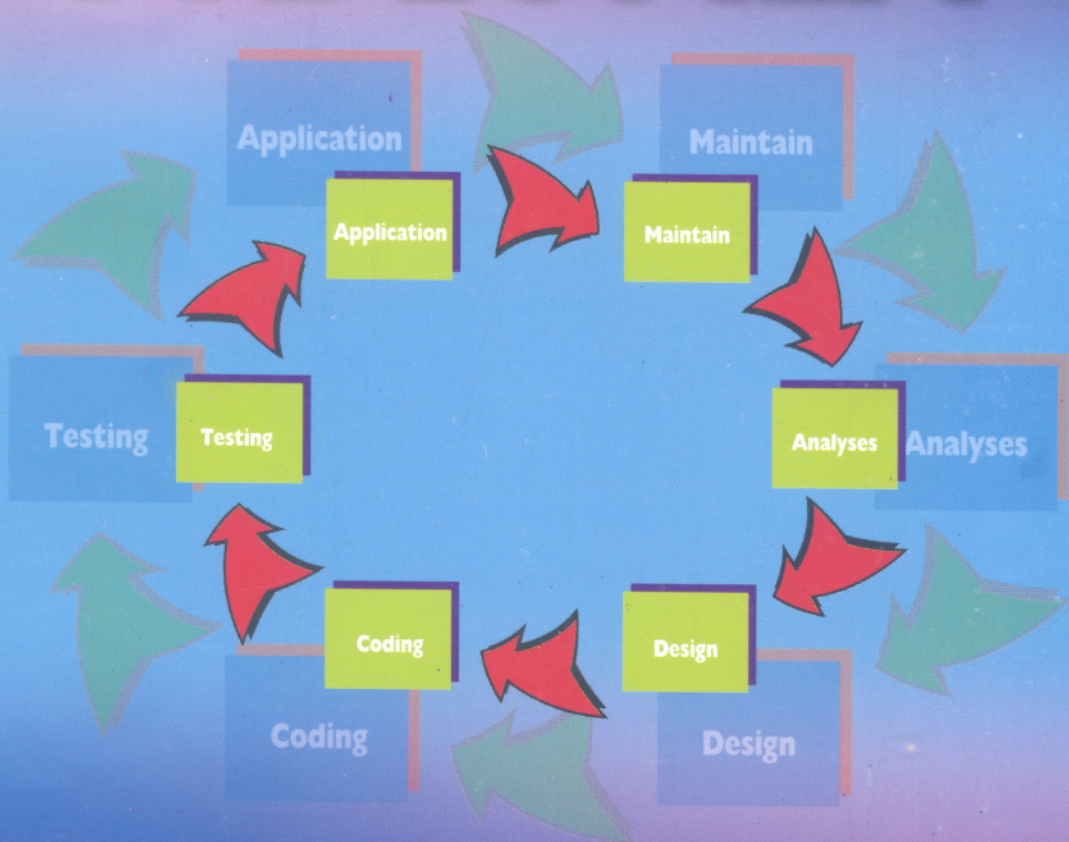


NGUYỄN VĂN VY - NGUYỄN VIỆT HÀ

GIÁO TRÌNH KỸ NGHỆ PHẦN MỀM



NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

PGS.TS. NGUYỄN VĂN VY – TS. NGUYỄN VIỆT HÀ

Giáo trình

KỸ NGHỆ PHẦN MỀM

NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

Công ty Cổ phần Sách Đại học – Dạy nghề, Nhà xuất bản Giáo dục Việt Nam giữ quyền công bố tác phẩm.

161 – 2009/CXB/14 – 208/GD

Mã số: 7B737Y9 – DAI

LỜI NÓI ĐẦU

Sau hơn 3 thập kỷ phát triển, kỹ nghệ phần mềm (*SE – Software Engineering*) đến nay được xem như một phân ngành quan trọng của ngành Tin học (Computing).

Trong công nghiệp phần mềm, người xây dựng phần mềm gọi là kỹ sư phần mềm. Kỹ nghệ phần mềm được xem như là một tên gọi chỉ cách thức làm phần mềm một cách công nghiệp, là những công việc dành cho kỹ sư phần mềm. Các phương pháp, công cụ, thủ tục của kỹ nghệ phần mềm đã được chấp nhận là những nội dung chính yếu và ứng dụng thành công trong rất nhiều lĩnh vực ứng dụng công nghiệp phần mềm. 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. *Kỹ nghệ phần mềm* không đơn thuần là việc sản sinh ra sản phẩm phần mềm, mà nó liên quan đến việc tạo ra sản phẩm phần mềm một cách hiệu quả. Với những nguồn lực không hạn chế, thì đa số các vấn đề trong phần mềm đều có thể giải quyết được. Thử thách đối với kỹ sư phần mềm là phải tạo ra phần mềm chất lượng cao, với những hạn chế về nguồn lực và phải tuân thủ một lịch trình định trước.

Giáo trình này trình bày những vấn đề cơ bản sau đây của kỹ nghệ phần mềm:

1. *Phân tích hệ thống và yêu cầu phần mềm*: Nghiên cứu một hệ thống thực và chuyển các yêu cầu về một phần mềm thành một bản đặc tả hệ thống phần mềm, với việc vận dụng các mô hình, phương pháp khác nhau.
2. *Thiết kế và cài đặt phần mềm*: Nhấn mạnh tới việc chuyển đặc tả yêu cầu thành các bản thiết kế phần mềm ở các mức khác nhau, thành chương trình nhờ các giải pháp công nghệ và công cụ được sử dụng một cách có hệ thống.
3. *Đảm bảo, kiểm chứng và duy trì tính toàn vẹn phần mềm*: Nhấn mạnh vào các hoạt động và kỹ thuật được ứng dụng để đảm bảo chất lượng của phần mềm trong suốt tiến trình phát triển.

4. *Tự động hóa các hoạt động phát triển phần mềm*: Nhấn mạnh sự trợ giúp đặc lực của công cụ phần mềm máy tính cho tiến trình phát triển phần mềm nhằm đạt được năng suất và chất lượng cao.
5. *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.

Giáo trình này là sự bổ sung và hoàn chỉnh *Giáo trình kỹ nghệ phần mềm* của Trường Đại học Công nghệ (Đại học Quốc gia Hà Nội). Với nội dung có tính cơ sở nền tảng, không chỉ cần thiết cho các kỹ sư làm phần mềm mà rất có ích cho một phạm vi rộng rãi các đọc giả. Nó cũng cung cấp các kiến thức có tính nền tảng cho việc tiếp thu và nâng cao trình độ sau này của những người hoạt động trong lĩnh vực tin học, đặc biệt các học viên cao học về công nghệ thông tin, hệ thống thông tin.

Các tác giả chân thành cảm ơn các đồng nghiệp của Khoa Công nghệ thông tin, Trường Đại học Công nghệ đã cho những ý kiến đóng góp để hoàn thiện tài liệu này. Đặc biệt xin cảm ơn ThS. Đào Kiến Quốc, ThS. Vũ Diệu Hương, PGS.TS. Hà Quang Thụy, TS. Phạm Hồng Thái, ThS. Nguyễn Quang Vinh (Khoa Công nghệ thông tin, Trường Đại học Công nghệ), PGS.TS. Đặng Văn Đức, TS. Lê Văn Phùng (Viện Công nghệ thông tin, Viện Khoa học và Công nghệ Việt Nam) và các đồng nghiệp đã cung cấp tài liệu và có những đóng góp cụ thể, thiết thực và hữu ích.

Mặc dù đã rất cố gắng tham khảo nhiều tài liệu khác nhau, đúc rút nhiều kinh nghiệm từ các hoạt động thực tiễn, tổ chức biên soạn một cách công phu và kỹ càng, bổ sung nhiều lần, nhưng giáo trình khó tránh khỏi thiếu sót. Chúng tôi chân thành mong đợi những đánh giá, nhận xét và đóng góp của các bạn đồng nghiệp, học viên và các đọc giả để cuốn sách được hoàn thiện hơn trong lần xuất bản sau.

Thư góp ý xin gửi về: Công ty Cổ phần Sách Đại học – Dạy nghề, 25 Hàn Thuyên, Hà Nội.

CÁC TÁC GIẢ

CHỮ VIẾT TẮT

CAM/CAD	Computer Aided Manufacture/Design
CASE	Computer Aided Software Engineering
CBS	Computer – Based System
CBSE	Component – Based Software Engineering
CMM	Capability Maturity Model
CMMI	Capability Maturity Model, Integrattion
CPM	Critical Path Method
CSDL	Cơ sở dữ liệu/Database
DFD	Data Flow Diagram
ERM	Entity – Relationships Model
FSM	Finite State Machine
HCI	Human Computer Interface
IEEE	Institute Electrical and Electronic Engineers
ISO	International Standards Organization
MTTR	Mean Time To Repair
OSI	Open Systems Interconnection (reference model)
PDL	Program Decription Language
PERT	Program Evaluation and Review Technicques
RAD	Rapid Applicaron Development
RMM	Reuse Maturity Model
RUP	Rational Unified Process
SE	Software Engineering
STD	State Transition Diagram
SQA	Software Quality Assuarance
UML	Unified Modeling Language
USD	US đôla
VND	Việt Nam đồng
VORD	Viewpoint – Oriented Requirements Definition
V&V	Verification and Validation

Chương 1

PHẦN MỀM VÀ KỸ NGHỆ PHẦN MỀM

Ngày nay, khi hàng triệu máy tính lớn và máy tính cá nhân được đưa vào sử dụng, thì các hệ phần mềm hoạt động trên các máy trở thành điều kiện sống còn của các máy tính đó. Hoạt động của con người, mỗi tổ chức, mỗi nước và toàn cầu ngày càng phụ thuộc nhiều vào các máy tính cùng các hệ phần mềm giúp nó hoạt động. Với mức tăng của giá trị phần mềm hằng năm là 12%, hiệu quả của các nền kinh tế công nghiệp hóa phụ thuộc rất lớn vào hiệu quả hoạt động của các phần mềm trên các hệ thống máy tính. Một sự trục trặc của các hệ thống phần mềm chỉ trong một thời gian ngắn, có thể làm nhiều hoạt động ngừng trệ và mất đi hàng triệu đôla. Các cá nhân sử dụng máy tính ngày càng nhiều và càng không thể thiếu nó trong công việc và cuộc sống sinh hoạt hằng ngày. Trong những nền kinh tế phát triển, thu nhập của các hoạt động dịch vụ do các phần mềm máy tính mang lại ngày càng gia tăng và đang chiếm một tỷ trọng đáng kể trong thu nhập quốc dân của mỗi nước (và người giàu nhất thế giới là Bill Gate, Chủ tịch hãng phần mềm Microsoft với tài sản lên tới 50 tỷ đôla Mỹ).

Trong mỗi tổ chức, mỗi hệ thống, sự khác biệt của các phần mềm được sử dụng so với các phần mềm nơi khác đã quyết định sự khác biệt về phong cách làm việc, về năng lực tổ chức và sức cạnh tranh của nó so với các hệ thống và các tổ chức khác. Hammer và Champy (1993) đã nhận định rằng: *"Công nghệ thông tin đã đóng vai trò sống còn trong việc tái thiết lại các công ty"*.

Vai trò của phần mềm không ngừng mở rộng, như Norman (1998) và Levy (1999) đã viết: *"Ngày nay, hoạt động tính toán có mặt khắp mọi nơi đã sản sinh ra một thế hệ những người ứng dụng thông tin sử dụng web để cung cấp một mạng lưới thông tin bao phủ liên kết mọi nhà, mọi công sở đến các xa lộ"* [25].

1.1. Khái niệm về phần mềm

1.1.1. Khái niệm

Phần mềm thường được mô tả với ba bộ phận cấu thành:

- *Tập các lệnh* (chương trình máy tính) trên máy tính khi được thực hiện sẽ tạo ra các dịch vụ và đem lại những kết quả mong muốn cho người dùng.
- *Các cấu trúc dữ liệu* (lưu giữ trên các bộ nhớ) làm cho chương trình thao tác hiệu quả với các thông tin thích hợp và nội dung thông tin được số hóa⁽¹⁾.
- *Các tài liệu* để mô tả thao tác, cách sử dụng và bảo trì phần mềm (hướng dẫn sử dụng, tài liệu kỹ thuật, tài liệu phân tích, thiết kế, kiểm thử,...).

1.1.2. Các đặc trưng của phần mềm

Phần mềm là phần tử hệ thống logic, không phải là hệ thống vật lý. Nó có các đặc trưng khác biệt đáng kể so với các đặc trưng của phần cứng.

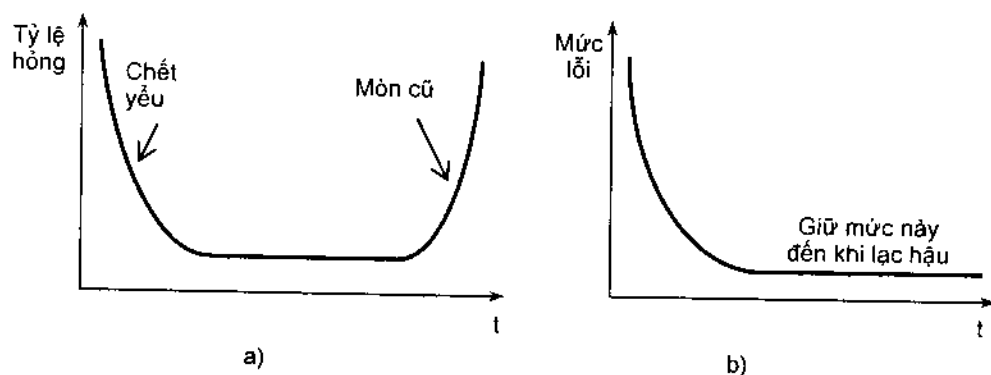
a) Phần mềm được kỹ nghệ, không được chế tạo theo nghĩa cổ điển

Phần mềm cũng được thiết kế, chế tạo như phần cứng, nhưng nó không được định hình trước. Chỉ khi phát triển xong người ta mới có sản phẩm cụ thể và biết được nó hoạt động có hiệu quả hay không. Quá trình thiết kế và sản xuất phần mềm phụ thuộc vào con người, vào điều kiện môi trường cụ thể mà tại đó nó được phát triển. Người ta không thể nói trước được giá thành của phần mềm và hiệu quả của nó. Chính quá trình phát triển phần mềm quyết định giá thành và chất lượng của nó.

b) Phần mềm không "hồng đĩ" nhưng "thoái hóa" theo thời gian

Phần mềm không bị ảnh hưởng từ những tác động của môi trường vốn gây cho phần cứng bị mòn cũ (hình 1.1a). Về mặt lý thuyết, tỷ lệ lỗi của phần mềm là không đổi (hình 1.1b) khi đã đưa vào sử dụng. Nhưng trên thực tế, nó lại "thoái hóa" theo thời gian do lỗi mới sinh ra mỗi khi tiến hành bảo trì trong quá trình sử dụng.

⁽¹⁾ Tham khảo Quyết định 128/2000/QĐ – TTg của Thủ tướng Chính phủ.

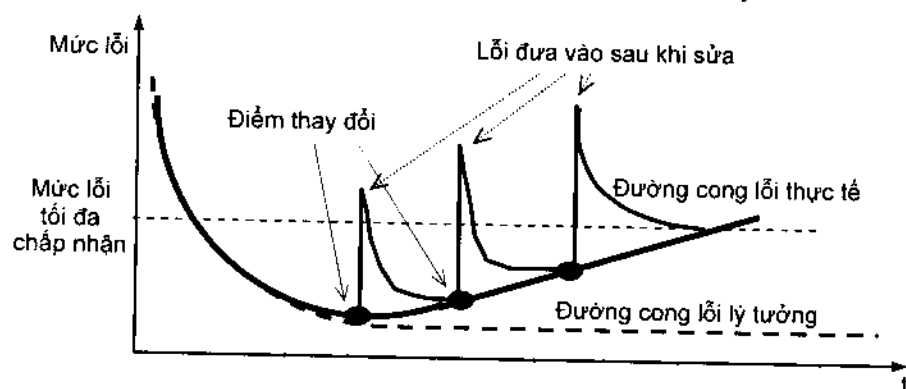


Hình 1.1

- a) Đường cong hỏng hóc của phần cứng cho phần mềm;
b) Đường cong lỗi lý tưởng cho phần mềm.

Khi sửa đổi (bảo trì) phần mềm, một số khiếm khuyết mới sinh ra, làm cho đường cong tỷ lệ lỗi trở thành có đầu nhọn (hình 1.2). Trước khi đường cong lỗi trở về tỷ lệ lỗi ổn định, thì một thay đổi khác lại được yêu cầu và việc bổ sung, sửa đổi chương trình được thực hiện, làm cho đường cong lỗi phát sinh có đỉnh nhọn tiếp theo. Dần dần, mức lỗi tối thiểu tăng lên – phần mềm bị "thoái hóa" do tỷ lệ sai hỏng tăng lên đến mức gây ra những thiệt hại không thể chấp nhận được, hoặc người dùng không muốn sử dụng nữa, vì có những phần mềm khác tốt hơn.

Phần mềm còn lạc hậu do các công nghệ mới ra đời, người dùng không lựa chọn phần mềm cũ nữa, hay khi các công nghệ mới (hệ điều hành, các hệ quản trị cơ sở dữ liệu, phiên bản dịch mới,...) được đưa vào tổ chức, làm cho các phần mềm cũ không thể tiếp tục vận hành trên máy tính.



Hình 1.2. Đường cong lỗi thực tế của phần mềm

c) Phần lớn phần mềm vẫn được xây dựng theo đơn đặt hàng

Phần mềm không có danh mục các thành phần cố định như phần cứng. Nó thường được đặt hàng theo một đơn vị hoàn chỉnh, theo yêu cầu riêng của mỗi tổ chức, mỗi khách hàng, ít khi có thể lắp ráp theo một khuôn mẫu có sẵn. Ngày nay, nhờ công nghệ hướng đối tượng ra đời, việc phát triển phần mềm dựa trên thành phần có sẵn (*software reuse*) đã được đề xuất, nhưng mới ở giai đoạn ban đầu.

Yêu cầu đối với phần mềm của khách hàng tùy thuộc vào môi trường cụ thể, mà ở đó nó được phát triển và sử dụng. Môi trường này (phần cứng, phần mềm nền, con người và tổ chức, công nghệ có được,...) không thể định dạng từ trước và luôn thay đổi theo không gian và thời gian. Tính bất định của môi trường quy định sự bất định của phần mềm cho đến khi nó được xây dựng xong.

d) Sự phức tạp và tính thay đổi luôn là bản chất của phần mềm

Phần mềm không phải là thực thể vật lý, mà bản chất là những khái niệm được thể hiện bằng *một hệ thống logic* dưới các ngôn ngữ khác nhau và được lưu trên giấy hay vật mang. Do đó, nó là khó hiểu vì nhận thức được nó phải thông qua các khái niệm và quá trình tư duy trừu tượng về những gì mà nó được mô tả. Mặt khác, nó là một *mô hình của thế giới thực*. Bản chất đa dạng của thế giới thực không cho phép mỗi người có đủ kiến thức để có thể hiểu nó một cách dễ dàng. Phần mềm là "*không nhìn thấy được*", mà chỉ có thể nhận biết qua sự mô tả từ những khía cạnh khác nhau (sơ đồ điều khiển, sơ đồ luồng dữ liệu, sơ đồ tương tác giữa các môđun...) của nó. Hơn nữa, thế giới thực (mô hình tổ chức và quản lý, các tiến trình nghiệp vụ, các điều kiện công nghệ, môi trường kinh tế – xã hội) luôn thay đổi theo thời gian và không gian, và ngày càng diễn ra một cách nhanh chóng hơn. Phần mềm là mô hình của thế giới thực để phục vụ con người, nên nó chỉ có ý nghĩa khi thích nghi được với sự thay đổi đã diễn ra, tức là nó phải thay đổi một cách tương ứng cùng thời gian. *Thay đổi trở thành yêu cầu và thuộc tính tất yếu của phần mềm.*

e) Ngày nay phần mềm được phát triển theo nhóm

Càng ngày các phần mềm càng có quy mô lớn, mỗi thành phần của nó thường đòi hỏi những kỹ năng khác nhau để thực hiện. Cùng với sự phát

triển nhanh chóng của công nghệ, khả năng hiểu biết và kỹ năng của những người phát triển với các công nghệ mới bị giới hạn trong một phạm vi nhất định. Trong khi đó, nhu cầu về sản phẩm ngày càng tăng và đòi hỏi thời gian sản xuất ra ngày càng phải được rút ngắn. Trong điều kiện đó, việc sản xuất phần mềm cần nhiều người với kỹ năng khác nhau, tham gia trong các đội phát triển, để tạo được phần mềm chất lượng cao với thời gian ngắn. Phát triển phần mềm theo nhóm là một cách lựa chọn duy nhất.

1.1.3. Các thành phần phần mềm

Phần mềm máy tính là thông tin tồn tại dưới hai dạng: *thành phần máy thực hiện được* và *thành phần máy không thực hiện được*. Thành phần phần mềm hoạt động trên máy gồm các câu lệnh được xây dựng trên một *ngôn ngữ lập trình* nào đó, để chuyển hóa một tập các bước thủ tục đã xác định trước và cấu trúc dữ liệu thành chương trình. Có nhiều loại ngôn ngữ lập trình khác nhau dùng cho việc phát triển phần mềm. Đến nay, các ngôn ngữ lập trình đã phát triển đến thế hệ thứ tư. Các thế hệ ngôn ngữ đã được biết đến nay là: *ngôn ngữ máy, hợp ngữ, các ngôn ngữ bậc cao, ngôn ngữ thế hệ thứ tư*.

Việc sản xuất ra các phần mềm là một quá trình phức tạp và tốn kém. Rất nhiều vấn đề nảy sinh khi một phần mềm cần được xây dựng. Vì vậy, người ta cố gắng dùng lại các thành phần đã có. Việc sử dụng lại làm rút ngắn thời gian và chi phí chế tạo và bảo trì phần mềm, đảm bảo độ tin cậy cao. Nhờ vậy có thể sản xuất nhanh, nhiều phần mềm một cách hiệu quả. Chính vì vậy, *tính sử dụng lại* của phần mềm được xem như một đặc trưng quan trọng của việc phát triển phần mềm chất lượng cao.

Ngày nay, khái niệm sử dụng lại bao hàm không chỉ các mã nguồn (các hàm, thủ tục), mà còn cả các mô hình phân tích, cấu trúc dữ liệu và xử lý trong một bộ trình riêng, hay các mẫu thiết kế, mẫu kiến trúc. Vì thế, khả năng sử dụng lại các thành phần phần mềm rất rộng rãi, công sức dành cho hoạt động lập trình giảm đi rất nhiều. Ví dụ, các giao diện tương tác thường được xây dựng bằng cách sử dụng các thành phần dùng lại, để tạo ra cửa sổ đồ họa như thực đơn kéo thả và rất nhiều cơ chế tương tác khác.

Các thành phần không thực hiện được của phần mềm gồm các phần giải thích trong các chương trình phần mềm, các tài liệu phân tích, thiết kế,

các hướng dẫn sử dụng và khai thác nó,... Với những công nghệ mới và những công cụ mới, một phần không nhỏ của thành phần này đã được tự động hóa. Tuy nhiên, cho đến nay việc tạo ra các thành phần này của phần mềm chủ yếu vẫn làm thủ công, một công việc vất vả và đang là vấn đề nan giải trong phát triển phần mềm.

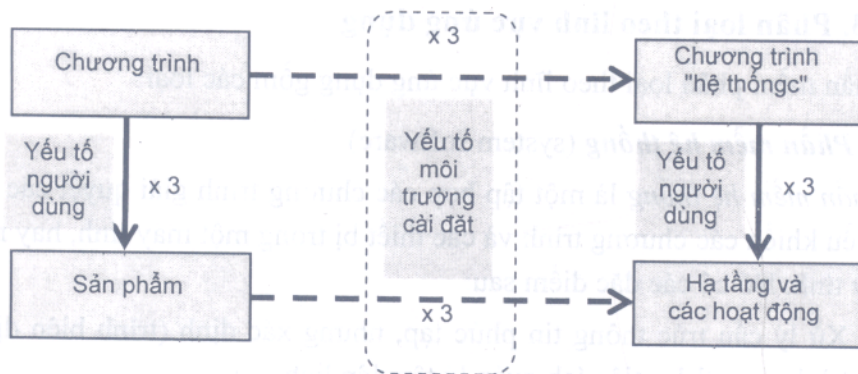
1.1.4. Phân loại các phần mềm

Có thể phân loại các phần mềm theo một số cách:

- Theo mức độ hoàn thiện của phần mềm được xây dựng;
- Theo vai trò – chức năng mà phần mềm thực hiện;
- Theo lĩnh vực được ứng dụng.

1.1.4.1. Phân loại theo mức độ hoàn thiện

Một phần mềm có thể đơn giản là một chương trình (*program*) do một người làm ra cho chính mình dùng, hay là một sản phẩm (*product*) được làm ra cho nhiều người dùng. Khi chương trình làm ra cho một người dùng thường thiếu tài liệu sử dụng, chưa tính đến các yêu cầu khác nhau của người dùng như các tính năng, sự tiện dụng, thói quen,... Vì vậy, để nâng cấp một chương trình thành một sản phẩm, cần đòi hỏi thêm nhiều công sức và kinh nghiệm. Một phần mềm có thể chỉ hoạt động trong một môi trường xác định mà ở đó nó được phát triển, đó là *phần mềm đơn lẻ*. Một phần mềm có thể phát triển để hoạt động trong nhiều môi trường có sự khác nhau nhất định. Trong trường hợp này, một số yếu tố của hệ thống liên quan (số tệp được mở, dung lượng bộ đệm, bộ nhớ sử dụng, thiết bị ngoại vi sử dụng, tương tác với các phần mềm khác, phong chữ,...) được đưa vào phần mềm, nên nó có thể chạy được trên những môi trường khác nhau. Phần mềm như vậy gọi là *phần mềm mang tính hệ thống*, hay *phần mềm trong một hệ thống*. Từ *hệ thống* ở đây chỉ một tổng thể các phần mềm, môi trường triển khai, quy trình và người sử dụng (một *Hệ thống thông tin*) chứ không phải là hệ điều hành. Từ phần mềm đơn lẻ nâng cấp thành phần mềm mang tính hệ thống cần thêm nhiều công sức, bao gồm cả kiến thức và kinh nghiệm (hình 1.3).



1.1.4.2. Phân loại theo vai trò – chức năng

Các phần mềm được phân loại theo vai trò – chức năng gồm ba loại:

a) Phần mềm hệ thống (system software)

Phần mềm hệ thống là một tập hợp các chương trình thực hiện chức năng quản lý, điều khiển các nguồn lực của máy tính và các chương trình khác nhau chạy trên các máy. Chúng bao gồm các hệ điều hành, các hệ tiện ích hệ thống.

b) Phần mềm công cụ (tools)

Phần mềm công cụ là các phần mềm trợ giúp quá trình tạo ra các phần mềm. Chúng bao gồm các công cụ trợ giúp quá trình phát triển phần mềm bằng máy tính (chẳng hạn như CASE – Computer Aided Software Engineering), các ngôn ngữ lập trình (trình soạn thảo, trình soát lỗi, chương trình dịch, thư viện chương trình, chương trình liên kết,...). Nhờ các phần mềm công cụ này, mà quá trình phát triển phần mềm ngày càng dễ dàng hơn, đơn giản hơn và nhanh chóng hơn. Nhờ những trợ giúp hiệu quả, nhiều hoạt động phát triển được tự động hóa một phần hay toàn bộ.

c) Phần mềm ứng dụng (application softwares)

Phần mềm ứng dụng là các phần mềm trợ giúp thực hiện những nhiệm vụ trong các hoạt động nghiệp vụ khác nhau. Các phần mềm này được chia làm hai loại: *phần mềm ứng dụng đa năng* (xử lý văn bản, bảng tính, hệ quản trị cơ sở dữ liệu,...) và các *phần mềm ứng dụng chuyên biệt/chuyên dụng* (kế toán, marketing, nghe nhạc,...). Trong số này, nhiều phần mềm được các nhà phát triển thương mại hóa ở dạng "phần mềm đóng gói" (packaged softwares), làm sẵn cho người dùng.

1.1.4.3. Phân loại theo lĩnh vực ứng dụng

Phần mềm phân loại theo lĩnh vực ứng dụng gồm các loại:

a) Phần mềm hệ thống (system software)

Phần mềm hệ thống là một tập hợp các chương trình giải quyết các bài toán điều khiển các chương trình và các thiết bị trong một máy tính, hay một hệ máy tính. Nó có các đặc điểm sau:

- Xử lý cấu trúc thông tin phức tạp, nhưng xác định (trình biên dịch, trình soạn thảo, tiện ích quản lý tệp, lập lịch,...).
- Đặc trưng bởi sự tương tác chủ yếu với phần cứng máy tính: quản lý các tài nguyên (bộ nhớ) và hoạt động của thiết bị ngoại vi (máy in, ổ đĩa,...).
- Cấu trúc dữ liệu phức tạp và nhiều giao diện (để tương tác với các hệ khác).

b) Phần mềm thời gian thực (real – time software)

Phần mềm thời gian thực dùng để điều phối, phân tích, hoặc kiểm soát các sự kiện của thế giới thực ngay khi chúng xuất hiện, nên được gọi là *phần mềm thời gian thực*. Hệ thống thời gian thực phải đáp ứng với các sự kiện của môi trường trong các điều kiện ràng buộc về thời gian rất chặt chẽ. Các phần mềm loại này bao gồm các phần mềm điều khiển tự động, điều khiển rôbốt, các hệ quan sát,...

c) Phần mềm nghiệp vụ (business software)

Phần mềm nghiệp vụ dùng để xử lý thông tin nghiệp vụ là lĩnh vực ứng dụng phần mềm lớn nhất. Nó gồm các hệ thống thông tin, các hệ trợ giúp ra quyết định, các hệ xử lý nghiệp vụ theo quy tắc nghiệp vụ, bao gồm cả tính toán tương tác (như xử lý giao tác cho các điểm bán hàng)...

d) Phần mềm khoa học và công nghệ (engineering and scientific software)

Phần mềm loại này được đặc trưng bằng các thuật toán phức tạp và xử lý nhiều số liệu, nên còn được gọi là "máy nghiền số". Nó là những phần mềm trong các ứng dụng mới, thiết kế hay chế tạo có trợ giúp của máy tính (*CAD/CAM – Computer Aided Design/ Computer Aided Manufacture*), các mô phỏng, các tương tác khác có chú ý đến các đặc trưng thời gian thực và cả phần mềm hệ thống.

e) Phần mềm nhúng (embedded software)

Phần mềm nhúng nằm trong các chip gắn với thiết bị. Nó chỉ được kích hoạt để điều khiển sản phẩm và hệ thống cho người dùng, cho thị trường công nghiệp khi các hệ thống này hoạt động. Chúng có thể thực hiện các chức năng rất giới hạn và huyền bí (điều khiển bàn phím cho lò vi sóng) hay đưa ra các khả năng điều khiển và vận hành (chức năng số hóa ở ô tô, kiểm soát xăng, biểu thị bảng đồng hồ, hệ thống phanh). Xây dựng phần mềm nhúng gắn chặt với các đặc trưng của thiết bị, bao gồm cả cấu trúc và các đặc trưng xử lý và trao đổi thông tin.

f) Phần mềm máy tính cá nhân (personal computer software)

Phần mềm máy tính cá nhân phục vụ trực tiếp cho một cá nhân làm việc với một máy. Nó phát triển trong hơn hai thập kỷ qua, như các hệ xử lý văn bản, trang tính, đồ họa, hệ quản trị cơ sở dữ liệu, trò chơi,... Phần mềm loại này tiếp tục được phát triển và cải thiện không ngừng, đặc biệt là các giao diện người – máy.

g) Phần mềm trên nền web (web – based software)

Người ta có thể truy cập vào các trang web nhờ một phần mềm chuyên dụng, là sự phối hợp của các câu lệnh có thể thực hiện được (như CGI, HTML, Perl, hay Java) và các dữ liệu ở dạng siêu văn bản (Hypertext). Về bản chất, mạng toàn cầu gồm hàng loạt máy tính, đã cung cấp nguồn phần mềm không hạn chế, cho phép người dùng có thể truy cập tới mọi thông tin ở bất kỳ đâu trên mạng toàn cầu thông qua modem. Vì thế có thể nói rằng, ngày nay mọi người đều có thể tiếp cận được với mọi sự kiện của thế giới, cũng như với kho trí tuệ vô tận mà nhân loại đã và đang tích lũy được.

h) Phần mềm trí tuệ nhân tạo (artificial intelligent software)

Phần mềm loại này sử dụng các thuật toán phi số để giải quyết các vấn đề phức tạp thông qua tính toán, hay phân tích trực tiếp. Đó là các phần mềm như các hệ chuyên gia (hệ cơ sở tri thức), các phần mềm trong lĩnh vực nhận dạng (hình ảnh và tiếng nói), chứng minh định lý và chơi trò chơi, các hệ mạng nơron nhân tạo để mô phỏng cấu trúc của việc xử lý trong bộ óc người.

1.2. Sự phát triển của phần mềm và thách thức

Trước thập kỷ 90 của thế kỷ XX, phát triển 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 hơn vào sự cải thiện chất lượng và giảm giá thành của các giải pháp dựa trên máy tính, tức là các giải pháp được cài đặt bằng phần mềm. Sự cấp bách này do nhu cầu phần mềm ngày càng tăng lên nhanh cả về số lượng, quy mô, cũng như các tính năng.

1.2.1. Quá trình "tiến hóa" của phần mềm

Quá trình "tiến hóa" của phần mềm gắn liền với sự "tiến hóa" của phần cứng hệ thống máy tính. Phần cứng ngày càng mạnh hơn, kích cỡ nhỏ hơn và giá thành thấp hơn (từ bóng điện tử với vài trăm phép tính/giây đến mạch tích hợp xử lý hàng trăm triệu phép tính/giây). Theo định luật Moore: "Năng lực của máy tính tăng gấp đôi sau 18 tháng". Định luật này cho đến nay vẫn còn đúng và chúng ta đang chứng kiến sự tăng trưởng đến kỳ diệu trong kỹ thuật phần cứng. 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 đầu từ năm 1950 đến năm 1960

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 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) Thời kỳ từ những năm 1960 đến giữa những năm 1970

Đặc trưng của giai đoạn này là:

- 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 và phải xuất 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 phải làm thích nghi chương trình với những thay đổi của môi trường. *Công việc bảo trì phần mềm đặt ra bức xúc*, chúng tiêu tốn nhiều công sức và tài nguyên đến mức báo động.

c) Thời kỳ giữa những năm 1970 đến năm 1990

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 (bao gồm nhiều máy tính, mỗi máy thực hiện một chức năng và liên kết với các máy khác) *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* (ô tô, rô bốt, lò vi sóng, thiết bị chẩn 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, bao gồm cả lĩnh vực công nghiệp, dịch vụ và đời sống sinh hoạt.

Phần cứng ngày càng ổn định, chi phí phần mềm có khuynh hướng tăng nhanh hơn chi phí mua máy. Từ đó nảy sinh yêu cầu tăng năng suất sản xuất phần mềm.

Phương pháp luận và phương pháp phát triển phần mềm hướng cấu trúc đạt đến một mức hoàn thiện cao và cùng với nó là sự phát triển các công cụ

trợ giúp kỹ nghệ phần mềm bằng máy (CASE), đã làm tăng năng suất và chất lượng phần mềm một cách đáng kể.

d) Thời kỳ sau năm 1990 – thời kỳ thứ tư mới bắt đầu

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

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

Sự phát triển 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 hệ ứng dụng nghiệp vụ. *Công nghệ hướng đối tượng* (tiêu biểu như các hệ .NET) và *phát triển phần mềm theo hướng sử dụng lại* (Patterns và Frameworks) đang trở thành một xu hướng công nghệ. Tất cả những yếu tố trên tạo nên những thách thức mới cho việc phát triển phần mềm hiện nay.

1.2.2. Khủng hoảng phần mềm và sự ra đời của kỹ nghệ phần mềm

Nhìn lại sự "tiến hóa" của phần mềm như trên thấy rằng, nhiều vấn đề liên tục phát sinh, tạo ra những thách thức cho việc phát triển phần mềm:

- *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 một phần mềm (do phần mềm lớn).
- *Sự phụ thuộc nhiều vào kinh nghiệm* của người làm phần mềm.
- *Chất lượng của phần mềm* không ổn định do phụ thuộc vào con người.
- *Sự thiếu nghiêm trọng kỹ sư* làm 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 70 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 Kỹ nghệ phần mềm quốc tế (*ICSE – International Conference of SE*), nhiều lý thuyết, phương pháp luận và kỹ thuật được đề nghị. Vào những năm 90 của thế kỷ XX, *công cụ trợ giúp kỹ nghệ phần mềm bằng máy tính (CASE)* phát triển mạnh. Nhờ vậy, việc tự động hóa các quá trình phần mềm đã trở nên phổ biến trong các hoạt động phát triển phần mềm. 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 nó còn chưa giải quyết trọn vẹn. Vì vậy, kỹ nghệ phần mềm ra đời như một đòi hỏi tất yếu của sự phát triển phần mềm.

1.2.3. Thách thức đối với phát triển phần mềm máy tính hiện nay

Những trình bày trên cho thấy, những thách thức đối với phát triển phần mềm máy tính gia tăng trong những năm gần đây vì những nguyên nhân sau:

- Sự tinh vi và năng lực của phần cứng đã vượt xa khả năng xây dựng phần mềm để có thể sử dụng được các tiềm năng của nó.
- Khả năng xây dựng các phần mềm mới không giữ được cùng nhịp so với nhu cầu về phần mềm tăng lên nhanh chóng, đặc biệt khi Internet phát triển.
- Quy mô và độ phức tạp của các phần mềm mới ngày càng tăng. Khả năng bảo trì các hệ thống phần mềm cũ hiện đang tồn tại rất khó khăn và tốn kém các nguồn tài nguyên vì các thiết kế sơ sài. Phát triển các phần mềm mới phải nhanh chóng và dễ bảo trì trở thành nhu cầu cấp bách.

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

1.3. Kỹ nghệ phần mềm

1.3.1. Các khái niệm

1.3.1.1. Định nghĩa

Fritz Bauer lần đầu tiên đã đưa ra định nghĩa về kỹ nghệ phần mềm (*SE – Software Engineering*) như sau: "*Kỹ nghệ phần mềm là việc thiết lập và sử dụng đúng đắn các nguyên lý công nghệ để thu được phần mềm một*

cách kinh tế, vừa tin cậy, vừa làm việc hiệu quả trên các máy tính thực" [23]. Với nguồn lực không hạn chế, thì đa số các vấn đề của phần mềm đều có thể giải quyết được. Thách thức ở đây là, phải tạo ra phần mềm chất lượng cao với sự hạn chế về nguồn lực và tuân thủ một lịch trình định trước. Nhiều định nghĩa khác về sau đều nhấn mạnh vào các yêu cầu về một kỷ luật công nghệ trong việc phát triển. Parnas (1987) định nghĩa kỹ nghệ phần mềm là *"việc xây dựng phần mềm nhiều phiên bản bởi nhiều người"*. Ghezzi (1991) định nghĩa kỹ nghệ phần mềm là *"một lĩnh vực của khoa học máy tính, liên quan đến việc xây dựng các hệ thống phần mềm vừa lớn, vừa phức tạp bởi một hay một số nhóm kỹ sư"*. Sommerville [34] định nghĩa kỹ nghệ phần mềm là *"lĩnh vực liên quan đến lý thuyết, phương pháp và công cụ dùng cho phát triển phần mềm"*. Pressman [29] định nghĩa kỹ nghệ phần mềm là *"bộ môn tích hợp cả quy trình, các phương pháp, các công cụ để phát triển phần mềm máy tính"*.

Kỹ nghệ phần mềm là kỹ nghệ phát triển một hệ thống, gồm một loạt các bước chứa đựng ba yếu tố chủ chốt sau đây:

- *Phương pháp (method);*
- *Công cụ (tool);*
- *Thủ tục (procedure).*

Sự hình thành của kỹ nghệ phần mềm có liên quan mật thiết đến sự phát triển của khoa học máy tính (Phụ lục A về mối quan hệ giữa hai lĩnh vực này).

1.3.1.2. Các phương pháp trong kỹ nghệ phần mềm

Phương pháp cho một "cách làm" về mặt kỹ thuật để xây dựng phần mềm. Các phương pháp thường bao gồm phương pháp luận và các phương pháp cụ thể.

