộc).

Một vấn đề thường gặp khi xác định yêu cầu là sự mô tả không tách biệt rõ ràng giữa các mức của yêu cầu. Ta thường dùng thuật ngữ "*yêu cầu người dùng*" để mô tả yêu cầu tóm tắt ở mức cao, và thuật ngữ "*yêu cầu hệ thống*" để mô tả cụ thể công việc hệ thống sẽ làm. Ngoài hai mức trên, một mức mô tả chi tiết hơn – *đặc tả thiết kế phần mềm* là cầu nối giữa tiến trình xác định yêu cầu và các hoạt động thiết kế sau này. Các yêu cầu phân theo mức trừu tượng có thể định nghĩa như sau:

1. Yêu cầu người sử dụng (user requirements) là những yêu cầu phát biểu bằng ngôn ngữ tự nhiêu cùng biểu đồ để mô tả các dịch vụ mà hệ thống cung cấp và ràng buộc khi nó hoạt động. Đó là những phát biểu hướng người dùng.

2. Yêu cầu hệ thống (system requirements) nêu ra các dịch vụ của hệ thống và chi tiết các ràng buộc của nó. Tài liệu này đôi khi còn gọi là đặc tả chức năng – cần phải rõ ràng và chính xác. Nó được sử dụng làm cơ sở cho hợp đồng giữa người mua và người phát triển phần mềm.

3. Đặc tả phần mềm (software specification) là sự mô tả khái quát các chức năng phần mềm trợ giúp hoạt động nghiệp vụ, làm cơ sở để thiết kế và triển khai phần mềm sau này. Tài liệu đặc tả phần mềm được bổ sung thêm các chi tiết để trở thành tài liệu đặc tả yêu cầu hệ thống.

Hình 4.1 là ví dụ về yêu cầu người dùng của chức năng trong hệ thống quản lý thư viện và sự chi tiết hóa yêu cầu người dùng này thành yêu cầu hệ thống.

Yêu cầu người dùng

Hệ thống phải lưu trữ được toàn bộ thông tin cần thiết để quản lý bản quyền tác giả

Đặc tả yêu cầu hệ thống

1.1. Với mỗi yêu cầu mượn tài liệu, người sử dụng phải điền vào một mẫu thông tin cá nhân chính xác

1.2. Mẫu này sẽ được lưu trữ ít nhất trong vòng 5 năm

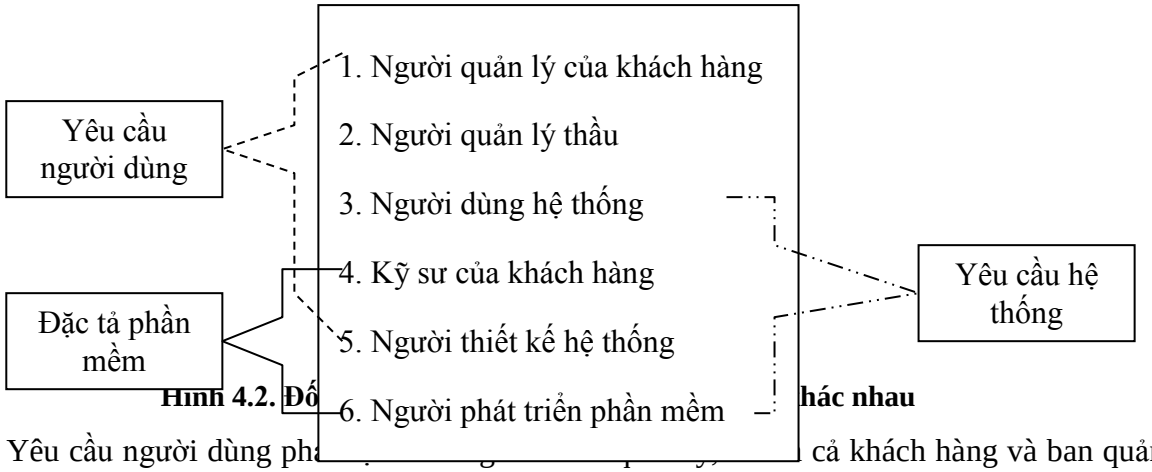
1.3. Các mẫu này phải được chỉ số hoá bởi người sử dụng, bởi tài liệu được yêu cầu và bởi tác giả tài liệu.

1.4. Hệ thống phải yêu cầu người sử dụng đăng nhập trong mỗi lần mượn tài liệu

1.5. Với những tài liệu đang được mượn bởi người sử dụng, hàng tháng phải gửi những thông tin về tài liệu cho người mượn

Hình 4.1. Yêu cầu người dùng và yêu cầu hệ thống

Việc mô tả các yêu cầu của hệ thống ở các mức trừu tượng khác nhau là rất cần thiết, vì chúng giúp truyền đạt thông tin về hệ thống tới các đối tượng người đọc khác nhau. Hình vẽ 4.2 biểu diễn các lớp người đọc tương ứng với các mức trừu tượng khác nhau của tài liệu yêu cầu.



Hình 4.2. Đố[illegible]hác nhau

Yêu cầu người dùng ph[illegible] cả khách hàng và ban quản lý đều hiểu được. Trong khi đó, đặc tả yêu cầu hệ thống lại hướng tới nhóm kỹ thuật và những người quản lý dự án. Người sử dụng hệ thống có thể đọc cả hai loại tài liệu nói trên. Cuối cùng, đặc tả yêu cầu phần mềm là tài liệu hướng tới việc triển khai. Nó được viết cho các kỹ sư phần mềm và những người tham gia vào quá trình phát triển hệ thống.

4.1.2. Phân loại yêu cầu phần mềm

Yêu cầu hệ thống thường được phân thành các loại: yêu cầu chức năng, yêu cầu phi chức năng và yêu cầu miền lĩnh vực ứng dụng phần mềm.

- *Yêu cầu chức năng:* là những phát biểu về các dịch vụ mà hệ thống cần phải cung cấp, hệ thống phản ứng như thế nào với các dữ liệu đầu vào cụ thể và hệ thống cần phải ứng xử như thế nào trong các tình huống cụ thể. Trong một số trường hợp, yêu cầu chức năng cũng có những phát biểu cụ thể về những gì mà chúng không thực hiện được.

- *Yêu cầu phi chức năng:* là những ràng buộc đối với các dịch vụ hoặc những chức năng được đưa ra bởi hệ thống. Bao gồm những ràng buộc về thời gian, ràng buộc về tiến trình và các chuẩn phải sử dụng. Yêu cầu phi chức năng thường áp dụng cho những hệ thống hoàn chỉnh chứ không chỉ áp dụng cho một tính năng hoặc một dịch vụ của hệ thống.

- *Yêu cầu lĩnh vực:* những yêu cầu này đến từ lĩnh vực ứng dụng của hệ thống. Nó phản ánh những đặc tính cũng như những ràng buộc của lĩnh vực. Chúng có thể là yêu cầu chức năng hoặc yêu cầu phi chức năng.

Trên thực tế, việc phân biệt giữa các kiểu yêu cầu không thể thực hiện một các rõ ràng như khi định nghĩa. Một yêu cầu người dùng liên quan tới tính an toàn, người ta có thể coi đó là một yêu cầu phi chức năng. Tuy nhiên, khi phát triển một cách chi tiết hơn, những yêu cầu này có thể sinh ra những yêu cầu chức năng khác, chẳng hạn như cần phải có chức năng đăng nhập, phân quyền khi sử dụng hệ thống.

a) Yêu cầu chức năng

Những yêu cầu chức năng của hệ thống mô tả hệ thống sẽ làm gì. Những yêu cầu này phụ thuộc vào kiểu phần mềm sẽ phát triển, đối tượng sử dụng hệ thống và cách tiếp cận nói chung được tổ chức sử dụng khi viết yêu cầu. Khi diễn tả một yêu cầu người dùng, những yêu cầu này thường được diễn tả một cách khá trừu tượng. Tuy nhiên, những yêu cầu chức năng của hệ thống lại mô tả chức năng của hệ thống một cách rất chi tiết với đầu vào và đầu ra, các trường hợp ngoại lệ… Yêu cầu chức năng cho một hệ thống phần mềm có thể được diễn tả theo các cách khác nhau. Ví dụ ở đây là các chức năng cho một hệ thống thư viện của một trường đại học có tên là LIBSYS, được sử dụng bởi sinh viên và các giảng viên để đặt trước sách và tài liệu từ các thư viện khác nhau:

- Người dùng có thể tìm kiếm tất cả tập hợp thông tin ban đầu trong CSDL hoặc lựa chọn một phần trong CSDL đó.

- Hệ thống phải cung cấp một giao diện thích hợp để người sử dụng có thể đọc tài liệu trong kho dữ liệu.

- Mỗi hóa đơn đặt sách phải có một mã số duy nhất, người dùng có thể sao chép sang một vùng lưu trữ riêng thuộc tài khoản của mình.

Những yêu cầu chức năng người dùng này xác định những chức năng cụ thể được cung cấp bởi hệ thống. Những chức năng này được lấy từ tài liệu yêu cầu người dùng và chúng minh họa rằng các yêu cầu chức năng có thể được viết ở những mức độ chi tiết khác nhau.

Việc thiếu chính xác khi diễn tả yêu cầu là nguyên nhân dẫn đến nhiều vấn đề trong kỹ nghệ phần mềm. Nó là nguồn gốc dẫn đến việc người phát triển hệ thống có thể suy diễn một yêu cầu nhập nhằng thành một yêu cầu dễ thực hiện hơn. Tuy nhiên, thông thường nó lại không hoàn toàn giống với yêu cầu của khách hàng. Những yêu cầu mới lại được sinh ra và thay đổi việc thực hiện hệ thống. Điều này sẽ dẫn tới việc chậm trễ bàn giao sản phẩm và tăng chi phí phát triển.

Chúng ta có thể xem xét ví dụ thứ hai cho hệ thống thư viện có đề cập đến việc "hiển thị hợp lý" các tài liệu được cung cấp bởi hệ thống. Hệ thống thư viện có thể phân phối tài liệu qua nhiều định dạng khác nhau. Khi quan tâm đến yêu cầu này, có nghĩa là hệ thống phải hiển thị được thông tin ở tất cả các loại định dạng khác nhau đó. Tuy nhiên, yêu cầu này được xem là không rõ ràng, vì nó không chỉ ra được một cách cụ thể rằng hệ thống phải cung cấp chức năng hiển thị cho mỗi kiểu định dạng tài liệu khác nhau. Người phát triển dưới áp lực là phải hoàn thành nhanh công việc, có thể đưa ra chức năng hiển thị văn bản và cho rằng nó đã thỏa mãn yêu cầu người dùng.

Về cơ bản, đặc tả yêu cầu chức năng của một hệ thống phải đầy đủ và đồng nhất. Đầy đủ có nghĩa là tất cả các dịch vụ mà người dùng yêu cầu đều phải được xác định. Đồng nhất có nghĩa là không có những phát biểu mâu thuẫn nhau. Trên thực tế, với những hệ thống lớn và phức tạp, khó có thể đưa ra được các yêu cầu đồng nhất và đầy đủ.

Một lý do khác cho vấn đề này là rất dễ mắc lỗi và bỏ sót khi viết đặc tả cho những hệ thống lớn và phức tạp hoặc các tác nhân chính khác nhau của hệ thống sẽ có những yêu cầu khác nhau, đôi khi trái ngược nhau. Những vấn đề này chỉ được phát hiện khi bước sang giai đoạn phân tích và đôi khi sau giai đoạn lập trình hoặc tới khi chuyển giao cho khách hàng.

b) Yêu cầu phi chức năng

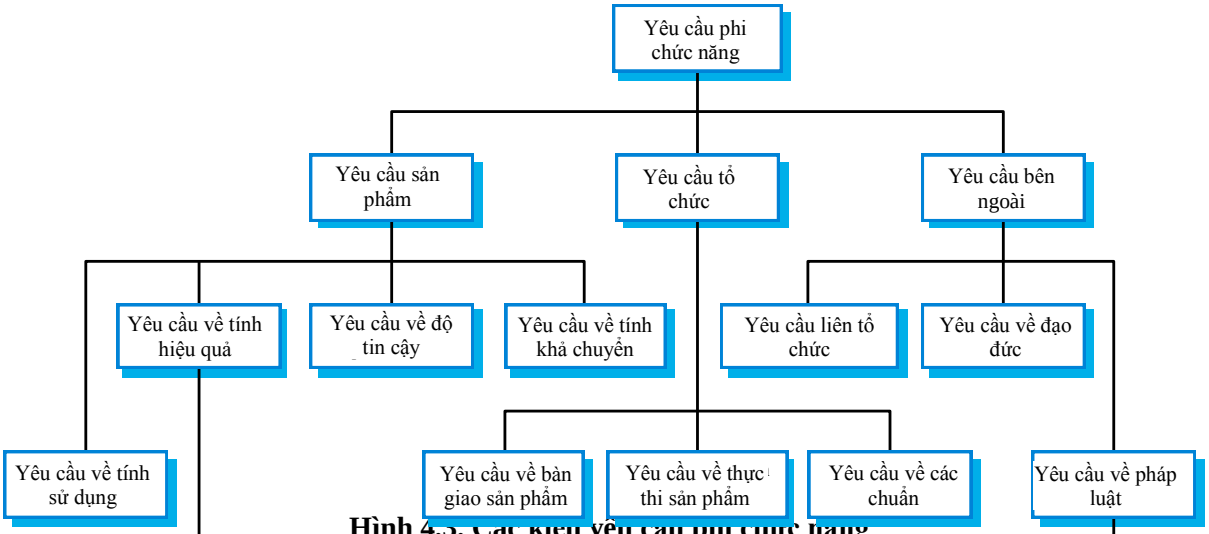
Yêu cầu phi chức năng, giống như tên gọi của nó, là những yêu cầu không liên quan trực tiếp tới những chức năng cụ thể mà hệ thống sẽ cung cấp. Chúng có thể liên quan tới những thuộc tính của hệ thống, chẳng hạn như tính tin cậy, thời gian trả lời và không gian lưu trữ. Chúng cũng có thể xác định những ràng buộc đối với hệ thống, chẳng hạn như khả năng của các thiết bị vào ra và dữ liệu sẽ được hiển thị như thế nào trong giao diện hệ thống.

Những yêu cầu phi chức năng hiếm khi được kết hợp với những chức năng cụ thể của hệ thống. Thông thường, những yêu cầu này chỉ ra hoặc ràng buộc với những thuộc tính của hệ thống. Tuy nhiên, chúng có thể xác định tính hiệu năng, tính an toàn, tính sẵn sàng và những thuộc tính nổi bật khác của hệ thống. Điều này có nghĩa là chúng thường mang tính phê bình hơn là những yêu cầu chức năng riêng rẽ. Người sử dụng hệ thống có thể tìm ra cách làm việc với một chức năng của hệ thống mà không đáp ứng được nhu cầu thực sự của họ.

Tuy nhiên, việc không đáp ứng được những yêu cầu phi chức năng có thể làm cho toàn bộ hệ thống không sử dụng được. Ví dụ: nếu một hệ thống điều khiển trên máy bay không đáp ứng được yêu cầu về tính tin cậy, nó sẽ không được xác nhận để bán cho người sử dụng; nếu một hệ thống thời gian thực không đáp ứng được yêu cầu về tính hiệu năng, các chức năng điều khiển sẽ không thực thi đúng. Các yêu cầu phi chức năng không chỉ liên quan tới hệ thống phần mềm được phát triển, một số yêu cầu phi chức năng có thể ràng buộc cho cả tiến trình được sử dụng để phát triển hệ thống. Chẳng hạn, những yêu cầu về tiến trình bao gồm việc xác định các chuẩn chất lượng sẽ được sử dụng trong tiến trình, một bản đặc tả thiết kế phải được sinh ra bởi một tập công cụ CASE cụ thể và một cách mô tả tiến trình cần phải tuân theo.

Các yêu cầu phi chức năng được sinh ra từ nhu cầu của người dùng, bởi những ràng buộc về ngân sách, bởi những chính sách của tổ chức, bởi sự tương hợp với các hệ thống phần cứng và các hệ thống phần mềm khác, hoặc bởi những nhân tố bên ngoài, chẳng hạn như quy định của pháp luật về tính an toàn hoặc quyền riêng tư. Hình 4.3 phân loại các yêu cầu phi chức năng,

chúng ta có thể nhìn thấy trong hình những yêu cầu phi chức năng này đến từ những đặc tính của phần mềm, đến từ tổ chức phát triển phần mềm hoặc từ các nguồn bên ngoài.



Hình 4.3. Các kiểu yêu cầu phi chức năng

- *Yêu cầu sản phẩm:* những yêu cầu này xác định cách thức thực hiện của sản phẩm. Ch[illegible] về tính hiệu năng liên quan tới việ[illegible] ó cần là bao nhiêu. Những yêu cầu [illegible] lệ lỗi có thể chấp nhận được; yêu cầu về tính khả chuyển, yêu cầu về tính sử dụng.

- *Những yêu cầu của tổ chức:* những yêu cầu này thường xuất phát từ những chính sách và thủ tục trong tổ chức của khách hàng và của nhóm phát triển. Ví dụ: các chuẩn tiến trình phải được sử dụng, yêu cầu về cách thực hiện như ngôn ngữ lập trình hoặc phương pháp thiết kế được sử dụng. Những yêu cầu về việc bàn giao, chẳng hạn như việc xác định khi nào sản phẩm và các tài liệu liên quan phải được bàn giao cho khách hàng.

- *Những yêu cầu bên ngoài:* nhóm này bao gồm tất cả những yêu cầu xuất phát từ những nhân tố bên ngoài hệ thống và tiến trình phát triển của nó. Chúng có thể bao gồm những yêu cầu về tính tương hợp để xác định xem hệ thống sẽ tương tác với các hệ thống trong các tổ chức khác, những yêu cầu pháp lý phải tuân thủ để đảm bảo rằng việc thực thi hệ thống tuân theo đúng pháp luật và những yêu cầu mang tính đạo đức. Những yêu cầu về tính đạo đức là những yêu cầu đối với một hệ thống để đảm bảo rằng nó sẽ được người sử dụng trực tiếp cũng như công chúng chấp nhận.

Một vấn đề chung đối với các yêu cầu phi chức năng là chúng có thể khó xác định. Người sử dụng và khách hàng thường phát biểu những yêu cầu này một cách chung chung, chẳng hạn như tính dễ sử dụng, khả năng của hệ thống bao gồm từ việc kiểm soát lỗi của người sử dụng đến việc trả lời nhanh các yêu cầu của người dùng. Những mục tiêu mơ hồ này là nguyên nhân dẫn đến những vấn đề đối với người phát triển để họ tránh khỏi việc vi phạm và giải thích cho những tranh cãi sau này. Nếu có thể, ta nên viết những yêu cầu phi chức năng một cách định lượng, để chỉ ra rằng mục tiêu này có thể xác nhận được. Bảng 4.1 chỉ ra một số thước đo mà ta có thể sử dụng để xác định những yêu cầu phi chức năng của hệ thống. Từ đó có thể đo lường những thuộc tính này khi hệ thống được kiểm thử và kiểm tra xem nó có đáp ứng được những yêu cầu phi chức năng hay không.

Bảng 4.1. Thước đo định lượng các thuộc tính phi chức năng

Thuộc tính	Thước đo
Tốc độ	Số giao dịch được xử lý/giây Thời gian trả lời một sự kiện/người dùng Thời gian làm mới màn hình
Kích thước	M Bytes Dung lượng bộ nhớ ROM/RAM
Tính dễ sử dụng	Thời gian huấn luyện Số màn hình trợ giúp
Độ tin cậy	Thời gian trung bình kiểm soát lỗi Phần trăm thời gian hệ thống không thực hiện Tỷ lệ lỗi xảy ra Tính sẵn sàng
Sức kháng cự	Thời gian để khởi động lại sau một lỗi Phần trăm của các sự kiện phát sinh lỗi Xác suất của việc sai lệch dữ liệu khi có lỗi
Tính khả chuyển	Lựa chọn ngôn ngữ cho giao diện phần mềm Lựa chọn hệ thống cho việc cài đặt phần mềm

Tuy nhiên, trong thực tế, khách hàng khó có thể đưa ra được những yêu cầu mang tính định lượng. Đối với một số mục tiêu, chẳng hạn như tính bảo trì, không có thước đo nào có thể sử dụng được. Trong trường hợp khác, thậm chí là yêu cầu phi chức năng mang tính định lượng, khách hàng cũng không thể tìm ra được mối liên hệ giữa nhu cầu của họ với những đặc tả này. Họ không hiểu được độ tin cậy có nghĩa là gì bằng kinh nghiệm hàng ngày của họ đối với hệ thống máy tính. Hơn nữa, chi phí để xác định tính định lượng cho những yêu cầu phi chức năng có thể rất lớn. Và khách hàng trả tiền cho hệ thống không nghĩ gì tới những chi phí này. Tuy nhiên, trong các tài liệu yêu cầu thường bao gồm cả mục tiêu lẫn với các yêu cầu. Những mục tiêu này có thể được sử dụng cho các nhà phát triển bởi chúng chỉ ra những ưu tiên của khách hàng. Ví dụ: yêu cầu chỉ ra dung lượng bộ nhớ tối đa mà hệ thống có thể sử dụng nhỏ hơn 4 Mbytes. Những ràng buộc về bộ nhớ là những ràng buộc chung cho các hệ thống nhúng. Một yêu cầu khác, chẳng hạn như chương trình được viết bằng ngôn ngữ Ada, một ngôn ngữ thường được sử dụng cho các hệ thống thời gian thực và các hệ thống quan trọng, tuy nhiên, ta lại không thể biên dịch một chương trình viết bằng Ada với yêu cầu dung lượng bộ nhớ nhỏ hơn 4 Mbytes. Tuy nhiên, ta có thể thương lượng giữa các yêu cầu này: phát triển phần mềm bằng ngôn ngữ khác hoặc thêm bộ nhớ cho hệ thống.

Việc phân biệt được những yêu cầu chức năng và yêu cầu phi chức năng trong tài liệu yêu cầu trong thực tế là rất khó. Nếu yêu cầu phi chức năng được mô tả riêng biệt với yêu cầu chức

năng, thì chúng ta khó xác định được những mối liên hệ giữa chúng. Nếu những yêu cầu phi chức năng được kết hợp với những yêu cầu chức năng thì khó phân biệt được chúng cũng như xác định xem yêu cầu phi chức năng này là của một chức năng riêng biệt hay của toàn bộ hệ thống. Tuy nhiên, đối với những yêu cầu rõ ràng, chẳng hạn như tính hiệu năng hoặc độ tin cậy, ta có thể tách chúng thành những phần riêng biệt trong tài liệu yêu cầu hoặc phân biệt chúng với các yêu cầu khác của hệ thống.

Yêu cầu của sản phẩm:

4.C.8 Nó có thể dùng cho mọi giao tiếp cần thiết giữa APSE và người dùng để thể hiện một tập các ký tự chuẩn của Ada.

6.D.2 Những người dùng kinh nghiệm phải học được cách sử dụng hệ thống tối đa sau 2 giờ đào tạo. Sau thời gian đào tạo trên thì số lỗi trung bình xảy ra đối với người dùng có kinh nghiệm là không quá 2 lỗi trong một ngày.

Yêu cầu mang tính tổ chức:

9.3.2 Quá trình phát triển hệ thống và phân phối tài liệu phải phù hợp với tiến trình và sản phẩm được xác định trong chuẩn XYZCo – SP – STAND95.

Những yêu cầu mở rộng:

7.6.5. Hệ thống cung cấp khả năng cho phép bất cứ người dùng nào cũng có thể kiểm tra dữ liệu cá nhân của mình khi được đưa vào hệ thống. Mỗi thủ tục phải được định nghĩa và phần mềm trợ giúp người dùng có thể sửa lỗi dữ liệu cá nhân của mình.

Hình 4.4. Ví dụ về các yêu cầu phi chức năng

c) Yêu cầu về lĩnh vực

Những yêu cầu này xuất phát từ lĩnh vực mà phần mềm sẽ được sử dụng hơn là những yêu cầu của người dùng. Nó thường bao gồm những thuật ngữ hoặc những khái niệm của lĩnh vực. Chúng có thể là những yêu cầu chức năng mới trong quyền hạn của họ, những ràng buộc của yêu cầu chức năng đang tồn tại hoặc tập những tính toán cụ thể phải được thực hiện. Vì những yêu cầu này là cụ thể, nên những kỹ sư phần mềm thường khó hiểu được mối liên hệ giữa chúng với những yêu cầu hệ thống khác.

Những yêu cầu lĩnh vực là quan trọng bởi chúng thường phản ánh những vấn đề cốt lõi của lĩnh vực ứng dụng. Nếu những yêu cầu này không được thỏa mãn, nó sẽ dẫn tới việc hệ thống không thể làm việc. Ví dụ: xét các yêu cầu sau đây đặt ra cho hệ thống thông tin thư viện:

- Có một giao diện người dùng chuẩn cho tất cả các CSDL dựa trên chuẩn Z39.50;

- Do hạn chế bởi bản quyền tác giả, một số tài liệu phải được phát hiện ngay khi người dùng truy cập tới nó;

- Dựa vào yêu cầu người dùng, những tài liệu này sẽ phải xác định được địa chỉ của máy chủ mà người sử dụng kết nối vào máy in.

Yêu cầu đầu tiên là một ràng buộc trong thiết kế, nó chỉ ra rằng giao diện người dùng để kết nối vào CSDL phải thực hiện theo một thư viện chuẩn xác định. Tuy nhiên, người phát triển phải đưa ra được chuẩn trước khi bắt đầu thiết kế giao diện. Yêu cầu thứ hai được đưa ra vì luật bản quyền tác giả được áp dụng trong hệ thống thư viện. Yêu cầu này hạn chế một số tài liệu không được chuyển ra máy in, hay nói cách khác, người sử dụng không có quyền sở hữu riêng một số tài liệu.

4.2. TIẾN TRÌNH KỸ NGHỆ YÊU CẦU

4.2.1. Khảo sát hệ thống và phân tích tính khả thi

Sau khi được khách hàng đặt hàng, nhóm phát triển cần tiến hành khảo sát hệ thống đang vận hành để thu thập thông tin phục vụ việc phân tích tính khả thi. Trên cơ sở thông tin thu được và những yêu cầu của khách hàng, nhóm nghiên cứu phải đưa ra một số phương án về việc xây dựng hệ thống được yêu cầu. Đối với một dự án, cần đưa ra ít nhất ba phương án có thể: phương án thấp, phương án trung bình và phương án cao. Các nhà phân tích tiến hành phân tích các phương án, lập luận về tính khả thi và chọn phương án thích hợp nhất. Phân tích tính khả thi và phân tích rủi ro có liên quan với nhau trên nhiều khía cạnh, theo nhiều cách khác nhau. Nếu rủi ro của dự án là lớn thì tính khả thi của việc tạo ra phần mềm chất lượng sẽ giảm. Phân tích tính khả thi thường tập trung vào các khía cạnh:

1. Khả thi về kinh tế: chi phí phát triển cần phải cân xứng với khả năng và lợi ích cuối cùng mà hệ thống được xây dựng đem lại.

2. Khả thi về mặt kỹ thuật: Vấn đề chức năng, hiệu năng và ràng buộc có thể ảnh hưởng tới khả năng của hệ thống được chấp nhận. Mặt khác, cần phải xem xét các khả năng kỹ thuật có đủ đảm bảo để thực hiện các giải pháp công nghệ được áp dụng cho hệ thống không.

3. Khả thi về pháp lý: Loại trừ bất kỳ một sự xâm phạm, vi phạm hay khó khăn nào về pháp lý có thể nảy sinh từ việc xây dựng hệ thống.

4. Khả thi về hoạt động: Đánh giá tính khả thi của phương án khi hệ thống được xây dựng đi vào hoạt động. 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 có.

5. Khả thi về thời gian: Dự án có thể hoàn thành trong thời gian dự kiến.

Nghiên cứu về tính khả thi không đảm bảo chắc chắn rằng dự án không còn phải đối mặt với rủi ro. Tuy nhiên, một trong những yếu tố trên không được đáp ứng thì nguy cơ thất bại của dự án rất lớn và cần phải tiến hành nghiên cứu lại. Bản nghiên cứu về tính khả thi cần phải được cung cấp cho người quản lý cấp cao và đính kèm vào dự án như phụ lục.

Bản nghiên cứu tính khả thi cần được ban quản lý dự án xem xét và đánh giá độ tin cậy, độ chính xác về nội dung, sau đó cấp quản lý cao hơn sẽ xét duyệt để ra quyết định xem có nên tiến hành dự án hay không. Trong trường hợp dự án được tiếp tục, thì đây là cơ sở để quyết định khác sẽ được đưa ra trong bước lập kế hoạch. Sau khi giải pháp và phương án đã được thông qua, nhóm phân tích cần lập hồ sơ nhiệm vụ. Đây là cơ sở để xây dựng các thỏa thuận chưa chính thức giữa ba bên: người phân tích, nhà đầu tư, khách hàng và người sử dụng.

4.2.2. Tiến trình phát hiện và phân tích yêu cầu

Sau khi có phương án khả thi, đội dự án cần tiếp tục thu thập thông tin, xác định yêu cầu và tiến hành phân tích. Quá trình này liên quan tới nhiều đối tượng khác nhau như: người quản lý, kỹ sư dự án, khách hàng, người dùng, chuyên gia, nhà đầu tư… những đối tượng này được gọi chung là các tác nhân quan trọng (stakeholder).

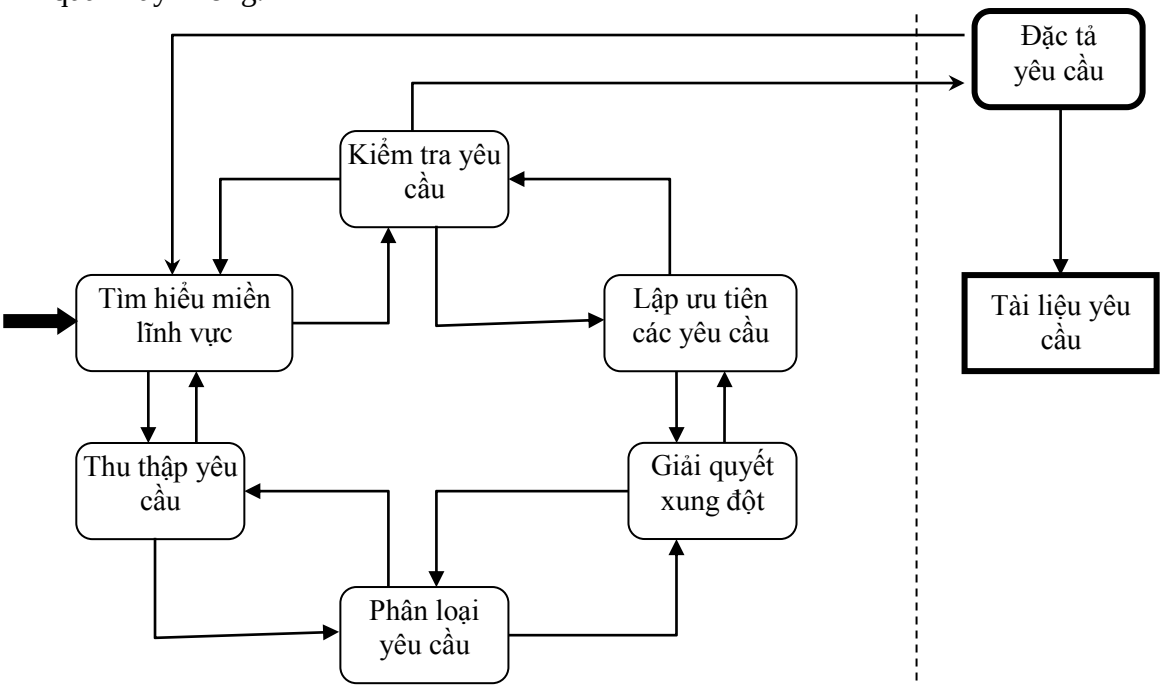
Giai đoạn này cũng phải đối mặt với không ít khó khăn vì người liên quan (người sử dụng) thường không thực sự biết họ cần gì từ hệ thống, do đó họ diễn đạt các yêu cầu của mình

theo cách nói riêng, có thể có nhiều thuật ngữ chuyên ngành; nhóm phát triển cần hiểu rõ những yêu cầu này và chuyển chúng sang một hình thức thể hiện thống nhất. Có trường hợp các đối tượng người dùng khác nhau có yêu cầu khác nhau, lúc này các kỹ sư phần mềm phải nhận biết được điểm chung và điểm khác biệt giữa chúng. Quá trình phân tích diễn ra trong một bối cảnh cụ thể của tổ chức, đôi khi nó bị ảnh hưởng bởi các yếu tố chính trị và kinh tế. Hơn nữa, môi trường kinh doanh luôn biến động nên các yêu cầu mới có thể xuất hiện từ những người liên quan mà ban đầu ta không xác định hết.

Ngoài ra, các yêu cầu phần mềm thường không có định nghĩa chính xác, không có công thức cho trước. Thông qua việc xử lý thông tin đã thu thập, hiểu biết của người phát triển không ngừng được bổ sung và nâng cao, yêu cầu sẽ được thay đổi và hoàn thiện. Điều này cũng dẫn tới sự thiếu ổn định của yêu cầu hệ thống. Vì vậy, việc mong chờ xác định rõ ràng rồi mới bắt tay vào phát triển hệ thống sẽ là không thực tế.

Hình 4.4 minh họa các hoạt động của tiến trình phân tích yêu cầu. Các hoạt động này thông thường bao gồm:

- *Tìm hiểu miền ứng dụng:* Nhóm phát triển cần phải tìm hiểu các hoạt động nghiệp vụ, nội dung và các quy tắc nghiệp vụ cần tuân thủ, sản phẩm và các lĩnh vực liên quan.
- *Thu thập yêu cầu:* Tiếp xúc với những người liên quan tới hệ thống để phát hiện ra những yêu cầu của họ đối với hệ thống.
- *Phân loại yêu cầu:* Đây là hoạt động sắp xếp và phân loại các yêu cầu.
- *Giải quyết xung đột:* hoạt động này liên quan tới việc phát hiện xung đột giữa các yêu cầu và tìm cách giải quyết chúng.
- *Sắp ưu tiên:* Các hệ thống nói chung đều có các yêu cầu quan trọng và yêu cầu ít quan trọng hơn. Bằng cách tiếp xúc và tìm hiểu những người liên quan, nhà phân tích sẽ nhận ra tầm quan trọng của các yêu cầu và sắp xếp chúng một cách hợp lý.
- *Kiểm tra yêu cầu:* Các yêu cầu cần được kiểm tra theo từng nghiệp vụ để xem chúng có đầy đủ và nhất quán không, hơn nữa nó có phù hợp với nhu cầu của những người liên quan hay không.



Hình 4.5. Tiến trình phát hiện và phân tích yêu cầu

Tiến trình của các hoạt động trên là tiến trình lặp, các thông tin phản hồi sẽ được luân chuyển từ hoạt động này sang hoạt động khác nhằm hoàn thiện và bổ xung các thông tin đã thu thập được.

4.2.3. Các phương pháp phát hiện yêu cầu

Phân tích yêu cầu là một hoạt động không đơn giản, nhóm phân tích phải đối mặt với rất nhiều khó khăn trong giai đoạn này. Dưới đây là một số phương pháp giúp thu thập và phân tích yêu cầu. Mỗi phương pháp đều có điểm mạnh và điểm yếu riêng, nó có thể thích hợp để phát hiện yêu cầu của dự án này nhưng lại không thích hợp trong dự án khác. Trên thực tế, người ta có thể kết hợp các phương pháp khác nhau để tìm ra yêu cầu cuối cùng của hệ thống cần phát triển.

a) Phương pháp DataMining

Phương pháp này sử dụng các hệ thống đang tồn tại để đưa ra các yêu cầu. Do đó, điều kiện tiên quyết là nhóm phân tích yêu cầu có thể truy cập vào hệ thống, nghiên cứu tất cả các chức năng của hệ thống và lấy đó làm cơ sở để xác định yêu cầu cho các hệ thống mới.

Ưu điểm chính của phương pháp này là người phân tích gần như làm việc độc lập, có thể chủ động về thời gian và công việc của mình. Sử dụng công cụ cũ để khám phá yêu cầu nên thông tin thu được nói chung phù hợp với nhu cầu của người dùng.

Tuy nhiên, phương pháp này tồn tại một số nhược điểm, ví dụ như người phân tích dễ lặp lại những ưu điểm và cả nhược điểm của hệ thống cũ, không phát hiện được những ràng buộc trong quá trình thực hiện dự án và quan trọng nhất là không khảo sát được những nhu cầu mới phát sinh của khách hàng.

b) Phương pháp quan sát (Observation)

Khi sử dụng phương pháp này, nhóm phân tích cần thâm nhập vào thực tế công việc của người sử dụng. Do đó họ cần phải vào được nơi làm việc của người sử dụng và đưa ra các câu hỏi điều tra.

Ưu điểm lớn nhất của phương pháp này là khám phá được nhu cầu của người sử dụng và các ràng buộc ngầm bên trong. Đã áp dụng thành công trong các dự án về điều khiển giao thông hàng không, tàu điện ngầm và những công việc trong văn phòng.

Tuy nhiên, khi sử dụng phương pháp này, nhóm phân tích có thể thu thập cả những thông tin không quan trọng, hơn nữa, những thông tin mà họ thu thập được rất khó xác định giá trị nên việc phân loại và sắp xếp thứ tự ưu tiên không đơn giản. Ngoài ra, nhóm phân tích cũng không nắm được những yêu cầu mang tính tổ chức hoặc lĩnh vực phát triển phần mềm nên khó đánh giá và lường trước những rủi ro, những ràng buộc trong tiến trình xây dựng phần mềm.

c) Phương pháp phân tích nhiệm vụ

Khi khảo sát yêu cầu người dùng, nhóm phân tích phát cho người sử dụng các đầu phiếu, khi người dùng thực hiện một công việc nào đó, họ sẽ ghi lại thông tin trên đầu phiếu. Phương pháp này chỉ phù hợp khi cần xác định những nhiệm vụ lâu dài.

Ưu điểm của phương pháp này là nhóm phân tích có thể thâm nhập vào nơi làm việc của khách hàng và đưa ra được những thông tin chi tiết đối với từng nhiệm vụ đặc biệt. Tuy nhiên, thu thập thông tin kiểu này sẽ có những công việc rất khó hiểu, hơn nữa quá trình phân tích kéo dài và đôi khi quá chi tiết.

d) Phương pháp phỏng vấn phi cấu trúc

Nhóm phân tích đưa ra các câu hỏi với các tác nhân quan trọng mà không cần chuẩn bị trước. Khi sử dụng phương pháp này, nhóm phân tích không cần mất thời gian chuẩn bị mà lại cho phép xác định được các yêu cầu quan trọng của các tác nhân chính.

Tuy nhiên, phương pháp này có thể mất nhiều thời gian vào những câu hỏi không quan trọng, thông tin thu được đôi khi khó phân tích và phân lớp. Hơn nữa, nó đòi hỏi sự khéo léo và kinh nghiệm của người làm phỏng vấn. Phương pháp này cũng không phát hiện và giải quyết được xung đột khi quyền lợi, yêu cầu của các đối tượng người dùng khác nhau mâu thuẫn nhau.

e) Phương pháp phỏng vấn cấu trúc

Khi sử dụng phương pháp phỏng vấn cấu trúc, nhóm phân tích sẽ đưa cho các tác nhân một danh sách những câu hỏi đã được chuẩn bị trước. Những câu hỏi quan trọng phải được xác định trước khi phỏng vấn.

Phương pháp này có ưu điểm là tất cả các tác nhân đều được hỏi cùng một hệ thống câu hỏi và người phỏng vấn là người điều khiển cuộc đối thoại. Khi các tác nhân được hỏi cùng một hệ thống câu hỏi thì dễ phát hiện ra những xung đột trong yêu cầu của các đối tượng người dùng khác nhau. Tuy nhiên, hệ thống câu hỏi đưa ra có thể thiếu những điểm quan trọng và không giải quyết được những vấn đề xung đột.

Trong cả hai phương pháp phỏng vấn trên, người phỏng vấn phải có những kỹ năng cần thiết sau:

- Không có những ý kiến bảo thủ về những yêu cầu.

- Có khả năng đưa ra những ý kiến tranh luận trước một câu hỏi, một giả thiết.

- Khả năng lắng nghe.

- Khả năng tạo ra một cuộc đối thoại:

  - Đưa ra những câu hỏi để bắt đầu cuộc tranh luận.
  - Nói về phần mềm mẫu (Prototype) sẽ làm cùng nhau.
  - Đưa ra một ngữ cảnh hẹp để thảo luận.
  - Sử dụng các bảng câu hỏi.

f) Phương pháp Brainstorming

Đặc điểm chính của phương pháp này là cần phải tập hợp những tác nhân quan trọng để thu thập càng nhiều ý kiến càng tốt, tập trung vào những ý tưởng sáng tạo, tránh những ý kiến đánh giá. Nhóm các tác nhân phải thực sự nghiêm túc và có khả năng làm việc theo nhóm.

Ưu điểm chính của phương pháp brainstorming là cho phép đề cập đến những câu hỏi không bị chi phối bởi những mối quan hệ và người phỏng vấn luôn giữ quyền kiểm soát. Tuy

nhiên, hiệu quả của phương pháp lại phụ thuộc nhiều vào tính năng động của nhóm và những thông tin thu được thường không có tính hệ thống. Vì thế, việc sắp xếp và phân loại yêu cầu sẽ gặp khó khăn.

g) Phương pháp Rapid application development workshop

Khi muốn thu thập yêu cầu người dùng, nhóm phát triển cần tổ chức những cuộc hội thảo từ 8-20 người, là những người có khả năng ra quyết định dưới sự dẫn dắt của một người có khả năng đưa ra những quan điểm độc lập và khách quan. Để sử dụng phương pháp này thì nhóm phát triển cần mất thời gian để chuẩn bị cho cuộc hội thảo (công việc khá nặng, mất khoảng 3 tuần), sau đó tiến hành tập hợp nhóm.

Ưu điểm của phương pháp này là nhóm sẽ tập hợp được những tác nhân quan trọng, tránh được xung đột giữa các yêu cầu do có thể tìm được sự thống nhất giữa các tác nhân ngay trong cuộc hội thảo. Những yêu cầu thu thập được thường đảm bảo chất lượng và hiệu quả. Tuy nhiên, khả năng thành công của phương pháp này phụ thuộc vào khả năng tập hợp nhóm và tài năng của người chủ trì. Hơn nữa nó lại đòi hỏi phải tập hợp những người quan trọng trong nhiều ngày, do đó chỉ có hiệu quả đối với những dự án quan trọng nhưng không quá lớn.

h) Phương pháp Rapid rototyping

Phương pháp rapid prototyping thực chất là xây dựng các phần mềm mẫu cho người dùng dùng thử. Do đó những tác nhân chính phải tham gia vào việc sử dụng các phần mềm mẫu của hệ thống cần phát triển. Bên cạnh đó, dự án cũng cần phải có môi trường để phát triển các phiên bản mẫu.

Phương pháp này rất hiệu quả đối với việc phát triển những phần mềm gặp nhiều khó khăn trong việc tương tác với người sử dụng, những phần mềm có giao diện biến đổi. Tuy nhiên, khi sử dụng phương pháp này sẽ rất khó đánh giá kinh phí của dự án. Bên cạnh đó, nhóm phát triển sau này thường có xu hướng xây dựng phần mềm theo phiên bản mẫu, dễ lặp lại cả những điểm yếu kỹ thuật của phiên bản mẫu.

4.2.4. Các kỹ thuật phân tích yêu cầu

a) Tiếp cận yêu cầu định hướng cách nhìn (viewpoint)

Đối với các hệ thống phần mềm vừa và lớn, thường có nhiều đối tượng sử dụng khác nhau, mỗi đối tượng lại có một cách nhìn khác nhau đối với hệ thống. Cách tiếp cận này ghi nhận những cách nhìn khác nhau của những người có liên quan để sử dụng vào tiến trình phát hiện và tổ chức các yêu cầu. Mỗi cách nhìn có thể xem xét các góc độ sau:

- *Từ nguồn hay từ đích của dữ liệu:* Cách nhìn ở đây tập trung vào điểm đại diện cho việc tạo ra hay sử dụng dữ liệu. Việc phân tích chủ yếu hướng tới việc xác định xem dữ liệu gì được tạo ra, được sử dụng và xử lý như thế nào. Cách tiệp cận này là điển hình của phương pháp phân tích vào/ra (Input – Output analysis).
- *Từ khung làm việc trình diễn:* Cách nhìn này dựa vào các công cụ để mô tả mô hình hệ thống (mô hình quan hệ thực thể, mô hình trạng thái…) mỗi cách tiếp cận sẽ phát hiện ra những đối tượng khác nhau của hệ thống, từ đó sẽ phân tích và tìm ra mối liên hệ giữa chúng.

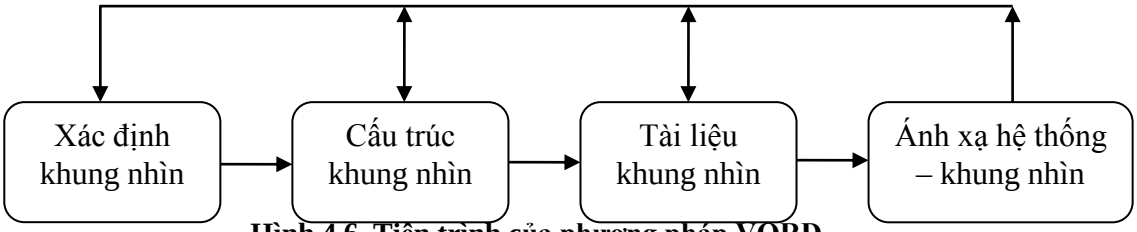
- *Từ sự tiếp nhận dịch vụ:* Cách nhìn này tập trung vào những dịch vụ mà hệ thống sẽ cung cấp và tiếp nhận. Dựa trên cách nhìn này, người ta xác định những dữ liệu cho các dịch vụ và các tín hiệu kiểm tra. Quá trình phân tích bao gồm: kiểm tra các dịch vụ nhận được của các đối tượng sử dụng khác nhau, thu thập và giải quyết các xung đột.

b) Kỹ thuật xác định yêu cầu hướng cách nhìn VORD (Viewpoint Oriented Requirement Definition)

Phương pháp này được đề xuất bởi Ian Sommeville và Kotonua, các giai đoạn của VORD bao gồm:

- *Xác định khung nhìn:* Tìm kiếm các khung nhìn để thu nhận các dịch vụ hệ thống và xác định các dịch vụ cung cấp cho từng khung nhìn.
- *Cấu trúc khung nhìn:* Nhóm các khung nhìn liên quan trong một hệ thống phân cấp. Các dịch vụ chung được cung cấp ở mức cao trong hệ thống này và được các khung nhìn ở mức thấp hơn kế thừa.
- *Làm tài liệu khung nhìn:* làm mịn các mô tả khung nhìn và dịch vụ được xác định.
- *Ánh xạ hệ thống – khung nhìn:* xác định các đối tượng nếu dùng phương pháp phân tích hướng đối tượng khi sử dụng các thông tin dịch vụ được giới hạn trong một khung nhìn.

Hình vẽ 4.5 chỉ ra các hoạt động trong tiến trình phân tích yêu cầu theo phương pháp VORD.



Hình 4.6. Tiến trình của phương pháp VORD

c) Kỹ thuật phân tích yêu cầu dựa trên mô hình

Đây là một kỹ thuật phổ biến để phân tích yêu cầu. Các loại mô hình khác nhau được sử dụng và thường đi theo 2 hướng tiếp cận khác nhau:

- *Tiếp cận hướng chức năng:* là các phương pháp phân tích hướng cấu trúc dựa trên luồng dữ liệu.
- *Tiếp cận hướng đối tượng:* là các phương pháp phân tích hướng cấu trúc dựa trên các thực thể.

Để phát hiện ra các yêu cầu, cả hai cách tiếp cận trên đều hướng tới nghiệp vụ của hệ thống, từ đó xây dựng nên các mô hình nghiệp vụ. Mô hình nghiệp vụ này có thể bao gồm:

- ***Mô hình ngữ cảnh:*** mô tả hệ thống được xét trong môi trường hoạt động của nó, đặc biệt là các yếu tố môi trường tương tác với hệ thống, gọi là các tác nhân. Mô hình này cũng cho phép xác định rõ phạm vi của hệ thống cần nghiên cứu. Tùy theo cách tiếp cận mà khái niệm về tác nhân và mối quan hệ tương tác của hệ thống được biểu diễn khác nhau.
- ***Các mô hình cấu trúc chức năng:*** mô tả cấu trúc các chức năng bên trong hệ thống. Các mô hình này thường có nhiều mức, mỗi mức có thể mô tả một chức năng tổng hợp hoặc

chi tiết khác nhau. Ở mức thấp nhất, nội dung các chức năng đủ dễ hiểu và có thể mô tả chi tiết. Tùy theo cách tiếp cận mà ta có các dạng biểu diễn khác nhau.

- ***Mô tả chi tiết các chức năng***: Mô tả chi tiết chức năng là cần thiết và có thể thực hiện được với các chức năng ở mức thấp nhất (chức năng cơ sở). Tuy nhiên, cũng có thể mô tả chức năng ở mức cao (theo phương pháp hướng đối tượng) để dễ hiểu và dễ lần vết trong quá trình tiếp cận dần. Tùy theo cách tiếp cận, nội dung và cách mô tả chức năng cũng khác nhau.
- ***Mô tả các đối tượng dữ liệu:*** cần cho việc lưu giữ sau này. Do bản chất của mỗi cách tiếp cận mà sự mô tả ở đây là khác nhau. Tiếp cận theo hướng cấu trúc, mô tả dữ liệu bao gồm tất cả hồ sơ được sử dụng trong hoạt động nghiệp vụ và các bản mẫu thực của chúng. Trong khi đó, theo định hướng đối tượng, mô tả lại bao gồm các đối tượng và khái niệm của thế giới thực.
- ***Mô tả mối liên kết giữa chức năng và dữ liệu:*** Mô tả này chỉ cần thiết đối với cách tiếp cận hướng cấu trúc.
- ***Từ điển dữ liệu:*** liên quan đến các khái niệm và dữ liệu sử dụng. Từ điển này là cần thiết để có sự hiểu biết chung và thống nhất về hệ thống thực được mô tả cũng như các đặc trưng của các dữ liệu cần lưu giữ.

d) Các cách biểu diễn của mô hình phân tích

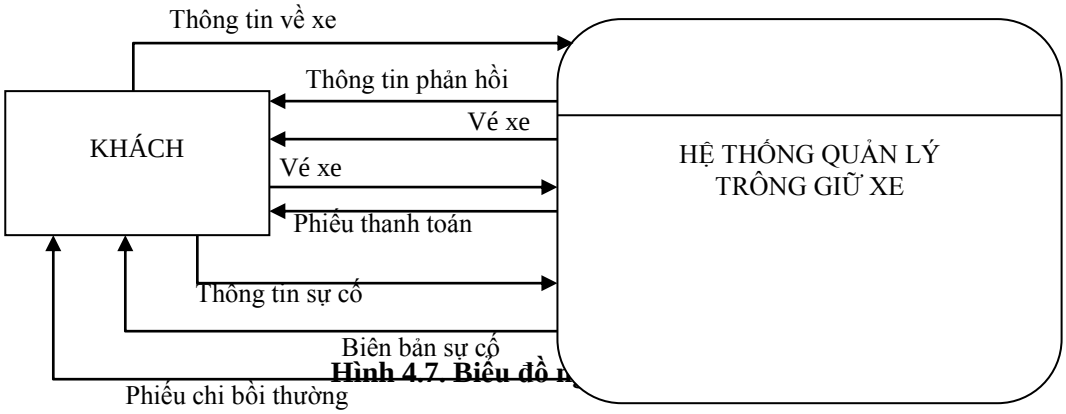
Bảng 4.2. Biểu diễn mô hình nghiệp vụ theo 2 cách tiếp cận

Các thành phần của mô hình phân tích	Thể hiện của mô hình theo định hướng cấu trúc	Thể hiện của mô hình theo định hướng đối tượng
1. Mô hình ngữ cảnh	Gồm 3 thành phần: - 1 tiến trình (hình chữ nhật góc tròn) duy nhất mô tả hệ thống - Các tác nhân (hình chữ nhật), mô tả người, tổ chức, hệ thống khác không thuộc hệ thống - Các luồng dữ liệu (hình mũi tên) liên kết giữa tác nhân và hệ thống	Gồm 3 thành phần: - Các ca sử dụng (hình chữ nhật) mức cao mô tả hệ thống - Các tác nhân (hình người) là người, các hệ thống khác có tương tác với hệ thống - Các liên kết tương tác giữa các tác nhân với các trường hợp sử dụng và giữa các trường hợp sử dụng với nhau
2. Mô hình cấu trúc chức năng	- Gồm một hoặc một số biểu đồ phân cấp hình cây, mỗi nút lá là một chức năng - Mỗi chức năng ở mức dưới nhận được bằng cách phân rã chức năng trên của nó. Chức năng mức thấp nhất là chức năng cơ sở	- Là biểu đồ trường hợp sử dụng (use case) tương ứng với các mức khác nhau - Biểu đồ của mức dưới nhận được từ các ca phân rã ở mức trên - Ca sử dụng mức thấp có thể mô tả sự tương tác
3. Mô tả chi tiết chức năng	- Chỉ mô tả chức năng cơ sở - Mô tả gồm tên chức năng, các dữ liệu vào, dữ liệu ra, nguồn, đích, các quy tắc nghiệp vụ để thực hiện chức năng	- Mô tả mỗi ca sử dụng bằng các luồng tương tác chính/ngoại lệ giữa tác nhân và hệ thống - Các ràng buộc lên luồng tương tác
4. Mô tả các đối tượng dữ liệu	- Danh sách các hồ sơ dữ liệu sử dụng - Các mẫu hồ sơ, từ điển dữ liệu	Mô hình miền lĩnh vực với các đối tượng là các thực thể hay khái niệm được liên kết với nhau.
5. Mô tả liên kết giữa chức năng và dữ liệu	- Ma trận thực thể - chức năng: mỗi cột ứng với một hồ sơ, mỗi dòng ứng với một chức năng - Mỗi ô ghi tương tác giữa chức năng và hồ sơ	Không cần
6. Từ điển giải thích	Từ điển giải thích khái niệm	Từ điển giải thích khái niệm

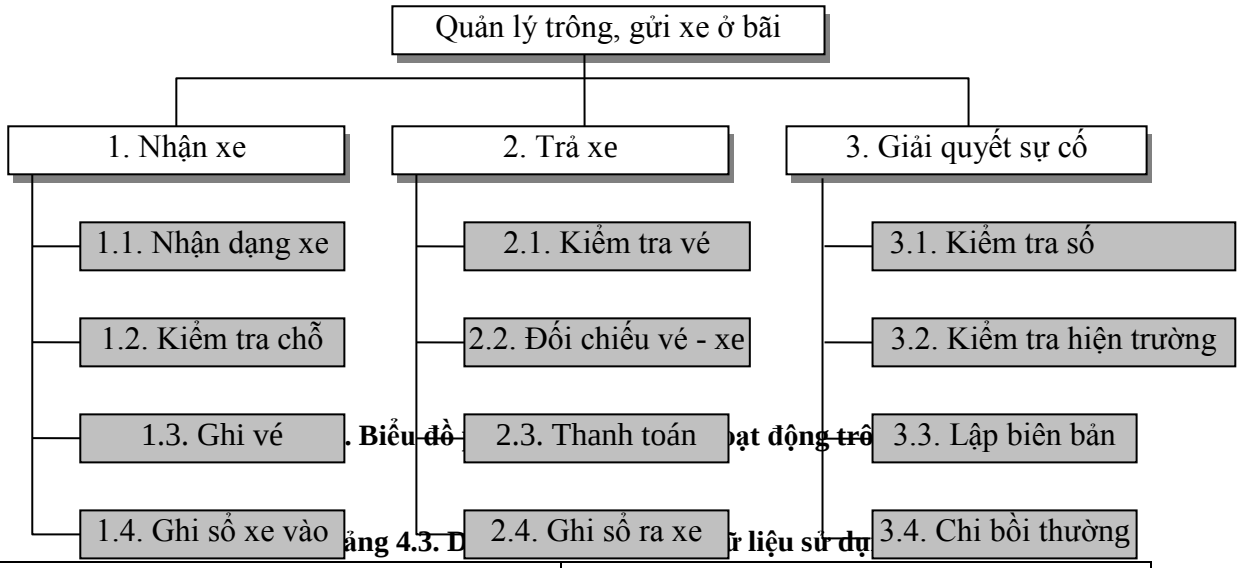
4.2.5. Ví dụ phân tích phát hiện yêu cầu

a) Ví dụ phân tích hướng cấu trúc

Ví dụ: Hãy phân tích yêu cầu cho mô hình nghiệp vụ của hoạt động trông giữ xe ở một bãi gửi xe.



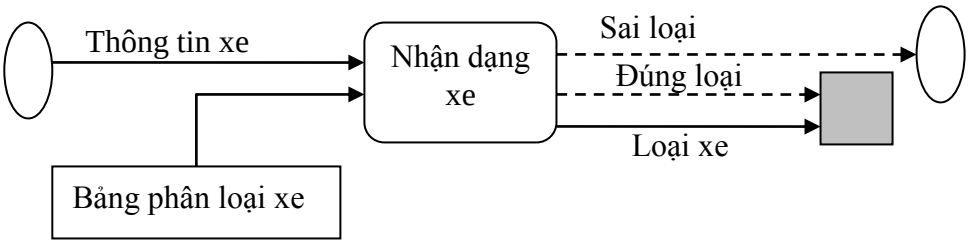
Hình 4.7. Biểu đồ n



. Biểu đồ ạt động trô

ảng 4.3. D ữ liệu sử dụ

a. Bảng giá (phân loại xe)	e. Phiếu thanh toán
b. Vé xe	f. Biên bản sự cố
c. Sổ vào xe	g. Phiếu chi bồi thường
d. Sổ ra xe	



Quy tắc nghiệp vụ: Xe đúng loại là xe thuộc loại đã ghi trong bảng phân loại

Hình 4.9. Mô tả yêu cầu một chức năng sơ cấp

Các thực thể						
a. Bảng giá (phân loại xe)						
b. Vé xe						
c. Sổ vào xe						
d. Sổ ra xe						
e. Phiếu thanh toán						
f. Biên bản sự cố						
g. Phiếu chi bồi thường						
Các chức năng nghiệp vụ	**a**	**b**	**c**	**d**	**e**	**f**
1. Nhận xe	R	C	U	R		
2. Trả xe		R		U	C	
3. Giải quyết sự cố			R	R	C	C

(Chú thích: **C** – Create; **R** – Read; **U** – Update)

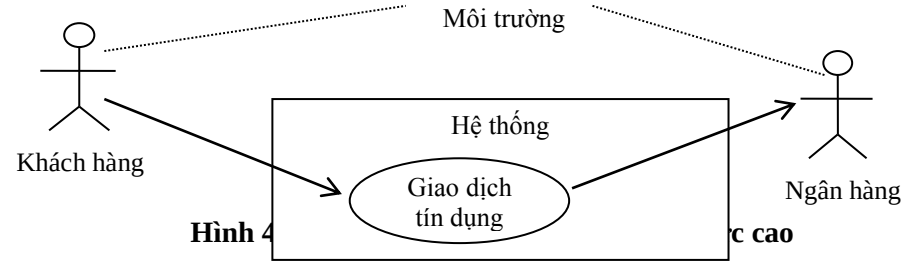
Ma trận này mô tả: một chức năng ở dòng (nhận xe) có thể tạo (C),

Cập nhật (U) hay đọc (R) một thực thể dữ liệu ở cột (b: Vé xe)

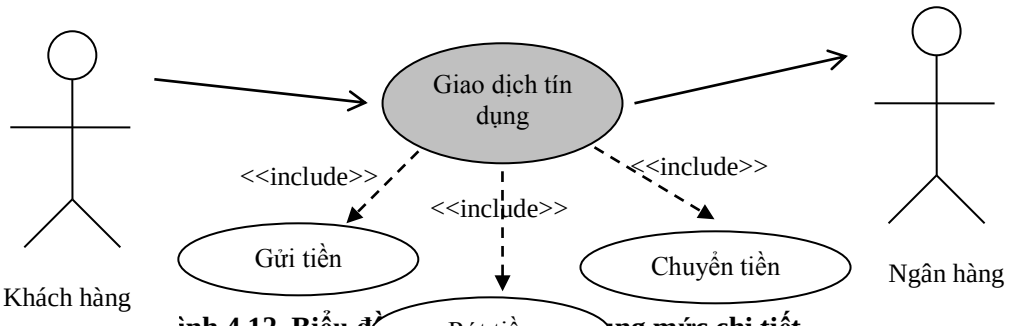
Hình 4.10. Ma trận thực thể - chức năng

b) Phân tích yêu cầu theo định hướng đối tượng

Ví dụ dưới đây minh họa về việc phân tích yêu cầu cho hoạt động của một máy ATM của ngân hàng cung cấp dịch vụ tín dụng cho khách hàng theo định hướng đối tượng.



Hình 4 c cao



ình 4.12. Biểu đ ng mức chi tiết

Bảng 4.4. Mô tả chi tiết chức năng rút tiền

Tên chức năng	**Rút tiền**

Các tác nhân liên quan	Khách hàng
Hành động tác nhân	Phản ứng của hệ thống
1. Đưa thẻ vào hệ thống	2. Hiện cửa sổ yêu cầu nhập định danh
3. Nhập định danh	4. Hiện các chức năng giao dịch
5. Chọn chức năng rút tiền	6. Hiện cửa sổ tài khoản, số tiền rút
7. Nhập tài khoản và số tiền rút	8. Trả số tiền cho khách rút, in hóa đơn
9. Nhấn nút kết thúc	10. Trả lại thẻ, đóng hệ thống

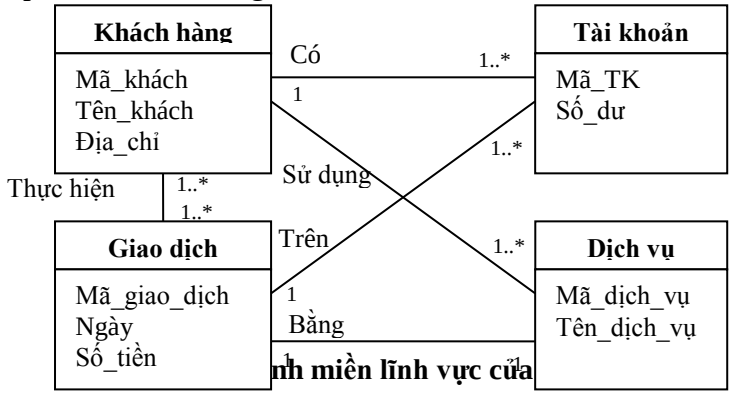
Ngoại lệ:

Bước 2: Nếu định danh sai, chức năng kết thúc.

Bước 4: Nếu tài khoản sai, có thể hệ thống yêu cầu nhập lại.

Bước 8: Nếu số tiền rút quá số dư tài khoản thì hệ thống phải có quy tắc nghiệp vụ xử lý.

Trừ những trường hợp ngoại lệ, các phương pháp cấu trúc thu thập rất nhiều thông tin và tạo ra một khối lượng lớn tài liệu. Vấn đề quản lý thông tin của phương pháp này là một vấn đề quan trọng trong sự phát triển các công cụ CASE.



4.3. ĐẶC TẢ YÊU CẦU PHẦN MỀM

4.3.1. Khái niệm

Tài liệu xác định yêu cầu là một tài liệu 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 đặc 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.

4.3.2. Các phương pháp đặc tả

Có hai phương pháp đặc tả:

- *Đặc tả phi hình thức:* là đặc tả sử dụng ngôn ngữ tự nhiên. Tuy nó không được chặt chẽ bằng đặc tả hình thức nhưng được nhiều người biết và có thể dùng để trao đổi với nhau để làm chính xác hóa các điểm chưa rõ, chưa thống nhất giữa các bên phát triển hệ thống.

- *Đặc tả hình thức:* là đặc tả mà ở đó các từ ngữ, cú pháp, ngữ nghĩa được định nghĩa hình thức dựa vào toán học. Đặc tả hình thức có thể coi là một phần của hoạt động đặc tả phần mềm. Các đặc tả yêu cầu được phân tích chi tiết. Các mô tả trừu tượng của các chức năng chương trình có thể được tạo ra để làm rõ yêu cầu.

a) Đặc tả hình thức

Đặc tả hình thức là một đặc tả được trình bày trên một ngôn ngữ bao gồm: từ vựng, cú pháp và ngữ nghĩa được định nghĩa. Định nghĩa ngữ nghĩa đảm bảo ngôn ngữ đặc tả không phải là ngôn ngữ tự nhiên mà dựa trên toán học. Các chức năng nhận các đầu vào và trả lại các kết quả. Các chức năng có thể định ra các điều kiện tiền tố và hậu tố. Điều kiện tiền tố là điều kiện cần thỏa mãn để có dữ liệu vào, điều kiện hậu tố là điều kiện cần thỏa mãn sau khi có kết quả.

Có hai hướng tiếp cận đặc tả hình thức để phát triển các hệ thống tương đối phức tạp.

- Tiếp cận đại số, hệ thống được mô tả dưới dạng các toán tử và các quan hệ.

- Tiếp cận mô hình, mô hình hệ thống được cấu trúc sử dụng các thực thể toán học như là các tập hợp và các thứ tự.

Sử dụng đặc tả hình thức, ta có các thuận lợi:

- Cho phép chúng ta thấy và hiểu được bản chất bên trong của các yêu cầu, đây là cách tốt nhất để làm giảm các lỗi, các thiếu sót có thể xảy ra và giúp cho công việc thiết kế được thuận lợi.

- Do chúng ta sử dụng toán học cho việc đặc tả nên có thể dựa vào các công cụ toán học khi phân tích và điều này làm tăng thêm tính chắc chắn và tính đầy đủ của hệ thống.

- Đặc tả hình thức, bản thân nó cho chúng ta một cách thức cho việc kiểm tra hệ thống sau này.

Tuy vậy, đặc tả hình thức cũng bộc lộ một vài khó khăn:

- Quản lý phần mềm có tính bảo thủ cố hữu của nó, không sẵn sàng chấp nhận các kỹ thuật mới.

- Chi phí cho việc đặc tả hình thức thường cao hơn so với các đặc tả khác (tuy phần cài đặt sẽ thấp hơn), nên khó để chứng minh rằng chi phí tương đối cao cho đặc tả sẽ làm giảm tổng chi phí dự án.

- Phần lớn, những người đặc tả hệ thống không được đào tạo một cách chính quy về việc sử dụng đặc tả hình thức cho đặc tả hệ thống mà dựa trên thói quen của họ.

- Thông thường, nhiều thành phần của hệ thống là khó cho việc đặc tả bằng ngôn ngữ hình thức. Thêm vào đó là khách hàng không thể hiểu được nó.

- Khách hàng không thích các đặc tả toán học.

b) Đặc tả phi hình thức

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:

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

- 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.

- 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 nhiều thời gian và công sức.

Một cách tiếp cận khác là bên cạnh các đặc tả hình thức người ta viết các chú giải bằng ngôn ngữ tự nhiên để giúp khách hàng dễ hiểu các yêu cầu phần mềm hơn.

4.3.3. Cấu trúc tài liệu đặc tả

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 SRS thông dụng (theo chuẩn của IEEE 830-1984). Cấu trúc của tài liệu đặc tả như sau:

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 XZY".

1.2. Phạm vi

Nêu phạm 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, phụ 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

- Giới thiệu
- Dữ liệu vào
- Xử lý
- 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, khi bị tấn công, khi bị quá tải… 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.

Phân tích và định rõ yêu cầu là bước kỹ thuật đầu tiên trong tiến trình công 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 sản xuất 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.

CÂU HỎI ÔN TẬP

1. Yêu cầu phần mềm là gì? Có những loại yêu cầu phần mềm gì? Cho ví dụ minh họa đối với hệ thống phần mềm nhúng cho máy rút tiền tự động ATM của các ngân hàng.
2. Có những loại tài liệu yêu cầu phần mềm nào? Đối tượng chính của những loại tài liệu yêu cầu này là ai?
3. Trình bày các bước của tiến trình phát hiện và phân tích yêu cầu. Trình bày các kỹ thuật khác nhau để nhận biết và phân tích yêu cầu.
4. Cấu trúc cơ bản của tài liệu yêu cầu gồm những mục gì?
5. Đặc tả yêu cầu phần mềm là gì? Có những hình thức đặc tả nào? Khi nào cần áp dụng những hình thức đặc tả này?
6. Việc làm bản mẫu có vai trò gì trong đặc tả phần mềm?

Chương 5: UML – XÂY DỰNG VÀ THIẾT KẾ CÁC MÔ HÌNH HỆ THỐNG

Yêu cầu người dùng thường được viết bằng ngôn ngữ tự nhiên để những người không phải là chuyên gia CNTT có thể hiểu được. Tuy nhiên, yêu cầu hệ thống chi tiết hơn và có thể được diễn tả theo hướng kỹ thuật nhiều hơn. Trong đó, cách được sử dụng nhiều nhất là đặc tả hệ thống thành một tập các mô hình hệ thống. ***Những mô hình này là cách biểu diễn đồ họa dùng để mô tả các tiến trình nghiệp vụ, vấn đề cần giải quyết và hệ thống được phát triển.*** Thông thường biểu diễn bằng đồ họa dễ hiểu hơn diễn đạt chi tiết bằng ngôn ngữ tự nhiên. Nó cũng là cầu nối quan trọng giữa nhóm phân tích và nhóm thiết kế.

Ta có thể sử dụng các mô hình này trong các tiến trình phân tích để hiểu rõ hơn hệ thống đang tồn tại cần được thay thế, nâng cấp hay tạo mới hoàn toàn. Bên cạnh đó ta cũng cần phải sử dụng các loại mô hình khác nhau để diễn tả những khía cạnh khác nhau của hệ thống.

Chương này giới thiệu về ngôn ngữ UML và tìm hiểu một số loại sơ đồ UML để ứng dụng trong mô hình hóa hệ thống.

5.1. GIỚI THIỆU VỀ UML

5.1.1. Mô hình hóa hệ thống phần mềm

Như đã trình bày ở phần trước, mục tiêu của giai đoạn phân tích hệ thống là sản xuất ra một mô hình tổng thể của hệ thống cần xây dựng. Mô hình này cần phải được trình bày theo hướng nhìn của khách hàng và người sử dụng để họ có thể hiểu được. Mô hình này cũng có thể được sử dụng để xác định các yêu cầu của người dùng đối với hệ thống và qua đó giúp chúng ta đánh giá tính khả thi của dự án.

Mô hình thường được mô tả bằng ngôn ngữ trực quan, điều đó có nghĩa là đa phần các thông tin được thể hiện bằng các ký hiệu đồ họa và các mối liên kết giữa chúng, chỉ khi cần thiết một số thông tin mới được biểu diễn ở dạng văn bản. Việc biểu diễn mô hình phải thoả mãn các yếu tố sau:

- *Chính xác*: Mô tả đúng hệ thống cần xây dựng.

- *Đồng nhất*: Các ngữ cảnh khác nhau không được mâu thuẫn với nhau.

- *Có thể hiểu được*: Cho cả người xây dựng lẫn người sử dụng.

- *Dễ thay đổi, dễ dàng liên lạc với các mô hình khác.*

Có thể nói thêm rằng,mô hình là một sự đơn giản hoá hiện thực. Mô hình được xây dựng để chúng ta dễ dàng hiểu và hiểu tốt hơn hệ thống cần xây dựng. Tạo mô hình sẽ giúp cho chúng ta hiểu thấu đáo một hệ thống phức tạp trong sự toàn thể của nó. Nói tóm lại, mô hình hóa một hệ thống nhằm mục đích:

- Hình dung một hệ thống theo thực tế hay theo mong muốn của chúng ta.

- Chỉ rõ cấu trúc hoặc cách thức ứng xử của hệ thống.

- Tạo một khuôn mẫu hướng dẫn nhà phát triển trong suốt quá trình xây dựng hệ thống.

- Ghi lại các quyết định của nhà phát triển để sử dụng sau này.

5.1.2. Lịch sử hình thành và phát triển

a) Bối cảnh ra đời của UML

Đầu những năm 1980, lĩnh vực phát triển phần mềm chỉ có duy nhất một ngôn ngữ hướng đối tượng là Simula. Sang nửa sau của thập kỷ 1980, các ngôn ngữ hướng đối tượng như Smalltalk và C++ xuất hiện. Cùng với chúng, nảy sinh nhu cầu mô hình hoá các hệ thống phần mềm theo phương pháp hướng đối tượng. Một ngôn ngữ mô hình hoá xuất hiện những năm đầu của thập kỷ 90 được nhiều người dùng là:

- Grady Booch's Booch Modeling Methodology.

- James Rambaugh's Object Modeling Technique – OMT.

- Ivar Jacobson's OOSE Methodology.

- Hewlett- Packard's Fusion.

- Coad and Yordon's OOA and OOD.

Mỗi phương pháp và ngôn ngữ trên đều có hệ thống ký hiệu riêng, cách thức xử lý riêng, công cụ hỗ trợ riêng làm phát sinh các cuộc tranh luận phương pháp nào là tốt nhất. Đây là cuộc tranh luận khó có câu trả lời, bởi tất cả các phương pháp đều có điểm mạnh và điểm yếu riêng. Vì thế, các nhà phát triển phần mềm nhiều kinh nghiệm thường sử dụng phối hợp các điểm mạnh của mỗi phương pháp cho ứng dụng của mình. Trong thực tế, sự khác biệt giữa các phương pháp hầu như không đáng kể. Theo cùng tiến trình thời gian, tất cả những phương pháp trên đã tiệm cận lại và bổ sung lẫn cho nhau. Chính hiện thực này đã được những người tiên phong trong lĩnh vực mô hình hoá hướng đối tượng nhận ra và họ quyết định ngồi lại cùng nhau để tích hợp những điểm mạnh của mỗi phương pháp, cuối cùng đưa ra một mô hình hóa thống nhất.

Trong bối cảnh trên, người ta nhận thấy cần thiết phải cung cấp một phương pháp tiệm cận được chuẩn hoá và thống nhất cho việc mô hình hoá hướng đối tượng. Yêu cầu cụ thể là đưa ra một tập hợp chuẩn hoá các ký hiệu (Notation) và các biểu đồ (Diagram) để nắm bắt các quyết định về mặt thiết kế một cách rõ ràng, rành mạch. Đã có ba công trình tiên phong nhắm tới mục tiêu đó, chúng được thực hiện dưới sự lãnh đạo của James Rumbaugh, Grady Booch và Ivar Jacobson. Từ những cố gắng này, ngôn ngữ mô hình hoá thống thất (Unifield Modeling Language – UML) đã ra đời.

b) UML (Unifield Modeling Language)

Ngôn ngữ mô hình hóa thống nhất (Unifield Modeling Language – UML) là một ngôn ngữ để biểu diễn mô hình theo hướng đối tượng được, xây dựng bởi ba tác giả trên với mục đích:

- Mô hình hoá các hệ thống sử dụng các khái niệm hướng đối tượng.

- Thiết lập một kết nối từ nhận thức của con người đến các sự kiện cần mô hình hoá.

- Giải quyết vấn đề về mức độ thừa kế trong các hệ thống phức tạp, có nhiều ràng buộc khác nhau.

- Tạo một ngôn ngữ mô hình hoá có thể sử dụng được bởi người và máy.

5.1.2. UML và các giai đoạn phát triển hệ thống

- ***Nghiên cứu tính khả thi*** *(Preliminary Investigation)*: sử dụng mô hình trường hợp sử dụng (use cases) thể hiện các yêu cầu của người dùng. Phần miêu tả mô tả từng trường hợp sử dụng để xác định các yêu cầu, phần mô hình thể hiện mối quan hệ và giao tiếp của các tác nhân với hệ thống.

- ***Phân tích*** *(Analysis)*: Mục đích chính của giai đoạn này là trừu tượng hóa và tìm hiểu các cơ cấu có trong phạm vi bài toán. Mô hình lớp trên bình diện trừu tượng hóa các thực thể ngoài đời thực được sử dụng để làm rõ sự tồn tại cũng như mối quan hệ của chúng. Chỉ những lớp nằm trong phạm vi bài toán mới đáng quan tâm.

- ***Thiết kế*** *(Design)*: Kết quả phần phân tích được phát triển thành giải pháp kỹ thuật. Các biểu đồ lớp được mô hình hóa chi tiết để cung cấp hạ tầng kỹ thuật như giao diện, nền tảng CSDL… Kết quả phần thiết kế là các đặc tả chi tiết cho giai đoạn xây dựng phần mềm.

- ***Phát triển*** *(Development)*: Mô hình thiết kế được chuyển thành mã nguồn chương trình. Người lập trình sẽ sử dụng các mô hình UML trong giai đoạn thiết kế để hiểu vấn đề và tạo mã nguồn chương trình.

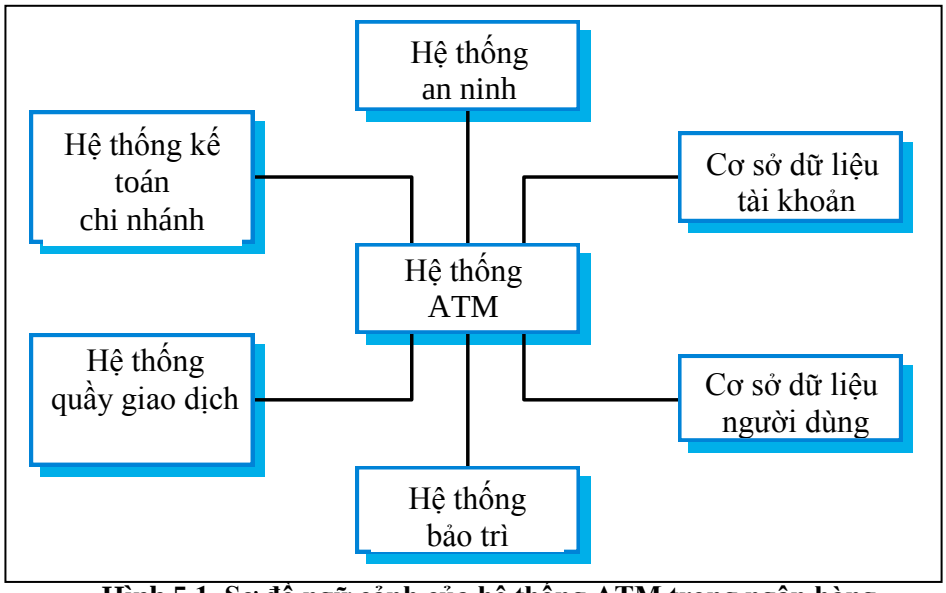
- ***Kiểm thử*** *(Testing)*: Sử dụng các mô hình UML trong các giai đoạn trước làm cơ sở cho việc kiểm thử. Có bốn hình thức kiểm thử hệ thống: kiểm thử đơn vị, kiểm thử tích hợp, kiểm thử hệ thống và kiểm thử chấp nhận.

5.2. MỘT SỐ MÔ HÌNH UML DÙNG TRONG PHÂN TÍCH VÀ THIẾT KẾ

5.2.1. Mô hình ngữ cảnh

Tại giai đoạn đầu của tiến trình phân tích và xác định yêu cầu cần quyết định ranh giới của hệ thống. Khi đó người phân tích phải làm việc với các tác nhân quan trọng của dự án để phân biệt giữa hệ thống và môi trường hệ thống. Quyết định này cần được đưa ra sớm để hạn chế chi phí phát triển và thời gian cần để phân tích hệ thống. Trong một số trường hợp, giới hạn giữa hệ thống và môi trường là rất rõ ràng. Ví dụ: khi một hệ thống tự động thay thế một hệ thống tính toán đang tồn tại, môi trường của hệ thống mới thông thường giống như môi trường của hệ thống cũ. Tuy nhiên, cách phân chia này có thể mềm dẻo hơn, nhóm phát triển quyết định thiết lập giới hạn giữa hệ thống và môi trường của nó trong quá trình phân tích yêu cầu. Một ví dụ khác, nhóm phát triển đang làm đặc tả cho hệ thống Website quản lý và giới thiệu sản phẩm cho một công ty chuyên kinh doanh các mặt hàng thời trang. Khi đặc tả hệ thống, nhóm phải quyết định xem việc đặt hàng từ các hãng sản xuất có nằm trong giới hạn hệ thống hay không. Nếu đúng, hệ thống sau đó phải cho phép nhân viên công ty có giao diện để liên hệ với các nhà cung cấp. Nếu không, công ty phải liên hệ trực tiếp với nhà cung cấp.

Khi những quyết định về giới hạn của hệ thống được thiết lập, một phần của hoạt động phân tích là xác định ngữ cảnh và sự phụ thuộc của hệ thống này vào môi trường của nó. Thông thường, việc sinh ra một mô hình kiến trúc đơn giản là bước đầu tiên trong hoạt động này. Hình 5.1 chỉ ra mô hình kiến trúc minh họa cho cấu trúc của hệ thống ATM. Mô hình kiến trúc ở mức cao thường được diễn tả bằng những sơ đồ khối đơn giản, mỗi hệ thống con là một hình chữ nhật được đặt tên và các đường chỉ ra mối liên kết giữa các hệ thống đó.

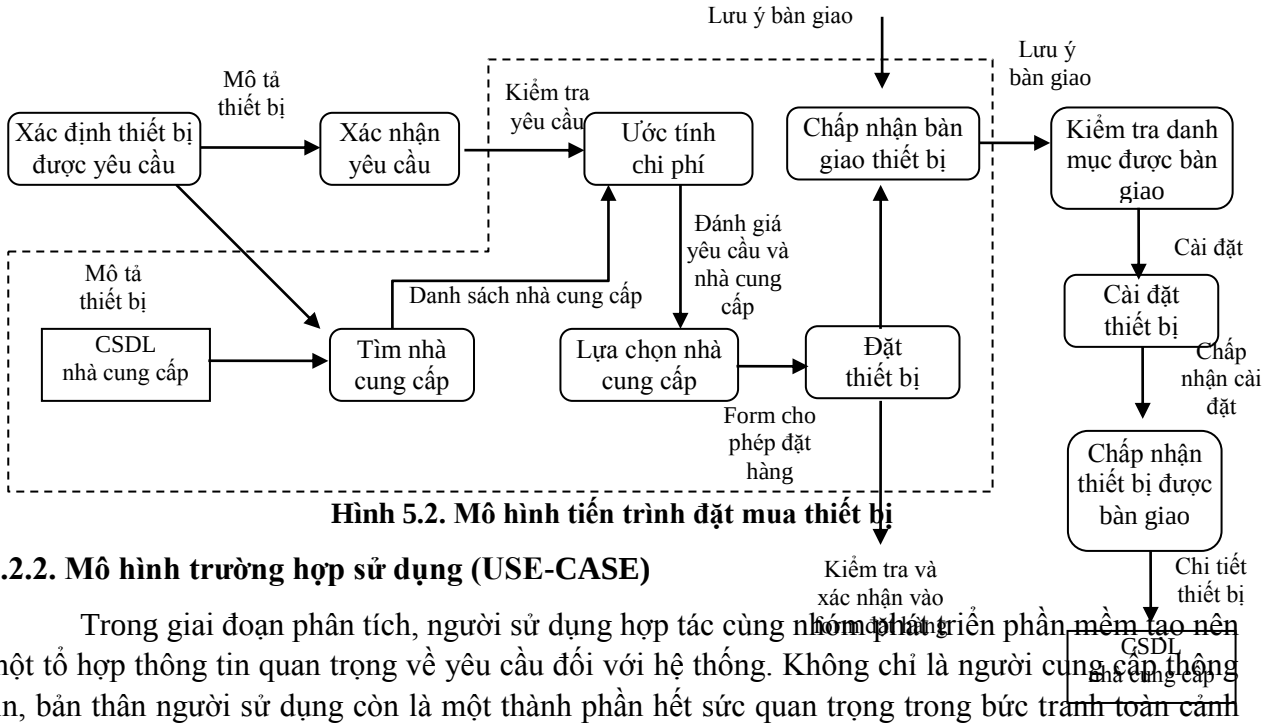


Hình 5.1. Sơ đồ ngữ cảnh của hệ thống ATM trong ngân hàng

Từ hình 5.1, chúng ta có thể thấy rằng mỗi máy ATM được kết nối với một cơ sở dữ liệu tài khoản, một hệ thống kế toán chi nhánh, một hệ thống an ninh và một hệ thống hỗ trợ cho việc bảo trì máy. Hệ thống cũng được kết nối với một hệ thống cơ sở dữ liệu kiểm soát mạng lưới các máy ATM được sử dụng và các hệ thống kế toán của các chi nhánh. Hệ thống kế toán này cung cấp các dịch vụ như sao lưu hoặc in ấn. Tất nhiên, nó không nằm trong hệ thống máy ATM.

Mô hình kiến trúc trên mô tả môi trường của hệ thống, tuy nhiên chúng không chỉ ra mối quan hệ giữa các hệ thống khác trong môi trường với hệ thống đang được đặc tả. Các hệ thống bên ngoài có thể sinh ra dữ liệu cho việc tổng hợp dữ liệu từ hệ thống. Chúng có thể chia sẻ dữ liệu với hệ thống, kết nối trực tiếp qua một mạng hoặc không kết nối. Chúng có thể được đặt ở những vị trí khác nhau. Tất cả những mối quan hệ này đều có ảnh hưởng tới yêu cầu của hệ thống đang được định nghĩa và chúng ta phải tính tới khi tiến hành phát triển hệ thống.

Tuy nhiên, mô hình kiến trúc đơn giản thường không được hỗ trợ bởi các mô hình khác, chẳng hạn như mô hình tiến trình, chỉ ra các hoạt động trong tiến trình được cung cấp bởi hệ thống. Mô hình luồng dữ liệu có thể cũng được sử dụng để chỉ ra dữ liệu được truyền đi giữa các hệ thống trong cùng một môi trường. Hình 5.2 minh họa mô hình tiến trình cho hệ thống đặt mua một thiết bị trong một tổ chức. Mô hình này liên quan tới việc xác định thiết bị được yêu cầu, tìm và lựa chọn nhà cung cấp, đặt trước thiết bị, bàn giao thiết bị và kiểm thử sau khi bàn giao. Khi máy tính hỗ trợ cho tiến trình này, bạn phải quyết định những hoạt động nào được hệ thống hỗ trợ, những hoạt động nào nằm ngoài giới hạn của hệ thống. Hình 5.2 chỉ ra những hoạt động nằm trong hình chữ nhật bao quanh là những hoạt động được hệ thống hỗ trợ.



Hình 5.2. Mô hình tiến trình đặt mua thiết bị

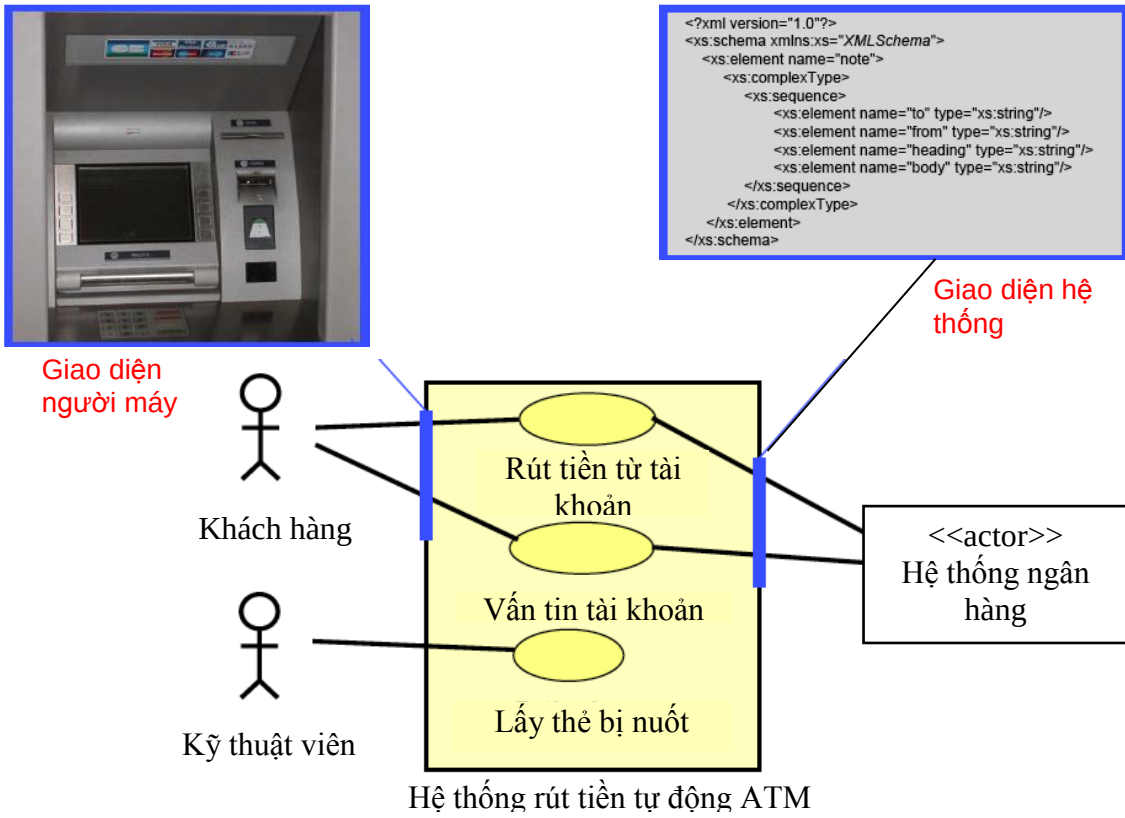
5.2.2. Mô hình trường hợp sử dụng (USE-CASE)

Trong giai đoạn phân tích, người sử dụng hợp tác cùng nhóm phát triển phần mềm tạo nên một tổ hợp thông tin quan trọng về yêu cầu đối với hệ thống. Không chỉ là người cung cấp thông tin, bản thân người sử dụng còn là một thành phần hết sức quan trọng trong bức tranh toàn cảnh đó và nhóm phát triển cần phải chỉ ra phương thức hoạt động của hệ thống tương lai theo hướng nhìn của người sử dụng. Đây là giai đoạn cần thiết để hỗ trợ cho việc xây dựng thành công những hệ thống vừa thỏa mãn yêu cầu đặt ra của khách hàng, vừa có môi trường thân thiện với người sử dụng. Công cụ giúp ta mô hình hóa hệ thống từ hướng nhìn của người sử dụng gọi là mô hình trường hợp sử dụng. Một mô hình gồm ba thành phần chính:

- Hệ thống: được thể hiện trong một hình chữ nhật trong đó có chứa tên của hệ thống.

- Tác nhân: có thể là một đối tượng sử dụng hoặc một hệ thống khác. Tác nhân là người sử dụng được thể hiện bằng một hình người. Nếu tác nhân là hệ thống thì thể hiện bằng một hình chữ nhật trong đó có ghi tên của tác nhân. Các tác nhân này giao tiếp với hệ thống thông qua giao diện. Giao diện có thể là giao diện người máy hoặc giao diện hệ thống (hình 5.3).

- Use Case: là các trường hợp sử dụng hay các chức năng của hệ thống, được thể hiện bằng một hình eclipse đi kèm với tên của chức năng (use-case).



Hình 5.3 l
một biểu đồ chứa
quan hệ giữa các t

Kết hợp với mô hình trường hợp sử dụng là lời mô tả nội dung các use-case, mô tả các tác nhân và hệ thống. Các mô tả này thường được cung cấp dưới dạng văn bản. Trong UML, lời mô tả đó được coi là thuộc tính văn bản của use-case. Lời mô tả này bao gồm những thông tin quan trọng, định nghĩa các yêu cầu và chức năng cụ thể. Đây là lời đặc tả đơn giản và nhất quán về việc các tác nhân và các use-case tương tác với nhau ra sao, tập trung vào ứng xử đối ngoại của hệ thống và không đề cập tới việc thực hiện nội bộ trong hệ thống. Ngôn ngữ và các thuật ngữ được sử dụng trong lời miêu tả chính là ngôn ngữ và các thuật ngữ được sử dụng bởi khách hàng/người dùng. Văn bản miêu tả bao gồm những điểm sau:

- *Mục đích của use-case:* Mục đích cuối cùng của use-case là gì? Các use-case nói chung đều mang tính hướng mục đích và mục đích của mỗi use-case cần phải rõ ràng.

- *Use-case được khởi chạy như thế nào:* Tác nhân nào làm use-case thực thi? Thực thi trong hoàn cảnh nào?

- *Chuỗi các thông điệp giữa tác nhân và use-case:* use-case và các tác nhân trao đổi thông điệp hay sự kiện nào để thông báo lẫn cho nhau, cập nhật hoặc nhận thông tin và giúp đỡ nhau thực thi các chức năng? Yếu tố nào sẽ miêu tả dòng chảy chính của các thông điệp giữa hệ thống và tác nhân, những thực thể nào trong hệ thống được sử dụng hoặc bị thay đổi?

- *Dòng chảy thay thế trong một use-case:* Một use-case có thể có những dòng thực thi thay thế tùy thuộc vào điều kiện. Hãy nhắc đến các yếu tố này, nhưng chú ý đừng miêu tả chúng quá chi tiết đến mức độ chúng có thể "che khuất" dòng chảy chính của các hoạt động trong trường hợp căn bản. Những động tác xử lý lỗi đặc biệt sẽ được miêu tả thành các use-case khác.

- *Use-case sẽ kết thúc như thế nào:* Hãy miêu tả khi nào use-case được coi là đã kết thúc, và kết quả mà nó cung cấp đến tác nhân.

Để bổ sung cho lời miêu tả một use-case, người ta thường đưa ra một loạt các kịch bản cụ thể để minh họa điều gì sẽ xảy ra một khi use-case được thực thi. Lời miêu tả kịch bản minh họa một trường hợp đặc biệt, khi cả tác nhân lẫn use-case đều được coi là một thực thể cụ thể. Khách hàng có thể dễ dàng hiểu hơn toàn bộ một use-case phức tạp nếu có những kịch bản được miêu tả thực tiễn hơn, minh họa lại lối ứng xử và phương thức hoạt động của hệ thống. Tuy nhiên, một lời miêu tả kịch bản chỉ là một sự bổ sung chứ không để thay thế cho lời miêu tả use-case.

Bảng 5.1. Bảng các thông tin mô tả Use-case

Tên Use-case:	**Mức độ khó:**
Tác nhân chính:	Tác nhân phụ:
Mô tả Use-case:	
Điều kiện bắt đầu (pre-condition):	
Điều kiện kết thúc (post-condition):	
Trình tự thực hiện:	
Yêu cầu phi chức năng:	
Các trường hợp ngoại lệ:	

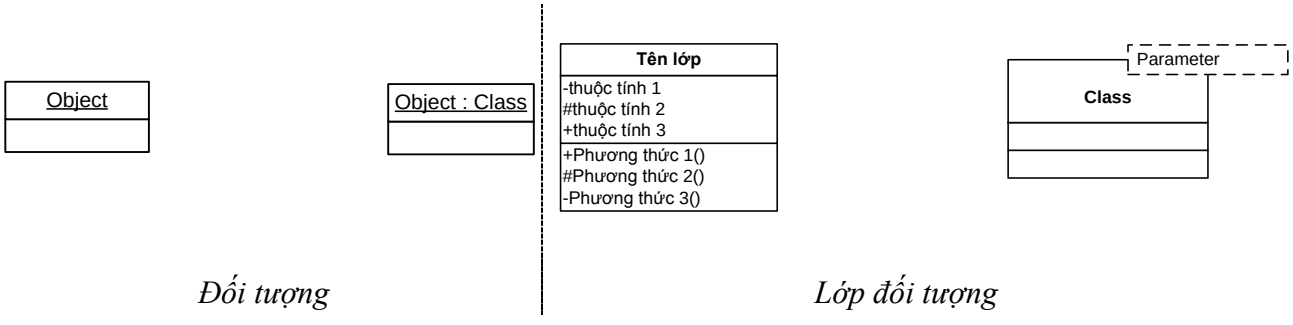
Bảng 5.1 minh họa các thông tin cần làm rõ khi mô tả chi tiết các Use-case của hệ thống. Một số thông tin trong bảng có thể bỏ trống. Một Use-case có thể có nhiều tác nhân chính và tác nhân phụ.

5.2.3. Mô hình lớp đối tượng

Với một số hệ thống, các mô hình đối tượng thường phản ánh các thực thể trong thế giới thực và được hệ thống quản lý. Điều này là hợp lý khi hệ thống xử lý những thông tin về các thực thể hữu hình, chẳng hạn như ô tô, máy bay, sách… và có những thuộc tính xác định. Ở mức trìu tượng hơn, chẳng hạn như khái niệm trong một thư viện, một hệ thống lưu trữ dữ liệu y tế, hoặc một bộ xử lý ngôn ngữ sẽ khó mô hình hóa theo phương pháp này hơn. Chúng không cần có một giao diện đơn giản gồm các thuộc tính và phương thức độc lập.

Việc phát triển các mô hình đối tượng trong quá trình phân tích yêu cầu thường giúp đơn giản hóa việc chuyển sang lập trình và thiết kế hướng đối tượng. Tuy nhiên, những người dùng cuối thường nhận thấy các mô hình đối tượng là không tự nhiên và khó hiểu. Nhiều người thích sử dụng mô hình hướng chức năng để hiển thị việc xử lý dữ liệu.

Một lớp đối tượng là sự trìu tượng hóa một tập hợp các đối tượng có chung thuộc tính, dịch vụ hoặc phương thức. Các đối tượng được xem như các thực thể với thuộc tính, dịch vụ của đối tượng. Các đối tượng là một sự kiện của một lớp đối tượng, nhiều đối tượng có thể được tạo ra từ một lớp. Nói chung, các mô hình được phát triển sử dụng việc phân tích tập trung vào các lớp đối tượng và quan hệ giữa chúng. Hình 5.4 phân biệt giữa đối tượng và lớp đối tượng.



Đối tượng *Lớp đối tượng*

Hình 5.4. Phân biệt đối tượng và lớp đối tượng

Một biểu đồ lớp là một dạng mô hình tĩnh. Một biểu đồ lớp miêu tả hướng nhìn tĩnh của hệ thống bằng các khái niệm lớp và mối quan hệ giữa chúng với nhau. Mặc dù cũng có những nét tương tự như mô hình dữ liệu, nhưng các lớp không phải chỉ thể hiện cấu trúc thông tin mà còn miêu tả cả hành vi.

Một trong các mục đích của biểu đồ lớp là tạo nền tảng cho các biểu đồ khác, thể hiện các khía cạnh khác của hệ thống (ví dụ như biểu đồ trạng thái của đối tượng hay biểu đồ cộng tác giữa các đối tượng). Một lớp trong một biểu đồ lớp có thể được thực thi trực tiếp trong một ngôn ngữ hướng đối tượng có hỗ trợ trực tiếp khái niệm lớp. Một biểu đồ lớp chỉ ra các lớp, nhưng bên cạnh đó còn được dùng để chỉ ra các đối tượng thật sự là các thực thể của các lớp này (biểu đồ đối tượng).

a) Biểu diễn đối tượng

UML thể hiện lớp bằng hình chữ nhật có 3 phần. Phần thứ nhất chứa tên lớp, phần thứ hai là thuộc tính và các dữ liệu thành phần của lớp, phần thứ ba là các phương thức hay hàm thành phần của lớp.

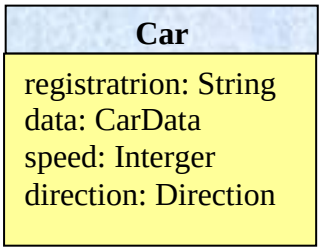
- *Tên lớp (class name):* Tên lớp được in đậm và căn giữa. Tên lớp phải được dẫn xuất từ phạm vi vấn đề và rõ ràng nhất có thể. Vì thế nó là danh từ, ví dụ như tài khoản, nhân viên...

- *Thuộc tính (attribute):* Lớp có thuộc tính miêu tả những đặc điểm của đối tượng. Giá trị của thuộc tính thường là những dạng dữ liệu đơn giản được đa phần các ngôn ngữ lập trình hỗ trợ như Integer, Boolean, Floats, Char…

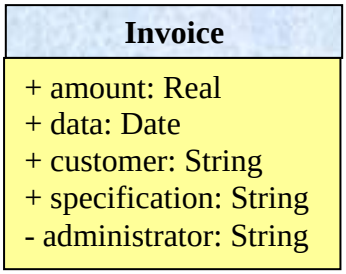
Thuộc tính có thể có nhiều mức độ trông thấy được (visibility) khác nhau, miêu tả liệu thuộc tính đó có thể được truy xuất từ các lớp khác với lớp định nghĩa ra nó. Nếu thuộc tính có tính trông thấy là công cộng (public – kí hiệu "+"), thì nó có thể được nhìn thấy và sử dụng ngoài lớp đó. Nếu thuộc tính có tính trông thấy là riêng (private – kí hiệu "-"), bạn sẽ không thể truy cập nó từ bên ngoài lớp đó. Một thuộc tính trông thấy khác là bảo vệ (protected), được sử dụng chung với công cụ khái quát hóa và chuyên biệt hóa. Nó cũng giống như các thuộc tính riêng nhưng được thừa kế bởi các lớp dẫn xuất.

Giá trị được gán cho thuộc tính có thể là một cách để miêu tả trạng thái của đối tượng, khi các giá trị này thay đổi thì trạng thái của đối tượng cũng thay đổi theo.

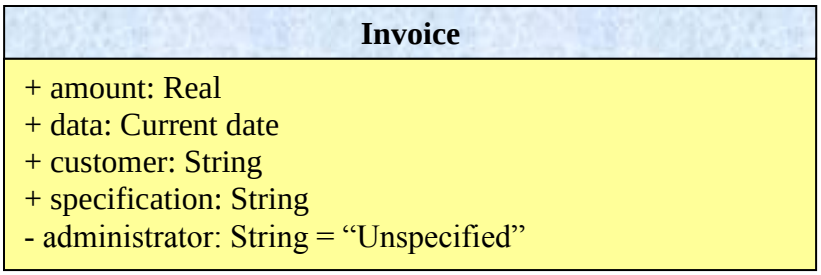
- *Phương thức (methods):* Phương thức định nghĩa các hoạt động mà lớp có thể thực hiện. Tất cả các đối tượng được tạo từ một lớp sẽ có chung thuộc tính và phương thức. Phương thức được sử dụng để thay đổi giá trị thuộc tính cũng như thực hiện các công việc khác. Phương thức thường được gọi là các hàm (function), nhưng chúng nằm trong một lớp và chỉ có thể được áp dụng cho các đối tượng của lớp này. Một phương thức được miêu tả qua tên, giá trị trả về và danh sách của không hoặc nhiều tham số. Lúc thi hành, phương thức được gọi kèm theo tên một đối tượng của lớp. Vì nhóm các phương thức miêu tả những dịch vụ mà lớp có thể cung cấp nên chúng được coi là giao diện của lớp này. Giống như thuộc tính, phương thức cũng có tính trông thấy được như công cộng, riêng và bảo vệ.



Hình 5.5. Một lớp với các thuộc tính tiêu biểu

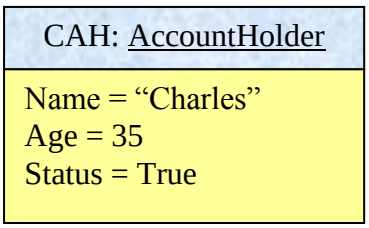


Hình 5-6. Một lớp với các thuộc tính chung và riêng



Hình 5.7. Một lớp với các thuộc tính và giá trị mặc định

Đối tượng là thực thể của lớp nên kí hiệu dùng cho đối tượng cũng là kí hiệu dùng cho lớp. Tuy nhiên, trong UML mỗi đối tượng chỉ bao gồm hai phần: tên đối tượng và danh sách các thuộc tính của đối tượng được gán giá trị cụ thể. Hình 5.8 là một ví dụ về việc biểu diễn đối tượng trong UML.