Phương pháp luận mô tả một cách tiếp cận mang tính kỹ thuật để tạo ra phần mềm, thường bao gồm các hoạt động theo một định hướng công nghệ và trình tự thực hiện chúng. Các định hướng công nghệ như định hướng cấu trúc, định hướng dữ liệu, định hướng đối tượng,... Các hoạt động bao gồm việc lập kế hoạch và ước lượng dự án, phân tích yêu cầu hệ thống và phần mềm, thiết kế cấu trúc dữ liệu, kiến trúc chương trình và thuật toán, mã hoá, kiểm thử và bảo trì, và tùy thuộc hướng công nghệ được lựa chọn mà các phương pháp cụ thể thích hợp được sử dụng.

Các **phương pháp cụ thể** là cách thức giải quyết các vấn đề cụ thể phát sinh khi thực hiện các nhiệm vụ khác nhau trong các giai đoạn khác nhau của quá trình phát triển phần mềm.

Bảng 1.1. Ví dụ về một phương pháp

Thành phần	Mô hình luồng dữ liệu
Các khái niệm	- <i>Tác nhân</i>: người, tổ chức, bộ phận tổ chức, hệ khác. - <i>Tiến trình</i>: dãy các hoạt động diễn ra tác động lên dữ liệu vào để tạo ra dữ liệu ra. - <i>Luồng dữ liệu</i>: các dữ liệu di chuyển từ một nơi đến nơi khác. - <i>Kho dữ liệu</i>: các dữ liệu được lưu trữ tại một vị trí.
Ký pháp	
Các quy tắc	- Một tiến trình là duy nhất trong mô hình. - Các luồng dữ liệu vào một tiến trình phải khác các luồng dữ liệu ra. - Các luồng dữ liệu vào một tiến trình phải đủ để tạo ra các luồng dữ liệu ra. - Không có luồng dữ liệu vòng (quay lại nơi nó đi ra). - Không có các luồng dữ liệu giữa: kho dữ liệu – kho dữ liệu, kho dữ liệu – tác nhân, tác nhân – tác nhân. - Tiến trình logic không có phần thứ 3.
Quy trình xây dựng	- Vẽ biểu đồ ngữ cảnh: chỉ có tác nhân, một tiến trình duy nhất mô tả cả hệ thống, các luồng dữ liệu giữa tác nhân và hệ thống (mô tả sự tương tác giữa môi trường với hệ thống). - Phát triển biểu đồ luồng dữ liệu mức 0: xuất phát từ biểu đồ ngữ cảnh và các yếu tố của mô hình nghiệp vụ: • <i>Thay</i> tiến trình duy nhất bằng các tiến trình con; • <i>Giữ nguyên</i>: các tác nhân ngoài và các luồng dữ liệu. Đặt đầu mút các luồng dữ liệu vào các tiến trình con tương ứng; • <i>Thêm</i>: các kho dữ liệu, các luồng dữ liệu giữa kho – tiến trình, tiến trình – tiến trình. - Phát triển các biểu đồ luồng dữ liệu mức i ($i \geq 1$)...
Hướng dẫn	- Dùng biểu đồ luồng dữ liệu để biểu diễn mô hình ngữ cảnh của hệ thống. - Dùng biểu đồ luồng dữ liệu mô tả tiến trình xử lý nghiệp vụ ở các mức chi tiết dần: trước hết ở dạng vật lý (mô tả thế giới thực), sau ở dạng logic để phân tích.

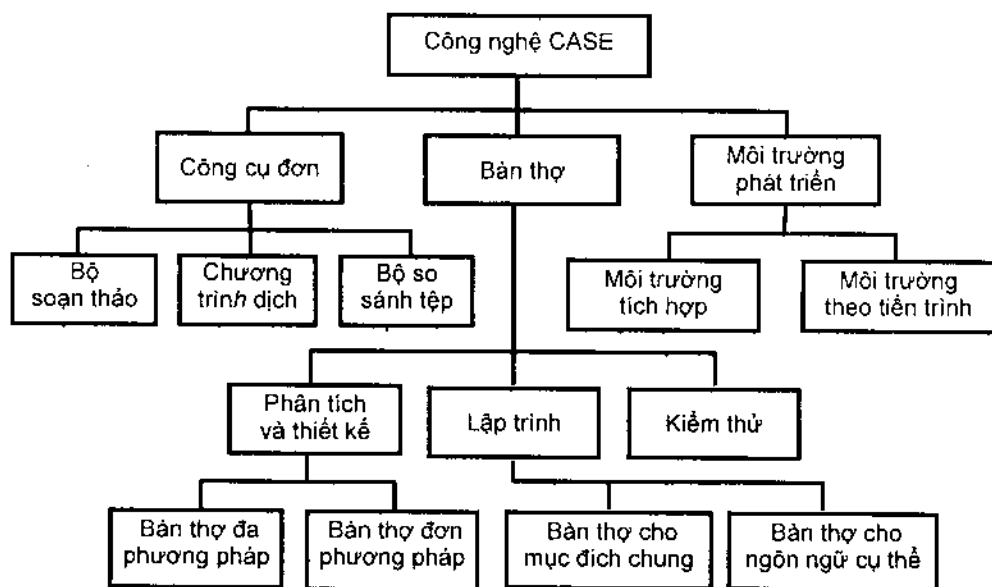
Tất cả các phương pháp đều dựa trên ý tưởng phát triển các mô hình của hệ thống, mà có thể biểu diễn ở dạng đồ họa, hay biểu diễn hình thức (toán học) và dùng các mô hình này làm đặc tả, hay thiết kế cho hệ thống được phát triển hay thẩm định. Các phương pháp thường bao gồm các thành phần sau (bảng 1.1):

- Các khái niệm mô tả các thành phần của mô hình.
- Các ký pháp (toán học, đồ họa) biểu diễn các phần tử mô hình.
- Các quy tắc liên kết các phần tử tạo thành mô hình một cách đúng đắn.
- Một quy trình hướng dẫn các bước để phát triển mô hình.
- Những gợi ý, đề xuất cho việc vận dụng phương pháp vào những trường hợp thích hợp để có được mô hình mô tả thế giới thực tốt nhất có thể.

1.3.1.3. Các công cụ trong kỹ nghệ phần mềm

Các công cụ là những phần mềm khác nhau được xây dựng trên cơ sở của những mô hình và phương pháp cụ thể, nhằm cung cấp sự trợ giúp cho việc tự động hay bán tự động hóa các hoạt động phát triển. Các công cụ thường được tích hợp để thực hiện một chức năng trọn vẹn hay một số chức năng. Khi các công cụ được tích hợp đến mức các thông tin do chúng tạo ra, có thể được dùng cho các công cụ khác hay cho các giai đoạn tiếp theo của quá trình phát triển, thì tạo nên những bộ công cụ – bàn thợ (workbenches), hay một hệ thống trợ giúp phát triển phần mềm và được gọi là *môi trường phát triển* (development environments). Tất cả chúng được gọi chung là *kỹ nghệ phần mềm được máy tính trợ giúp* (CASE – Computer Aided Software Engineering) (hình 1.4).

Có rất nhiều phương pháp được sử dụng, nhưng chúng có thể không nhất quán và thiếu sự đồng bộ. Ngày nay, ngôn ngữ *Mô hình hóa thống nhất* (UML – Unified Modeling Language [8]) ra đời đã cung cấp một ngôn ngữ chung, dùng cho tất cả các giai đoạn *phát triển phần mềm hướng đối tượng* trước khi có chương trình. Các công cụ được xây dựng trên cơ sở ngôn ngữ này (*Rational Rose, PowerDesigner, ...*) đã trợ giúp cho việc thực hiện hầu như các hoạt động phát triển. Mặc dù đã đạt nhiều thành tựu, việc tự động hóa quá trình phát triển phần mềm vẫn còn rất hạn chế và đang là những vấn đề còn mở.



Hình 1.4. Phân loại các công cụ phát triển phần mềm

1.3.1.4. Các thủ tục trong kỹ nghệ phần mềm

Các thủ tục là chất keo dán các phương pháp và công cụ lại với nhau, làm cho chúng được sử dụng hợp lý và đúng hạn. Các thủ tục thường bao gồm:

- Xác định trình tự theo đó các phương pháp sẽ được áp dụng cho mỗi dự án.
- Định ra các sản phẩm cần bàn giao (tài liệu báo cáo, bản mẫu,...), các chuẩn, các độ đo để đánh giá kết quả công việc và sản phẩm, kiểm soát quá trình phát triển, đảm bảo chất lượng phần mềm, điều tiết sự thay đổi,...
- Xác định những cột mốc mà tại đó có các sản phẩm được bàn giao để người quản lý nắm được tiến độ, phân bổ tài nguyên và kiểm soát được kết quả.

Kỹ nghệ phần mềm bao gồm nhiều giai đoạn, mỗi giai đoạn lại bao gồm cả phương pháp, công cụ và thủ tục được nêu ở trên và thường được gọi là *một tiến trình* (paradigm) hay *tiến trình của kỹ nghệ phần mềm*.

1.3.1.5. Tiến trình phần mềm

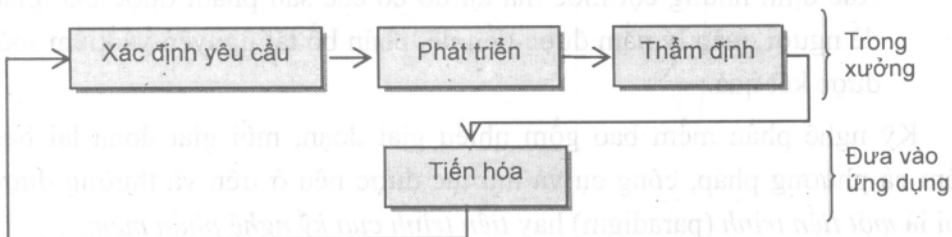
a) Tiến trình

Khi xây dựng một phần mềm một cách công nghiệp, điều quan trọng là phải vạch ra được một loạt các bước hoạt động dự kiến trước – một lộ trình, mà theo đó có thể tạo ra một phần mềm chất lượng, hiệu quả và đúng thời hạn. Một lộ trình như vậy được gọi là một *tiến trình phần mềm* (software process).

Định nghĩa trên đây cho thấy, đặc trưng trước hết của tiến trình là *các hoạt động cần thực hiện*. Trong các hoạt động đó phải có *các hoạt động mang tính kỹ thuật* để tạo ra sản phẩm. Tiếp đến, sản phẩm phải đảm bảo các chất lượng và các ràng buộc đặt ra. Vì vậy, loại hoạt động thứ hai là *các hoạt động quản lý*.

Mặt khác của tiến trình đó là *trình tự thực hiện các hoạt động*. Để tiện cho quản lý, các hoạt động thường được phân chia vào các bước. Trình tự thực hiện hoạt động trong tiến trình được thể hiện qua trình tự thực hiện các bước. Tất nhiên, các bước có thể được thực hiện tuần tự hoặc có thể lặp lại một số bước nào đó. Về mặt kỹ thuật, bốn hoạt động cơ bản là chung cho mọi tiến trình phần mềm (hình 1.5):

1. *Xác định yêu cầu phần mềm*: Xác định các chức năng của phần mềm, các ràng buộc mà nó cần tuân thủ khi phát triển và hoạt động, và đặc tả nó.
2. *Phát triển phần mềm*: Tạo ra phần mềm theo đúng đặc tả.
3. *Thẩm định phần mềm*: Phần mềm cần được kiểm tra xem có đáp ứng yêu cầu người dùng không.
4. *Tiến hóa phần mềm*: Phần mềm cần được tiến hóa để đáp ứng yêu cầu thay đổi của người dùng và môi trường.



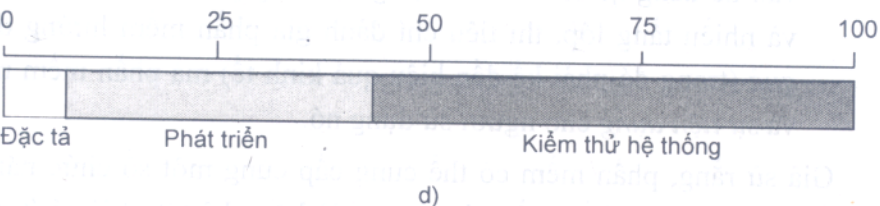
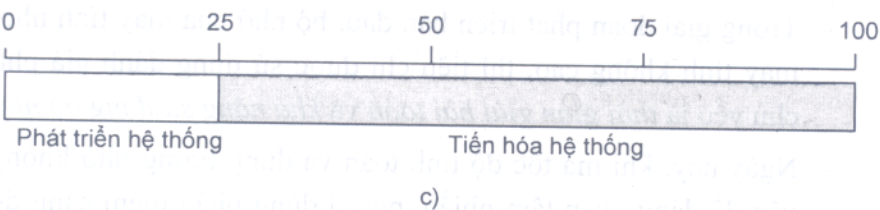
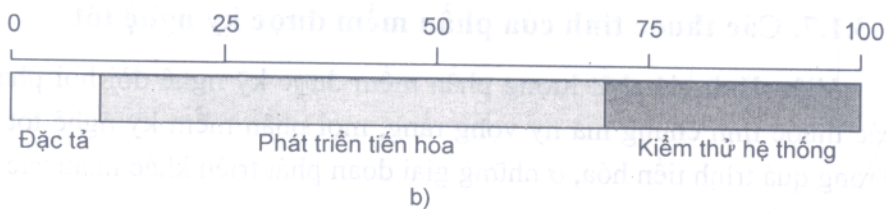
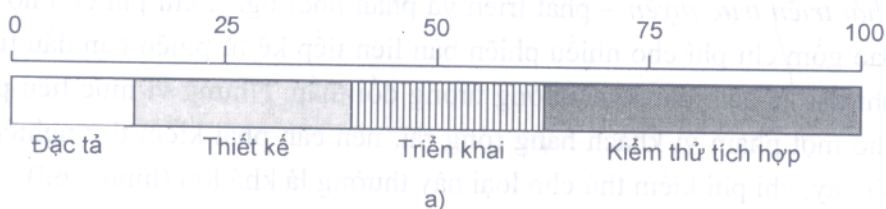
Hình 1.5. Tiến trình chung cho phát triển phần mềm

Trong các tiến trình khác nhau, các hoạt động này được tổ chức theo những cách khác nhau và được mô tả theo những mức chi tiết khác nhau. Tuy nhiên, mỗi kiểu tiến trình có thể là thích hợp với các loại ứng dụng này hơn là với loại khác.

b) Mô hình tiến trình

Mô hình tiến trình phần mềm (process model) là sự mô tả tiến trình một cách đơn giản hóa khi xem xét nó từ một cách nhìn cụ thể. Về bản chất, mô hình tiến trình là một sự trừu tượng hóa một lớp các tiến trình thực. Một vài mô hình tiến trình phần mềm được nhiều người biết đến như: *mô hình thác nước*, *mô hình xoắn ốc*, *mô hình làm bản mẫu*.

1.3.1.6. Chi phí trong kỹ nghệ phần mềm



Hình 1.6

- a) Phân bổ chi phí phát triển; b) Phân bổ chi phí phát triển tiến hóa;
c) Phân bổ chi phí tiến hóa; d) Phân bổ chi phí phát triển sản phẩm chung.

Chi phí cho một phần mềm bao gồm toàn bộ chi phí chi cho tiến trình kỹ nghệ nó. Mặc dù mong muốn, nhưng người ta khó có thể ước lượng chính xác phần chi phí cụ thể của mỗi tiến trình phần mềm theo các yếu tố tham gia. Chi phí này thường phụ thuộc vào nhiều yếu tố khách quan như: *tiến trình được sử dụng, loại phần mềm đặt hàng, chất lượng sản phẩm được yêu cầu,...*

Nếu xem tổng chi phí cho việc phát triển phần mềm phức tạp là 100 đơn vị, thì sự phân bổ chi phí cho các *hoạt động phát triển* được chỉ ra trên hình 1.6a, còn chi phí cho phát triển theo cách *tiếp cận tiến hóa* được cho ở hình 1.6b. Đối với những hệ thống có vòng đời dài, chi phí bảo trì thường vượt xa chi phí phát triển (hình 1.6c). Các phần mềm chung thường được *phát triển trực tuyến* – phát triển và phân phối ngay, chi phí cho nó thường bao gồm chi phí cho nhiều phiên bản liên tiếp kể từ phiên bản đầu tiên. Chi phí đặc tả của loại này thường tương đối thấp. Nhưng vì mục tiêu phục vụ cho một phạm vi khách hàng rộng rãi, nên cần phải kiểm thử nghiêm ngặt. Vì vậy, chi phí kiểm thử cho loại này thường là khá lớn (hình 1.6d).

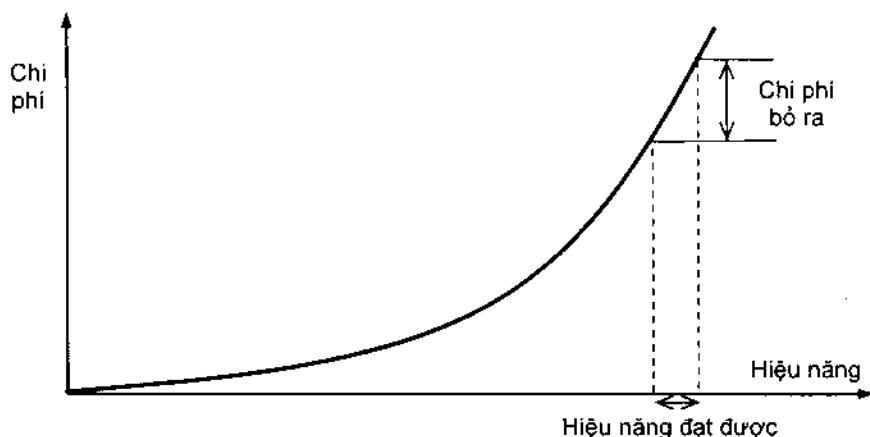
1.3.1.7. Các thuộc tính của phần mềm được kỹ nghệ tốt

Việc đánh giá chất lượng phần mềm được kỹ nghệ đòi hỏi phải chỉ ra các thuộc tính chung mà hy vọng rằng, mọi phần mềm kỹ nghệ tốt đều có. Trong quá trình tiến hóa, ở những giai đoạn phát triển khác nhau các tiêu chí là khác nhau:

- Trong giai đoạn phát triển ban đầu, bộ nhớ của máy tính nhỏ, tốc độ máy tính không cao, thì tiêu chí được sử dụng đánh giá phần mềm chủ yếu là *thời gian giải bài toán và khả năng sử dụng bộ nhớ*.
- Ngày nay, khi mà tốc độ tính toán và dung lượng nhớ không còn là vấn đề đáng quan tâm nhiều, người dùng phần mềm càng đông đảo và nhiều tầng lớp, thì tiêu chí đánh giá phần mềm hướng đến *hiệu quả* (trong đó phải kể đến hiệu quả kinh tế) mà phần mềm mang lại và *sự tiện dụng* cho người sử dụng nó.

Giả sử rằng, phần mềm có thể cung cấp cùng một số chức năng được đòi hỏi, khi đó một phần mềm được xem là kỹ nghệ tốt phải có ít nhất bốn thuộc tính sau (Phụ lục B: Chuẩn ISO 9126 về chất lượng phần mềm):

- *Có thể bảo trì được*: Phần mềm thường có tuổi thọ dài, phải được xây dựng và lập tư liệu, sao cho có thể dễ sửa đổi, bổ sung mà không quá tốn kém.
- *Đáng tin cậy*: Phần mềm phải thực hiện được điều mà người tiêu dùng mong mỏi và không thất bại nhiều hơn những gì đã được đặc tả.
- *Có hiệu quả*: Hệ thống phải không lãng phí nguồn lực như bộ nhớ, bộ xử lý. Thực tế chỉ ra rằng, không thể tối ưu hóa độ hiệu quả, vì rằng, việc đó có thể làm cho phần mềm rất khó thay đổi và chi phí cao (hình 1.7).
- *Khả dụng*: Hệ thống phải có giao diện người sử dụng thích hợp, giao diện người sử dụng phải phù hợp với khả năng và kiến thức của người dùng hệ thống, hỗ trợ được những người dùng có trình độ khác nhau (dễ học, dễ sử dụng và nhớ lâu).



Hình 1.7. Chi phí phụ thuộc vào mức hiệu năng đạt được

Giá cả phải được xem là một yếu tố xác định một phần mềm được kỹ nghệ tốt. Tuy nhiên, *có thể bảo trì được* coi là thuộc tính chủ chốt vì rằng các chi phí cho sản phẩm phần mềm chủ yếu là trong giai đoạn phần mềm được đưa vào sử dụng.

1.3.2. Một số loại mô hình tiến trình phần mềm tiêu biểu

Vào năm 1996, lần đầu tiên Huff hệ thống hóa một số mô hình tiến trình phần mềm. Sommerville [34] nhắc tới một số loại tiêu biểu là:

- *Mô hình thác nước* (waterfall model).

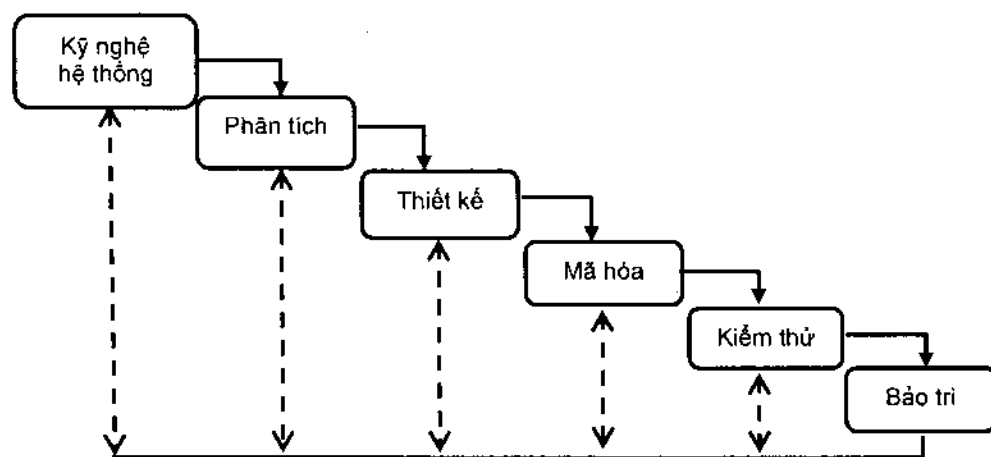
- *Mô hình phát triển tiến hóa (evolutionary model)* là mô hình phát triển với quá trình lặp để xây dựng dần phần mềm.
- *Phát triển hệ thống hình thức (formal system development)*: Một cách tiếp cận dựa trên đặc tả hệ thống bằng toán học để chứng minh, hay chuyển đổi nó thành chương trình nhờ các công cụ toán học chuyên dụng.
- *Phát triển phần mềm theo hướng sử dụng lại (reuse oriented software development)*: Quá trình phát triển tập trung vào việc tích hợp các thành phần đã có để nhận được hệ thống, đáp ứng được các yêu cầu đặt ra.

1.3.3. Mô hình thác nước

Mô hình thác nước hay còn gọi là mô hình vòng đời truyền thống do Royce đề xuất năm 1970. Nó yêu cầu tiếp cận một cách hệ thống, tuần tự và chặt chẽ đối với việc phát triển phần mềm, bắt đầu ở mức hệ thống và tiến dần xuống phân tích, thiết kế, mã hóa, kiểm thử và bảo trì (hình 1.8). Mô hình gồm các hoạt động sau:

a) Kỹ nghệ hệ thống

Kỹ nghệ hệ thống bao gồm việc thu thập yêu cầu ở mức hệ thống với một lượng nhỏ thiết kế và phân tích ở mức đỉnh. Phần mềm chỉ là một phần của hệ thống lớn hơn, nên công việc phải bắt đầu từ việc thiết lập yêu cầu cho toàn hệ thống, rồi trích ra một tập con yêu cầu cho phần mềm.



Hình 1.8. Mô hình thác nước của tiến trình phần mềm

b) Phân tích (analysis)

Trước hết việc phân tích yêu cầu được tập trung vào việc thu thập và phân tích các thông tin cần cho phần mềm, các chức năng phần mềm cần phải thực hiện, hiệu năng cần có của mỗi chức năng, các giao diện cho người sử dụng, các ràng buộc mà phần mềm cần tuân thủ khi nó được phát triển và hoạt động. Sau đó, lập tư liệu về yêu cầu cho hệ thống và phần mềm để khách hàng duyệt lại.

c) Thiết kế (design)

Thiết kế là quá trình chuyển hóa các yêu cầu phần mềm thành các mô tả thiết kế, mà từ đó các nhà phát triển và lập trình có thể lắp đặt hệ thống và chuyển thiết kế thành chương trình vận hành được. Thiết kế chia làm hai giai đoạn là *thiết kế logic* và *thiết kế vật lý*. Thiết kế logic nhằm tạo ra hệ thống có kiến trúc tốt và có các giải pháp hợp lý cho mỗi vấn đề. Thiết kế vật lý là áp các điều kiện thực tế có được vào thiết kế logic để được một mô tả hệ thống vật lý như nó tồn tại trên thực tế. Thiết kế gồm nhiều bước, tập trung vào 4 bước chính sau: *thiết kế kiến trúc, thiết kế dữ liệu, thiết kế các thủ tục và thiết kế các giao diện*. Giống như phân tích yêu cầu, việc thiết kế cần lập tư liệu đầy đủ để phê duyệt.

d) Mã hóa⁽¹⁾ (coding)

Mã hóa là dịch các đặc tả thiết kế thành các chương trình mã nguồn trong một ngôn ngữ lập trình nào đó mà máy có thể thực hiện được. Nếu thiết kế được đặc tả một cách chi tiết với mức hình thức hóa cao, thì công việc mã hóa có thể thực hiện một cách "máy móc" (bằng thủ công hay tự động).

e) Kiểm thử (testing)

Khi đã có chương trình mã nguồn, việc kiểm thử bắt đầu. Tiến trình kiểm thử tập trung vào phát hiện và sửa lỗi phần logic trong chương trình. Về chức năng bên ngoài, việc kiểm thử đảm bảo phát hiện ra lỗi và kiểm tra xem với cái vào xác định có tạo ra kết quả mong muốn hay không. Việc kiểm thử được tiến hành từ mức các đơn vị riêng rẽ, sau tích hợp chúng ở mức từng chương trình, từng hệ con và toàn hệ thống. Các kiểm thử cuối

⁽¹⁾ Cần phân biệt với các khái niệm *mã hóa thông tin* và *mật mã hóa*.

cùng liên quan đến việc thẩm định phần mềm xem yêu cầu của người dùng có được thỏa mãn không.

f) Bảo trì (maintenance)

Phần mềm cần phải hoàn thiện trước khi nó được giao cho khách hàng. Đó là việc *kiểm tra và sửa lỗi* khi đưa chương trình vào sử dụng, việc *thích ứng nó với thay đổi* của môi trường bên ngoài hoặc *bổ sung chức năng* hay *nâng cao hiệu năng* cần có khi khách hàng yêu cầu. Việc bảo trì có thể áp dụng lại các bước của vòng đời phát triển cho hệ chương trình hiện có khi có nhu cầu tiếp tục phát triển.

Vòng đời cổ điển là mô hình có sớm nhất và được sử dụng rộng rãi nhất. Nó đưa ra các phương pháp khoa học, các bước tổng quát áp dụng được cho nhiều mô hình. Nó tốt hơn nhiều lần cách tiếp cận tự nhiên. Tuy nhiên, các dự án phần mềm thực tế hiếm khi tuân theo trình tự tuần tự. Quá trình lặp lại là không thể tránh khỏi. Khi áp dụng mô hình này, nếu không quay lại thì dễ gặp bất trắc, mà lặp lại thì khó quản lý được tiến độ và không đáp ứng kịp thời yêu cầu của khách hàng. Mặt khác, khách hàng phải kiên nhẫn đến cuối dự án mới có chương trình làm việc được. Thời gian thực hiện dự án theo mô hình này thường dài, các tài liệu đặc tả rất lớn. Nếu chương trình gặp lỗi khi kết thúc mới phát hiện ra thì có thể sẽ là một thảm họa.

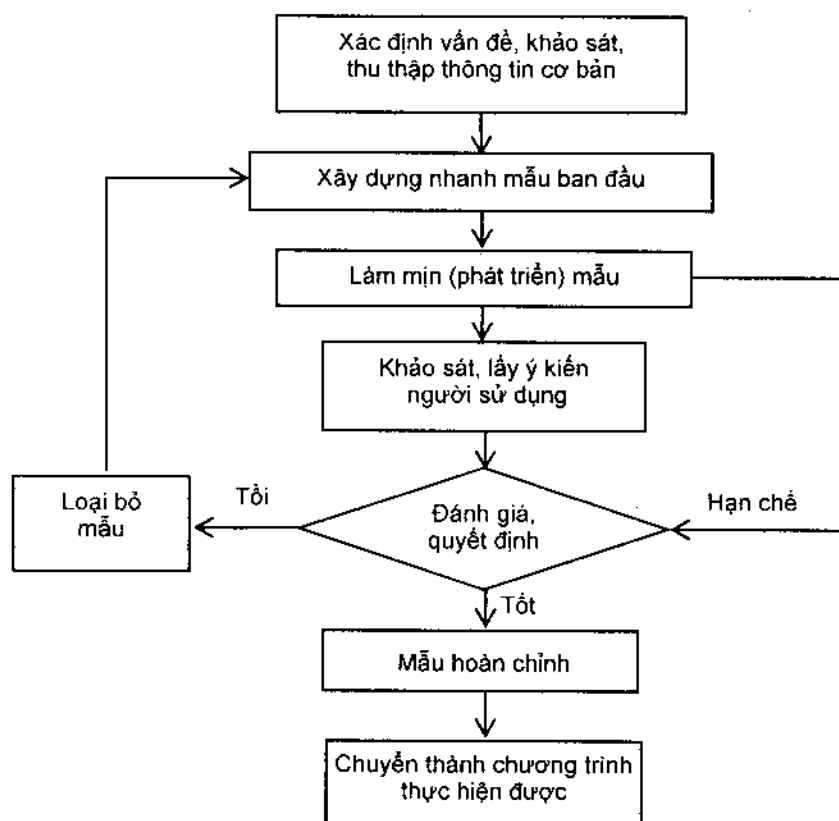
1.3.4. Các mô hình tiến hóa

1.3.4.1. Mô hình làm bản mẫu

Làm bản mẫu (prototyping) là tạo ra một mô hình như thực tế cho phần mềm cần xây dựng. Mô hình có thể có ba dạng:

- *Bản mẫu trên giấy* hay trên PC mô tả giao diện người – máy làm cho người dùng hiểu được cách các tương tác xuất hiện. Nó là một bản mô tả yêu cầu.
- *Bản mẫu cài đặt* chỉ chứa một tập con chức năng của phần mềm mong đợi.
- *Bản mẫu là một chương trình có thể thực hiện một phần*, hay tất cả chức năng mong muốn, nhưng sơ lược, cần cải tiến thêm các tính năng khác.

Trước hết, người phát triển và khách hàng gặp nhau và xác định mục tiêu tổng thể cho phần mềm, xác định các yêu cầu đã biết, các miền cần khảo sát thêm, rồi thiết kế nhanh bản mẫu ban đầu. Thiết kế nhanh tập trung vào việc biểu diễn các khía cạnh của phần mềm thấy được đối với người dùng (cách đưa vào và định dạng đưa ra), sau đó làm mịn mẫu, trình diễn để người dùng đánh giá, rồi tiếp tục làm mịn các yêu cầu cho phần mềm. Tiến trình này lặp đi lặp lại cho đến khi bản mẫu thỏa mãn yêu cầu của khách, đồng thời giúp người phát triển hiểu kỹ hơn nhu cầu nào cần phải thực hiện (hình 1.9).



Hình 1.9. Mô hình làm bản mẫu

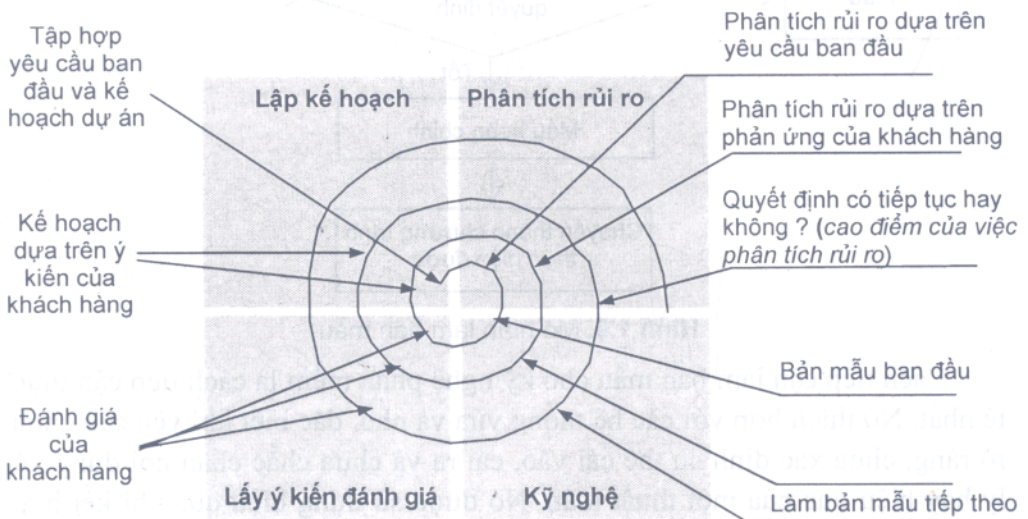
Cách tiếp cận làm bản mẫu cho kỹ nghệ phần mềm là cách tiếp cận thực tế nhất. Nó thích hợp với các hệ thống vừa và nhỏ, đặc biệt khi yêu cầu chưa rõ ràng, chưa xác định cụ thể cái vào, cái ra và chưa chắc chắn nội dung xử lý hay hiệu quả của một thuật toán. Nó được sử dụng hiệu quả khi kết hợp với các mô hình khác, chẳng hạn, ở giai đoạn xác định yêu cầu của mô hình thác nước,...

1.3.4.2. Mô hình xoắn ốc

Mô hình xoắn ốc (spiral model) được Boehm đưa ra năm 1988 [5]. Nó dựa trên ý tưởng là tối thiểu hóa rủi ro, bằng việc phân tích yếu tố rủi ro ở mỗi bước lặp và sử dụng phương pháp làm bản mẫu. Quá trình phát triển được chia thành nhiều bước lặp lại, mỗi bước bắt đầu bằng việc lập kế hoạch, phân tích rủi ro, rồi tạo bản mẫu, hoàn thiện và phát triển hệ thống, duyệt lại, và cứ thế tiếp tục (hình 1.10). Nội dung một bước gồm bốn hoạt động chính:

- *Lập kế hoạch*: Xác định mục tiêu, giải pháp và ràng buộc;
- *Phân tích rủi ro*: Phân tích các phương án, xác định và giải quyết rủi ro;
- *Kỹ nghệ*: Phát triển sản phẩm "mức tiếp theo".
- *Đánh giá của khách hàng*: Khẳng định kết quả của kỹ nghệ.

Với mỗi lần lặp xoắn ốc (bắt đầu từ tâm), các phiên bản được hoàn thiện dần. Nếu phân tích rủi ro chỉ ra rằng, yêu cầu không chắc chắn thì bản mẫu có thể được sử dụng trong góc phần tư kỹ nghệ; các mô hình và các mô phỏng khác cũng được dùng để làm rõ hơn vấn đề và làm mịn yêu cầu. Tại một vòng xoắn ốc, phân tích rủi ro phải đi đến quyết định "tiến hành tiếp hay dừng". Nếu rủi ro quá lớn, thì có thể đình chỉ dự án hay thay đổi yêu cầu đặt ra cho thích hợp.



Hình 1.10. Mô hình xoắn ốc

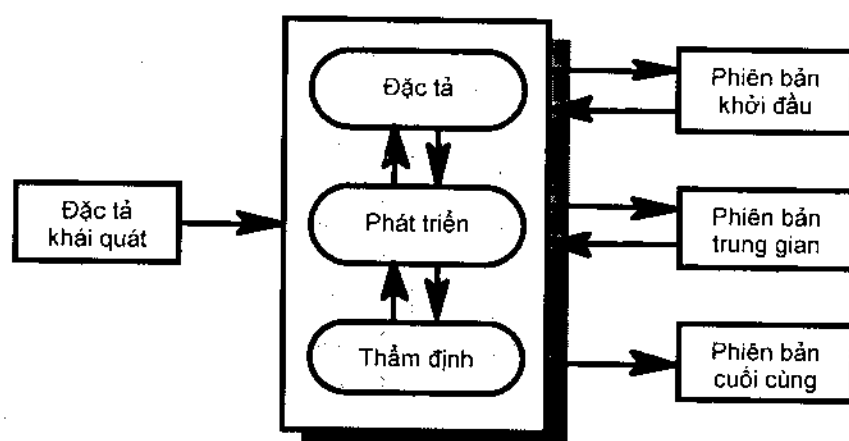
Mô hình này thích hợp để phát triển các hệ thống quy mô lớn. Trong mô hình này không có sự phân biệt rõ ràng giữa hoạt động bảo trì và phát triển. Tuy nhiên, việc thay đổi một cách linh hoạt khi phát triển đòi hỏi nhà phát triển và khách hàng phải có liên kết chặt chẽ, nếu không sẽ không làm được gì khi hợp đồng đã ký kết. Mặt khác, quy mô dự án phải đủ lớn để chịu được các chi phí trả cho chuyên gia phân tích rủi ro. Boehm cải biên mô hình này với tên gọi là "*WINWIN Spiral Model*" [6], dùng cho việc phát triển phần mềm hướng đối tượng.

1.3.4.3. Mô hình RUP

Mô hình RUP (Rational Unified Process) là mô hình phát triển phần mềm dành riêng cho hướng đối tượng. Nó có ba đạo trung là: *lấy kiến trúc làm trung tâm, điều khiển bởi các ca sử dụng, lặp và tăng dần* [37]. Nó có nhiều điểm tương đồng với mô hình xoắn ốc được Boehm [6] làm thích ứng cho việc phát triển phần mềm định hướng đối tượng. Tuy nhiên, ở mỗi bước lặp của RUP, nội dung các hoạt động có những đặc trưng riêng gắn với ngôn ngữ mô hình hóa thống nhất – UML [37].

1.3.4.4. Các mô hình phát triển tiến hóa khác

Ý tưởng chung của phương pháp tiến hóa là phát triển một phiên bản sơ khai ban đầu và giới thiệu để người dùng bình luận, rồi làm mịn nó thông qua các bước cho đến khi nhận được một hệ thống thích hợp.



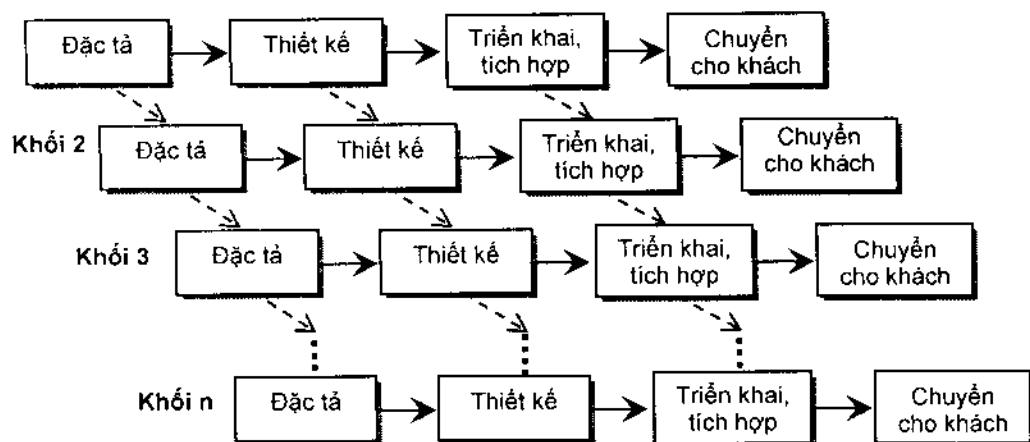
Hình 1.11. Biểu đồ mô hình phát triển tiến hóa

Phương pháp này thích hợp khi mà khó thực hiện các đặc tả chi tiết ngay từ đầu. Phương pháp thiếu đặc tả thường hay được dùng để phát triển các hệ thống trí tuệ nhân tạo, nhằm kích thích những khả năng tư duy của con người. Chìa khóa thành công trong cách tiếp cận này là cho phép thực hiện các bước lập hệ thống một cách nhanh chóng, những sự thay đổi như những gợi ý được phối hợp với việc trình diễn nhanh đến mức có thể. Nó yêu cầu những ngôn ngữ lập trình rất cao để phát triển các phần mềm như LISP, Prolog. Biểu đồ tiến trình tổng quát của nó được mô tả như hình 1.11. Với phương pháp này, không có đặc tả đầy đủ, nên không thể kiểm tra và không thể nói đến một chương trình đúng.

a) Mô hình phát triển tăng dần

Mô hình phát triển tăng dần (incremental model): Phát triển một hệ thống càng nhanh càng tốt, rồi cải biên hệ thống đó cho tới khi nó thực hiện được mọi yêu cầu đặt ra. Nó là một biến thể của mô hình tiến hóa. Nó có ý tưởng cơ bản gần giống với mô hình làm bản mẫu và mô hình xoắn ốc, nhưng được thực hiện trên từng khối tương đối độc lập (hình 1.12).