Hình 5.8. Ký hiệu đối tượng

Hình 5.8 được đọc như sau: CAH là đối tượng của lớp AccountHolder. Các thuộc tính được gán giá trị, đây là các giá trị khi lớp được thực thể hóa. Chú ý rằng kí hiệu đối tượng không chứa phần phương thức.

b) Quan hệ giữa các lớp đối tượng

Biểu đồ lớp thể hiện các lớp và mối quan hệ giữa chúng. Quan hệ giữa các lớp gồm có bốn loại:

- Liên hệ (Association)

- Khái quát hóa (Generalization)

- Phụ thuộc (Dependency) và nâng cấp (Refinement)

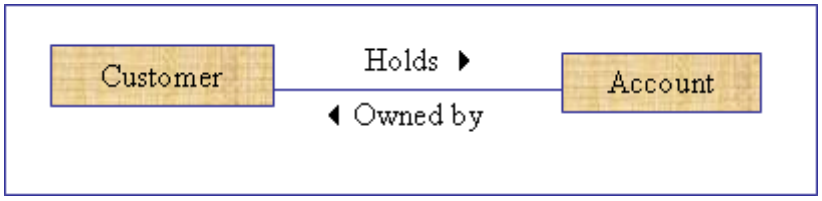
Một *liên hệ* là một sự nối kết giữa các lớp, cũng có nghĩa là sự nối kết giữa các đối tượng của các lớp này. Trong UML, một liên hệ được định nghĩa là một mối quan hệ miêu tả một tập hợp các liên kết (là một sự liên quan về ngữ nghĩa giữa một nhóm các đối tượng).

Khái quát hóa là mối quan hệ giữa một yếu tố mang tính khái quát cao hơn và một yếu tố mang tính chuyên biệt hơn. Yếu tố mang tính chuyên biệt hơn có thể chỉ chứa các thông tin bổ sung. Một thực thể (một đối tượng là một thực thể của một lớp) của yếu tố mang tính chuyên biệt hơn có thể được sử dụng ở bất cứ nơi nào mà đối tượng mang tính khái quát hóa hơn được phép.

Sự phụ thuộc là một mối quan hệ giữa các yếu tố, gồm một yếu mang tính độc lập và một yếu tố mang tính phụ thuộc. Một sự thay đổi trong yếu tố độc lập sẽ ảnh hưởng đến yếu tố phụ thuộc. Một sự nâng cấp là mối quan hệ giữa hai lời miêu tả của cùng một sự vật, nhưng ở những mức độ trừu tượng hóa khác nhau.

- ***Liên hệ (association)***

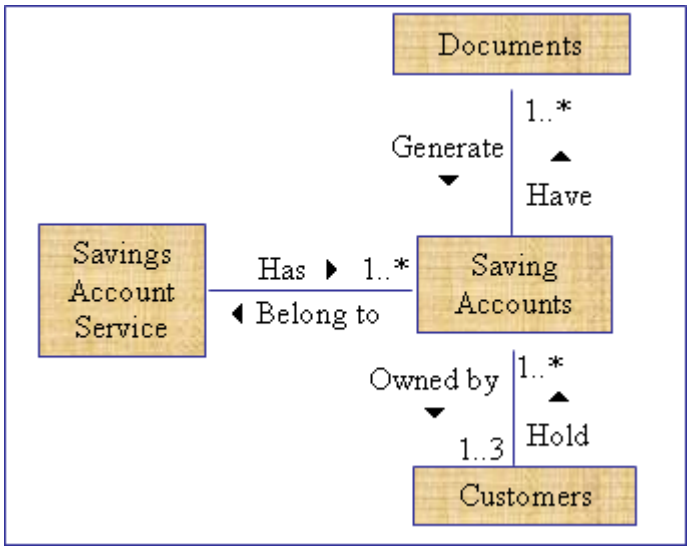
Một liên hệ là một sự nối kết giữa các lớp, một liên quan về ngữ nghĩa giữa các đối tượng của các lớp tham gia. Liên hệ thường mang tính hai chiều, có nghĩa khi một đối tượng này có liên hệ với một đối tượng khác thì cả hai đối tượng này nhận thấy nhau. Một mối liên hệ biểu thị bằng các đối tượng của hai lớp có nối kết với nhau, ví dụ: "chúng biết về nhau", "được nối với nhau", "cứ mỗi X lại có một Y"… Mối liên kết được thể hiện trong biểu đồ UML bằng một đường thẳng nối hai lớp. Trên đường thẳng này có thể có chú thích thêm tên vai trò, là một chức năng mà lớp đó đảm nhận nhìn từ góc nhìn của lớp kia. Tên vai trò được viết kèm với một mũi tên chỉ từ hướng lớp chủ nhân ra, thể hiện lớp này đóng vai trò như thế nào đối với lớp mà mũi tên chỉ đến.



Hình 5.9. Vai trò trong liên hệ giữa Customer và Account

Trong ví dụ trên: một khách hàng có thể là chủ nhân của một tài khoản và tài khoản được chiếm giữ bởi khách hàng. Đường thẳng thể hiện liên hệ giữa hai lớp.

Ở hai đầu đường thẳng thể hiện liên hệ giữa hai lớp có thể có ***số lượng của liên hệ***. Số lượng được ghi ở phía đầu đường thẳng thể hiện liên hệ, sát vào lớp là miền áp dụng của nó. Phạm vi của số lượng phần tử trong liên hệ có thể từ 0-tới-1 (0..1), 0-tới-nhiều (0..* hay **),** một-tới-nhiều (1..), hai (2), năm-tới-mười một (5..11). Cũng có thể miêu tả một dãy số ví dụ (1,4,6, 8..12). Giá trị mặc định là 1.



Hình 5.10. Một biểu đồ lớp tiêu biểu

Hình 5.10 là ví dụ cho một biểu đồ lớp tiêu biểu. Biểu đồ giải thích rằng bộ phận dịch vụ tài khoản tiết kiệm của một nhà băng có thể có nhiều tài khoản tiết kiệm nhưng tất cả những tài khoản này đều thuộc về bộ phận đó. Một tài khoản tiết kiệm lại có thể có nhiều tài liệu, nhưng những tài liệu này chỉ thuộc về một tài khoản tiết kiệm mà thôi. Một tài khoản tiết kiệm có thể thuộc về từ một cho tới nhiều nhất là ba khách hàng. Mỗi khách hàng có thể có nhiều hơn một tài khoản.

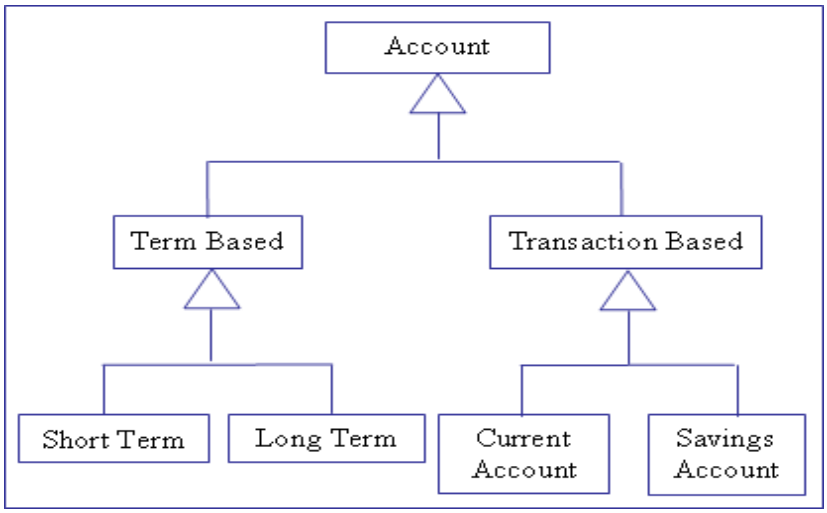
- ***Khái quát hóa và chuyên biệt hóa (generalization & specialization)***

Trong biểu đồ trên, các lớp trong một cấu trúc cây được nối với nhau bằng một mũi tên rỗng, chỉ từ lớp chuyên biệt hơn tới lớp khái quát hơn.

Quá trình bắt đầu với một lớp khái quát để sản xuất ra các lớp mang tính chuyên biệt cao hơn được gọi là quá trình ***chuyên biệt hoá (Specialization).*** Chuyên biệt hóa là quá trình tinh chế một lớp thành những lớp chuyên biệt hơn. Chuyên biệt hóa bổ sung thêm chi tiết và đặc tả cho lớp kết quả. Lớp mang tính khái quát được gọi là ***lớp cha (superclass)***, kết quả chuyên biệt hóa là việc tạo ra các ***lớp con (Subclass)***.

Chuyên biệt hóa và khái quát hóa là hai con đường khác nhau để xem xét cùng một mối quan hệ. Một lớp là lớp con của một lớp này có thể đóng vài trò là một lớp cha của lớp khác.

Hãy quan sát cấu trúc lớp trong biểu đồ sau:



Hình 5.11. Khái quát hóa và chuyên biệt hoá

Trong hình trên, tài khoản là khái niệm chung của các loại tài khoản khác nhau và chứa những đặc tả cần thiết cho tất cả các loại tài khoản. Ví dụ như nó có thể chứa số tài khoản và tên chủ tài khoản. Ta có thể có hai loại tài khoản đặc biệt suy ra từ dạng tài khoản chung này, một loại mang tính kỳ hạn và một loại mang tính giao dịch. Yếu tố chia cách hai lớp này với nhau là các quy định chuyên ngành hay đúng hơn là phương thức hoạt động của hai loại tài khoản.

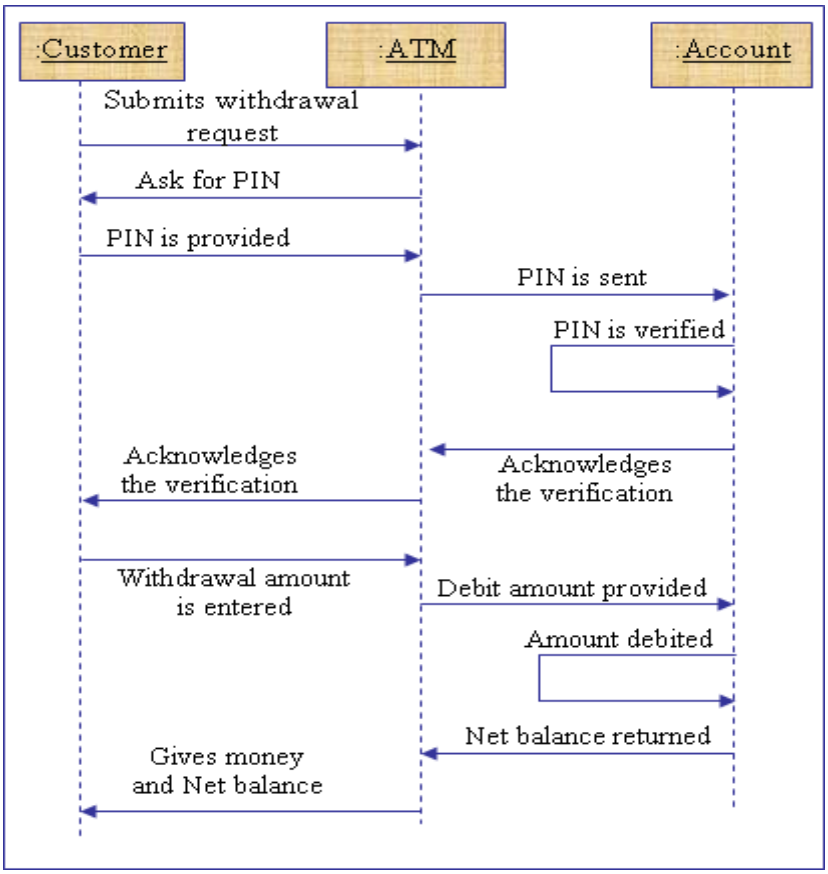
Tương tự như vậy, tài khoản đầu tư trung hạn và dài hạn lại là những khái niệm chuyên biệt của khái niệm tài khoản có kỳ hạn. Mặt khác, tài khoản bình thường và tài khoản tiết kiệm là những trường hợp đặc biệt của loại tài khoản giao dịch.

5.2.4. Mô hình tuần tự (Sequence diagram)

Biểu đồ tuần tự minh họa các đối tượng tương tác với nhau ra sao. Chúng tập trung vào các chuỗi thông điệp, có nghĩa là các thông điệp được gửi và nhận giữa một loạt các đối tượng như thế nào. Biểu đồ tuần tự có hai trục: trục nằm dọc chỉ thời gian, trục nằm ngang chỉ ra một tập hợp các đối tượng.

Từ các hình chữ nhật biểu diễn đối tượng có các đường gạch rời thẳng đứng biểu thị đường đời đối tượng, tức là sự tồn tại của đối tượng trong chuỗi tương tác. Trong khoảng thời gian này, đối tượng được thực thể hóa, sẵn sàng để gửi và nhận thông điệp. Quá trình giao tiếp giữa các đối tượng được thể hiện bằng các đường thẳng, thông điệp nằm ngang nối các đường đời đối tượng. Mũi tên ở đầu đường thẳng sẽ chỉ ra loại thông điệp này mang tính đồng bộ, không đồng bộ hay đơn giản. Để đọc biểu đồ tuần tự, hãy bắt đầu từ phía bên trên của biểu đồ rồi chạy dọc xuống và quan sát sự trao đổi thông điệp giữa các đối tượng xảy ra dọc theo tiến trình thời gian.

Ví dụ hãy quan sát một cảnh kịch rút tiền mặt tại một máy ATM của một ngân hàng trong hình 5.12.



Hình 5.12. Biểu đồ kịch bản chức năng rút tiền mặt tại máy ATM

Biểu đồ trên có thể được diễn giải theo trình tự thời gian như sau:

- Có ba lớp tham gia kịch bản này: khách hàng, máy ATM và tài khoản.

- Khách hàng đưa yêu cầu rút tiền vào máy ATM.

- Đối tượng máy ATM yêu cầu khách hàng cung cấp mã số.

- Mã số được gửi cho hệ thống để kiểm tra tài khoản.

- Đối tượng tài khoản kiểm tra mã số và báo kết quả kiểm tra đến cho ATM.

- ATM gửi kết quả kiểm tra này đến khách hàng.

- Khách hàng nhập số tiền cần rút.

- ATM gửi số tiền cần rút đến cho tài khoản.

- Đối tượng tài khoản trừ số tiền đó vào mức tiền trong tài khoản. Tại thời điểm này, chúng ta thấy có một mũi tên quay trở lại chỉ vào đối tượng tài khoản. Ý nghĩa của nó là đối tượng tài khoản xử lý yêu cầu này trong nội bộ đối tượng và không gửi sự kiện đó ra ngoài.

- Đối tượng tài khoản trả về mức tiền mới trong tài khoản cho máy ATM.

- Đối tượng ATM trả về mức tiền mới trong tài khoản cho khách hàng và dĩ nhiên, cả lượng tiền khách hàng đã yêu cầu được rút.

Đối tượng tài khoản chỉ bắt đầu được sinh ra khi đối tượng ATM cần tới nó để kiểm tra mã số và đối tượng tài khoản tiếp tục sống cho tới khi giao dịch được hoàn tất. Sau đó, nó chết đi.

Bởi khách hàng có thể muốn tiếp tục thực hiện các giao dịch khác nên đối tượng khách hàng và đối tượng máy ATM vẫn tiếp tục tồn tại, điều này được chỉ ra qua việc các đường đời đối tượng được kéo vượt quá đường thẳng thể hiện sự kiện cuối cùng trong chuỗi tương tác.

Loại tương tác này là rất hữu dụng trong một hệ thống có một số lượng nhỏ các đối tượng với một số lượng lớn các sự kiện xảy ra giữa chúng. Mặc dù vậy, khi số lượng các đối tượng trong một hệ thống tăng lên thì mô hình này sẽ không còn mấy thích hợp.

Để có thể vẽ biểu đồ tuần tự, đầu tiên hãy xác định các đối tượng liên quan và thể hiện các sự kiện xảy ra giữa chúng. Khi vẽ biểu đồ tuần tự, cần chú ý:

- Sự kiện được biểu diễn bằng các đường thẳng nằm ngang.

- Đối tượng bằng các đường nằm dọc.

- Thời gian được thể hiện bằng đường thẳng nằm dọc bắt đầu từ trên biểu đồ. Điều đó có nghĩa là các sự kiện cần phải được thể hiện theo đúng thứ tự mà chúng xảy ra, vẽ từ trên xuống dưới.

5.2.5. Mô hình trạng thái máy

Mô hình trạng thái máy mô tả hệ thống phản ứng lại các sự kiện bên trong và bên ngoài. Mô hình trạng thái máy biểu diễn các trạng thái của hệ thống và các sự kiện gây ra sự biến đổi trạng thái này chứ không chỉ ra luồng dữ liệu trong hệ thống. Kiểu mô hình này thường được sử dụng trong các hệ thống thời gian thực bởi thông thường các hệ thống này được điều khiển bởi việc mô phỏng từ môi trường của hệ thống.

Các mô hình trạng thái máy là một phần của phương pháp thiết kế thời gian thực. Trong UML có lược đồ trạng thái (StateChart) để biểu diễn mô hình này.

Mô hình trạng thái máy của một hệ thống chỉ ra rằng, tại bất kỳ thời điểm nào, hệ thống cũng có một tập hợp các trạng thái có thể. Khi một tín hiệu được nhận, nó có thể biến đổi hệ thống từ trạng thái này sang trạng thái khác. Ví dụ một hệ thống điều khiển van có thể chuyển từ trạng thái van đóng sang trạng thái van mở khi hệ thống nhận được lệnh.

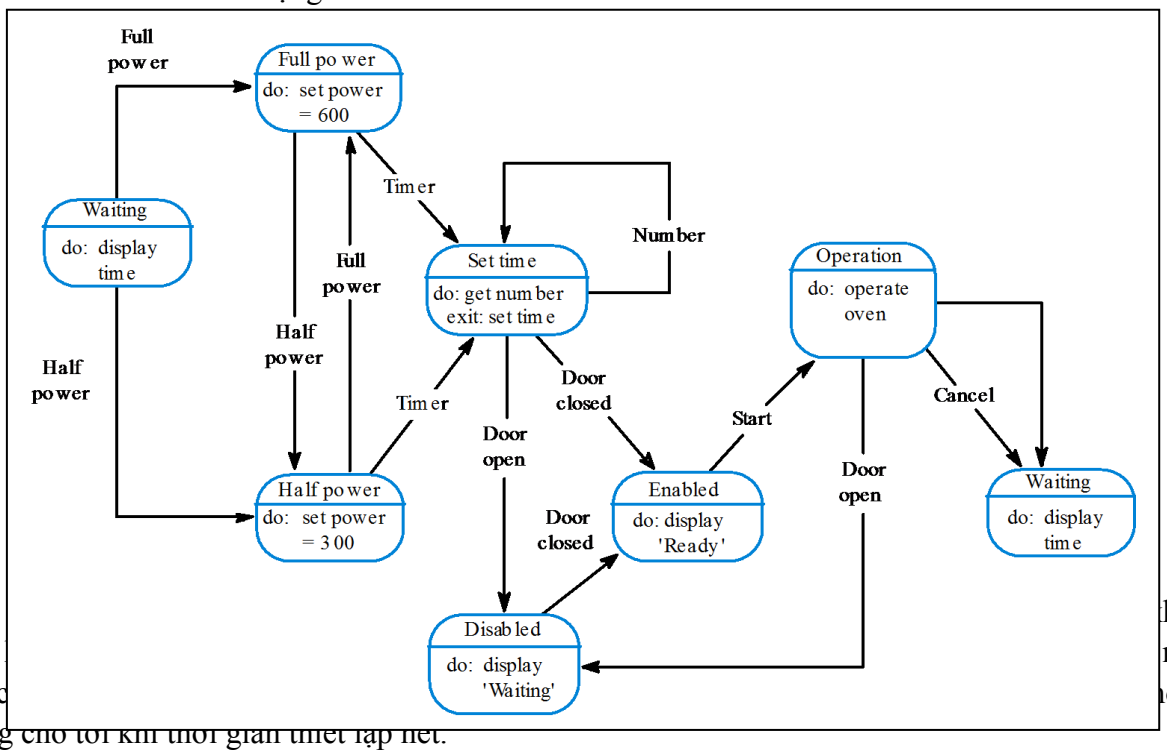
Cách tiếp cận để mô hình hóa hệ thống được minh họa trong hình 5.14. Sơ đồ này chỉ ra một mô hình trạng thái máy của một lò vi sóng đơn giản có gắn các nút chọn nhiệt độ (power) và nút thiết lập thời gian bắt đầu. Lò vi sóng hiện đại thì có nhiều chức năng phức tạp hơn. Tuy nhiên, mô hình này bao gồm những chức năng cơ bản của hệ thống. Để đơn giản hóa, ta giả sử rằng các bước tuần tự để sử dụng lò vi sóng là:

1. Chọn nhiệt độ
2. Đặt thời gian
3. Ấn nút start, máy sẽ hoạt động cho đến khi thời gian kết thúc.

Vì lý do an toàn, lò vi sóng sẽ không hoạt động khi cánh cửa được mở, khi hoàn thành việc sử dụng, có một âm thanh nhắc nhở. Lò vi sóng có thể có nhiều cách hiển thị thông báo và cảnh báo khác nhau.

Một lược đồ UML được sử dụng để mô tả mô hình trạng thái máy được thiết kế theo phương pháp hướng đối tượng. Tuy nhiên, nó cũng có thể sử dụng cho tất cả các mô hình trạng thái máy. Hình chữ nhật góc tròn biểu diễn trạng thái của hệ thống, trong đó có chứa một lời mô

tả ngắn gọn về hoạt động có thể được thực hiện trong trạng thái này. Một mũi tên được gán nhãn chỉ ra tác nhân biến đổi trạng thái.

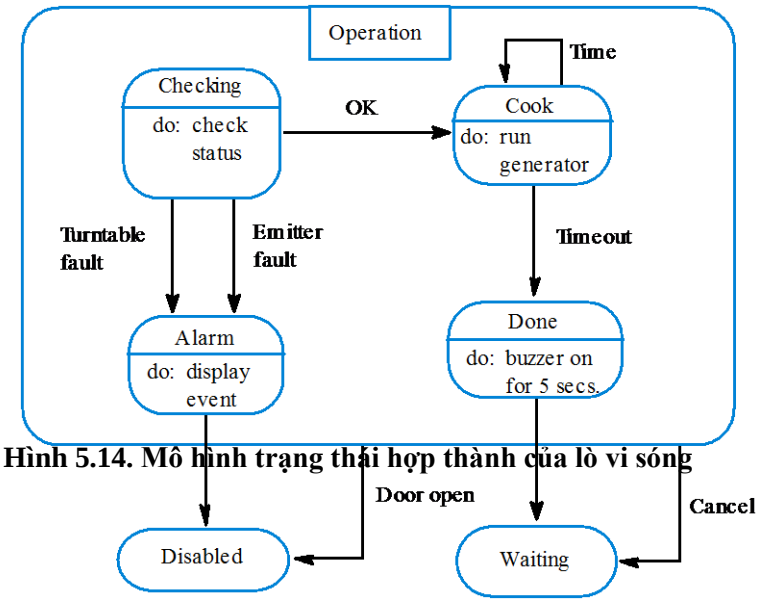


đầu hỏi
khác nút
động cho tới khi thời gian thiết lập hết. hoạt

Lược đồ UML để chỉ ra các hoạt động được thực hiện trong một trạng thái. Trong một đặc tả hệ thống chi tiết, bạn phải cung cấp thông tin chi tiết hơn về việc kích hoạt và các trạng thái của hệ thống, những thông tin này có thể được lưu trong một từ điển dữ liệu hoặc một bách khoa thư được lưu trữ bởi một công cụ CASE được sử dụng để tạo ra mô hình hệ thống.

Vấn đề nảy sinh với cách tiếp cận này là số lượng các trạng thái của hệ thống có thể tăng lên nhanh chóng. Đối với những mô hình hệ thống lớn, việc cấu trúc hóa các mô hình trạng thái này là cần thiết.

Một cách để thực hiện điều này là sử dụng một khái niệm bao chứa các khái niệm khác (một trạng thái có thể được chi tiết hóa trong một mô hình khác – một trạng thái có nhiều trạng thái thành phần). Ví dụ như trong hình 5.14.



Hình 5.14. Mô hình trạng thái hợp thành của lò vi sóng

CÂU HỎI ÔN TẬP

1. Hãy nêu vai trò của việc mô hình hóa hệ thống trong tiến trình phát triển phần mềm.
2. UML là gì? Vai trò của các loại biểu đồ UML trong các giai đoạn phát triển phần mềm.
3. Hãy xây dựng mô hình ngữ cảnh và mô hình trường hợp sử dụng (use-case) của hệ thống máy rút tiền tự động ATM.
4. Xác định các lớp và xây dựng mô hình lớp của hệ thống phần mềm nhúng điều khiển hoạt động của hệ thống máy rút tiền tự động ATM.
5. Xây dựng mô hình tuần tự của chức năng chuyển tiền từ tài khoản này sang tài khoản khác trong máy rút tiền tự động ATM.
6. Xây dựng mô hình trạng thái của lớp đối tượng thẻ ATM trong hệ thống rút tiền tự động.

Chương 6: THIẾT KẾ PHẦN MỀM

Trong phát triển phần mềm, giai đoạn phân tích và đặc tả yêu cầu sẽ xác định những chức năng cơ bản của phần mềm và các ràng buộc đối với phần mềm cũng như tiến trình phát triển. Nói cách khác, giai đoạn này phải trả lời câu hỏi "hệ thống cần làm gì?". Sau khi xác định rõ mục tiêu cần thực hiện sẽ chuyển qua giai đoạn thiết kế, giai đoạn này trả lời câu hỏi "làm như thế nào" thông qua việc đưa ra các giải pháp để thực hiện những yêu cầu đã đề ra trong quá trình đặc tả.

Chương này sẽ giới thiệu về các khái niệm cơ bản trong thiết kế, vai trò và các hoạt động cơ bản trong thiết kế. Phần đầu tiên của chương đề cập đến một số chiến lược và phương pháp thiết kế. Phần tiếp theo trình bày về các vấn đề cơ bản trong thiết kế kiến trúc. Phần cuối cùng giới thiệu về một phương pháp thiết kế được sử dụng rộng rãi bên cạnh phương pháp thiết kế hướng chức năng mà sinh viên đã quen thuộc trong các môn học trước, đó là phương pháp thiết kế hướng đối tượng. Qua đó sinh viên có thể hiểu được những nguyên tắc, công việc phải thực hiện trong quá trình thiết kế phần mềm. Từ đó có khả năng ứng dụng trong các đề án thực tế.

6.1. TỔNG QUAN VỀ THIẾT KẾ PHẦN MỀM

6.1.1. Giới thiệu chung

a. Khái niệm thiết kế

"Thiết kế phần mềm là quá trình chuyển các đặc tả yêu cầu phần mềm thành một biểu diễn thiết kế của hệ thống phần mềm cần xây dựng, sao cho người lập trình có thể dựa trên cơ sở đó để xây dựng thành các chương trình vận hành được".

Cụ thể hơn, người kỹ sư thiết kế phần mềm cần dựa vào các mô tả về các dịch vụ mà phần mềm sẽ cung cấp cũng như những ràng buộc cần tuân thủ khi hoạt động để đưa ra các giải pháp công nghệ thích hợp, sẽ vận hành trên thực tế, nhằm đáp ứng các yêu cầu đặt ra.

Mục tiêu của giai đoạn này là đưa ra một giải pháp cho vấn đề đã đặt ra trong giai đoạn đặc tả và cấu trúc hoá giải pháp này một cách rõ ràng, đầy đủ, không nhập nhằng, không có sự dư thừa. Cuối cùng sẽ mô tả phương án thiết kế trong một tài liệu thiết kế.

Để đưa ra được các giải pháp phù hợp, người thiết kế phải thực hiện các công việc sau:

- Nghiên cứu tìm hiểu vấn đề.
- Chọn một số giải pháp và xác định các đặc điểm ban đầu của chúng. Các giải pháp này cần khả thi và có hiệu quả cao đối với hệ thống.
- Mô tả trìu tượng cho mỗi giải pháp thiết kế theo phương châm chi tiết hóa dần từng bước nhằm phát hiện và chỉnh sửa các sai sót trước khi đưa ra tài liệu thiết kế chính thức.

Hoạt động thiết kế là một hoạt động rất phức tạp, đòi hỏi sự chính xác, sáng tạo và kinh nghiệm thực tiễn.

b. Vai trò của thiết kế

Trong phát triển phần mềm, nếu bỏ qua giai đoạn thiết kế thì nguy cơ sinh ra một hệ thống không tin cậy, khả năng thất bại cao. Đặc biệt khi phần mềm là một sản phẩm vô hình và rất phức tạp, rất khó xác định chất lượng trước khi kiểm thử và vận hành hệ thống. Thiết kế là một khâu then chốt quyết định chất lượng sản phẩm. Về cơ bản, hoạt động thiết kế có vai trò:

- Thiết kế là cách duy nhất để chuyển hóa một cách chính xác các yêu cầu của khách hàng thành mô hình hệ thống phần mềm, làm cơ sở cho việc triển khai chương trình phần mềm.
- Tài liệu thiết kế là cơ sở để giao tiếp giữa các nhóm cùng tham gia vào việc phát triển sản phẩm. Giúp quản lý rủi ro và lập kế hoạch phát triển phần mềm một cách hiệu quả.
- Một tài liệu thiết kế bao gồm một tập hợp các mô hình cấu trúc hoá, đồng nhất và hoàn thiện, các mô hình này có thể được hiển thị theo nhiều cách khác nhau (bằng các ngôn ngữ hình thức hoặc phi hình thức). Do đó, nó có thể là tài liệu tham khảo quan trọng trong các bước tiếp theo của tiến trình phát triển phần mềm như: giai đoạn phát triển, giai đoạn kiểm thử, giai đoạn bảo trì.

c. Một số khái niệm cơ bản trong thiết kế

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

- ***Trừu tượng hóa (abstraction)***

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

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

- ***Phân rã (decomposition)***

Phân rã là việc phân chia một đối tượng thành những đối tượng nhỏ hơn. Đây là một cách để có thể dễ dàng nghiên cứu các đối tượng con đơn giản hơn của nó.

- ***Làm mịn (Refinement)***

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

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

- ***Tính module***

Phần mềm được chia thành các phần có tên riêng biệt và định địa chỉ được, gọi là các module. Các module được tích hợp với nhau để giải quyết yêu cầu của vấn đề đặt ra. Khái niệm này có ý nghĩa rất quan trọng trong quá trình thiết kế: đây là một thuộc tính riêng của phần mềm, nó cho phép tổ chức một chương trình trở nên quản lý được theo một cách thông minh.

Nếu một vấn đề x có thể phân thành hai vấn đề x_1, x_2 nhỏ hơn để giải quyết thì thực nghiệm đã chứng minh được rằng:

$C(x_1 + x_2) > C(x_1) + C(x_2)$ và $E(x_1 + x_2) > E(x_1) + E(x_2)$

Trong đó C(x) là hàm xác định độ phức tạp cảm nhận được của vấn đề x, và E(x) là hàm xác định nỗ lực cần có để giải quyết vấn đề x. Như vậy, khi chia nhỏ một vấn đề thì việc giải quyết nó trở nên dễ hơn và công sức bỏ ra để giải quyết cũng ít hơn.

- ***Thủ tục phần mềm (software procedure)***

Thủ tục phần mềm tập trung vào việc mô tả chi tiết các bước xử lý cho từng module riêng biệt. Thủ tục phải cung cấp một đặc tả chính xác về một quá trình xử lý, có đầu vào, đầu ra, trình tự các sự kiện, các điểm quyết định rẽ nhánh điều khiển, các thao tác lặp lại, có thể bao gồm cả cấu trúc/tổ chức của dữ liệu đã được sử dụng.

- ***Che dấu thông tin (information hiding)***

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

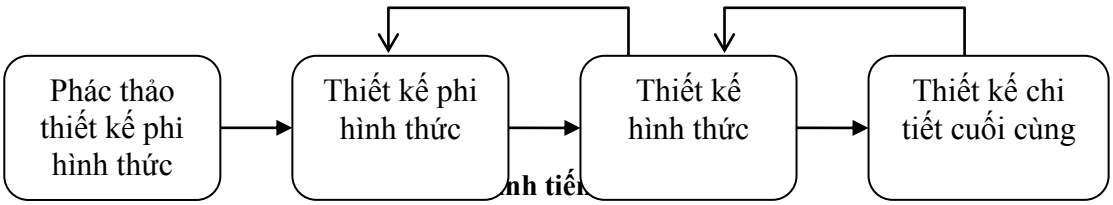
Che dấu thông tin là một tiêu chuẩn thiết kế đối với hệ thống module vì những lợi ích mà nó mang lại. Khi có sai sót xảy ra, sự thay đổi sẽ ít có khả năng lan truyền sang những vị trí khác bên trong phần mềm.

6.1.2. Thiết kế phần mềm

Tiến trình thiết kế phần mềm có thể xem xét từ những góc độ khác nhau: *nội dung công việc, trình tự thực hiện, phương pháp thiết kế và công cụ sử dụng.*

a. Tiến trình thiết kế

Mỗi giai đoạn trong tiến trình thiết kế là tiến hành làm cụ thể hóa dần các mức độ trìu tượng. Tiến trình thiết kế khởi đầu bằng việc tập trung vào lĩnh vực cần phát triển phần mềm, sau đó giảm dần mức độ trìu tượng để đi đến mã nguồn. Tiến trình thiết kế có thể mô tả như hình 6.1.



nh tiến

Sau mỗi hoạt động thiết kế, nhóm thiết kế cần đưa ra một bản đặc tả thiết kế. Bản đặc tả này có thể là một đặc tả trìu tượng, nửa hình thức, hình thức hay cũng có thể là một đặc tả về một phần nào đó của hệ thống phải được thực hiện như thế nào. Thực chất quá trình thiết kế là việc bổ sung dần các chi tiết vào đặc tả thiết kế. Kết quả cuối cùng là các đặc tả về các thuật toán, cấu trúc dữ liệu và đặc tả giao diện được dùng làm cơ sở cho việc mã hóa.

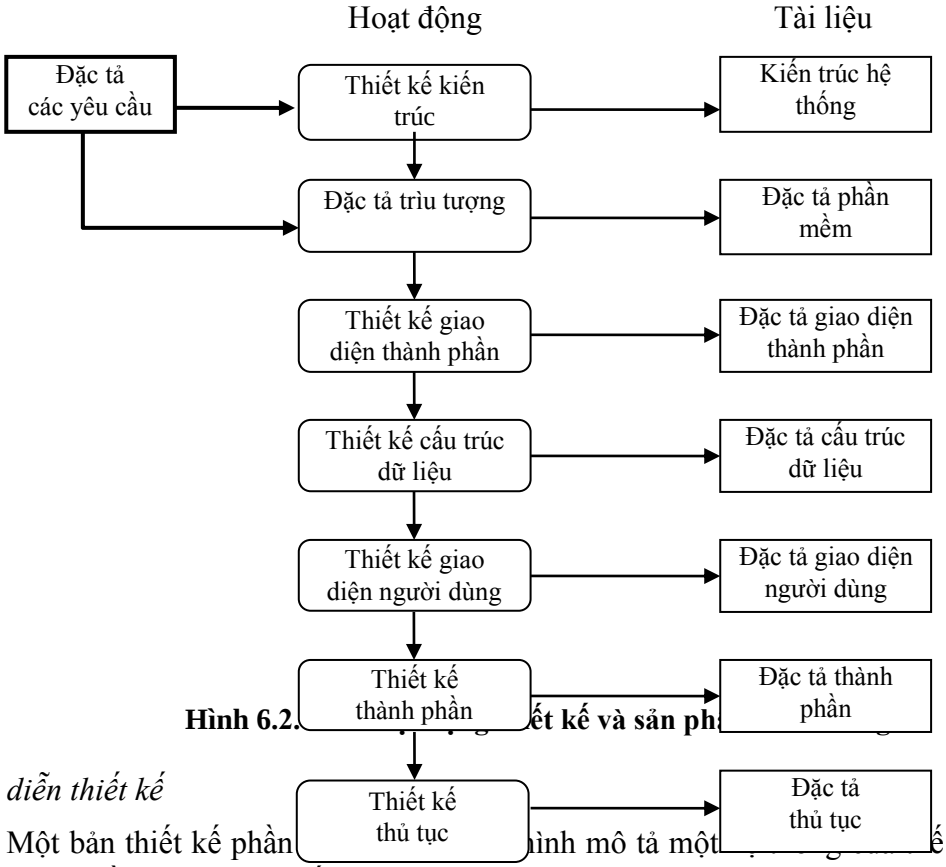
Phương pháp tiếp cận hệ thống thường được sử dụng là phương pháp tiếp cận từ trên xuống: vấn đề được phân chia một cách đệ quy thành các vấn đề con cho tới khi các vấn đề nhận được có thể giải quyết một cách dễ dàng. Ngược lại, khi thiết kế người ta lại thường thực hiện từ dưới lên, vì các vấn đề ở mức thấp nhất là đủ nhỏ, đủ thông tin cần thiết để người thiết kế có thể dễ dàng tìm ra giải pháp công nghệ thích hợp và cũng dễ nhận ra thành phần nào có thể dùng lại được trong phần khác hay chương trình khác.

Quá trình này được lặp lại cho đến khi các thành phần hợp thành của mỗi hệ con được xác định để có thể ánh xạ trực tiếp vào các thành phần khác (ví dụ như các gói, các thủ tục và các hàm) biểu diễn bằng một ngôn ngữ lập trình nào đó.

b. Các hoạt động và sản phẩm thiết kế

Đối với việc phát triển các hệ thống phần mềm lớn, hoạt động thiết kế được tiến hành theo các bước sau:

- Thiết kế kiến trúc: Xác định các hệ thống con tạo nên hệ thống và mối liên hệ giữa chúng.
- Đặc tả trìu tượng: Đối với mỗi hệ thống con, mô tả trìu tượng các dịch vụ mà nó cung cấp và các ràng buộc mà nó phải tuân thủ khi cung cấp dịch vụ.
- Thiết kế các giao diện thành phần: Thiết kế giao diện của từng hệ con khi chúng giao tiếp với hệ con khác, các hệ thống khác trong cùng môi trường hoạt động của hệ thống sao cho các hệ con có thể thực thi các dịch vụ trong các hệ con khác mà không cần biết quá trình thực hiện được diễn ra như thế nào.
- Thiết kế cấu trúc dữ liệu: thiết kế và đặc tả cấu trúc dữ liệu được dùng khi phát triển hệ thống.
- Thiết kế giao diện người dùng: Thiết kế các giao diện người dùng để người sử dụng có thể tương tác với hệ thống.
- Thiết kế thành phần: Phân chia các dịch vụ mà các hệ thống con cung cấp vào các thành phần của nó.
- Thiết kế thủ tục: Đặc tả các thuật toán, quy trình thực hiện các dịch vụ của mỗi thành phần sao cho có thể ánh xạ trực tiếp vào ngôn ngữ lập trình.



Hình 6.2. ết kế và sản ph

c. Biểu diễn thiết kế

Một bản thiết kế phần ình mô tả một ế giới thực với rất nhiều thành phần và những mối quan hệ giữa chúng. Thông thường người ta sử dụng ba hình thức sau để biểu diễn thiết kế:

- *Các biểu đồ:* Các biểu đồ dùng để thể hiện mối liên hệ giữa các thành phần tạo nên hệ thống và là mô hình mô tả thế giới thực. Việc sử dụng các biểu đồ để mô hình hóa hệ thống có rất nhiều ưu điểm, nó mô tả một cách trực quan bức tranh tổng thể về hệ thống.
- *Ngôn ngữ mô tả chương trình:* Các ngôn ngữ này được dùng để kiểm tra và cấu trúc các thiết kế dựa trên cấu trúc của một ngôn ngữ lập trình được lựa chọn để phát triển hệ thống.
- *Dạng văn bản bằng ngôn ngữ tự nhiên:* Dạng biểu diễn này sẽ giúp mô tả những thông tin không thể hình thức hóa như các yêu cầu phi chức năng và các mô tả khác.

d. Các giai đoạn thiết kế

Trong giai đoạn thiết kế, ta có thể nhìn nhận tiến trình thiết kế phần mềm theo nhiều khía cạnh khác nhau: theo người quản lý, theo nhà phát triển, theo mức độ hình thức hóa. Đứng trên phương diện quản lý, người ta chia thiết kế thành hai giai đoạn: thiết kế sơ bộ và thiết kế chi tiết. Thiết kế sơ bộ là thiết kế kiến trúc tổng thể. Giai đoạn tiếp theo là thiết kế chi tiết với hai giai đoạn sau:

- ***Thiết kế logic:*** Xác định cấu trúc thiết kế logic mô tả các thành phần của hệ thống và các mối quan hệ giữa chúng mà không gắn với bất kỳ phương tiện vật lý nào. Nhờ loại bỏ các

yếu tố vật lý mà nhà thiết kế có thể vận dụng các nguyên tắc thiết kế và sự sáng tạo của mình để vẽ lên một hệ thống phần mềm lý tưởng đáp ứng các yêu cầu đặt ra.

- ***Thiết kế vật lý:*** Lựa chọn các giải pháp công nghệ hiện có để thực hiện các cấu trúc logic đã cho một cách phù hợp với điều kiện của môi trường dự kiến của hệ thống phần mềm Giai đoạn này đòi hỏi nhà thiết kế phải có khả năng lựa chọn các giải pháp kỹ thuật và các phương tiện thích hợp trong điều kiện hiện hữu để thực hiện các chức năng của hệ thống nhằm đáp ứng tốt nhất yêu cầu người dùng.

6.1.3. Các chiến lược và phương pháp thiết kế

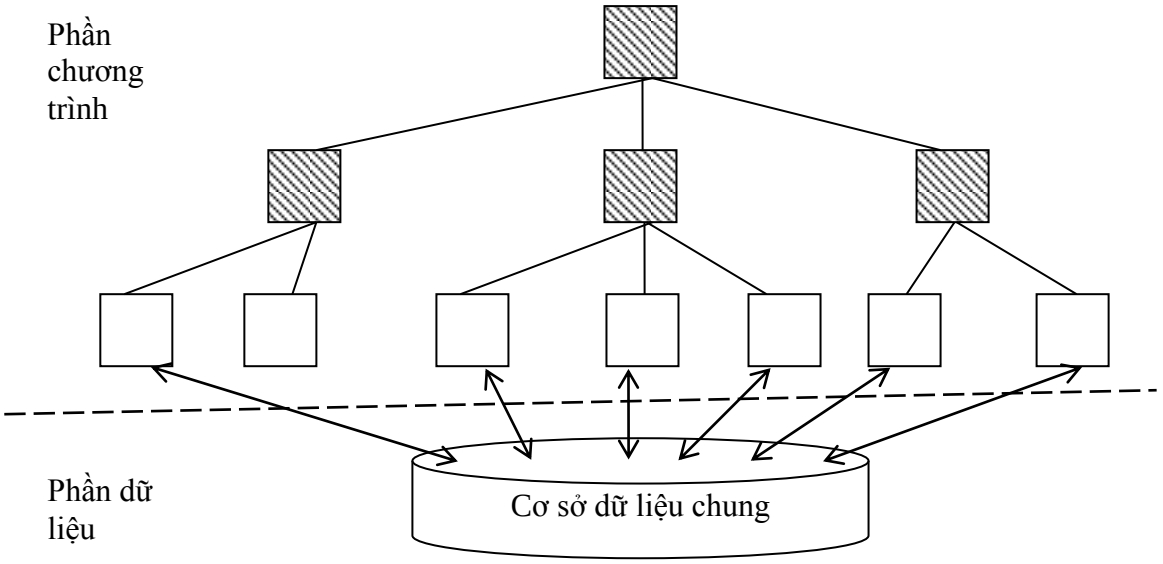
Hiện nay có rất nhiều phương pháp tiếp cận khác nhau được áp dụng cho thiết kế. Các tiếp cận này giúp cho quá trình thiết kế trở nên rõ ràng hơn, có thể theo dõi được và mang tính khoa học hơn. Trong đó, có 2 cách tiếp cận phổ biến là thiết kế hướng chức năng (cấu trúc) và thiết kế hướng đối tượng. Tương ứng với từng cách tiếp cận này là các chiến lược cho việc phát triển một hệ thống phần mềm. Với mỗi chiến lược, do đặc trưng của cách tiếp cận và đặc thù của từng hệ thống phần mềm mà các phương pháp được sử dụng rất khác nhau. Tuy nhiên, cũng có những phương pháp kết hợp cả hai chiến lược thiết kế này. Chẳng hạn như các phương pháp thiết kế giao diện, máy trạng thái hữu hạn…

a. Thiết kế hướng chức năng

Theo cách tiếp cận này, hệ thống được phân chia thành các chức năng, bắt đầu ở mức cao nhất, sau đó làm mịn dần để thành thiết kế với các chức năng chi tiết hơn. Trạng thái của hệ thống thể hiện qua cơ sở dữ liệu chung và được chia sẻ cho các chức năng thao tác trên nó (hình 6.3).

Trong phương pháp hướng cấu trúc (hướng chức năng), phần mềm được thiết kế dựa trên một trong hai hướng: hướng dữ liệu và hướng hành động.

- *Cách tiếp cận hướng dữ liệu* xây dựng phần mềm dựa trên việc phân rã theo các chức năng cần đáp ứng và dữ liệu cho các chức năng đó. Cách tiếp cận hướng dữ liệu sẽ giúp cho những người phát triển hệ thống dễ dàng xây dựng ngân hàng dữ liệu.
- *Cách tiếp cận hướng hành động* lại tập trung phân tích hệ thống phần mềm dựa trên các hoạt động thực thi các chức năng của phần mềm đó.



Hình 6.3. Mô hình hệ thống hướng cấu trúc

Cách thức thực hiện của phương pháp hướng cấu trúc là ***phương pháp thiết kế từ trên xuống (top-down).*** Phương pháp này tiến hành phân rã bài toán thành các bài toán nhỏ hơn, rồi tiếp tục phân rã các bài toán con cho đến khi nhận được các bài toán có thể cài đặt được ngay, sử dụng các hàm của ngôn ngữ lập trình hướng cấu trúc. Phương pháp hướng cấu trúc có ưu điểm là tư duy phân tích thiết kế rõ ràng, chương trình sáng sủa dễ hiểu. Tuy nhiên, phương pháp này có một số nhược điểm sau:

- *Không hỗ trợ việc tái sử dụng.* Các chương trình hướng cấu trúc phụ thuộc chặt chẽ vào cấu trúc dữ liệu và bài toán cụ thể, do đó không thể dùng lại một module nào đó trong phần mềm này cho phần mềm mới với các yêu cầu về dữ liệu khác.

- *Không phù hợp để phát triển các phần mềm lớn.* Nếu hệ thống thông tin lớn, việc phân rã thành các bài toán con cũng như phân các bài toán con thành các module và quản lý mối quan hệ giữa các module đó sẽ không dễ dàng, dễ gây ra các lỗi trong phân tích thiết kế hệ thống cũng như khó kiểm thử và bảo trì.

b. Thiết kế hướng đối tượng

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

Mặc dù mới phát triển từ những năm 90 của thế kỷ XX nhưng đã có rất nhiều phương pháp phát triển hướng đối tượng ra đời. Một ngôn ngữ chuẩn như UML và nhiều công cụ mạnh (Jbuilder, Rational Rose) đã được phát triển để trợ giúp cho phương pháp này.

6.1.4. Chất lượng thiết kế và các giải pháp đảm bảo chất lượng

Như đã đề cập ở trên, rất khó để xác định được thế nào là thiết kế tốt, điều này phụ thuộc vào ứng dụng và vào yêu cầu dự án. Một thiết kế tốt phải là một thiết kế mà nó cho phép sản sinh ra mã nguồn hữu hiệu; nó có thể là một thiết kế tối thiểu mà theo đó việc thực hiện càng chặt chẽ càng tốt; hoặc nó là thiết kế bảo dưỡng được tốt nhất hay chỉ là tiêu chuẩn tốt cho người dùng. Một thiết kế bảo dưỡng tốt có thể thích nghi với việc thay đổi hoặc thêm các chức năng mới.

Để đảm bảo điều đó, thiết kế phải là dễ hiểu và việc sửa đổi chỉ có hiệu ứng cục bộ. Các thành phần của thiết kế được kết dính theo nghĩa là tất cả các phần trong thành phần đó phải có mối quan hệ logic chặt chẽ. Các thành phần ấy phải được nối ghép lỏng lẻo. Chính sự nối ghép này là một độ đo tính độc lập của các thành phần. Nối ghép càng lỏng lẻo thì càng dễ thích nghi.

a. Sự kết dính

Sự kết dính của một thành phần là độ đo về tính gắn kết chặt chẽ với nhau giữa các bộ phận của nó. Một thành phần thực hiện một chức năng logic hoặc thực hiện một thực thể logic,

tất cả các bộ phận của nó đều phải tham gia vào việc thực hiện này. Nếu một thành phần không trực tiếp tham gia vào chức năng logic đó thì mức độ kết dính của nó là thấp.

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

- *Kết dính gom góp (kết dính ngẫu nhiên):* Các thành phần trong một module được nhóm lại với nhau một cách ngẫu nhiên, không có tính logic (ví dụ: module bao gồm tất cả các chức năng; kho chứa dữ liệu được sử dụng bởi các thành phần khác nhau).
- *Kết dính hội hợp logic:* Các thành phần cùng thực hiện các chức năng tương tự chẳng hạn như vào/ra dữ liệu, xử lý lỗi… được đặt vào cùng một thành phần.
- *Kết dính theo thời điểm:* Tất cả các thành phần cùng kích hoạt một lúc, chẳng hạn như khởi sự và kết thúc được bó lại với nhau.
- *Kết dính thủ tục:* Các phần tử trong thành phần được ghép lại trong một dãy điều khiển.
- *Kết dính truyền thông:* Tất cả các phần tử của thành phần cùng thao tác trên một dữ liệu vào và đưa cùng một dữ liệu ra.
- *Kết dính tuần tự:* Trong một thành phần, đầu ra của phần tử này là đầu vào của phần tử khác.
- *Kết dính chức năng:* Mỗi phần của thành phần đều cần thiết để thi hành cùng một chức năng nào đó.

Trong bảy mức độ kết dính nói trên, chỉ có mức độ kết dính cuối cùng là được khuyến khích sử dụng vì nhóm các thành phần được sử dụng cho một chức năng của hệ thống, mỗi thay đổi trong thành phần này kéo theo việc thay đổi và kiểm định lại các module thuộc các chức năng khác có liên quan.

b. Sự ghép nối

Sự ghép nối (coupling) chỉ ra độ ghép nối giữa các đơn vị của chương trình. Hệ thống có ghép nối cao sẽ có độ ghép nối mạnh giữa các đơn vị, nói cách khác là các đơn vị phụ thuộc lẫn nhau. Hệ thống nối ghép lỏng lẻo làm cho các đơn vị là độc lập hoặc tương đối độc lập với nhau.

Sự ghép nối của một thành phần với các thành phần khác thể hiện bằng số lượng mối liên kết phụ thuộc giữa nó với các thành phần khác. Các module là được ghép nối chặt chẽ nếu chúng dùng các biến chung và nếu chúng trao đổi các thông tin điều khiển (ghép nối chung nhau và ghép nối điều khiển). Ghép nối lỏng lẻo đạt được khi các thông tin biểu diễn được giữ trong thành phần này và chỉ giao tiếp với các thành phần khác thông qua danh sách tham số. Việc tối thiểu hóa sự ghép nối không phải lúc nào cũng thực hiện được. Dưới đây là các loại ghép nối được trình bày theo trình tự từ tốt đến xấu:

- *Không có liên kết trực tiếp:* Các module không có mối liên hệ trực tiếp với nhau, không có sự trao đổi thông tin. Việc nâng cấp module này không ảnh hưởng tới module khác.

- *Ghép nối dữ liệu:* Hai module chỉ trao đổi các dữ liệu đơn giản (không có cấu trúc) và thông qua giao diện. Ví dụ: một module gọi module khác thông qua các tham trị. Trong trường hợp này, việc thay đổi một module không ảnh hưởng tới module kia, kể cả giao diện.

- *Ghép nối nhãn:* Các module trao đổi dữ liệu dạng có cấu trúc thông qua giao diện (interface). Ví dụ trường hợp truyền các tham số. Trong trường hợp này, các module có thể thay đổi cấu trúc dữ liệu (thông qua việc khai báo kiểu). Khi thực hiện chương trình, nó có thể lưu các tham chiếu (con trỏ, địa chỉ đối tượng).

- *Ghép nối điều khiển:* Giao diện cho phép chi phối cách thực hiện bên trong module. Trong trường hợp này, các module đưa ra các thông tin có liên quan đến module khác, hạn chế việc nâng cấp sửa đổi các module.

- *Ghép nối mở rộng:* Hai module trao đổi thông tin với nhau bởi các phần tử trung gian của môi trường bên ngoài ứng dụng. Ví dụ: Hai ứng dụng trong hệ quản trị của trường học là đăng ký học và tính toán kết quả cần trao đổi thông tin qua người sử dụng trung gian (giáo viên, điểm thi). Trong trường hợp này, kênh trao đổi thông tin không được xác định, nó có thể bị bỏ sót trong quá trình nâng cấp.

- *Ghép nối chung:* Hai module dùng chung các biến toàn cục. Ví dụ như việc trao đổi thông tin qua các biến toàn cục bằng cấu trúc COMMON trong ngôn ngữ Fortran, biến Public trong Visual Basic. Càng nhiều biến toàn cục được sử dụng chung, càng khó nâng cấp cấu trúc.

- *Ghép nối nội dung:* Một module biết và mở rộng nội dung của một module khác (truy cập vào các biến private, vào cấu trúc logic…). Ví dụ: Sử dụng các giá trị hằng số; khai thác thông tin không được công khai qua dữ liệu của giao diện (một module sử dụng kết quả của việc sắp xếp dữ liệu trong module khác). Trong trường hợp này, khi nội dung của module được triển khai, không thể mở rộng hoặc nâng cấp module.

c. Tính hiểu được

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

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

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

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

- *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 và một cấu trúc logic phức tạp mà nó liên quan đến độ sâu lồng nhau của phát biểu. Các thành phần phức tạp là khó hiểu, vì thế người thiết kế nên làm cho thiết kế thành phần càng đơn giản càng tốt.

Đa số công việc về đo chất lượng thiết kế được tập trung vào việc 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 cách mô tả thiết kế. Các số đo độ phức tạp có thể chỉ cung cấp một chỉ số cho độ dễ hiểu của một thành phần.

Sự thừa kế trong một thiết kế hướng đối tượng phản ánh độ dễ hiểu. Nếu sự thừa kế được dùng để gắn các chi tiết thiết kế thì thiết kế sẽ dễ hiểu hơn. Mặt khác, nếu sử dụng sự thừa kế đòi

hỏi người đọc thiết kế phải nhìn nhiều lớp đối tượng khác nhau trong cây thừa kế thì độ dễ hiểu của thiết kế là được rút gọn.

d. Sự thích nghi được

Nếu một thiết kế nhằm tăng tính bảo trì thì nó phải sẵn sàng thích nghi được, nghĩa là các thành phần trong hệ thống được ghép nối với nhau một cách lỏng lẻo để khi có thay đổi hoặc bổ sung các thành phần mới thì các thành phần còn lại ít bị can thiệp nhất.

Bên cạnh đó, cũng phải soạn thảo tốt tài liệu thiết kế, dễ hiểu và đồng nhất với việc xây dựng phần mềm, có mối quan hệ rõ ràng giữa các mức khác nhau của thiết kế. Người đọc bản thiết kế có thể tìm được các biểu diễn liên quan trong lược đồ cấu trúc hay có thể thấy được sự biến đổi của dòng dữ liệu.

Cần phải luôn luôn kết hợp chặt chẽ những thay đổi thiết kế trong toàn bộ tư liệu thiết kế, nếu không tư liệu thiết kế đó có thể trở nên không nhất quán và những thay đổi tiếp sau là khó thực hiện.

e. Một số hướng dẫn thiết kế

- Linh hoạt đối với những yêu cầu thay đổi không định trước.

- Dễ thử nghiệm.

- Sáng sủa, dễ đọc.

- Kích thước module nhỏ.

- Tính độc lập module (tính mở/đóng giữa các module).

- Phải có mối quan hệ chặt chẽ giữa thiết kế và yêu cầu.

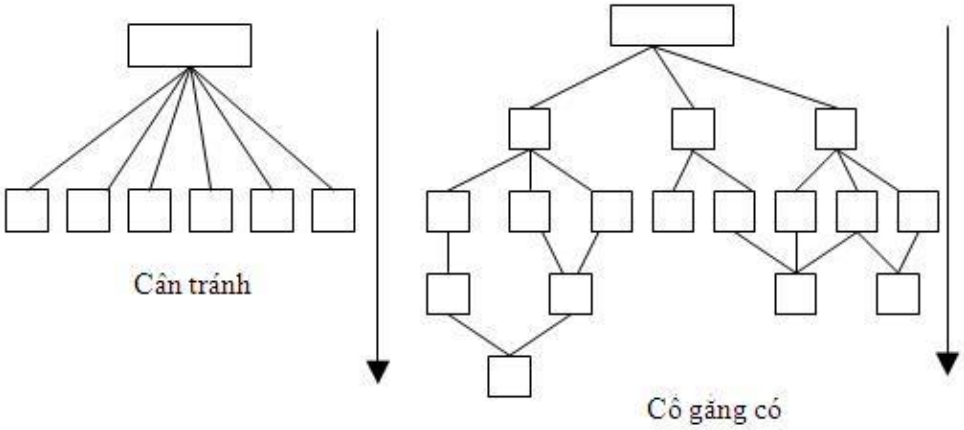
- Mỗi module hoàn toàn độc lập, thực hiện một chức năng duy nhất và thực hiện trọn vẹn chức năng đó.

- Mọi thứ trong module ràng buộc với nhau qua việc xử lý nối tiếp nhau trên cùng một dòng dữ liệu.

- Mọi thứ trong module được điều khiển bởi cùng một dữ liệu vào, hay cùng một phức hợp thiết bị, hay cùng thực hiện từng phần của cùng một kết xuất.

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

- Khi thiết kế cố gắng giảm thiểu cấu trúc bằng việc tản ra nhiều ở độ sâu thấp và cố gắng co cụm khi chiều sâu tăng thêm (hình 6.4).

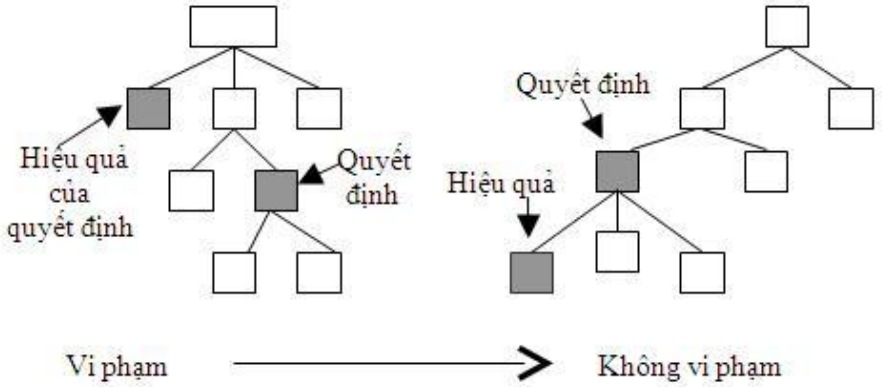


Hình 6.4. Hướng dẫn thiết kế tránh chia nhỏ module và cố gắng co cụm khi tăng chiều sâu

- Giữ phạm vi hiệu quả của một module bên trong phạm vi kiểm soát của module đó:

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

+ Phạm vi kiểm soát của module m là tất cả các module thuộc cấp của m (tính tới mức dưới cùng).



Hình 6.5. Phạm vi hiệu quả trong việc kiểm soát module

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

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

- Cố gắng giữ các module một đầu vào và một đầu ra, tránh các "mối nối mang khả năng gây lỗi".

6.2. THIẾT KẾ KIẾN TRÚC

6.2.1. Khái niệm – tầm quan trọng của thiết kế kiến trúc

a. Khái niệm

Kiến trúc phần mềm (software architecture) là một cấu trúc tổng thể của phần mềm, qua đó cung cấp sự tích hợp về mặt khái niệm của một hệ thống. Hiểu một cách đơn giản, kiến trúc là cấu trúc phân cấp của các thành phần chương trình, qua đó thể hiện sự tương tác giữa chúng với nhau và với cấu trúc dữ liệu mà chúng sử dụng. Theo nghĩa rộng, kiến trúc biểu diễn các thành phần lớn, cốt lõi của hệ thống và mối quan hệ giữa chúng với nhau được nhìn theo những quan điểm khác nhau.

b. Vai trò và tầm quan trọng của kiến trúc

Bass (Software Engineering, 8th Edition) và các đồng nghiệp đã thảo luận về những ưu điểm của việc thiết kế kiến trúc một cách rõ ràng và việc tài liệu hóa kiến trúc phần mềm:

- *Là công cụ giao tiếp giữa những người có liên quan (Stackeholder communication):* Kiến trúc là một cách giới thiệu hệ thống ở mức cao, có thể sử dụng nó như một tiêu điểm thảo luận bởi các nhóm khác nhau có liên quan trong dự án.

- *Phân tích hệ thống (System Analysis):* Tạo ra một kiến trúc hệ thống rõ ràng ở giai đoạn đầu của quá trình phát triển hệ thống. Quyết định thiết kế hệ thống sẽ giúp ta đáp ứng tốt hơn những yêu cầu phi chức năng của hệ thống.

- *Tái sử dụng ở quy mô lớn (Large-scale reuse):* một mô hình kiến trúc hệ thống là một bản mô tả chắc chắn, dễ sử dụng về cách tổ chức hệ thống và sự tác động qua lại giữa các thành phần. Kiến trúc hệ thống của các hệ thống có yêu cầu tương tự thường giống nhau, vì vậy chúng ta có thể sử dụng lại phần lớn kiến trúc này.

Hofmeister (Software Engineering, 8th Edition) và các đồng nghiệp đã thảo luận về việc thiết kế kiến trúc trong giai đoạn thiết kế phần mềm để xem xét những khía cạnh thiết kế quan trọng trong giai đoạn đầu của tiến trình. Họ gợi ý rằng kiến trúc phần mềm có thể được sử dụng như một bản kế hoạch dự án, có thể dùng để đàm phán về những yêu cầu hệ thống và có thể dùng nó để thảo luận với khách hàng, những người phát triển hệ thống và các nhà quản lý. Họ cũng gợi ý rằng nó là một công cụ thiết yếu cho công việc quản lý phức tạp, bỏ qua sự chi tiết hóa và cho phép người thiết kế tập trung vào hệ thống ở mức trìu tượng.

c. Kiến trúc và đặc điểm của hệ thống

Kiến trúc hệ thống ảnh hưởng tới sự thực thi, tính hiệu quả, tính phân tán và bảo trì của hệ thống. Kiểu và cấu trúc được lựa chọn cho một ứng dụng cũng có thể phụ thuộc vào những yêu cầu phi chức năng của hệ thống:

- *Tính hiệu năng (Performance):* Nếu hiệu năng là một yêu cầu quan trọng đối với hệ thống thì kiến trúc nên được thiết kế để tập trung vào những chức năng quan trọng với một số lượng nhỏ những hệ thống con, giảm thiểu các mối liên lạc trao đổi thông tin có thể giữa những hệ thống con này. Điều này có nghĩa là nên sử dụng những thành phần lớn (large_grain) hơn là những thành phần nhỏ (fine_grain) để giảm sự tương tác qua lại giữa các thành phần.

- *Tính bảo mật (security):* Nếu bảo mật là một yêu cầu quan trọng, nên sử dụng kiến trúc phân tầng, như vậy những thông tin quan trọng nhất sẽ được bảo vệ trong tầng sâu nhất và việc kiểm tra tính an toàn phải được tiến hành một cách chắc chắn đối với những tầng này.

- *Tính an toàn (safety):* Nếu an toàn là một yêu cầu quan trọng, hệ thống nên được thiết kế sao cho những chức năng đòi hỏi tính an toàn cao nằm tập trung trong một hệ thống con hoặc nằm trong một số ít những hệ thống con. Việc này sẽ giảm được chi phí và những vấn đề phát sinh trong việc thẩm định (validation) tính an toàn, đồng thời có thể giảm được chi phí để mua và bảo trì các công cụ bảo vệ có liên quan.

- *Tính sẵn sàng (availability):* Nếu tính sẵn sàng là một yêu cầu quan trọng, kiến trúc nên được thiết kế có nhiều thành phần dư thừa để nó có thể thay thế và cập nhật các thành phần mà không cần phải dừng hệ thống.

- *Tính bảo trì (maintainability):* Nếu tính bảo trì là một yêu cầu quan trọng, kiến trúc hệ thống nên được thiết kế bằng việc sử dụng những thành phần nhỏ (fine_grain), những thành phần chứa chính nó (self-contained) có thể dễ dàng thay đổi. Nên sử dụng phương án chia dữ liệu theo khách hàng, chứ không nên phân chia theo cấu trúc dữ liệu.

Có thể nhận thấy sự xung đột trong các kiến trúc trên một cách rõ ràng. Ví dụ: sử dụng những thành phần kích thước lớn để tăng hiệu suất nhưng giảm tính bảo trì; đưa ra những dữ liệu dư thừa (dự trữ) để nâng cao tính sẵn sàng nhưng lại làm cho tính bảo mật khó kiểm soát hơn; tập trung vào những tính năng liên quan đến độ an toàn có nghĩa là tăng nhu cầu trao đổi thông tin, vì thế làm giảm hiệu suất của hệ thống. Nếu hai yếu tố mâu thuẫn nhau đều quan trọng với hệ thống thì cần phải có sự thỏa thuận để tìm ra giải pháp chung.

6.2.2. Các quyết định thiết kế kiến trúc

Thiết kế kiến trúc là một công việc sáng tạo, đòi hỏi phải đưa ra được một cách tổ chức hệ thống có thể đảm bảo được những yêu cầu chức năng và phi chức năng. Bởi nó là một tiến trình sáng tạo, các hoạt động bên trong tiến trình này hoàn toàn khác nhau, phụ thuộc vào kiến trúc hệ thống và những yêu cầu đặc biệt của hệ thống.

Trong quá trình thiết kế kiến trúc, người làm thiết kế kiến trúc thực hiện những quyết định cơ bản và có ảnh hưởng sâu sắc tới hệ thống cũng như tiến trình phát triển. Dựa trên kiến thức và kinh nghiệm, họ phải trả lời những câu hỏi cơ bản sau:

- Có kiến trúc ứng dụng chung có thể sử dụng không?
- Hệ thống sẽ được phân tán như thế nào?
- Kiểu kiến trúc nào là phù hợp?
- Cách tiếp cận nào sẽ được sử dụng để cấu trúc hệ thống?
- Hệ thống được phân chia thành các module như thế nào?
- Chiến lược kiểm soát nào nên được sử dụng?
- Thiết kế kiến trúc sẽ được đánh giá như thế nào?
- Kiến trúc có được chuyển thành tài liệu không?

a. Tái sử dụng các mẫu kiến trúc

Mặc dù mỗi hệ thống phần mềm là duy nhất, các hệ thống trong cùng một lĩnh vực cũng có những kiến trúc tương tự phản ánh những khái niệm cơ bản của lĩnh vực ứng dụng. Các kiến trúc của lĩnh vực ứng dụng này có thể có đặc điểm chung rõ ràng, chẳng hạn như kiến trúc của các hệ thống quản lý thông tin được xây dựng quanh một lõi kiến trúc với sự biến đổi cho phù hợp với yêu cầu của từng khách hàng. Khi thiết kế kiến trúc hệ thống, bạn phải quyết định hệ thống của bạn là gì, những lớp ứng dụng nào có thể dùng chung và quyết định xem bao nhiêu phần trăm kiến thức từ kiến trúc của ứng dụng này có thể sử dụng lại được.

Đối với những hệ thống nhúng và các hệ thống được thiết kế cho máy tính cá nhân, thông thường chỉ chạy trên một bộ vi xử lý, không sử dụng kiến trúc phân tán. Tuy nhiên, hầu hết các hệ thống lớn hiện nay là các hệ thống phân tán, được xử lý trên nhiều máy tính khác nhau. Việc lựa chọn kiến trúc phân tán là một quyết định then chốt ảnh hưởng tới hiệu suất và độ tin cậy của hệ thống.

Kiến trúc của một hệ thống phần mềm có thể dựa trên một mô hình hoặc kiểu kiến trúc riêng biệt. Một kiểu kiến trúc là một mẫu (pattern) của tổ chức hệ thống. Kiến thức, khả năng ứng dụng, điểm mạnh và điểm yếu của những mẫu này rất quan trọng. Tuy nhiên, kiến trúc của hầu hết những hệ thống lớn không tương thích với một kiểu kiến trúc riêng lẻ nào. Những thành phần khác nhau của hệ thống có thể được thiết kế bởi các kiểu kiến trúc khác nhau. Trong một số trường hợp, kiến trúc của toàn bộ hệ thống được tạo ra bởi sự kết hợp các kiểu kiến trúc khác nhau.

b. Phát triển và sử dụng các mô hình kiến trúc

Sản phẩm của tiến trình thiết kế kiến trúc là tài liệu thiết kế kiến trúc. Tài liệu này có thể bao gồm các sơ đồ giới thiệu về hệ thống kết hợp với những mô tả bằng các đoạn văn bản. Tài liệu này nên mô tả hệ thống được cấu trúc thành các hệ thống con như thế nào, mỗi hệ thống con lại được phân chia thành các module như thế nào. Mô hình đồ họa của hệ thống đưa ra những cái nhìn khác nhau về kiến trúc. Các kiểu mô hình được phát triển trong tiến trình thiết kế kiến trúc có thể bao gồm:

- *Mô hình cấu trúc tĩnh* chỉ ra các thành phần cơ bản của hệ thống. Được phát triển thành các đơn vị riêng biệt.
- *Mô hình tiến trình động* chỉ ra cấu trúc của các tiến trình trong hệ thống. Mô hình này có thể khác với mô hình tĩnh.
- *Mô hình giao diện* xác định các dịch vụ được cung cấp bởi các hệ thống con thông qua giao diện công khai.
- *Mô hình quan hệ* như mô hình luồng dữ liệu data-flow chỉ ra mối quan hệ giữa các hệ thống con.
- *Mô hình phân tán* chỉ ra các hệ thống con được phân chia như thế nào trong các hệ thống máy tính.

Có một số ngôn ngữ và công cụ hỗ trợ để phục vụ cho quá trình thiết kế, chẳng hạn ngôn ngữ ADLs. Các thành phần cơ bản của ADLs là: components, connectors, các quy tắc và hướng dẫn cho việc thiết kế kiến trúc. Tuy nhiên, giống như tất cả những ngôn ngữ chuyên dụng khác,

ADLs chỉ có thể được hiểu bởi các chuyên gia và khó sử dụng đối với những chuyên gia của lĩnh vực ứng dụng (ví dụ các chuyên gia trong lĩnh vực tài chính). Điều này gây khó khăn trong việc phân tích. Vì thế, ta có thể sử dụng các ngôn ngữ thông dụng hơn, chẳng hạn như UML cũng có thể dùng để mô tả kiến trúc hệ thống.

6.2.3. Tổ chức hệ thống

Tổ chức hệ thống phản ánh chiến lược cơ bản được sử dụng để cấu trúc hệ thống. Ta phải thực hiện các quyết định trên mô hình tổ chức tổng quát của hệ thống ở giai đoạn đầu của tiến trình thiết kế. Việc tổ chức hệ thống có thể được phản ánh trực tiếp thông qua cách cấu trúc các hệ thống con. Tuy nhiên, các mô hình hệ thống con thường bao gồm những thông tin chi tiết hơn về mô hình tổ chức và không phải lúc nào cũng có mối liên hệ đơn giản giữa các hệ thống con và tổ chức hệ thống. Trong phần này, chúng ta sẽ cùng thảo luận về ba kiểu tổ chức phổ biến nhất:

- Mô hình kho dữ liệu;
- Mô hình máy khách/máy chủ;
- Mô hình máy ảo hoặc phân tầng.

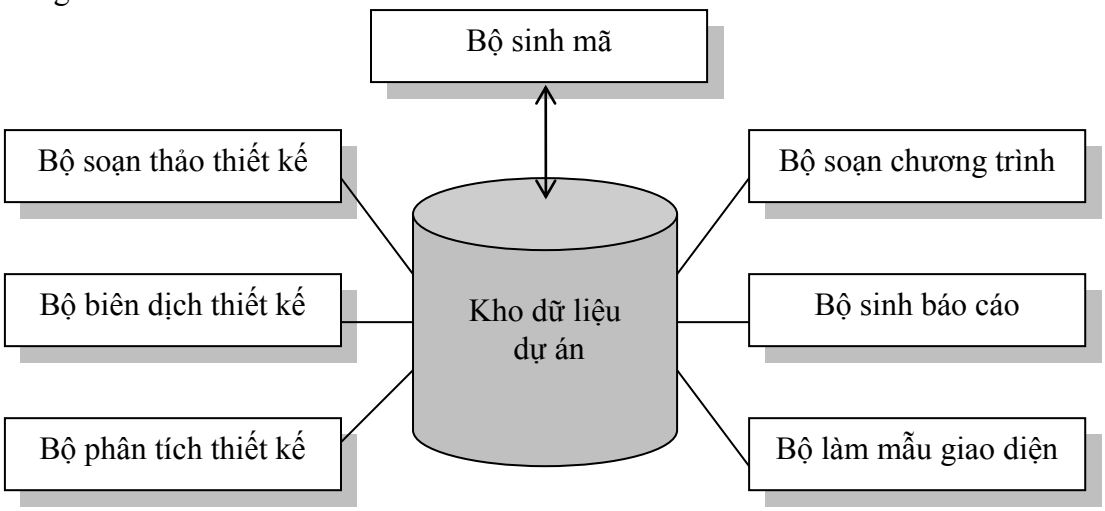
Trong một hệ thống, ta có thể sử dụng ba kiểu tổ chức này một cách riêng rẽ hoặc phối hợp chúng với nhau.

a. Mô hình kho dữ liệu

Các hệ thống con tạo thành một hệ thống phải trao đổi thông tin, vì thế chúng có thể làm việc với nhau một cách hiệu quả. Có hai cách cơ bản để thực hiện điều này:

- Dữ liệu được chia sẻ nằm trong một kho hoặc một cơ sở dữ liệu tập trung và có thể được truy nhập bởi tất cả các hệ thống con. Một mô hình hệ thống dựa trên một cơ sở dữ liệu được chia sẻ đôi khi được gọi là mô hình kho dữ liệu (repository model).
- Mỗi hệ thống con có một cơ sở dữ liệu riêng. Dữ liệu được trao đổi với các hệ thống con khác bằng việc gửi các thông điệp tới chúng.

Phần lớn các hệ thống sử dụng một lượng rất lớn dữ liệu được tổ chức xung quanh một cơ sở dữ liệu được chia sẻ hoặc kho dữ liệu. Tuy nhiên, mô hình này thích hợp với các ứng dụng được sinh ra bởi một hệ thống con và được sử dụng bởi một hệ thống khác. Ví dụ như các hệ thống bao gồm các câu lệnh và các hệ thống điều khiển, các hệ thống quản lý thông tin, hệ thống CAM/CAD và các bộ công cụ CASE. Hình 6.6 là một ví dụ về hệ thống sử dụng mô hình kho dữ liệu tập trung.



Hình 6.6. Kiến trúc của bộ công cụ CASE tích hợp

- ***Ưu điểm***

- Hiệu quả khi có nhu cầu chia sẻ dữ liệu lớn. Không cần phải trao đổi dữ liệu từ hệ thống này tới hệ thống khác.
- Các hoạt động như sao lưu dữ liệu, đảm bảo an ninh, kiểm soát sự truy cập và tìm ra lỗi được xử lý tập trung. Đó là trách nhiệm của người quản lý kho dữ liệu. Các công cụ có thể tập trung vào những chức năng chính của nó chứ không cần quan tâm đến những vấn đề này.
- Mô hình chia sẻ được đưa ra như một sơ đồ các kho. Điều này có thể đơn giản hóa việc tích hợp thêm các công cụ mới có cùng mô hình dữ liệu.

- ***Nhược điểm***

- Các hệ thống con phải chấp thuận mô hình kho dữ liệu; và một điều không thể tránh khỏi, đó là việc thỏa hiệp giữa các nhu cầu đặc biệt của từng công cụ. Hiệu năng của hệ thống có thể sẽ bị những tác động tiêu cực bởi sự thỏa hiệp này. Điều đó có thể rất khó hoặc không thể tích hợp thêm các hệ thống con mới nếu các mô hình dữ liệu của nó không thích hợp với lược đồ đã được xác nhận.
- Việc phát triển có thể sẽ rất khó khăn khi một khối lượng lớn thông tin được sinh ra theo một mô hình dữ liệu đã được chấp thuận. Việc biến đổi những thông tin này sang một mô hình dữ liệu mới chắc chắn là sẽ rất tốn kém, khó và đôi khi là không thực hiện được.
- Các hệ thống con khác nhau có thể có những yêu cầu khác nhau về các chính sách bảo mật, kiểm soát lỗi và backup dữ liệu. Mô hình kho dữ liệu bắt buộc các hệ thống con phải thực hiện cùng một cơ chế quản lý như nhau.

- Khó để phân chia một cách hiệu quả kho dữ liệu qua một hệ thống máy tính. Mặc dù ta có thể phân chia kho dữ liệu tập trung một cách logic, cũng có rất nhiều vấn đề về sự xung đột và dư thừa dữ liệu.

b. Mô hình máy khách/chủ (client/server)

Mô hình kiến trúc Client-Server là một mô hình hệ thống được tổ chức như một tập các dịch vụ và kết nối giữa Servers với Clients để có thể truy cập và sử dụng các dịch vụ. Các thành phần chính của mô hình này là:

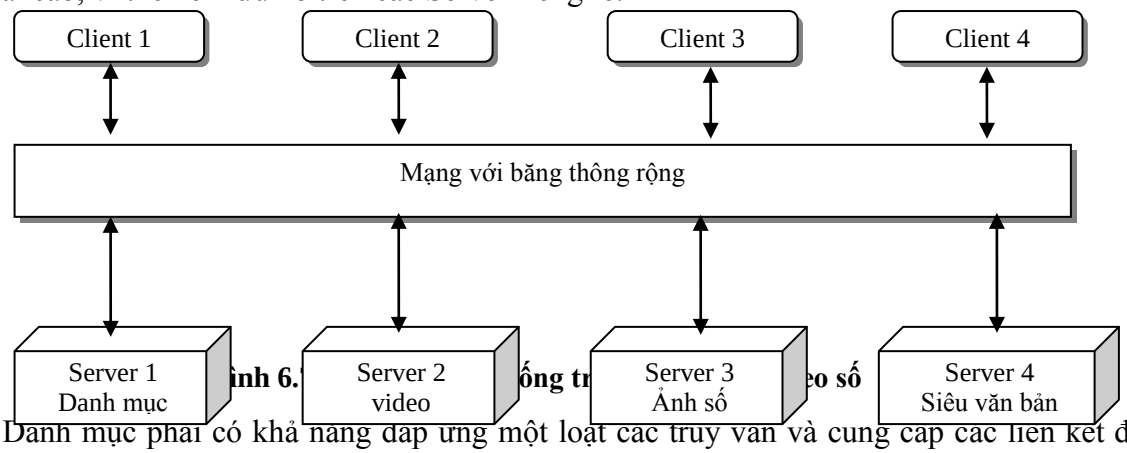
- *Một nhóm các máy chủ (Servers)* cung cấp các dịch vụ cho các hệ thống con. Ví dụ như các máy chủ chuyên thực hiện công việc in ấn. Các tệp cần in phải được quản lý bởi một dịch vụ quản lý tệp và một máy chủ biên soạn tài liệu được cung cấp bởi các dịch vụ biên soạn.

- *Một nhóm các máy trạm (Clients)* gọi các dịch vụ được cung cấp bởi các máy chủ. Có thể có một vài trường hợp một chương trình của hệ thống con được tiến hành song song.

- *Một mạng* cho phép các máy trạm truy nhập vào các dịch vụ này. Điều này giúp cho một hệ thống có thể có nhiều máy chủ. Trên thực tế, hầu hết các hệ thống Client-Server được thực hiện trên các hệ thống phân tán.

Máy trạm có thể biết tên của các máy chủ và các dịch vụ máy chủ này cung cấp. Tuy nhiên, các máy chủ không cần biết danh tính cũng như có bao nhiêu máy trạm tham gia vào hệ thống. Các máy trạm truy cập vào các dịch vụ được cung cấp bởi máy chủ thông qua một thủ tục gọi từ xa sử dụng giao thức request-reply giống như giao thức http sử dụng trong world wide web. Hiển nhiên, một máy trạm đưa ra một yêu cầu cho máy chủ và đợi cho đến khi nhận được câu trả lời.

Hình vẽ 6.7 là một ví dụ về một hệ thống được xây dựng dựa trên mô hình Client-Server. Hệ thống này có nhiều người dùng trên nền Web để cung cấp một thư viện hình ảnh và phim. Trong hệ thống này, một vài máy chủ quản lý và hiển thị các kiểu dữ liệu đa phương tiện (phim, nhạc, video, hình ảnh…). Các video cần truyền nhanh và đồng bộ nhưng có độ phân giải tương đối thấp. Chúng có thể được nén trong một kho CSDL, vì thế Server video có thể thực hiện việc nén và giải nén sang các định dạng khác nhau. Tuy nhiên, đối với ảnh thì phải giữ nguyên độ phân giải cao, vì thế nên lưu nó trên các Server riêng rẽ.



ình 6. ống tr eo số

Danh mục phải có khả năng đáp ứng một loạt các truy vấn và cung cấp các liên kết đến các hệ thống thông tin web bao gồm dữ liệu về film, video clip và các hệ thống thương mại điện tử cho phép tải về phim và các video clip. Các máy khách sử dụng dịch vụ mà máy chủ cung cấp thông qua thủ tục gọi từ xa (hình 6.7).

- ***Ưu điểm***

Ưu điểm của mô hình này là sự phân tán dữ liệu không phức tạp; tăng hiệu quả sử dụng của các hệ thống mạng; có thể đòi hỏi phần cứng rẻ hơn đồng thời dễ bổ sung thêm các máy chủ mới hoặc nâng cấp các máy chủ đang tồn tại mà hệ thống vẫn hoạt động bình thường.

- ***Nhược điểm***

Tuy nhiên, bên cạnh đó cũng tồn tại một số nhược điểm như: không được chia sẻ mô hình dữ liệu, vì thế các hệ thống con có thể sử dụng cách tổ chức dữ liệu khác nhau. Do đó việc trao đổi dữ liệu có thể kém hiệu quả; dư thừa việc quản lý trên mỗi máy chủ và nếu không có bộ phận lưu trữ tên và các dịch vụ thì khó tìm ra những máy chủ và các dịch vụ đang sẵn sàng hoạt động.

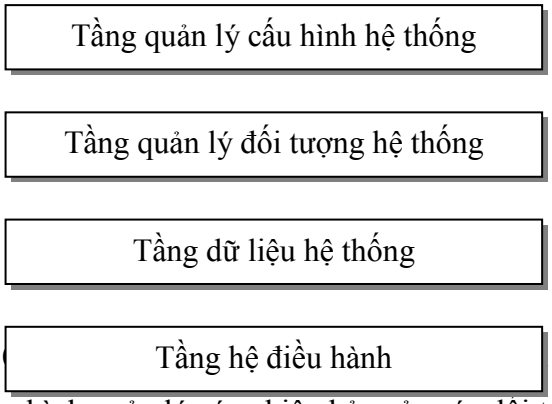
c. Mô hình máy ảo/phân tầng

Kiến trúc phân tầng (hay còn gọi là kiến trúc máy ảo) tổ chức hệ thống thành nhiều tầng, mỗi tầng cung cấp một nhóm các dịch vụ. Mỗi tầng có thể thông qua một máy ảo có ngôn ngữ máy được xác định bởi các dịch vụ được cung cấp bởi tầng này. Ngôn ngữ này được sử dụng để

thực hiện trên tầng tiếp theo của máy ảo. Bước dịch tiếp theo sẽ chuyển đổi mã của máy ảo này thành mã máy thực.

Một ví dụ về kiến trúc phân tầng chính là mô hình OSI của giao thức mạng hay mô hình 3 tầng của môi trường hỗ trợ lập trình Ada (APSE - ada programming support environnement).

Hình vẽ 6.8 phản ánh cấu trúc của APSE và chỉ ra làm thế nào một hệ thống quản lý cấu hình có thể được tích hợp sử dụng cách tiếp cận này.



Hình hình

Hệ thống quản lý cấu hình quản lý các phiên bản của các đối tượng và cung cấp cơ sở dữ liệu quản lý cấu hình tổng hợp. Để cung cấp cơ sở quản lý cấu hình tổng hợp này, nó sử dụng một hệ thống quản lý đối tượng cung cấp những thông tin lưu trữ và quản lý các dịch vụ cho các bản ghi cấu hình hoặc các đối tượng. Hệ thống này được xây dựng bên trên hệ thống cơ sở dữ liệu để cung cấp các kho dữ liệu cơ bản và các dịch vụ, chẳng hạn như quản lý các giao dịch, phục hồi cơ sở dữ liệu về trạng thái cũ (rolleback); phục hồi dữ liệu (recovery) và truy nhập điều khiển. Việc quản lý cơ sở dữ liệu sử dụng hệ điều hành cơ sở và các kho lưu trữ tài nguyên của dự án.

- ***Ưu điểm***

Cách tiếp cận phân tầng hỗ trợ phát triển tăng dần các hệ thống. Khi một tầng được phát triển, một số dịch vụ được cung cấp bởi tầng này có thể được người dùng sử dụng. Kiến trúc này cũng mang tính thay đổi và tính khả chuyển. Vì khi giao diện không đổi, một tầng có thể được thay thể bởi tầng khác, tương đương. Hơn nữa, khi các giao diện của tầng thay đổi hoặc có thêm những dịch vụ mới được bổ xung vào tầng này thì chỉ có tầng liền kề bị ảnh hưởng.

- ***Nhược điểm***

Điểm bất lợi của kiến trúc phân tầng là cấu trúc các hệ thống theo cách này rất khó. Các tầng bên trong có thể cung cấp các chức năng cơ bản, chẳng hạn như quản lý tệp, nhưng tất cả các tầng đều cần chức năng này. Các dịch vụ mà người sử dụng cần thường ở tầng trên cùng, có thể phải xuyên thủng tầng liền kề để truy nhập vào các dịch vụ được cung cấp bởi các tầng sâu hơn.

Hiệu năng của hệ thống cũng có thể là một vấn đề cần xem xét bởi vì đôi lúc có lệnh được thực hiện ở nhiều tầng. Nếu có quá nhiều tầng, một dịch vụ yêu cầu từ tầng trên cùng phải mất thời gian để đi qua các tầng trung gian trước khi được xử lý. Để tránh vấn đề này, các ứng dụng có thể phải trao đổi trực tiếp với các tầng khác hơn là sử dụng các dịch vụ được cung cấp bởi tầng liền kề.

6.2.4. Các mô hình điều khiển

Các mô hình cấu trúc hóa một hệ thống liên quan đến việc quyết định xem một hệ thống được phân chia thành các hệ thống con như thế nào. Để làm việc như một hệ thống, các hệ thống con phải điều khiển để các dịch vụ của nó truyền đến đúng địa điểm và đúng thời gian yêu cầu. Các mô hình cấu trúc không (và không nên) tích hợp thêm các thông tin điều khiển. Hơn nữa, người kiến trúc sư nên tổ chức các hệ thống con theo một vài mô hình điều khiển bổ xung vào mô hình cấu trúc đã được sử dụng. Các mô hình điều khiển ở tầm kiến trúc liên quan đến luồng dữ liệu giữa các hệ thống con. Có 2 kiểu điều khiển chung được sử dụng trong các hệ thống phần mềm:

Điều khiển tập trung: một hệ thống con chịu trách nhiệm điều khiển, khởi động và kết thúc một hoạt động của các hệ thống con khác. Nó cũng có thể ủy thác quyền điều khiển cho một hệ thống con khác nhưng sẽ vẫn chờ để nắm lại quyền kiểm soát.

Điều khiển dựa trên sự kiện: việc kiểm soát thông tin được nhúng trong một hệ thống con, mỗi hệ thống con có thể trả lời các sự kiện được sinh ra từ bên ngoài. Các sự kiện này có thể đến từ các hệ thống con khác hoặc từ môi trường chung của hệ thống.

Các kiểu điều khiển được sử dụng kết hợp với các kiểu cấu trúc. Tất cả các kiểu cấu trúc mà chúng ta đã thảo luận có thể liên quan đến điều khiển tập trung hoặc điều khiển theo sự kiện.

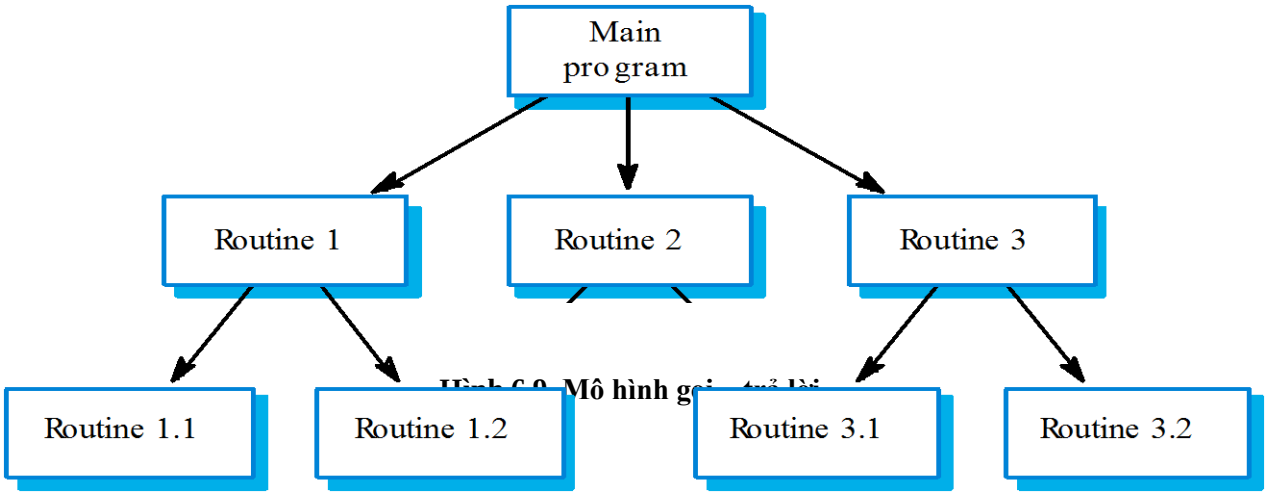
a. Điều khiểu tập trung

Trong mô hình điều khiển tập trung, một hệ thống con được thiết kế như một hệ thống điều khiển chịu trách nhiệm quản lý việc thực thi của các hệ thống con khác. Các mô hình điều khiển tập trung có thể chia thành hai lớp, phụ thuộc vào các hệ thống con được điều khiển một cách tuần tự hoặc song song.

- ***Mô hình gọi – trả lời***

Mô hình này tương tự như mô hình thủ tục con top-down, điều khiển bắt đầu ở trên đỉnh của một cây thứ bậc và thông qua việc gọi các thủ tục con để đi xuống mức thấp hơn trong cây. Mô hình thủ tục con này chỉ được ứng dụng trong các hệ thống tuần tự.

Mô hình gọi-trả lời được minh họa qua hình vẽ 6.9. Chương trình chính có thể gọi các thủ tục routines 1, 2 và 3, routine 1 có thể gọi các thủ tục 1.2 hoặc 1.1; routine 3 có thể gọi routine 3.1 hoặc 3.2…



Hình 6.9. Mô hình gọi...trả lời

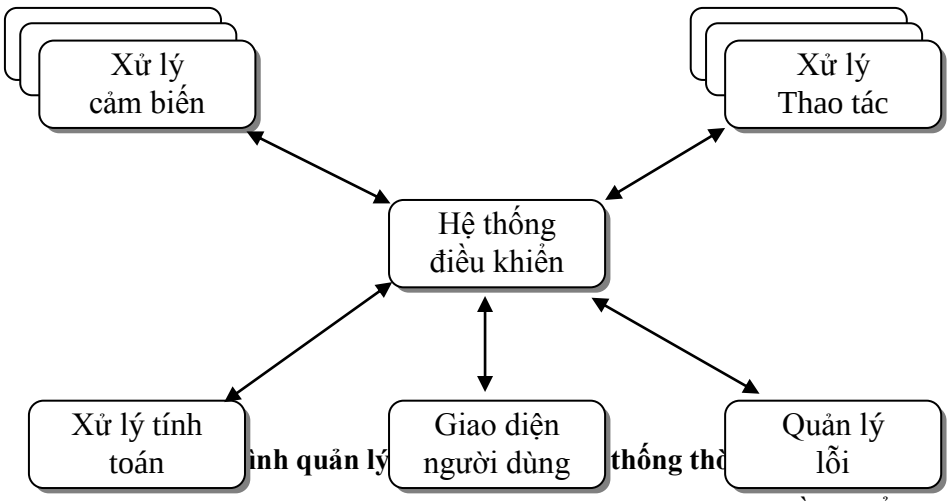
Mô hình tương tự này được nhúng trong các ngôn ngữ lập trình, chẳng hạn như ngôn ngữ C, Ada và Pascal. Điều khiển chuyển từ thủ tục ở mức cao trong cây xuống các thủ tục ở mức thấp. Sau đó nó trả lại con trỏ tại điểm mà thủ tục được gọi. Thủ tục đang được thực thi có trách nhiệm điều khiển và cũng có thể gọi các thủ tục con khác hoặc trả lại điều khiển cho cha của nó. Nó là một kiểu lập trình đơn giản trả về một vài điểm khác trong chương trình.

Mô hình gọi-trả lời có thể được sử dụng tại mức module để quản lý các chức năng hoặc các đối tượng. Các thủ tục con trong một ngôn ngữ lập trình được gọi bởi các thủ tục con khác là chức năng tự nhiên.

Tính chất cứng nhắc và bị giới hạn của mô hình này vừa là điểm mạnh vừa là điểm yếu của nó. Nó là điểm mạnh vì nó tương đối đơn giản để phân tích các luồng điều khiển và công việc, là một điểm yếu vì ngoài trừ các thao tác thông thường, nó rất khó để điều khiển.

- ***Mô hình quản lý***

Mô hình này được ứng dụng trong các hệ thống song song. Một thành phần hệ thống được thiết kế như một hệ thống quản lý, điều khiển việc bắt đầu, kết thúc và kết hợp các tiến trình của các hệ thống khác. Một tiến trình là một hệ thống con hay một module có thể thực thi song song với các tiến trình khác. Một dạng của mô hình này có thể được ứng dụng trong các hệ thống tuần tự mà việc quản lý các cuộc gọi thông thường tới các hệ thống phụ thuộc vào giá trị của một số biến trạng thái.



ình quản lý **thống thờ**

Hình 6.10 là một ví dụ cho mô hình quản lý tập trung trong việc điều khiển các hệ thống song song. Mô hình này thường được sử dụng trong các hệ thống thời gian thực "mềm", không có những ràng buộc quá chặt chẽ về mặt thời gian. Điều khiển trung tâm quản lý việc thực thi một tập hợp các tiến trình kết hợp với các bộ cảm biến và bộ tác động.

Tiến trình điều khiển hệ thống quyết định khi nào tiến trình được bắt đầu và kết thúc phụ thuộc vào các biến trạng thái. Nó kiểm tra xem các tiến trình khác có sinh ra các thông tin được xử lý hoặc chuyển thông tin tới chúng để xử lý. Điều khiển thông thường lặp liên tục, thu các bộ cảm biến và các tiến trình khác cho các sự kiện hoặc những thay đổi trạng thái. Với lý do này, mô hình này còn được gọi là mô hình lặp sự kiện (event-loop).

b. Mô hình điều khiển dựa trên sự kiện

Trong các mô hình điều khiển tập trung, các quyết định điều khiển thường được xác định bởi giá trị của các biến trạng thái. Ngược lại, các mô hình điều khiển hướng sự kiện được điều khiển bằng các sự kiện sinh ra từ bên ngoài.

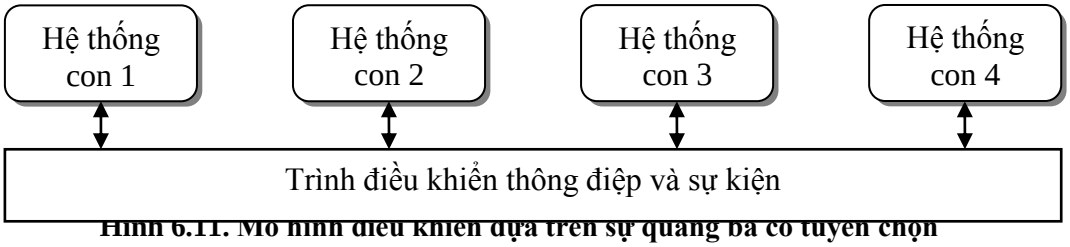
Thuật ngữ *"event" – sự kiện* trong ngữ cảnh này không chỉ là một tín hiệu nhị phân. Mà nó có thể là một tín hiệu để lấy một tập các giá trị hoặc một câu lệnh nhập từ menu. Sự khác biệt giữa một sự kiện và một đầu vào đơn giản là thời gian của sự kiện này nằm ngoài sự kiểm soát của tiến trình xử lý các sự kiện.

Có hai mô hình điều khiển bởi sự kiện:

- ***Mô hình broadcast (quảng bá)***

Trong các mô hình này, một sự kiện là một bản thông báo tới tất cả các hệ thống con. Các hệ thống con được lập trình để có thể trả lời sự kiện mà nó nhận được. Mô hình broadcast có hiệu lực trong các hệ thống con tích hợp, được phân bổ vào các máy tính khác nhau trên một mạng.

Trong hình 6.11, chính sách điều khiển không được nhúng vào sự kiện và thông điệp điều khiển. Các hệ thống con quyết định sự kiện nào mà chúng yêu cầu, điều khiển sự kiện và thông điệp phải đảm bảo rằng các sự kiện này được gửi tới đúng địa chỉ.



Hình 6.11. Mô hình điều khiển dựa trên sự quảng bá có tuyển chọn

Các sự kiện có thể truyền tới tất cả các hệ thống con, nhưng điều này là không thể đối với các hệ thống rất lớn. Thông thường, với các hệ thống này, mỗi hệ thống con có một thanh ghi lưu giữ các sự kiện và thông điệp mà chúng quan tâm. Còn đối với những hệ thống đơn giản (ví dụ như hệ thống máy tính cá nhân) được điều khiển bởi các sự kiện thông qua giao diện người dùng sẽ có những hệ thống riêng để lấy sự kiện từ chuột, bàn phím… và biến đổi thành những câu lệnh đặc biệt.

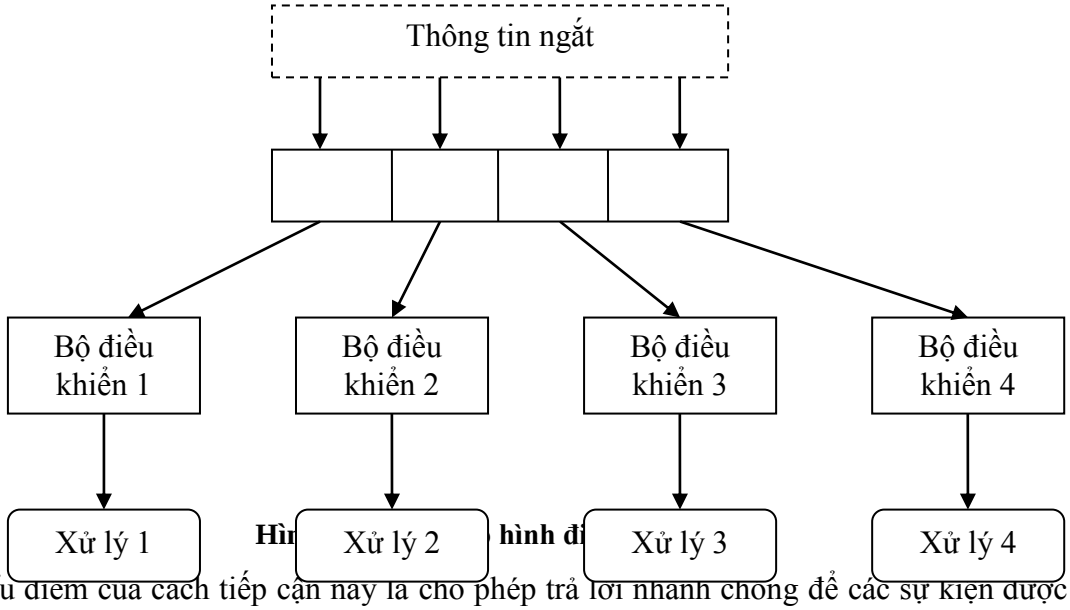
Ưu điểm của cách tiếp cận quảng bá (broadcast) là việc phát triển và mở rộng khá đơn giản. Một hệ thống con mới độc lập để điều khiển các lớp riêng biệt của các sự kiện có thể được tích hợp thông qua việc ghi nhớ các sự kiện của nó trong bộ điều khiển sự kiện. Bất kỳ hệ thống con nào cũng có thể kích hoạt hệ thống con khác mà không cần biết tên cũng như vị trí của hệ thống con. Các hệ thống con có thể được thực hiện trên các hệ thống máy phân tán. Việc phân tán hoàn toàn trong suốt đối với các hệ thống con.

Điểm bất lợi của mô hình này là các hệ thống con không biết khi nào các sự kiện sẽ được xử lý. Khi một hệ thống con sinh ra một sự kiện, nó không biết hệ thống con nào khác quan tâm đến sự kiện mà nó sinh ra. Vì thế dẫn đến khả năng nhiều hệ thống con cùng ghi nhớ các sự kiện giống nhau. Điều này có thể dẫn đến sự xung đột khi các kết quả điều khiển sự kiện được thực hiện.

- ***Mô hình điều khiển ngắt (Interrupt-driven)***

Các hệ thống thời gian thực yêu cầu việc tiếp nhận và trả lời các sự kiện một cách nhanh chóng. Ví dụ: nếu một hệ thống thời gian thực được ứng dụng để điều khiển các hệ thống an toàn trong ôtô. Nó phải phát hiện một vụ va chạm có thể xảy ra và đưa ra các tín hiệu cảnh báo sớm trước khi người lái xe đâm phải chướng ngại vật. Để cung cấp cơ chế trả lời sự kiện một cách nhanh chóng, ta phải sử dụng mô hình điều khiển ngắt.

Mô hình này thường được sử dụng trong các hệ thống thời gian thực, là những hệ thống cần có sự phản hồi các sự kiện nhanh và phải xử lý các sự kiện ngay lập tức. Nó có thể kết hợp với mô hình điều khiển tập trung. Mô hình điều khiển ngắt được minh họa trong hình 6.12. Mỗi loại ngắt kết hợp với một bộ nhớ cục bộ để lưu trữ địa chỉ của điều khiển. Khi một ngắt của một kiểu đặc biệt được nhận, một ngắt cứng (swich) gây ra điều khiển được truyền ngay tới điều khiển của nó. Điều khiển ngắt này sau đó có thể bắt đầu hoặc kết thúc một tiến trình khác để trả lời sự kiện mà ngắt này gửi tín hiệu đến.