Khối 1



Hình 1.12. Mô hình phát triển tăng dần

b) Phát triển ứng dụng nhanh RAD

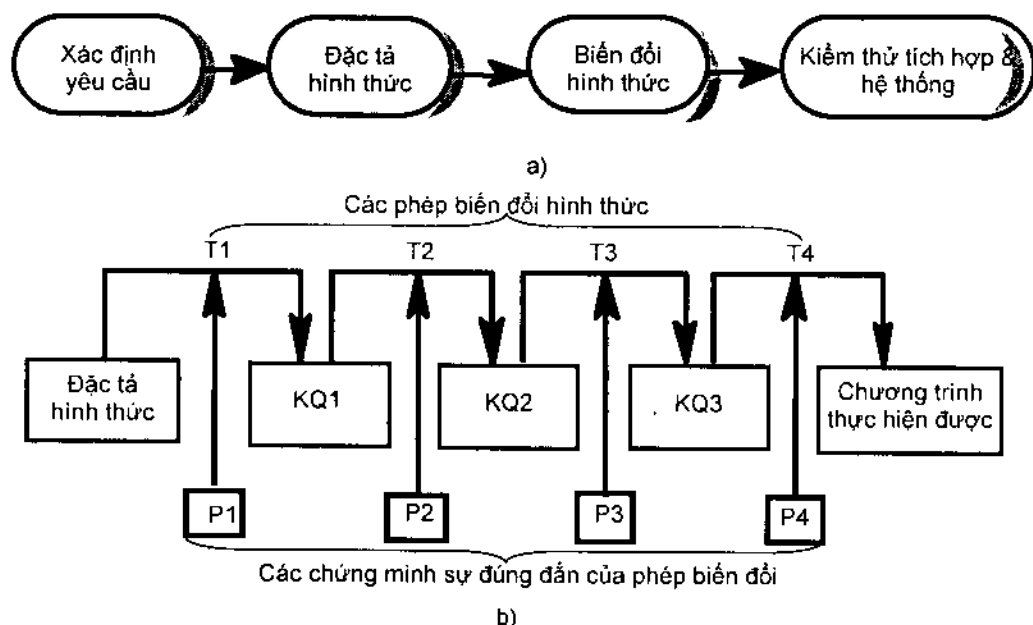
RAD (Rapid Application Development) là phương pháp luận gộp các hoạt động phân tích, thiết kế, xây dựng vào một loạt vòng lặp phát triển ngắn, bước lặp ngắn đảm bảo tính hiệu quả và tự sửa chữa. Kỹ thuật RAD hướng đến nhu cầu đưa người dùng tham gia vào việc phân tích và thiết kế hệ thống bằng cách sử dụng CASE. Hệ thống phát triển bằng RAD đáp ứng

nhu cầu hiệu quả và chi phí bảo trì thấp. Chất lượng hệ thống được xác định ở đây là mức độ hệ thống đáp ứng các yêu cầu nghiệp vụ về thời gian.

RAD thích hợp cho đội phát triển nhỏ, người dùng và nguồn lực của hệ thống thông tin không lớn, kết hợp với vòng lặp ngắn, tối ưu tốc độ, thống nhất cái nhìn và mục tiêu giao tiếp không hình thức, hiệu quả và quản lý dự án đơn giản, mỗi bước xuất ra một phiên bản chức năng của hệ thống cuối cùng. Nó được kỹ nghệ riêng, một phần làm việc đầy đủ, không giống như làm mẫu.

1.3.5. Phát triển hệ thống hình thức hóa

Quá trình phát triển này được mô tả trên hình 1.13. Tư tưởng chính của *phương pháp hình thức* (formal methods) [42] là biểu diễn các đặc tả yêu cầu ở dạng biểu diễn các ký pháp toán học. Sau đó áp dụng các biến đổi khác nhau để chuyển đặc tả hình thức thành chương trình. Trong quá trình chuyển đổi, biểu diễn hình thức của đặc tả ngày càng được chi tiết dần, nhưng luôn đảm bảo tính đúng đắn toán học của biểu diễn hệ thống, mà không cần đòi hỏi một sự kiểm tra tốn kém. Nhờ vậy mà nó đảm bảo rằng, chương trình nhận được là một triển khai đúng của đặc tả.

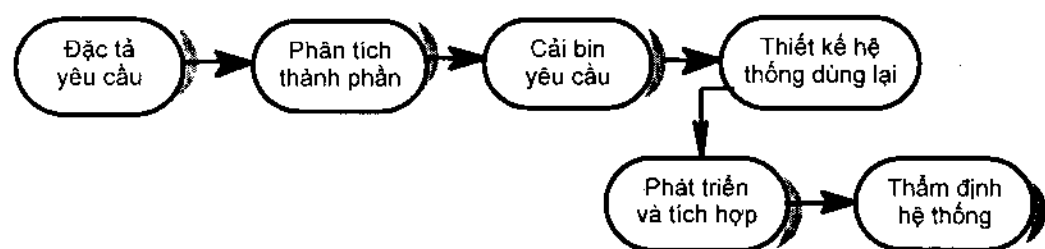


Hình 1.13. a) Các bước của tiến trình phát triển hình thức;
b) Các bước của tiến trình phát triển biến đổi hình thức

Ưu điểm của cách tiếp cận này là có thể áp dụng để chứng minh tính đúng đắn của đặc tả, chứng minh được chương trình là đáp ứng được yêu cầu của đặc tả đã cho. Với phương pháp này, chi phí cho đặc tả cao, nhưng các chi phí sau đó là nhỏ hơn nhiều so với các phương pháp khác. Quá trình chuyển đổi tạo thành một dãy các bước nhỏ, dễ theo dõi. Tuy nhiên, việc đặc tả đòi hỏi một trình độ trừu tượng cao; việc chọn các chuyển đổi nào để áp dụng lại, yêu cầu kỹ năng rất chuyên nghiệp; việc chứng minh sự đúng đắn của chuyển đổi là rất khó khăn. Vì vậy, với đa số các kỹ sư phần mềm, phương pháp này là khó áp dụng.

1.3.6. Phát triển phần mềm theo hướng sử dụng lại

Phát triển theo hướng *sử dụng lại phần mềm* (Software Reuse) [42] dựa trên cơ sở có rất nhiều các thành phần phần mềm có thể sử dụng và tích hợp chúng trong một khung làm việc để nhận được một hệ thống làm việc. Sơ đồ tổng quát của quá trình được mô tả như ở hình 1.14. Trước hết, giống như các tiến trình khác, cần đặc tả yêu cầu, sau đó phân tích các yêu cầu thành những thành phần. Tiến hành cải biên các thành phần này để chúng tương ứng với các thành phần sẵn có. Sau đó, thiết kế hệ thống dựa trên các thành phần đã có. Trong đó có thể sử dụng các mẫu thiết kế [13]. Công việc tiếp theo là phát triển và tích hợp hệ thống. Trong trường hợp chung nhất, khi còn có những thành phần chưa có sẵn, khi đó cần phải phát triển bổ sung. Trong nhiều trường hợp, khi có sẵn các tài sản những phần mềm, khi đó ta có phương pháp *phát triển dựa trên thành phần* (CBSE – Component – Based Software Engineering) [10].

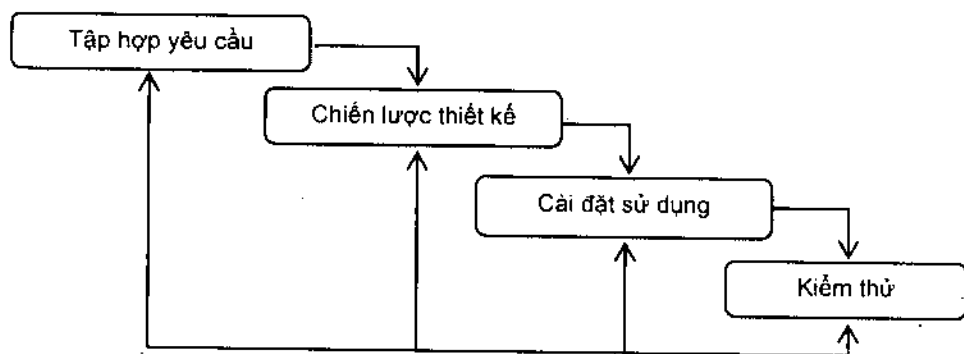


Hình 1.14. Tiến trình phát triển hướng sử dụng lại

Phương pháp này có ưu điểm vượt trội là rút ngắn được thời gian phát triển (70% vòng đời), giảm chi phí phát triển (đến 84%) [42] và đặc biệt là ít gặp rủi ro. Hơn thế nữa, nó còn cho hệ thống có độ tin cậy cao. Ngày nay, công nghệ hướng đối tượng phát triển phần mềm tiến triển rất nhanh, phát triển phần mềm theo hướng sử dụng lại được sự trợ giúp rất lớn từ công nghệ này. Đặc biệt là việc dùng lại *các khung làm việc* (Framework) và *các mẫu thiết kế* (patterns) [18].

1.3.7. Kỹ thuật thể hệ thứ tư

Thuật ngữ "*kỹ thuật thể hệ thứ tư*" (4GT) bao gồm một phạm vi rộng các công cụ phần mềm có điểm chung là: Mỗi công cụ đều cho phép xác định một số đặc trưng của phần mềm ở mức cao. Tiếp đó, công cụ tự động sinh ra mã chương trình gốc theo nhu cầu của người phát triển. Tiến trình 4GT tập trung vào khả năng xác định phần mềm ở mức độ gần với ngôn ngữ tự nhiên, hay dùng một số ký pháp tương ứng với chức năng. Tiến trình 4GT cho kỹ nghệ phần mềm được thể hiện trên sơ đồ hình 1.15.



Hình 1.15. Mô hình tiến trình sử dụng ngôn ngữ thể hệ thứ tư

Môi trường phát triển phần mềm hỗ trợ cho tiến trình 4GT bao gồm một số hoặc tất cả các công cụ sau:

- Ngôn ngữ phi thủ tục để truy vấn cơ sở dữ liệu;
- Bộ sinh báo cáo;
- Bộ thao tác dữ liệu;
- Bộ tương tác và xác định màn hình;
- Bộ sinh chương trình/mã;
- Khả năng làm trang tính.

1.3.8. Tổ hợp các tiến trình

Các tiến trình trên đều có thể dùng làm nền tảng để tích hợp lên một tiến trình khác. Trong mọi trường hợp, kỹ nghệ phần mềm bắt đầu bằng việc: *xác định mục tiêu, phương án, ràng buộc* (thu thập yêu cầu sơ bộ). Từ điểm này, bất kỳ một con đường nào rẽ nhánh đều có thể chọn làm một tiến trình. Nếu yêu cầu còn chưa được chắc chắn, thì có thể sử dụng làm bản mẫu để xác định yêu cầu đầy đủ hơn.

1.3.9. Các hoạt động của tiến trình phát triển và các tài liệu

Bảng 1.2 cho biết các hoạt động chính và các tài liệu tương ứng đưa ra trong quá trình phát triển, được xét duyệt, kiểm soát và lưu trữ trong cấu trúc phần mềm.

Bảng 1.2. Các hoạt động phát triển và các tài liệu của nó

Các hoạt động	Tài liệu đưa ra
1. Phân tích yêu cầu	- Nghiên cứu khả thi - Yêu cầu khái quát
2. Xác định yêu cầu	Tài liệu yêu cầu
3. Đặc tả hệ thống	- Đặc tả chức năng - Kế hoạch kiểm thử chấp nhận - Giới thiệu tóm tắt cách sử dụng
4. Thiết kế kiến trúc	- Đặc tả kiến trúc - Kế hoạch kiểm thử hệ thống
5. Đặc tả phần mềm	- Cấu trúc các hệ con và giao diện - Mô tả các hệ con phần mềm
6. Thiết kế dữ liệu	- Cấu trúc dữ liệu logic/vật lý - Từ điển dữ liệu
7. Thiết kế giao diện	- Đặc tả giao diện - Kế hoạch kiểm thử tích hợp
8. Thiết kế chi tiết	- Đặc tả thiết kế - Kế hoạch kiểm thử đơn vị
9. Mã hóa	Chương trình nguồn
10. Kiểm thử đơn vị	Báo cáo kiểm thử đơn vị
11. Kiểm thử môđun	Báo cáo kiểm thử môđun
12. Kiểm thử tích hợp	- Báo cáo kiểm thử tích hợp - Hướng dẫn sử dụng cuối cùng
13. Kiểm thử hệ thống	Báo cáo kiểm thử hệ thống
14. Kiểm thử chấp nhận	Hệ thống chương trình và các tài liệu

1.4. Chất lượng phần mềm

Trả lời câu hỏi thế nào là một phần mềm có chất lượng là rất khó. Ngay đối với một phần mềm đang hoạt động, những người khác nhau có thể đánh giá về nó rất khác nhau. Người ta thường nói đến đảm bảo chất lượng tổng thể của phần mềm, tức là những giải pháp để đảm bảo đạt được những độ đo khác nhau của phần mềm trong quá trình phát triển nó. Người ta hy vọng rằng, trong quá trình phát triển, khi các độ đo đã được đảm bảo, thì phần mềm sẽ chắc chắn là có chất lượng. Chẳng hạn, khi nói về độ tin cậy của hệ thống, người ta thường quan tâm đến *độ đo về tính sẵn sàng*. Tuy nhiên, khi đánh giá về phần mềm, người ta thường đưa ra một số tiêu chí để nói đến chất lượng tổng thể của nó (Phụ lục B). Những tiêu chí đó bao gồm:

- *Đạt được các mục tiêu thiết kế đề ra* của tổ chức (thực hiện được các chức năng thiết kế cho tổ chức).
- *Chi phí vận hành là chấp nhận được*: Chi phí không quá cao so với lợi ích mà nó mang lại.
- *Đáp ứng được các chuẩn mực* của một hệ thống thông tin hiện hành. Chẳng hạn, tính sẵn sàng: thời gian làm việc trong ngày, tuần; thời gian thực hiện một dịch vụ, một tìm kiếm, kết quả đưa ra đúng chuẩn (mẫu bảng biểu, số chỉ tiêu,...).
- *Sản phẩm tạo ra có giá trị xác đáng*: Thông tin đưa ra là đúng đắn, kịp thời, có ý nghĩa thiết thực đối với hoạt động chức năng và quản lý, góp phần nâng cao chất lượng sản phẩm và dịch vụ của tổ chức.
- *Bảo trì được*: Dễ bảo trì, bảo trì không quá tốn kém.
- *Khả dụng*: Dễ học và dễ sử dụng.
- *Mềm dẻo – có khả năng làm thích nghi được*: Có thể kiểm tra, mở rộng ứng dụng và phát triển tiếp được.
- *Có tính khả chuyển*: Có thể chuyển đổi từ môi trường làm việc này sang môi trường làm việc khác.

1.5. Kỹ nghệ hệ thống dựa trên máy tính

1.5.1. Hệ thống dựa trên máy tính

1.5.1.1. Khái niệm

Người kỹ sư hệ thống không chỉ quan tâm đến phần mềm cần phát triển mà cả các phần mềm khác, phần cứng và sự tương tác của toàn hệ thống với người dùng, với môi trường của nó. Người kỹ sư hệ thống cần có sự hiểu biết về kỹ nghệ một hệ thống dựa trên máy tính, vì những vấn đề của kỹ nghệ phần mềm (là một phần của hệ thống) thường là hậu quả tất yếu từ các quyết định kỹ nghệ ở mức toàn hệ thống.

Kỹ nghệ hệ thống là hoạt động đặc tả, thiết kế, thực thi, thẩm định, triển khai và bảo trì toàn bộ hệ thống. Có rất nhiều định nghĩa khác nhau về hệ thống từ trừu tượng đến chi tiết. Sau đây là một định nghĩa tốt: Một hệ thống dựa trên máy tính gồm có các thiết bị tin học (phần cứng), các chương trình (phần mềm), các dữ liệu, con người và các thủ tục làm việc được tổ chức lại để tạo ra các chức năng hay dịch vụ phục vụ cho một tổ chức [37].

Sự hoạt động thành công của mỗi thành phần trong một hệ thống phụ thuộc vào sự hoạt động của một số thành phần khác. Hệ thống luôn mang tính phân cấp, theo nghĩa là chúng luôn bao hàm trong nó những hệ thống nhỏ hơn, gọi là các hệ con – các thành phần của chúng. Khi các thành phần liên kết lại cấu thành một hệ thống, chúng tạo ra những *đặc tính nổi trội (tính trội)* mà mỗi thành phần riêng rẽ không thể có được.

1.5.1.2. Các thuộc tính nổi trội của hệ thống dựa trên máy tính

Một hệ thống dựa trên máy tính có hai loại thuộc tính nổi trội:

1. *Các thuộc tính chức năng* xuất hiện khi tất cả các thành phần của một hệ thống cùng làm việc với nhau để cung cấp một số chức năng hay dịch vụ.
2. *Các thuộc tính phi chức năng* như tính đáng tin cậy, hiệu suất, độ an toàn và bảo mật,... Chúng liên quan đến hành vi của hệ thống trong môi trường hoạt động. Chúng thường mang tính quyết định, vì rằng, sự thất bại ở một mức tối thiểu nào đó của những thuộc tính này cũng có thể làm cho toàn hệ thống không thể sử dụng được.

Vì dụ, *tính đáng tin cậy* là một khái niệm phức tạp, cần được quan tâm ở mức hệ thống hơn là ở mức từng thành phần riêng rẽ. Những nhà thiết kế hệ thống thường không thể biết trước các hậu quả của sự thất bại lan truyền trong hệ thống như thế nào, và do đó không thể đưa ra các dự đoán về tính tin cậy của dữ liệu, của các thành phần khác nhau trong hệ thống. Độ tin cậy của một hệ thống dựa trên máy tính phụ thuộc vào:

1. Độ tin cậy của phần cứng;
2. Độ tin cậy của phần mềm;
3. Độ tin cậy của người điều khiển.

Tất cả những yếu tố này liên quan chặt chẽ với nhau. Vì vậy, đối với một hệ thống được tích hợp từ nhiều thành phần, hay nhiều hệ thống khác nhau, thì càng khó để đưa ra những đánh giá chính xác về độ tin cậy của nó. Các thuộc tính khác như hiệu suất, hoặc tính khả dụng cũng khó đánh giá, nhưng lại có thể đo được sau khi hệ thống đi vào hoạt động.

1.5.1.3. Các hệ thống và môi trường của chúng

Các hệ thống không phải là các thực thể độc lập, mà chúng tồn tại trong một môi trường. Môi trường ảnh hưởng đến chức năng và hiệu suất của hệ thống. Có hai lý do chính để người kỹ sư hệ thống cần nắm và hiểu môi trường của một hệ thống:

1. Trong rất nhiều trường hợp, hệ thống được dự kiến để tạo ra một số thay đổi lên môi trường của nó.
2. Chức năng của một hệ thống có thể bị ảnh hưởng bởi các thay đổi trong môi trường của nó theo những cách rất khó dự đoán.

Các nhân tố con người và tổ chức ảnh hưởng đến thiết kế hệ thống bao gồm: các thay đổi về tiến trình nghiệp vụ, các thay đổi về công việc, các thay đổi tổ chức và quản lý. Nhân tố con người, xã hội và tổ chức này thường là yếu tố quyết định trong việc xác định một hệ thống có đạt được những mục tiêu của nó hay không. Tuy nhiên, việc dự đoán ảnh hưởng của chúng lên hệ thống là rất khó khăn đối với các kỹ sư phát triển, vì họ có rất ít kinh nghiệm về quản lý, xã hội hay văn hóa.

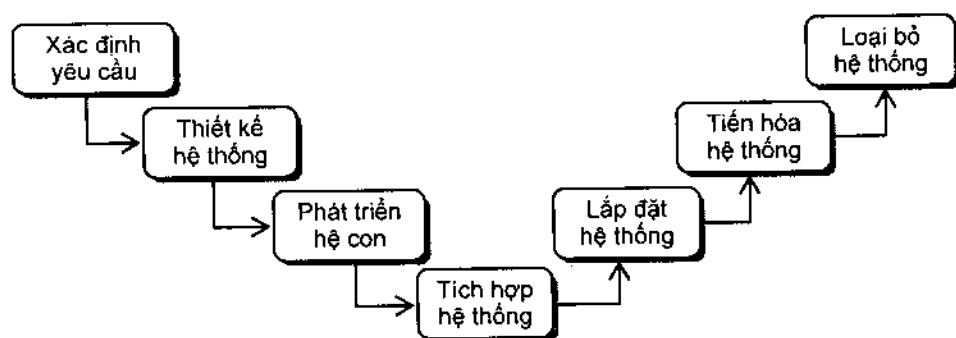
1.5.2. Mô hình hóa hệ thống

Kiến trúc hệ thống thường được mô tả như một biểu đồ khối, biểu diễn các hệ thống con chính và các kết nối bên trong giữa chúng. Mỗi hệ thống con được biểu diễn bởi một hình chữ nhật trong biểu đồ khối và quan hệ giữa các hệ thống con được mô tả bằng các mũi tên nối các hình chữ nhật này.

Mô hình kiến trúc hệ thống được sử dụng để xác định các thành phần phần cứng và phần mềm, là những thành phần có thể phát triển song song. Một kiến trúc hệ thống nên được thiết kế dưới dạng các hệ thống con chức năng, mà không cần quan tâm hệ thống đó là hệ thống con phần cứng hay phần mềm. Các thành phần chức năng tiêu biểu trong một hệ thống cần được phân loại theo một số tiêu chí sau:

- Các thành phần thu nhận (sensor, thiết bị ngoại vi) thông tin.
- Các thành phần thao tác (actuator) (gây ra sự thay đổi lên môi trường).
- Các thành phần tính toán (biến đầu vào thành đầu ra).
- Các thành phần truyền thông (giữa các thành phần và với môi trường).
- Các thành phần phối hợp (phối hợp hoạt động của các thành phần).
- Các thành phần giao diện (chuyển đổi dạng biểu diễn thông tin giữa các thành phần).

1.5.3. Tiến trình kỹ nghệ hệ thống



Hình 1.16. Tiến trình kỹ nghệ hệ thống dựa trên máy tính

Có một sự khác biệt quan trọng giữa tiến trình kỹ nghệ hệ thống và tiến trình kỹ nghệ phần mềm, đó là:

1. *Các nguyên lý bên trong*: Có nhiều nguyên lý khác nhau được áp dụng cho kỹ nghệ hệ thống.
2. *Giảm bớt phạm vi phải làm lại trong quá trình phát triển hệ thống*: Việc làm lại ở mức hệ thống sẽ là rất đắt. Vì vậy, kỹ nghệ hệ thống nhằm giảm phạm vi gây ra những chi phí này. Tiến trình kỹ nghệ được mô tả trong sơ đồ hình 1.16.

1.5.3.1. Xác định các yêu cầu hệ thống

Xác định các yêu cầu của hệ thống bao gồm:

1. *Các yêu cầu chức năng trừu tượng*: Các chức năng cơ bản mà hệ thống phải cung cấp được xác định ở mức trừu tượng, khái quát. Các đặc tả yêu cầu chức năng chi tiết được xác định ở mức hệ thống con.
2. *Các thuộc tính hệ thống*: Đó là các thuộc tính nổi trội phi chức năng, bao gồm tính sẵn sàng, hiệu suất, an toàn,... Những thuộc tính phi chức năng này ảnh hưởng đến các yêu cầu của mọi hệ thống con trong nó.
3. *Các đặc tính mà hệ thống không bộc lộ*.

Một phần quan trọng của giai đoạn xác định các yêu cầu là thiết lập một tập các mục tiêu tổng thể mà hệ thống cần có. Chúng giải đáp được cho câu hỏi đặt ra là: Tại sao hệ thống được phát triển là cần thiết cho môi trường đích cụ thể nêu ra?

1.5.3.2. Thiết kế hệ thống

Các hoạt động trong tiến trình thiết kế hệ thống bao gồm:

1. *Phân chia các yêu cầu*: Các yêu cầu của hệ thống được phân tích, thu thập và đưa vào trong các nhóm liên quan.
2. *Xác định các hệ thống con*: Xác định các hệ thống con khác nhau có thể đáp ứng được các yêu cầu một cách độc lập hay hợp tác.
3. *Gán các yêu cầu cho các hệ thống con*: Các yêu cầu được gán cho các hệ thống con. Điều này sẽ là đơn giản nếu việc phân chia yêu cầu được sử dụng để xác định các hệ thống con.
4. *Đặc tả chức năng hệ thống con*: Đặc tả các chức năng cụ thể mà mỗi hệ thống con cung cấp. Đây có thể xem là một phần trong thiết kế hệ thống nếu hệ thống con là một hệ thống phần mềm, hoặc là một phần của hoạt động đặc tả các yêu cầu cho hệ thống đó.

5. *Xác định các giao diện của mỗi hệ thống con*: Nó bao gồm việc xác định các giao diện được yêu cầu đối với mỗi hệ thống con. Khi những giao diện này được chấp nhận, các hệ thống con có thể được phát triển song song.

1.5.3.3. Phát triển hệ thống con

Trong quá trình phát triển hệ thống, các hệ thống con được xác định khi thiết kế hệ thống được thực thi. Nó có thể bao gồm việc sử dụng các tiến trình kỹ nghệ hệ thống khác nhau cho mỗi hệ thống đơn. Trong số đó, hệ thống con quan trọng là hệ thống phần mềm. Các hệ thống con được phát triển độc lập và song song. Khi có vấn đề xuất hiện trong tất cả các hệ thống con, thì yêu cầu sửa đổi cần được thực hiện ở mức toàn hệ thống.

1.5.3.4. Tích hợp hệ thống

Tích hợp hệ thống bao gồm việc lấy các hệ thống đã phát triển độc lập và lắp ghép chúng lại cùng nhau để tạo lên một hệ thống hoàn thiện. Việc tích hợp có thể được theo phương pháp "big bang", trong đó tất cả các hệ thống con được tích hợp cùng một lúc, hay bằng một tiến trình tăng dần. Tiến trình tăng dần là cách tiếp cận phù hợp nhất vì hai lý do sau:

1. Không thể lập kế hoạch phát triển tất cả các hệ thống con cùng một lúc.
2. Tích hợp tăng dần làm giảm chi phí của việc định vị và sửa lỗi.

1.5.3.5. Cài đặt hệ thống

Trong quá trình cài đặt hệ thống, mỗi hệ thống con được đặt vào môi trường mà ở đó nó được dự định sẽ hoạt động. Những vấn đề có thể nảy sinh là:

1. Môi trường hệ thống được cài đặt không giống với môi trường mà người phát triển hệ thống dự kiến (vì thế khi phát triển phải giả lập môi trường đích).
2. Những người dùng của hệ thống có thể không mong muốn đưa hệ thống vào hoạt động.
3. Một hệ thống mới có thể cùng tồn tại với một hệ thống cũ cho tới khi tổ chức thỏa mãn với hoạt động của hệ thống mới.
4. Có thể gặp phải những vấn đề liên quan đến cài đặt vật lý.

1.5.3.6. Hoạt động của hệ thống

Để đưa hệ thống vào hoạt động cần các khóa đào tạo cho người vận hành và hoàn thiện quy trình nghiệp vụ, để có thể sử dụng hiệu quả hệ thống mới. Các vấn đề không được phát hiện có thể sẽ phát sinh trong giai đoạn này, vì đặc tả hệ thống có thể có các lỗi hay bỏ sót vấn đề. Một vấn đề có thể chỉ phát sinh sau khi hệ thống đi vào hoạt động là vấn đề tương thích của hệ thống mới với các hệ thống đang tồn tại. Vì vậy, hoạt động bảo trì như là một yêu cầu cấp thiết.

1.5.3.7. Tiến hóa hệ thống

Các hệ thống lớn và phức tạp có thời gian sống dài. Trong suốt thời gian sống, chúng phải sửa các lỗi trong các yêu cầu hệ thống ban đầu và có các yêu cầu mới phát sinh. Sự tiến hóa hệ thống, giống như sự tiến triển của phần mềm, là khá tốn kém. Việc bổ sung, sửa đổi, hay tiến hóa hệ thống đến mức nào đó có thể tạo thành một hệ mới khác về cơ bản với hệ thống ban đầu. Khi đó hệ thống bước sang vòng đời của một hệ mới.

1.5.3.8. Loại bỏ hệ thống

Loại bỏ hệ thống có nghĩa là làm cho hệ thống ngừng làm việc sau khi nó kết thúc thời gian hoạt động có ích.

Tổng kết

- ◆ Phần mềm là một sản phẩm logic, gồm ba bộ phận cấu thành là: các tập mã lệnh, các cấu trúc dữ liệu và các tài liệu mô tả cách sử dụng và bảo trì nó. Phần mềm không mòn cũ, nhưng thoái hóa theo thời gian. Thay đổi và phức tạp luôn là bản chất của nó.
- ◆ Có ba cách phân loại phần mềm: theo *mức độ hoàn thiện* (chương trình và sản phẩm, theo chức năng); theo *vai trò* (hệ thống, công cụ và các ứng dụng) và theo *lĩnh vực được ứng dụng* trong đời sống xã hội.
- ◆ Thách thức đối với việc sản xuất phần mềm hiện nay là cần tạo ra phần mềm mới nhanh, quy mô lớn, giá hạ và cố gắng sử dụng được khả năng kỹ thuật tăng rất nhanh của phần cứng. Cần bảo trì phần mềm cũ một cách hiệu quả.
- ◆ Kỹ nghệ phần mềm là việc thiết lập và sử dụng đúng đắn các nguyên lý công nghệ có được để tạo ra phần mềm một cách kinh tế, vừa tin

cây, vừa làm việc hiệu quả trên các máy tính thực. Ba yếu tố chủ chốt của kỹ nghệ phần mềm là phương pháp, công cụ và các thủ tục. Phương pháp liên quan đến giải pháp kỹ thuật giải quyết các vấn đề; công cụ liên quan đến phương tiện trợ giúp và thủ tục liên quan đến cách tổ chức và quản lý quá trình làm phần mềm.

- ◆ Một tập các hoạt động có thứ tự và cách thức thực hiện và theo đó có thể tạo ra phần mềm một cách công nghiệp gọi là một tiến trình phần mềm. Mô hình tiến trình phần mềm là sự mô tả một cách khái quát, đơn giản hóa tiến trình phần mềm. Có một số loại mô hình tiến trình tiêu biểu là: mô hình thác nước, các mô hình tiến hóa (làm bản mẫu, xoắn ốc, RUP, tăng dần, phát triển nhanh), mô hình biến đổi hình thức và mô hình phát triển hướng sử dụng lại. Mỗi mô hình có mặt mạnh và hạn chế của nó, và thích hợp với những loại bài toán ứng dụng khác nhau. Trên thực tế, người ta thường sử dụng mô hình lai.
- ◆ Các thuộc tính của phần mềm được kỹ nghệ tốt là có thể bảo trì được, đáng tin cậy, có hiệu quả, khả dụng và giá thành chấp nhận được. Đánh giá chất lượng của phần mềm chỉ có tính tương đối. Một số tiêu chí để xem xét chất lượng phần mềm là đạt được các mục tiêu thiết kế, chi phí vận hành chấp nhận được, đáp ứng được các chuẩn mực chung, sản phẩm tạo ra có giá trị, bảo trì được, khả dụng, mềm dẻo, có tính khả chuyển.
- ◆ Một hệ thống dựa trên máy tính gồm có các thiết bị tin học, các phần mềm, các dữ liệu, con người và các thủ tục làm việc được tổ chức lại để tạo ra các chức năng hay dịch vụ phục vụ cho một tổ chức,... Các thuộc tính nổi trội của hệ thống chính là các chức năng và dịch vụ mà nó tạo ra ở mức hệ thống như: tính hoàn thiện, tính tin cậy, tính bảo mật, an toàn.
- ◆ Hệ thống có thể tác động lên môi trường và ngược lại, cũng bị ảnh hưởng bởi các yếu tố từ môi trường. Vì vậy, thiết kế hệ thống cần tính đến các yếu tố đó. Tiến trình kỹ nghệ hệ thống là khó khăn và phức tạp do xuất phát từ các nguyên tắc của kỹ nghệ. Nó bao gồm các hoạt động xác định yêu cầu, thiết kế, phát triển hệ con, tích hợp hệ thống, tiến hóa hệ thống, loại bỏ hệ thống...

Câu hỏi ôn tập

1. Phần mềm là gì? Nêu các đặc trưng của nó. Có những loại ngôn ngữ nào để phát triển phần mềm?
2. Phân loại phần mềm và nội dung cơ bản mỗi loại. Phân biệt chương trình với sản phẩm. Phân biệt chương trình hệ thống với sản phẩm hệ thống.
3. Định nghĩa kỹ nghệ phần mềm. Những yếu tố chủ chốt trong kỹ nghệ phần mềm là gì?
4. Hiểu thế nào là các phương pháp trong kỹ nghệ phần mềm? Nêu các thành phần chính của một phương pháp.
5. Công cụ cho kỹ nghệ phần mềm là gì? Có những loại công cụ gì? CASE là gì? Ý nghĩa của nó.
6. Các thủ tục trong kỹ nghệ phần mềm là gì? Ý nghĩa của nó.
7. Tiến trình phần mềm là gì? Mô hình tiến trình là gì? Hãy trình bày mô hình của một số tiến trình cơ bản.
8. Thế nào là một phần mềm được kỹ nghệ tốt?
9. Phân biệt giữa khoa học máy tính và kỹ nghệ phần mềm.
10. Phân biệt giữa kỹ nghệ hệ thống và kỹ nghệ phần mềm.
11. Các bước tổng quát của tiến trình phần mềm gồm những giai đoạn nào? Nêu các hoạt động của tiến trình phần mềm và tài liệu mà nó sinh ra?
12. Chất lượng phần mềm là gì? Các tiêu chí của chất lượng phần mềm.
13. Hệ thống dựa trên máy tính là gì? Tại sao kỹ sư phát triển phần mềm cần phải có hiểu biết về kỹ nghệ hệ thống?
14. Một hệ thống dựa trên máy tính gồm những thành phần cơ bản nào?
15. Những thuộc tính gì là nội trội của một hệ thống dựa trên máy tính?
16. Độ tin cậy của một hệ thống phụ thuộc vào những yếu tố nào?
17. Kỹ nghệ hệ thống hướng đến những mục tiêu gì trong phát triển hệ thống?
18. Trình bày mối quan hệ giữa tiến trình hệ thống và tiến trình kỹ nghệ phần mềm.
19. Vẽ sơ đồ của tiến trình kỹ nghệ hệ thống. Mô tả nội dung cụ thể của mỗi bước của tiến trình hệ thống.

Chương 2

XÁC ĐỊNH VÀ ĐẶC TẢ YÊU CẦU PHẦN MỀM

2.1. Tổng quan về yêu cầu phần mềm

2.1.1. Khái niệm yêu cầu phần mềm

Hiều được bản chất vấn đề cần phải giải quyết thường là công việc vô cùng phức tạp, đặc biệt khi nó là một hệ thống mới hay một hệ thống đang tồn tại được dùng làm mô hình cho phần mềm muốn tạo ra. Trên thực tế, người ta dễ lẫn lộn giữa *nhu cầu* và *yêu cầu* của người dùng, mà thực sự chúng không phải là 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 (mong muốn)* đơn giản như thế cho người kỹ sư phần mềm và hy vọng có được một phần mềm chấp nhận được và dùng tốt là điều không thể. Thông tin của vấn đề cần giải quyết phải được thu thập, phân tích, phải xác định nội dung vấn đề một cách rõ ràng, đầy đủ mới có được *những yêu cầu* xác đáng cho hệ thống. Khi đó giải pháp phần mềm cho *những yêu cầu* này mới có thể được thiết kế và thực thi.

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* được gọi là *nắm bắt và phân tích yêu cầu*. Kết quả của công việc này là bản *đặc tả yêu cầu*, nó 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 việc mà hệ thống 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) hay *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 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 phát biểu bằng ngôn ngữ tự nhiên cùng biểu đồ để mô tả các dịch vụ mà hệ thống cung cấp và các 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 2.1 cho một ví dụ về yêu cầu người dùng và sự mở rộng, chi tiết nó để trở thành một vài yêu cầu hệ thống.

Yêu cầu người dùng:

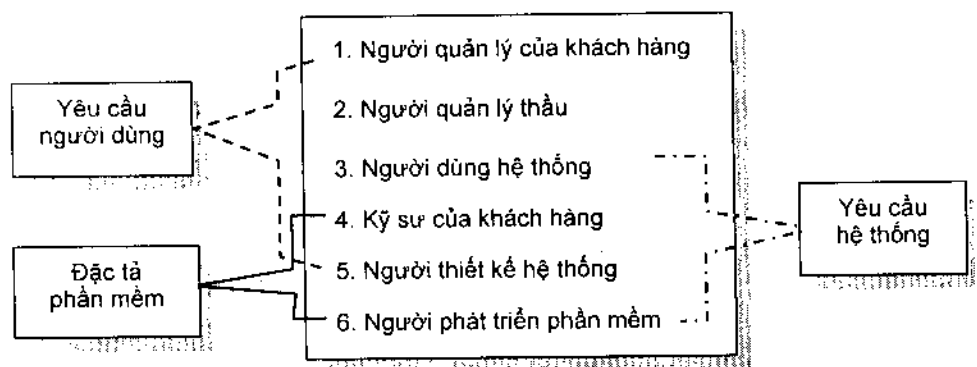
Phần mềm phải cung cấp phương tiện biểu diễn và truy cập tới những file ngoài được tạo ra bởi một công cụ khác.

Đặc tả yêu cầu hệ thống:

1. Người dùng phải được cung cấp phương tiện để dễ dàng xác định được kiểu file ngoài.
2. Với mỗi kiểu file ngoài cần có một công cụ thích hợp cho nó.
3. Mỗi kiểu file ngoài có thể được thể hiện bằng một biểu tượng (icon) trên màn hình của người dùng.
4. Phương tiện phải được cung cấp cho mỗi biểu tượng đại diện của một kiểu file ngoài khi người dùng xác định.
5. Khi người dùng chọn một biểu tượng cho kiểu file ngoài, thì công cụ thích hợp được áp dụng để truy nhập và trình diễn kiểu file đó.

Hình 2.1. Tài liệu xác định yêu cầu người dùng và yêu cầu hệ thống

Mô tả yêu cầu hệ thống bằng các mức khác nhau là rất có ích, vì chúng giúp truyền đạt thông tin về hệ thống tới những người đọc khác nhau. Hình 2.2 biểu diễn các lớp người đọc tương ứng với các mức trừu tượng khác nhau của tài liệu yêu cầu.



Hình 2.2. Người đọc của những tài liệu đặc tả khác nhau

Yêu cầu người dùng phải định hướng tới mức quản lý, tức là cả khách hàng và ban quản lý thầu 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, những người tham gia vào quá trình phát triển hệ thống.

2.1.2. Phân loại các yêu cầu phần mềm

Các yêu cầu của một hệ thống phần mềm có thể phân thành ba loại:

- *Các yêu cầu chức năng* (functional requirements) là những phát biểu về các chức năng, hay dịch vụ mà hệ thống cung cấp. Yêu cầu chức năng cũng có thể là chức năng mà hệ thống không nên thực hiện.
- *Các yêu cầu phi chức năng* (non – functional requirements) là những ràng buộc lên các dịch vụ, hoặc chức năng mà hệ thống cung cấp. Nó bao gồm các ràng buộc về thời gian, về ngân sách; các ràng buộc trong quá trình phát triển, hay các ràng buộc về chuẩn sử dụng,...
- *Các yêu cầu miền lĩnh vực* (domain requirements) là các yêu cầu xuất phát từ miền ứng dụng. Nó phản ánh các đặc trưng của lĩnh vực

ứng dụng này (các quy tắc nghiệp vụ). Chúng có thể là chức năng hay phi chức năng.