Hìn hình đi

Ưu điểm của cách tiếp cận này là cho phép trả lời nhanh chóng để các sự kiện được thực hiện. Nhưng điểm bất lợi của nó là khó lập trình và khó kiểm thử. Đôi khi không thể tái tạo các nền của ngắt thời gian trong khi kiểm thử hệ thống. Rất khó để thay đổi các hệ thống được phát triển dựa trên mô hình này nếu như số lượng các ngắt bị giới hạn bởi phần cứng. Một khi giới hạn này được đạt tới, không có các kiểu sự kiện khác được điều khiển. Đôi khi ta có thể khắc phục nhược điểm này bằng cách bố trí một vài kiểu sự kiện vào một ngắt đơn. Tuy nhiên, việc sắp xếp lại các ngắt có thể không thực tế nếu như yêu cầu trả lời thật nhanh của ngắt.

6.2.5. Tiến trình thiết kế kiến trúc

Trong quá trình thiết kế, một hệ thống lớn sẽ được phân rã thành các hệ thống con độc lập tương đối với nhau. Tiến trình xác định các hệ thống con của một hệ thống và thiết lập một khung làm việc cho việc điều khiển và giao tiếp giữa chúng gọi là thiết kế kiến trúc. Kết quả của hoạt động này là một tài liệu thiết kế kiến trúc, mô tả kiến trúc của phần mềm.

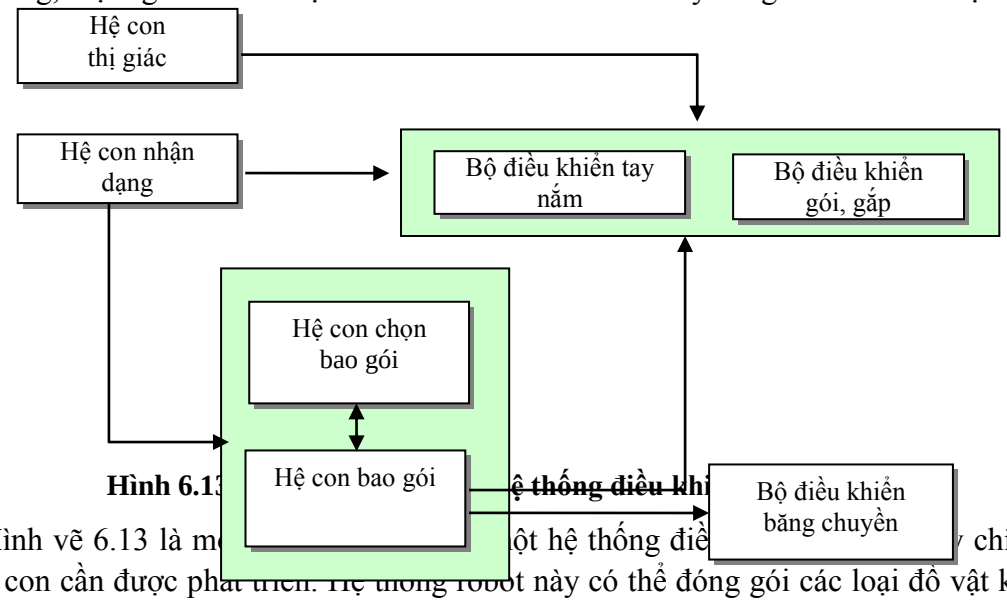
Thiết kế kiến trúc tập trung vào việc biểu diễn cấu trúc các thành phần, các thuộc tính và mối tương tác giữa chúng. Có nhiều cách tiếp cận khác nhau trong giai đoạn thiết kế. Tuy nhiên, ba hoạt động chủ chốt trong tiến trình thiết kế là:

1. *Cấu trúc hệ thống:* phân chia hệ thống thành các hệ thống con.
2. *Mô hình hóa điều khiển:* Thiết lập một mô hình biểu diễn mối quan hệ điều khiển giữa các thành phần trong hệ thống.
3. *Phân rã các module:* Mỗi hệ thống con được phân rã thành các module, ở bước này người làm thiết kế cần chỉ ra danh sách các module và mối liên kết giữa chúng.

Ở phần trên đã trình bày một số mô hình kiến trúc tiêu biểu, mỗi mô hình kiến trúc có ưu điểm và nhược điểm riêng, do đó nó có thể phù hợp với các hệ thống có đặc điểm khác nhau. Trên thực tế, các hệ thống lớn không phù hợp với một mô hình kiến trúc đơn, mà mỗi thành phần của nó có thể sử dụng một kiểu kiến trúc khác nhau.

a. Cấu trúc hệ thống

Giai đoạn đầu tiên trong tiến trình thiết kế kiến trúc thường liên quan tới việc phân rã một hệ thống thành một tập các hệ thống con có tương tác với nhau. Ở mức độ trìu tượng cao nhất, người ta sử dụng sơ đồ khối để minh họa việc thiết kế hệ thống con, mỗi khối trong sơ đồ là một hệ thống con. Khối trong khối chỉ ra rằng trong mỗi hệ thống con lại có thể bao gồm các hệ thống con nhỏ hơn. Các mũi tên chỉ ra rằng dữ liệu và các tín hiệu điều khiển được truyền từ hệ thống con này sang hệ thống con khác theo hướng mũi tên. Sơ đồ khối là bức tranh tổng quan về kiến trúc hệ thống, một người ở lĩnh vực khác khi nhìn vào sơ đồ này cũng có thể hiểu được.



Hình 6.13 ệ thống điều khi

Hình vẽ 6.13 là m ột hệ thống điề chỉ ra những hệ thống con cần được phát triển. Hệ thống robot này có thể đóng gói các loại đồ vật khác nhau. Nó sử dụng một hệ thống hiển thị để nhặt ra những đối tượng trên một băng chuyền, xác định kiểu đồ vật và lựa chọn đúng loại đóng gói. Hệ thống sau đó di chuyển đồ vật từ băng chuyền đến bộ phận đóng gói, xếp các đồ vật đã được đóng gói sang một băng chuyền khác.

b. Phân chia module

Sau khi tổ chức hệ thống thành các hệ thống con, giai đoạn tiếp theo của thiết kế kiến trúc là phân rã các hệ thống con thành các module. Có 2 chiến lược cơ bản có thể sử dụng khi phân chia một hệ thống con thành các modules:

- *Phân chia hướng đối tượng (Object-oriented decomposition):* phân tích hệ thống thành một tập hợp các đối tượng trao đổi thông tin, liên kết với nhau.

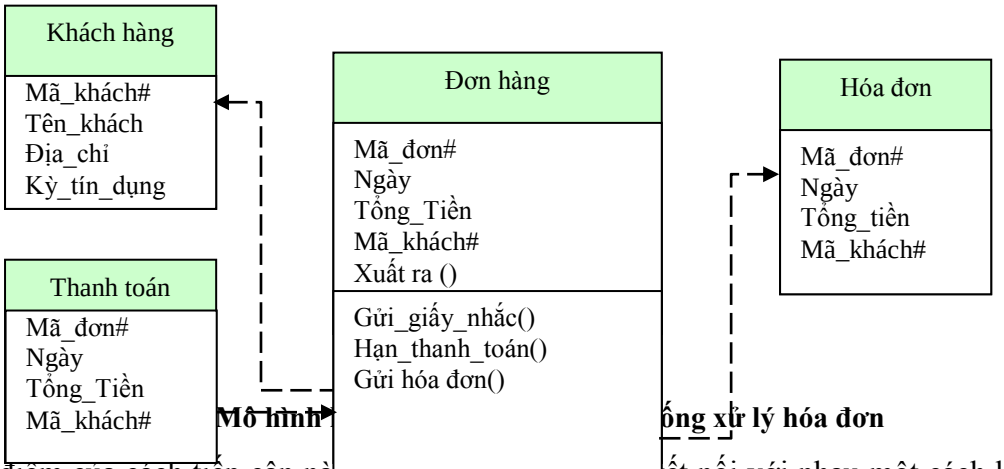
- *Đường ống hướng chức năng (Function-oriented pipelining):* phân tích hệ thống thành các module chức năng theo cơ chế mỗi module nhận dữ liệu đầu vào để xử lý và đưa kết quả ra.

- ***Mô hình hướng đối tượng***

Trong mô hình hướng đối tượng, mô hình kiến trúc phân tích hệ thống thành một tập các đối tượng kết hợp với nhau từng đôi một thông qua giao diện được định nghĩa rõ ràng. Các đối tượng gọi các dịch vụ được cung cấp bởi các đối tượng khác.

Sự phân chia hướng đối tượng liên quan đến việc xác định các lớp đối tượng, thuộc tính và các phương thức của nó. Khi được thực hiện, các đối tượng được tạo ra từ các lớp này và một vài mô hình điều khiển được sử dụng để phối hợp các phương thức của đối tượng.

Hình vẽ 6.14 là một ví dụ về mô hình kiến trúc hướng đối tượng của hệ thống xử lý đơn hàng. Hệ thống này có thể tự động chuyển các đơn hàng đến khách hàng, nhận tiền trả và sinh ra các hóa đơn đã thanh toán, đồng thời có thể ghi nhớ các đơn hàng chưa được thanh toán. Ở đây tác giả đã sử dụng lược đồ UML để biểu diễn quan hệ giữa các lớp. Mỗi lớp có tên, thuộc tính và các phương thức. Lớp *Đơn hàng* có các phương thức khác nhau để thực hiện các chức năng của hệ thống. Lớp này sẽ tạo ra các phương thức để xử lý các lớp *Hóa đơn, Khách hàng và Thanh toán.*



Mô hình ống xử lý hóa đơn

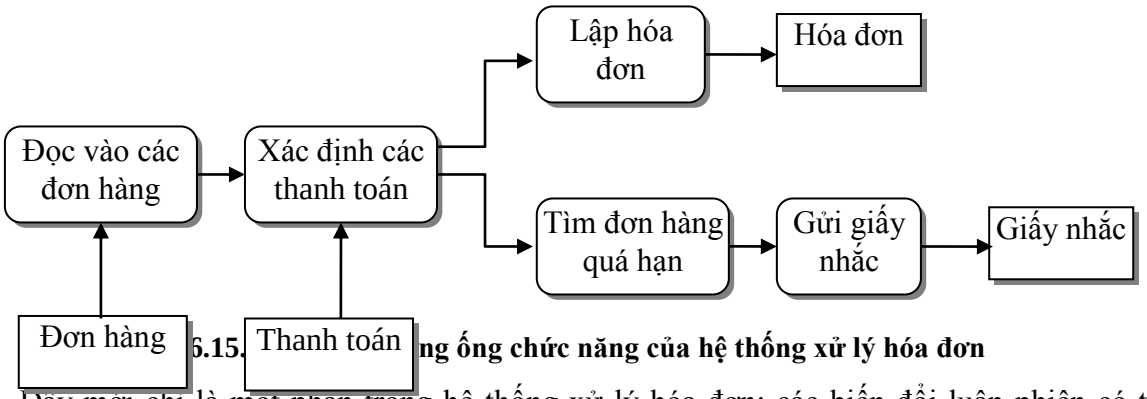
Ưu điểm của cách tiếp cận này là các đối tượng được kết nối với nhau một cách lỏng lẻo, vì thế khi một đối tượng thay đổi sẽ không ảnh hưởng nhiều đến các đối tượng khác. Các đối tượng thông thường là các thực thể của hệ thống, vì thế, người ta có thể hiểu chúng một cách nhanh chóng. Bởi các thực thể này được sử dụng trong những hệ thống khác nhau, nên có khả năng tái sử dụng. Các ngôn ngữ lập trình hướng đối tượng ngày các phong phú để hỗ trợ cách tiếp cận này.

Tuy nhiên, cách tiếp cận hướng đối tượng cũng có những điểm bất lợi. Để sử dụng các dịch vụ, các đối tượng phải tham chiếu một cách rõ ràng đến tên và giao diện của các đối tượng khác. Nếu một giao diện thay đổi, đòi hỏi cả hệ thống phải thay đổi theo, sự thay đổi này còn ảnh hưởng đến cả người sử dụng. Khi các đối tượng có thể liên kết với nhau để thể hiện các thực thể trong thế giới thực, các thực thể phức tạp đôi khi khó trìu tượng hóa thành các đối tượng.

- ***Mô hình luồng dữ liệu (đường ống hướng chức năng)***

Trong mô hình đường ống hướng chức năng hay luồng dữ liệu, các bộ biến đổi chức năng xử lý dữ liệu đầu vào và các thủ tục đầu ra. Các luồng dữ liệu chuyển từ nơi này sang nơi khác và được biến đổi liên tục. Mỗi bước trong tiến trình được thực hiện như một bước biến đổi. Luồng dữ liệu vào đi qua bộ biến đổi này sẽ được chuyển thành dữ liệu đầu ra. Sự biến đổi này có thể được thực hiện tuần tự hoặc song song. Dữ liệu có thể được xử lý từng bước hoặc thành từng đợt.

Một ví dụ về kiểu hệ thống ứng dụng kiến trúc này được chỉ ra trong hình 6.15. Một tổ chức phát hành hóa đơn cho khách hàng. Một tuần một lần, việc thanh toán phải được thực hiện hợp lý với các hóa đơn. Với những hóa đơn đã được thanh toán, phải đưa ra giấy biên nhận, với những hóa đơn chưa được thanh toán trong thời gian trả cho phép, thì phải có một lệnh nhắc nhở.



6.15. ng ống chức năng của hệ thống xử lý hóa đơn

Đây mới chỉ là một phần trong hệ thống xử lý hóa đơn: các biến đổi luân phiên có thể được sử dụng để sinh ra các hóa đơn. Mô hình đối tượng trìu tượng hơn vì nó không có những thông tin về các hoạt động tuần tự. Mô hình luồng dữ liệu có các ưu điểm sau:

- Có thể sử dụng lại các bộ biến đổi (transformation).

- Trực quan khi nhiều người nghĩ công việc của họ là xử lý dữ liệu đầu vào để thu được kết quả ở đầu ra.

- Cải tiến hệ thống bằng việc bổ xung thêm những bộ biến đổi mới sẽ không phức tạp.

- Đơn giản đối với những hệ thống song song cũng như tuần tự.

Vấn đề cơ bản của mô hình này là định dạng chung cho dữ liệu cần biến đổi, sao cho tất cả các bộ biến đổi đều nhận biết và xử lý được dữ liệu truyền qua. Mỗi biến đổi hoặc phải được sự chấp của các bộ biến đổi giao tiếp với nó, hoặc phải có một định dạng chuẩn cho tất cả các dữ liệu được truyền qua. Hơn nữa, cách tiếp cận này chỉ khả thi khi các bộ biến đổi là độc lập và có thể sử dụng lại.

6.3. THIẾT KẾ HƯỚNG ĐỐI TƯỢNG

6.3.1. Một số đặc điểm cơ bản của thiết kế hướng đối tượng

a. Chiến lược thiết kế hướng đối tượng

Trong chiến lược thiết kế hướng đối tượng, các bước phân tích, thiết kế, lập trình hướng đối tượng có quan hệ với nhau nhưng lại có những điểm khác biệt:

- Phân tích hướng đối tượng (AOO) liên quan đến việc phát triển một mô hình hướng đối tượng trong một lĩnh vực ứng dụng. Các đối tượng trong mô hình phản ánh các thực thể và các phương thức kết hợp với các vấn đề được giải quyết.

- Thiết kế hướng đối tượng (DOO) liên quan đến việc phát triển một mô hình hướng đối tượng cho một hệ thống phần mềm để thực hiện những yêu cầu đã đặt ra. Các đối tượng trong thiết kế hướng đối tượng liên quan tới phương án giải quyết các vấn đề. Đây có thể là mối quan hệ mật thiết giữa vấn đề và phương án giải quyết, nhưng người làm thiết kế không thể tránh được việc phải bổ xung thêm các đối tượng mới và biến đổi các đối tượng đã có để thực hiện giải pháp.

- Lập trình hướng đối tượng (OOP) liên quan đến việc thực thi một bản thiết kế phần mềm bằng một ngôn ngữ lập trình hướng đối tượng, chẳng hạn như Java, Pithon, C++... Một ngôn ngữ lập trình hướng đối tượng cung cấp các cấu trúc để định nghĩa các lớp đối tượng và các hệ thống chạy theo thời gian (run-time) để tạo ra các đối tượng từ các lớp này.

Một cách lý tưởng, sự chuyển tiếp giữa các giai đoạn phát triển là liên tục với các khái niệm thích hợp được sử dụng trong mỗi giai đoạn. Việc chuyển sang giai đoạn tiếp theo liên quan đến việc làm mịn các giai đoạn trước bằng việc thêm chi tiết vào các lớp đối tượng đang tồn tại và tạo ra các lớp đối tượng mới để cung cấp các chức năng bổ sung. Vì thông tin ẩn trong các đối tượng, nên quyết định thiết kế chi tiết về việc biểu diễn dữ liệu có thể được trì hoãn cho tới khi hệ thống được thực hiện. Trong một số trường hợp, các quyết định về việc phân chia đối tượng, các đối tượng tuần tự hay song song cũng có thể được trì hoãn.

b. Đặc điểm của thiết kế hướng đối tượng

Các đối tượng được trìu tượng hóa từ các thực thể của hệ thống hoặc của thế giới thực. Các đối tượng là độc lập có ẩn chứa trạng thái và những thông tin mô tả còn chức năng hệ thống được diễn tả qua thuật ngữ "dịch vụ đối tượng". Trong phương pháp này, vùng chia sẻ dữ liệu được loại trừ. Các đối tượng trao đổi thông tin qua các thông điệp (message). Các đối tượng có thể tổ chức phân tán và có thể thực thi tuần tự hoặc song song.

- ***Ưu điểm của phương pháp hướng đối tượng***

Các hệ thống hướng đối tượng dễ thay đổi (bảo trì) hơn các hệ thống được phát triển bằng cách sử dụng các cách tiếp cận khác bởi các đối tượng là độc lập. Chúng có thể được hiểu và thay đổi như các thực thể độc lập (standalone – không cần phải cài đặt). Thay đổi việc thực thi các đối tượng hoặc thêm các dịch vụ sẽ không làm ảnh hưởng tới các đối tượng khác trong hệ thống. Từ đó nâng được tính dễ hiểu và do đó tăng tính bảo trì cho hệ thống.

Một cách tiềm năng, các đối tượng là những thành phần có thể được sử dụng lại bởi chúng là những thực thể độc lập trong đó ẩn chứa các thuộc tính và phương thức. Những thiết kế có thể được phát triển sử dụng các đối tượng được tạo ra trong những bản thiết kế trước. Điều

này giúp giảm chi phí cho thiết kế, lập trình và kiểm chứng. Nó cũng có thể dẫn đến việc sử dụng các đối tượng tiêu chuẩn (từ đó tăng tính dễ hiểu của bản thiết kế) và giảm được những rủi ro liên quan đến việc phát triển phần mềm.

Đối với một số hệ thống, có thể có mối liên hệ rõ ràng giữa các thực thể trong thế giới thực với các đối tượng của hệ thống.

6.3.2. Đối tượng và lớp đối tượng

a. Khái niệm đối tượng và lớp đối tượng

Thuật ngữ đối tượng và hướng đối tượng được ứng dụng trong các vấn đề khác nhau như: phương pháp thiết kế, các hệ thống và các ngôn ngữ lập trình. Người ta cho rằng, một đối tượng là một gói thông tin, điều này được phản ánh trong định nghĩa về đối tượng và lớp đối tượng theo các định nghĩa sau:

Một đối tượng là một thực thể có một trạng thái và một tập các thao tác được thực hiện trong một trạng thái. Một trạng thái được mô tả là một tập các thuộc tính của đối tượng. Các thao tác kết hợp với đối tượng cung cấp các dịch vụ cho các đối tượng khác, những đối tượng sẽ có yêu cầu dịch vụ khi cần phải tính toán.

Các đối tượng được tạo ra từ một vài định nghĩa lớp đối tượng. ***Một lớp đối tượng*** xác định các dịch vụ, như một khuôn mẫu cho các đối tượng, bao gồm việc khai báo tất cả các thuộc tính và dịch vụ mà một đối tượng thuộc lớp đó cần phải có. Hình 6.16 dưới đây là một ví dụ về một lớp đối tượng được biểu diễn bằng ngôn ngữ UML.

Tài khoản
Số_tài_khoản: String Tên_Khách: String Số_dư: Money
+Tạo_lập (Tên: String, số_tiền: Money) +Gửi (số_TK: String, số_tiền: Money) +Chuyển (Số_TK: String, số_tiền: Money, Số_Tk2: String) +Rút (Số_tk: String, số_tiền: Money)

Hình 6-16. Mô hình lớp đối tượng "Tài khoản" với các thuộc tính và phương thức của nó

b. Trao đổi thông tin giữa các lớp đối tượng

Các đối tượng trao đổi thông tin qua việc yêu cầu các dịch vụ từ đối tượng khác, nếu cần thiết, nó có thể trao đổi thông tin được yêu cầu với các dịch vụ dự trữ. Các bản sao của thông tin cần để thực thi dịch vụ và kết quả được truyền qua các tham số.

Về mặt logic, các đối tượng giao tiếp, trao đổi thông tin với nhau thông qua việc truyền các thông điệp. Mỗi thông điệp bao gồm:

- Tên của dịch vụ được yêu cầu bởi đối tượng gọi;
- Danh sách các tham số cần thiết để thực thi một dịch vụ và giá trị trả về của dịch vụ đó.

Dưới đây là một số ví dụ về thông điệp và việc truyền thông điệp giữa các lớp đối tượng:

// Gọi một phương thức với một đối tượng trong bộ đệm và sau đó trả lại giá trị tiếp theo trong bộ đệm

V = circularBuffer.Get

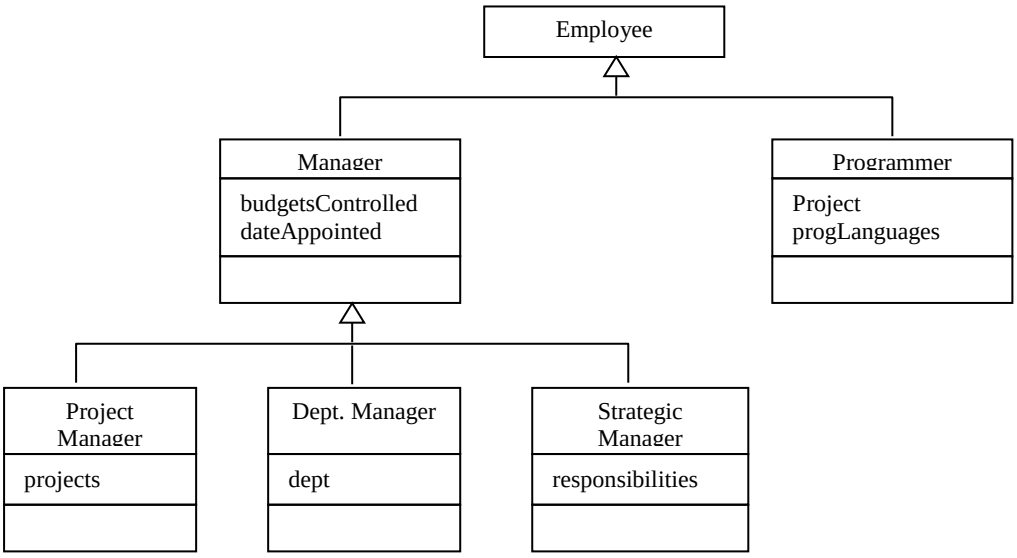
//Gọi phương thức kết hợp với đối tượng máy điều nhiệt để thiết lập nhiệt độ được bảo trì

Thermostat.setTemp(20);

c. Khái quát hóa và kế thừa giữa các lớp đối tượng

Các lớp đối tượng có thể được sắp xếp trong một cây kế thừa để chỉ ra mối quan hệ giữa các lớp đối tượng chung và các lớp đối tượng đặc trưng. Những đối tượng ở lớp con sẽ giữ nguyên các phương thức và thuộc tính của lớp cha, ngoài ra nó còn có thể bổ sung thêm các phương thức và thuộc tính mới của riêng nó. Trong UML, mũi tên chỉ từ lớp con đến lớp cha thể hiện lớp mà nó được kế thừa. Trong các ngôn ngữ lập trình hướng đối tượng, sự tổng quát hóa được thực hiện bằng cách sử dụng mô hình kế thừa. Các lớp con được kế thừa các phương thức và thuộc tính của lớp cha.

Hình 6.17 là một ví dụ về một cây đối tượng kế thừa chỉ ra các lớp nhân viên khác nhau. Các lớp thấp hơn trong cây kế thừa có các thuộc tính và phương thức của lớp cha nhưng có thêm một số thuộc tính và phương thức mới hoặc thay đổi một số thông tin của lớp cha.



Hình 6-17. Mô hình kế thừa của lớp nhân viên trong một công ty phần mềm

Lớp Manager trong hình trên có tất cả các thuộc tính và phương thức của lớp *Employee*, nhưng thêm vào đó, nó có hai thuộc tính mới để lưu ngân sách được sử dụng bởi người quản lý và ngày mà người quản lý được chỉ định để giữ vai trò này. Tương tự như vậy, lớp lập trình viên cũng được bổ xung thêm thuộc tính mới để thể hiện dự án mà người đó đang tham gia, cũng như những kỹ năng nghề nghiệp của người đó. Các đối tượng của các lớp *Manager* và *Programmer* được sử dụng ở bất kỳ đâu khi một đối tượng của lớp nhân viên được yêu cầu.

Ưu điểm: Mô hình kế thừa trong hướng đối tượng giống như một máy ảo và có thể được sử dụng để phân lớp các thực thể. Đây là một kỹ thuật tái sử dụng ở cả mức thiết kế và mức lập trình. Sơ đồ kế thừa là cơ sở của kiến thức tổ chức về hệ thống và lĩnh vực ứng dụng.

Tuy nhiên, kế thừa cũng có các ***nhược điểm*** như: các lớp đối tượng không độc lập, không thể hiểu được nếu thiếu sự tham chiếu đến lớp cha của nó. Những người làm thiết kế có khuynh hướng sử dụng lại sơ đồ kế thừa được tạo ra trong quá trình phân tích. Có thể dẫn tới việc thực hiện không hiệu quả.

6.3.3. Tiến trình thiết kế hướng đối tượng

Để minh họa tiến trình thiết kế hướng đối tượng, chúng ta cùng phân tích quá trình thiết kế một phần mềm điều khiển nhúng trong hệ thống trạm dự báo thời tiết tự động. Hoạt động thiết kế hướng đối tượng về cơ bản bao gồm các bước sau:

- *Hiểu và xác định ngữ cảnh và các mô hình của việc sử dụng của hệ thống;*
- *Thiết kế kiến trúc hệ thống;*
- *Xác định những đối tượng chính trong hệ thống;*
- *Xây dựng các mô hình thiết kế;*
- *Đặc tả giao diện của đối tượng.*

Các hoạt động này được mô tả riêng biệt. Tuy nhiên trên thực tế, tất cả những hoạt động nói trên được đan xen vào nhau nên nó ảnh hưởng tới những hoạt động khác. Các đối tượng được xác định và các giao diện được đặc tả một phần hoặc đầy đủ khi kiến trúc của hệ thống đã được xác định. Khi các mô hình đối tượng được sinh ra, các định nghĩa đối tượng đặc biệt này có thể được làm mịn dần, điều này có thể làm thay đổi kiến trúc hệ thống.

a. Giới thiệu hoạt động của trạm thời tiết

Phần này sử dụng hệ thống tạo ra bản đồ thời tiết để minh họa những hoạt động trong tiến trình thiết kế hướng đối tượng, trong đó tập trung chủ yếu vào hệ thống thu thập tự động các dữ liệu phục vụ công tác dự báo thời tiết. Mô hình kiến trúc tổng quan của hệ thống có thể được mô tả tóm tắt như sau:

Hệ thống bản đồ thời tiết được tạo ra trên cơ sở thường xuyên thu thập dữ liệu từ các địa điểm khác nhau bằng các thiết bị được điều khiển từ xa và không cần người giám sát như: kinh khí cầu, vệ tinh, trạm quan sát... Các trạm thời tiết truyền dữ liệu mà nó nhận được vào máy tính để trả lời những yêu cầu mà máy tính đó nhận được.

Hệ thống máy tính xác nhận dữ liệu thu thập được và tích hợp nó với các dữ liệu từ các nguồn khác nhau. Dữ liệu này được tích hợp và lưu trữ, kết hợp với dữ liệu bản đồ được số hóa để xây dựng bản đồ thời tiết cho từng địa phương. Các bản đồ này có thể được in ra hoặc hiển thị ở nhiều dạng nhằm đáp ứng các yêu cầu khác nhau. Tóm lại, hệ thống bản đồ thời tiết thực hiện bốn công việc chính: Thu thập dữ liệu, tích hợp dữ liệu từ nhiều nguồn khác nhau, cuối cùng là

lưu trữ và tạo ra các bản đồ thời tiết. Tuy nhiên, trong phần này chúng ta chỉ đi vào phân tích một phần của hệ thống xây dựng bản đồ thời tiết, đó là các trạm khí tượng.

Một trạm khí tượng là một gói phần mềm điều khiển các thiết bị để thu thập dữ liệu, thực hiện việc xử lý dữ liệu và truyền những dữ liệu này để tiếp tục xử lý. Các thiết bị bao gồm các dụng cụ đo nhiệt độ không khí và mặt đất, đo tốc độ gió, bánh lái chỉ hướng gió, thiết bị đo áp suất không khí, đo lượng mưa. Việc thu thập dữ liệu được thực hiện định kỳ. Khi một lệnh được đưa ra để truyền dữ liệu thời tiết, các trạm thời tiết xử lý và tóm lược dữ liệu thu thập được. Những dữ liệu được tóm lược sẽ được truyền đến máy tính lập bản đồ khi nhận được yêu cầu.

b. Mô hình ngữ cảnh và mô hình sử dụng

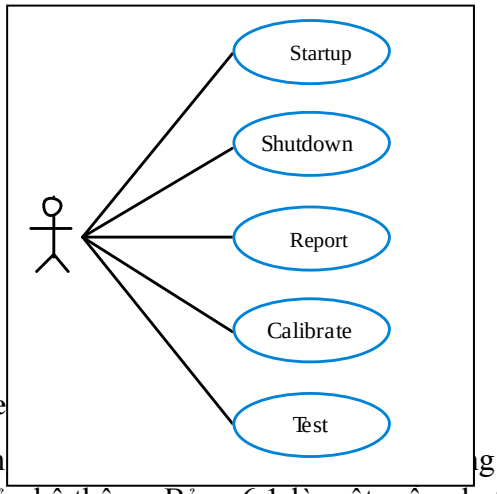
Giai đoạn đầu tiên trong bất kỳ tiến trình thiết kế nào cũng là việc tăng cường sự hiểu biết về các mối quan hệ giữa phần mềm đang được thiết kế với môi trường bên ngoài. Người làm thiết kế cần phải hiểu điều này để có thể đưa ra các quyết định như: làm thế nào đáp ứng được những yêu cầu chức năng của hệ thống, làm thế nào để cấu trúc hệ thống, giúp nó giao tiếp với môi trường thực thi. Ngữ cảnh hệ thống và các mô hình sử dụng của hệ thống được biểu diễn bằng hai mô hình bổ xung nhau qua mối quan hệ giữa hệ thống và môi trường của nó.

- Mô hình ngữ cảnh: là một mô hình tĩnh mô tả các hệ thống khác trong môi trường, sử dụng mô hình hệ thống con để chỉ ra các hệ thống khác.

- Mô hình sử dụng: là một mô hình mô tả hệ thống hiện tại tương tác với môi trường như thế nào. Để thể hiện điều này, người ta sử dụng các biểu đồ Use-case để chỉ ra sự tương tác.

Mô hình sử dụng (use-case) cho trạm thời tiết được chỉ ra trong hình 6.18. Hình vẽ này nói lên rằng trạm thời tiết tương tác với các thực thể bên ngoài để thực hiện các dịch vụ sau:

- Khởi động hệ thống (Startup)
- Tắt hệ thống (Shutdown)
- Báo cáo dữ liệu thời tiết đã thu thập được (report)
- Hiệu chỉnh thông số của các thiết bị trong trạm (Celibrate)
- Kiểm tra trạng thái các thiết bị (test)



Use-case ngôn n g người làm thiết kế xác định các đối tượng và chức năng của hệ thống. Bảng 6.1 là một mẫu chuẩn cho việc mô tả một use-case. Trong đó có những thông tin chi tiết liên quan tới tên chức năng.

Bảng 6.1. Mô tả một use-case

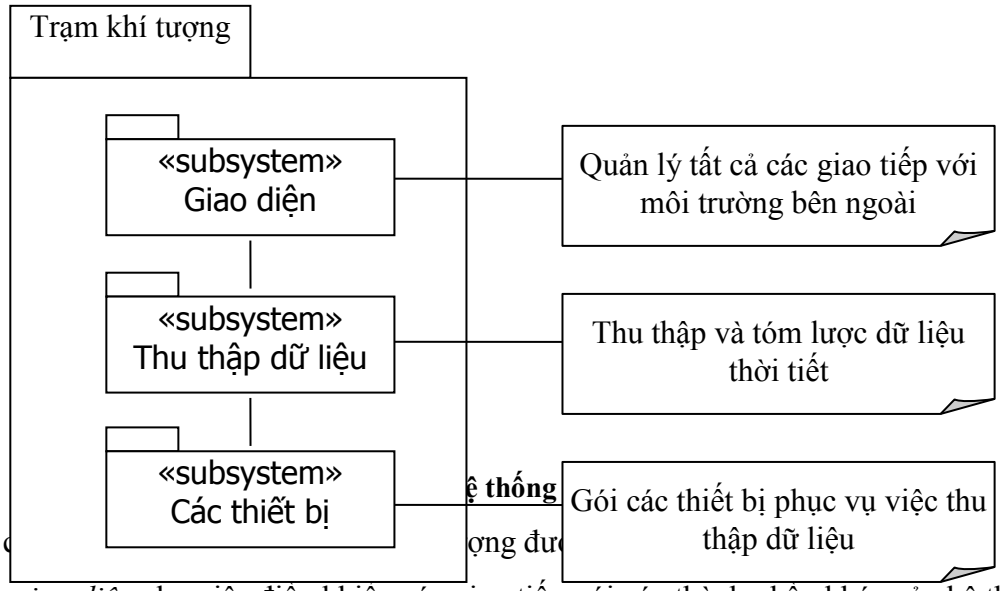
System (hệ thống)	Trạm khí tượng
Use-case (chức năng)	Báo cáo dữ liệu thời tiết thu thập được

Actor (tác nhân)	Hệ thống bản đồ thời tiết, trạm khí tượng
Data (dữ liệu)	Trạm khí tượng gửi một bản tóm lược dữ liệu thời tiết mà nó thu thập được từ các thiết bị trong trạm tới hệ thống bản đồ thời tiết. Dữ liệu được thu thập bao gồm: giá trị lớn nhất, giá trị nhỏ nhất và giá trị trung bình của nhiệt độ không khí và nhiệt độ trên mặt đất; của tốc độ gió, áp suất không khí, tổng lượng mưa và hướng gió… 5 phút một lần.
Stimulus	Hệ thống thu thập dữ liệu thời tiết có một đường tín hiệu kết nối với các trạm khí tượng và chuyển các yêu về việc truyền dữ liệu
Response	Dữ liệu tóm lược được sẽ được gửi tới hệ thống thu thập dữ liệu thời tiết
Comments (chú ý)	Các trạm khí tượng thường xuyên được yêu cầu chuyển dữ liệu về trung tâm (mỗi giờ một lần). Tuy nhiên tần suất này có thể thay đổi tùy theo vị trí của các trạm khí tượng

Mô tả use case giúp ta xác định các đối tượng và các chức năng của hệ thống. Từ mô tả use-case Report, người ta có thể hình dung ra những thiết bị cần để thu thập dữ liệu khí tượng, đối tượng biểu diễn thông tin, các thao tác yêu cầu dữ liệu và gửi dữ liệu được yêu cầu.

c. Thiết kế kiến trúc

Mối tương tác giữa hệ thống phần mềm đang được thiết kế và môi trường hệ thống được lấy làm cơ sở cho việc thiết kế kiến trúc. Tuy nhiên cũng cần phải kết hợp với những kiến thức cơ bản về thiết kế kiến trúc cũng như những hiểu biết trong lĩnh vực của dự án. Trong trường hợp này, kiến trúc phân tầng là thích hợp cho trạm khí tượng. Kiến trúc của hệ thống này được mô tả trong hình 6.19.



Như c ệ thống ợng đư

- *Tầng giao diện* cho việc điều khiển các giao tiếp với các thành phần khác của hệ thống và môi trường bên ngoài.

- *Tầng thu thập dữ liệu* liên quan tới việc quản lý và thu thập dữ liệu từ các thiết bị tóm lược dữ liệu trước khi chuyển tới hệ thống bản đồ thời tiết.

- *Tầng thiết bị* là tầng che dấu tất cả các thiết bị để thu thập dữ liệu thời tiết.

Nói chung, ta nên phân tích hệ thống bằng một kiến trúc đơn giản nhất có thể. Thực tế cho thấy không nên có nhiều hơn bảy thực thể cơ bản trong một mô hình kiến trúc. Mỗi thực thể này có thể mô tả riêng biệt và có thể sử dụng để khám phá ra cấu trúc bên trong của nó.

d. Xác định đối tượng chính trong hệ thống

Trong giai đoạn này của tiến trình, nên có một vài ý tưởng về các đối tượng chính trong hệ thống đang được thiết kế. Trong hệ thống trạm khí tượng, các thiết bị sẽ là các đối tượng, ta cần ít nhất một đối tượng tại mỗi mức kiến trúc. Điều này phản ánh những nét cơ bản mà các đối tượng hướng tới trong quá trình thiết kế. Tuy nhiên, cũng thường xuyên phải tìm kiếm và tài liệu hoá các đối tượng có thể có liên quan.

Mặc dù phần này có tiêu đề là xác định đối tượng, trong thực tế, tiến trình này liên quan đến việc xác định các lớp đối tượng. Việc thiết kế được mô tả trong các thuật ngữ của các lớp này. Hiển nhiên, ta cần phải làm mịn các lớp đối tượng đã xác định từ giai đoạn khởi đầu và xem xét lại ở giai đoạn này để hiểu sâu hơn việc thiết kế.

- ***Các cách tiếp cận để xác định đối tượng***

Có bốn cách tiếp cận để xác định các đối tượng của hệ thống:

1. *Sử dụng một phân tích ngữ pháp của ngôn ngữ tự nhiên mô tả hệ thống.* Các đối tượng và những thuộc tính của nó phải là danh từ, các phương thức hoặc các dịch vụ phải là động từ. Cách tiếp cận này được đưa vào phương pháp HOOD - cho thiết kế hướng đối tượng (Robinson, 1992) đã được sử dụng rộng rãi trong các lĩnh vực công nghiệp ở Châu Âu.

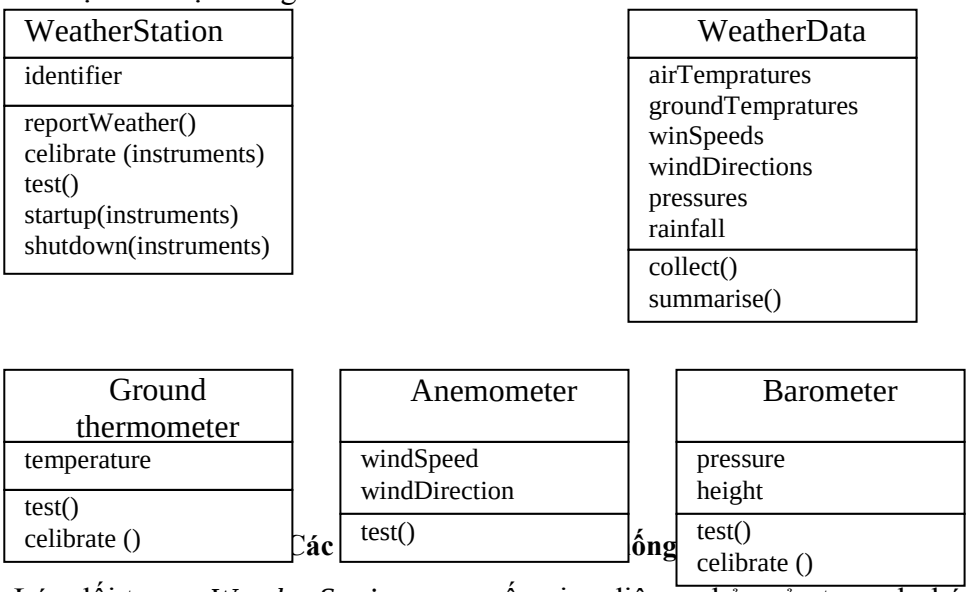
2. *Sử dụng các vật thể hữu hình trong lĩnh vực ứng dụng*, chẳng hạn như máy bay, các vai trò như người quản lý, các sự kiện như yêu cầu, các tương tác như gặp gỡ, các vị trí như văn phòng, tổ chức như công ty... Hỗ trợ bằng cách xác định các cấu trúc lưu trữ trong giải pháp của lĩnh vực có thể được yêu cầu để hỗ trợ các đối tượng này.

3. *Sử dụng cách tiếp cận hành vi*, trong đó người thiết kế đầu tiên hiểu qua cách xử lý của hệ thống. Những cách thực hiện khác nhau được phân công cho những phần khác nhau của hệ thống và sự hiểu biết được bắt nguồn từ việc ai khởi động và tham gia vào những công việc này. Những người tham gia đóng vai trò có ý nghĩa được xem như các đối tượng.

4. *Sử dụng phân tích dựa trên kịch bản.* Khi một kịch bản được phân tích, đội chịu trách nhiệm phân tích phải xác định những đối tượng được yêu cầu, các thuộc tính và phương thức. Một phương thức của việc phân tích được gọi là thẻ CRC, nơi những người làm phân tích dựa vào vai trò của các đối tượng được hỗ trợ có hiệu quả trong phương pháp tiếp cận dựa trên kịch bản này.

Những cách tiếp cận này sẽ giúp ta bắt đầu xác định đối tượng. Trong thực tế, phải sử dụng vào nhiều nguồn kiến thức cơ bản khác nhau để khám phá ra các lớp đối tượng. Các lớp đối tượng, thuộc tính và các phương thức được xác định ban đầu thông qua các mô tả không chính thức về hệ thống có thể là điểm bắt đầu của việc thiết kế. Thông tin bổ xung từ những hiểu biết về lĩnh vực ứng dụng hoặc phân tích kịch bản sau đó sẽ được sử dụng để làm mịn và mở rộng các đối tượng ban đầu. Những thông tin này có thể được thu thập từ các tài liệu yêu cầu, từ việc thảo luận với người sử dụng và từ việc phân tích các hệ thống đang tồn tại.

Bằng cách tiếp cận lai, chúng ta có thể nhận thấy hệ thống trạm khí tượng có năm lớp đối tượng chính được thể hiện trong hình vẽ 6.20.



Các ống

1. Lớp đối tượng *WeatherStation* cung cấp giao diện cơ bản của trạm dự báo thời tiết với môi trường của nó. Tuy nhiên, các phương thức phản ánh những tương tác này được chỉ ra trong hình 6.20. Trong trường hợp này, sử dụng một lớp đối tượng đơn để che dấu tất cả những tương tác, nhưng trong các bản thiết kế khác, bạn có thể lựa chọn để thiết kế giao diện của hệ thống như là các lớp khác nhau.

2. Lớp đối tượng *WeatherData* che dấu dữ liệu thu được từ các thiết bị trong trạm dự báo thời tiết. Các phương thức của đối tượng này liên quan đến việc thu thập và tóm lược dữ liệu được yêu cầu.

3. Các lớp đối tượng *Ground thermometer, Anemometer và Barometer* liên quan trực tiếp tới các thiết bị trong hệ thống. Chúng phản ánh những thực thể phần cứng hữu hình trong hệ thống và các phương thức liên quan đến việc điều khiển phần cứng.

- ***Làm mịn đối tượng***

Trong giai đoạn tiếp theo của tiến trình thiết kế, cần phải sử dụng những kiến thức của lĩnh vực ứng dụng để xác định các đối tượng và các dịch vụ. Chúng ta biết rằng, các trạm khí tượng thường để ở những nơi xa và có một số thiết bị mà thỉnh thoảng chúng có thể bị hỏng. Các thiết bị nếu bị lỗi thì cần phải có thông báo lỗi tự động. Điều này đòi hỏi phải có những thuộc tính và những phương thức để kiểm tra xem các thiết bị có hoạt động theo đúng chức năng mong muốn hay không. Như bạn thấy, có rất nhiều trạm khí tượng từ xa nên phải xác định dữ liệu được thu thập từ mỗi trạm, vì thế cũng cần phải đánh số cho các trạm.

Trong ví dụ này, chúng ta nhận thấy các đối tượng gắn với các thiết bị không phải là những đối tượng chủ động. Phương thức collect() của đối tượng WeatherData gọi trên các đối tượng thiết bị để thực hiện việc đọc dữ liệu khi có yêu cầu.

Nếu để các đối tượng này là các đối tượng chủ động thì nó phải bao gồm cả điều khiển của chính nó, trong trường hợp này có nghĩa là mỗi thiết bị sẽ tự quyết định khi nào nó sẽ tiến hành đọc dữ liệu. Điểm bất lợi ở đây là nếu một quyết định được thực hiện để thay đổi thời gian thu thập dữ liệu hoặc nếu các trạm làm việc khác nhau có thời gian thu thập dữ liệu khác nhau, thì

các lớp đối tượng mới phải được đưa ra. Bằng việc tạo ra các đối tượng thiết bị đọc yêu cầu, bất kỳ sự thay đổi nào trong chiến lược thu thập dữ liệu đều có thể thực hiện một cách dễ dàng mà không cần thay đổi các đối tượng kết nối với các thiết bị.

e. Xây dựng các mô hình thiết kế

Các mô hình thiết kế chỉ ra các đối tượng hoặc các lớp đối tượng trong hệ thống và mối quan hệ giữa các thực thể này. Các mô hình thiết kế có nhiều khả năng sẽ trở thành các bản thiết kế. Nó là cầu nối giữa yêu cầu hệ thống và việc thực hiện hệ thống. Điều này có nghĩa là sẽ có những yêu cầu xung đột trong các mô hình này. Những xung đột này có thể được giải quyết bằng cách phát triển các mô hình ở những mức độ chi tiết khác nhau. Tuy nhiên, một bước quan trọng trong tiến trình thiết kế là việc quyết định mô hình nào sẽ được sử dụng và mức độ chi tiết của các mô hình. Điều này phụ thuộc vào kiểu hệ thống sẽ được phát triển. Một hệ thống xử lý tuần tự sẽ được thiết kế theo một cách khác so với các hệ thống nhúng thời gian thực, do đó sẽ sử dụng các mô hình thiết kế khác nhau. Có rất ít hệ thống cần đến tất cả các mô hình thiết kế. Tối thiểu hóa các mô hình thiết kế sẽ làm giảm chi phí và thời gian cần thiết cho tiến trình thiết kế.

Có 2 kiểu mô hình thiết kế thường được sử dụng trong thiết kế hướng đối tượng:

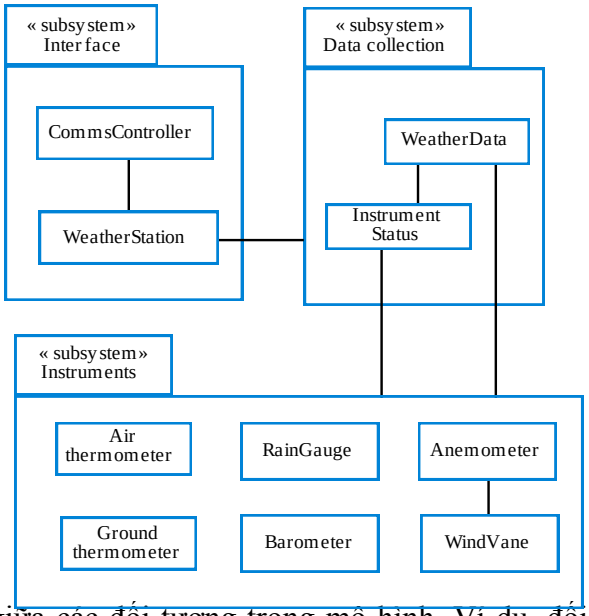
- *Các mô hình tĩnh:* mô tả cấu trúc tĩnh của hệ thống sử dụng các lớp đối tượng và các mối quan hệ giữa chúng. Các mối quan hệ quan trọng có thể được tài liệu hóa trong giai đoạn này là các mối quan hệ kế thừa và khái quát hóa, sử dụng/được sử dụng bởi các mối quan hệ và các mối quan hệ hợp thành.

- *Các mô hình động:* mô tả cấu trúc động của hệ thống và chỉ ra mối tương tác giữa các đối tượng hệ thống (không phải là các lớp đối tượng). Các mối tương tác có thể được tài liệu hóa bao gồm thứ tự của dịch vụ yêu cầu được thực hiện bởi các đối tượng và cách thức mà trạng thái của hệ thống liên quan tới các tương tác đối tượng này.

UML cung cấp 12 mô hình động và tĩnh khác nhau để giúp cho quá trình thiết kế. Phần này không giới thiệu tất cả các mô hình, mà chỉ giới thiệu những mô hình thích hợp cho ví dụ trạm dự báo thời tiết, đó là các mô hình sau:

- ***Mô hình hệ thống con***

Các mô hình hệ thống con chỉ ra việc nhóm một cách logic các đối tượng vào những hệ thống con. Điều này được thể hiện bằng cách sử dụng một mẫu sơ đồ các lớp, trong đó mỗi hệ thống con là một gói. Một mô hình hệ thống con là một mô hình tĩnh hữu ích khi nó chỉ ra cách thức nhóm các đối tượng có liên quan vào các nhóm. Hình 6.21 chỉ ra các hệ thống con trong hệ thống bản đồ thời tiết. Các gói UML thể hiện tính cấu trúc và không phản ánh trực tiếp các đối tượng sẽ được phát triển trong hệ thống.



H **g**

Hình vẽ trên chỉ r khí tượng. Mô hình trên cũng chỉ ra mối liên kết giữa các đối tượng trong mô hình. Ví dụ, đối tượng CommsController được kết hợp với đối tượng WeatherStation và đối tượng WeatherStation được kết hợp vào gói Data collection. Điều này có nghĩa là đối tượng này được kết hợp với một hoặc nhiều đối tượng khác trong gói. Một mô hình gói kết hợp với một mô hình lớp đối tượng có thể mô tả các nhóm logic trong hệ thống.

- ***Mô hình tuần tự***

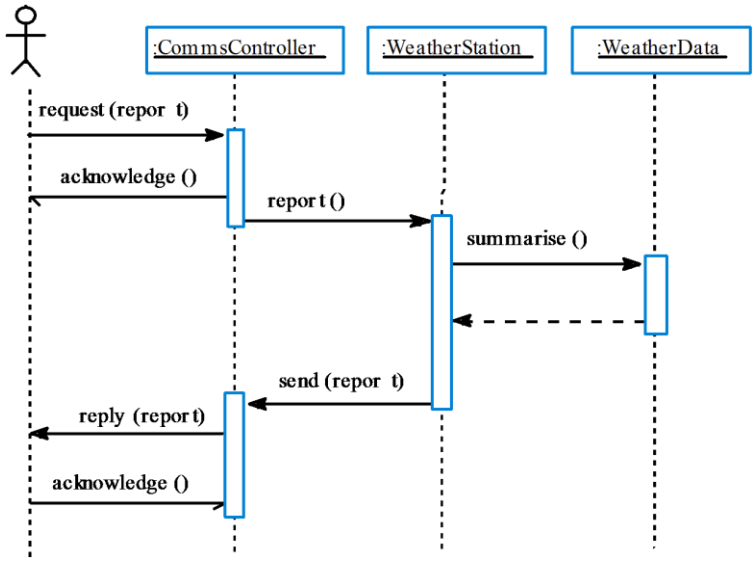
Các mô hình tuần tự chỉ ra thứ tự các tương tác của đối tượng. Các mô hình này được thể hiện bằng cách sử dụng một lược đồ tuần tự hoặc song song. Các mô hình tuần tự là các mô hình động, thể hiện một tương tác trong hệ thống và thứ tự của các tương tác này. Trong một mô hình tuần tự:

- Các đối tượng liên quan trong tương tác này được bố trí theo chiều ngang với một đường thẳng đứng được liên kết với từng đối tượng.

- Thời gian được biểu diễn theo chiều thẳng đứng, trình tự tính từ trên xuống dưới.

- Các mũi tên được gán nhãn kết nối giữa các trục thẳng đứng biểu diễn các tương tác giữa các đối tượng. Đây không phải là các luồng dữ liệu mà nó biểu diễn các thông điệp hoặc những sự kiện quan trọng của tương tác.

- Các hình chữ nhật rỗng biểu diễn thời gian mà một đối tượng kiểm soát một đối tượng khác trong hệ thống. Một đối tượng lấy quyền điều khiển ở trên đỉnh của hình chữ nhật và trả lại quyền điều khiển cho đối tượng khác tại điểm đáy của hình chữ nhật. Nếu có sự kế thừa trong lời gọi, điều khiển sẽ không được trả lại cho tới khi kết quả cuối cùng của lời gọi phương thức đầu tiên được hoàn thành.



Hình 6.22. Mô hình tuần tự của phương thức report() trong hệ thống trạm khí tượng

Hình 6.22 chỉ ra thứ tự của các tương tác khi hệ thống bản đồ bên ngoài yêu cầu dữ liệu từ các trạm thời tiết. Ta có thể đọc sơ đồ này từ trên xuống dưới:

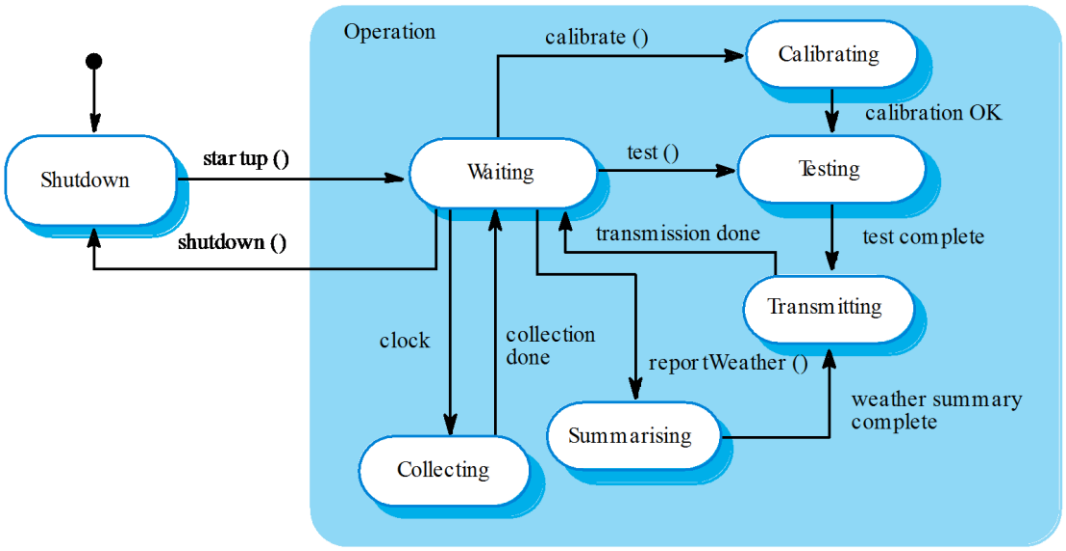
1. Một đối tượng là một sự kiện của lớp CommsController (:CommsController) nhận một tín hiệu từ môi trường (bên ngoài) yêu cầu thu thập dữ liệu thời tiết. Nó báo cho đối tượng gửi yêu cầu là đã nhận được tín hiệu yêu cầu này. Đầu mũi tên của thông điệp phản hồi chỉ ra rằng bên gửi thông điệp không mong chờ thông tin phản hồi.
2. Đối tượng này gửi một thông điệp tới một đối tượng là một sự kiện của WeatherStation để tạo ra một bản tin thời tiết. Sự kiện của CommsController sau đó rơi vào trạng thái treo. Kiểu mũi tên này được sử dụng để chỉ ra rằng một sự kiện đối tượng CommsController và một sự kiện của đối tượng WeatherStation là các đối tượng có thể thực thi đồng thời.
3. Đối tượng là một sự kiện của WeatherStation gửi một thông điệp tới đối tượng WeatherData để tóm lược dữ liệu thời tiết. Trong trường hợp này, mũi tên đặc chỉ ra rằng sự kiện của WeatherStation đang đợi câu trả lời.
4. Bản tóm tắt này được tính toán và điều khiển trả lại cho đối tượng WeatherStation. Đường mũi tên nét đứt chỉ ra việc trả điều khiển.
5. Đối tượng này gửi một thông điệp cho yêu cầu CommsController để truyền dữ liệu tới một hệ thống từ xa. Đối tượng WeatherStation sau đó rơi vào trạng thái treo.
6. Đối tượng CommsController gửi dữ liệu đã được tóm lược tới hệ thống từ xa, nhận một thông tin phản hồi, sau đó rơi vào trạng thái treo, đợi yêu cầu tiếp theo.

Từ lược đồ tuần tự này, chúng ta có thể nhận thấy rằng đối tượng CommsCotroller và đối tượng WeatherStation là những tiến trình song song, nơi thực thi có thể bị treo hoặc trở về trạng thái ban đầu. Thực ra, sự kiện của đối tượng CommsController nghe thông điệp từ hệ thống bên ngoài, giải mã thông điệp này và khởi tạo các phương thức của trạm thời tiết.

- ***Mô hình trạng thái máy***

Các mô hình trạng thái máy chỉ ra cách thức một đối tượng thay đổi trạng thái của mình khi phải trả lời các sự kiện. Các mô hình này được biểu diễn bằng các biểu đồ trạng thái. Các mô hình trạng thái là các mô hình động được sử dụng cho mô hình kết hợp các hoạt động của một nhóm đối tượng, nhưng cũng có thể sử dụng để tóm lược hoạt động của một đối tượng đơn khi nó trả lời một thông điệp mà nó phải xử lý. Để làm được điều này, ta có thể sử dụng mô hình trạng thái máy để chỉ ra các đối tượng thay đổi trạng thái như thế nào phụ thuộc vào thông điệp mà nó nhận được. UML sử dụng các lược đồ trạng thái để mô tả các mô hình trạng thái.

Hình 6.23 là một lược đồ trạng thái cho đối tượng WeatherStation chỉ ra làm thế nào nó trả lời các yêu cầu của các dịch vụ khác nhau.



S

1. Nếu trạng thái của đối tượng là *Shutdown* thì sau đó nó chỉ có thể trả lời thông điệp *startup()*, nó sẽ được chuyển sang trạng thái chờ thông điệp tiếp theo được gửi đến. Đường mũi tên không gán nhãn và có hình tròn ở đầu chỉ ra rằng trạng thái *Shutdown* là trạng thái khởi đầu.
2. Trong trạng thái chờ, hệ thống chờ đợi thông điệp tiếp theo. Nếu nó nhận được thông điệp *shutdown()*, thì đối tượng trở về trạng thái *Shutdown*.
3. Nếu nhận được thông điệp *reportWeather()*, hệ thống sẽ chuyển sang trạng thái *Summarising*. Khi hoàn thành việc tóm lược dữ liệu, hệ thống chuyển sang trạng thái *Transmiting*, thông tin được chuyển qua *CommsController*, nó quay trở lại trạng thái chờ.
4. Nếu nó nhận được thông điệp *calibrate()*, hệ thống sẽ chuyển sang trạng thái *Calibrating*, sau đó sang trạng thái *Testing* và cuối cùng là chuyển sang trạng thái *Transmitting* trước khi trở về trạng thái *Waiting*. Nếu nó nhận được thông điệp *test()*, hệ thống sẽ chuyển ngay sang trạng thái *Testing*.

5. Nếu có một tín hiệu từ đồng hồ, hệ thống sẽ chuyển sang trạng thái *Collecting*, nó tiến hành thu thập dữ liệu từ các thiết bị. Mỗi thiết bị được chỉ thị để trả lại kết quả thu thập được.

f. Đặc tả giao diện đối tượng

Một phần quan trọng trong bất kỳ tiến trình thiết kế nào là việc đặc tả giao diện giữa các thành phần trong bản thiết kế. Giao diện của các đối tượng và các hệ thống con cần phải đặc tả trước vì chúng có thể được thiết kế song song. Khi một giao diện được đặc tả, người xây dựng các đối tượng khác có thể cho rằng giao diện này sẽ được giữ nguyên trong giai đoạn lập trình.

Trong quá trình đặc tả giao diện, cố gắng tránh thêm các chi tiết của việc biểu diễn giao diện trong bản thiết kế giao diện. Việc biểu diễn nên để ẩn và các phương thức của đối tượng được cung cấp để truy nhập và cập nhật dữ liệu. Nếu việc biểu diễn được ẩn đi, nó có thể được thay đổi mà không ảnh hưởng tới những đối tượng có sử dụng các thuộc tính này. Điều này khiến cho bản thiết kế dễ bảo trì hơn. Ví dụ, biểu diễn mảng của một ngăn xếp có thể được thay đổi thành biểu diễn danh sách mà không ảnh hưởng tới các đối tượng khác có sử dụng ngăn xếp này. Ngược lại, trong một mô hình thiết kế tĩnh, nó thường thể hiện các thuộc tính của đối tượng vì đây là cách nhỏ gọn nhất để minh họa các tính chất cơ bản của đối tượng.

Không nhất thiết phải có mối quan hệ đơn giản 1-1 giữa các đối tượng và các giao diện. Cùng một đối tượng có thể có vài giao diện khác nhau, mỗi giao diện lại chỉ ra các phương thức mà nó cung cấp. Điều này được hỗ trợ trực tiếp trong Java, giao diện được khái báo riêng rẽ với các đối tượng và các đối tượng "thực hiện" các giao diện. Một nhóm các đối tượng có thể được truy nhập thông qua một giao diện.

Việc thiết kế giao diện liên quan đến đặc tả chi tiết giao diện cho một đối tượng hoặc một nhóm các đối tượng. Điều này có nghĩa là xác định các ký hiệu và ngữ nghĩa của các dịch vụ được cung cấp bởi đối tượng hoặc bởi một nhóm các đối tượng.

Các giao diện có thể được đặc tả trong ngôn ngữ UML bằng cách sử dụng các khái niệm giống như trong lược đồ lớp. Tuy nhiên, không có phần thuộc tính và ký hiệu <interface> nên được thêm vào trong phần tên lớp.

```
interface WeatherStation {

        public void WeatherStation () ;

        public void startup () ;
        public void startup (Instrument i) ;

        public void shutdown () ;
        public void shutdown (Instrument i) ;

        public void reportWeather ( ) ;

        public void test () ;
        public void test ( Instrument i ) ;

        public void calibrate ( Instrument i) ;

        public int getID () ;

} //WeatherStation
```

Hình 6.24. Giao diện của đối tượng WeatherStation

Có thể sử dụng cách tiếp cận xen kẽ, đó là việc sử dụng một ngôn ngữ lập trình để biểu diễn. Điều này được minh họa trong hình 6.24, giao diện của đối tượng trạm thời tiết được biểu diễn bằng ngôn ngữ lập trình Java. Khi các giao diện phức tạp hơn, thì cách biểu diễn này sẽ hiệu quả hơn, bởi việc kiểm tra cú pháp sẽ dễ dàng hơn khi biên dịch, do đó có thể dùng để khám phá ra những lỗi và sự mâu thuẫn trong việc mô tả các giao diện. Việc mô tả bằng Java có thể chỉ ra một số phương thức có thể lấy số tham số khác nhau. Tuy nhiên, phương thức *shutdown()* có thể được áp dụng cho trạm không có tham số hoặc có một thiết bị.

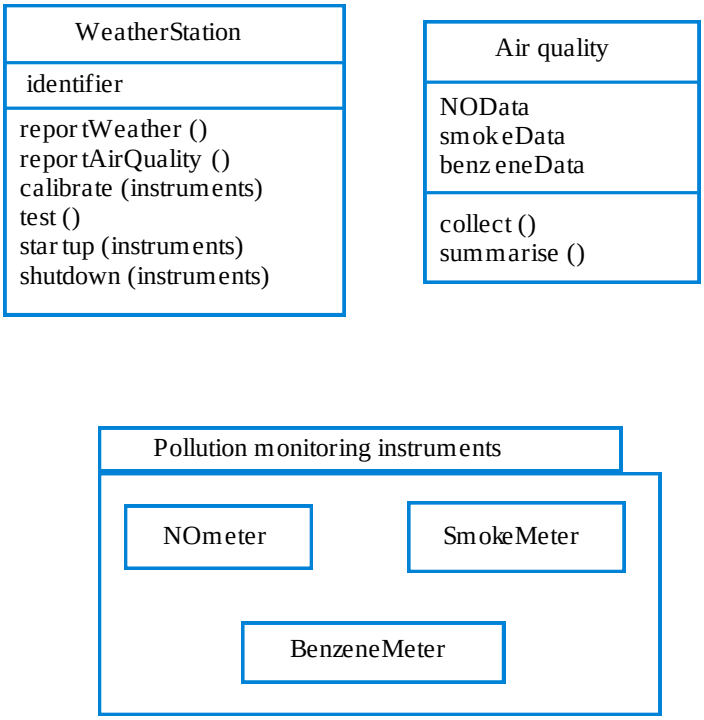
6.3.4. Cải tiến và tái sử dụng bản thiết kế

Sau khi có quyết định xây dựng một hệ thống, chẳng hạn như hệ thống thu thập dữ liệu thời tiết, điều không tránh khỏi là có thể có những yêu cầu thay đổi hệ thống được đề xuất. Một ưu điểm quan trọng của cách tiếp cận hướng đối tượng trong thiết kế là thay đổi thiết kế khá đơn giản. Lý do là việc biểu diễn trạng thái của đối tượng không ảnh hưởng tới thiết kế. Việc thay đổi những chi tiết bên trong của một đối tượng không ảnh hưởng tới các đối tượng khác trong hệ thống. Hơn nữa, do các đối tượng được liên kết với nhau một cách lỏng lẻo, nên thường không phức tạp khi thêm một đối tượng mới mà không ảnh hưởng tới phần còn lại của hệ thống.

Để chỉ ra làm thế nào để việc thiết kế hướng đối tượng có thể thay đổi một cách dễ dàng, giả sử rằng các tính năng kiểm soát mức độ ô nhiễm được tích hợp thêm vào trạm thời tiết. Điều này liên quan đến việc thêm một máy đo chất lượng không khí để tính toán độ ô nhiễm trong không khí. Yêu cầu thu thập dữ liệu về mức độ ô nhiễm được truyền cùng thời điểm với dữ liệu thời tiết.

Để thay đổi bản thiết kế, thì phải thay đổi một số vấn đề sau:

1. Một lớp đối tượng được gọi là *"Airquality"* được xem như là một phần của WeatherStation tại cùng mức với WeatherData.
2. Phương thức *"reportAirQuality"* nên được thêm vào WeatherStation để gửi thông tin về độ ô nhiễm tới máy tính trung tâm. Phần mềm điều khiển trạm thời tiết phải được thay đổi vì việc đọc độ ô nhiễm được thu thập một cách tự động khi được yêu cầu bởi đối tượng ở mức đỉnh *"WeatherStation"*.
3. Các đối tượng biểu diễn các kiểu thiết bị kiểm soát độ ô nhiễm cũng được bổ xung. Trong trường hợp này, các mức độ của khí nito, oxy, khói, benzen, cũng có thể đo được.



Hình 6.25. Cá iễm

Các đối tượng ki ược gọi là các *thiết bị kiểm soát độ ô n* có mối liên hệ với AirQuality và Weath được sử dụng để thu thập dữ liệu thời tiet. Hình 6.25 chỉ ra các phương thức mới được them vào đối tượng WeatherStation và các đối tượng mới được bổ sung vào hệ thống, không có sự thay đổi phần mềm được yêu cầu của các đối tượng ban đầu trong trạm khí tượng. Việc thêm dữ liệu về ô nhiễm không hề ảnh hưởng tới việc thu thập dữ liệu thời tiết.

CÂU HỎI ÔN TẬP

1. Hãy nêu khái niệm và vai trò của thiết kế trong phát triển phần mềm.
2. Phân tích các khái niệm trong thiết kế?
3. Trình bày các bước và các sản phẩm trong giai đoạn thiết kế. Người ta có thể sử dụng công cụ gì để biểu diễn các sản phẩm trong giai đoạn thiết kế.
4. Hãy nêu những điểm quan trọng để đánh giá chất lượng thiết kế. Có những phương pháp nào để kiểm soát chất lượng thiết kế?
5. Hãy phân tích các hướng dẫn thiết kế.
6. Trình bày vai trò của thiết kế kiến trúc. Thiết kế kiến trúc ảnh hưởng thế nào tới các yêu cầu phi chức năng của hệ thống?
7. Thiết kế kiến trúc được chia làm mấy giai đoạn? Hãy nêu những nét khái quát của từng giai đoạn đó.
8. Có mấy kiểu mô hình kiến trúc cơ bản? Hãy nêu những nét chính, ưu điểm và nhược điểm của từng mô hình. Một hệ thống có thể áp dụng nhiều mô hình kiến trúc hay chỉ được lựa chọn một mô hình kiến trúc duy nhất?
9. Mô hình hóa điều khiển xác định cách thức các hệ thống con trao đổi thông tin và thực hiện các dịch vụ của hệ thống. Có mấy loại mô hình điều khiển cơ bản? Những yếu tố nào ảnh hưởng tới việc lựa chọn mô hình điều khiển của hệ thống?
10. Có mấy chiến lược phân rã module? Việc lựa chọn chiến lược phân rã module dựa trên những đặc điểm của hệ thống hay thói quen của nhóm phát triển?
11. Chiến lược phát triển hướng đối tượng trải qua mấy bước, các bước đó có quan hệ với nhau như thế nào?
12. Hãy phân biệt hai khái niệm Đối tượng và Lớp đối tượng trong phát triển phần mềm hướng đối tượng.
13. Nêu những ưu điểm và nhược điểm của lập trình hướng đối tượng.
14. Trình bày các bước trong phân tích và thiết kế hướng đối tượng.
15. Tại sao người ta lại nói phương pháp thiết kế hướng đối tượng dễ dàng cải tiến và tái sử dụng các bản thiết kế.

Chương 7: KIỂM THỬ PHẦN MỀM

Kiểm thử và đảm bảo chất lượng phần mềm là một quá trình liên tục, xuyên suốt các giai đoạn phát triển nhằm đảm bảo phần mềm được tạo ra thỏa mãn các yêu cầu thiết kế và các yêu cầu đó đáp ứng được nhu cầu của người sử dụng. Kiểm thử phần mềm là một hoạt động rất tốn kém về thời gian cũng như tiền bạc và khó phát hiện được hết lỗi. Vì vậy, kiểm thử phần mềm đòi hỏi phải có chiến lược phù hợp, kế hoạch hợp lý và việc thực hiện phải được quản lý một cách chặt chẽ, nghiêm túc, hiệu quả.

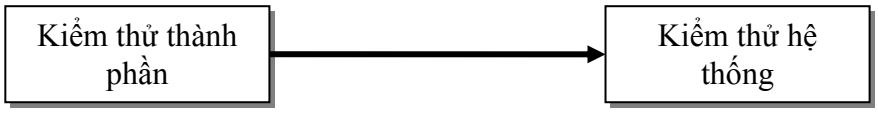
Chương này giới thiệu một số khái niệm, mục tiêu và các giai đoạn cơ bản trong tiến trình kiểm thử. Phần quan trọng của chương trình bày về các phương pháp xây dựng kịch bản và trường hợp kiểm thử. Qua đó sinh viên có thể nắm được những vấn đề cơ bản và áp dụng để kiểm thử các phần mềm do mình phát triển. Phần cuối cùng của chương giới thiệu những đặc điểm cơ bản và một số module chính của các công cụ kiểm thử tự động.

7.1. GIỚI THIỆU CHUNG

Một tiến trình kiểm thử thường bắt đầu bằng việc kiểm thử các đơn vị chương trình riêng rẽ, chẳng hạn như các chức năng hoặc đối tượng. Sau đó chúng được tích hợp vào các hệ thống và sự tương tác giữa các đơn vị được kiểm thử. Cuối cùng, sau khi bàn giao hệ thống, khách hàng có thể thực hiện một loạt các kiểm thử chấp nhận để đảm bảo rằng hệ thống thực thi theo đúng đặc tả.

Mô hình này của tiến trình kiểm thử thích hợp cho việc phát triển các hệ thống lớn, nhưng đối với các hệ thống nhỏ, hoặc với các hệ thống phát triển theo mô hình mẫu hoặc tái sử dụng thường khó phân biệt một cách rõ ràng các giai đoạn trong tiến trình phát triển phần mềm.

Một cách nhìn trìu tượng hơn về việc kiểm thử được chỉ ra trong hình 7.1. Hai hoạt động kiểm thử cơ bản là kiểm thử thành phần – kiểm thử một phần của hệ thống, và kiểm thử hệ thống – kiểm thử các hoạt động của toàn bộ hệ thống.



Người phát triển phần mềm Nhóm kiểm thử độc lập

Hình 7.1. Các giai đoạn kiểm thử

Mục tiêu của kiểm thử thành phần là khám phá ra các lỗi bằng cách kiểm thử các thành phần chương trình riêng rẽ. Những thành phần này có thể là các chức năng, các đối tượng hoặc các thành phần tái sử dụng.

Trong quá trình kiểm thử hệ thống, các thành phần được tích hợp với nhau tạo thành các hệ thống con hoặc một hệ thống hoàn chỉnh. Trong giai đoạn này, kiểm thử hệ thống cần tập trung vào việc đánh giá xem hệ thống có đáp ứng được các yêu cầu chức năng, phi chức năng và những hoạt động không theo mong muốn. Một điều không thể tránh khỏi, đó là các lỗi trong các thành phần có thể bị bỏ qua trong giai đoạn kiểm thử trước đó và chỉ được phát hiện khi kiểm thử hệ thống.

7.1.1. Mục tiêu của kiểm thử

Kiểm thử phần mềm đáp ứng 2 mục tiêu cơ bản:

- Để chứng minh với nhà phát triển và khách hàng là phần mềm đã đáp ứng đúng các yêu cầu đặt ra. Với khách hàng, điều này có nghĩa là nên có ít nhất một thử nghiệm cho mỗi yêu cầu của người sử dụng trong tài liệu yêu cầu. Với những sản phẩm phần mềm dùng chung, cần phải kiểm thử tất cả các chức năng của hệ thống sẽ được tích hợp vào sản phẩm cuối cùng. Một số hệ thống có thể cần giai đoạn kiểm thử tích hợp độc lập để khách hàng kiểm tra xem hệ thống được bàn giao có đúng như đặc tả không.

- Để khám phá ra các lỗi trong phần mềm hoặc những điểm yếu như cách hoạt động của phần mềm không đúng, không thỏa mãn hoặc không phù hợp với đặc tả. Phát hiện lỗi liên quan tới việc khám phá ra nguyên nhân của tất các hoạt động không đúng của hệ thống, chẳng hạn như hệ thống bị phá hủy, những tương tác không mong muốn với các hệ thống khác, việc tính toán sai và mất dữ liệu.

Mục tiêu đầu tiên được thực hiện bằng kiểm thử thẩm định, trong đó người kiểm thử mong hệ thống thực hiện một cách đúng đắn bằng việc đưa ra một tập hợp các trường hợp kiểm thử (test cases) phản ánh những mong muốn của người sử dụng đối với hệ thống.

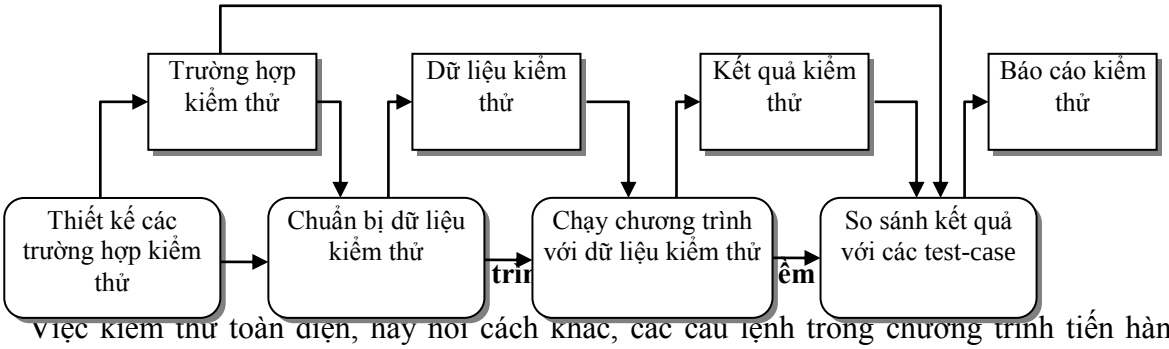
Mục tiêu thứ hai là phát hiện khiếm khuyết còn tồn tại, khi đó các trường hợp kiểm thử được thiết kế để bộc lộ các khiếm khuyết của phần mềm:

- Các trường hợp kiểm thử có thể bị che khuất một cách có chủ ý và không phản ánh xem hệ thống được sử dụng một cách thông thường như thế nào.
- Đối với việc kiểm thử thẩm định, một trường hợp kiểm thử thành công là khi hệ thống thực thi đúng. Còn đối với kiểm thử khiếm khuyết, một kiểm thử thành công là một khiếm khuyết được bộc lộ khi hệ thống thực thi không đúng.

Nhìn chung, kiểm thử không thể chứng mình rằng phần mềm hoàn toàn không còn lỗi hoặc nó thích nghi được trong mọi môi trường làm việc. Luôn luôn tồn tại các trường hợp kiểm thử mà bạn có thể khám phá ra các lỗi khác của hệ thống. Tuy nhiên, mục tiêu của kiểm thử phần mềm là tạo niềm tin cho khách hàng, rằng phần mềm đủ tốt và có thể sử dụng được. Kiểm thử là quá trình làm tăng độ tin cậy của phần mềm.

7.1.2. Tiến trình kiểm thử phần mềm

Một mô hình kiểm thử nói chung được mô tả trong hình 7.2. Các trường hợp kiểm thử là các bản đặc tả về đầu vào để kiểm thử và đầu ra được mong đợi từ hệ thống đối với mỗi chức năng của hệ thống. Dữ liệu kiểm thử là dữ liệu mà ta có thể đưa ra nhằm kiểm thử hệ thống. Dữ liệu kiểm thử đôi khi được sinh ra một cách tự động. Tuy nhiên, việc sinh ra các trường hợp kiểm thử tự động là không thể. Đầu ra của kiểm thử có thể chỉ là một phán đoán của người sử dụng hoặc người hiểu hệ thống cần phải hoạt động như thế nào.



Việc kiểm thử toàn diện, nay nói cách khác, các câu lệnh trong chương trình tiến hành tuần tự đều được kiểm thử là điều không thể. Tuy nhiên, kiểm thử dựa trên một tập con các trường hợp kiểm thử là có thể. Một cách lý tưởng, các công ty phần mềm nên có những chiến lược để lựa chọn tập dữ liệu kiểm thử hơn là để cho nhóm phát triển. Những chiến lược này có thể dựa trên các chiến lược kiểm thử chung (ví dụ, một chiến lược kiểm thử mà tất cả các câu lệnh của chương trình cần phải được thực thi ít nhất một lần). Các chiến lược có thể dựa trên kinh nghiệm sử dụng hệ thống và có thể tập trung vào những tính năng của hệ thống, chẳng hạn như:

- Tất cả các chức năng của hệ thống được truy cập thông qua menu phải được kiểm thử.

- Việc kết hợp các chức năng (ví dụ như chức năng định dạng văn bản) được truy cập qua cùng một menu phải được kiểm thử.

- Với dữ liệu được người dùng cung cấp, tất cả các chức năng phải được kiểm thử đối với cả trường hợp dữ liệu đúng và dữ liệu sai.

Khi các tính năng của phần mềm được sử dụng tách biệt, thường thì chúng hoạt động tốt. Vấn đề sẽ nảy sinh khi các tính năng tích hợp không được kiểm thử cùng nhau.

Một phần của tiến trình lập kế hoạch kiểm thử, người quản lý phải quyết định ai là người chịu trách nhiệm trong các giai đoạn khác nhau của tiến trình kiểm thử. Đối với hầu hết các hệ thống, người lập trình chịu trách nhiệm kiểm thử thành phần của đoạn mã nguồn mà họ đã tạo ra. Ngay khi nó hoàn thành, công việc được chuyển qua nhóm tích hợp, nhóm này chịu trách nhiệm tích hợp các tính năng được phát triển từ các nhóm khác nhau, hình thành nên phần mềm và kiểm thử toàn hệ thống. Đối với các hệ thống quan trọng, nhiều tiến trình hình thức được sử dụng khi những kỹ sư kiểm thử chịu trách nhiệm đối với tất cả các giai đoạn của tiến trình kiểm thử. Khi kiểm thử các hệ thống quan trọng, kịch bản kiểm thử được phát triển riêng rẽ và việc ghi lại các chi tiết được duy trì trong kết quả kiểm thử.

Kiểm thử thành phần bởi người phát triển thường dựa trên việc hiểu cặn kẽ về chức năng của thành phần. Tuy nhiên, kiểm thử hệ thống lại phải dựa trên một bản đặc tả hệ thống. Nó có thể là một bản đặc tả yêu cầu chi tiết của hệ thống.

Trong thực tiễn, khi tiến hành kiểm thử người ta thường bắt đầu kiểm thử thành phần, sau đó mới chuyển sang giai đoạn kiểm thử hệ thống. Tuy nhiên, trong phần này hai nội dung lại được giới thiệu đảo ngược bởi ngày càng nhiều hệ thống được xây dựng bằng cách tích hợp các thành phần tái sử dụng và cấu hình lại hệ thống đang tồn tại để nó đáp ứng được yêu cầu của người dùng.

7.2. KIỂM THỬ HỆ THỐNG

Kiểm thử hệ thống liên quan tới việc tích hợp hai hoặc nhiều thành phần để thực thi các chức năng của hệ thống, sau đó kiểm thử toàn bộ hệ thống đã được tích hợp.

Trong tiến trình phát triển lặp, kiểm thử hệ thống liên quan tới việc kiểm thử một thành phần (increment) được bàn giao cho khách hàng; trong mô hình thác nước, kiểm thử hệ thống liên quan tới kiểm thử hệ thống hoàn chỉnh.

Với hầu hết các hệ thống phức tạp, kiểm thử hệ thống được chia thành 2 giai đoạn:

1. *Kiểm thử tích hợp:* Hệ thống được kiểm thử khi các thành phần được tích hợp vào hệ thống. Nhóm kiểm thử có thể truy cập vào mã nguồn chương trình. Khi một vấn đề được phát hiện, nhóm sẽ cố gắng để tìm ra nguồn gốc của vấn đề và xác định những thành phần có thể gây lỗi. Kiểm thử tích hợp hầu hết đều liên quan đến việc tìm ra những khiếm khuyết của hệ thống.
2. *Kiểm thử phát hành:* là giai đoạn một phiên bản của hệ thống hoàn thành và được bàn giao cho khách hàng dùng thử. Giai đoạn này liên quan tới việc thẩm định xem hệ thống có đáp ứng đúng các yêu cầu và chắc chắn rằng hệ thống là đáng tin cậy. Kiểu kiểm thử này thường gọi là kiểm thử hộp đen, nghĩa là nhóm làm kiểm thử chỉ ra cho nhóm phát triển thấy phần mềm này làm việc hợp lý hay không hợp lý. Khi khách hàng có liên quan tới giai đoạn kiểm thử này thì người ta gọi là kiểm thử chấp nhận. Nếu phiên bản đủ tốt, người sử dụng có thể chấp nhận sử dụng nó.

Về cơ bản, có thể cho rằng việc kiểm thử tích hợp là kiểm thử một nhóm các thành phần của hệ thống một cách không đầy đủ. Kiểm thử phát hành liên quan tới việc kiểm thử một hệ thống được bàn giao cho khách hàng. Hiển nhiên, hai kiểu kiểm thử này có những điểm trùng lặp, đặc biệt là khi phát triển hệ thống theo kiểu gia tăng và hệ thống được chuyển giao là chưa hoàn thiện. Nói chung, kiểm thử tích hợp ưu tiên việc khám phá ra những khiếm khuyết của hệ thống, còn kiểm thử hệ thống liên quan tới việc đáp ứng những yêu cầu. Tuy nhiên, trong thực tế, có một số kiểm thử thẩm định và kiểm thử khiếm khuyết được tiến hành trong cả hai tiến trình này.

7.2.1. Kiểm thử tích hợp

Tiến trình kiểm thử tích hợp liên quan tới việc xây dựng một hệ thống từ các thành phần, sau đó kiểm tra hệ thống đối với những vấn đề nảy sinh từ việc tương tác giữa các thành phần. Các thành phần được tích hợp có thể là các gói phần mềm thương mại, các thành phần tái sử dụng đã được phát triển cho một hệ thống cụ thể hoặc một thành phần mới được phát triển. Với các hệ thống lớn, cả ba kiểu thành phần này đều có thể được sử dụng. Kiểm thử tích hợp kiểm tra xem các thành phần này làm việc với nhau như thế nào, gọi và biến đổi dữ liệu đúng, tại những thời điểm thích hợp thông qua giao diện.

Tích hợp hệ thống liên quan đến việc xác định các nhóm thành phần cung cấp một số chức năng của hệ thống và việc tích hợp các hệ thống này là viết thêm những đoạn mã để chúng có thể làm việc cùng nhau.

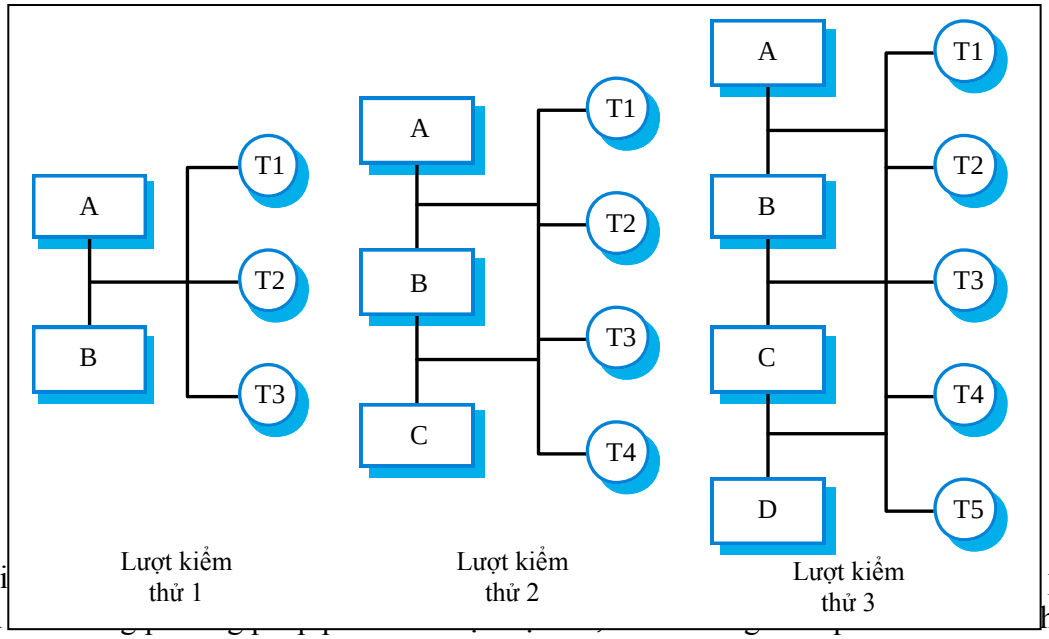
Đôi khi, bộ khung tổng thể của hệ thống được phát triển trước, các thành phần được bổ sung sau. Mô hình này được gọi là *tích hợp trên xuống*.

Ngược lại, ta có thể lựa chọn phương pháp tích hợp các thành phần cơ sở của hệ thống cung cấp các chức năng chung, chẳng hạn như mạng hoặc truy nhập CSDL, sau đó thêm các thành phần chức năng. Đây được gọi là phương pháp *tích hợp dưới lên*.

Trong thực tế, với nhiều hệ thống, các chiến lược tích hợp thường kết hợp cả hai phương pháp này, cả những thành phần cơ sở hạ tầng và các thành phần chức năng được thêm vào một cách gia tăng. Với cả hai phương pháp này, thường phải phát triển thêm mã nguồn để mô phỏng các thành phần khác và cho phép hệ thống thực thi.

Vấn đề chính nảy sinh trong quá trình tích hợp là việc xác định vị trí lỗi. Việc tương tác phức tạp giữa các thành phần hệ thống và khi một đầu ra bất thường được khám phá, có thể rất khó để xác định nơi xảy ra lỗi. Để dễ dàng hơn, người ta thường sử dụng cách tiếp cận gia tăng để tích hợp và thử nghiệm. Đầu tiên, tích hợp một cấu hình hệ thống tối thiểu và kiểm thử hệ thống này, sau đó lần lượt thêm các thành phần và kiểm thử sau từng bước tích hợp.

Ví dụ trong hình 7.3, A, B, C, D là các thành phần và T1-T5 là tập hợp các kịch bản kiểm thử của các chức năng được kết hợp trong hệ thống. T1, T2, T3 đầu tiên được thực thi trên một hệ thống bao gồm 2 thành phần A, B. nếu phát hiện ra khiếm khuyết, chúng sẽ được sửa. Thành phần C được tích hợp thêm vào và T1-T3 được lặp lại để nó không có sự bất thường với A và B. Nếu có vấn đề nảy sinh trong các trường hợp kiểm thử này, có nghĩa là lỗi sinh ra khi tích hợp thành phần mới. Nguồn gốc của vấn đề sẽ được xác định, việc phát hiện và sửa lỗi đơn giản hơn. Trường hợp kiểm thử T4 cũng được thực thi với hệ thống mới. Và cuối cùng, thành phần D được tích hợp vào hệ thống và sử dụng các kiểm thử đã có, cộng với một trường hợp kiểm thử mới (T5).



Khi phần. Chẳng hạn h phát triển và là người quyết định thành phần nào sẽ được ưu tiên tích hợp trước. Trong cách tiếp cập tích hợp các thành phần thương mại, khách hàng có thể không liên quan và nhóm tích hợp sẽ quyết định thứ tự ưu tiên.

Trong trường hợp đó, thông thường người ta sẽ chọn những thành phần có các chức năng được sử dụng thường xuyên nhất. Ví dụ: trong hệ thống thư viện LIBSYS, bạn nên bắt đầu bằng

việc tích hợp chức năng tìm kiếm. Và sau đó có thể thêm chức năng cho phép người sử dụng download tài liệu, cuối cùng là thêm các thành phần để thực hiện các chức năng khác.

Tất nhiên, trong thực tế, việc tích hợp không đơn giản như mô hình gợi ý. Thực thi các tính năng của hệ thống có thể được tiến hành bởi nhiều thành phần khác nhau. Để kiểm tra một tính năng mới, bạn có thể phải tích hợp vài thành phần khác nhau. Việc kiểm thử có thể phát hiện ra những lỗi trong tích hợp các thành phần độc lập này. Việc sửa lỗi sẽ khó hơn vì có thể phải thay đổi một nhóm các thành phần liên quan tới việc thực thi chức năng này. Hơn nữa, việc tích hợp và kiểm thử một thành phần mới có thể thay đổi framework của những thành phần tích hợp đã được kiểm thử. Những lỗi có thể không bộc lộ trong những kiểm thử đối với cấu hình hệ thống đơn giản hơn.

Điều này có nghĩa là khi một thành phần mới được tích hợp, điều quan trọng là phải chạy lại các kịch bản kiểm thử đã được thực thi trước đó như một kịch bản mới, đây là điều bắt buộc để xác nhận một chức năng mới của hệ thống. Việc chạy lại các kịch bản trước được gọi là kiểm thử ngược. Nếu kiểm thử ngược phát hiện ra vấn đề, ta phải kiểm tra xem vấn đề phát sinh từ đâu trong các thành phần trước mà việc tích hợp thêm thành phần mới làm xuất hiện lỗi.

Việc kiểm thử ngược là một công việc tốn kém và thường là không thể thực thi được nếu không có các công cụ kiểm thử tự động hỗ trợ. Trong phương pháp lập trình cực đại (XP), tất cả các kịch bản kiểm thử được viết trước khi thực thi mã nguồn. Khi đó dữ liệu kiểm thử và kết quả mong đợi được đặc tả và có thể sử dụng phương pháp kiểm thử tự động. Khi sử dụng một framework kiểm thử tự động, chẳng hạn như JUnit, các kịch bản kiểm thử có thể chạy lại một cách tự động. Phương pháp này là một đặc trưng của phương pháp lập trình cực đại, một tập hợp kịch bản kiểm thử hoàn thiện được tiến hành ngay khi có mã nguồn mới được tích hợp và mã nguồn mới này không được chấp nhận cho tới khi tất cả các kịch bản kiểm thử thực thi thành công.

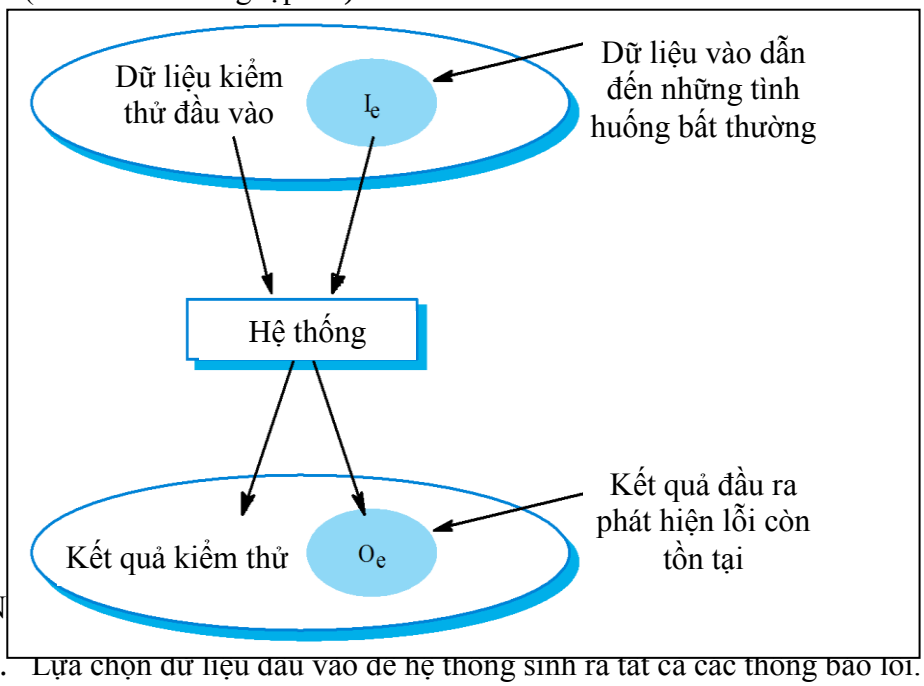
7.2.2. Kiểm thử phát hành

Kiểm thử phát hành là một tiến trình kiểm thử việc phát hành của hệ thống sẽ được chuyển giao cho khách hàng. Mục tiêu chính của tiến trình kiểm thử này là làm tăng độ tin cậy của nhà cung cấp, chứng tỏ rằng hệ thống đáp ứng được những yêu cầu khách hàng đặt ra. Nếu đúng, nhóm phát triển có thể phát hành sản phẩm hoặc chuyển giao cho khách hàng. Để chỉ ra rằng hệ thống đáp ứng đúng những yêu cầu đặt ra, ta phải chỉ ra rằng nó thực hiện theo đúng chức năng đã đặc tả, hiệu quả, tin cậy và không có lỗi khi sử dụng bình thường.

Kiểm thử phát hành thường là tiến trình kiểm thử hộp đen, các kịch bản kiểm thử được bắt nguồn từ đặc tả hệ thống. Hệ thống được xem như một hộp đen, các hoạt động của nó chỉ được xác định bởi việc nghiên cứu dữ liệu đầu vào và đầu ra có liên quan. Một tên khác của kiểu kiểm thử này là kiểm thử chức năng, bởi các kỹ sư kiểm thử chỉ quan tâm tới chức năng của sản phẩm, chứ không liên quan tới việc thực thi sản phẩm.

Hình 7.4 minh họa mô hình kiểm thử hộp đen. Các kỹ sư kiểm thử nhập dữ liệu đầu vào của thành phần hoặc hệ thống và kiểm tra xem kết quả đầu ra có phù hợp với mong đợi hay không. Nếu kết quả đầu ra không như dự đoán, việc kiểm thử sẽ phát hiện những vấn đề còn tồn tại của phần mềm.

Trong quá trình kiểm thử hệ thống phát hành, người kiểm thử nên cố gắng "phá hủy" hệ thống bằng cách lựa chọn các kịch bản kiểm thử nằm trong tập Ie trong hình 7.4. Điều đó có nghĩa là mục tiêu của kiểm thử là lựa chọn dữ liệu đầu vào sao cho có nhiều khả năng hệ thống sinh ra lỗi (đầu ra nằm trong tập Oe).



N sau:

1. Lựa chọn dữ liệu đầu vào để hệ thống sinh ra tất cả các thông báo lỗi.
2. Thiết kế dữ liệu đầu vào sao cho nó là nguyên nhân gây ra lỗi tràn bộ đệm đầu vào.
3. Lặp lại với cùng dữ liệu đầu vào hoặc một loạt các dữ liệu đầu vào ở nhiều thời điểm khác nhau.
4. Lựa chọn dữ liệu đầu vào để sinh ra các kết quả sai.
5. Kiểm thử đối với dữ liệu tính toán từ lớn đến nhỏ.

7.2.3. Xây dựng kịch bản kiểm thử hệ thống

Để xác minh hệ thống đáp ứng được những yêu cầu, cách kiểm thử tốt nhất là kiểm thử dựa trên các kịch bản, khi đưa ra một số kịch bản và phát triển các trường hợp kiểm thử từ những kịch bản này, ví dụ một kịch bản kiểm thử trong hệ thống phần mềm quản lý thư viện:

> Một sinh viên ở Scotlan theo học ngành lịch sử Hoa Kỳ cần viết một báo cáo về chiến tranh biên giới ở miền tây nước Mỹ từ năm 1840 – 1880. Để làm báo cáo này, anh ta cần tìm tài liệu ở các thư viện. Anh ta truy cập vào hệ thống thư viện LIBSYS và sử dụng tiện ích tìm kiếm để tìm ra những tài liệu cần thiết.Anh ta tìm được một số tài liệu trong các thư viện khác nhau ở Mỹ và tải về. Tuy nhiên, với một tài liệu, anh ta cần phải xác minh từ phía trường học của mình rằng tài liệu này được sử dụng vào mục đích phi thương mại. Sinh viên này sau đó sử dụng tiện ích trong hệ thống để xác minh thông tin. Nếu việc xác minh thành công, tài liệu sẽ được tải xuống máy chủ của thư viện đã được đăng ký và được in ra. Anh ta sẽ nhận được một tin nhắn từ hệ thống LIBSYS rằng anh ta sẽ nhận được một thông báo khi tài liệu được in xong.

Hình 7.5. Một kịch bản mô tả một chức năng của hệ thống thư viện LIBSYS

Từ kịch bản này, nó có thể sinh ra một số trường hợp kiểm thử có thể được ứng dụng cho mục đích kiểm thử phát hành của LIBSYS:

1. Kiểm thử việc đăng nhập với cả 2 trường hợp: đăng nhập đúng và đăng nhập sai để chứng minh rằng, người đăng nhập đúng sẽ được chấp nhận, còn người đăng nhập sai sẽ bị từ chối.
2. Kiểm thử chức năng tìm kiếm sử dụng một truy vấn vào các nguồn tài nguyên để kiểm tra xem chức năng tìm kiếm này có tìm ra được tài liệu cần dùng hay không.
3. Kiểm thử chức năng hiển thị để kiểm tra thông tin về tài liệu được hiển thị chính xác.
4. Kiểm thử chức năng tải tài liệu sử dụng một truy vấn cho phép người dùng tải tài liệu về.
5. Kiểm thử email trả lời để chỉ ra rằng tài liệu đã tải xuống đã sẵn sàng sử dụng.

Với mỗi trường hợp kiểm thử, ta nên thiết kế một tập các dữ liệu kiểm thử, bao gồm cả trường hợp dữ liệu đúng và sai. Ta cũng nên tổ chức việc kiểm thử dựa trên các kịch bản, do đó những trường hợp thích hợp sẽ được kiểm thử trước, sau đó những trường hợp bất thường hoặc ngoại lệ sẽ được xem xét sau, khi đó những thành phần được sử dụng thường xuyên sẽ được kiểm thử nhiều lần nhất.

Trong trường hợp đặc tả phần mềm có sử dụng mô hình trường hợp sử dụng (use-case) thì những use-case này được kết hợp với các lược đồ tuần tự có thể là cơ sở cho việc kiểm thử. Hai loại sơ đồ này có thể được sử dụng trong cả kiểm thử tích hợp và kiểm thử phát hành.

7.2.4. Kiểm thử hiệu năng

Khi một hệ thống được tích hợp hoàn chỉnh, có thể kiểm tra hệ thống với các thuộc tính nổi bật như hiệu năng và độ tin cậy. Kiểm thử hiệu năng được thiết kế để đảm bảo rằng hệ thống có thể xử lý việc tải dữ liệu có dụng ý. Loại kiểm thử này thường liên quan tới việc lập kế hoạch cho một loạt các trường hợp kiểm thử mà việc tải dữ liệu sẽ tăng dần cho tới khi hệ thống không chấp nhận nổi.

Như các kiểu kiểm thử khác, kiểm thử hiệu năng liên quan tới việc chứng minh rằng hệ thống đáp ứng được các yêu cầu của nó và khám phá ra các vấn đề và các khiếm khuyết trong hệ thống. Để kiểm thử yêu cầu hiệu năng của hệ thống đã hoàn thành, ta cần phải xây dựng một tập hợp các kịch bản kiểm thử phản ánh công việc hỗn hợp sẽ được thực hiện bởi hệ thống. Tuy nhiên, nếu 90% giao dịch trong hệ thống là kiểu A, 5% là kiểu B và phần còn lại là kiểu C, D, E, thì ta phải thiết kế tập các trường hợp kiểm thử mà phần lớn các trường hợp thuộc kiểu A. Mặt khác, sẽ không có một trường hợp kiểm thử chính xác về tính hiệu năng của hệ thống. Tuy nhiên, ta có thể kiểm thử trường hợp thực hiện hệ thống vượt ra ngoài giới hạn thiết kế của hệ thống.

Ví dụ: một hệ thống được thiết kế để xử lý 300 giao dịch trong một giây, một trường hợp kiểm thử có thể được thiết kế để xử lý trên 1000 giao dịch. Kiểm thử áp lực tiếp tục thực thi các trường hợp kiểm thử này cho tới khi hệ thống bị lỗi. Kiểm thử này có 2 chức năng:

1. *Kiểm thử lỗi hoạt động của hệ thống.* Các tình huống có thể nảy sinh qua việc hợp nhất các sự kiện không mong đợi khi lượng tải tối đa được hệ thống xử lý bị vượt quá. Trong trường hợp này, điều quan trọng là hệ thống lỗi không gây ra hiện tượng sai lệch dữ liệu hoặc mất các dịch vụ của người dùng.
2. *Kiểm thử gâp áp lực với hệ thống* và có thể dẫn đến những lỗi mà ta không thể phát hiện trong tình huống thông thường.