Sự phân biệt giữa các loại yêu cầu trên chỉ là tương đối.

2.1.2.1. Yêu cầu chức năng

Các yêu cầu chức năng cho một hệ thống mô tả các hoạt động của hệ thống hay các dịch vụ mà hệ thống sẽ cung cấp. Những chức năng này phụ thuộc vào loại phần mềm được xây dựng, vào sự mong đợi của người dùng, vào quy mô hệ thống được phát triển và môi trường mà ở đó nó được cài đặt (môi trường đích).

2.1.2.2. Yêu cầu phi chức năng

Yêu cầu phi chức năng là các yêu cầu không liên quan trực tiếp tới các hoạt động chức năng cụ thể của hệ thống. Chúng có thể liên quan đến các thuộc tính quan trọng của hệ thống, đến hiệu năng của hệ thống như *độ tin cậy, thời gian đáp ứng của dịch vụ, việc sử dụng bộ nhớ*, hay đến những ràng buộc lên hệ thống như khả năng vào/ra của thiết bị, sự biểu diễn dữ liệu trên các giao diện của hệ thống,...

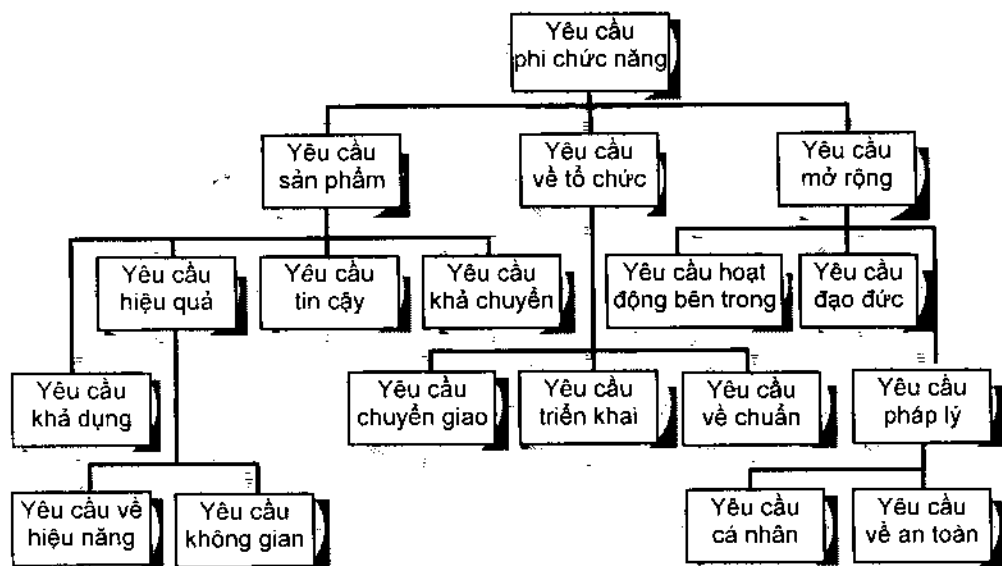
Rất nhiều yêu cầu phi chức năng liên quan đến tổng thể hệ thống hơn là đến từng bộ phận riêng rẽ. Điều đó có nghĩa là, chúng thường quan trọng hơn rất nhiều những chức năng bộ phận (trong khi lỗi của một chức năng bộ phận có thể sửa đổi và nâng cấp, thì lỗi của một yêu cầu phi chức năng có thể làm cho hệ thống ngừng hoạt động, không sử dụng được).

Về nguyên tắc, đặc tả yêu cầu chức năng cần phải *đầy đủ và nhất quán*. Đầy đủ có nghĩa là, tất cả chức năng mà người dùng cần ở hệ thống phải được xác định. Nhất quán nghĩa là, các yêu cầu nêu ra không được mâu thuẫn với nhau. Tuy nhiên, đối với một hệ thống lớn và phức tạp, điều này rất khó có thể đạt ngay được.

Sommeville [34] đã mô tả các kiểu khác nhau của yêu cầu phi chức năng như hình 2.3 và hình 2.4 là ví dụ về đặc tả các yêu cầu phi chức năng.

1. *Yêu cầu về sản phẩm:* Yêu cầu này xuất hiện từ những đòi hỏi của sản phẩm sẽ được phân phối sử dụng trong những trường hợp cụ thể. Ví dụ, yêu cầu về thời gian hệ thống phải trả lại kết quả, về tính tin cậy (tỷ lệ lỗi chấp nhận được),... Các yêu cầu này thường trực tiếp từ nhu cầu của người sử dụng.

2. *Yêu cầu mang tính tổ chức*: Những yêu cầu này xuất phát từ các quy định và thủ tục của các tổ chức. Ví dụ, một số chuẩn bắt buộc phải áp dụng, như *yêu cầu ngôn ngữ lập trình* hoặc *phương pháp thiết kế* được sử dụng,... Các yêu cầu này có thể từ cả người sử dụng và từ các tổ chức phát triển.
3. *Yêu cầu mở rộng*: Khái niệm này đề chỉ tất cả những yêu cầu xuất phát từ các nhân tố bên ngoài tác động lên hệ thống trong quá trình phát triển và vận hành. Nó bao gồm yêu cầu về *tính tương hợp*: xác định cách thức hệ thống tương tác với các hệ thống khác trong môi trường mà nó hoạt động. Yêu cầu về *tính pháp lý*, yêu cầu về *tính văn hóa* đặt ra nhằm đảm bảo sản phẩm đưa ra phải phù hợp với phong tục, đạo đức, thói quen của người dùng,...



Hình 2.3. Các kiểu khác nhau của yêu cầu phi chức năng

Một lỗi thường thấy là sự lẫn lộn giữa yêu cầu phi chức năng với đòi hỏi chung chung của người dùng. Cách tốt nhất, để chắc chắn rằng, những yêu cầu phi chức năng là xác định đúng khi chúng có thể đo được. Điều này tùy thuộc vào khả năng đo lường được một số đặc trưng của các thuộc tính thuộc hệ thống.

2.1.2.3. Yêu cầu miền lĩnh vực

Các yêu cầu về miền lĩnh vực vốn có từ miền ứng dụng của hệ thống hơn là từ nhu cầu người dùng nó. Chúng có thể là những yêu cầu chức năng mới về quyền sở hữu, những hạn chế đối với các yêu cầu chức năng đang tồn tại,... Các yêu cầu miền là quan trọng, vì nó phản ánh những cơ sở nền tảng của miền ứng dụng. Nếu các yêu cầu đó không được thỏa mãn, hệ thống có thể sẽ không hoạt động một cách đúng đắn. Ví dụ, xét các yêu cầu sau đây đặt ra cho hệ thống thông tin thư viện:

1. Một giao diện người dùng chuẩn dựa trên chuẩn Z39.50 cần được áp dụng cho mọi cơ sở dữ liệu của thư viện.
2. Vì sự hạn chế về bản quyền, một số tài liệu buộc phải xóa ngay lập tức sau khi nhận được.

Trong hai yêu cầu trên, yêu cầu đầu tiên là chức năng, liên quan đến hoạt động chức năng của cơ sở dữ liệu. Yêu cầu thứ hai là phi chức năng. Nó được đưa vào vì Luật Bản quyền áp dụng cho việc sử dụng các tài liệu trong lĩnh vực thư viện.

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 sau tối đa 2 giờ đào tạo. Sau thời gian đào tạo trên thì số lỗi trung bình xảy ra đối với người dùng có kinh nghiệm 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 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.

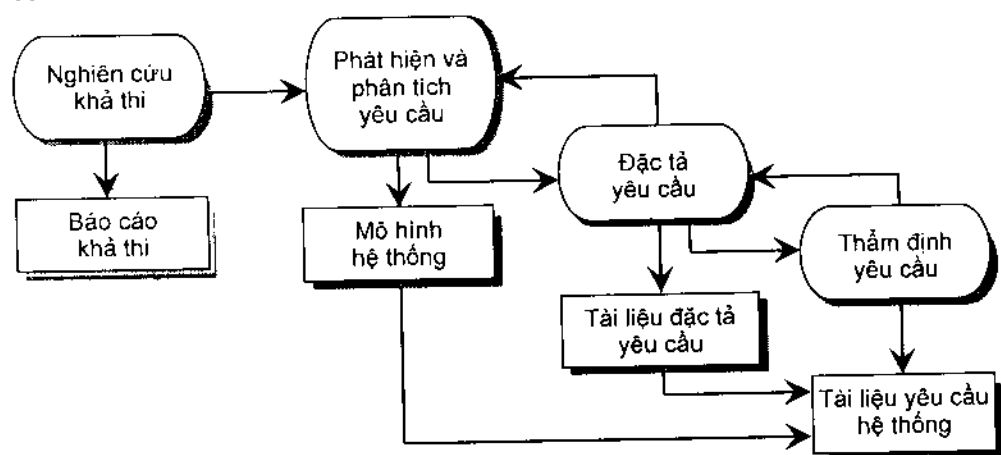
Hình 2.4. Ví dụ về các yêu cầu phi chức năng

2.1.3. Tiến trình kỹ nghệ yêu cầu

Tiến trình kỹ nghệ yêu cầu được biểu diễn ở hình 2.5. Kết quả các hoạt động của nó là các tài liệu yêu cầu phần mềm. Các thông tin khác như *báo cáo về tính khả thi* của hệ thống và *đặc tả phần mềm* cũng có thể sinh ra trong quá trình này. Tiến trình bao gồm bốn giai đoạn chính:

2.1.3.1. Nghiên cứu khả thi (feasibility study)

Trước hết cần đưa ra ít nhất một phương án phát triển hệ thống và luận chứng sự khả thi của nó. Đánh giá khả thi của phương án phải dựa vào các phần mềm và công nghệ phần cứng hiện có có thỏa mãn nhu cầu người dùng hay không. Việc nghiên cứu khả thi sẽ quyết định đưa ra một hệ thống đáp ứng được yêu cầu khách hàng, đảm bảo về công nghệ, có giá cả phải chăng, có thể thực hiện được trong điều kiện ngân sách và thời hạn định trước. Nghiên cứu tính khả thi phải rõ và đủ nhanh. Kết quả cuối cùng phải khẳng định có tiếp tục triển khai phát triển hệ thống theo phương án đó hay không. Chỉ khi dự án khả thi được chấp nhận, quá trình triển khai dự án mới có thể bắt đầu.



Hình 2.5. Tiến trình kỹ nghệ yêu cầu

2.1.3.2. Phát hiện và phân tích yêu cầu (Requirements elicitation and analysis)

Phát hiện và phân tích yêu cầu là bước tiếp theo sau phân tích khả thi. Những nhà phát triển tiếp tục khảo sát các hệ thống đang tồn tại để thu thập thông tin, làm việc với khách hàng và người dùng cuối để nắm và hiểu về miền ứng dụng, và chuyển thông tin thu thập được từ hoạt động phân tích thành tài liệu xác định các yêu cầu. Các tài liệu này phải phản ánh chính xác mong muốn của khách hàng, và được viết sao cho người dùng và khách hàng đặt mua hệ thống có thể hiểu được.

Việc phát hiện và phân tích yêu cầu cũng bao gồm cả việc *phát hiện các loại đối tượng khác nhau* trong tổ chức có liên quan trực tiếp, hay gián tiếp đến yêu cầu của hệ thống, như các kỹ sư đang phát triển và bảo trì các hệ thống liên quan, những người quản lý nghiệp vụ, các chuyên gia miền,... Chính họ là người trợ giúp cho việc làm chính xác hóa và hoàn thiện tài liệu yêu cầu.

Quá trình này cũng bao gồm việc phát triển thử nghiệm một hay nhiều mô hình hệ thống khác nhau. Điều đó giúp các nhà phân tích hiểu được hệ thống. Làm bản mẫu hệ thống có thể được sử dụng trong giai đoạn này.

2.1.3.3. Đặc tả yêu cầu (*requirements specification*)

Đặc tả yêu cầu là mô tả chính xác và chi tiết yêu cầu hệ thống để làm cơ sở cho giao kèo giữa khách hàng và người phát triển hệ thống. Khi đặc tả yêu cầu thường phải sử dụng những công cụ đặc biệt, như các mô hình hay các ngôn ngữ đặc tả chuyên dụng. Việc lập tài liệu này thường được tiến hành song song với các thiết kế mức cao (thiết kế sơ bộ). Trong khi viết tài liệu này, các sai sót trong xác định yêu cầu sẽ được phát hiện và được sửa chữa.

2.1.3.4. Thẩm định yêu cầu (*requirements validation*)

Thẩm định yêu cầu là xét xem đặc tả yêu cầu có mô tả chính xác những gì được đặt ra cho hệ thống và có thể thực hiện được không? Một số thuộc tính của yêu cầu cần thẩm định như: *tính đúng đắn, tính đầy đủ, tính nhất quán (không mâu thuẫn) và tính hiện thực*. Các hoạt động của tiến trình kỹ nghệ yêu cầu thường là quá trình lặp. Phân tích yêu cầu tiếp tục trong quá trình xác định và chi tiết hóa. Các yêu cầu mới thường nảy sinh trong quá trình này. Do đó, các tài liệu sẽ thường xuyên thay đổi và có thể phải thay thế do yêu cầu của hệ thống quản lý cấu hình.

2.1.4. Tài liệu yêu cầu phần mềm

2.1.4.1. Nội dung và yêu cầu đối với tài liệu yêu cầu phần mềm

Yêu cầu hệ thống được biểu diễn bằng *tài liệu yêu cầu phần mềm* hay *đặc tả yêu cầu phần mềm* (SRS – Software Requirements Specification). Nó là tài liệu chính thức cho người phát triển phần mềm. Tài liệu này bao gồm

tài liệu xác định yêu cầu và tài liệu đặc tả yêu cầu. Có thể được trình bày hai tài liệu đó trong một bản mô tả chung, trong đó phần đặc tả như là phụ lục của tài liệu xác định yêu cầu. Nó được dùng để viết bản thiết kế cuối cùng. Vì thế, từ một yêu cầu phải tham chiếu được tới các phần thiết kế tương ứng để thực thi nó. Nếu các dịch vụ, ràng buộc và thuộc tính trong tài liệu yêu cầu phù hợp với bản thiết kế phần mềm thì bản thiết kế đó là lời giải đúng cho bài toán đặt ra.

Heniger [15] đã đưa ra *sáu yêu cầu* cho một tài liệu yêu cầu phần mềm:

1. Chỉ mô tả các hoạt động của hệ thống từ bên ngoài.
2. Chỉ ra được các ràng buộc của hệ thống trong quá trình vận hành.
3. Dễ thay đổi.
4. Phục vụ như tài liệu tham khảo cho người bảo trì hệ thống.
5. Dự toán trước được vòng đời của hệ thống.
6. Mô tả được các đáp ứng đối với những sự cố, thay đổi ngoài dự tính.

Về nguyên tắc, yêu cầu được xác định phải là *đầy đủ* và *nhất quán*. Nhưng việc tồn tại lỗi và thiếu sót trong xác định yêu cầu là điều không thể tránh khỏi. Vì thế, tài liệu cần được cấu trúc sao cho sau này dễ sửa đổi mà không cần phải viết lại. Quá trình quản lý thay đổi yêu cầu được trợ giúp bởi các hệ thống phần mềm quản lý thay đổi cấu hình phần mềm, trong đó có sự thay đổi về yêu cầu phần mềm.

2.1.4.2. Cấu trúc tài liệu yêu cầu phần mềm

Các tổ chức lớn như Bộ Quốc phòng Mỹ hay IEEE đã xác định các chuẩn cho tài liệu yêu cầu. Chuẩn IEEE/ANSI 830 – 1993 đã gợi ý một cấu trúc cho tài liệu yêu cầu như sau:

1. Giới thiệu

- 1.1. Mục tiêu của tài liệu yêu cầu
- 1.2. Phạm vi của sản phẩm
- 1.3. Các định nghĩa, chữ viết tắt và từ viết rút gọn
- 1.4. Tài liệu tham khảo
- 1.5. Tổng quan về tài liệu

2. Mô tả chung

- 2.1. Các đặc điểm của sản phẩm
- 2.2. Các chức năng của sản phẩm
- 2.3. Các đặc trưng người dùng
- 2.4. Các ràng buộc chung
- 2.5. Các giả định và các sự phụ thuộc

3. Các yêu cầu cụ thể

4. Phụ lục

5. Các chỉ số chỉ dẫn

Tài liệu yêu cầu có thể có thêm các chương hay phụ lục với các thông tin sau:

1. *Kiến trúc chung của hệ thống phần cứng*: Nếu hệ thống chạy trên những phần cứng chuyên biệt thì giao diện với nó phải được nói đến. Nếu sử dụng phần cứng đã có thì cấu hình tối ưu cho hệ thống cũng phải được xác định.
2. *Cơ sở dữ liệu*: Tổ chức logic của dữ liệu mà hệ thống sử dụng và các mối quan hệ của chúng cần phải mô tả. Mô hình thực thể – mối quan hệ có thể dùng để mô tả về yêu cầu này. Ngoài ra, cũng đề xuất những hệ quản trị cơ sở dữ liệu có thể sử dụng.
3. *Chú dẫn*: Có thể có nhiều hơn một kiểu chú dẫn trong tài liệu. Ngoài bảng chú dẫn sắp xếp theo bảng chữ cái như thông thường, có thể chú dẫn theo từng chương, chú dẫn trong một chức năng,...

Tài liệu yêu cầu cũng là một công cụ tham khảo: Nó phải bao gồm cả dự đoán vòng đời của hệ thống. Các kỹ sư bảo trì dùng tài liệu này để xác định hệ thống cần làm gì trong vòng đời của nó. Khả năng lần vết là một ưu điểm của đặc tả yêu cầu, vì nó cho phép dễ dàng tìm ra các yêu cầu liên quan. Các kỹ thuật từ đơn giản đến các công cụ CASE được dùng để hỗ trợ chức năng này.

2.2. Xác định yêu cầu phần mềm

2.2.1. Khảo sát hệ thống và phân tích khả thi

Sau khi được khách hàng đặt hàng, các nhà phát triển cần tiến hành khảo sát hiện trạng hệ thống đang vận hành để thu thập thông tin phục vụ

phân tích khả thi [37]. 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 hệ thống cần xây dựng. Ít nhất là ba phương án có thể: *phương án thấp*, *trung bình* và *phương án cao*. Các nhà phân tích tiến hành phân tích các phương án này, lập luận chứng khả thi và chọn phương án thích hợp nhất. Phân tích khả thi và rủi ro có liên quan với nhau theo nhiều cách. Nếu rủi ro của dự án là lớn, thì tính khả thi của việc tạo ra phần mềm chất lượng sẽ bị giảm đi. Phân tích khả thi thường tập trung vào một số mặt:

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 kỹ thuật*: Khảo cứu về chức năng, hiệu suất và ràng buộc có thể ảnh hưởng tới khả năng của hệ thống được chấp nhận. Nói khác đi, là xem xét khả năng kỹ thuật hiện có 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 khả thi không đảm bảo hoàn toàn rằng, hệ thống được minh chứng về kinh tế là hiển nhiên, rủi ro kỹ thuật thấp, ít các vấn đề pháp lý và không có các giải pháp hợp lý khác. Tuy nhiên, nếu bất kỳ một điều kiện nào nói trên không được đáp ứng, thì dự án có nguy cơ thất bại và cần tiến hành nghiên cứu lại. Có thể soạn tư liệu về nghiên cứu khả thi thành một báo cáo riêng cho cấp quản lý và đính kèm các đặc tả hệ thống như phụ lục.

Bản nghiên cứu khả thi được cấp quản lý dự án xem xét độ tin cậy và chính xác về nội dung, sau đó cấp quản lý cao hơn xét duyệt ra quyết định. Bản nghiên cứu *phải đề xuất quyết định "tiến hành/không tiến hành"*. Các quyết định tiến hành khác sẽ được đưa ra trong các bước lập kế hoạch. Sau khi giải pháp và phương án đã được chọn, nhóm phân tích cần lập *hồ sơ*

nhệm vụ. Nó được dùng làm cơ sở cho một thỏa thuận không chính thức giữa ba phía: người phân tích, nhà đầu tư, khách hàng và người sử dụng.

2.2.2. Phát hiện và phân tích yêu cầu

2.2.2.1. Những khó khăn trong việc nhận ra yêu cầu

Sau khi có phương án khả thi, cần tiếp tục thu thập thông tin, nhận biết các yêu cầu và tiến hành phân tích chúng. Việc phân tích này có thể liên quan đến nhiều người khác nhau trong tổ chức: người quản lý, kỹ sư, khách hàng, người dùng, chuyên gia nghiệp vụ, nhà đầu tư. Thuật ngữ "stakeholder" được sử dụng để chỉ tất cả những người có liên quan này.

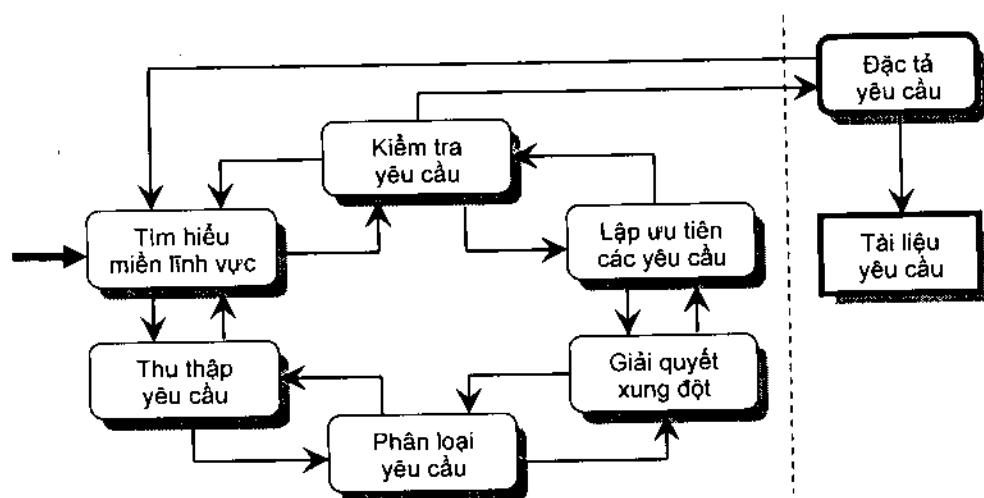
Phân tích yêu cầu là một quá trình khó khăn vì một số lý do sau: Người liên quan thường không thực sự biết họ cần gì từ hệ thống. Họ thường bày tỏ những yêu cầu của mình theo cách nói riêng; các nhà phát triển phải hiểu những yêu cầu này và dịch chúng sang một hình thức thể hiện thống nhất. Họ có thể có các yêu cầu khác nhau; các kỹ sư phải nhận ra điểm chung cũng như điểm khác biệt giữa chúng. Quá trình phân tích diễn ra trong một bối cảnh cụ thể của tổ chức, nên các yêu cầu có thể bị ảnh hưởng bởi các yếu tố chính trị. Ngoài ra, môi trường kinh doanh luôn luôn biến động. Do vậy, yêu cầu mới có thể xuất hiện từ những người liên quan mà lúc đầu không được tham khảo.

Khó khăn có tính nguyên tắc của việc thiết lập các yêu cầu hệ thống phần mềm là các vấn đề "của độc", các vấn đề không được định nghĩa, không có công thức cho trước. Thông qua việc xử lý thông tin thu thập, hiểu biết của người phát triển không ngừng được bổ sung và nâng cao, yêu cầu được thay đổi và hoàn thiện. Điều này dẫn tới sự thiếu ổn định của yêu cầu hệ thống. Vì vậy, nếu mong đợi yêu cầu được xác định rõ ràng rồi mới bắt tay phát triển hệ thống sẽ là không thực tế.

2.2.2.2. Tiến trình phát hiện và phân tích yêu cầu

Hình 2.6 minh họa các hoạt động của tiến trình phân tích yêu cầu. Một chu kỳ của tiến trình bắt đầu bằng việc tìm hiểu lĩnh vực ứng dụng và kết thúc bằng việc thẩm định các yêu cầu. Sự hiểu biết về các yêu cầu được tăng cường sau mỗi chu kỳ. Các hoạt động chung của một tiến trình thường bao gồm:

1. *Tìm hiểu miền ứng dụng*: Đó là các hoạt động nghiệp vụ, nội dung của nó, các quy tắc nghiệp vụ cần tuân thủ, sản phẩm của nó, lĩnh vực liên quan.
2. *Thu thập các yêu cầu*: Đó là quá trình giao tiếp với những người liên quan đến hệ thống để phát hiện ra các yêu cầu của họ đối với hệ thống.
3. *Phân loại các yêu cầu*: Hoạt động này bao gồm việc tập hợp lại các yêu cầu thu được và sắp xếp nó vào loại tương ứng.
4. *Giải quyết xung đột*: Hoạt động này liên quan đến việc tìm ra các xung đột và tìm cách giải quyết chúng.
5. *Sắp ưu tiên*: Giữa các yêu cầu bao giờ cũng có những yêu cầu quan trọng hơn. Bằng cách giao tiếp với những người liên quan, nhà phân tích nhận ra được tầm quan trọng của mỗi yêu cầu và thực hiện sắp xếp chúng.
6. *Kiểm tra yêu cầu*: Các yêu cầu cần được kiểm tra theo từng nghiệp vụ để xem chúng có đầy đủ không, nhất quán không và có phù hợp với những gì mà những người liên quan trông chờ ở hệ thống không.



Hình 2.6. Tiến trình phát hiện và phân tích yêu cầu

Tiến trình các hoạt động trên đây là một quá trình lặp, các thông tin phản hồi luôn nhận được từ một hoạt động này sang hoạt động khác. Mỗi cách nhìn thế giới thực thường được thể hiện dưới dạng như một mô hình

thực thể – thuộc tính – quan hệ. Các mô hình khác nhau được sử dụng để mô tả hệ thống thực. Nhưng cách nhìn tự nhiên là cách nhìn "nghịệp vụ", nên mô hình sử dụng thường được gọi chung là *mô hình nghịệp vụ*. Tuy nhiên, tùy theo mỗi cách tiếp cận mà mô hình nghịệp vụ được biểu diễn theo những cách khác nhau [37].

2.2.3. Các kỹ thuật phân tích yêu cầu

Có nhiều phương pháp khác nhau để tiếp cận việc xác định yêu cầu. Dưới đây sẽ khái quát hóa về các cách tiếp cận đó.

2.2.3.1. Tiếp cận yêu cầu định hướng cách nhìn

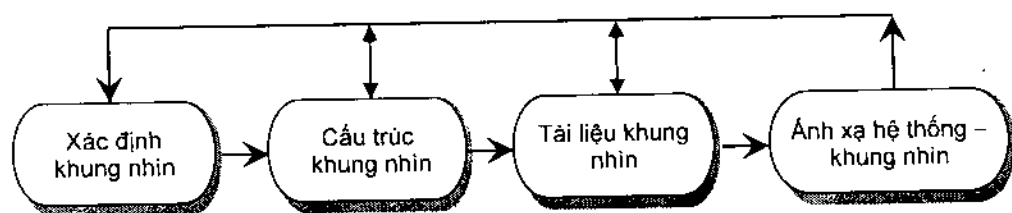
Với những hệ thống vừa và lớn, thường có nhiều loại người dùng cuối khác nhau. Cách tiếp cận hướng cách nhìn (viewpoint) [34] đối với kỹ nghệ yêu cầu ghi nhận những cách nhìn nhận khác nhau của những người liên quan, sử dụng nó vào tiến trình phát hiện yêu cầu và tổ chức các yêu cầu. Mỗi cách nhìn có thể được xem xét từ một trong các góc độ sau:

1. *Từ nguồn hay đích tới 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 bao gồm việc xác định dữ liệu gì được tạo ra, được sử dụng và những xử lý nào được tiến hành. Nó diễn hình cho phương pháp phân tích vào/ra (Input – Output analysis).
2. *Từ khung làm việc trình diễn*: Cách nhìn này xem xét từ loại hình công cụ mô tả mô hình hệ thống. Ví dụ như, mô hình *thực thể – mối quan hệ*, mô hình *máy trạng thái hữu hạn*. Mỗi cách tiếp cận phát hiện ra những đối tượng khác nhau của hệ thống được phân tích và những mối quan hệ của chúng.
3. *Từ sự tiếp nhận dịch vụ*: Cách nhìn này từ ngoài vào hệ thống và sự tiếp nhận dịch vụ từ hệ thống. Các quan điểm này cung cấp dữ liệu cho các dịch vụ và các tín hiệu kiểm tra. Việc phân tích bao gồm, kiểm tra các dịch vụ nhận được theo quan điểm khác nhau, thu thập chúng và giải quyết các xung đột.

2.2.3.2. Kỹ thuật xác định yêu cầu định hướng cách nhìn – VORD (Viewpoint – Oriented Requirements Definition)

Phương pháp này được Kotonua and Sommeville đề xuất năm 1996 [17], giai đoạn cơ bản của VORD được chỉ ra trên hình 2.7.

1. *Xác định khung nhìn*, bao gồm việc tìm kiếm các khung nhìn thu nhận các dịch vụ hệ thống và việc xác định các dịch vụ cung cấp cho từng khung nhìn.
2. *Cấu trúc khung nhìn*, bao gồm 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.
3. *Làm tài liệu khung nhìn*, bao gồm việc làm mịn các mô tả khung nhìn và dịch vụ được xác định.
4. *Ánh xạ hệ thống – khung nhìn*: Xác định các đối tượng trong một thiết kế hướng đối tượng khi sử dụng các thông tin dịch vụ được bao gói trong một khung nhìn.



Hình 2.7. Tiến trình của phương pháp VORD

Phương pháp VORD sử dụng các mẫu chuẩn để thu thập các thông tin khung nhìn và dịch vụ (bảng 2.1). Nó cũng sử dụng các biểu đồ khác như biểu đồ phân cấp khung nhìn, đồ thị dữ liệu/điều khiển. Nói chung, một hành động đơn lẻ không thể có đầy đủ thông tin cho một mẫu. Các thông tin hệ thống thường xuất hiện trong tất cả các giai đoạn. Vì vậy, các thông tin này chỉ được hoàn chỉnh ở giai đoạn cuối của quá trình phân tích.

Bảng 2.1. Các mẫu tiêu bản dịch vụ và khung nhìn

Tiêu bản khung nhìn		Tiêu bản dịch vụ	
Tham chiếu	Tên khung nhìn	Tham chiếu	Tên dịch vụ
Thuộc tính	Các thuộc tính cung cấp thông tin khung nhìn	Cơ sở hợp lý	Lý do vì sao dịch vụ được cung cấp
Các sự kiện	Một tham chiếu tới một tập hợp các kịch bản mô tả cách thức hệ thống đáp lại các sự kiện	Đặc tả	Tham chiếu tới danh sách các đặc tả dịch vụ. Chúng có thể được mô tả bằng các ký hiệu khác nhau
Các dịch vụ	Một tham chiếu tới một tập các mô tả dịch vụ	Các khung nhìn	Danh sách tên các khung nhìn nhận dịch vụ
Các khung nhìn con	Tên của các khung nhìn con	Các yêu cầu phi chức năng	Tham chiếu tới tập hợp các yêu cầu phi chức năng ràng buộc lên dịch vụ
		Nhà cung cấp	Danh sách các đối tượng hệ thống cung cấp dịch vụ

2.2.3.3. Kỹ thuật phân tích yêu cầu dựa trên mô hình

a) Mô hình phân tích yêu cầu – mô hình nghiệp vụ

Phân tích dựa trên mô hình được sử dụng rộng rãi để phân tích yêu cầu. Một số mô hình được xây dựng để dành riêng cho việc suy luận và phân tích yêu cầu. Một số khác lại rất gần với việc định nghĩa cho một phương pháp. Các phân tích dựa trên mô hình điển hình đi theo hai hướng tiếp cận:

1. *Tiếp cận định hướng chức năng* là các phương pháp phân tích hướng cấu trúc dựa trên luồng dữ liệu.
2. *Tiếp cận định hướng đối tượng*.

Để phát hiện được các yêu cầu, cả hai cách tiếp cận trên đều tập trung hướng vào mô tả mặt nghiệp vụ của hệ thống thực và kết quả thu được là một *mô hình nghiệp vụ*. Mô hình nghiệp vụ được xây dựng trên cơ sở của phương pháp phân tích hệ thống [37], bao gồm:

1. Một *mô hình ngữ cảnh* mô tả hệ thống được xét trong môi trường của nó. Ở đây đặc biệt chú ý đến các yếu tố môi trường có tương tác với hệ thống, gọi là các *tác nhân*. Thông qua 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

thông với tác nhân được mô tả bằng cách khác nhau (cả khái niệm và ký pháp).

2. *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 mô tả chức năng ở mức gộp hay chi tiết khác nhau. Ở mức thấp nhất, nội dung các chức năng đủ để hiểu và có thể mô tả chi tiết nó với các thông tin cần thiết. Tùy theo cách tiếp cận mà khái niệm chức năng và liên kết giữa chúng cũng như ký pháp sử dụng là khác nhau. Ở đây các phương pháp phân hoạch/phân rã các chức năng khác nhau được sử dụng.
3. *Mô tả chi tiết các chức năng*: Một cách tự nhiên, mô tả chi tiết chi cần thiết và có thể thực hiện với các chức năng mức thấp nhất – chức năng cơ sở. Tuy nhiên, cũng có thể mô tả chức năng ở mức cao (theo hướng đối tượng) để dễ hiểu và lần vết trong quá trình tiếp cận dần. Tùy thuộc vào cách tiếp cận, nội dung và cách mô tả chức năng cũng khác nhau.
4. *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 về cơ bản. 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.
5. *Mô tả mối liên kết giữa dữ liệu và chức năng*: Mô tả này chỉ cần thiết đối với cách tiếp cận hướng cấu trúc.
6. *Từ điển giải thích* liên quan đến các khái niệm và dữ liệu sử dụng. Từ điển này là cần thiết để có sự hiểu biết chung thống nhất về hệ thống thực được mô tả cũng như các đặc trưng của các dữ liệu cần lưu giữ.

b) Các thể hiện của mô hình phân tích

Mô hình nghiệp vụ của hoạt động phân tích yêu cầu theo hai cách tiếp cận định hướng cấu trúc và định hướng đối tượng được thể hiện trong bảng 2.2.

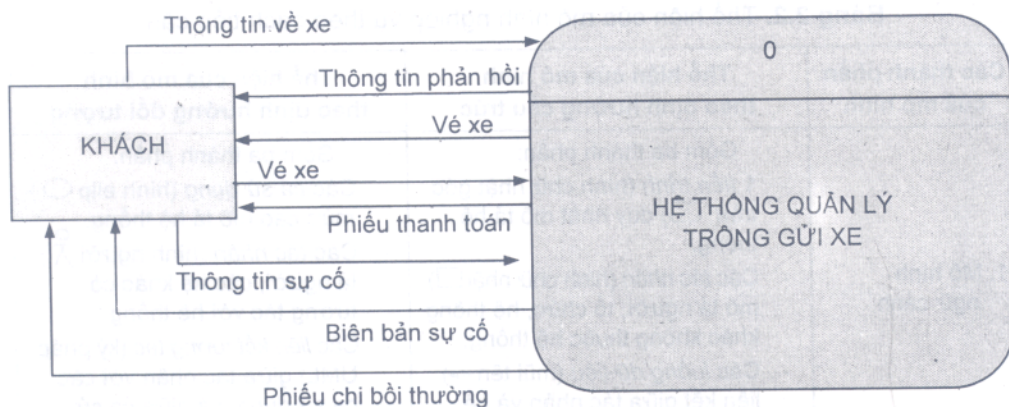
Bảng 2.2. Thể hiện của mô hình nghiệp vụ theo cách tiếp cận

Các thành phần của mô hình	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 ba thành phần: - 1 <i>tiền trình</i> (hình chữ nhật góc tròn □) duy nhất mô tả hệ thống. - Các <i>tác nhân</i> (hình chữ nhật □) mô tả người, tổ chức, hệ thống khác không thuộc hệ thống. - Các <i>luồng dữ liệu</i> (mũi tên →) liên kết giữa tác nhân và hệ thống.	Gồm ba thành phần: - Các <i>ca sử dụng</i> (hình elip ○) mức cao mô tả hệ thống. - Các <i>tác nhân</i> (hình người 人) là người, các hệ khác có tương tác với hệ thống. - Các <i>liên kết tương tác</i> (ký pháp UML) giữa tác nhân với các ca sử dụng, và giữa ca sử dụng với nhau.
2. Mô hình cấu trúc các chức năng	- Gồm một hay một số <i>biểu đồ phân cấp hình cây</i>, mỗi nút 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 các <i>biểu đồ ca sử dụng</i> tương ứng với các mức khác nhau. - Biểu đồ của mức dưới nhận được từ sự phân rã các ca sử dụng 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	- <i>Danh sách các hồ sơ dữ liệu</i> sử dụng. - Các mẫu hồ sơ, từ điển dữ liệu.	- <i>Mô hình miền lĩnh vực</i> 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.	Từ điển giải thích khái niệm.

c) Ví dụ về đặc tả yêu cầu

c1) Ví dụ đặc tả hướng cấu trúc

Ví dụ dưới đây là một minh họa cho mô hình nghiệp vụ của hoạt động trông gửi xe ở một bãi gửi xe, giúp cho việc phân tích và nắm bắt yêu cầu.



Hình 2.8. Biểu đồ ngữ cảnh của hệ thống



Hình 2.9. Biểu đồ phân rã chức năng hoạt động trông gửi xe

Bảng 2.3. Danh sách các hồ sơ dữ liệu sử dụng

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



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

Hình 2.10. Mô tả yêu cầu một chức năng sơ cấp

Các thực thể						
a. Bảng giá (phân loại xe)						
b. Vé xe						
c. Sổ xe vào						
d. Sổ xe ra						
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

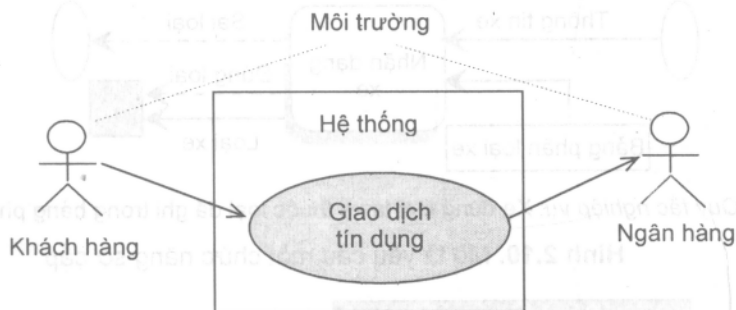
(Chú thích: **C** – Create; **R** – Read; **U** – Update.

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)

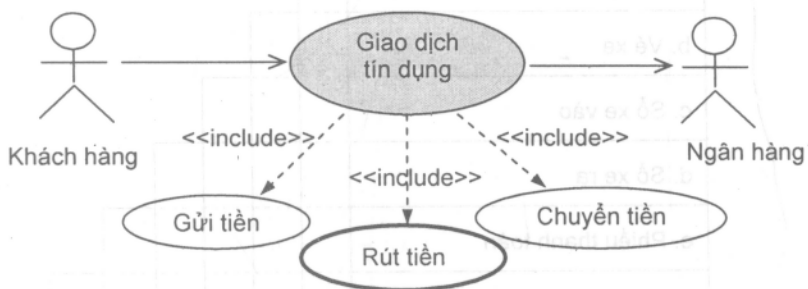
Hình 2.11. Ma trận thực thể – chức năng

c2) Ví dụ đặc tả theo hướng đối tượng

Dưới đây đưa ra một ví dụ về một mô hình nghiệp vụ cho hoạt động của một máy ATM của ngân hàng cung cấp dịch vụ tín dụng cho khách hàng (bỏ qua bảng từ điển giải thích).



Hình 2.12. Biểu đồ ca sử dụng mức cao



Hình 2.13. Biểu đồ ca sử dụng chi tiết

Bảng 2.4. Mô tả chi tiết ca sử dụng rút tiền

Tên ca sử dụng	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:

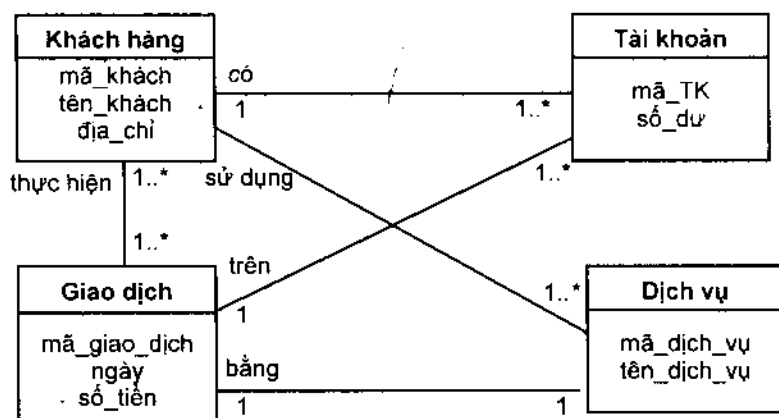
Bước 2: Nếu định danh sai, ca sử dụng kết thúc.

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 là một vấn đề quan trọng trong sự phát triển các công cụ CASE. Thực vậy, đối với nhiều người, thuật ngữ *CASE tool* có ý nghĩa như một công cụ để trợ giúp trong việc phân tích và phát triển mô hình của một ứng dụng.



Hình 2.14. Mô hình miền lĩnh vực của ATM

2.2.3.4. Kỹ thuật phân tích hình thức hóa

Kỹ thuật phân tích hình thức hóa dựa trên việc sử dụng các khái niệm, ký pháp và mô hình toán học để phân tích và biểu diễn hệ thống. Việc phân tích không tách thành mô hình riêng, mà kết quả của hoạt động phân tích và mô hình hóa cho ta ngay đặc tả yêu cầu của hệ thống. Các mô hình đặc tả yêu cầu sẽ xét ở mục sau.

2.2.4. Tiến hóa và quản lý yêu cầu

2.2.4.1. Sự cần thiết tiến hóa yêu cầu

Cùng với việc xác định yêu cầu, nhu cầu từ phía người dùng cũng được hiểu tốt hơn. Thông tin phản hồi lại tới người dùng làm cho yêu cầu bị thay đổi. Hơn nữa, phân tích yêu cầu và phát triển một hệ thống lớn có thể cần thời gian. Trong thời gian đó, môi trường và mục tiêu nghiệp vụ nói chung đều thay đổi. Yêu cầu do vậy cũng phải thay đổi theo cho phù hợp. Vì thế, *tiến hóa yêu cầu* là một nhu cầu khách quan. Để thuận tiện cho việc tiến hóa, người ta chia yêu cầu thành hai loại:

1. *Yêu cầu lâu dài* (enduring requirements): Đây là những yêu cầu ổn định. Ví dụ, bệnh viện luôn có yêu cầu liên quan tới *bệnh nhân, bác sĩ, y tá, ...*
2. *Yêu cầu dễ thay đổi* (volatile requirements): Đây là những yêu cầu thường bị thay đổi trong quá trình phát triển hoặc sau khi đưa hệ thống vào hoạt động. Ví dụ, những yêu cầu phát sinh từ chính sách chăm sóc sức khỏe của Chính phủ trong hệ thống quản lý y tế,...

Sommeville [34] phân các yêu cầu này gồm bốn lớp (bảng 2.5).

Bảng 2.5. Phân loại các yêu cầu

Kiểu yêu cầu	Mô tả
Yêu cầu dễ thay đổi	Những yêu cầu phải thay đổi theo môi trường làm việc của tổ chức. Chẳng hạn, trong hệ thống bệnh viện, quỹ hỗ trợ chăm sóc bệnh nhân có thể thay đổi và do đó cần thu thập các thông tin điều trị khác nhau.
Yêu cầu phát sinh	Những yêu cầu xuất hiện cùng với hiểu biết của khách hàng về hệ thống trong quá trình phát triển. Khi thiết kế có thể gặp những yêu cầu mới.
Yêu cầu hệ quả	Việc sử dụng hệ thống máy tính có thể làm thay đổi cơ cấu tổ chức và mở ra phong cách làm việc mới, và chính những thay đổi về tổ chức và phong cách làm việc mới này lại làm nảy sinh những yêu cầu mới.
Yêu cầu tương thích	Những yêu cầu phụ thuộc vào từng hệ thống cụ thể. Khi hoạt động nghiệp vụ hay tổ chức thay đổi, yêu cầu cần phải thay đổi theo để tương thích với hệ thống mới này.

Nhiều yêu cầu phi chức năng thường bị ảnh hưởng do những thay đổi của công nghệ phần cứng. Mặc dù có thể dự báo được sự tiến bộ của phần cứng, việc đặc tả yêu cầu càng ít phụ thuộc vào phần cứng càng tốt.

2.2.4.2. Quản lý sự thay đổi yêu cầu

Bước đầu tiên trong tiến trình quản lý yêu cầu là lập kế hoạch. Trong giai đoạn này cần quyết định:

1. *Định danh các yêu cầu*: Mỗi yêu cầu phải có định danh duy nhất sao cho có thể tham chiếu đến nó từ các yêu cầu khác và có thể sử dụng nó trong việc đánh giá ảnh hưởng bằng cách lần vết.

2. *Định ra một tiến trình quản lý thay đổi*: Đó là một tập các hoạt động để đánh giá tác động và chi phí của các thay đổi. Các hoạt động này bao gồm:

- *Phân tích vấn đề đặt ra và đặc tả các thay đổi*: Những thay đổi được đề xuất cần được kiểm tra để đảm bảo sự đúng đắn và nhất quán.
- *Phân tích thay đổi và dự toán chi phí*: Hiệu ứng của các thay đổi đề xuất cần được đánh giá bằng cách sử dụng các thông tin lần vết và những kiến thức chung về yêu cầu. Chi phí để thay đổi phải bao gồm cả chi phí thay đổi tài liệu, làm thích ứng các thiết kế và triển khai.
- *Triển khai sự thay đổi*: Các tài liệu yêu cầu cần thay đổi, thiết kế hệ thống và triển khai hệ thống liên quan được cải biên.

Bảng 2.6. Ma trận lần vết giữa một số yêu cầu

Chỉ số của yêu cầu	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		U	R					
1.2			U			R		U
1.3	R			R				
2.1			U		U			U
2.2								U
2.3		R		U				
3.1								R
3.2							R	

Chú thích: R – reference: tham chiếu; U – use: sử dụng.

3. *Có một chính sách lần vết*: Chính sách này cho phép ghi nhận được các mối quan hệ giữa các yêu cầu, giữa các yêu cầu và thiết kế hệ thống và cách thức bảo trì thường xuyên chúng như thế nào. Ma trận lần vết liên kết yêu cầu với những người liên quan, với những yêu cầu

khác hay các môđun thiết kế thường được sử dụng. Bảng 2.6 là một ví dụ về ma trận lần vết.

4. *Sự trợ giúp của công cụ CASE*: Quản lý yêu cầu bao gồm việc xử lý một số lớn các thông tin về yêu cầu có mối quan hệ phức tạp. Các công cụ được sử dụng có thể bao gồm từ những hệ thống quản lý chuyên dụng đến bảng tính hay hệ thống cơ sở dữ liệu đơn giản.

2.3. Đặc tả yêu cầu

2.3.1. Khái niệm về đặc tả yêu cầu

Đặc tả yêu cầu là quá trình bổ sung các thông tin vào tài liệu xác định yêu cầu và là sự phát triển tiếp tục (làm mịn) các mô hình phân tích. Đặc tả yêu cầu thường được trình bày bằng các mô hình hệ thống. Nó rất quan trọng vì rằng chi phí cho việc sửa đổi các yêu cầu sẽ là rất cao, đặc biệt nếu các sai sót còn chưa được phát hiện trước khi hệ thống được bàn giao (hình 2.28). Vì vậy, việc tìm ra các phương pháp và công cụ đặc tả luôn được đặt ra và không ngừng hoàn thiện.

Ngôn ngữ tự nhiên thường được sử dụng để viết đặc tả yêu cầu. Tuy nhiên, nó không phải là một ngôn ngữ đặc tả tốt do các nguyên nhân sau:

- Không hoàn toàn xác định;
- Vốn quá mềm dẻo;
- Không thể phân chia yêu cầu một cách hiệu quả.

Do những hạn chế của nó, nhiều lựa chọn khác đã được nghiên cứu thay cho việc đặc tả bằng ngôn ngữ tự nhiên, đó là:

- Ngôn ngữ tự nhiên có cấu trúc;
- Ngôn ngữ mô tả chương trình;
- Ngôn ngữ đặc tả yêu cầu;
- Đặc tả dựa trên mô hình với ký pháp đồ họa;
- Đặc tả bằng toán học – hình thức hóa.

2.3.2. Các ngôn ngữ đặc tả yêu cầu

2.3.2.1. Ngôn ngữ tự nhiên có cấu trúc

Ngôn ngữ tự nhiên có cấu trúc là một dạng mở rộng của ngôn ngữ tự nhiên. Nó phụ thuộc vào *các định dạng* hoặc *mẫu chuẩn* được sử dụng để biểu diễn yêu cầu. Bảng quyết định của Moret (1992) là một dạng của ngôn ngữ có cấu trúc. Chúng là sự kết hợp các cấu trúc điều khiển thu được từ ngôn ngữ lập trình và các hình vẽ nổi bật để phân chia đặc tả. Lợi thế của cách tiếp cận này là nó giữ được phần lớn những đặc điểm dễ hiểu của ngôn ngữ tự nhiên, lại có cấu trúc rõ ràng, chính xác của ngôn ngữ lập trình.

2.3.2.2. Ngôn ngữ mô tả chương trình – PDL

Ngôn ngữ mô tả chương trình – PDL (Program Description Language) là tương tự với một ngôn ngữ lập trình nào đó nhưng trừu tượng hơn, có nhiều tính năng đặc tả hơn ngôn ngữ tự nhiên có cấu trúc. Ưu điểm của nó là có thể kiểm tra lỗi cú pháp và ngữ nghĩa bằng các công cụ phần mềm. Những mâu thuẫn và sai sót có thể được phát hiện nhờ việc kiểm tra này. Hình 2.15 là một ví dụ về đặc tả bằng PDL. Sử dụng PDL để đặc tả thích hợp trong hai trường hợp sau:

1. Khi thao tác được xác định là một chuỗi các hành động đơn giản và thứ tự thực hiện của chúng là quan trọng.
2. Khi cần phải đặc tả giao diện giữa phần cứng và phần mềm, giữa các hệ thống con trong hệ thống (kể cả với các hệ con đang tồn tại).

Người đọc có hiểu biết về ngôn ngữ lập trình có thể đọc hiểu các đặc tả yêu cầu bằng PDL một cách dễ dàng. Nếu PDL dựa trên một ngôn ngữ thực thi cụ thể thì quá trình chuyển đổi từ đặc tả sang thiết kế là rất tự nhiên và đơn giản.

2.3.2.3. Ngôn ngữ đặc tả yêu cầu chuyên dụng

Nhiều ngôn ngữ chuyên dụng khác nhau có chức năng đặc biệt được thiết kế để biểu diễn yêu cầu phần mềm. Ưu thế của cách tiếp cận này là nó được trợ giúp của CASE có các chức năng đặc biệt. Trong số đó phải kể đến các đại diện sau:

PSL/PSA (Problem Statement Language/ the Problem Statement Analyser – dùng để mô tả sản phẩm và cơ sở dữ liệu, và tạo báo cáo dựa trên bảng hỏi. *SADT (Structural Analysis – Design Technique)* là kỹ thuật phân tích cấu trúc đã mở rộng hơn nhiều so với các kỹ thuật đã có khác. Nó thành công cho nhiều loại sản phẩm, đặc biệt các dự án lớn, phức tạp và hệ thời gian thực. *SREM (Software Requirements Engineering Method)* dùng để đặc tả các điều kiện có hành động diễn ra đồng thời, đặc biệt là hệ thời gian thực và đã mở rộng cho các hệ thống phân tán.

```

- ATM/RS/CONT/1 control specification for an ATM
procedure ATM is
  PIN : Pin no;
  Acc_no:Account number;
  Balance: Amount;
  Service: Available servives;
  Valid card, Valid PIN Boolean;
  Begin
  Loop
    Get card(Acc no, PIN, Valid card);
    If valid_card then
      Validddate PIN(PIN Valid PIN)
      If valid PIN then
        Get account(Acc no balance);
        Get service(service);
        While service is selected loop
          Deliver selected service;
          Get service(Service);
        End loop;
        Return card;
      End if;
    End if ;
  End loop;
end ATM

```

Hình 2.15. Một mô tả các thao tác của ATM bằng PDL

2.3.3. Đặc tả yêu cầu dựa trên mô hình

Khi đặc tả mô hình theo hướng cấu trúc, đặc tả dữ liệu và đặc tả các hành vi (xử lý) của hệ thống là hai mô hình tách biệt nhau. Sau đây ta sẽ xét hai đặc tả đó.

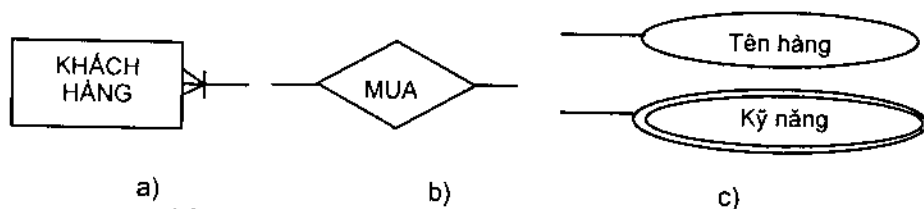
2.3.3.1. Đặc tả dữ liệu hướng cấu trúc – Mô hình thực thể mối quan hệ

Mô hình thực thể mối quan hệ (*Entity Relationships Model*) được Chen [9] đề xuất, sau được Chouinard (1989) bổ sung và mở rộng. Nó mô tả dữ liệu của thế giới thực đúng như chúng đang tồn tại. Nó cũng là phương tiện để giao tiếp với người sử dụng, giúp xác định và mô tả đúng đắn và đầy đủ yêu cầu thông tin của hệ thống. Mô hình được viết ngắn gọn là ERM.

a) Các thành phần chính của mô hình

Mô hình sử dụng một số khái niệm chính [37] sau:

- *Thực thể* (Entity) là khái niệm chỉ một lớp các đối tượng hay khái niệm (độc lập) của thế giới thực có những đặc trưng chung (hình 2.16a).
- *Mối quan hệ* (Relationship) là khái niệm chỉ mối quan hệ vốn có giữa các bản thể của các thực thể. Tên của nó là động từ chỉ sự tương tác hay mối quan hệ phụ thuộc, sở hữu giữa các bản thể của các thực thể (hình 2.16b).
- *Thuộc tính* (attribute) là đặc trưng của thực thể hay mối quan hệ. của thực thể (hình 2.16c).



Hình 2.16. Các ký pháp sử dụng trong ERM
a) Thực thể; b) Mối quan hệ; c) Thuộc tính

b) Xây dựng mô hình ERM

Đầu vào để xây dựng một ERM của một hệ thống là danh sách các hồ sơ và các nội dung của mô hình nghiệp vụ. Các bước xây dựng mô hình [38] bao gồm:

Bước 1: Xét từng hồ sơ: liệt kê, chính xác hóa và chọn lọc các mục tin trong các hồ sơ để được bảng danh mục các mục tin cơ sở (bảng 2.7).

Bước 2: Sử dụng bảng danh sách mục tin để tìm các thực thể và các thuộc tính của nó (bảng 2.8).

Bước 3: Xác định các mối quan hệ và thuộc tính của chúng (bảng 2.9).

Bước 4: Vẽ biểu đồ của mô hình (hình 2.17).

Bước 5: Rút gọn mô hình và xác định các bản số của thực thể.

c) Ví dụ về ERM

Trong bài toán trông gửi xe (mục 2.2.3.3c1), để đơn giản ta chỉ sử dụng các hồ sơ sau: (a) *bảng giá*, (b) *vé xe*, (c) *số xe vào*, (d) *số xe ra* và ta được bảng 2.7 là danh mục các mục tin (cuối bước 1), bảng 2.8 gồm các thực thể và thuộc tính của nó và bảng 2.9 cho các mối quan hệ liên kết các thực thể và có các thuộc tính riêng của nó. Thực hiện bước 4 và 5 ta được sơ đồ hình 2.17.

Bảng 2.7. Các mục tin cơ sở

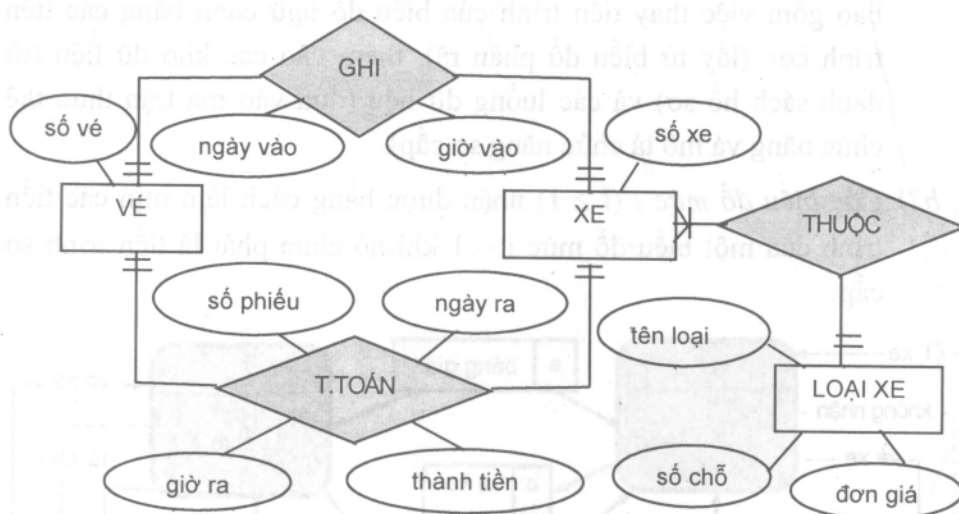
TT	Các mục tin	TT	Các mục tin	TT	Các mục tin
1	Số vé	5	Số phiếu TT	9	Tên loại xe
2	Số xe	6	Ngày ra	10	Số chỗ
3	Ngày vào	7	Giờ ra	11	Giá gửi loại xe
4	Giờ vào	8	Thành tiền		

Bảng 2.8. Các thực thể và thuộc tính

Thực thể	Thuộc tính	Thực thể	Thuộc tính
VÉ	<u>số vé</u>	LOẠI XE	<u>tên loại xe</u>
XE	<u>số xe</u>		số chỗ (trong bãi)
			đơn giá gửi

Bảng 2.9. Các mối quan hệ và thuộc tính của nó

Mối quan hệ	Thực thể tham gia và thuộc tính	Mối quan hệ	Thực thể tham gia và thuộc tính
1. GHI	VÉ, XE	2. THANH TOÁN	VÉ, XE
	ngày vào		số phiếu thanh toán
	giờ vào		ngày ra
			giờ ra
3. THUỘC	XE, LOẠI XE		thành tiền



Hình 2.17. Mô hình ERM của hệ thống trông gửi xe

2.3.3.2. Đặc tả xử lý hướng cấu trúc – Mô hình luồng dữ liệu

Mô hình luồng dữ liệu (DFD – Data Flow Diagram) là một kỹ thuật quan trọng đặc tả phần mềm trong những năm 80 của thế kỷ XX. Nó biểu diễn quá trình dữ liệu di chuyển và biến đổi trong hệ thống từ các vị trí đầu vào (nguồn) đến các vị trí đầu ra (đích), mô tả toàn bộ quá trình vận hành của hệ thống để thực hiện các chức năng hay dịch vụ mà nó đảm nhận. Mô hình được thể hiện ra là các biểu đồ luồng dữ liệu.

a) Mô hình luồng dữ liệu

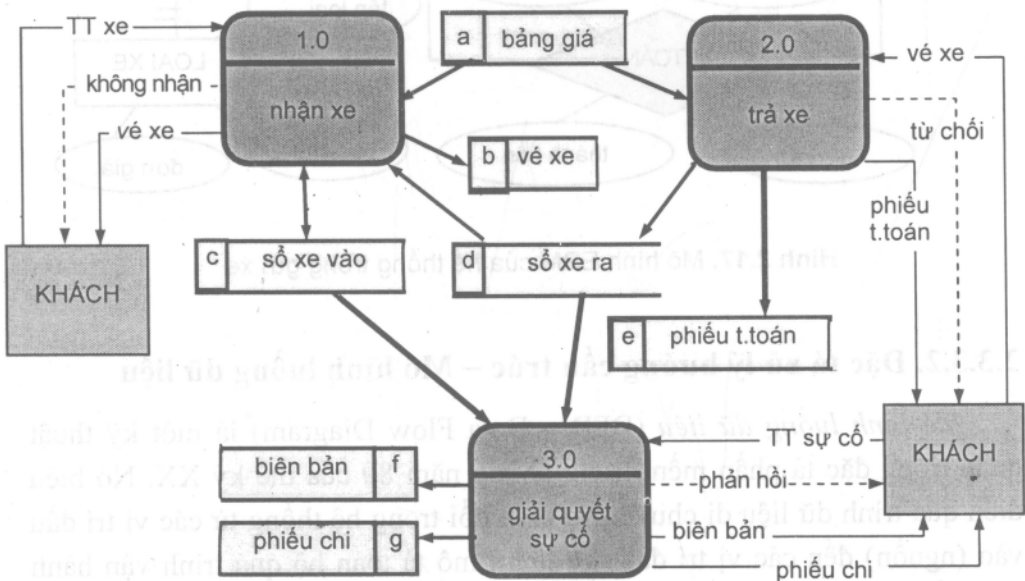
Trong mô hình sử dụng bốn khái niệm sau: *Luồng dữ liệu*, *tác nhân*, *kho dữ liệu*, *tiến trình*. Nội dung chi tiết của mô hình có thể xem ở bảng 1.1 chương 1.

b) Xây dựng đặc tả bằng biểu đồ luồng dữ liệu

Biểu đồ luồng dữ liệu đặc tả hệ thống gồm một số biểu đồ ở các mức khác nhau: mức 0, 1, 2,... Đầu vào của nó là *mô hình nghiệp vụ* (xem mục 2.2.3.3.c1)

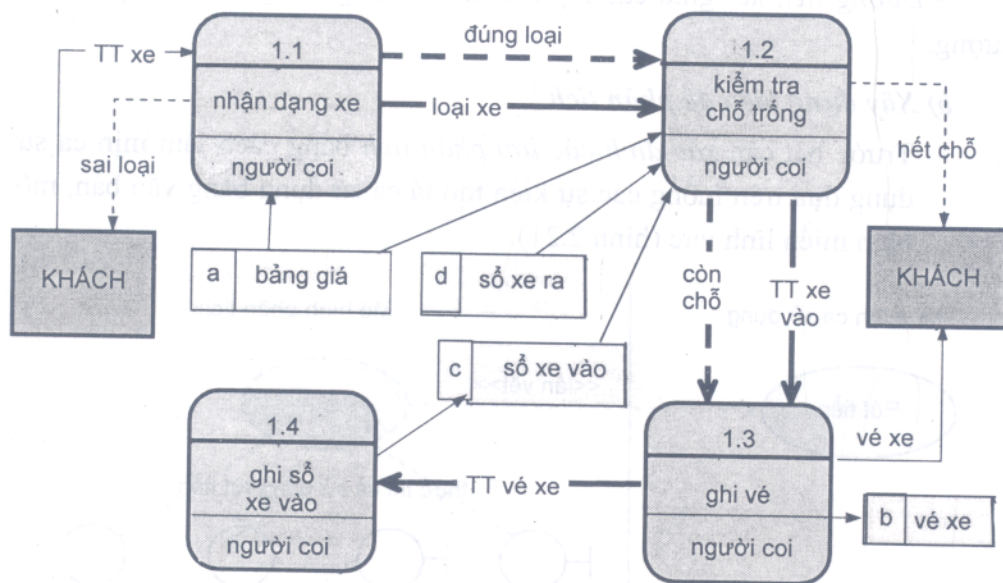
b1) *Biểu đồ mức 0* nhận được bằng cách làm mịn *biểu đồ ngữ cảnh* và sử dụng các thành phần khác của mô hình nghiệp vụ (hình 2.18): bao gồm việc thay tiến trình của biểu đồ ngữ cảnh bằng các tiến trình con (lấy từ biểu đồ phân rã), thêm vào các kho dữ liệu (từ danh sách hồ sơ) và các luồng dữ liệu (dựa vào ma trận thực thể chức năng và mô tả chức năng sơ cấp).

b2) *Các biểu đồ mức i ($i \geq 1$)* nhận được bằng cách làm mịn các tiến trình của một biểu đồ mức $i - 1$ khi nó chưa phải là tiến trình sơ cấp.



Hình 2.18. Biểu đồ luồng dữ liệu mức 0
(hình nét đậm là phần thay thế hay thêm vào)

Hình 2.18 là biểu đồ luồng dữ liệu mức 0 và hình 2.19 là một biểu đồ luồng dữ liệu mức 1 (nhận được khi làm mịn tiến trình 1.0 trong biểu đồ mức 0) của ví dụ hệ thống trông gửi xe đã nêu ở mục 2.2.3.3c1. Các biểu đồ luồng dữ liệu này là dạng vật lý: nó mô tả đúng như hoạt động của thể giới thực. Trong đó bao gồm các mô tả cả người và phương tiện cụ thể để thực hiện các hoạt động tương ứng của tiến trình trong điều kiện hiện hành.



Hình 2.19. Biểu đồ luồng dữ liệu mức 1 của tiến trình:
1.0 Nhận xe (nét đậm là phần được thay thế hay thêm vào)

2.3.3.3. Đặc tả hành vi hướng đối tượng – Biểu đồ phân tích tương tác

Biểu đồ đặc tả hành vi đối tượng nhận được từ việc phân tích sự thực thi từng ca sử dụng thuộc các gói ca sử dụng đã xét trong mô hình nghiệp vụ phân tích.

a) Các thành phần của biểu đồ phân tích

– Thực thi ca sử dụng phân tích là một mô tả về sự cộng tác giữa các phân tử trong mô hình để thực hiện chức năng của một ca sử dụng cụ thể và thể hiện ra dưới dạng các lớp phân tích và các đối tượng của nó tương tác với nhau. Các lớp phân tích có ba dạng cơ bản: lớp biên, lớp điều khiển và lớp thực thể [37] (hình 2.20).



Lớp biên



Lớp điều khiển



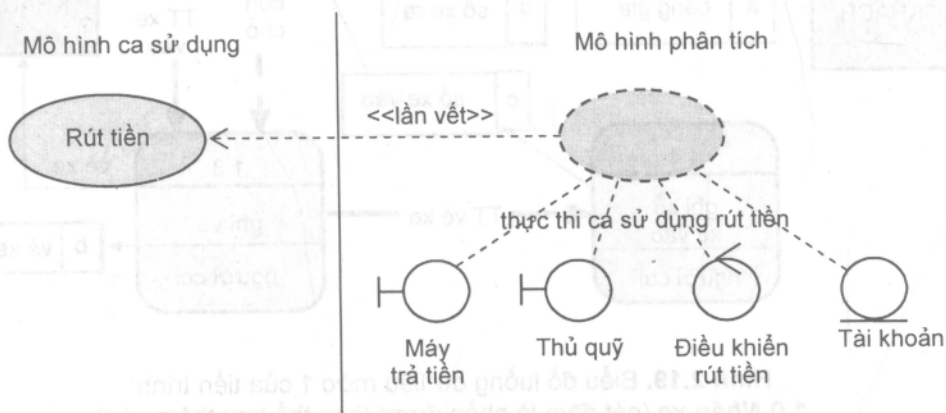
Lớp thực thể

Hình 2.20. Các lớp phân tích

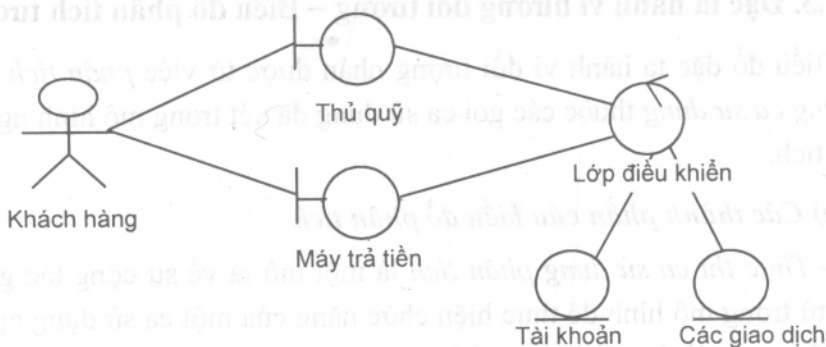
– Đường liên kết giữa các lớp mô tả sự tương tác giữa các lớp và đối tượng.

b) Xây dựng biểu đồ phân tích

- Trước hết cần xác định các lớp phân tích bằng cách làm mịn ca sử dụng dựa trên luồng các sự kiện mô tả ca sử dụng bằng văn bản, mô hình miền lĩnh vực (hình 2.21).



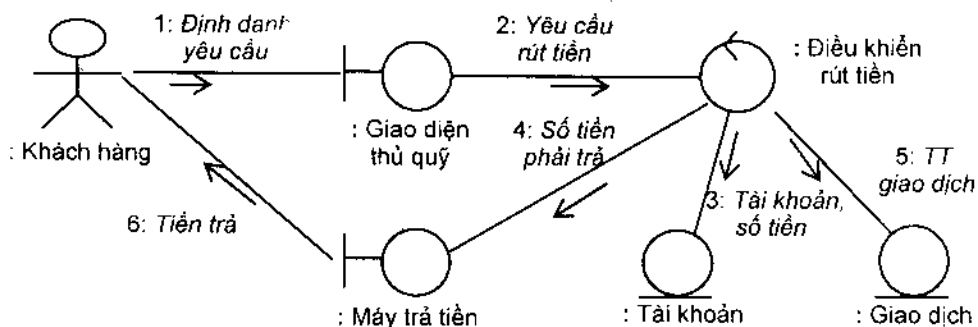
Hình 2.21. Các lớp phân tích tham gia vào thực thi ca sử dụng rút tiền



Hình 2.22. Quan hệ giữa các lớp phân tích và tác nhân của ca sử dụng

- Xác định mối liên kết giữa các lớp phân tích và tác nhân (hình 2.22).

- *Mô tả luồng các sự kiện phân tích.* Bổ sung thêm các mô tả văn bản giải thích và mũi tên vào biểu làm cho biểu đồ dễ hiểu hơn (hình 2.23).



Hình 2.23. Biểu đồ cộng tác giữa các lớp để thực thi ca sử dụng rút tiền

- *Phân tích để xác định các trách nhiệm của lớp trong sự cộng tác thực thi ca sử dụng.* Ví dụ, lớp *giao diện thủ quỹ* có các trách nhiệm:

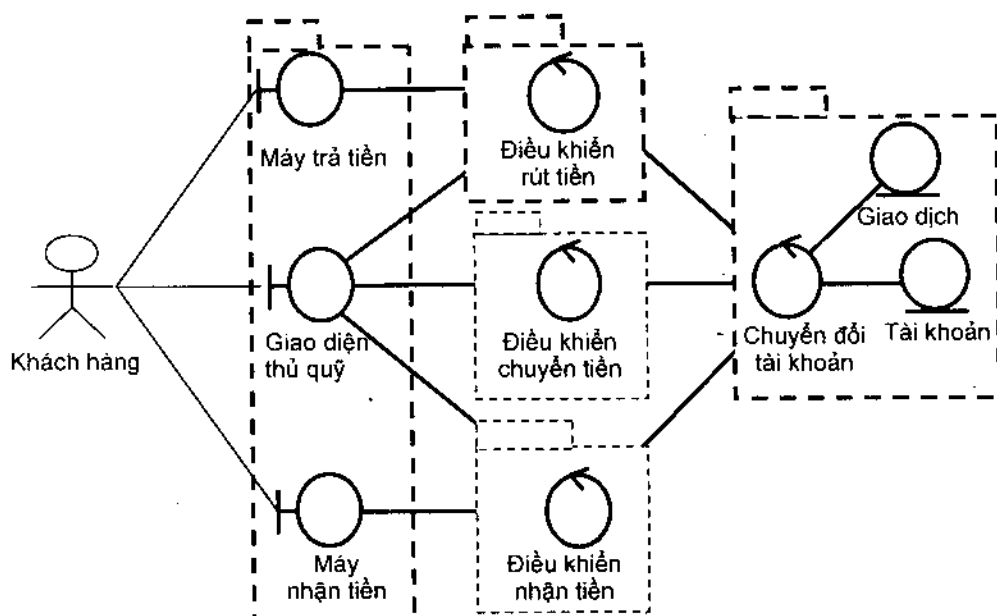
- + Hiển thị bảng tiếp nhận yêu cầu;
- + Tiếp nhận các yêu cầu của khách (hay từ lớp khác gửi đến);
- + Chuyển yêu cầu đến lớp điều khiển (hay đến một lớp khác).

Đây là các trách nhiệm điển hình của một giao diện người dùng để nhận ra.

- *Xác định các thuộc tính của các lớp.* Một thuộc tính đặc tả một tính chất của một lớp phân tích và nó thường được gợi ý và đòi hỏi từ các trách nhiệm của mỗi lớp.
- *Xác định các liên kết và các kết hợp sao cho hợp lý.* Số lượng các mối quan hệ giữa các lớp phải được tối thiểu.
- *Nắm bắt các yêu cầu đặc biệt của lớp phân tích.* Ví dụ: Các đặc trưng yêu cầu về tính lâu bền của lớp *tài khoản* có thể định chuẩn như sau:

- + *Quy mô:* 1 tới 4 KB cho mỗi đối tượng;
- + *Số lượng:* tối đa là 100.000 tài khoản.

- *Phân tích để nhóm các lớp thành các gói.* Mục đích của việc phân tích nhằm đảm bảo các gói phân tích càng độc lập đối với các gói khác càng tốt (cho ta đặc trưng cấu trúc hệ thống) (hình 2.24).



Hình 2.24. Mô hình lớp phân tích đặc tả hệ thống với các gói (đường đứt nét là hình ảnh của các gói)

2.3.4. Đặc tả yêu cầu hình thức hóa

2.3.4.1. Các phương pháp đặc tả hình thức hóa

Tiến trình kỹ nghệ phần mềm luôn bao gồm việc sử dụng các kỹ thuật toán học như một phần của tiến trình phát triển và thẩm định phần mềm, trong đó có các phương pháp đặc tả hình thức. Các phương pháp hình thức bao gồm đặc tả hệ thống hình thức, phân tích và chứng minh đặc tả, phát triển chuyển đổi và thẩm định chương trình. Tất cả chúng phụ thuộc vào phương pháp đặc tả nào được sử dụng. Một số các lĩnh vực toán học được vận dụng phải kể đến toán rời rạc, những khái niệm được rút ra từ lý thuyết tập hợp, logic và đại số. Một số phương pháp hình thức được biết đến nhiều như đặc tả đại số, máy trạng thái hữu hạn và nhiều kỹ thuật hình thức khác được đề nghị.

Một số phương pháp được biết đến nhiều hơn cả như: Phương pháp VDM là một kỹ thuật dựa trên ngữ nghĩa của ký hiệu. VDM không chỉ áp dụng cho đặc tả, mà còn cho cả thiết kế và triển khai. Ngôn ngữ giao tiếp các tiến trình tuần tự (CSP – Communicating Sequential Processes) được Hoare phát hiện ra dựa trên ý tưởng mô tả hành vi của hệ thống theo thuật

ngữ sự kiện. CSP cho phép đi từ đặc tả đến thiết kế và đến triển khai. *Lưới Petri* được Carl Adam Petri đề xuất năm 1962, nó thực sự có lợi cho việc đặc tả các hoạt động tương tác đồng thời. Phương pháp Z được Spivey đề xuất năm 1992, nó là một ngôn ngữ đặc tả hình thức được nhiều người biết đến. Trong phương pháp Z, ngoài các ký pháp thông dụng còn có nhiều ký pháp đặc biệt khác được sử dụng như: $\oplus$, $\triangleleft$, $\rightarrow$, $\leftarrow$, ... Đặc tả bằng phương pháp Z được biểu diễn bằng các mô tả hình thức, kèm theo mô tả văn bản không hình thức và tách thành những phần nhỏ khi sử dụng sơ đồ làm cho dễ đọc. Chúng là một cơ chế mạnh để đặc tả hệ thống.

Vào những năm 80 của thế kỷ XX, nhiều nghiên cứu đặc tả hình thức được xem như một tiến trình để hoàn thiện chất lượng các phần mềm, và người ta dự đoán rằng, thế kỷ XX, phần lớn phần mềm sẽ được phát triển bằng phương pháp hình thức. Nhưng thực tế đã không đúng như vậy, kỹ nghệ phần mềm đã thành công không nhờ vào các phương pháp hình thức. Sự thay đổi của yêu cầu thị trường đòi hỏi phải được đáp ứng nhanh. Người ta chấp nhận phần mềm có một số lỗi nhất định và áp dụng các phương pháp cho kết quả nhanh không phải là phương pháp hình thức.

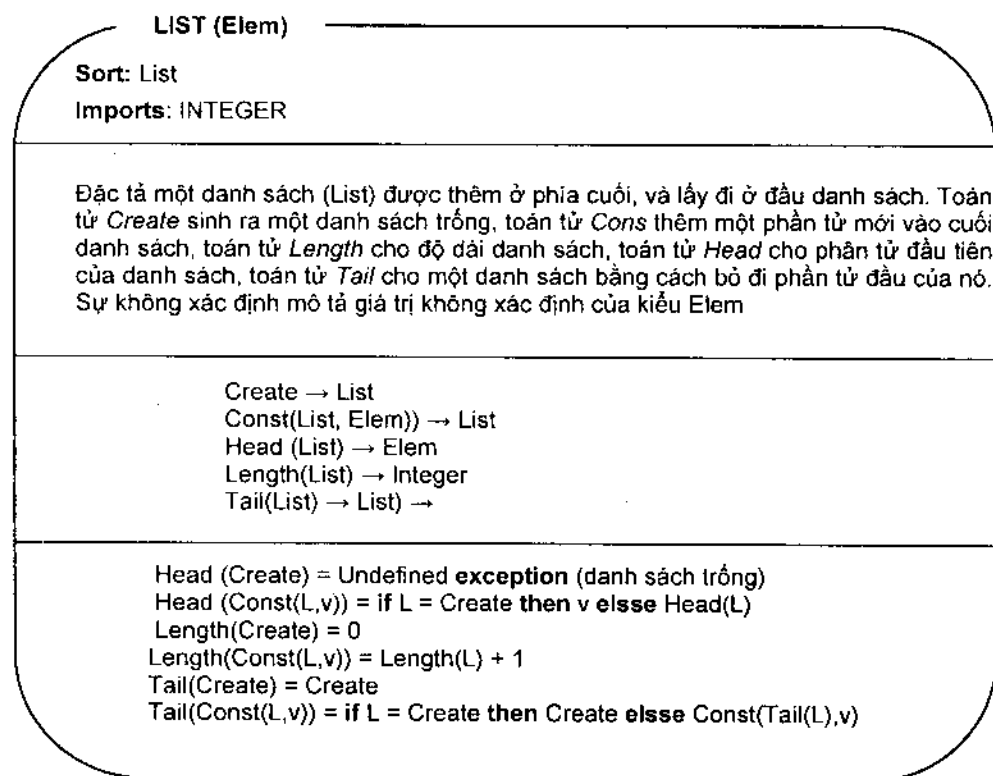
2.3.4.2. Đặc tả đại số

Các hệ lớn thường chia thành các hệ con được phát triển độc lập. Một hệ con có thể sử dụng dịch vụ của hệ khác, nên phần cơ bản của tiến trình đặc tả là giao diện giữa các hệ con. Đặc tả này nhằm giảm sinh lỗi giữa hệ thống con cung cấp dịch vụ và hệ con khác. *Đặc tả đại số* đã được sử dụng cho loại đặc tả này.