Kiểm thử áp lực thường liên quan tới các hệ thống phân tán trong môi trường mạng. Những hệ thống này thường xảy ra hiện tượng giảm hiệu suất khi bị quá tải. Mạng trở nên mất tác dụng khi phối hợp dữ liệu từ các bộ xử lý khác nhau. Vì thế tiến trình ngày càng chậm hơn khi phải chờ dữ liệu từ bộ xử lý khác.

7.3. KIỂM THỬ THÀNH PHẦN

Kiểm thử thành phần (đôi khi còn được gọi là kiểm thử đơn vị) là một tiến trình kiểm thử các thành phần độc lập trong hệ thống. Như đã thảo luận trong phần giới thiệu, đối với hầu hết các hệ thống, người phát triển thành phần chịu trách nhiệm kiểm thử thành phần do mình tạo ra. Các kiểu kiểm thử khác nhau được kiểm tra trong giai đoạn này là:

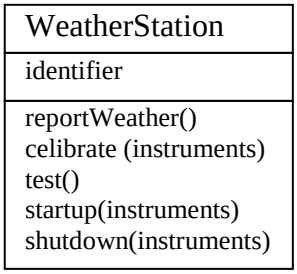
1. *Kiểm thử các chức năng hoặc các phương thức độc lập* của lớp đối tượng.
2. *Kiểm thử các lớp đối tượng* có một số thuộc tính và phương thức.
3. *Kiểm thử các thành phần hợp thành* từ các đối tượng hoặc chức năng khác nhau. Những thành phần hợp thành này có giao diện xác định được sử dụng để truy cập vào các chức năng của chúng.

7.3.1. Kiểm thử lớp đối tượng

Các chức năng hoặc phương thức độc lập là kiểu đơn giản nhất của thành phần và được kiểm thử như một tập những lời gọi các phương thức này từ các tham số đầu vào khác nhau. Ta có thể sử dụng cách tiếp cận này để thiết kế các trường hợp kiểm thử. Khi đang kiểm thử các lớp đối tượng, nên thiết kế các kịch bản kiểm thử bao trùm được tất cả các tính năng của đối tượng. Tuy nhiên, lớp đối tượng kiểm thử thường bao gồm:

1. Kiểm thử cách ly tất cả các chức năng của đối tượng.
2. Thiết lập và truy vấn tất cả các thuộc tính của đối tượng.
3. Kiểm tra đối tượng với tất cả các trạng thái có thể. Điều này có nghĩa là tất cả các sự kiện dẫn đến sự thay đổi trạng thái của đối tượng đều phải mô phỏng.

Ví dụ, hình 7.6 là giao diện của hệ thống trạm khí tượng được nêu ra ở chương trước. Đối tượng này chỉ có một thuộc tính xác định số hiệu của trạm, đây là một hằng số được đặt khi trạm được thiết lập. Tuy nhiên, thuộc tính này chỉ cần kiểm tra khi nó được thiết lập. Nhưng ta cần xây dựng các trường hợp kiểm thử các phương thức của đối tượng như: *reportWeather, calibrate…* Một cách lý tưởng, các phương thức phải kiểm tra một cách độc lập, nhưng trong một số trường hợp, kiểm thử tuần tự là cần thiết. Ví dụ, khi kiểm tra phương thức *shutdown*, cần phải thực thi phương thức *Start*.

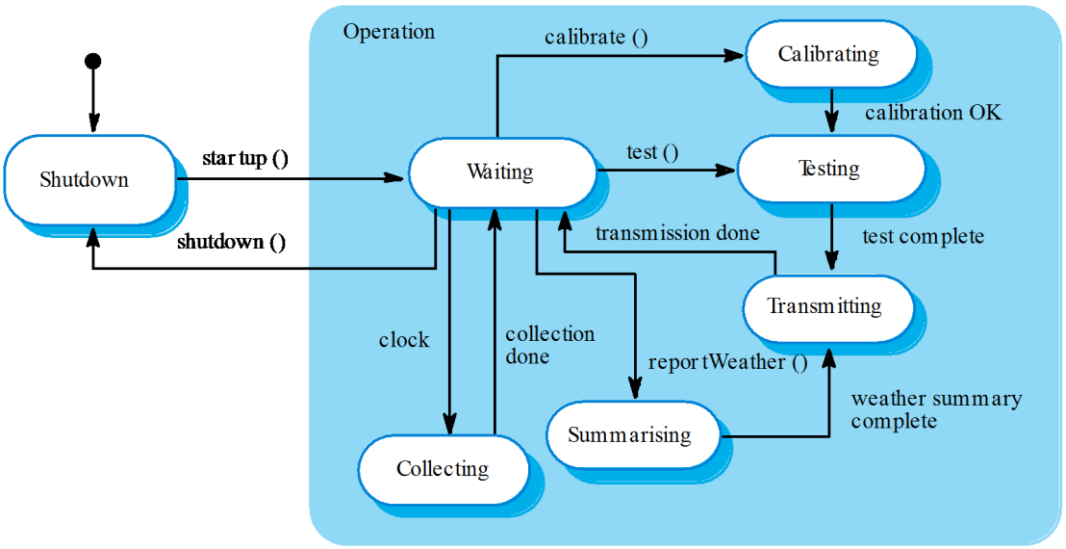


Hình 7 tion

Để kiểm tra trạng thái của trạm thời tiết, ta sử dụng mô hình trạng thái chỉ ra trong hình 7.7. Qua mô hình này ta có thể xác định thứ tự các biến đổi trạng thái, do đó phải kiểm tra và xác

định thứ tự các sự kiện gây ra sự biến đổi trạng thái. Về cơ bản, ta cần phải kiểm thử tất cả các trạng thái biến đổi có thể, mặc dù trên thực tế nó rất tốn kém.

Hì



:

Shutdown -> waiting -> Shutdown

Waiting ->calibrationg -> Testing -> transmiting – Waiting

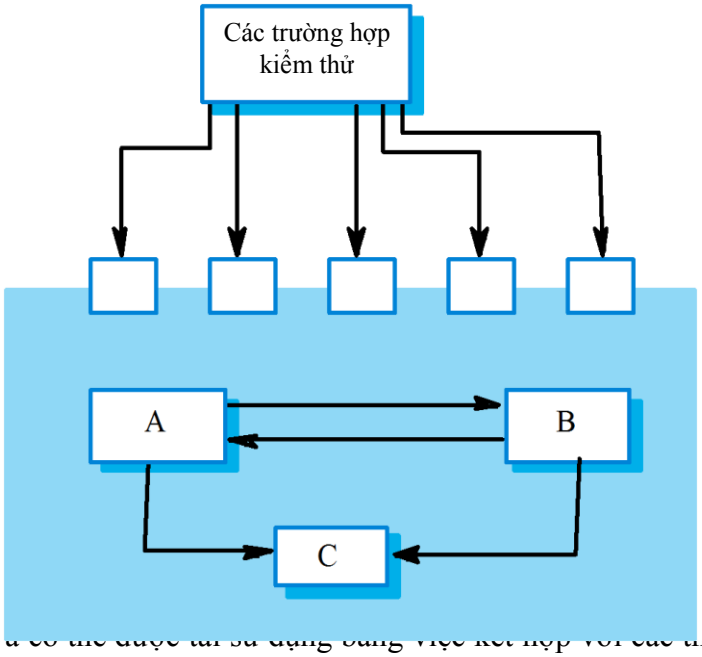
Waiting -> Collecting -> Waiting -> Summarising -> Transmitting -> Waiting

Nếu trong hệ thống có các lớp kế thừa thì sẽ khó thiết kế các trường hợp kiểm thử cho các lớp đối tượng. Khi một lớp cha cung cấp các phương thức được kế thừa cho các lớp con khác nhau, tất cả các lớp con phải được kiểm thử các phương thức kế thừa này. Lý do phải kiểm thử lại là những thuộc tính hoặc phương thức kế thừa này đã thay đổi trong các lớp con.

7.3.2. Kiểm thử giao diện

Nhiều thành phần trong hệ thống không phải là những chức năng hoặc đối tượng đơn giản mà nó được tích hợp từ một số đối tượng khác nhau. Khi đó ta cần truy nhập vào những chức năng của các thành phần sau đó kiểm tra xem giao diện của các thành phần này có tương hợp với đặc tả hay không.

Hình 7.8 minh họa tiến trình kiểm tra giao diện. Giả sử rằng các thành phần A, B, C được tích hợp để tạo ra những thành phần lớn hoặc các hệ thống con. Các trường hợp kiểm thử không được xây dựng cho từng thành phần độc lập nhưng giao diện của thành phần hợp thành được tạo ra bằng việc kết hợp các thành phần này.



Kiểm thử mềm theo phương pháp hướng đối tượng phần được xác định bởi giao diện của nó và có thể được tái sử dụng bằng việc kết hợp với các thành phần khác trong các hệ thống khác nhau. Những lỗi giao diện trong thành phần hợp thành không thể phát hiện khi kiểm thử các đối tượng hoặc thành phần độc lập. Những lỗi trong các thành phần hợp thành có thể phát sinh bởi sự tương tác giữa các thành phần trong nó.

Có các kiểu giao diện khác nhau giữa các thành phần chương trình, kết quả là có những lỗi giao diện khác nhau có thể xảy ra:

1. *Giao diện dạng tham số:* giao diện là dữ liệu hoặc đôi khi là chức năng được chuyển từ thành phần này sang thành phần khác.
2. *Giao diện bộ nhớ được chia sẻ:* là những giao diện mà các khối bộ nhớ được chia sẻ giữa các thành phần. Dữ liệu được đặt trong bộ nhớ bởi một hệ thống con và được đưa ra từ những hệ thống con khác.
3. *Giao diện thủ tục:* là những giao diện mà một thành phần ẩn trong nó một tập hợp các thủ tục có thể được gọi từ những thành phần khác. Các đối tượng và các thành phần có thể tái sử dụng có dạng giao diện này.
4. *Giao diện chuyển thông điệp:* những giao diện này thực hiện việc chuyển các thông điệp từ thành phần này sang thành phần khác. Một thông điệp trả về sẽ bao gồm kết quả của việc thực thi dịch vụ. Một số hệ thống hướng đối tượng có giao diện dạng này, chẳng hạn như hệ thống client-server.

Lỗi trong giao diện thường rơi vào 3 loại sau:

1. *Sử dụng sai giao diện:* một thành phần gọi thành phần khác qua giao diện và gặp lỗi trong sử dụng giao diện. Kiểu lỗi này thường gặp khi truyền các tham số sai kiểu, sai thứ tự.
2. *Hiểu sai giao diện:* một thành phần gọi một thành phần khác thông qua giao diện nhưng lại hiểu sai về hoạt động của thành phần đó, gọi thành phần không có những hoạt động mong đợi. Ví dụ: khi tìm kiếm nhị phân có thể được gọi trong một mảng không được sắp xếp, do đó việc tìm kiếm sẽ thực hiện sai.

3. *Lỗi thời gian:* lỗi này xảy ra trong các hệ thống thời gian thực có sử dụng việc chia sẻ bộ nhớ hoặc giao diện truyền thông điệp. Việc sinh ra và xử lý dữ liệu đôi khi không đồng bộ (tốc độ xử lý khác nhau).

Kiểm thử để phát hiện những khiếm khuyết của giao diện là khó vì đôi khi những lỗi giao diện này chỉ sinh ra trong các trường hợp sử dụng bất thường. Một vấn đề nữa nảy sinh do sự tương tác giữa các lỗi trong các module hoặc các đối tượng khác nhau. Lỗi trong một đối tượng có thể chỉ được phát hiện khi một đối tượng khác hoạt động không đúng.

Các kỹ thuật kiểm thử tĩnh thường hiệu quả hơn về mặt chi phí trong việc khám phá ra các lỗi giao diện. Một kiểu ngôn ngữ lập trình mạnh như Java cho phép nhiều lỗi giao diện được phát hiện khi biên dịch. Trong khi đó các ngôn ngữ yếu hơn, chẳng hạn như C, sử dụng phân tích tĩnh có thể giúp phát hiện các lỗi giao diện. Kiểm tra chương trình có thể tập trung vào các giao diện thành phần và những câu hỏi về giao diện thực hiện trong quá trình thẩm tra.

7.4. THIẾT KẾ TRƯỜNG HỢP KIỂM THỬ (TEST CASE DESIGN)

Mục tiêu của tiến trình thiết kế các trường hợp kiểm thử là tạo ra một tập các trường hợp kiểm thử hiệu quả trong việc phát hiện ra những lỗi của chương trình và chỉ ra rằng hệ thống đáp ứng được yêu cầu.

Để thiết kế một trường hợp kiểm thử, ta cần lựa chọn một tính năng của hệ thống hoặc một thành phần đang kiểm thử, sau đó chọn một tập dữ liệu đầu vào để thực thi tính năng này.

Dưới đây là một số cách tiếp cận khác nhau được dùng để thiết kế các trường hợp kiểm thử:

1. *Kiểm thử dựa trên yêu cầu:* các trường hợp kiểm thử được thiết kế để kiểm tra các yêu cầu của hệ thống. Kỹ thuật này thường được sử dụng trong các giai đoạn kiểm thử hệ thống. Với mỗi yêu cầu cần xác định các trường hợp kiểm thử để có thể chứng minh rằng các yêu cầu của hệ thống được đáp ứng.
2. *Kiểm thử phân vùng (partition):* xác định các phân vùng dữ liệu đầu vào (input) và phân vùng kết quả đầu ra (output), thiết kế các trường hợp kiểm thử sao cho hệ thống thực thi từ tất cả các phân vùng và sinh ra tất cả các phân vùng của kết quả đầu ra. Các phân vùng là những nhóm dữ liệu có các thuộc tính chung, chẳng hạn như tất cả các số âm, tất cả những tên có độ dài <30 kí tự, tất cả các sự kiện sinh ra khi chọn một mục trên menu…
3. *Kiểm thử cấu trúc:* sử dụng kiến thức về cấu trúc chương trình để thiết kế các trường hợp kiểm thử thực thi tất cả các phần của chương trình. Về cơ bản, khi kiểm thử một chương trình ta nên cố gắng thực thi mỗi câu lệnh ít nhất một lần. Kiểm thử cấu trúc giúp xác định các trường hợp kiểm thử có thể.

Nói chung, khi thiết kế các trường hợp kiểm thử, ta nên bắt đầu bằng kiểm thử mức cao, từ kiểm thử yêu cầu sau đó mới đến kiểm thử phân vùng và kiểm thử cấu trúc.

7.4.1. Kiểm thử dựa trên yêu cầu

Một nguyên tắc trong phát triển phần mềm là các yêu cầu phải được đặc tả sao cho nó có khả năng kiểm thử được. Các yêu cầu nên được viết theo cách để có thể thiết kế các trường hợp kiểm thử, do đó người quan sát có thể kiểm tra xem yêu cầu đó đã được thỏa mãn hay chưa.

Kiểm thử dựa trên yêu cầu là một cách tiếp cận ngữ nghĩa (semantic) để thiết kế các trường hợp kiểm thử thông qua việc xem xét mỗi yêu cầu và sinh ra một tập các trường hợp kiểm thử cho các yêu cầu này. Kiểm thử dựa trên yêu cầu được xem là kiểm thử thẩm định hơn là kiểm thử khiếm khuyết – nghĩa là ta đang cố gắng chứng minh rằng hệ thống hoạt động đúng như đặc tả.

Ví dụ, hãy xem xét những yêu cầu cho hệ thống quản lý thư viện LIBSYS:

1. Người sử dụng có thể tìm kiếm từ tập hợp cơ sở dữ liệu ban đầu hoặc từ một tập con của tập cơ sở dữ liệu ban đầu đó.
2. Hệ thống phải cung cấp công cụ hiển thị hợp lý để người sử dụng xem được tài liệu trong kho tài liệu.
3. Mỗi phiếu đặt sách phải được cấp một số hiệu duy nhất để người sử dụng có thể sao lưu qua tài khoản trong vùng lưu trữ thường trực cá nhân.

Giả sử ta đang xây dựng các kịch bản kiểm thử cho chức năng tìm kiếm, khi đó những trường hợp kiểm thử có thể sinh ra từ những yêu cầu trong chức năng tìm kiếm là:

- Tìm kiếm trong một cơ sở dữ liệu với hai trường hợp: tài liệu có và không có trong cơ sở dữ liệu.
- Tìm kiếm trong hai cơ sở dữ liệu với hai trường hợp: Tài liệu có và không có trong cơ sở dữ liệu.
- Tìm kiếm trong nhiều hơn hai cơ sở dữ liệu với hai trường hợp: Tài liệu có và không có trong cơ sở dữ liệu.
- Chọn một trong những cơ sở dữ liệu từ các thiết lập của người sử dụng và bắt đầu tìm kiếm các tài liệu mà được biết là có mặt và không có mặt trong cơ sở dữ liệu đã chọn.
- Chọn nhiều hơn một cơ sở dữ liệu trong tập cơ sở dữ liệu và bắt đầu tìm kiếm các tài liệu mà được biết là có mặt và được biết không có mặt trong các cơ sở dữ liệu đã chọn.

Thông qua ví dụ này, ta có thể nhận thấy việc kiểm thử một yêu cầu không chỉ viết một trường hợp kiểm thử mà thông thường ta phải viết vài kịch bản kiểm thử để đảm bảo rằng các kịch bản kiểm thử này đã bao phủ được tất cả các trường hợp của yêu cầu.

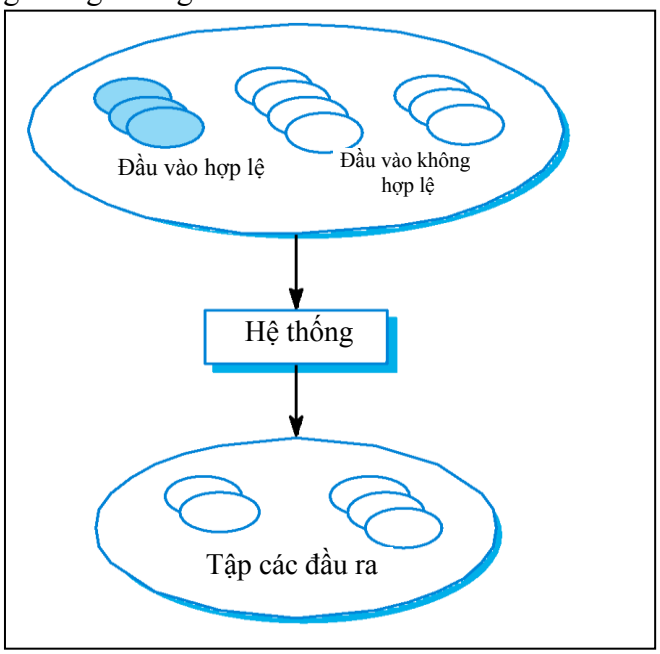
Kiểm thử các yêu cầu khác trong hệ thống LYBSYS có thể được thực hiện giống như trên. Với yêu cầu thứ hai, ta sẽ đưa ra các trường hợp kiểm thử để phân phối tất cả các kiểu tài liệu có thể được xử lý bởi hệ thống và kiểm tra việc hiển thị các tài liệu đó. Với yêu cầu thứ ba, ta có thể đưa vào một vài yêu cầu về việc đặt tài liệu, sau đó kiểm tra định danh yêu cầu được hiển thị trong phiếu chứng nhận của người dùng và kiểm tra định danh yêu cầu đó có phải là duy nhất hay không.

7.4.2. Kiểm thử phân vùng

Dữ liệu đầu vào và kết quả đầu ra của một chương trình thường rơi vào một số lớp khác nhau có chung các thuộc tính, chẳng hạn như tập số âm, tập số dương và việc lựa chọn từ thanh thực đơn. Các chương trình vận hành theo cách để có thể so sánh được tất cả các thành viên của một lớp. Nghĩa là, nếu cần kiểm thử một chương trình có một số phép toán và yêu cầu có 2 số dương, ta mong rằng chương trình cũng được thực thi như thế với tất cả các số dương.

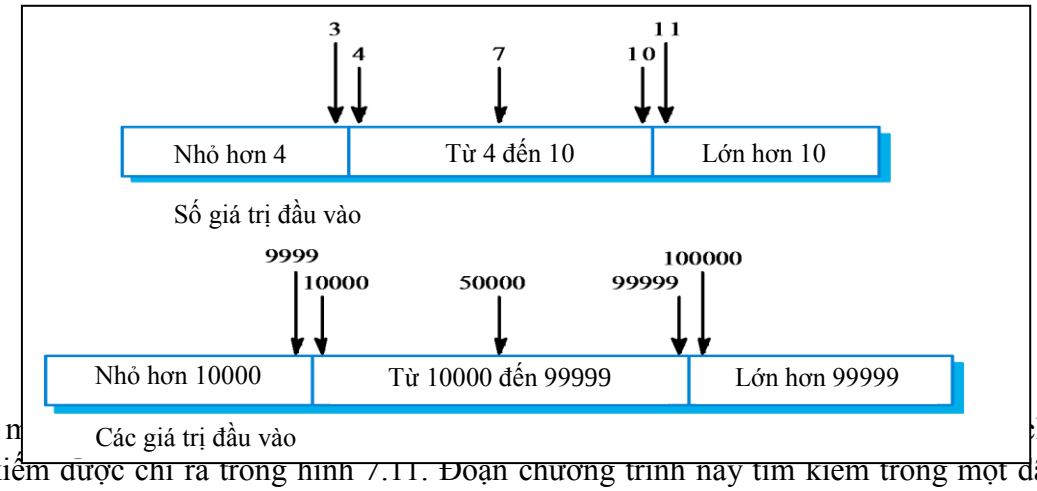
Những lớp này đôi khi được gọi là phân vùng tương đương. Một cách tiếp cận có hệ thống để thiết kế các trường hợp kiểm thử là dựa trên việc xác định tất cả các vùng cho một hệ thống hoặc một thành phần. Các trường hợp kiểm thử được thiết kế sao cho dữ liệu kiểm thử và kết quả đầu ra nằm trong những vùng này. *Kiểm thử phân vùng có thể được sử dụng để thiết kế các trường hợp kiểm thử cho cả hệ thống và các thành phần.*

Trong hình 7.9, mỗi vùng tương đương được chỉ ra trong một hình ellipse. Các vùng dữ liệu kiểm thử tương đương là các tập dữ liệu mà tất cả các thành viên của tập hợp nên được xử lý cùng một cách. Vùng kết quả đầu ra tương đương là các chương trình có những đặc tính chung, do đó chúng có thể được xem như một lớp riêng biệt. Ta cũng có thể xác định các vùng mà dữ liệu kiểm thử nằm bên ngoài các vùng do ta lựa chọn. Trường hợp kiểm thử này được thực hiện ngay khi chương trình xử lý dữ liệu đầu vào không đúng một cách hợp lý. Dữ liệu đúng và không đúng cũng tạo nên các vùng tương đương nhau.



Khi có một tập các vùng xác định, ta có thể lựa chọn các trường hợp kiểm thử từ các vùng này. Một nguyên tắc nhỏ cho việc lựa chọn các trường hợp kiểm thử là nên lựa chọn các trường hợp nằm ở vùng biên hơn là những trường hợp nằm giữa vùng. Lý do căn bản cho cách lựa chọn này là người thiết kế và lập trình chỉ chú ý tới những giá trị điển hình của dữ liệu đầu vào khi phát triển hệ thống. Ta kiểm tra trường hợp này với dữ liệu thuộc điểm giữa của phân vùng. Những giá trị ngoại biên thường là không đúng kiểu, vì thế nó hay nằm ngoài tầm kiểm soát của người phát triển. Lỗi chương trình thường xảy ra khi thực thi với những dữ liệu này.

Khi xác định các phân vùng ta thường sử dụng tài liệu đặc tả hoặc tài liệu người dùng. Theo kinh nghiệm, ta có thể phán đoán những lớp giá trị dữ liệu vào có nhiều khả năng phát hiện ra lỗi. Ví dụ: một đặc tả chương trình xác định rằng chương trình chấp nhận từ 4 đến 8 đầu vào bao gồm 5 chữ số nguyên lớn hơn 10.000. Hình 7.10 chỉ ra các vùng cho tình huống này và những giá trị đầu vào có thể xảy ra.



Để n chương trình tìm kiếm được chỉ ra trong hình 7.11. Đoạn chương trình này tìm kiếm trong một dãy các thành phần đầu ra, một thành phần có giá trị định trước. Chương trình sẽ trả lại vị trí của thành phần trong dãy nếu nó tìm thấy. Trong trường hợp này, ta xác định phân vùng bằng việc xác định tiền điều kiện, nó đúng khi chương trình được gọi và một hậu điều kiện, nó đúng sau khi chương trình được thực thi.

Tiền điều kiện chỉ ra rằng thuật toán tìm kiếm chỉ thực hiện với một mảng không rỗng. Hậu điều kiện chỉ ra rằng biến tìm ra được thiết lập nếu như thành phần khóa nằm trong dãy. Vị trí của thành phần khóa là L trong dãy. Chỉ số giá trị không được xác định nếu thành phần cần tìm kiếm không tồn tại trong dãy. Từ đặc tả này, bạn có thể nhận thấy 2 vùng tương đương nhau:

1. Input mà thành phần cần tìm nằm trong dãy (found = true)
2. Input mà thành phần cần tìm không nằm trong dãy (found = false)

```
procedure Search (Key : ELEM ; T: SEQ of ELEM;
     Found: in out BOOLEAN; L: in out ELEM_INDEX);

Pre-condition
          -- Dãy có ít nhất một phần tử
          T'FIRST <= T'LAST
Post-condition
          -- Phần tử được tìm thấy và được chỉ bằng L
          ( Found and T (L) = Key)
or
          -- Phần tử không thuộc dãy
          ( not Found and
          not (exists i, T'FIRST >= i <= T'LAST, T (i) = Key ))
```

Để kiểm thử chương trình này, ta có thể áp dụng các chỉ dẫn sau:

1. *Kiểm thử phần mềm với các dãy giá trị đơn.* Người lập trình thông thường nghĩ rằng các dãy được hình thành bởi một số giá trị và thỉnh thoảng họ gắn giả thiết này vào chương trình của họ. Kết quả là chương trình có thể làm việc không hợp lý khi được thực hiện với một dãy giá trị đơn.
2. *Sử dụng các dãy khác nhau với kích thước khác nhau trong các kịch bản khác nhau.* Điều này làm giảm cơ hội mà một chương trình có lỗi sẽ sinh ra kết quả đúng một cách ngẫu nhiên bởi các đặc tính ngẫu nhiên của dữ liệu đầu vào.

Vì thế các trường hợp kiểm thử với các thành phần đầu tiên, ở giữa, cuối cùng của dãy được truy cập. Cách tiếp cận này sẽ phát hiện ra những vấn đề tại các điểm biên của phân vùng.

Từ những hướng dẫn trên, có hai phân vùng tương đương cần phải được xác định:

1. Dãy dữ liệu vào có một giá trị đơn.
2. Số thành phần trong dãy dữ liệu vào nhiều hơn một thành phần.

Sau đó có thể xác định thêm các phân vùng bằng cách kết hợp các vùng này – ví dụ, phân vùng mà số thành phần trong dãy lớn hơn số thành phần không nằm trong phân vùng.

Bảng 7.1 chỉ ra các phân vùng phải xác định để kiểm thử chức năng tìm kiếm. Một tập có thể các trường hợp kiểm thử dựa trên những phân vùng này cũng được chỉ ra như trong bảng 7.1. Nếu thành phần cần tìm kiếm không nằm trong dãy, giá trị của L là không xác định ('??'). Gợi ý sử dụng các dãy với kích thước khác nhau để kiểm thử trường hợp này.

Tập hợp các giá trị đầu vào để kiểm thử giải thuật tìm kiếm không thể bao quát được tất cả các khía cạnh. Giải thuật có thể sai nếu dãy dữ liệu vào bao gồm các giá trị 1, 2, 3, 4.

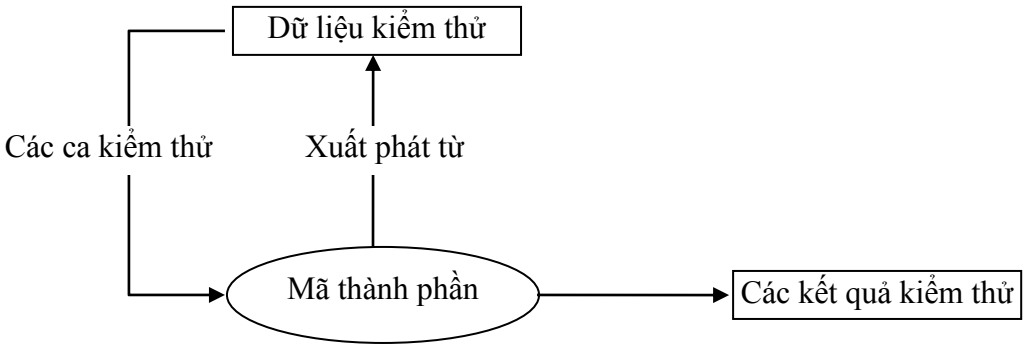
Bảng 7.1. Các phân hoạch tương đương cho chương trình tìm kiếm

Dãy	**Phần tử**
Có một giá trị	Thuộc dãy
Có một giá trị	Không thuộc dãy
Nhiều hơn một giá trị	Là phần tử đầu tiên trong dãy
Nhiều hơn một giá trị	Là phần tử cuối cùng trong dãy
Nhiều hơn một giá trị	Là phần tử nằm giữa trong dãy
Nhiều hơn một giá trị	Không thuộc dãy

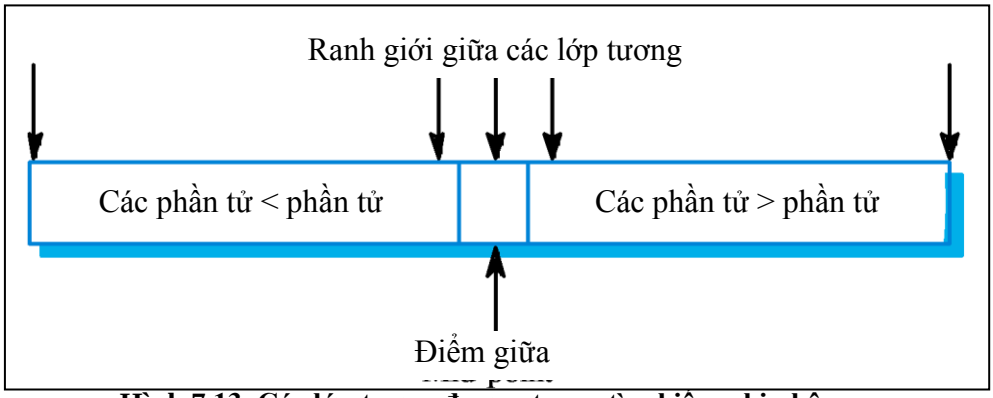
Dãy đầu vào (input)	**Khóa (Key)**	**Đầu ra (Found, L)**
17	17	True, 1
17	0	False, ??
17, 29, 21, 23	17	True, 1
41, 18, 9, 31, 30, 16, 45	45	True, 7
17, 18, 21, 23, 29, 41, 38	23	True, 4
21, 23, 29, 33, 38	25	False, ??

7.4.3. Kiểm thử cấu trúc

Kiểm thử cấu trúc (hình 7.12) là một cách tiếp cận để thiết kế các trường hợp kiểm thử khi biết cấu trúc và mã nguồn của phần mềm. Cách tiếp cận này đôi khi còn gọi là kiểm thử hộp trắng, hộp gương hoặc hộp trong – để phân biệt với kiểm thử hộp đen.



Hình 7.12. Kiểm thử cấu trúc



Hình 7.13. Các lớp tương đương trong tìm kiếm nhị phân

Việc hiểu giải thuật được sử dụng trong chương trình có thể giúp bạn xác định các phân vùng và các trường hợp kiểm thử. Để minh họa cho điều này, ta tiến hành đặc tả thủ tục tìm kiếm (hình 7.11) bằng giải thuật tìm kiếm nhị phân (hình 7.14). Giải thuật này phải có tiền điều kiện chặt chẽ. Dãy được thực hiện phải là một dãy đã được sắp xếp và giá trị ở cận dưới của mảng phải nhỏ hơn giá trị cận trên.

```
Class BinSearch{
// Đây là hàm tìm kiếm nhị phân được thực hiện trên một dãy các đối tượng đã có thứ tự và một khóa
// Trả về một đối tượng với hai thuộc tính là:
// Index – giá trị chỉ số của khóa trong dãy
// found – có kiểu logic cho biết có hay không có khóa trong dãy
// Một đối tượng được trả về bởi trong java không thể thông qua các kiểu cơ bản bằng tham chiếu
// tới một hàm và trả về hai giá trị. Giá trị Index = -1 nếu khóa không có trong dãy.

public static void search(int key, int[] elemArray, Result r)
{
1.      int bottom = 0;
2.      int top = elemArray.length – 1;
                Int mid;
3.      r.found = false;
4.      r.index = -1;
5.      while (bottom <=top)
        {
6.              mid = (top + bottom)/2;
7.              if(elemArray[mid] = key;
                {
8.                      r,index = mid
9.                      r.found = true;
10.                     return;
                } //if
                Else
                {
11.                     if(elemArray[mid]<key)
12.                             bottom = mid + 1;
                        Else
13.                             Top = mid – 1;
                }
        } //while loop
} //search
```

Sau đó ta thiết kế các trường hợp kiểm thử sao cho khóa nằm ở biên của các phân vùng. Đây chính là cơ sở để xây dựng các trường hợp kiểm thử cho giải thuật tìm kiếm, như chỉ ra trong bảng 7.2. Dãy dữ liệu đầu vào được sắp xếp theo thứ tự tăng dần và phải thêm vào các kiểm thử mà thành phần cần tìm kiếm sát với điểm giữa của dãy.

Bảng 7.2. Các trường hợp kiểm thử cho thuật toán tìm kiếm nhị phân

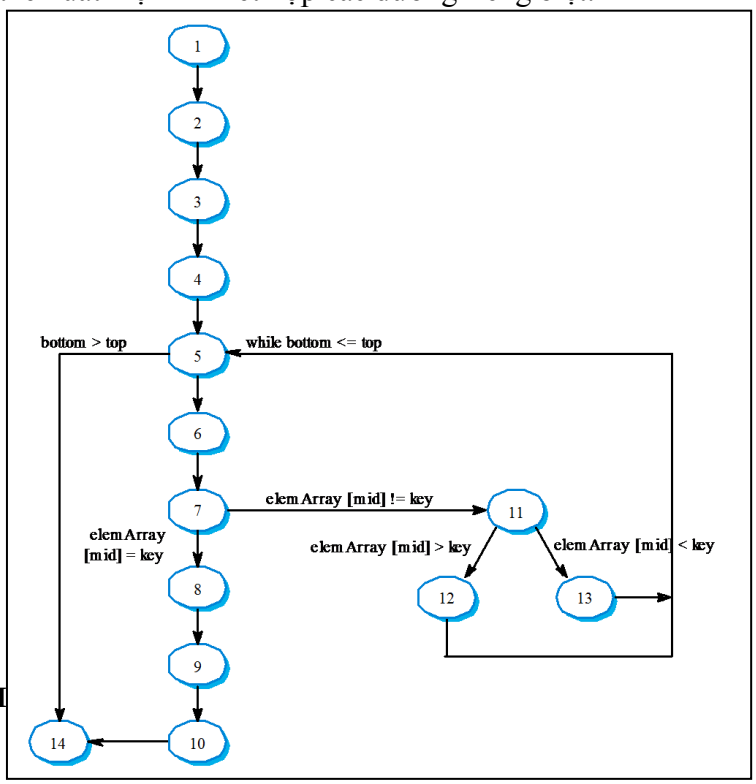
Dãy đầu vào (input)	Khóa (Key)	Đầu ra (Found, L)
17	17	True, 1
17	0	False, ??
17, 21, 23, 29	17	True, 1
9, 16, 18, 30, 31, 41, 45	45	True, 7
17, 18, 21, 23, 29, 38, 41	23	True, 4
17, 18, 21, 23, 29, 33, 38	21	True, 3
12, 18, 21, 23, 32	23	True, 4
21, 23, 29, 33, 38	25	False, ??

7.4.4. Kiểm thử đường (path testing)

Kiểm thử đường là một chiến lược kiểm thử cấu trúc với mục tiêu thực thi tất cả các đường trong một thành phần hoặc một chương trình. Trước tiên, mọi đường độc lập đều được thực thi, sau đó tất cả các câu lệnh trong thành phần phải được thực thi ít nhất một lần. Hơn nữa, tất cả các điều kiện cũng đều phải được kiểm thử với hai trường hợp đúng và sai.

Trong tiến trình phát triển hướng đối tượng, kiểu kiểm thử này cũng có thể được sử dụng để kiểm thử việc thực thi các phương thức của đối tượng. Số lượng đường độc lập trong một chương trình thường cân xứng với kích thước của nó. Khi một module được tích hợp vào trong hệ thống, nó trở nên khó sử dụng phương pháp kiểm thử cấu trúc. Kỹ thuật kiểm thử đường được sử dụng nhiều khi kiểm thử đơn vị.

Kiểm thử đường không kiểm thử tất cả những kết hợp có thể của các thành phần trong chương trình. Đối với bất kỳ thành phần nào ngoài thành phần thông thường không có vòng lặp, thì điều này là không khả thi vì có vô số đường kết hợp có thể trong một chương trình có lặp. Thậm chí khi tất cả các câu lệnh của chương trình đều đã được thực thi ít nhất một lần, lỗi của chương trình vẫn có thể xuất hiện khi kết hợp các đường riêng biệt.



H

Điểm bắt đầu của kiểm thử đường là sơ đồ luồng chương trình, đó là một mô hình khung của tất cả các đường trong chương trình. Một đồ thị luồng bao gồm các điểm biểu diễn các lệnh và đường mũi tên biểu diễn luồng điều khiển. Đồ thị luồng được xây dựng từ việc thay thế các lệnh điều khiển trong chương trình bởi các lược đồ tương đương. Nếu không có lệnh goto trong chương trình, việc xây dựng lược đồ này khá đơn giản. Mỗi nhánh của câu lệnh rẽ nhánh (if/case) là một đường riêng biệt. Một mũi tên lặp để chỉ tới điều kiện trong câu lệnh lặp. Sơ đồ của giải thuật tìm kiếm nhị phân được biểu diễn trong hình 7.15, nó tương đương với giải thuật được biểu diễn trong hình 7.14.

Mục tiêu của kiểm thử đường là đảm bảo rằng mỗi đường chạy qua chương trình được thực thi ít nhất một lần. Các đường độc lập là đường có ít nhất một cạnh mới trong đồ thị. Cả các nhánh điều kiện đúng và điều kiện sai đều phải được thực thi.

Đồ thị cho giải thuật tìm kiếm được biểu diễn trong hình 7.15, mỗi đỉnh của đồ thị biểu diễn một câu lệnh thực thi. Ta có thể tìm thấy các đường trong chương trình tìm kiếm nhị phân theo sơ đồ trên:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14

1, 2, 3, 4, 5, 14

1, 2, 3, 4, 5, 6, 7, 11, 12, 5…

1, 2, 3, 4, 6, 7, 2, 11, 13, 5…

Nếu tất cả các đường được thực thi, chúng ta có thể chắc chắn rằng mọi lệnh trong phương thức đã được thực hiện ít nhất một lần. Và mỗi nhánh được thực thi với cả hai điều kiện: đúng và sai. Thông thường, ta có thể tính toán được số lượng các đường độc lập trong chương trình thông qua luồng đồ thị của chương trình. Tuy nhiên, khi chương trình có cấu trúc rẽ nhánh phức tạp, rất khó xác định được số trường hợp kiểm thử đã được thực hiện. Khi đó, có thể sử dụng các công cụ kiểm thử tự động để phát hiện ra cách thức thực thi thực sự của chương trình.

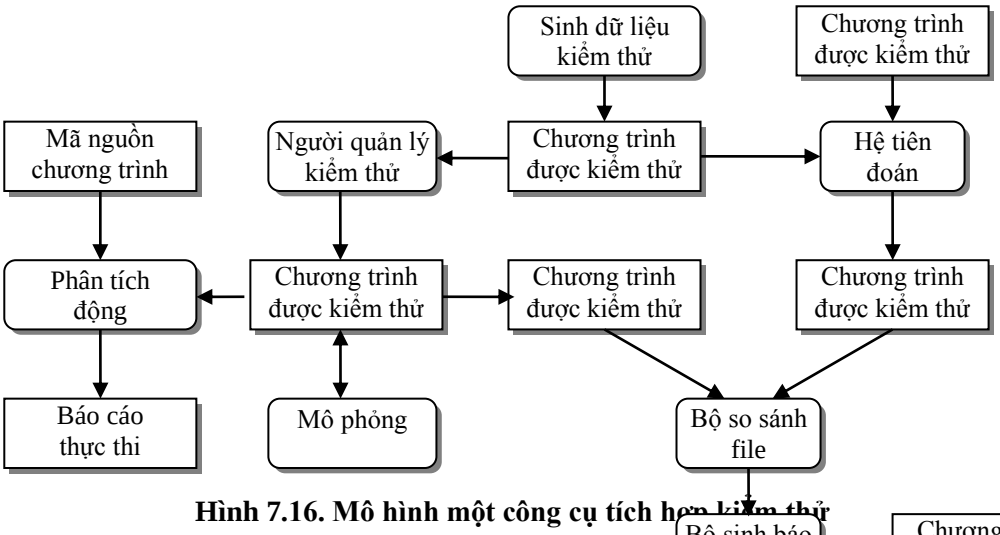
Các công cụ kiểm thử tự động cùng làm việc với trình biên dịch. Trong lúc biên dịch, người phân tích có thể thêm các chỉ thị phụ để sinh ra mã. Chúng đếm số lần mỗi câu lệnh đã được thực hiện. Sau khi chương trình đã thực hiện, một bản phác thảo thực thi có thể được in ra. Nó chỉ ra những phần chương trình đã được thực thi và chưa được thực thi bằng cách sử dụng các trường hợp kiểm thử đặc biệt. Qua đó có thể phát hiện ra những phần chương trình chưa được thực thi.

7.5. CÔNG CỤ KIỂM THỬ TỰ ĐỘNG

Kiểm thử là một giai đoạn tốn kém và được nghiên cứu nhiều nhất trong tiến trình phần mềm. Do đó, công cụ kiểm thử là một trong những công cụ hỗ trợ việc phát triển phần mềm đầu tiên ra đời. Những công cụ này giúp giảm đáng kể thời gian và công sức cho kiểm thử.

Có thể tìm hiểu qua về một cách tiếp cận để kiểm thử tự động, như JUnit, được sử dụng để kiểm thử hồi quy. JUnit là một tập các lớp Java mà người sử dụng có thể mở rộng để tạo ra một môi trường kiểm thử tự động. Mỗi kiểm thử độc lập được thực hiện như một đối tượng và một người kiểm thử chạy tất cả các trường hợp kiểm thử. Các trường hợp kiểm thử nên được viết để chỉ ra rằng các hệ thống được kiểm thử sẽ vận hành như mong muốn.

Một môi trường kiểm thử (workbench testing) là một tập các công cụ được tích hợp hỗ trợ cho tiến trình kiểm thử. Hình 7.16 chỉ ra một số công cụ có thể được tích hợp trong môi trường kiểm thử.



Hình 7.16. Mô hình một công cụ tích hợp kiểm thử

1. Bộ phận quản lý kiểm thử quản lý việc thực thi các [illegible] kiểm thử trong chương trình. Quản lý kiểm thử lưu vết dữ liệu kiểm thử, kết quả mong muốn và các chức năng trong chương trình đã được kiểm thử. JUnit là một ví dụ về người quản lý kiểm thử.
2. Bộ sinh dữ liệu kiểm thử: sinh ra các dữ liệu kiểm thử cho việc kiểm thử chương trình. Điều này có thể được hoàn thành bởi việc lựa chọn dữ liệu từ cơ sở dữ liệu hoặc bằng việc sử dụng các mẫu (pattern) để sinh ra dữ liệu ngẫu nhiên.
3. Hệ tiên đoán (Oracle): sinh ra những phán đoán về kết quả mong đợi. Các hệ tiên đoán cũng có thể là các phiên bản chương trình trước đó hoặc các hệ thống bản mẫu (prototype). Kiểm thử quay lui liên quan tới việc chạy hướng dẫn và chương trình để được kiểm thử song song. Những điểm khác nhau của kết quả đầu ra sẽ được lưu ý.
4. Bộ so sánh file (File comparator) so sánh kết quả của các kịch bản kiểm thử chương trình với những kết quả kiểm thử trước đó và báo cáo về những khác biệt. Việc so sánh được sử dụng trong kiểm thử hồi quy, khi kết quả của các phiên bản khác nhau được so sánh.
5. Bộ sinh báo cáo (Report generator): cung cấp báo cáo và sinh ra các tiện ích cho kết quả kiểm thử.
6. Bộ phận phân tích (Dynamic analyzer): thêm mã nguồn vào chương trình để đếm số lần thực thi mỗi câu lệnh. Sau khi kiểm thử, sinh ra bản tổng kết để chỉ ra câu lệnh nào được thực hiện thường xuyên nhất.
7. Bộ mô phỏng (Simulator) phân biệt các kiểu mô phỏng khác nhau được cung cấp. Mục tiêu của các bộ mô phỏng là mô phỏng cơ chế mà chương trình được thực thi. Mô phỏng giao diện người dùng của các bộ mô phỏng là các chương trình dạng script-driven để mô phỏng đa tương tác người dùng.

Khi sử dụng cho các hệ thống lớn, các công cụ phải được cấu hình cho phù hợp với hệ thống đặc tả. Ví dụ:

1. Các công cụ mới có thể được thêm vào để kiểm thử các đặc trưng của một ứng dụng cụ thể, một số công cụ hiện có có thể không được dùng tới.
2. Các kịch bản có thể được viết cho việc mô phỏng giao diện người dùng và các mẫu (patterns) được xác định cho bộ sinh dữ liệu kiểm thử. Các định dạng cho báo cáo đôi khi cũng phải khai báo.
3. Tập hợp kết quả kiểm thử mong muốn có thể phải so sánh thủ công nếu không có các phiên bản chương trình trước đó có thể dùng như một hệ tiên đoán.
4. Bộ so sánh tệp tin có thể phải viết thêm vào việc mô tả các cấu trúc của kết quả kiểm thử trong tệp tin.

Để tạo ra một bộ công cụ kiểm thử toàn diện đòi hỏi một lượng lớn thời gian và công sức. Do đó, các Workbench hoàn chỉnh (như trong hình 7.16) chỉ được sử dụng khi phát triển các hệ thống lớn. Đó là những hệ thống mà chi phí cho việc kiểm thử có thể chiếm tới 50% tổng chi phí phát triển sản phẩm. Vì thế, nó chỉ hiệu quả để đầu tư cho công cụ chất lượng cao CASE hỗ trợ việc kiểm thử. Tuy nhiên, vì các hệ thống khác nhau đòi hỏi sự hỗ trợ của các loại kiểm thử khác nhau nên đôi khi các công cụ kiểm thử có sẵn không đáp ứng được tất cả các yêu cầu.

CÂU HỎI ÔN TẬP

1. Trình bày các khái niệm cơ bản trong kiểm thử.

2. Trong quá trình kiểm thử hệ thống, người ta có thể sử dụng các hình thức kiểm thử nào?

3. Hãy xây dựng các kịch bản kiểm thử cho chức năng rút tiền từ một máy ATM của hệ thống ngân hàng.

4. Xây dựng các kịch bản kiểm thử phân vùng cho chương trình tìm ước chung lớn nhất của hai số UCLN (a, b).

5. Xây dựng kịch bản kiểm thử đường cho chương trình giải phương trình bậc hai:

$$ax^2 + bx + c = 0$$

Chương 8: BẢO TRÌ PHẦN MỀM VÀ QUẢN LÝ THAY ĐỔI

Bảo trì là giai đoạn cuối cùng trong chu trình phát triển phần mềm. Các chương trình máy tính luôn thay đổi do nhu cầu mở rộng, sửa lỗi, tối ưu hoá, yêu cầu người dùng phát sinh, môi trường vận hành của hệ thống thay đổi... Theo thống kê thì bảo trì chiếm đến 70% toàn bộ công sức bỏ ra cho một dự án phần mềm. Do vậy, bảo trì là một hoạt động phức tạp nhưng nó lại là vô cùng cần thiết trong chu trình sống của sản phẩm phần mềm để đảm bảo rằng phần mềm luôn đáp ứng được yêu cầu của người sử dụng.

Do nhu cầu phát triển của các hệ thống thông tin, rất hiếm hay không muốn nói là không thể có một hệ thống thông tin không có sự thay đổi trong suốt chu trình sống của nó. Để duy trì tính đúng đắn, trật tự trong giai đoạn bảo trì thì quản lý sự thay đổi phần mềm là một hoạt động cần thiết.

Chương này giới thiệu về các hoạt động bảo trì và các loại bảo trì phần mềm. Một số đặc trưng cơ bản của các hoạt động bảo trì, từ đó sinh viên có thể hiểu được các bước cơ bản khi bảo trì, những khó khăn trong quá trình bảo trì các hệ thống phần mềm. Phần cuối của chương giới thiệu về các hình thức bảo trì và hoạt động quản lý thay đổi trong tiến trình bảo trì phần mềm.

8.1. PHÂN LOẠI HOẠT ĐỘNG BẢO TRÌ PHẦN MỀM

Bảo trì phần mềm là phức tạp và chúng ta có thể chia hoạt động bảo trì ra làm bốn hoạt động như sau:

8.1.1. Bảo trì hiệu chỉnh

Hoạt động kiểm thử phần mềm không thể khẳng định được việc loại bỏ hoàn toàn lỗi trong chương trình do có những lỗi ẩn chứa bên trong một hệ thống phần mềm lớn là rất khó phát hiện. Phần mềm sau khi kiểm thử sẽ được bàn giao cho khách hàng, trong quá trình sử dụng, bất kỳ lỗi nào xuất hiện cũng được báo về cho người phát triển.

Bảo trì hiệu chỉnh chính là quá trình phân tích và hiệu chỉnh một hay nhiều lỗi của chương trình đã được đưa vào sử dụng.

8.1.2. Bảo trì cải tiến

Hoạt động bảo trì cải tiến diễn ra do môi trường vận hành hệ thống thường xuyên thay đổi. Những thế hệ phần cứng mới dường như được công bố theo chu trình 24 tháng một lần. Hệ điều hành mới hay phiên bản mới của các hệ điều hành cũ đều đặn xuất hiện; thiết bị ngoại vi và các thành phần hệ thống khác liên tục được nâng cấp và thay đổi. Mặt khác, thời gian hữu dụng của một phần mềm ứng dụng dễ dàng vượt qua thời hạn mười năm, lâu hơn môi trường vận hành hệ thống.

Tóm lại, Bảo trì cải tiến là hoạt động sửa đổi phần mềm để thích ứng được với những thay đổi của môi trường.

8.1.3. Bảo trì hoàn thiện

Hoạt động thứ ba diễn ra khi một phần mềm đã được hoàn tất thành công. Đây là giai đoạn bảo trì quan trọng, thường xuyên và tiêu tốn nhiều công sức nhất trong các loại bảo trì. Vì

trong quá trình sử dụng phần mềm, yêu cầu có thêm các chức năng mới, yêu cầu thay đổi những chức năng đã có và mở rộng tổng thể sản phẩm được người dùng đề xuất. Để thỏa mãn những yêu cầu phát triển của người sử dụng, cần phải tiến hành bảo trì hoàn thiện sản phẩm.

8.1.4. Bảo trì phòng ngừa

Bảo trì phòng ngừa là hoạt động bảo trì diễn ra khi phần mềm được thay đổi để cải thiện tính năng bảo trì hay độ tin cậy trong tương lai hoặc để cung cấp một nền tảng tốt hơn cho những mở rộng sau này. Bảo trì phòng ngừa là hoạt động vẫn còn xa lạ trong thế giới phần mềm hiện nay.