Đặc tả đại số gồm 4 phần:

- *Phần giới thiệu*: Gồm các đối tượng là các thực thể được đặc tả.
- *Phần mô tả*: Gồm các toán tử được mô tả không hình thức, làm cho đặc tả hình thức dễ hiểu hơn. Nó cung cấp cú pháp và ngữ nghĩa cho toán tử kiểu.
- *Phần chữ ký (signature)*: Xác định cú pháp của giao diện đối với lớp đối tượng hay kiểu dữ liệu trừu tượng.
- *Phần phương trình định nghĩa (axioms)*: Xác định ngữ nghĩa của toán tử bằng cách xác định một tập các phương trình định nghĩa đặc trưng hành vi của các kiểu dữ liệu trừu tượng.

Hình 2.25 minh họa một đặc tả đại số của một kiểu dữ liệu danh sách (LIST) đơn giản với năm toán tử: *Create()*, *Cons()*, *Head()*, *Tail()* và *Length()*.



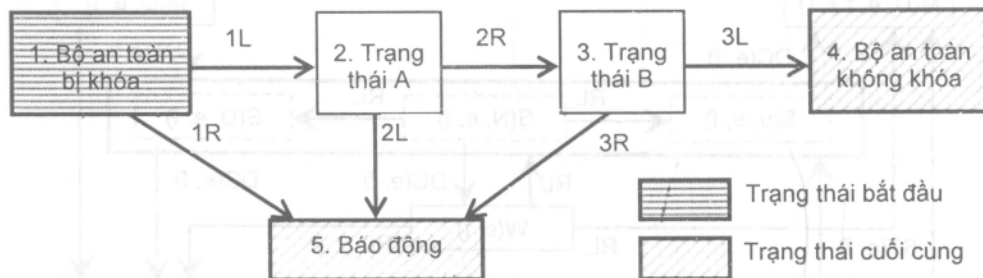
Hình 2.25. Cấu trúc của một đặc tả đại số

2.3.4.3. Máy trạng thái hữu hạn

a) Khái niệm về máy trạng thái hữu hạn

Xét một ví dụ do một nhóm tại một trường đại học của Anh đưa ra năm 1977 và được Kampen [16] mở rộng: Một bộ khóa tổ hợp an toàn với ba vị trí 1, 2 và 3. Luồng điều khiển ở mỗi vị trí có thể chuyển sang phải (R) hay sang trái (L). Ở tại một thời điểm bất kỳ, sự di chuyển có thể có sáu khả năng: 1R, 1L, 2R, 2L, 3R, 3L. Một tổ hợp các chuyển dịch an toàn là: (1L, 2R, 3L). Bất kỳ một tổ hợp chuyển dịch nào khác đều gây ra báo động hay kết thúc tiến trình. Tình huống trên đây được mô tả ở hình 2.26. Hình 2.26 là một biểu đồ chuyển trạng thái (state transition diagram – STD) của một

máy trạng thái hữu hạn (finite state machine – FSM). Tuy nhiên, ta cũng có thể thay nó bằng một bảng chuyển trạng thái.



Hình 2.26. Máy trạng thái hữu hạn biểu diễn bộ khóa an toàn tổ hợp

Một máy trạng thái tổng quát được xác định bởi một bộ gồm năm thành phần (J, K, T, S, F), trong đó:

- J – tập hữu hạn các trạng thái (states) khác trống $\{(1), (2), (3), (4), (5)\}$.
- K – tập hữu hạn các cái vào (inputs) khác trống $\{1R, 1L, 2R, 2L, 3R, 3L\}$.
- T – một hàm chuyển trạng thái (transition function) từ $(J \sim F) \times K$ vào J .

$S \in J$ – một trạng thái bắt đầu (initial) {bộ an toàn bị khóa}.

$F \subseteq J$ – tập các trạng thái cuối (final states) {báo động, bộ an toàn không khóa}.

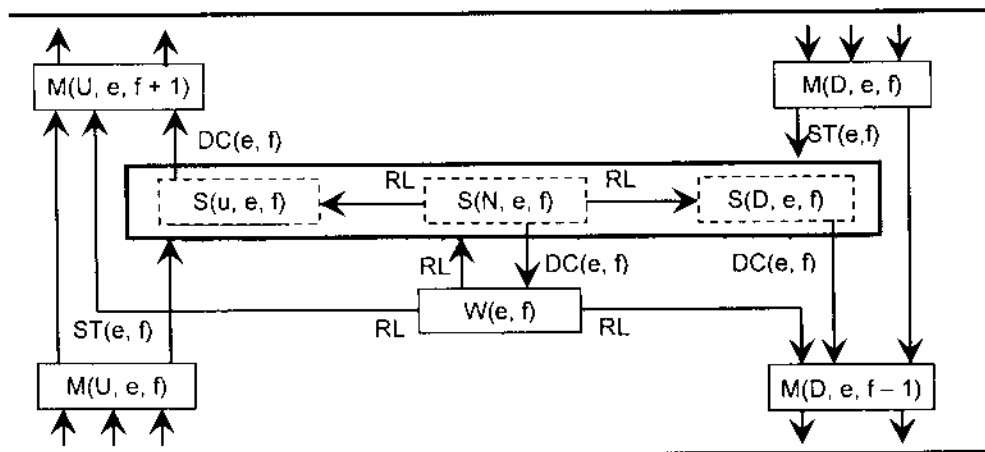
Cách tiếp cận máy trạng thái được sử dụng rất rộng rãi: Mỗi giao diện người dùng điều khiển bằng thực đơn là sự triển khai một máy trạng thái hữu hạn. Hiện một thực đơn là tương ứng với *một trạng thái*, và đưa vào một thông tin từ bàn phím hay chọn một biểu tượng bằng chuột là *một sự kiện* gây ra việc chuyển sang trạng thái khác. Trong trường hợp này mỗi sự chuyển trạng thái đều có dạng:

(trạng thái hiện thời) [thực đơn] và (sự kiện) [tùy chọn] $\Rightarrow$ trạng thái sau

FSM được mở rộng bằng cách thêm vào một thành phần thứ sáu gọi là *tập tiền định P*. Khi đó, hàm chuyển đổi T là hàm từ $(J \sim F) \times K \times P$ vào J . Quy tắc chuyển trạng thái bây giờ có dạng sau:

(trạng thái hiện thời) và (sự kiện) và (tiền định) $\Rightarrow$ (trạng thái sau)

Hình 2.27 là một ví dụ về đặc tả hệ thống n thang máy trong nhà với m tầng.



Hình 2.27. Sơ đồ chuyển trạng thái của thang máy

2.3.5. So sánh các loại hình đặc tả

Như trình bày ở trên, có rất nhiều phương pháp đặc tả khác nhau. Mỗi phương pháp đều có những mặt mạnh, mặt yếu của chúng. Bảng 2.10 đưa ra những so sánh chung nhất về các đặc trưng của mỗi loại phương pháp đặc tả.

Bảng 2.10. Bảng so sánh giữa các phương pháp đặc tả

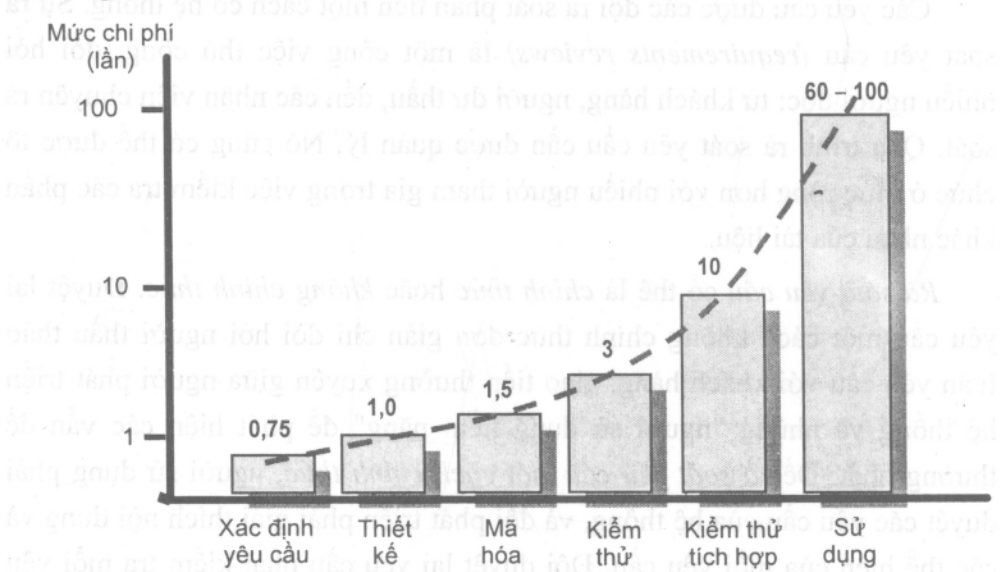
Phương pháp đặc tả	Loại hình	Mặt mạnh	Mặt yếu
Ngôn ngữ tự nhiên	Không hình thức hoá	Dễ học, dễ sử dụng, dễ hiểu đối với khách hàng.	Không chính xác, có thể mù mờ, mâu thuẫn, không đầy đủ.
- Mô hình thực thể mối quan hệ - PSL/DSA - SADT - SCREM - Phân tích HT có cấu trúc	Bán hình thức hoá	- Khách hàng có thể hiểu được. - Chính xác hơn phương pháp không hình thức. - Có thể tự động hóa một phần (nhờ CASE). - Cho kết quả nhanh.	Mức chính xác hóa chưa cao, không thể điều khiển được thời gian.
- Ann - CSP - Máy trạng thái hữu hạn - Gist - Lưới Petri - VDM - Z	Hình thức hóa	- Tuyệt đối chính xác, có thể làm giảm lỗi đáng kể. - Có thể giảm chi phí và công sức phát triển. - Có thể trợ giúp chứng minh tính đúng đắn. - Có thể tự động hóa cao (nhờ các CASE).	Khó học, khó sử dụng, khách hàng nói chung không thể hiểu được.

2.4. Thẩm định yêu cầu

2.4.1. Khái niệm về thẩm định yêu cầu

Thẩm định yêu cầu là xét xem đặc tả yêu cầu có thật sự xác định được hệ thống mà khách hàng mong muốn hay không. Nó khác với phân tích yêu cầu ở chỗ là đặc biệt chú trọng đến một số tính chất của đặc tả có ảnh hưởng nghiêm trọng đến hoạt động thiết kế và bảo trì hệ thống sau này.

Thẩm định yêu cầu là quan trọng, vì nếu việc thẩm định không tốt sẽ sót lỗi. Các lỗi còn sót lại từ yêu cầu sẽ lan truyền sang giai đoạn thiết kế và cài đặt hệ thống. Khi sửa đổi yêu cầu đều dẫn đến thay đổi thiết kế và cài đặt hệ thống. Do đó, chi phí của việc thay đổi yêu cầu hệ thống và sửa các lỗi sót lại hay mới nảy sinh như là sự tích tụ của nhiều bước sẽ không ngừng tăng lên và sẽ lớn gấp nhiều lần nếu chỉ được phát hiện sau khi đã cài đặt đưa vào sử dụng (hình 2.28).



Hình 2.28. Mức chi phí phải trả do sót lỗi qua các giai đoạn

Khi thẩm định, những mặt sau đây của yêu cầu phải được kiểm tra:

- *Tính đúng đắn (Validity):* Hệ thống cần bao quát được các nhu cầu khác nhau của người dùng và yêu cầu phải đáp ứng được cả cộng đồng người sử dụng.

- *Tính đầy đủ (Completeness)*: Phải xác định được mọi chức năng và ràng buộc mà người sử dụng hệ thống mong muốn.
- *Tính nhất quán (Consistency)*: Mọi yêu cầu đều không được phép xung đột với bất kỳ một yêu cầu nào khác.
- *Tính hiện thực (Realism)*: Không có một điểm nào trong đặc tả yêu cầu là không thể thực hiện được.
- *Tính kiểm tra được*: Yêu cầu phải viết sao cho có thể kiểm tra được. Điều đó có nghĩa là có một tập các kiểm tra được thiết kế để có thể chỉ ra rằng sản phẩm gửi cho khách là đáp ứng các yêu cầu đó.

2.4.2. Các kỹ thuật thẩm định yêu cầu

2.4.2.1. Rà soát yêu cầu

Các yêu cầu được các đội rà soát phân tích một cách có hệ thống. Sự rà soát yêu cầu (*requirements reviews*) là một công việc thủ công, đòi hỏi nhiều người đọc: từ khách hàng, người dự thầu, đến các nhân viên chuyên rà soát. Quá trình rà soát yêu cầu cần được quản lý. Nó cũng có thể được tổ chức ở mức rộng hơn với nhiều người tham gia trong việc kiểm tra các phần khác nhau của tài liệu.

Rà soát yêu cầu có thể là *chính thức* hoặc *không chính thức*. Duyệt lại yêu cầu một cách không chính thức đơn giản chỉ đòi hỏi người thảo luận yêu cầu với khách hàng, giao tiếp thường xuyên giữa người phát triển hệ thống và những "người sử dụng tiềm năng" để phát hiện các vấn đề thường nhật. Để *rà soát yêu cầu một cách chính thức*, người sử dụng phải duyệt các yêu cầu của hệ thống, và đội phát triển phải giải thích nội dung và các thể hiện của mỗi yêu cầu. Đội duyệt lại yêu cầu phải kiểm tra mỗi yêu cầu về tính nhất quán và tính đầy đủ khi xem xét chúng như một thể thống nhất. Họ cũng phải kiểm tra về:

- *Tính kiểm tra được (Verifiability)*: Các yêu cầu đặt ra có thực sự kiểm tra được hay không?
- *Tính hiểu được (Comprehensibility)*: Người sử dụng cuối cùng của hệ thống có thể hiểu được yêu cầu không?

- *Tính lần vết được (Traceability)*: Nguồn gốc của yêu cầu có được phát biểu rõ ràng không? Khi cần thiết có thể phải quay lại khởi điểm của yêu cầu để đánh giá ảnh hưởng của các yếu tố liên quan không?
- *Tính thích nghi được (Adaptability)*: Yêu cầu có làm thích nghi được không, nghĩa là có thể thay đổi mà không gây ảnh hưởng lớn đến các yêu cầu khác.

Xung đột, mâu thuẫn, lỗi và thiếu sót trong yêu cầu cần được phát hiện và ghi nhận trong quá trình rà soát. Sau đó người dùng, người trung gian và người phát triển hệ thống cần thỏa thuận giải pháp cho các vấn đề được phát hiện ra.

2.4.2.2. Làm bản mẫu

Làm bản mẫu là một kỹ thuật thẩm định yêu cầu rất quan trọng; nhờ nó mà mô hình hệ thống thực hiện được có thể trình diễn cho người sử dụng. Bản mẫu giúp đào tạo người dùng sử dụng hệ thống. Họ có thể thấy được hệ thống có thật sự trợ giúp công việc của họ hay không. Việc xem xét lại yêu cầu một cách đúng thủ tục đòi hỏi sự tham gia của cả người sử dụng và kỹ sư phần mềm trong quá trình chuẩn bị và trình bày tài liệu yêu cầu với sự trợ giúp của bản mẫu.

2.4.2.3. Tạo sinh các ca kiểm thử

Một cách lý tưởng thì mọi yêu cầu là có thể kiểm thử được. Nếu các ca kiểm thử được thực hiện như một phần của công việc thẩm định, thì rất dễ dàng phát hiện các vấn đề của yêu cầu. Nếu các ca kiểm thử rất khó khăn hay không thể kiểm thử được, điều đó có nghĩa là các yêu cầu khó triển khai và có thể phải xem xét lại.

2.4.2.4. Phân tích tính nhất quán được tự động

Nếu các yêu cầu được biểu diễn ở dạng các mô hình với các ký pháp cấu trúc hay hình thức hóa thì các công cụ CASE có thể được sử dụng để kiểm tra tính nhất quán của mô hình. Điều này rất có ý nghĩa khi đặc tả các hệ thống lớn.

2.5. Đặc tả hệ thống và làm bản mẫu

2.5.1. Tiến trình làm bản mẫu phần mềm

Duyệt lại yêu cầu có phê phán chính là nhiệm vụ của quá trình thẩm định yêu cầu. Tuy nhiên, nhiều người sử dụng tiềm năng rất khó hình dung được hệ thống sẽ được sử dụng như thế nào nếu chỉ đọc các bản đặc tả yêu cầu. Một chức năng đặc tả có thể xem là được xác định tốt, nhưng khi sử dụng thực tế, nó được kết hợp với các chức năng khác lại để lộ ra cái nhìn ban đầu của người sử dụng đã không đúng và không đầy đủ. Một cách để giải quyết khó khăn này là phát triển một *bản mẫu hệ thống* cho phép người sử dụng thử nghiệm trên nó.

Việc phát triển bản mẫu thực hiện ngay từ giai đoạn sớm của tiến trình phần mềm sẽ đem lại những lợi ích rất thiết thực:

1. Khắc phục *sự hiểu lầm* giữa những người phát triển phần mềm và những người sử dụng phần mềm.
2. Phát hiện sự thiếu hụt các dịch vụ người dùng.
3. Nhận ra sự khó sử dụng hoặc sự nhầm lẫn các dịch vụ người dùng.
4. Đội ngũ phát triển phần mềm có thể tìm ra được các yêu cầu không đầy đủ hoặc không nhất quán ngay khi phát triển bản mẫu.
5. Giúp thuyết minh cho tính khả thi và tính hữu ích của ứng dụng đối với các nhà quản lý.
6. Bản mẫu được dùng làm cơ sở cho việc viết đặc tả của sản phẩm.

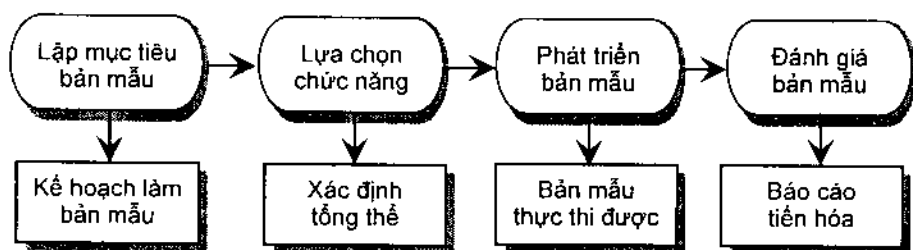
Ngoài ra, một bản mẫu phần mềm cũng có thể sử dụng cho mục tiêu khác:

1. Dùng để *huấn luyện người sử dụng* trước khi hệ thống được phân phối.
2. Dùng *trong quá trình thử nghiệm hệ thống* để chạy back – to – back. Điều này có nghĩa là cùng các thử nghiệm như nhau vừa dùng để thử bản mẫu vừa dùng để thử hệ thống. Hai kết quả này khác nhau, nghĩa là có sai sót.

Tiến trình phát triển bản mẫu phần mềm gồm bốn bước (hình 2.29):

1. *Thiết lập các mục tiêu* của việc tạo bản mẫu.

2. Chọn các chức năng cho việc tạo bản mẫu và quyết định những đặc tả phi chức năng nào cần phải tạo bản mẫu.
3. Phát triển bản mẫu.
4. Đánh giá bản mẫu.



Hình 2.29. Tiến trình phát triển bản mẫu

Đầu tiên phải làm rõ các mục tiêu của bản mẫu. *Mục tiêu* đó có thể là phát triển một hệ thống chỉ liên quan đến *tạo bản mẫu giao diện người sử dụng*; cũng có thể là *để thẩm định các yêu cầu* hệ thống chức năng hoặc *để trình diễn tính khả thi* của ứng dụng cho các nhà quản lý xem xét... Nếu mục tiêu còn chưa rõ ràng thì người quản lý hoặc người sử dụng có thể hiểu nhầm chức năng khi thử bản mẫu và như vậy không thể đạt được kết quả mong muốn.

Tiếp theo là quyết định xem cái gì được đưa vào và cái gì được lấy ra từ bản mẫu đó. Bản mẫu là đắt nếu nó được thực hiện như cách làm cho chính hệ thống cuối cùng. Bởi vậy, có thể tạo bản mẫu với tất cả các chức năng của hệ thống, nhưng ở mức thu gọn. Cũng có thể chỉ một số chức năng hệ thống được tạo mẫu; thường người ta hay bỏ bớt các yêu cầu phi chức năng và các chuẩn chất lượng.

Giai đoạn cuối cùng là đánh giá bản mẫu. Cần huấn luyện người sử dụng từ các mục tiêu của việc tạo mẫu và đưa ra một kế hoạch đánh giá. Phải có đủ thời gian cho người sử dụng làm quen với hệ thống và thiết lập một cách sử dụng mẫu chuẩn tắc, nhờ vậy mà phát hiện được các sai sót của yêu cầu.

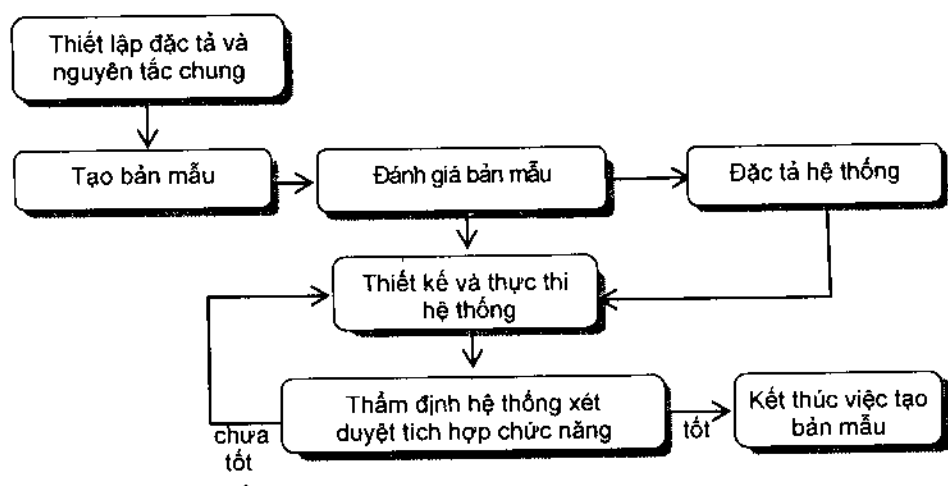
2.5.2. Tạo bản mẫu trong tiến trình phần mềm

Mô hình tiến trình phần mềm dựa trên tạo bản mẫu được mô tả trong hình 2.30. Việc sử dụng nó có thể hướng đến những mục tiêu khác nhau:

- *Làm bản mẫu là sự mở rộng quá trình phân tích yêu cầu:* Mỗi bản mẫu chỉ nhằm làm sáng tỏ các yêu cầu, cung cấp các thông tin phụ cho người quản lý để đánh giá các rủi ro của quá trình. Sau khi đánh giá thì bỏ bản mẫu đi.
- *Làm bản mẫu là một phần hệ thống:* Một bộ phận của hệ thống được đặc tả, rồi được phân phối và cải biên cho tới khi khách hàng thỏa mãn với các chức năng của nó. Tại đây một mô hình tiến trình phần mềm quy ước ra đời và bản đặc tả dẫn xuất từ bản mẫu được thực hiện để thành một sản phẩm.
- *Làm bản mẫu là tạo ra một hệ thống:* Thay cho việc tạo ra đặc tả bản mẫu, người ta đề xuất phát triển chính hệ thống. Chỉ dẫn này cho người ký hợp đồng thấy rằng, chi đơn giản là *hãy viết một hệ thống kiểu như thế này*.

Có một số cách tiếp cận làm bản mẫu sau đây:

1. *Cách tiếp cận lập trình thăm dò* đã được sử dụng để phát triển hệ thống: Trước hết trình bày cho người dùng một hệ thống chưa đầy đủ rồi cải tiến, tăng cường nó cho đến khi mà các yêu cầu thực của người dùng được thỏa mãn. Khi đó bản mẫu bị thải loại và một hệ thống chất lượng được xây dựng.



Hình 2.30. Tiến trình tạo bản mẫu khi đặc tả yêu cầu

2. *Cách tiếp phát triển gia tăng.* Nó bao hàm việc *phát triển yêu cầu* và *việc phân phối hệ thống được thực hiện ngay sau đó*. Khi một phần của hệ thống được phân phối, người dùng có thể thực nghiệm và phản hồi trở lại. Cách này khắc phục được các vấn đề thay đổi liên tục của lập trình thăm dò. Kiến trúc hệ thống tổng thể được thiết lập sớm như một khung. Một khi chúng đã được thẩm định và phân phối thì khung đó xem như "cố định" cho đến khi phát hiện ra sai sót. Tuy nhiên, thông tin phản hồi từ phía người dùng về các thành phần đã được phân phối có thể ảnh hưởng đến thiết kế các thành phần được lập trong kế hoạch phân phối tiếp sau.
3. *Phát triển tiến hóa* là dễ quản lý hơn lập trình thăm dò vì rằng ở đây các chuẩn tiến trình phần mềm luôn được tuân thủ. Các kế hoạch và các tư liệu có thể được xuất ra cho mỗi một phần gia tăng. Nó cho phép một vài thông tin phản hồi từ phía người dùng đến sớm hơn trong tiến trình giúp hạn chế các sai sót ở từng phần khi mà đội ngũ phát triển không quan tâm. Mỗi khi một phần gia tăng được phân phối thì giao diện của nó cố định lại. Những phần gia tăng sau phải thích nghi với các giao diện đó và có thể được thử lại.

2.5.3. Các kỹ thuật tạo bản mẫu

Các kỹ thuật tạo bản mẫu hệ thống cho phép phát triển nhanh một bản mẫu hệ thống. Vì giá nhân công là giá cả chính của một phần mềm cho nên phát triển nhanh có nghĩa là giá bản mẫu là tối thiểu. Nó cũng có nghĩa là thông tin ngược từ phía người dùng có thể nhận được sớm trong tiến trình phần mềm để giảm rủi ro. Một số kỹ thuật được sử dụng để tạo bản mẫu hệ thống là:

- Sử dụng ngôn ngữ đặc tả thi hành được.

Sử dụng ngôn ngữ bậc rất cao.

Sử dụng ngôn ngữ bậc cao hướng ứng dụng (ngôn ngữ thể hệ thứ tư).

Lắp ghép từ các thành phần dùng lại được."

2.5.3.1. Ngôn ngữ đặc tả thi hành được

Ưu điểm của phương pháp phát triển này là giá của việc sản sinh ra một đặc tả là hầu như không đáng kể. Nhưng nó lại là khó hiểu đối với người dùng và đặc tả cần có kiến thức và kỹ năng toán học. Có ba vấn đề đối với phát triển bản mẫu theo cách tiếp cận này là:

- Không thể dùng kỹ thuật này để tạo bản mẫu cho giao diện đồ họa.
- Phát triển bản mẫu không đặc biệt nhanh.
- Hệ thi hành được đó thường hoạt động chậm chạp và không hiệu quả.

Hai vấn đề đầu có thể được giải quyết bằng cách dùng ngôn ngữ lập trình chức năng để phát triển hệ thống. Ngôn ngữ lập trình chức năng là một ngôn ngữ hình thức dựa trên việc biểu thị hệ thống như là một hàm toán học.

2.5.3.2. Các ngôn ngữ bậc rất cao

Các ngôn ngữ bậc rất cao là các ngôn ngữ lập trình thừa kế các cấu trúc dữ liệu động và các đặc điểm bậc cao như khả năng quay lui. Ví dụ: LISP, PROLOG, Smalltalk, APL, SETL. Chúng là các ngôn ngữ tạo bản mẫu có ích vì có thể phát triển hệ thống nhanh. Các ngôn ngữ bậc rất cao không được dùng rộng rãi vì chúng cần một hệ thống trợ giúp vận hành lớn. Hệ thống này đòi hỏi bộ nhớ lớn và làm giảm tốc độ thi hành của chương trình viết bằng ngôn ngữ đó.

2.5.3.3. Các ngôn ngữ thế hệ thứ tư

Trong một số lớp các ứng dụng, đặc biệt là trong bộ xử lý dữ liệu thương mại, người ta dùng cách tiếp cận tạo bản mẫu để phát triển hệ thống. Một ngôn ngữ thế hệ thứ tư (*fourth generation language – 4GL*) dựa trên việc sử dụng lại phần mềm, ở đó các chương trình con được cung cấp để truy cập một dữ liệu và để sản ra các báo cáo và người sử dụng chỉ cần mô tả làm thế nào để điều khiển các mô đun đó.

2.5.3.4. Lắp ghép từ các thành phần dùng lại được

Có lẽ ví dụ tốt nhất cho cách tiếp cận này để tạo bản mẫu là các ngôn ngữ lập trình vỏ (*shell script*) đã có sẵn ở UNIX. Người ta lắp ráp bản mẫu từ các thành phần có sẵn thành hệ thống có thể vận hành được. Khi liên kết

các môđun làm mẫu nhận được ta có một hệ thống ban đầu với những chức năng nhất định có thể đáp ứng những dịch vụ đặt ra.

Tổng kết

- ◆ Yêu cầu cho một hệ thống phần mềm là mô tả những gì hệ thống cần làm (chức năng hay dịch vụ) và những ràng buộc lên sự hoạt động và triển khai nó.
- ◆ Yêu cầu chia làm ba loại: *yêu cầu chức năng*, *phi chức năng* và *yêu cầu miễn lĩnh vực*. Yêu cầu chức năng là phát biểu về các dịch vụ mà hệ thống cung cấp hay các mô tả chức năng nó cần làm. Yêu cầu miễn đặc trưng cho những đòi hỏi của miền ứng dụng. Các yêu cầu phi chức năng có thể là *yêu cầu về sản phẩm*, *về tiến trình phát triển* hay *từ môi trường bên ngoài*. Chúng thường liên quan đến các thuộc tính nổi trội của hệ thống nên áp dụng cho toàn hệ thống.
- ◆ Tiến trình kỹ nghệ yêu cầu bao gồm: nghiên cứu khả thi, phát hiện và phân tích yêu cầu, đặc tả yêu cầu, thẩm định yêu cầu và tiến hóa yêu cầu. Phân tích yêu cầu là một quá trình lặp, gồm việc hiểu miền ứng dụng, lựa chọn yêu cầu, phân loại, cấu trúc yêu cầu, sắp xếp ưu tiên và thẩm định.
- ◆ Phát hiện và xác định yêu cầu là việc khó, thường phải áp dụng nhiều cách tiếp cận khác nhau: theo *nguồn/dịch dữ liệu*, theo *khung trình diễn*, theo *tiếp nhận dịch vụ*. Với mỗi cách tiếp cận có nhiều phương pháp được phát triển và sử dụng. Một số phương pháp điển hình như *VORD*, hướng cấu trúc có *mô hình luồng dữ liệu* và *mô hình dữ liệu thực thể mối quan hệ*; hướng đối tượng có *mô hình miền*, *mô hình ca sử dụng các mức* và *mô hình lớp phân tích tương tác*.
- ◆ Có nhiều phương pháp và ngôn ngữ khác nhau được dùng để đặc tả yêu cầu, điển hình là ngôn ngữ tự nhiên, ngôn ngữ mô tả chương trình, ngôn ngữ dựa trên mô hình, ngôn ngữ đặc tả yêu cầu chuyên dụng (PSL/PSA, SADT,...), ngôn ngữ hình thức hóa (đặc tả đại số, máy trạng thái hữu hạn, VDM, lưới Petri,...).
- ◆ Yêu cầu phần mềm cần để giao tiếp giữa nhiều đối tượng: *người sử dụng*, *khách hàng*, *người thầu* và *những người liên quan*, *nhà phát*

triển. Vì vậy nó cần được viết và thể hiện ở nhiều mức độ trừu tượng thích hợp cho mỗi đối tượng. Đặc tả yêu cầu phần mềm cần phải đầy đủ, nhất quán. Nó cần phải được tổ chức và viết sao cho dễ đọc, dễ theo dõi, sẽ sửa đổi trong quá trình phát triển và cả bảo trì sau này. Có một số công cụ lưu trữ và tìm kiếm trợ giúp hoạt động này.

- ♦ *Thẩm định yêu cầu* là kiểm tra xem các yêu cầu có đúng đắn, nhất quán, có đầy đủ, có thể kiểm tra được, có thể thực hiện được không? Có một số phương pháp thẩm định: rà soát, tạo ca thử nghiệm phân tích sử dụng CASE, làm mẫu.
- ♦ *Quản lý yêu cầu* là tiến trình quản lý và kiểm soát sự thay đổi của các yêu cầu do môi trường nghiệp vụ, tổ chức và kỹ thuật tác động. Nó bao gồm việc xác định và phân tích các thay đổi có thể, thiết lập các chính sách và lập kế hoạch để quản lý chúng.

Câu hỏi ôn tập

1. Yêu cầu phần mềm là gì? Phân biệt giữa yêu cầu và nhu cầu. Có thể phân yêu cầu theo những đối tượng đọc nào và đặc trưng của mỗi loại đọc đó?
2. Phân loại yêu cầu theo tính chất có những loại nào? Giải thích nội dung mỗi loại. Yêu cầu phi chức năng có thể phân cấp như thế nào? Cho ví dụ về mỗi loại.
3. Trình bày tiến trình kỹ nghệ yêu cầu. Nội dung của nghiên cứu khả thi là gì? Đề cương của tài liệu nghiên cứu khả thi gồm những mục gì?
4. Đối tượng chính của tài liệu yêu cầu là ai? Những yêu cầu gì đặt ra để viết bản tài liệu yêu cầu? Vì sao?
5. 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.
6. Tại sao phải thay đổi và tiến hóa yêu cầu? Làm thế nào để quản lý sự thay đổi của yêu cầu? Yêu cầu cần được tổ chức như thế nào để có thể lần vết được?
7. Cấu trúc tài liệu yêu cầu gồm những mục gì? Phụ lục cần thêm nội dung gì?

8. Trình bày các bước của kỹ thuật xác định yêu cầu theo cách nhìn VORD.
9. Trình bày phương pháp phân tích yêu cầu theo hướng chức năng. Theo hướng đối tượng.
10. Tại sao phải tiến hóa yêu cầu? Để phục vụ tiến hóa cần phân yêu cầu thành những loại nào? Tại sao?
11. Làm thế nào để quản lý được yêu cầu? Tổ chức yêu cầu như thế nào? Tiến trình và phương pháp quản lý như thế nào?
12. Đặc tả yêu cầu là gì? Cần phải đặc tả những nội dung gì của phần mềm? Những kỹ thuật gì được sử dụng để đặc tả yêu cầu?
13. Trình bày những hạn chế của đặc tả bằng ngôn ngữ tự nhiên. Ngôn ngữ tự nhiên có cấu trúc là gì? Ưu và nhược điểm của nó.
14. Đặc tả bằng ngôn ngữ mô tả chương trình là gì? Ưu điểm của nó.
15. Đặc tả dựa trên mô hình là gì? Có những cách tiếp cận nào?
16. Trình bày đặc tả theo phương pháp cấu trúc. Theo phương pháp hướng đối tượng.
17. Đặc tả hình thức là gì? Nêu một số phương pháp đặc tả hình thức. Lợi ích và những hạn chế của đặc tả hình thức.
18. Trình bày tóm tắt phương pháp đặc tả đại số và máy trạng thái hữu hạn.
19. Thẩm định yêu cầu là gì? Cần phải thẩm định những nội dung gì của yêu cầu? Có những kỹ thuật gì để thẩm định yêu cầu? Trình bày tóm tắt nội dung của nó.
20. Trình bày tiến trình làm bản mẫu phần mềm. Lợi ích của làm bản mẫu phần mềm trong xác định yêu cầu. Bản mẫu phần mềm còn có thể dùng làm gì?
21. Có những cách tiếp cận làm mẫu nào được áp dụng trong tiến trình phần mềm? Chúng có những ưu, nhược điểm gì?
22. Có những kỹ thuật nào được sử dụng cho việc làm mẫu phần mềm? Trình bày tóm tắt mỗi loại.

Chương 3

THIẾT KẾ PHẦN MỀM

3.1. Tổng quan về thiết kế phần mềm

3.1.1. Khái niệm và vai trò của thiết kế

3.1.1.1. Khái niệm về 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 ánh xạ nó thành chương trình vận hành được. Trong trường hợp chung, các yêu cầu phần mềm thường được mô tả ở mức quan niệm, tức là mô tả các dịch vụ mà phần mềm cung cấp cũng như các ràng buộc nó cần tuân thủ khi hoạt động. Nhiệm vụ của người kỹ sư là chuyển những yêu cầu này thành một mô tả về hệ thống phần mềm với các giải pháp công nghệ thích hợp, sẽ vận hành trên thực tế, nhằm đáp ứng các yêu cầu đặt ra. Để làm được điều đó người kỹ sư thiết kế cần đến một số hoạt động chính sau:

- 1. Nghiên cứu để hiểu vấn đề. Vấn đề ở đây không chỉ liên quan đến yêu cầu đặt ra mà còn liên quan đến môi trường đích, mà ở đó hệ thống được cài đặt và vận hành.*
- 2. Chọn một số giải pháp thiết kế và xác định các đặc điểm thô của nó. Các giải pháp được chọn cần khả thi và cho hiệu quả cao đối với hệ thống.*
- 3. Mô tả trừu tượng cho mỗi giải pháp thiết kế. Trước hết cần xây dựng một mô tả ban đầu sơ khai, rồi chi tiết hóa dần. Các sai sót ở mỗi mức thiết kế trước đó được phát hiện và chỉnh sửa trước khi lập tự liệu thiết kế chính thức.*

Quá trình thiết kế nhằm tạo ra một mô hình sản phẩm mới bằng việc tìm tòi và vận dụng các giải pháp công nghệ khác nhau. Hoạt động thiết kế vì thế là một loại *hoạt động đặc biệt*: một quá trình sáng tạo, đòi hỏi kinh nghiệm và sự nhanh nhạy của người thiết kế. Kinh nghiệm cho thấy, nó phải được thực hành, được học bằng kinh nghiệm thực tiễn, bằng khảo sát các hệ đang tồn tại, không thể chỉ học qua sách vở.

3.1.1.2. Vai trò của thiết kế

Thiết kế là chuyển yêu cầu phần mềm thành các đặc tả thiết kế mô tả một hệ thống phần mềm mới sẽ xây dựng, vì thế:

- Thiết kế là cách duy nhất để chuyển hóa một cách chính xác các yêu cầu của khách hàng thành mô hình thiết kế hệ thống phần mềm cuối cùng, làm *cơ sở cho việc triển khai chương trình phần mềm*.
- Tài liệu thiết kế phần mềm là *công cụ giao tiếp* giữa các nhóm cùng tham gia vào việc phát triển sản phẩm, *để quản lý các rủi ro, đạt đến phần mềm hiệu quả*.
- Thiết kế phần mềm là tài liệu *cung cấp đầy đủ các thông tin cần thiết* cho những người kỹ sư hệ thống *để bảo trì hệ thống* sau này.

Không có thiết kế sẽ có nguy cơ sản sinh một hệ thống không tin cậy – một hệ thống dễ thất bại. Với bản chất không nhìn thấy và phức tạp, một hệ thống phần mềm rất khó xác định được chất lượng chừng nào chưa được kiểm thử, chưa vận hành, nên *thiết kế tốt là chìa khóa làm cho kỹ nghệ phần mềm hữu hiệu*. Không thể hình thức hóa mọi quá trình thiết kế đối với bất kỳ một sản phẩm phần mềm nào.

3.1.1.3. Các khái niệm trong thiết kế

Bất kể phương pháp luận thiết kế được dùng là phương pháp nào, những khái niệm dưới đây là cần thiết cho việc mô tả các nội dung thiết kế.

a) Trừu tượng (abstraction)

Trừu tượng là sự mô tả một cách khái quát một đối tượng thực và bỏ qua nhiều yếu tố, nhiều mặt không quan trọng của nó. Wasserman đã đưa ra định nghĩa sau: "*Ký pháp trừu tượng mang tính tâm lý, cho phép ta tập trung vào một vấn đề ở một mức nào đó của sự khái quát, bỏ qua các chi*

tiết ở mức thấp ít liên quan. Việc sử dụng sự trừu tượng cũng cho phép ta làm việc với các khái niệm và thuật ngữ gắn gũi trong môi trường của vấn đề đặt ra, mà không phải chuyển chúng thành một cấu trúc không quen thuộc". Nếu các mặt, các yếu tố của đối tượng được mô tả bị bỏ qua càng nhiều thì mức trừu tượng hóa càng cao. Như vậy, có thể mô tả đối tượng thiết kế với nhiều mức trừu tượng khác nhau tùy thuộc vào sự hiểu biết, nhận thức của người phát triển và yêu cầu đặt ra đối với nó.

a1) Các mức trừu tượng

Có nhiều mức trừu tượng cho các đối tượng thiết kế:

1. *Mức cao nhất:* Mô tả đại thể sử dụng ngôn ngữ của môi trường vấn đề (ngôn ngữ nghiệp vụ, ngôn ngữ người dùng).
2. *Mức vừa:* Mô tả hướng thủ tục nhiều hơn. Thuật ngữ hướng vấn đề thường đi đôi với các mô tả bằng các phương pháp được sử dụng.
3. *Mức thấp:* Phát biểu theo thuật ngữ chi tiết để có thể chuyển trực tiếp thành chương trình.