Các thuật ngữ dùng để mô tả ba hoạt động bảo trì đầu tiên là do Swanson đề xướng. Thuật ngữ thứ tư thường được dùng trong việc bảo trì phần cứng hay các hệ thống vật lý khác. Tuy nhiên, cần chú ý rằng những điểm tương tự giữa bảo trì phần mềm và bảo trì phần cứng có thể gây nhầm lẫn. Phần mềm khác với phần cứng là không thể tận dụng được, vì vậy hoạt động bảo trì phần cứng chủ yếu là thay thế các bộ phận bị hỏng hóc hay gãy vỡ - không được kể đến.

Trong thực tế của hoạt động bảo trì, các nhiệm vụ được thực hiện như một phần của bảo trì cải tiến và bảo trì hoàn thiện, cũng giống như các nhiệm vụ cần làm trong các giai đoạn phát triển của một chu trình phần mềm. Để cải tiến hay hoàn thiện, chúng ta đều phải xác định yêu cầu, thiết kế lại, tạo mã và kiểm tra phần mềm có được. Thông thường các nhiệm vụ đó đã được gọi là bảo trì.

8.2. ĐẶC ĐIỂM CỦA BẢO TRÌ PHẦN MỀM

Bảo trì phần mềm cho tới gần đây vẫn còn là một giai đoạn bị coi nhẹ của chu trình phần mềm. Kiến thức về bảo trì có được rất ít khi so sánh với các giai đoạn hoạch định và phát triển. Có rất ít các số liệu nghiên cứu và chế tạo tập trung vào đề tài này và rất ít phương pháp kỹ thuật được đưa ra. Để hiểu được bản chất của bảo trì, chúng ta sẽ xem xét các vấn đề từ ba góc độ khác nhau:

- Các hoạt động cần thiết để hoàn thành giai đoạn bảo trì đảm bảo tính toàn vẹn, ổn định và hiệu quả của phần mềm.

- Chi phí của giai đoạn bảo trì.

- Các vấn đề thường gặp phải khi tiến hành bảo trì phần mềm.

8.2.1. Bảo trì có cấu trúc và bảo trì không cấu trúc

Nếu trong quá trình bảo trì phần mềm, ta chỉ có mã nguồn chương trình thì hoạt động bảo trì sẽ bắt đầu bằng việc đánh giá chi tiết mã nguồn. Công việc này khá phức tạp nếu như thông tin trong mã nguồn nghèo nàn. Những đặc điểm tế nhị như cấu trúc phần mềm, các cấu trúc dữ liệu toàn cục, giao diên hệ thống, hoạt động và các ràng buộc thiết kế thường rất khó sáng tỏ và hay bị hiểu lầm. Các thay đổi lặt vặt thường xuyên làm cho mã nguồn rất khó đánh giá. Các kiểm tra hồi quy (lặp lại các kiểm tra trước kia để đảm bảo rằng những thay đổi không tạo ra lỗi trong phần mềm đã hoạt động) là không thể thực hiện được bởi không hề có các bản lưu về các kiểm tra đó. Đây là hình thức bảo trì không cấu trúc, nó gây lãng phí thời gian, công sức và ít khi có được kết quả hoàn hảo. Khó khăn này bắt nguồn từ việc phát triển phần mềm theo những phương pháp không đúng đắn.

Nếu có một cấu hình phần mềm hoàn thiện, nhiệm vụ bảo trì bắt đầu bằng việc đánh giá các tài liệu thiết kế. Sau đó là xác định các đặc điểm thuộc cấu trúc quan trọng, các đặc điểm hoạt động và giao diện. Tính toàn vẹn của những sửa đổi hay hiệu chỉnh cần thiết sẽ được đánh giá và một kế hoạch sửa đổi sẽ được thiết lập. Thiết kế được thay đổi, nhận xét đánh giá các giải pháp, mã nguồn được phát triển, sau đó tiến hành các kiểm tra hồi quy sử dụng thông tin chứa trong phần "đặc tả kiểm tra", cuối cùng phần mềm lại được phát hành.

Các mô tả trên đây là phép bảo trì cấu trúc và được tiến hành như là kết quả của những ứng dụng trước đây trong khoa học về công nghệ phần mềm. Mặc dù sự có mặt của một cấu hình phần mềm không đảm bảo được các vấn đề bảo trì nảy sinh, nhưng khối lượng công việc đã được giảm bớt và chất lượng chung của những thay đổi hoặc hiệu chỉnh đã được cải thiện.

8.2.2. Giá thành bảo trì

Theo thống kê, giá thành cho việc bảo trì các phần mềm tăng lên một cách đều đặn trong suốt 20 năm qua. Trong những năm 1970, bảo trì chiếm từ 35 đến 40% kinh phí phần mềm dành cho tổ chức hệ thống thông tin. Tỷ lệ này đã nhảy tới con số 60 vào giữa những năm 1980. Nhiều công ty đã chi 80% kinh phí cho việc bảo trì vào giữa những năm 1990.

Chi phí cho việc bảo trì là rõ ràng nhất. Tuy nhiên những chi phí khác khó thấy hơn có thể gây ra mối quan tâm lớn hơn:

- Một chi phí khó xác định của việc bảo trì phần mềm là các cơ hội phát triển bị bỏ qua vì các tài nguyên có sẵn đều dành cho nhiệm vụ bảo trì.

- Sự không hài lòng của người dùng khi các yêu cầu có vẻ hợp lý cho việc sửa chữa hay sửa đổi không được chú ý một cách hợp lý.

- Việc suy giảm chất lượng nói chung do lỗi tạo ra bởi sự thay đổi trong các phần mềm được bảo trì.

- Một yêu cầu bất chợt làm ngắt quãng quá trình phát triển của một nhân viên buộc anh ta tiến hành công việc bảo trì.

Chi phí cuối cùng cho việc bảo trì là sự giảm sút nghiêm trọng về năng suất lao động (được đo theo số dòng lệnh -LOC- của một người trong một tháng hay số tiền chi phí cho dòng lệnh). Sự giảm sút này xuất hiện khi tiến hành bảo trì đối với các phần mềm cũ. Người ta đã ghi nhận sự giảm sút năng suất lao động theo tỷ lệ 40:1, có nghĩa là chi phí cho việc phát triển trị giá $25.00 trên một dòng lệnh sẽ có thể trị giá tới $1000.00 cho việc bảo trì mỗi dòng lệnh.

Công sức cho việc bảo trì có thể được phân chia thành các thao tác làm việc: phân tích, ước lượng, thay đổi thiết kế, sửa chữa chương trình nguồn và các thao tác lặp lại, cố gắng hiểu chương trình nguồn làm gì, cố gắng sáng tỏ cấu trúc dữ liệu, các thuộc tính giao diện, giới hạn của việc thực hiện... Biểu thức dưới đây cung cấp một mô hình cho công việc bảo trì:

$M = p + K^* exp(c-d),$

Với M = toàn bộ các công việc cho việc bảo trì;

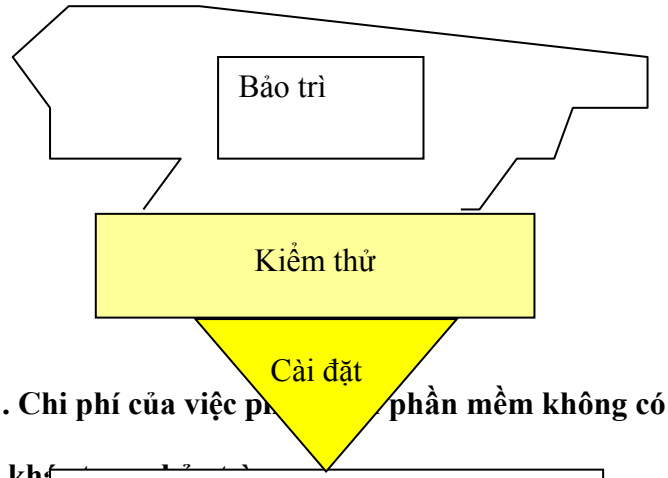
p = công việc làm (như miêu tả ở trên);

K = hằng số kinh nghiệm;

c = đánh giá mức độ phức tạp được tính cho việc thiếu thiết kế về cấu trúc và dữ liệu;

d = đánh giá mức độ hiểu biết về phần mềm.

Mô hình trên đây cho thấy công việc và giá thành có thể tăng lên theo cấp số mũ nếu một phương pháp phát triển phần mềm kém cỏi được sử dụng (tức là thiếu sót của công nghệ phần mềm), hoặc khi một người hay một nhóm dùng các phương pháp không có giá trị để bảo trì phần mềm. Chi phí cho bảo trì khi phần mềm được phát triển không đúng phương pháp được minh hoạ ở hình 8.1:



Hình 8.1. Chi phí của việc p phần mềm không có phương pháp

8.2.3. Một số khó khăn khá

Hầu hết các vấn đề liên quan tới bảo trì phần mềm đều liên quan tới các sai lệch trong cách xây dựng và phát triển phần mềm. Sự thiếu sót trong việc điều khiển và tổ chức trong hai giai đoạn đầu tiên của một chu trình phần mềm gần như luôn luôn tạo ra các vấn đề ở giai đoạn cuối. Nhiều vấn đề kinh điển có thể liên quan tới việc bảo trì phần mềm được trình bày dưới đây:

- Rất khó hoặc không thể theo dõi sự tiến hóa của phần mềm qua các phiên bản.

- Các thay đổi không được tư liệu hóa.

- Khó theo dõi được các quá trình xử lý được tạo bởi các phần mềm.

- Thường xuyên gặp rất nhiều khó khăn trong việc tìm hiểu chương trình của người khác viết.

Những khó khăn này tăng lên khi số thành phần các cấu hình của phần mềm giảm đi. Nếu chỉ có các chương trình nguồn không có tài liệu hướng dẫn thì không nên tìm hiểu phần mềm đó. Những người viết phần mềm thường không có mặt để giải thích. Chúng ta không thể trông cậy vào những giải thích cá nhân của các nhà phát triển phần mềm khi việc bảo trì được yêu cầu.

Các tài liệu chính xác không có hay thiếu trầm trọng. Phải thừa nhận rằng phải có tài liệu về phần mềm là bước đầu tiên, nhưng tài liệu phải hiểu được và phù hợp với chương trình lại là chuyện khác.

Phần lớn các phần mềm không thiết kế để thay đổi, trừ khi sử dụng phương pháp thiết kế dùng các khái niệm về phân tách chương trình thành các module độc lập. Việc thay đổi phần mềm sẽ rất khó khăn và dẫn đến xu hướng sai.

Việc bảo trì phần mềm không được coi là một công việc dễ dàng mà công việc bảo trì phần mềm luôn liên quan tới các sai lệch ở mức độ cao.

8.3. CÔNG VIỆC BẢO TRÌ PHẦN MỀM VÀ MỘT SỐ HIỆU ỨNG LỀ

8.3.1. Khả năng bảo trì

Khả năng bảo trì của phần mềm có thể coi như các khả năng hiểu, hiệu chỉnh, tiếp hợp hoặc có thể thêm vào khả năng phát triển. Khả năng bảo trì là chìa khóa để dẫn đến các phương pháp thiết kế xây dựng phần mềm.

a) Yếu tố kiểm soát

Khả năng bảo trì cơ bản bao gồm nhiều yếu tố. Sự thiếu cẩn trọng trong việc thiết kế, viết chương trình nguồn, kiểm tra có ảnh hưởng tiêu cực đến việc bảo trì có kết quả một phần mềm. Cấu hình yếu kém cũng có các tác động tương tự, thậm chí cả khi từng bước của kỹ thuật xây dựng phần mềm được xem xét một cách cẩn thận. Thêm vào đó nhiều yếu tố khác liên quan tới phương pháp phát triển phần mềm, như:

- Chất lượng hiệu quả của đội ngũ phát triển phần mềm.
- Cấu trúc của hệ thống dễ hiểu.
- Dễ dàng kiểm soát hệ thống.
- Dùng các ngôn ngữ lập trình chuẩn.
- Dùng các hệ điều hành chuẩn.
- Dùng các cấu trúc chuẩn tài liệu.
- Dùng được các tài liệu kiểm tra.
- Phương tiện gỡ rối đi kèm.
- Dùng được các máy tính tốt để thực hiện việc bảo trì.

Các yếu tố được đưa ra ở trên đã phản ánh những đặc điểm về phần cứng cũng như chương trình nguồn được dùng trong việc phát triển phần mềm. Những yếu tố khác chỉ ra sự cần thiết để có được phương pháp chuẩn, chương trình nguồn chuẩn. Có thể, yếu tố quan trọng nhất tác động tới khả năng bảo trì là kế hoạch cho khả năng bảo trì. Nếu coi phần mềm như là một hệ thống các thành phần sẽ phải chịu những thay đổi không tránh được, thì cơ hội tạo những phần mềm có khả năng bảo trì sẽ tăng thực sự.

b) Đánh giá định lượng

Khả năng bảo trì, như chất lượng hay độ tin cậy là hết sức khó xác định. Tuy nhiên chúng ta có thể đánh giá khả năng bảo trì gián tiếp bằng việc xem xét các thuộc tính của các công việc bảo trì có thể đánh giá được là:

- Thời gian nhận biết vấn đề.
- Thời gian trễ do các công việc hành chính.
- Thời gian lựa chọn công cụ bảo trì.
- Thời gian phân tích vấn đề.

- Thời gian xác định sự thay đổi.

- Thời gian hiệu chỉnh (hay sửa đổi) thực sự.

- Thời gian chạy thử cục bộ.

- Thời gian chạy thử tổng thể.

- Thời gian tổng kết bảo trì.

- Tổng thời gian của công việc bảo trì.

Mỗi đánh giá trên thực tế là các dữ liệu đó có thể cung cấp cho người quản lý cùng với chỉ số về hiệu quả của công việc.

8.3.2. Các công việc bảo trì

Những nhiệm vụ liên quan tới việc bảo trì gồm: các tổ chức bảo trì được thiết lập; các thủ tục ghi nhận và đánh giá được miêu tả và một loạt thứ tự chuẩn của các bước cho mỗi yêu cầu bảo trì phải được định nghĩa. Thêm vào đó, một thủ tục lưu trữ các hồ sơ cho các hoạt động bảo trì được thiết lập và bản tổng kết những tiêu chuẩn đánh giá được vạch rõ.

a) Cơ cấu bảo trì

Mặc dù những tổ chức bảo trì chuẩn không cần được thiết lập, nhưng sự ủy thác trách nhiệm rất cần thiết kể ngay cả với các tổ chức phát triển phần mềm nhỏ. Những yêu cầu bảo trì được chuyển qua cho người kiểm soát công việc bảo trì và từ đây chuyển tiếp yêu cầu tới người quản lý hệ thống để đánh giá, người quản lý hệ thống là thành viên của nhóm nhân viên kỹ thuật. Những nhân viên này có trách nhiệm về một phần nhỏ của chương trình sản phẩm. Khi một yêu cầu được đánh giá, người được ủy quyền quản lý việc thay đổi phải quyết định hành động nào được thực hiện tiếp theo.

Cơ cấu được miêu tả ở trên phục vụ cho việc thiết lập phạm vi trách nhiệm đối với công việc bảo trì. Người kiểm soát và người ủy quyền quản lý việc thay đổi có thể là một người hay là một nhóm quản lý và chuyên gia kỹ thuật cao cấp.

b) Báo cáo

Tất cả các yêu cầu về việc bảo trì phần mềm cần được trình bày theo một chuẩn. Người phát triển phần mềm thường cung cấp một đơn yêu cầu bảo trì còn được gọi là báo cáo các lỗi phần mềm. Báo cáo này được người sử dụng điền vào khi yêu cầu công việc bảo trì. Nếu xuất hiện một lỗi, bản mô tả đầy đủ tình huống dẫn đến lỗi bao gồm dữ liệu, đoạn chương trình và các yêu cầu khác phải được điền đầy đủ vào bản báo cáo. Nếu yêu cầu bảo trì là bảo trì cải tiến hay bảo trì hoàn thiện thì một yêu cầu chi tiết sẽ được thảo ra. Đơn yêu cầu bảo trì sẽ được người kiểm soát bảo trì và người quản lý hệ thống xem xét như phần trước đã nêu.

Đơn yêu cầu bảo trì được thiết lập từ bên ngoài và được coi như cơ sở để đề ra kế hoạch của công việc bảo trì. Ngoài ra, trong nội bộ của cơ quan phần mềm một báo cáo thay đổi phần mềm cũng được tạo ra. Báo cáo này chỉ ra công sức đòi hỏi để thỏa mãn một đơn yêu cầu bảo trì, trạng thái ban đầu của yêu cầu sửa đổi, mức ưu tiên của yêu cầu, các dữ liệu cần cho việc sửa đổi...

c) Lưu giữ các hồ sơ

Thường chúng ta không có đầy đủ các hồ sơ cho tất cả các giai đoạn trong chu kỳ sống của một phần mềm. Thêm nữa không có các hồ sơ về việc bảo trì phần mềm. Do đó, chúng ta thường khó có thể tiến hành các công việc bảo trì có hiệu quả, không có khả năng xác định tính chất của các chương trình và khó xác định được giá bảo trì phần mềm. Các thông tin cần phải lưu giữ trong hồ sơ bảo trì thường là:

- Dấu hiệu nhận biết chương trình.
- Số lượng các câu lệnh trong chương trình nguồn.
- Số lượng các lệnh mã máy.
- Ngôn ngữ lập trình được sử dụng.
- Ngày cài đặt chương trình.
- Số các chương trình chạy từ khi cài đặt.
- Số các lỗi xử lý xảy ra.
- Mức và dấu hiệu thay đổi chương trình.
- Số các câu lệnh được thêm vào chương trình nguồn khi chương trình thay đổi.
- Số các câu lệnh được xóa khỏi chương trình nguồn khi chương trình thay đổi.
- Số giờ mỗi người sử dụng cho mỗi lần sửa đổi.
- Ngày thay đổi chương trình.
- Dấu hiệu của kỹ sư phần mềm.
- Dấu hiệu của đơn yêu cầu bảo trì.
- Kiểu bảo trì.
- Ngày bắt đầu và kết thúc bảo trì.
- Tổng số giờ của mỗi người dùng cho việc bảo trì.

d) Xác định giá bảo trì

Việc xác định giá trị bảo trì thường phức tạp do thiếu thông tin. Nếu tiến hành việc lưu giữ các hồ sơ có thể tiến hành một số cách đánh giá về việc thực hiện bảo trì. Theo các chuyên gia thì đánh giá về việc thực hiện bảo trì dựa vào:

- Số lượng trung bình các lỗi xử lý cho một lần chạy chương trình.
- Tổng số giờ của mỗi người dùng cho mỗi loại bảo trì.
- Số lượng trung bình các thay đổi theo chương trình, theo ngôn ngữ lập trình, theo kiểu bảo trì.
- Số giờ trung bình của mỗi người cho một dòng lệnh được thêm vào và xóa đi.
- Số giờ trung bình của mỗi người cho một ngôn ngữ lập trình.
- Thời gian trung bình cho việc bảo trì một đơn yêu cầu bảo trì.
- Tỷ lệ phần trăm của mỗi kiểu bảo trì.

8.3.3. Một số hiệu ứng lề của công việc bảo trì

Sửa đổi phần mềm là công việc nguy hiểm, ta thường gặp ba loại chính của hiệu ứng lề như sau:

a) Hiệu ứng lề của việc thay đổi mã nguồn

Một thay đổi đơn giản tới một câu lệnh đơn cũng có thể đem lại một hậu quả thảm khốc. Mặc dù không phải các ảnh hưởng đều là tiêu cực, nhưng việc sửa lỗi luôn dẫn đến các vấn đề phức tạp. Mặc dù tất cả các thay đổi mã lệnh chương trình đều có thể tạo ra lỗi, nhưng tập hợp các thay đổi sau có thể gây ra nhiều lỗi hơn.

- Một chương trình con bị xóa hay thay đổi.
- Một dòng nhãn bị xóa hay thay đổi.
- Một biến bị xóa hay thay đổi.
- Các thay đổi để tăng khả năng thực hiện.
- Việc mở và đóng file bị thay đổi.
- Các phép toán logic bị thay đổi.
- Việc thay đổi thiết kế chuyển thành các thay đổi lớn về chương trình.
- Các thay đổi ảnh hưởng đến việc chạy thử các trường hợp biên.

b) Hiệu ứng lề của việc thay đổi dữ liệu

Trong quá trình bảo trì, việc sửa đổi thường được tiến hành đối với các phần tử riêng rẽ của cấu trúc dữ liệu. Khi dữ liệu thay đổi, việc thiết kế phần mềm sẽ không còn phù hợp với dữ liệu và lỗi có khả năng xảy ra. Hiệu ứng lề của dữ liệu xảy ra như là kết quả của việc thay đổi cấu trúc dữ liệu. Các thay đổi dữ liệu sau đây thường gây ra lỗi:

- Định nghĩa lại các hằng số cục bộ và hằng số địa phương.
- Định nghĩa lại cấu trúc bản ghi hay cấu trúc file.
- Tăng hoặc giảm kích thước một mảng.
- Thay đổi dữ liệu tổng thể.
- Định nghĩa lại các cờ điều khiển và các con trỏ.
- Xếp lại các tham số vào ra hay tham số của chương trình con.

Hiệu ứng lề dữ liệu có thể được hạn chế bằng tài liệu thiết kế mô tả cấu trúc dữ liệu và cung cấp một lời chỉ dẫn tham khảo đến từng phần từ dữ liệu, các bản ghi, các file và các cấu trúc khác của các module phần mềm.

c) Hiệu ứng lề của việc thay đổi tài liệu

Việc bảo trì thường tập trung vào cấu hình phần mềm và không tập trung riêng vào việc sửa đổi mã. Sự ảnh hưởng của tài liệu xảy ra khi thay đổi chương trình nguồn mà không thay đổi tài liệu thiết kế và tài liệu hướng dẫn sử dụng. Bất cứ lúc nào có thay đổi về luồng dữ liệu, cấu trúc phần mềm, các thủ tục hay bất cứ cái gì có liên quan, tài liệu kỹ thuật phải được cập nhật.

Tài liệu về thiết kế phản ánh không đúng trạng thái hiện tại của phần mềm có lẽ còn tồi tệ hơn không có tài liệu. Hiệu ứng lề xảy ra trong các lần bảo trì sau đó, khi việc nghiên cứu không kỹ càng các tài liệu kỹ thuật, dẫn tới sự đánh giá sai về các đặc tính của phần mềm. Đối với người sử dụng, phần mềm tốt chỉ khi có tài liệu hướng dẫn sử dụng chúng.

Các hiệu ứng lề trong tài liệu có thể được giảm về căn bản nếu toàn bộ cấu hình được xem xét trước khi phát hành phiên bản phần mềm tiếp sau. Thực tế một vài yêu cầu bảo hành có thể đòi hỏi không được thay đổi thiết kế của phần mềm hoặc mã chương trình, mà chỉ cần chỉ ra sự thiếu rõ ràng trong tài liệu của người sử dụng. Trong những trường hợp như vậy nỗ lực bảo trì tập trung vào tài liệu.

8.4. MỘT SỐ HÌNH THỨC BẢO TRÌ PHẦN MỀM

8.4.1. Bảo trì mã chương trình xa lạ

Các chương trình được gọi là mã chương trình xa lạ nếu:

- Không một thành viên nào trong phòng kỹ thuật tiếp tục phát triển chương trình đó nữa.

- Không tiếp tục áp dụng lý thuyết phát triển, vì vậy tồn tại vấn đề thiết kế nghèo nàn và ít tài liệu (theo tiêu chuẩn ngày nay).

- Cấu trúc khối chưa được thiết kế theo tiêu chuẩn, các khái niệm về thiết kế có cấu trúc chưa được áp dụng.

Dưới đây là một số đề nghị hữu dụng cho người quản trị hệ thống phải bảo trì các chương trình xa lạ:

- Nghiên cứu chương trình trước khi bạn bị đặt vào "chế độ khẩn cấp". Cố gắng thu nhận được càng nhiều thông tin cơ sở càng tốt.

- Cố gắng làm quen với toàn bộ các luồng điều khiển của chương trình; trước hết bỏ qua các chi tiết về mã chương trình. Sẽ rất có ích khi vẽ riêng sơ đồ cấu trúc và sơ đồ luồng hoạt động mức cao, nếu chưa có bản nào tồn tại.

- Đánh giá tình hình hợp lý của tài liệu hiện có; bổ sung thêm các lời chú thích của bản thân bạn vào chương trình nguồn nếu bạn thấy cần thiết.

- Sử dụng tốt các danh sách chỉ dẫn tham khảo, các bảng ký hiệu và các trợ giúp khác thường được chương trình dịch cung cấp.

- Thực hiện sửa đổi chương trình với sự chú ý lớn nhất. Lưu ý tới kiểu và dạng của chương trình tại tất cả các chỗ có thể. Đánh dấu trên chương trình những lệnh bạn đã sửa.

- Đừng loại bỏ chương trình trừ khi bạn chắc chắn nó không được sử dụng nữa.

- Đừng cố sử dụng chung các biến tạm thời và vùng nhớ làm việc mà đã có sẵn trong chương trình. Thêm các biến của riêng bạn để tránh các rắc rối.

- Giữ các bản ghi chép chi tiết (về hoạt động bảo trì và các kết quả).

- Tránh sự nóng vội vô ý ném chương trình cũ đi và viết lại nó.

- Thực hiện các kiểm tra lỗi.

8.4.2. Công nghệ phản hồi và công nghệ tái sử dụng

Công nghệ phản hồi -Reverse engineering- đối với phần mềm là đơn giản. Trong nhiều trường hợp, chương trình được tổ chức ngược không phải thuộc nhà cạnh tranh mà thuộc bản thân công ty. Bí mật cần khám phá là do không còn giữ được các đặc tả. Do đó, tổ chức ngược đối với phần mềm là quá trình phân tích chương trình trong cố gắng biểu diễn lại các chương trình ở mức độ trừu tượng cao hơn mã nguồn. Tổ chức lại là quá trình khám phá thiết kế. Các công cụ của công nghệ phản hồi lấy ra dữ liệu, kiến trúc, các thông tin thiết kế thủ tục từ chương trình đã tồn tại.

Công nghệ tái sử dụng, Re-engineering, không đơn thuần phát hiện các thông tin thiết kế mà còn dùng các thông tin này để biến đổi hoặc tổ chức lại hệ thống đã tồn tại với mục đích cải thiện chất lượng. Trong nhiều trường hợp, phần mềm ứng dụng lại các chức năng của hệ thống tồn tại. Nhưng trong cùng thời điểm, nhà phát triển phần mềm cũng phải thêm các chức năng mới và/hoặc cải thiện các xử lý.

8.4.3. Bảo trì dự phòng

Bảo trì dự phòng các phần mềm máy tính là một vấn đề khá mới và còn đang được tranh cãi. Thay vì đợi cho đến khi nhận được yêu cầu bảo trì, các tổ chức phát triển hay bảo trì chọn một chương trình mà:

- Sẽ được sử dụng trong một số năm định trước;

- Hiện đang được sử dụng tốt;

- Dễ bị thay đổi hoặc nâng cấp trong tương lai gần.

Thoạt đầu ý kiến đề nghị phát triển lại một chương trình lớn khi một phiên bản đang làm việc đã tồn tại ta thấy dường như quá lãng phí. Nhưng chúng ta hãy xem xét các điểm sau:

- Chi phí để bảo trì một dòng mã lệnh có thể lớn hơn 20 tới 40 lần chi phí cho phát triển ban đầu dòng lệnh đó.

- Thiết kế lại cấu trúc của phần mềm, với sự sử dụng các khái niệm thiết kế hiện tại có thể làm cho việc bảo hành tương lai dễ dàng hơn.

- Bởi vì khuôn mẫu phần mềm đã tồn tại, năng suất phát triển chương trình chắc sẽ cao hơn mức trung bình nhiều.

- Người sử dụng bây giờ đã làm quen với phần mềm. Vì vậy, các đòi hỏi mới và hướng thay đổi có thể tìm ra dễ dàng hơn nhiều.

- Các công cụ CASE dành cho reverse engineering và re-engineering sẽ thực hiện tự động một số phần của công việc.

- Một cấu hình phần mềm sẽ tồn tại trên sự hoàn thành của bảo trì phòng ngừa.

Khi một tổ chức phát triển phần mềm bán phần mềm như là một sản phẩm, bảo trì phòng ngừa được xem như "phiên bản mới" của chương trình. Nhiều hãng phát triển phần mềm lớn có thể có từ 500 tới 2000 sản phẩm chương trình trong phạm vi trách nhiệm của nó. Các chương trình như vậy có thể được xếp theo thứ tự ưu tiên và xem xét lại như các ứng cử cho bảo trì phòng ngừa.

8.4.4. Chiến lược phần mềm thành phần

Như đặc tính cổ điển của bảo hành phần cứng là tháo bỏ phần hỏng và thay thế bằng phụ tùng mới. Một khái niệm được gọi là nguyên mẫu phần mềm có thể dẫn tới việc phát triển các phụ tùng cho các chương trình.

Nguyên mẫu phần mềm là một quá trình mô hình hóa yêu cầu người dùng trong một hay nhiều mức chi tiết, bao gồm cả các mô hình làm việc. Các tài nguyên của dự án được xếp đặt làm sao để sản xuất các phiên bản phần mềm được mô tả theo yêu cầu phải nhỏ đi. Phiên bản nguyên mẫu làm cho người dùng, người thiết kế và quản trị... có thể xem lại được phần mềm. Quá trình đó sẽ tiếp tục khi được đề nghị, với phiên bản đang chạy chuẩn bị sẵn sàng phát hành sau vài lần làm lại.

Nếu các mức nguyên mẫu khác nhau được phát triển, nó có thể có một bộ các phụ tùng phần mềm có thể được sử dụng khi nhận được yêu cầu bảo trì hiệu chỉnh. Ví dụ một module phân tích có thể được thiết kế và thực hiện theo hai cách khác nhau nhưng có cùng giao diện bên ngoài. Một phiên bản của module có được sử dụng trong phần mềm làm việc. Nếu module đó hỏng, một phụ tùng có thể được thay ngay.

Mặc dù chiến lược phụ tùng thay thế cho phần mềm có vẻ khác thường một chút, nhưng không có bằng chứng gì nó tỏ ra tốn kém, khi chúng ta tính đến chi phí cho tất cả chu kỳ sống của phần mềm.

8.5. QUẢN LÝ THAY ĐỔI PHẦN MỀM

Các ứng dụng thường xuyên phải thiết kế lại do sự phân công của một nhóm quản lý mới, dự án vượt quá ngân sách, ứng dụng chậm và có nhiều lỗi, sự thiếu tin tưởng của chủ sử dụng về việc các kỹ sư phần mềm hiểu rõ các yêu cầu của mình...

Các thay đổi có thể là các yêu cầu, thiết kế, chương trình, giao diện, phần cứng hoặc phần mềm phải mua. Phần lớn các thay đổi bắt nguồn từ bên trong tổ chức phát triển ứng dụng, nhưng cũng có thể được kích hoạt từ các tác nhân bên ngoài, ví dụ như thay đổi về luật.

Việc quản lý thay đổi ứng dụng giúp cho nhóm triển khai bỏ qua những ý thích chợt nảy ra của người sử dụng trong khi vẫn cho phép thực hiện các yêu cầu hợp lý.

8.5.1. Các thủ tục quản lý thay đổi

Quản lý điều khiển thay đổi có hiệu lực từ khi sản phẩm đầu tiên được chấp nhận hoàn thiện cho đến khi dự án kết thúc. Trước tiên, các sản phẩm công việc cơ sở được tạo lập để đưa vào quản lý. Một sản phẩm công việc cơ sở là một sản phẩm được coi là hoàn thiện và là cơ sở cho các công việc hiện tại khác của nhóm triển khai dự án. Ví dụ như, một tài liệu cơ sở là bản quy định yêu cầu chức năng sau khi nó đã được chấp nhận bởi người sử dụng. Dưới đây là ví dụ một quá trình của các thao tác yêu cầu thay đổi của một đặc tả chức năng:

- Tạo yêu cầu mở;

- Khai báo file tác động;

- Phê chuẩn file về thời gian và chi phí do người quản lý, người sử dụng ký;

- Hoàn thiện danh sách và kiểm soát về thay đổi của người điều hành dự án. File tài liệu liên quan đến thay đổi. Nếu tài liệu hoặc chương trình bị thay đổi, thì xác định ngày và các

mục cập nhật đã hoàn thiện. Nếu các thủ tục hoặc thử nghiệm bị thay đổi, xác định các ngày mà việc sửa đổi xảy ra. Mẫu yêu cầu đóng file được chủ/người sử dụng thông qua.

- Tóm tắt các ngày tháng, quá trình và chi phí.

Trước tiên, tài liệu cơ sở được giữ nguyên, sau đó thêm vào các yêu cầu thay đổi. Khi quy định chức năng được cập nhật để điều tiết thay đổi, nó được đóng băng lại và công việc lại tiếp tục. Ba yêu cầu trước có thể được thêm vào ứng dụng nếu chúng không làm thay đổi ứng dụng nhiều. Chúng cũng có thể bị bỏ qua cho đến sau khi ứng dụng đã được thực hiện. Các thay đổi có thể phân loại theo một số cách:

- Thứ nhất, chúng được phân theo kiểu như loại bỏ lỗi, cải tiến thực hiện hoặc thay đổi chức năng;

- Thứ hai, thay đổi phân loại thành yêu cầu và lựa chọn;

- Thứ ba, phân theo độ ưu tiên như khẩn cấp, lệnh với một ngày kết thúc yêu cầu, lệnh với ngày bắt đầu yêu cầu hoặc ưu tiên thấp.

Thông thường, kiểu loại bỏ lỗi là khẩn cấp theo yêu cầu, trong khi thay đổi chức năng là bảo dưỡng lệnh theo yêu cầu và cải tiến thực hiện là lựa chọn và có thể không có ưu tiên. Việc biết được loại yêu cầu thay đổi quyết định xem liệu nó có cần phải chịu điều khiển thay đổi hay không. Các thay đổi khẩn cấp thường phá vỡ thủ tục điều khiển thay đổi do các công việc thực hiện tuần tự nhưng chúng lại được tài liệu hoá sau khi thay đổi đã kết thúc. Tất cả các loại thay đổi khác đều phải tuân theo các điều khiển thay đổi. Ví dụ như thay đổi về yêu cầu chức năng có thể xảy ra bất cứ lúc nào, nhưng khi quy định yêu cầu chức năng được thông qua thì nó đóng băng cho đến khi ứng dụng hoạt động. Các thay đổi phải chịu sự điều khiển thay đổi: chúng được thêm vào danh sách yêu cầu thay đổi để xem xét trong tương lai trừ khi đó là một thiết kế khẩn cấp.

Một thủ tục điều khiển thay đổi yêu cầu người sử dụng phải đệ trình một lời yêu cầu thay đổi chính thức cho người điều hành dự án:

- Người sử dụng gửi cho người điều hành dự án và người chủ một mẫu yêu cầu thay đổi.

- Người điều hành dự án và kỹ sư phần mềm triển khai một khai báo tự động. Vào lúc đó, danh sách kiểm soát của người điều hành dự án được dùng để xác định tất cả các hoạt động và thay đổi công việc có liên quan tới yêu cầu. Yêu cầu thay đổi được thảo luận với chủ sử dụng để vạch ra các thay đổi về ưu tiên, tiến trình và chi phí.

- Thoả thuận được chính thức hoá và chủ sử dụng thông qua thay đổi về tiến trình và chi phí.

- Sử dụng khai báo tác động, ứng dụng và tất cả các tài liệu có liên quan được thay đổi. Thực hiện thay đổi: khi các nhiệm vụ hoàn thành, xoá nhiệm vụ trong danh sách kiểm soát của người điều hành dự án.

- Chủ sử dụng thông qua việc đóng yêu cầu và yêu cầu được đóng. Người điều hành dự án và kỹ sư phần mềm định nghĩa các tác động tiến trình và chi phí của thay đổi. Sau đó, các thay đổi được bàn bạc với người sử dụng. Dựa trên thương lượng với người sử dụng, thay đổi được gán một ưu tiên hoạt động, chi phí và tiến trình được thay đổi.

- Yêu cầu, ngày dự định hoạt động, thay đổi tiến trình và tăng chi phí được thêm vào một file quá trình dự án. Các thay đổi có thể được quản lý bởi một nhân viên điều khiển thay đổi, là một người có nhiệm vụ bảo dưỡng quá trình dự án và các bản ghi điều khiển thay đổi, hàng tháng in ra một bản báo cáo điều khiển thay đổi. Một tệp điều khiển thay đổi chứa tất cả các yêu cầu, thư từ và tài liệu về các thay đổi. Một yêu cầu thay đổi mở có thể được tạo ra khi yêu cầu được đưa ra và một số lượng thay đổi được gán. Yêu cầu thay đổi mở nằm trong tệp cho đến khi yêu cầu được hoàn thành, đóng và được báo cáo.

Khi thay đổi được thực hiện, các mục có ảnh hưởng được cập nhật, bao gồm tư liệu tương ứng, mã nguồn chương trình,... Một danh sách kiểm soát của người điều hành dự án được dùng để loại bỏ các hoạt động đã được yêu cầu. Tài liệu mới được nhân viên điều khiển thay đổi sắp xếp và phân phối nó cho những người có quan tâm. Ngày hoàn thành thay đổi được đưa vào file điều khiển thay đổi. Thay đổi được xác định khi được đóng trong báo cáo tình trạng tới và yêu cầu mở được chuyển từ file điều khiển thay đổi sang.

Dựa trên tổ chức này, người điều hành hệ thống có thể theo dõi các yêu cầu thay đổi của dự án để nhận biết sự thành công trong nhóm các yêu cầu. Chi phí thay đổi chung của một năm thường được sử dụng như là một chỉ tiêu để chỉ ra xem ứng dụng đang có triển vọng hay cần vứt bỏ hay cần công nghệ hoá lại. Trong những trường hợp này, cả chi phí và số lượng các yêu cầu thay đổi đều được theo dõi thông qua quá trình điều khiển thay đổi. Các báo cáo tổng kết bởi dự án thay đổi trong một thời kỳ nhất định, hoặc so sánh theo thời kỳ có thể được triển khai.

8.5.2. Ghi quyết định theo thời gian

Khi bắt đầu một dự án, người điều hành dự án và kỹ sư phần mềm quyết định sử dụng các công cụ để lưu trữ quá trình quyết định. Có nghĩa là có thể dùng công cụ điện tử hoặc một phiên bản viết tay và các quyết định được duy trì dưới dạng văn bản. Với công cụ điện tử, các bản sao điện tử được lưu trữ. Với công cụ ghi tay, phiên bản cũ được cập nhật và đổi tên khi một tài liệu thay đổi. Ví dụ như, các quy định chức năng của công ty ABC có thể được đặt tên là ABCFS-mmddyy, trong đó ABC là công ty, FS là viết tắt của quy định chức năng (Functional Specification) và mmddyy là ngày tháng năm. Phần ngày tháng của tên sẽ thay đổi do bất kỳ một thay đổi quan trọng nào của tài liệu. Thủ tục quản lý thay đổi sẽ được nói đến trong phần tiếp theo.

8.5.3. Quản lý thay đổi tài liệu

Các thay đổi tài liệu có thể được xác định bởi một bảng nội dung các thay đổi tại đầu mỗi tài liệu. Bảng nội dung các thay đổi bao gồm ngày hiệu lực, các phần bị ảnh hưởng của tài liệu và một tóm tắt về thay đổi. Mục đích của bảng nội dung các thay đổi là để tóm tắt tất cả các thay đổi cho người đọc.

Các thay đổi nên được đánh dấu đỏ trong văn bản để xác định được bộ phận thay đổi. Nếu nội dung cũ là quan trọng thì nó có thể được đưa vào trong chú ý, được ghi ngày tháng và được dán nhãn là phiên bản trước. Cần nhớ rằng bạn cũng phải giữ tài liệu phiên bản cũ để dùng cho quá trình phát triển.

CÂU HỎI ÔN TẬP

1. Có mấy loại hoạt động bảo trì?
2. Khi thực hiện bảo trì phần mềm, những yếu tố nào được đánh giá là quan trọng?
3. Việc bảo trì phần mềm có thể gây ra những hiệu ứng gì?
4. Hãy nêu các hình thức bảo trì.
5. Hãy nêu những thủ tục cơ bản trong quản lý thay đổi. Yếu tố nào là quan trọng nhất trong quản lý thay đổi?

BÀI TẬP

HỆ THỐNG CHO THUÊ ĐĨA TỰ ĐỘNG AL2010

Một công ty cho thuê đĩa DVD muốn mở rộng mạng lưới kinh doanh ở nhiều điểm khác nhau bằng cách lắp đặt các hệ thống máy cho thuê tự động có tên AL2010. Các hệ thống này sẽ được lắp đặt tại những trung tâm thương mại nhỏ, các trung tâm vui chơi giải trí...

Các hệ thống máy này phải rất độc lập, có thể truy cập 24/24h và dễ sử dụng. Công ty chỉ can thiệp vào hệ thống máy trong những trường hợp đặc biệt. Còn để xác nhận xem các hệ thống máy hoạt động tốt hay không và một vài hoạt động bảo trì thường xuyên khác có thể được thực hiện từ xa thông qua một đường tín hiệu riêng. Không có mối liên hệ giữa các hệ thống máy khác nhau được lắp đặt tại các địa điểm khác nhau.

AL2010 lưu trữ được tối đa 100 đĩa DVD. Việc lựa chọn các đĩa DVD để đưa vào các hệ thống máy tính do nhân viên trong các chi nhánh thực hiện, thông qua một danh sách các phim mà công ty có. Danh sách này có thể lên tới hàng nghìn bản ghi và công ty cũng đưa ra một danh sách phim mới phát hành trong tháng.

Việc nhận và trả đĩa của nhân viên chi nhánh được thực hiện qua bưu điện, những đơn đặt hàng đặc biệt cũng được thực hiện qua bưu điện.

Hình thức và giao diện của các hệ thống máy tự động

Về mặt vật lý, các hệ thống máy này gần giống như các hệ thống ATM, chỉ nhân viên của công ty mới mở được máy để sửa chữa. Ở đằng sau máy, chỉ có 2 đường giây, một đường điện và một đường tín hiệu. Người sử dụng chỉ tiếp xúc với mặt trước của máy: (1) màn hình cảm ứng; (2) cửa sổ ra đĩa DVD; (3) cửa đưa thẻ vào. Màn hình đối diện với người sử dụng. Đầu ra đĩa DVD nằm bên dưới màn hình, chỗ đưa thẻ vào nằm bên tay phải màn hình.

AL2010 có thể đưa ra những phiên bản mới, phần mềm phải có khả năng đáp ứng yêu cầu này với chi phí thấp nhất.

Hệ thống máy tự động được cung cấp bởi một công ty khác, nó có tích hợp một thẻ có khả năng xử lý như CPU và có cả chức năng lưu trữ dữ liệu giống như một máy tính. Công ty có nhu cầu xây dựng một hệ thống phần mềm nhúng AL2010 để thực hiện được tất cả các chức năng cho thuê đĩa DVD và cho phép bảo trì hệ thống (chủ yếu là bảo trì từ xa). Các hoạt động bảo trì không liên quan đến dự án.

Tuy nhiên, thẻ khách hàng lại được cung cấp bởi một tổ chức khác. Trong thẻ có một thẻ nhớ để lưu số thẻ, thông tin khách hàng và số tiền còn lại trong thẻ.

Giao diện khách hàng

Màn hình cảm ứng cho phép người sử dụng:

- Xem danh sách các phim hiện có trong máy, với mỗi phim có tóm tắt, poster, đạo diễn và các diễn viên chính trong phim.

- Xem danh sách các phim mà công ty sở hữu và có thể đặt trước nếu muốn (chức năng này chỉ cung cấp cho những người có thẻ khách hàng).

- Xem danh sách các phim được nhiều người mượn nhất trong tuần và trong năm.

- Mượn một hoặc nhiều phim.

- Yêu cầu một hoặc nhiều phim.

- Nạp tiền vào thẻ.

- Truy cập vào các thông tin khác nhau trong tài khoản khách hàng.

Ngay khi người sử dụng đưa thẻ khách hàng vào, tên, số tài khoản và tình trạng của thẻ phải được hiển thị. Giao diện phải chỉ ra được khách hàng đang giao tiếp với máy ở giai đoạn nào (ví dụ như đang lựa chọn phim hay đang tiến hành các bước để mượn DVD) và cho phép quay trở lại bước trước đó.

Danh sách các phim mà công ty sở hữu là rất lớn, khách hàng có thể tìm kiếm theo thứ tự ABC, theo loại phim, theo tên đạo diễn, tên diễn viên. Khách hàng có thẻ của hệ thống có thể yêu cầu một bộ phim nằm trong danh sách các phim mà công ty sở hữu (danh sách này được cài đặt trong AL2010). Công ty có thể cập nhật danh sách phim, công việc này được thực hiện từ xa, một tháng một lần vào ban đêm.

Một giao dịch được thực hiện không quá 10 phút. Nếu quá thời gian này, hệ thống sẽ tự ngắt.

Giao diện của người quản trị máy

Người quản trị máy AL2010 có thể truy nhập vào một giao diện riêng của họ bằng việc nhập vào mật mã, giao diện này cho phép:

- Rút ra 1 hoặc nhiều đĩa từ máy.

- Bỏ vào máy một hoặc nhiều đĩa.

- Thu được những thông tin về tình trạng hoạt động của máy (ví dụ: số lượng đĩa hiện có trong máy, tỷ lệ các yêu cầu bị huỷ bỏ, tỷ lệ thuê đĩa, tiền phạt, tiền thu được, số lượng đĩa bị hỏng, tình trạng hỏng...).

- Xem xét yêu cầu của khách hàng.

- Xem các đĩa bị hỏng.

- Xác định giá cho thuê đĩa hiện tại (tùy thuộc vào loại phim, phim mới hay cũ...).

Khi người quản trị kết nối với máy AL2010 (bằng mật mã), các chức năng của hệ thống phải hiển thị và có thông báo nếu có đĩa trong tình trạng bị hỏng.

Khách hàng thường xuyên

Khách hàng có thể đăng ký, nếu họ muốn một hoặc nhiều thẻ khách hàng, các thẻ này được cung cấp bởi người bán hàng ở nơi đặt máy (để mua được thẻ này thì phải trả tiền qua thẻ ATM). Ban đầu, mỗi thẻ phải có tối thiểu 100.000 đồng. Thời gian sử dụng thẻ là không giới hạn.

Máy AL2010 cho phép nạp tiền vào thẻ khách hàng từ thẻ ngân hàng. Cụ thể hơn, khách hàng đưa thẻ khách hàng vào máy, sau đó đưa thẻ ngân hàng, nhập mật khẩu thẻ ngân hàng và xác định số tiền sẽ nạp cho thẻ. Số tiền này tối thiểu là 20.000 và tối đa là 200.000. Thẻ khách hàng phải luôn luôn có một số tiền tối thiểu để duy trì tình trạng hoạt động của thẻ. Tất cả thông tin về các thao tác nạp tiền vào thẻ phải được lưu trữ trong thẻ ít nhất 1 năm.

Hệ thống không quản lý được số tiền còn lại trong thẻ, nên trong trường hợp bị mất thẻ sẽ không được bồi thường tiền. Điều này giúp cho một khách hàng có thể sở hữu nhiều thẻ khác nhau, cũng như có thể cho người khác mượn thẻ.

Mỗi thẻ khách hàng cũng có lưu những thông tin mang tính lịch sử như danh sách các phim đã được mượn bằng thẻ này (thông tin này có ích để chủ sở hữu thẻ kiểm duyệt được thông tin trong trường hợp cho người khác mượn thẻ). Tuy nhiên chức năng này có thể huỷ bỏ nếu khách hàng thấy không cần thiết.

Nếu khách hàng thuê trên 20 đĩa trong một tháng, thì họ sẽ có thêm 50.000 tiền khuyến mại vào tài khoản trong thẻ.

Mượn đĩa phim

Khách hàng có thể mượn đĩa bằng thẻ khách hàng hoặc thẻ ngân hàng, tiền phải trả tuỳ thuộc vào thời gian mượn và tình trạng của phim (phim mới phát hành hay phim cũ). Nhìn chung, giá tiền thuê đĩa khi sử dụng thẻ ngân hàng sẽ cao hơn là sử dụng thẻ khách hàng.

Đối với khách hàng có thẻ khách hàng thì một lần có thể mượn tối đa là 3 phim, còn với người sử dụng thẻ ngân hàng thì chỉ được mượn 1 đĩa/1 lần.

Trả đĩa

Khi trả đĩa, trước tiên khách hàng phải đưa vào thẻ khách hàng hoặc thẻ ngân hàng, sau đó đưa vào đĩa DVD. Hệ thống sẽ kiểm tra tình trạng của đĩa. Nếu đĩa ở trong tình trạng tốt, hệ thống sẽ trả lại tiền đặt cọc vào tài khoản, ngược lại thì không được trả lại tiền.

Hệ thống cũng kiểm tra xem người mượn có mượn quá thời hạn đăng ký hay không. Nếu quá, thì hệ thống sẽ tính toán tiền phạt tuỳ theo số ngày quá hạn. Số tiền này sẽ bị trừ vào tiền đặt cọc.

Số tiền trong thẻ khách hàng có thể âm. Khách hàng có 7 ngày để nạp tiền vào thẻ, nếu không tiền trong thẻ ngân hàng sẽ tự động bị trừ, trong trường hợp này sẽ phải mất thêm tiền phạt.

Người mượn cũng có thể khai báo tình trạng hỏng cuả đĩa. Trong trường hợp này, đĩa DVD sẽ không được mượn nữa và tình trạng hỏng hóc này sẽ được thông báo đến các kỹ thuật viên để xem xét, nếu đĩa không bị hỏng, khách hàng sẽ được trả lại tiền. Nếu mã ở trên đĩa bị hỏng, khách hàng phải thông báo đến người quản lý hệ thống (địa chỉ và số điện thoại được ghi trên máy AL2010).

CÂU HỎI ÔN TẬP

1. Hãy xây dựng đội dự án và viết bản kế hoạch dự án để phát triển hệ thống phần mềm AL2010.

2. Viết bản đặc tả phần mềm cho hệ thống AL2010.

3. Với hệ thống trên thì ta có thể sử dụng mô hình kiến trúc nào? Hãy phân tích và xây dưng mô hình kiến trúc cho hệ thống phần mềm AL2010.

4. Phân tích và thiết kế hệ thống phần mềm AL2010 theo phương pháp hướng đối tượng.

5. Xây dựng kế hoạch kiểm thử và các kịch bản kiểm thử để kiểm thử hệ thống AL2010.

TÀI LIỆU THAM KHẢO

[1] Project Management Institute (2000). *A guide to the Project Management Body of Knowledge 2000 Edition*, Newtown Square, Pennsylvania USA.

[2] Nguyễn Việt Hà (2007). *Bài giảng nhập môn kỹ thuật phần mềm*, Đại học Quốc gia Hà nội.

[3] Philippe Lalanda (2008). *Cours de Génie Logiciel Introduction*, Université Joseph Fourier - Grenoble.

[4] Nguyễn Văn Vị, Nguyễn Việt Hà (2010). *Giáo trình Kỹ nghệ phần mềm*, NXB giáo dục Việt Nam, 2010.

[5] Huỳnh Văn Đức, Đoàn Thiện Ngân (2004). *Giáo trình nhập môn UML*, NXB Lao động Xã hội.

[6] Thạc Bình Cường, Nguyễn Đức Mận (2011). *Kiểm thử và bảo đảm chất lượng phần mềm*, NXB Đại học Bách Khoa Hà Nội.

[7] Ngô Trung Việt (2005). *Phương pháp luận quản lý dự án CNTT*, NXB Khoa học kỹ thuật.

[8] Dương Kiều Hoa, Tôn Thất Hòa An (2003). *Phân tích và thiết kế hệ thống thông tin với UML*, NXB Đại học Huế.

[9] Nguyễn Hữu Quốc (2007). *Quản lý dự án*, Học viện công nghệ bưu chính viễn thông.

[10] Ian Sommerville (2006). *Software Engineering 8th Edition*, Addsion Wasley.