Khi thiết kế, yêu cầu trước hết được chuyển vào thiết kế sơ bộ, rồi chuyển thành thiết kế chi tiết, mức độ trừu tượng từng bước giảm dần. Cuối cùng, thiết kế cần đạt tới mức trừu tượng thấp nhất để có thể sinh ra chương trình.

a2) Các dạng trừu tượng

Có một số dạng trừu tượng tương ứng với các đối tượng của thiết kế như:

1. *Trừu tượng thủ tục* là một dãy các lệnh có tên, có chức năng xác định và giới hạn. Một ví dụ về thủ tục trừu tượng là từ "đi vào" cửa. Đó là một dãy các "bước thủ tục": *bước tới cửa, nắm lấy quả đấm, xoay quả đấm và kéo cửa ra,...* Trừu tượng thủ tục được sử dụng để mô tả chương trình, thuật toán.
2. *Trừu tượng dữ liệu* là một tập các dữ liệu có tên mô tả cho một đối tượng dữ liệu. Ví dụ về dữ liệu trừu tượng là "séc thanh toán". Đối tượng dữ liệu này thực chất là một tập hợp nhiều mẫu thông tin khác nhau (số séc, tên người thanh toán, số tiền thanh toán, tiền thuế,...). Vì

vậy, chúng ta có thể tham khảo tới mọi dữ liệu bằng cách nói tên của trừu tượng dữ liệu.

3. *Trừu tượng điều khiển*. Trừu tượng điều khiển áp dụng để mô tả các đối tượng của hệ thống có quan hệ tương tác với nhau mà không xác định các chi tiết bên trong chúng. Một ví dụ về trừu tượng điều khiển là cơ chế đồng bộ hóa được dùng để điều phối các hoạt động trong hệ điều hành.

Nhờ có sự trừu tượng hóa mà ta có thể nghiên cứu các đối tượng của thế giới thực bằng cách nghiên cứu trên các mô hình của chúng. Thiết kế là việc tạo ra mô hình của hệ thống thực mà chúng ta cần xây dựng.

b) Phân rã (decomposition)

Phân rã là việc phân chia một đối tượng thành những đối tượng nhỏ hơn mà chúng vốn đã tồn tại. Nó là cách để có thể dễ dàng nghiên cứu những vấn đề lớn, phức tạp thông qua việc nghiên cứu các đối tượng con đơn giản hơn của nó.

c) Làm mịn (refinement)

Làm mịn từng bước là một chiến lược thiết kế từ trên – xuống, được Niklaus Wirth [40] đề nghị lần đầu tiên. Kiến trúc của một chương trình được phát triển bằng các mức được làm mịn liên tiếp. Bắt đầu với một phát biểu về chức năng như mức trừu tượng cao nhất. Phát biểu này không đưa ra cách làm trong nội bộ mỗi chức năng hay cấu trúc của thông tin đó. Việc làm mịn buộc người thiết kế phải đưa ra ngày càng nhiều phần tử chi tiết mỗi khi yêu cầu làm mịn xuất hiện và tiếp tục cho đến khi có thể biểu diễn chúng bằng cấu trúc của một ngôn ngữ lập trình. Làm mịn khác phân rã ở chỗ đó là *hoạt động mang tính chủ quan*: thiết kế được bổ sung các đối tượng thành phần nhỏ hơn nhưng theo chủ định của người thiết kế.

d) Môđun (modul)

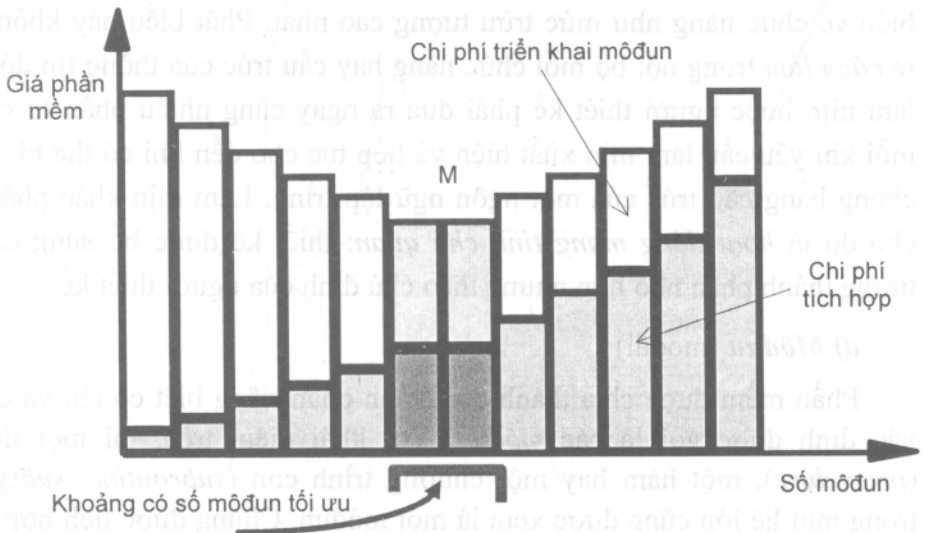
Phần mềm được chia thành các thành phần riêng biệt có tên và địa chỉ xác định được gọi là các *môđun*. Với khái niệm trên, thì một thủ tục (*procedure*), một hàm hay một chương trình con (*subroutine, subsystem*) trong một hệ lớn cũng được xem là một môđun. Chúng được tích hợp lại để giải quyết yêu cầu của vấn đề đặt ra. Khái niệm môđun có ý nghĩa rất lớn: *Tính môđun là thuộc tính riêng của phần mềm, nó cho phép tổ chức một*

chương trình trở nên quản lý được theo một cách thông minh. Người đọc không thể hiểu tốt một phần mềm nguyên khối. Cơ sở của việc môđun hóa dựa trên kết quả nghiên cứu thực nghiệm khi giải quyết các vấn đề thể hiện qua hai phương trình sau: Nếu một vấn đề p có thể phân thành hai vấn đề p_1, p_2 nhỏ hơn để giải quyết thì ta có:

$$C(p_1 + p_2) > C(p_1) + C(p_2) \text{ và } E(p_1 + p_2) > E(p_1) + E(p_2)$$

Trong đó, C là hàm xác định độ phức tạp và E là hàm xác định công sức cần để giải quyết vấn đề. Hai phương trình cho biết rằng: *Khi chia nhỏ vấn đề thì việc giải quyết nó trở nên dễ hơn và công sức bỏ ra để giải quyết nó cũng ít hơn.*

Nỗ lực để phân chia phần mềm thành các môđun riêng rẽ làm cho tổng số các môđun tăng lên. Với cùng một tập các yêu cầu, nhiều môđun hơn có nghĩa là kích cỡ từng môđun nhỏ đi, dễ hiểu hơn và chi phí thực hiện từng môđun ít hơn. Nhưng khi số các môđun tăng lên, thì công sức để liên kết chúng lại thành hệ thống tổng thể cũng tăng lên. Vấn đề đặt ra là: *Môđun hóa như thế nào để tổng chi phí phát triển phần mềm là ít nhất?* Parnas [28] đã đưa ra các tiêu chuẩn để phân rã một hệ thống thành các môđun. Trong nghiên cứu *thiết kế môđun hiệu quả*, Meyers [22] đã nêu ra năm tiêu chuẩn để đánh giá một phương pháp thiết kế nhằm tạo nên một hệ thống có số môđun hiệu quả (hình 3.1).



Hình 3.1. Một quan hệ giữa số lượng môđun và chi phí phần mềm

e) Thủ tục phần mềm (software procedure)

Thủ tục phần mềm tập trung vào mô tả chi tiết các bước xử lý cho từng môđun 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, và có thể bao gồm cả cấu trúc/tổ chức dữ liệu được sử dụng.

f) Che dấu thông tin (information hiding)

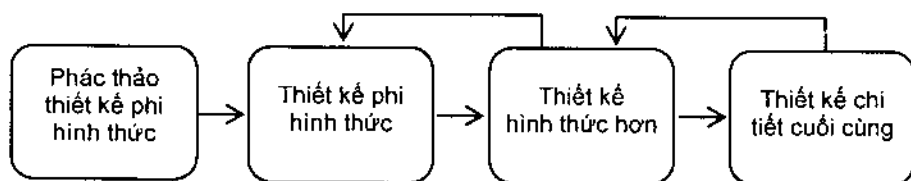
Nguyên lý về che dấu thông tin được Parnas [28] trình bày đã gợi ý rằng: *các môđun nên được thiết kế sao cho dấu kín dữ liệu đối với mọi môđun khác*. Nói cách khác, môđun nên được đặc tả và thiết kế sao cho những thông tin trong nó là không thể truy cập đến đối với những môđun khác không cần tới chúng. Che dấu thông tin là một tiêu chuẩn của thiết kế đối với một hệ thống cấu trúc từ các môđun.

3.1.2. Triển khai thiết kế

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*.

3.1.2.1. Tiến trình thiết kế

Tiến trình thiết kế (*design process*) là quá trình tăng cường hình thức hóa trong sự tiến triển của thiết kế và phải luôn quay trở lại các thiết kế đúng đắn ít hình thức có trước đó của quá trình này. Tiến trình thiết kế có thể mô tả như hình 3.2.



Hình 3.2. Mô hình tiến trình thiết kế

Kết quả của mỗi hoạt động thiết kế là một đặc tả. Đặc tả này có thể là một đặc tả trừu tượng, bán 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. Khi

quá trình thiết kế tiến triển thì các chi tiết được bổ sung dần vào đặc tả thiết kế. Các kết quả cuối cùng là các đặc tả về các thuật toán và các cấu trúc dữ liệu và đặc tả giao diện được dùng làm cơ sở cho việc mã hóa.

Tiến trình phát triển phần mềm là một sự hoàn thiện liên tục, nên *giữa thiết kế và đặc tả yêu cầu có quan hệ rất chặt chẽ với nhau*. Thực tế, người thiết kế sẽ phải lặp đi lặp lại giữa đặc tả yêu cầu và thiết kế. Vì rằng, khi ở mức ít trừu tượng hơn, người ta dễ phát hiện được những sai sót của các giai đoạn trước, do đó phải quay lại để chỉnh sửa sản phẩm trước đó. Hình 3.3 đưa ra các hoạt động của tiến trình thiết kế và các sản phẩm của nó. Các hoạt động này có thể là tùy biến, nhưng việc xác định nó làm cho tiến trình thiết kế trở nên nhìn thấy được và dễ quản lý.

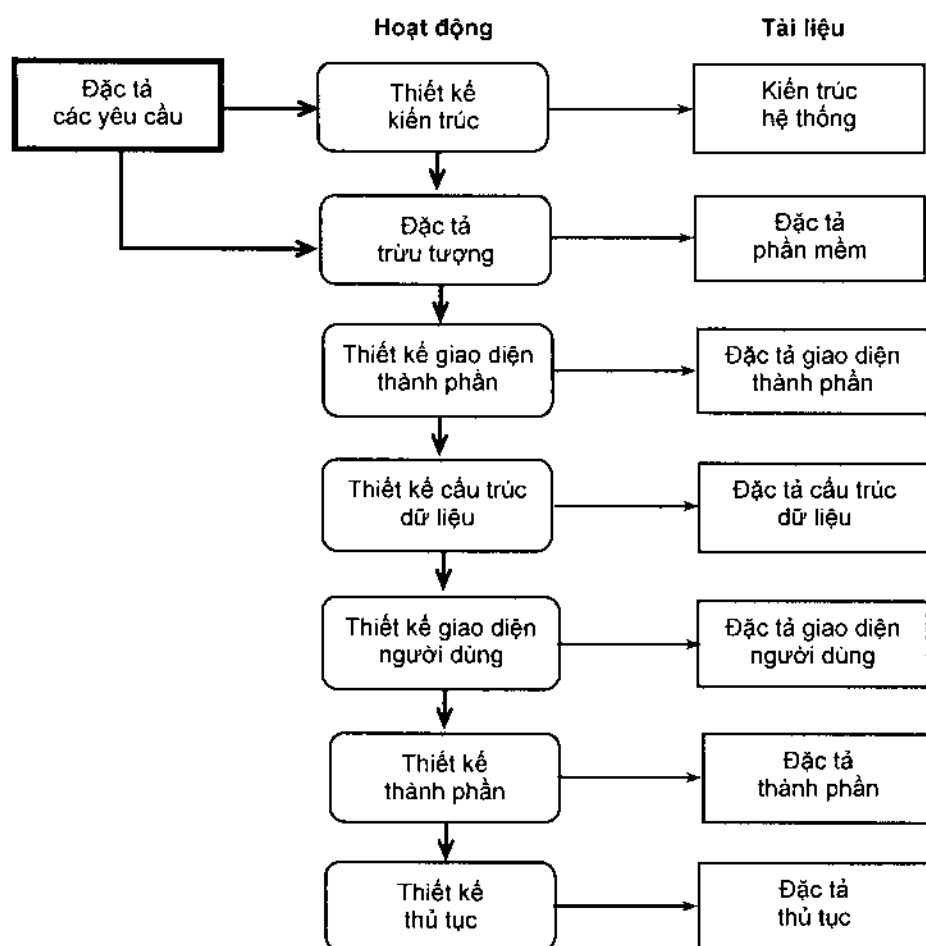
Phương pháp tiếp cận để phát triển hệ thống thường được dùng là *phương pháp tiếp cận từ trên xuống*: vấn đề được phân chia một cách đệ quy thành các vấn đề con cho tới khi các vấn đề nhận được có thể giải quyết được một cách dễ dàng (pha phân tích). Khi thiết kế thường được thực hiện *từ dưới lên*, vì rằng, 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 cho nó, và cũng nhận ra các thành phần nào có thể dùng lại được trong các phần khác (hay bài toán khác).

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

3.1.2.2. Các hoạt động và sản phẩm thiết kế

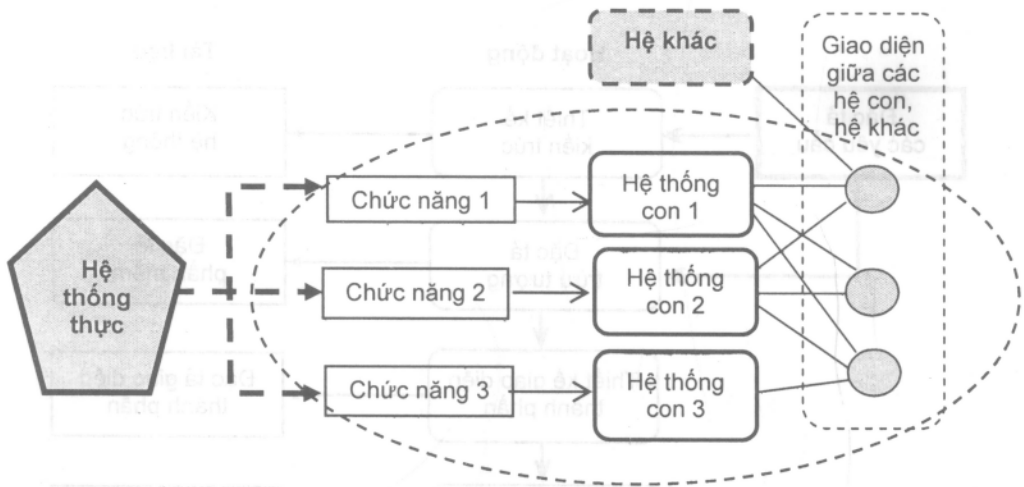
Thiết kế một hệ thống phần mềm lớn thường bao gồm các hoạt động cho ở hình 3.3:

1. *Thiết kế kiến trúc*: Xác định các hệ con tạo lên hệ thống tổng thể và các mối quan hệ giữa chúng.
2. *Đặc tả trừu tượng*: Đối với mỗi hệ con, cần mô tả trừu tượng các dịch vụ mà nó cung cấp cùng các ràng buộc mà nó phải tuân thủ khi cung cấp dịch vụ.



Hình 3.3. Các hoạt động thiết kế và sản phẩm của chúng

3. *Thiết kế các giao diện thành phần:* Thiết kế các giao diện của từng hệ con với các hệ con khác, với các hệ thống khác (từ môi trường) sao cho có thể sử dụng hệ con khác thông qua giao diện mà không cần biết nó sẽ thực hiện như thế nào, và nhờ vậy có thể phát triển các hệ con một cách độc lập (hình 3.4).
4. *Thiết kế cấu trúc dữ liệu:* Thiết kế cấu trúc dữ liệu lưu trữ và đặc tả các cấu trúc dữ liệu được dùng trong việc thực hiện hệ thống.
5. *Thiết kế hệ thống giao diện người dùng:* Thiết kế các giao diện bên ngoài mà qua đó người dùng có thể sử dụng được hệ thống.



Hình 3.4. Quá trình xác định hệ con phần mềm và giao diện giữa chúng

6. *Thiết kế các thành phần:* Phân chia các dịch vụ mà một hệ con cung cấp vào các thành phần hợp thành của nó.
7. *Thiết kế thủ tục:* Thiết kế đặc tả các thuật toán, quy trình dùng để thực hiện các dịch vụ của mỗi thành phần sao cho có thể ánh xạ trực tiếp nó vào một ngôn ngữ lập trình.

3.1.2.3. Biểu diễn thiết kế

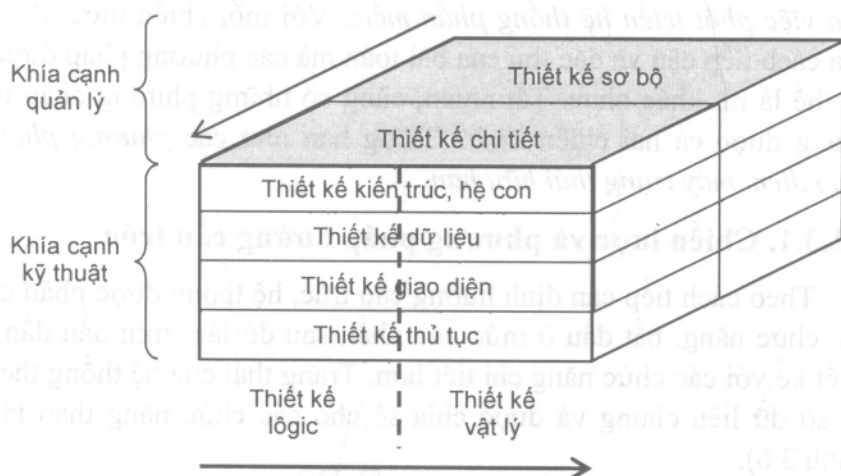
Một bản thiết kế phần mềm là một mô hình mô tả một đối tượng của thế giới thực với nhiều thành phần và các mối quan hệ giữa chúng với nhau. Có ba hình thức thường được sử dụng để mô tả thiết kế:

- *Các biểu đồ:* Các biểu đồ được dùng để thể hiện các mối quan hệ giữa các thành phần lập lên hệ thống và là mô hình mô tả thế giới thực. Việc mô tả đồ thị của các thiết kế là rất có lợi vì tính trực quan và cho một bức tranh tổng thể về hệ thống.
- *Ngôn ngữ mô tả chương trình:* Các ngôn ngữ này dùng để kiểm tra và cấu trúc các cơ cấu thiết kế dựa trên các cấu trúc của một ngôn ngữ lập trình.
- *Dạng văn bản không hình thức hóa:* Loại hình này mô tả các thông tin không thể hình thức hóa được như các thông tin phi chức năng bên cạnh các cách mô tả khác.

3.1.2.4. Các giai đoạn thiết kế

Có thể nhìn nhận tiến trình thiết kế phần mềm theo những cách nhìn khác nhau: *theo người quản lý, theo nhà phát triển, theo mức độ hình thức hóa* (hình 3.5). Theo quan điểm quản lý, thiết kế chia thành 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 trải qua hai giai đoạn (hình 3.5):

1. *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 một phương tiện vật lý nào. Nhờ loại bỏ các yếu tố vật lý, 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 được các yêu cầu đặt ra.



Hình 3.5. Quá trình thiết kế nhìn theo các cách khác nhau

2. *Thiết kế vật lý*: Chọn các giải pháp công nghệ hiện hữu để thực hiện các cấu trúc logic đã cho phù hợp với điều kiện của môi trường đích dự kiến của hệ thống phần mềm. Giai đoạn này đôi khi được gọi là *thiết kế chi tiết* và đôi khi là *lập trình*. Lúc 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.

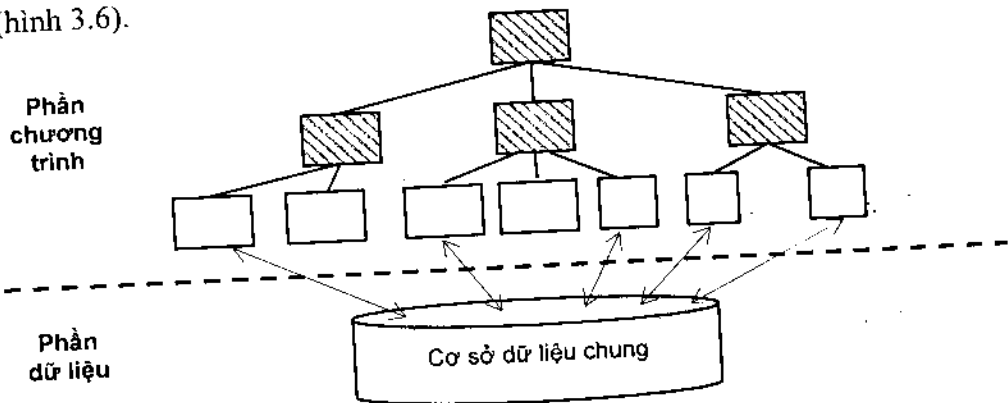
3.1.3. Các chiến lược và phương pháp thiết kế

Việc thiết kế phần mềm ở nhiều tổ chức vẫn còn là một quá trình tự học. Khi cho một tập hợp các yêu cầu (thường bằng ngôn ngữ tự nhiên) người ta dễ dàng có ngay một phác họa thiết kế không hình thức. Từ đó bắt đầu việc mã hóa và thiết kế được cải biên trong khi hệ thống được hình thành. Khi kết thúc giai đoạn triển khai thì sản phẩm thiết kế đã biến đổi khác xa so với đặc tả ban đầu đến mức mà các tài liệu thiết kế nguyên thủy là một mô tả hoàn toàn khác với hệ thống được tạo ra.

Hiện nay có rất nhiều phương pháp tiếp cận khác nhau được áp dụng cho thiết kế. Nó làm cho quá trình thiết kế trở nên rõ ràng hơn, có thể theo dõi được và mang tính khoa học nhiều hơn. Hai cách tiếp cận phổ biến hiện nay là: *Tiếp cận định hướng cấu trúc (hay chức năng)* và *tiếp cận định hướng đối tượng*. Tương ứng với mỗi cách tiếp cận này là một chiến lược cho việc phát triển hệ thống phần mềm. Với mỗi chiến lược, do đặc trưng của cách tiếp cận và đặc thù của bài toán mà các phương pháp được sử dụng có thể là rất khác nhau. Tất nhiên, cũng có những phương pháp thiết kế là chung được cả hai chiến lược. Chẳng hạn như các phương pháp thiết kế giao diện, máy trạng thái hữu hạn...

3.1.3.1. Chiến lược và phương pháp hướng cấu trúc

Theo cách tiếp cận định hướng cấu trúc, hệ thống được phân chia thành các chức năng, bắt đầu ở mức cao nhất, sau đó làm mịn dần 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 3.6).



Hình 3.6. Mô hình hệ thống hướng cấu trúc

Với đặc trưng trên, thiết kế chức năng thường có hai hoạt động tương đối độc lập: *thiết kế dữ liệu* và *thiết kế xử lý*. Sau giai đoạn phân tích – đặc tả được yêu cầu, thiết kế dữ liệu thường đi trước một bước để có cơ sở dữ liệu dùng chung (độc lập với chương trình). Các thiết kế xử lý (các chương trình) được thực hiện trên cơ sở sử dụng các cấu trúc dữ liệu lưu trữ đã có.

Vào cuối những năm 1960, Dijkstra và cộng sự đã đề nghị việc dùng một tập các cấu trúc logic đơn giản để từ đó tạo nên bất kỳ chương trình nào. Các cấu trúc đều nhấn mạnh vào "*duy trì miền chức năng*". Tức là, mỗi cấu trúc đều có một cấu trúc logic dự đoán được, có lối vào trên đỉnh, có lối ra ở đáy, làm cho độc giả theo dõi được luồng thủ tục dễ dàng hơn.

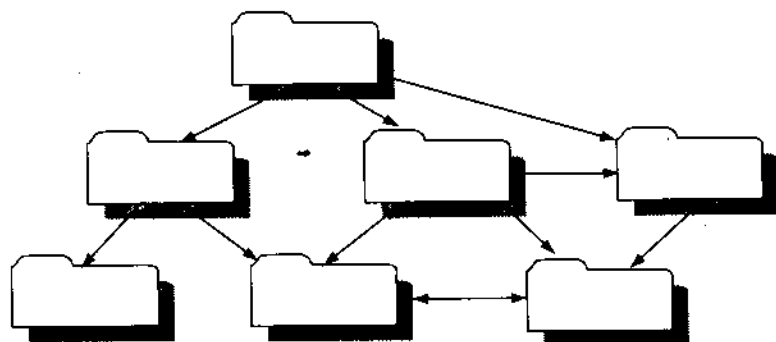
Ba cấu trúc tiêu biểu của lập trình cấu trúc được Bolun [7] nêu ra là *tuần tự*, *tuyển chọn* và *lặp*. Cấu trúc *tuần tự* dành cho các bước xử lý theo trình tự lần lượt mà về bản chất có trong đặc tả của bất kỳ thuật toán nào. Cấu trúc *tuyển chọn* dành cho việc xử lý có sự lựa chọn dựa trên một sự kiện logic nào đó, còn cấu trúc *lặp* thì thích hợp cho mọi quá trình lặp lại. Ba cấu trúc này là nền tảng cho lập trình có cấu trúc – một kỹ thuật thiết kế quan trọng trong lĩnh vực kỹ nghệ phần mềm.

Hàng loạt các phương pháp khác nhau được phát triển dựa trên tiếp cận định hướng cấu trúc như của Gane [14], của Yourdon và Constantine [41], của King (1984),... và đã sử dụng thành công trong nhiều dự án lớn từ nhiều năm nay, làm giảm giá thành đáng kể của nhiều dự án phần mềm, sử dụng được các khái niệm chuẩn và bảo đảm các thiết kế là tuân theo chuẩn. Nhiều công cụ CASE như *RWIN*, *Sybase* (2004), *Designer* (2000) và *Developer* (2000),... đã được phát triển để trợ giúp cho việc tự động hóa hoạt động phát triển theo hướng này.

3.1.3.2. Chiến lược và phương pháp hướng đối tượng

Theo hướng đối tượng, hệ thống được nhìn nhận như một bộ các đối tượng tương tác với nhau. Mỗi đối tượng bao hàm trong nó cả các dữ liệu và các thao tác thực hiện trên chúng, vì thế nó có trạng thái riêng. Mỗi đối tượng là một thể hiện của một lớp được xác định bởi *các thuộc tính* và *các phương thức*. Bộ các "*thuộc tính*" xác định trạng thái của đối tượng và các phương thức của nó thao tác với các dữ liệu của thuộc tính. Các lớp có thể

được thừa kế từ một vài lớp đối tượng cao hơn và khi định nghĩa nó chỉ cần nêu đủ những gì khác nhau giữa nó và các lớp cha. Các đối tượng liên lạc với nhau chỉ bằng cách trao đổi các thông báo để cộng tác thực hiện một dịch vụ chung (hình 3.7).



Hình 3.7. Mô hình cấu trúc của hệ thống hướng đối tượng

Mặc dù mới phát triển từ đầu những năm 90 của thế kỷ XX trở lại gần đây, có nhiều phương pháp phát triển phần mềm hướng đối tượng ra đời. Một ngôn ngữ chung có tính chuẩn cao UML cùng nhiều công cụ mạnh (như *Jbuilder*, *Rational Rose*) đã được phát triển để trợ giúp cho các phương pháp theo định hướng này.

3.1.3.3. Chiến lược thiết kế hệ thống tương tranh

Rất nhiều hệ thống, đặc biệt là các hệ thống thời gian thực được cấu trúc như là một hệ gồm một bộ các quá trình hoạt động song song và có liên quan chặt chẽ với nhau. Các hệ này thường phải tuân thủ những ràng buộc nghiêm ngặt về thời gian, trong đó các phần cứng thường hoạt động tương đối chậm, nên cần nhiều bộ xử lý hoạt động song song mới có thể đảm bảo đáp ứng được yêu cầu về thời gian.

Các chương trình tuần tự là dễ thiết kế, thực hiện, kiểm tra và thử nghiệm hơn là các hệ song song. Sự phụ thuộc thời gian giữa các quá trình là khó hình thức hoá, khó khống chế và thử nghiệm. *Phương pháp máy trạng thái hữu hạn* là công cụ biểu diễn tốt cho thiết kế hệ thời gian thực. Vì thế nó là một phần tích hợp của các phương pháp thiết kế thời gian thực. Nhiều thuật toán lập lịch khác nhau được phát triển cho các thiết kế loại này.

3.1.3.4. Cách tiếp cận chung của các phương pháp thiết kế

Tuy có nhiều phương pháp thiết kế khác nhau, nhưng chúng có nhiều điểm chung và thường được trợ giúp từ một số cách nhìn nhận [34] sau:

- *Cách nhìn cấu trúc* cho cái nhìn tổng thể thông qua lược đồ cấu trúc hệ thống. Hệ thống gồm các thành phần và mối quan hệ giữa chúng.
- *Cách nhìn quan hệ thực thể* mô tả các cấu trúc dữ liệu logic được dùng, nó liên quan đến đặc tả dữ liệu nhờ mô hình thực thể và các quan hệ của chúng.
- *Cách nhìn luồng dữ liệu*. Lược đồ luồng dữ liệu thể hiện quá trình vận động của dữ liệu và sự biến đổi của nó xảy ra ở mỗi chức năng mà nó được xử lý.
- *Cách nhìn vận động*. Lược đồ chuyển trạng thái để bổ sung cho các phương pháp trên, để nhìn một hệ thống diễn ra theo thời gian thực.

Như vậy, mỗi phương pháp thiết kế phần mềm đều đưa vào những cách nhìn nhận trực cảm và bộ ký pháp duy nhất để thể hiện, cũng như một cách nhìn riêng nào đó về những đặc trưng cho chất lượng thiết kế. Tuy vậy, các phương pháp thiết kế đều có *những đặc trưng chung* sau:

1. *Một cơ chế để chuyển hóa* từ biểu diễn miền thông tin (của đặc tả) thành biểu diễn thiết kế.
2. *Các ký pháp để biểu diễn* các thành phần chức năng và giao diện của chúng.
3. *Cách trực cảm để phân hoạch và làm mịn* vấn đề.

Sự thật, đối với các hệ phần mềm lớn, phức tạp, người ta phải dùng các phương pháp tiếp cận khác nhau cho các phần khác nhau. *Không có một chiến lược nào là tốt nhất cho mọi dự án lớn. Các cách tiếp cận trên là bổ sung hỗ trợ cho nhau mà không đối kháng với nhau.* Người kỹ sư phần mềm cần chọn cách tiếp cận thích hợp nhất cho từng giai đoạn thiết kế.

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

Không có một hệ tiêu chuẩn chính xác để xác định được thế nào là thiết kế tốt. Một thiết kế tốt có thể phải cho phép sản sinh mã hữu hiệu; phải tối

thiếu công sức để chuyển thành chương trình, hoặc thiết kế phải là tốt nhất cho bảo trì (không chỉ dễ bảo trì mà còn ít tốn kém)...

Tiêu chuẩn sau cùng không chỉ tốt cho người dùng. Một thiết kế cho bảo trì tốt *có thể thích nghi* với việc cải biên các chức năng đã có và thêm các chức năng mới. Muốn vậy nó phải là dễ hiểu, dễ sửa đổi và việc sửa đổi chỉ có thể gây ra hiệu ứng cục bộ. Do đó, mỗi thành phần của thiết kế phải là *liên kết chặt chẽ*, nghĩa là, tất cả các bộ phận của nó phải có một quan hệ logic chặt chẽ với nhau. Các thành phần khác nhau phải độc lập tương đối, tức là được *ghép nối với nhau một cách lỏng lẻo*, nhờ vậy mà dễ nhận dạng, dễ thích nghi và giảm ảnh hưởng lan truyền khi sửa đổi một thành phần. Vì vậy, *kết dính chặt* và *ghép nối lỏng* được xem là hai tiêu chí quan trọng của chất lượng thiết kế. Ngoài ra, tính bảo trì được cũng liên quan chặt chẽ đến *tính hiểu được*, đến *tính thích nghi được* của phần mềm.

3.1.4.1. Sự kết dính

Sự kết dính (Cohesion) của một thành phần là độ đo về tính gắn kết chặt chẽ với nhau giữa các bộ phận của nó. Muốn vậy, một thành phần phải thực hiện một chức năng logic hoặc một thực thể logic, và tất cả các bộ phận của nó đều phải tham gia vào việc thực hiện này. Nếu một phần không trực tiếp tham gia vào việc thực hiện chức năng logic đó thì mức độ kết dính của nó là thấp.

Page – Jones [27] đưa ra bảy mức kết dính theo thứ tự tăng dần sau đây:

1. *Kết dính gom góp*: Các phần của một thành phần không liên quan với nhau, song lại bị bỏ vào một thành phần.
2. *Kết dính hội hợp logic*: Các thành phần cùng thực hiện các chức năng tương tự (xử lý lỗi, cập nhật,...) được đặt vào cùng một thành phần.
3. *Kết dính theo thời điểm*: Tất cả các thành phần cùng hoạt hóa một lúc (cùng bắt đầu, cùng kết thúc,...) được ghép lại với nhau.
4. *Kết dính thủ tục*: Các phần tử trong thành phần được ghép lại trong một dãy điều khiển.
5. *Kết dính truyền thông*: Tất cả các phần tử của thành phần cùng thao tác trên một dữ liệu vào và đưa ra cùng một dữ liệu ra.

6. *Kết dính tuần tự*: Trong một thành phần, cái ra của phần tử này là cái vào của phần tử khác.
7. *Kết dính chức năng*: Mỗi phần của thành phần đều là cần thiết để thực hiện cùng một chức năng nào đó.

Các lớp kết dính trên đây không được định nghĩa chặt chẽ và cũng không phải dễ xác định. Một đối tượng kết dính thể hiện ra như một thực thể đơn, tức là mọi phép toán nó thực hiện đều nằm trong đó. Vậy có một lớp kết dính nữa là:

8. *Kết dính đối tượng*: Mỗi phép toán thực hiện một chức năng. Các chức năng này tác động chỉ lên các thuộc tính của đối tượng để cải biên, thanh tra và sử dụng chúng làm cơ sở cho việc cung cấp dịch vụ của đối tượng.

3.1.4.2. Sự ghép nối

Sự ghép nối (coupling) chỉ ra mức độ độc lập giữa các đơn vị thành phần của một chương trình. Hệ thống có ghép nối cao sẽ có độ liên kết mạnh giữa các thành phần, làm chúng phụ thuộc lẫn nhau nhiều hơn. Hệ thống ghép nối lỏng lẻo làm cho các thành phần trở nên độc lập hoặc tương đối độc lập với nhau hơn.

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, cũng như bản chất của các mối quan hệ đó. Chẳng hạn, mối quan hệ đồng bộ (gọi chờ đáp lại) là chặt hơn mối quan hệ không đồng bộ (gửi thông báo đi). Thừa kế trong hệ thống hướng đối tượng dẫn tới một dạng ghép nối chặt giữa các lớp có quan hệ kế thừa. Việc tối thiểu hóa sự ghép nối không phải luôn luôn có thể thực hiện được. Page – Jones [27] đã giới thiệu năm loại ghép nối được trình bày theo thứ tự từ tốt đến xấu:

1. *Ghép nối dữ liệu*: Các môđun truyền thông với nhau bằng các phần tử dữ liệu hay thông tin điều khiển (cờ). Không một môđun nào có thể biết những gì diễn ra trong một môđun khác.
2. *Ghép nối nhãn*: Các dữ liệu được gửi đi là những cấu trúc dữ liệu hay toàn bộ một bản ghi. Sự thay đổi trong cấu trúc dữ liệu sẽ tác động lên mọi môđun sử dụng nó, chúng phụ thuộc nhau nhiều hơn.

3. *Ghép nối điều khiển*: Việc điều khiển xảy ra khi một môđun gửi đi *thông tin điều khiển* đến môđun khác. Thông tin này thể hiện như một cờ trong sơ đồ cấu trúc, nó thông báo cho môđun nhận cần phải làm cái gì.
4. *Ghép nối chung*: khi hai môđun tham chiếu đến cùng một dữ liệu tổng thể, chúng được xem là ghép nối chung. Trong một số ngôn ngữ (như FORTRAN) cho phép có vùng dữ liệu tổng thể đó. Trong trường hợp này, lỗi từ vùng dữ liệu tổng thể có nguy cơ lan ra toàn hệ thống.
5. *Ghép nối nội dung*: Loại này xảy ra khi một môđun trực tiếp tham chiếu đến hoạt động của một môđun khác. Đây là loại ghép nối xấu nhất.

3.1.4.3. Tính hiểu được

Tính hiểu được (*Understandability*) liên quan tới một số các đặc trưng sau:

1. *Tính kết dính*: Ta có thể hiểu được một thành phần mà không cần tham khảo đến một thành phần nào khác. Điều này tùy thuộc sự liên kết chặt chẽ hay không giữa các phần tử của thành phần.
2. *Đặt tên*: Mọi tên được dùng trong thành phần đó đều phải có nghĩa. Tên có nghĩa là tên phản ánh các thực thể của thế giới thực được mô tả trong thành phần đó.
3. *Soạn tư liệu*: Thành phần được soạn tư liệu đầy đủ và rõ ràng để ánh xạ được các thực thể của thế giới thực vào các yếu tố của nó.
4. *Độ phức tạp*: Độ phức tạp của các thuật toán được dùng để thực hiện thành phần đó đến mức độ không quá cao.

Độ phức tạp cao ám chỉ các mối quan hệ giữa các thành phần khác nhau của thiết kế đó và một cấu trúc logic phức tạp liên quan đến độ sâu lồng nhau của phát biểu if – then – else hay vòng lặp. Các thành phần phức tạp là khó hiểu, nên phải thiết kế thành phần sao cho càng đơn giản càng tốt.

3.1.4.4. Tính thích nghi được

Một thiết kế dễ bảo trì thì phải là có *tính thích nghi được (Adaptability)*, tức là chúng được *ghép nối lỏng lẻo*. Sự thích nghi được còn đòi hỏi thiết kế

phải được *soạn thảo tư liệu tốt*: Tư liệu phải dễ hiểu và phải nhất quán với sự thực hiện (sự thực hiện phải được viết ra theo cách dễ đọc).

Một thiết kế dễ thích nghi cần phải có *mức nhìn thấy được cao*, tức là có một quan hệ rõ ràng giữa các mức khác nhau: Cấu trúc hệ thống được thể hiện từng bước qua việc làm mịn các biểu đồ luồng dữ liệu từ một mức sang mức sau và có thể lần vết ngược lại. Để có độ thích nghi tối ưu thì một thành phần phải là *tự chứa*. Muốn là tự chứa hoàn toàn thì một thành phần không nên dùng các thành phần ngoại lai. Tuy nhiên, điều đó lại mâu thuẫn với việc dùng lại các thành phần sẵn có. Vì vậy, cần có một sự thỏa hiệp giữa *tính dùng lại* và *sự mất tính thích nghi được*.

3.1.4.5. Các giải pháp cho một thiết kế tốt

Ta có thể nhận được một thiết kế tốt nếu *thực hiện đúng tiến trình thiết kế phần mềm thông qua việc áp dụng các nguyên lý thiết kế cơ bản, các phương pháp luận hệ thống, các công cụ trợ giúp và việc xét duyệt nghiêm túc*. Tuy nhiên, điều đó chưa đủ đảm bảo chắc chắn một thiết kế đã hoàn hảo. Về nguyên tắc, để *đảm bảo chất lượng thiết kế* cần có một hệ các tiêu chuẩn cho thiết kế. Xây dựng được một hệ thống tiêu chuẩn tốt đã không đơn giản, thì quá trình đảm bảo thiết kế thực hiện các chuẩn đó càng khó khăn hơn. Thay vào đó, người ta đã đưa ra các hướng dẫn để hướng thiết kế đạt được một mức độ chất lượng nhất định:

a) Những hướng dẫn cần được vận dụng trong thiết kế

- *Tổ chức phân cấp* để kiểm soát một cách thông minh các thành phần.
- *Tổ chức theo các môđun* để phân hoạch một cách logic các thành phần thực hiện thành những chức năng và chức năng con.
- *Sử dụng cách biểu diễn phân biệt và tách biệt giữa dữ liệu và thủ tục*.
- *Tổ chức các môđun dùng chung* cho các chức năng đặc trưng đặc biệt.
- *Hình thành giao diện* để rút gọn độ phức tạp của việc ghép nối giữa các môđun với môi trường bên ngoài cũng như giữa chúng với nhau.
- *Sử dụng lại các thành phần phần mềm* đã có, trong đó có các mã nguồn, các mẫu thiết kế...

b) Những nguyên lý thiết kế cần vận dụng [29]

1. Không ràng buộc tiến trình thiết kế trong một "*cách nhìn hạn hẹp*". Người thiết kế *cần tính đến mọi cách tiếp cận* khác nhau dựa trên yêu cầu của vấn đề đặt ra và khả năng các nguồn lực có được.
2. Thiết kế *có thể lần vết* trở lại mô hình phân tích hay các bước trước nó.
3. Thiết kế *không nên giải quyết vấn đề đã được giải quyết*. Nên sử dụng các mẫu đã gặp để cấu trúc hệ thống.
4. Thiết kế phải *rút ngắn được khoảng cách* giữa phần mềm và vấn đề tồn tại trong thế giới thực.
5. Thiết kế cần thể hiện tính *nhất quán và tích hợp*.
6. Thiết kế cần được cấu trúc để *dễ thay đổi*.
7. *Thiết kế không phải mã hóa*, mã hóa không phải thiết kế.
8. Thiết kế *cần được xem xét ngay từ đầu* để tối thiểu hóa các lỗi.
9. Thiết kế *cần được đánh giá và rà soát chất lượng* trong quá trình phát triển.

3.2. Thiết kế kiến trúc

3.2.1. Kiến trúc phần mềm và vai trò của nó

3.2.1.1. Định nghĩa kiến trúc

Kiến trúc phần mềm (software architecture) chỉ một cấu trúc tổng thể của phần mềm và qua đó cung cấp một sự tích hợp về mặt khái niệm của một hệ thống [34]. Ở dạng đơn giản nhất, 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 các 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 (cấu trúc, hành vi, dữ liệu,...). Trong UML [8] đã trình bày năm khung nhìn khác nhau xác định năm loại mô hình kiến trúc của hệ thống tương ứng với chúng.

3.2.1.2. Vai trò và tầm quan trọng của kiến trúc

Kiến trúc có một vai trò quan trọng trong phát triển phần mềm. Bass và đồng nghiệp của ông đã chỉ ra ba vai trò của kiến trúc phần mềm là:

- *Công cụ giao tiếp* giữa những người có liên quan.
- *Để phân tích hệ thống*: Khi đưa ra kiến trúc hệ thống ở giai đoạn đầu, nhà phân tích có thể tiến hành những quyết định thiết kế có tác động sâu sắc đến hệ thống sau này như tính hoàn thiện, độ ổn định, tính bảo trì được,...
- *Giúp sử dụng lại* ở quy mô lớn: Kiến trúc thu được có thể dùng cho các hệ thống có yêu cầu tương tự và vì vậy hỗ trợ việc sử dụng lại với quy mô lớn.

Kiến trúc không phải thành phần hoạt động, nhưng có tác động sâu rộng đến toàn bộ tiến trình phát triển phần mềm sau này. Nó là một mô tả về phần mềm, cho phép các kỹ sư phần mềm thực hiện những công việc sau đây:

1. *Tăng cường hiểu biết* về hệ thống cần xây dựng. Mô hình kiến trúc mô tả hệ thống theo nhiều khía cạnh khác nhau giúp các nhà phát triển có thể hình dung hệ thống cần xây dựng một cách đầy đủ và chính xác.
2. *Phân tích hiệu quả*. Với một tập các yêu cầu, có thể có nhiều phương án kiến trúc khác nhau. Phân tích giúp nhà thiết kế lựa chọn một kiến trúc hiệu quả nhất trong điều kiện có thể.
3. *Xem xét, sửa đổi kiến trúc từ sớm, giảm rủi ro*: Ở giai đoạn đầu khi sự thay đổi thiết kế còn dễ dàng, ít tốn kém, do đó việc phát hiện và sửa đổi những sai sót sẽ làm giảm thiểu được chi phí hay rủi ro có thể do kiến trúc gây ra.

3.2.2. Các mô hình kiến trúc

Kiến trúc phần mềm được mô tả ở dạng các mô hình khác nhau. Các mô hình kiến trúc được phát triển thường có thể nhìn nhận dưới một số mặt [34] sau:

1. *Mô hình kiến trúc tĩnh* chỉ ra các hệ con hay các thành phần được phát triển như một đơn vị độc lập.

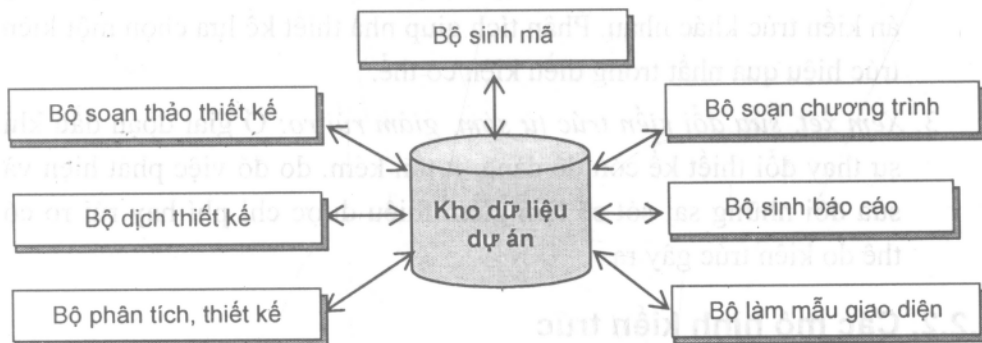
2. *Mô hình tiến trình động* chỉ ra hệ thống được tổ chức thành các tiến trình vận hành.
3. *Mô hình giao diện* xác định các giao diện đưa ra các dịch vụ mà mỗi hệ con cung cấp thông qua giao diện của chúng.
4. *Mô hình liên kết* chỉ ra mối liên kết giữa các hệ con hay giữa các thành phần.

3.2.2.1. Các kiến trúc theo cách tổ chức dữ liệu

Trong một hệ thống, các hệ thống con cần trao đổi dữ liệu với nhau để làm việc hiệu quả. Có hai cách cơ bản để tổ chức thực hiện được điều này: tổ chức dữ liệu tập trung tại một chỗ và tổ chức dữ liệu phân tán ở nhiều vị trí.

a) *Mô hình kho dữ liệu (repository model)*

Mô hình kho dữ liệu là một cách tổ chức dữ liệu tập trung. Tất cả dữ liệu chia sẻ được lưu giữ tại một cơ sở dữ liệu trung tâm, sao cho tất cả các hệ con đều có thể truy nhập. Mô hình này thích hợp với các ứng dụng khi dữ liệu do mỗi hệ con sinh ra được tập trung lại để các hệ con khác có thể sử dụng, chẳng hạn các hệ CAM/CAD hay các bộ công cụ CASE. Hình 3.8 là một ví dụ kiến trúc kho dữ liệu tập trung.



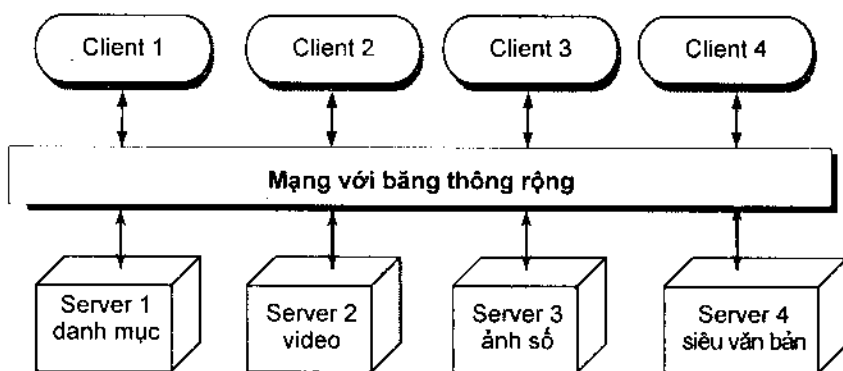
Hình 3.8. Kiến trúc trong bộ CASE tích hợp

b) *Mô hình máy khách/máy dịch vụ (client/server)*

Mô hình kiến trúc máy khách/máy dịch vụ là một mô hình hệ thống phân tán, ở đó dữ liệu và xử lý được phân bố trên các máy khác nhau. Thành phần chính của mô hình này bao gồm:

- Các máy dịch vụ cung cấp dịch vụ cho các hệ con khác.

- Các máy khách gọi để sử dụng dịch vụ từ các máy dịch vụ.
- Mạng liên kết cho phép các máy khách truy nhập đến máy dịch vụ.



Hình 3.9. Kiến trúc hệ thống tra cứu ảnh và video số

Các máy khách cần biết tên của các máy dịch vụ đang sẵn sàng và tên các dịch vụ mà nó cung cấp. Ngược lại, máy dịch vụ không cần biết định danh của máy khách hoặc có bao nhiêu máy khách. Các máy khách sử dụng dịch vụ mà máy dịch vụ cung cấp thông qua thủ tục gọi từ xa (hình 3.9). Một lợi thế của mô hình này là mô hình phân tán, có thể dễ dàng thêm vào nó một máy dịch vụ và tích hợp nó với các máy còn lại, hoặc nâng cấp máy dịch vụ mà không gây ảnh hưởng đến các phần khác của hệ thống.

3.2.2.2. Các mô hình điều khiển

Để làm việc như một hệ thống, các hệ con cần được điều khiển sao cho dịch vụ được gửi đến đúng chỗ, đúng thời gian. Các mô hình điều khiển ở mức kiến trúc liên quan đến luồng điều khiển giữa các hệ con. Có hai phương thức điều khiển:

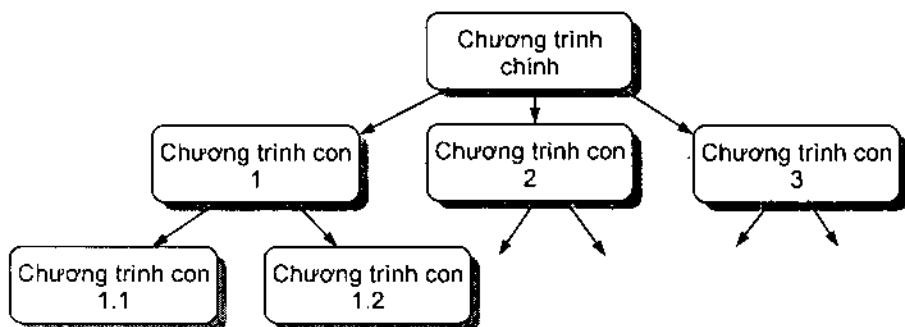
- *Điều khiển tập trung (centralised model)*: Một hệ con có trách nhiệm điều khiển, khởi động hay dừng các hệ con khác. Nó cũng có thể chuyển một điều khiển đến một hệ con khác và chờ để trách nhiệm điều khiển trả lại nó.
- *Điều khiển dựa trên sự kiện (event – based model)*: Thông tin điều khiển được chuyển đến một số hệ con, mỗi hệ con có thể đáp ứng sự

kiện sinh ra từ bên ngoài. Các sự kiện có thể đến từ hệ con khác hay từ môi trường của hệ thống.

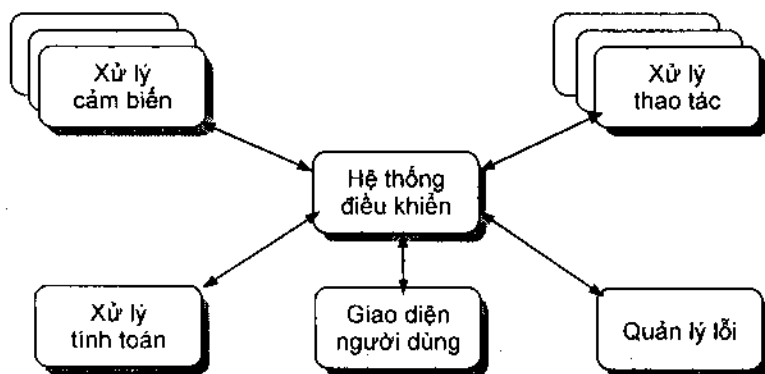
a) Mô hình điều khiển tập trung

Trong mô hình điều khiển tập trung, một hệ con được thiết kế như bộ điều khiển toàn hệ thống. Các mô hình điều khiển tập trung có thể thuộc vào hai lớp:

1. *Mô hình gọi – trả lại (call – return model)*: Nó tương tự mô hình hệ con trên xuống, mà ở đó điều khiển bắt đầu từ đỉnh của cây phân cấp và thông qua môđun để gọi, gửi đến tầng thấp hơn của cây. Mô hình này chỉ áp dụng cho các hệ tuần tự.



Hình 3.10. Mô hình điều khiển gọi – trả lại



Hình 3.11. Mô hình quản lý tập trung cho hệ thời gian thực

2. *Mô hình quản lý (manager model)*. Nó được áp dụng cho các hệ tương tranh. Một hệ thống thành phần được thiết kế như một hệ quản lý và điều khiển sự bắt đầu, dừng hay phối hợp các tiến trình của các hệ thống khác.

Hình 3.10 là một minh họa của mô hình gọi – trả lại. Hình 3.11 minh họa một mô hình điều khiển tập trung một hệ tương tranh. Mô hình này thường được sử dụng hệ thống thời gian thực "mềm", không ràng buộc quá chặt về thời gian.

b) Mô hình điều khiển dựa trên sự kiện

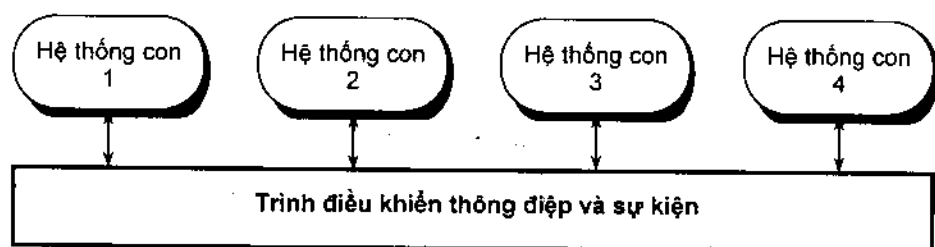
Các mô hình điều khiển loại này được điều khiển bằng các sự kiện sinh ra từ bên ngoài. Dưới đây xét hai loại mô hình tiêu biểu trong số đó:

- *Mô hình điều khiển quảng bá (Broadcast model):* Trong các mô hình này, một sự kiện về nguyên tắc sẽ chuyển tới tất cả các hệ thống con mà có thể được đáp ứng lại sự kiện đó.
- *Mô hình điều khiển ngắt (Interrupt – driven model):* Mô hình này được dùng trong các hệ thống thời gian thực, các ngắt được một bộ điều khiển ngắt nhận ra. Sau đó chúng được chuyển đến một số thành phần khác để xử lý.

b1) Mô hình điều khiển quảng bá

Trong một mô hình quảng bá (hình 3.12), các hệ thống con đăng ký một vài sự kiện cụ thể mà chúng quan tâm. Khi một sự kiện xảy ra, điều khiển được chuyển tới hệ thống con đăng ký loại sự kiện này.

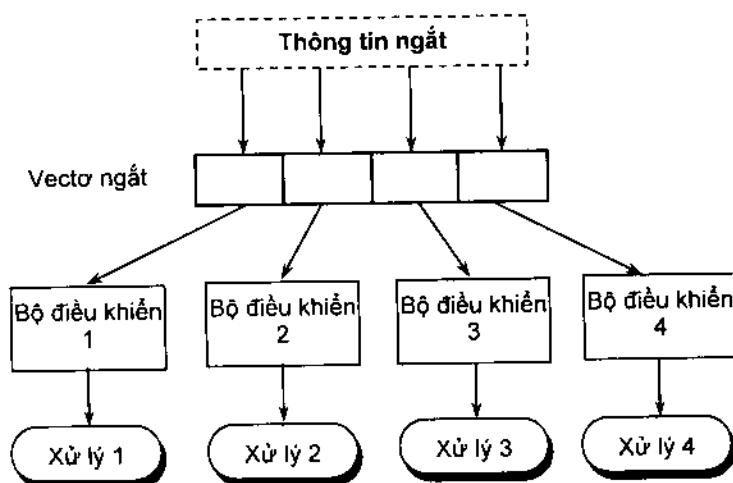
Ưu điểm của các tiếp cận quảng bá là sự giải quyết các mối quan hệ một cách đơn giản. Một hệ thống con mới điều khiển những lớp cụ thể của các sự kiện có thể được tích hợp bằng cách đăng ký những sự kiện của nó với bộ điều khiển sự kiện. Nhược điểm của mô hình này là các hệ thống con không biết khi nào sự kiện được điều khiển. Khi một hệ thống con phát sinh ra một sự kiện, nó không biết những hệ thống con nào đã đăng ký sự kiện đó. Điều này có thể là nguyên nhân gây ra những xung đột khi những kết quả của việc điều khiển sự kiện đã có sẵn.



Hình 3.12. Một mô hình điều khiển dựa trên sự quảng bá có tuyến chọn

b2) Mô hình điều khiển ngắt

Các hệ thống thời gian thực mà yêu cầu các sự kiện sinh ra từ bên ngoài là hệ thống điều khiển bởi sự kiện. Một mô hình điều khiển ngắt được minh họa như trong hình 3.13. Bộ điều khiển ngắt được xác định cho một số kiểu ngắt. Mỗi kiểu ngắt được kết hợp với một vùng nhớ lưu trữ địa chỉ của bộ điều khiển. Khi một kiểu ngắt cụ thể nhận được, bộ chuyển mạch phân cứng tạo ra điều khiển để truyền tức thời sự kiện đó tới bộ điều khiển ngắt. Bộ điều khiển ngắt sẽ khởi động hoặc dừng các tiến trình khác nhau để đáp ứng yêu cầu của sự kiện ngắt nhận được.



Hình 3.13. Một mô hình điều khiển ngắt

Mô hình này chỉ nên được sử dụng trong các hệ thống thời gian thực phức tạp mà ở đó sự đáp ứng tức thời sự kiện là cần thiết. Nó có thể kết hợp với mô hình điều khiển tập trung. Những bộ điều khiển tập trung thường vận hành hệ thống với điều khiển dựa trên ngắt cho các trường hợp khẩn cấp.

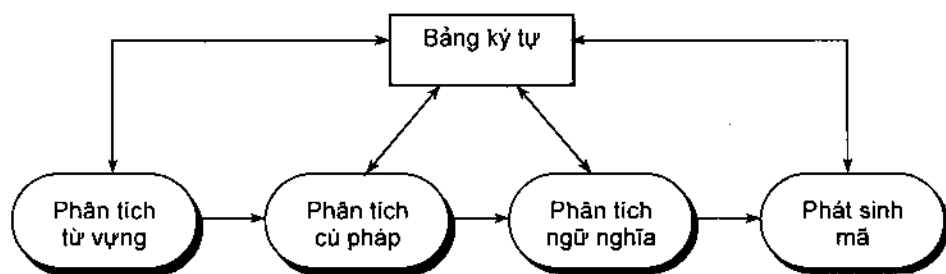
3.2.2.3. Các mô hình kiến trúc miền cụ thể

Những mô hình kiến trúc ở trên là những mô hình tổng quát. Ngoài ra, có những mô hình kiến trúc được xây dựng cho một miền ứng dụng cụ thể. Mặc dù các ứng dụng của một miền khác nhau rất nhiều về chi tiết, nhưng kiến trúc chung có thể được dùng lại khi phát triển những hệ thống mới. Những mô hình kiến trúc loại này được gọi là những *kiến trúc miền cụ thể*.

(*domain specific architecture*). Có hai kiểu mô hình kiến trúc xác định cho miền:

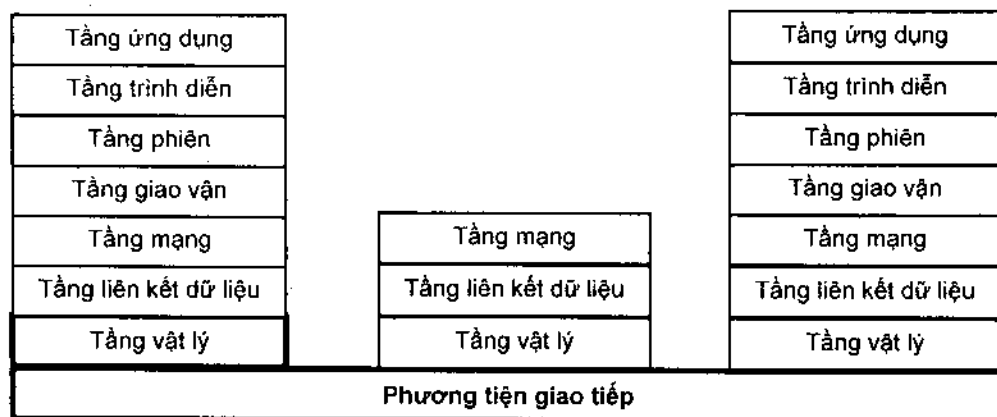
1. *Mô hình tổng quát (generic model)* là sự khái quát hóa từ một số lượng lớn các hệ thống thực cụ thể cùng kiểu. Ví dụ, trong những hệ thống thời gian thực, có những mô hình kiến trúc chung cho những kiểu hệ thống khác nhau như các *hệ thu thập dữ liệu*, các *hệ thống theo dõi*,... Mô hình tổng quát phản ánh kiến trúc của những hệ thống đang tồn tại và có thể dùng lại một cách trực tiếp trong một thiết kế.

Ví dụ mô hình trình biên dịch. Hàng ngàn trình biên dịch đã được tạo ra theo mô hình này. Nó thường bao gồm một số môđun và có thể được tổ chức theo những mô hình kiến trúc khác nhau về chi tiết. Các pha phân tích từ vựng, phân tích cú pháp, phân tích ngữ nghĩa được tổ chức tuần tự như trong hình 3.14.



Hình 3.14. Mô hình luồng dữ liệu của một trình biên dịch

2. *Kiến trúc tham chiếu (reference architectural model)* có mức trừu tượng cao hơn và khái quát hóa một số lớn các hệ thống. Nó cung cấp những thông tin về kiến trúc như những chuẩn để so sánh. Những mô hình tham chiếu thường nhận được từ một sự học tập, khái quát kinh nghiệm của miền ứng dụng. Chúng mô tả một kiến trúc lý tưởng hóa bao gồm tất cả những tính năng mà hệ thống có thể có. Nó được dùng như một cơ sở cho sự thực thi hệ thống. Điều này chính là mục đích của kiến trúc OSI (hình 3.15) do Zimmermann [44] đề xuất dùng cho việc liên kết các hệ thống mở. Nó dự kiến xây dựng như một chuẩn. Nếu một hệ thống phù hợp với mô hình này thì có thể giao tiếp với các hệ thống khác phù hợp với nó.



Hình 3.15. Kiến trúc của mô hình tham chiếu OSI

Trên thực tế không có sự phân biệt rạch ròi giữa hai kiểu mô hình này.

3.2.3. Tiến trình thiết kế kiến trúc

Một hệ thống lớn luôn có thể 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, nó mô tả một kiến trúc của phần mềm. Thiết kế kiến trúc tập trung vào việc *biểu diễn cấu trúc các thành phần phần mềm, các thuộc tính của nó và tương tác giữa chúng.* Các nhà thiết kế khác nhau có thể tiếp cận tiến trình thiết kế kiến trúc theo cách khác nhau. Tuy nhiên, những hoạt động sau đây là chung cho mọi tiến trình:

1. *Cấu trúc hệ thống:* Một hệ thống được cấu trúc thành một số hệ thống con.
2. *Mô hình hóa điều khiển:* Thiết lập một mô hình chung mô tả mối quan hệ điều khiển giữa các phần trong hệ thống.
3. *Phân rã các môđun:* Mỗi hệ thống con được phân rã thành các môđun. Kiến trúc này cần chỉ ra các loại môđun và liên kết giữa chúng.

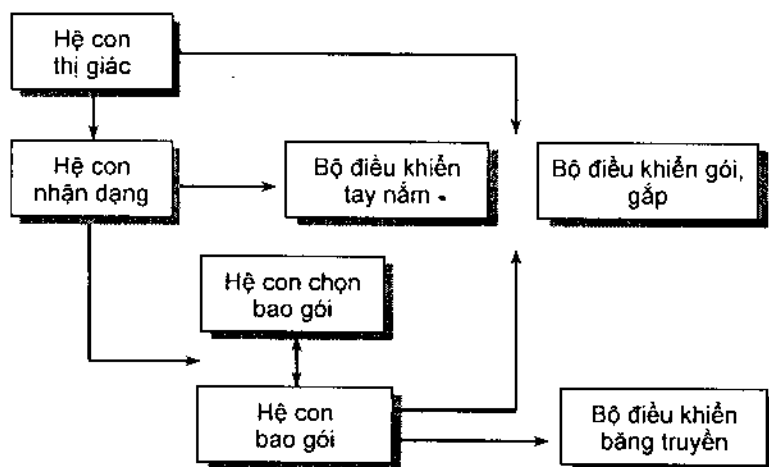
Thiết kế kiến trúc có thể dựa trên một hoặc một số mô hình kiến trúc cụ thể như đã trình bày ở trên. Để sử dụng chúng, việc nhận biết được các mô

hình, những mặt mạnh, mặt yếu của chúng là rất quan trọng. Trên thực tế, các hệ thống lớn không phù hợp với một mô hình kiến trúc đơn. Các phần của nó có thể được thiết kế từ các mô hình kiến trúc khác nhau.

3.2.3.1. Cấu trúc hệ thống

Giai đoạn đầu tiên của tiến trình thiết kế kiến trúc thường liên quan đến sự phân rã một hệ thống thành một tập các hệ thống con tương tác với nhau. Ở mức độ trừu tượng nhất, kiến trúc hệ thống được mô tả như một biểu đồ khối với các hộp đại diện là các hệ thống con. Các hộp này lại có thể phân thành các hệ thống con của nó. Các mũi tên chỉ ra luồng dữ liệu hay điều khiển từ một hệ con đến hệ con khác. Hình 3.16 là một phác thảo mô hình kiến trúc được cấu trúc từ hệ thống rôbot bao gói. Nó sử dụng một *hệ con thị giác* để nhận biết đối tượng trên băng truyền, *nhận dạng kiểu của đối tượng* và *chọn một loại bao gói*. Sau đó nó chuyển đối tượng từ băng truyền lên giá để *bao gói*. Bao gói xong, đối tượng được *đặt lên một băng truyền* khác để chuyển đi. Thông qua mô tả nghiệp vụ của hệ thống, và bằng tư duy, ta có thể cấu trúc một mô hình kiến trúc phác thảo của nó. Sau đó, bằng cách so sánh với các kiến trúc đã biết để tiếp tục hoàn thiện cho đến khi nhận được mô hình kiến trúc tốt nhất.

Trên thực tế, khi sử dụng các tiến trình phát triển phần mềm mang tính hệ thống cao (như *RUP* hay *tiến trình phát triển hướng cấu trúc* [37]), việc cấu trúc mô hình kiến trúc của hệ thống được thực hiện từ chính các bước phát triển của các tiến trình đó.



Hình 3.16. Biểu đồ khối của hệ thống điều khiển rôbot bao gói

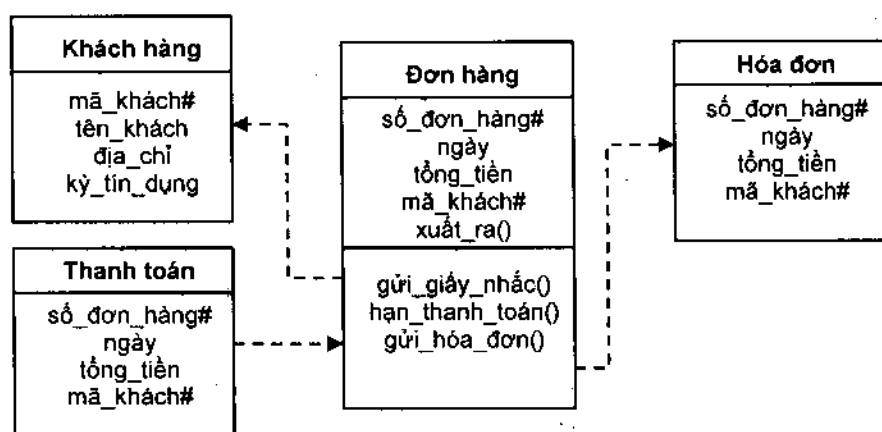
3.2.3.2. Phân rã hệ con thành môđun

Sau khi một kiến trúc đã được thiết kế, giai đoạn tiếp theo của tiến trình thiết kế kiến trúc là phân rã các hệ con thành môđun. Các cấp độ khác nhau của sự phân rã có thể là một phần của tiến trình thiết kế kiến trúc. Hai mô hình thường được sử dụng khi phân rã một hệ thống con thành các môđun là:

1. *Mô hình hướng đối tượng (object – oriented model)*: Hệ thống được phân rã thành một tập các đối tượng giao tiếp với nhau.
2. *Mô hình luồng dữ liệu (data – flow model)*: Hệ thống được phân rã thành các môđun chức năng cho phép chuyển đổi dữ liệu vào thành dữ liệu ra của nó. Phương thức này cũng được gọi là một cách tiếp cận đường ống (pipeline).

a) Các mô hình kiến trúc hướng đối tượng

Một mô hình kiến trúc hướng đối tượng mức môđun được cấu thành từ một tập các lớp đối tượng ghép nối lỏng lẻo với nhau bằng những giao diện được xác định tốt. Các đối tượng gọi dịch vụ từ các đối tượng khác. Hình 3.17 là một ví dụ về mô hình kiến trúc của một hệ thống xử lý hóa đơn. Sự cấu trúc mô hình hướng đối tượng liên quan tới việc xác định các lớp đối tượng, các thuộc tính, toán tử và các tương tác giữa chúng. Trong tiến trình RUP, mô hình lớp nhận được do kết quả của việc chuyển các mô hình cộng tác khái niệm sang biểu đồ lớp. Việc hoàn thiện tiếp tục mô hình kiến trúc các lớp được tiến hành trên cơ sở vận dụng rộng rãi các mẫu thiết kế sẵn có.

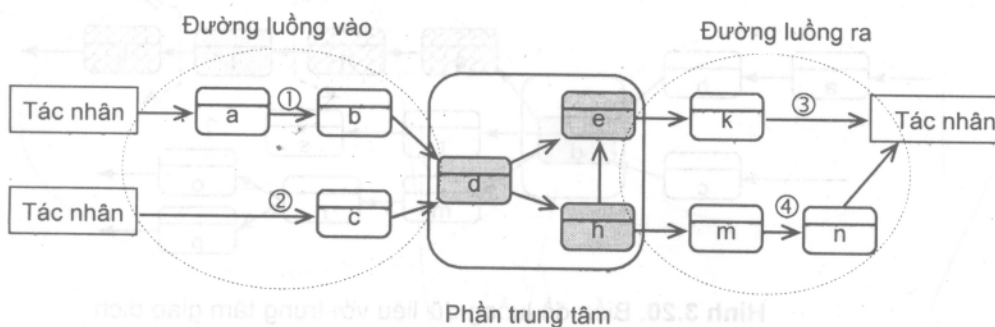


Hình 3.17. Mô hình hướng đối tượng của một hệ thống xử lý hóa đơn

b) Ánh xạ mô hình luồng dữ liệu sang cấu trúc môđun

Trong một mô hình luồng dữ liệu, nếu tiến trình có luồng dữ liệu vào và ra chỉ liên quan đến tác nhân ngoài và kho dữ liệu, thì việc chuyển nó thành một môđun không có khó khăn gì. Khi có nhiều tiến trình liên kết với nhau, nhưng chỉ có một số tiến trình có luồng dữ liệu vào và ra liên quan với tác nhân ngoài, việc biểu diễn nó lúc này cần có phương pháp. Stevens, Myer và Constantine [35] từ sớm đã trình bày thiết kế phần mềm dựa trên sự chuyển đổi luồng dữ liệu thành cấu trúc hệ thống môđun. Dưới đây là một số khái niệm liên quan đến phương pháp này:

Đường luồng đơn là một dãy liên tiếp các tiến trình mà mỗi tiến trình đều có một luồng dữ liệu vào và một luồng dữ liệu ra (bỏ qua kho dữ liệu). Nếu luồng dữ liệu đầu tiên của đường đó có nguồn là "tác nhân ngoài" thì gọi là *đường luồng vào*. Nếu luồng dữ liệu cuối cùng có đích là "tác nhân ngoài" thì gọi là *đường luồng ra* (hình 3.18). Ở đây "tác nhân ngoài" được hiểu là phần không thuộc phần biểu đồ luồng dữ liệu được xét.

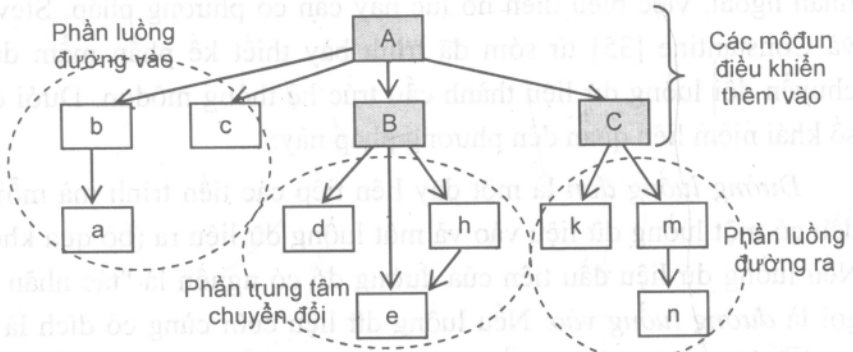


Hình 3.18. Biểu đồ luồng dữ liệu với đường luồng ra, đường luồng vào, trung tâm

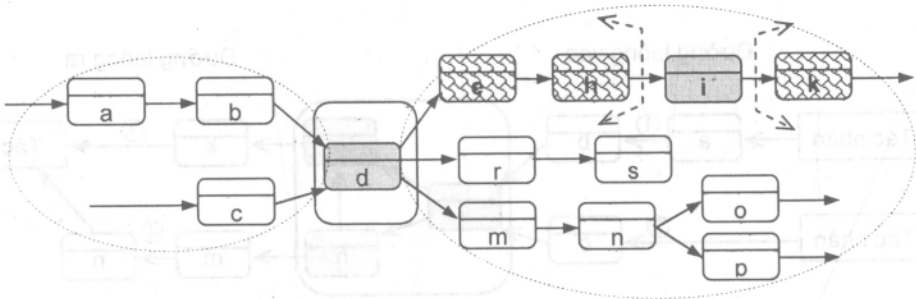
Tiến trình ánh xạ luồng dữ liệu thành cấu trúc môđun gồm các bước sau:

1. Xác định các đường luồng vào và ra để cô lập phần trung tâm.
2. Kiểm tra phần trung tâm xem thuộc loại nào ?
 - Nếu phần trung tâm là một "*trung tâm chuyển đổi*", tức là các luồng ra khỏi trung tâm ((3), (4)) khác với luồng vào trung tâm ((1), (2)) thì thêm một môđun gốc và hai môđun con tương ứng với hai phần (phần trung tâm, các đường ra), còn mỗi đường vào sẽ tạo một cây môđun tương ứng (xem hình 3.19).

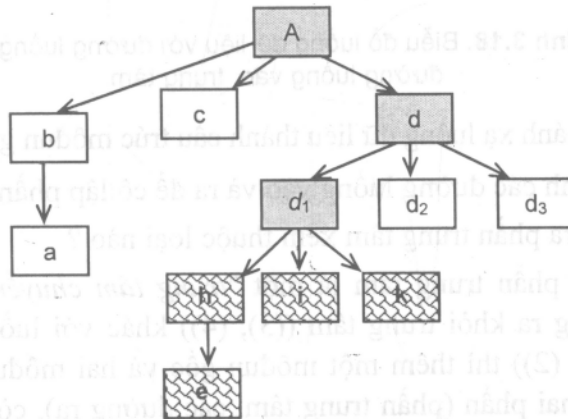
- Nếu phần trung tâm là một "trung tâm giao dịch" (hình 3.20), tức là mỗi luồng ra khỏi trung tâm ((3), (4)) trùng với một luồng vào trung tâm ((1), (2)) thì thêm một môđun gốc và một môđun con là tiến trình thuộc trung tâm giao dịch, và mỗi đường vào và mỗi đường ra tương ứng với một cây môđun sẽ xây dựng (hình 3.21).



Hình 3.19. Cấu trúc môđun nhận được từ biểu đồ 3.18



Hình 3.20. Biểu đồ luồng dữ liệu với trung tâm giao dịch



Hình 3.21. Cấu trúc môđun nhận được từ biểu đồ 3.20

3. Với mỗi phần được tách ra từ biểu đồ luồng dữ liệu trên xem như một biểu đồ luồng dữ liệu độc lập và lặp lại các bước trên cho đến khi ánh xạ được mọi tiến trình vào các mô đun.

Thông thường trung tâm giao dịch chỉ gồm một tiến trình. Tùy thuộc khả năng phân tích, việc nhận ra các đường luồng vào, đường luồng ra có thể khác nhau, và vì vậy phần trung tâm cũng có thể khác nhau. Thêm nữa, mô hình luồng dữ liệu dùng cho tiến trình ánh xạ có thể khác nhau. Thông thường, các biểu đồ đó sẽ là các biểu đồ luồng dữ liệu ở mức sơ cấp. Trong trường hợp không được như vậy, trọng quá trình ánh xạ có thể phải tiếp tục phân rã phần biểu đồ luồng dữ liệu nhận được trước khi thực hiện bước tiếp của tiến trình ánh xạ.

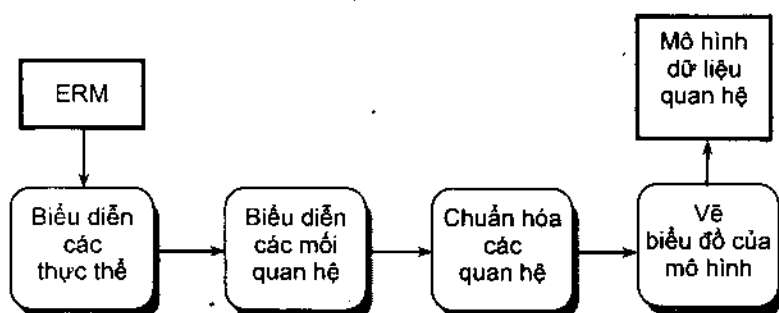
3.3. Thiết kế hệ thống hướng chức năng

Thiết kế hệ thống hướng chức năng là một giai đoạn của chiến lược phát triển định hướng chức năng. Đầu vào cho hoạt động thiết kế là các mô hình nhận được ở giai đoạn phân tích (mục 2.3.3). Tiến trình thiết kế bắt đầu với thiết kế logic, tiếp sau là thiết kế vật lý. Thiết kế xử lý được tiến hành với các luồng dữ liệu sơ cấp.

3.3.1. Thiết kế dữ liệu

3.3.1.1. Thiết kế mô hình dữ liệu logic

Đầu vào của thiết kế dữ liệu logic là mô hình thực thể – mối quan hệ. Tiến trình thiết kế dữ liệu logic được mô tả ở hình 3.22.



Hình 3.22. Tiến trình thiết kế dữ liệu logic

Hình 3.23 là mô hình dữ liệu logic nhận được khi thiết kế sử dụng đầu vào là ERM cho ở hình 2.17 và có các quan hệ sau:

- VÉ (số_vé)

XE (số_xe)

LOẠI_XE (tên_loại, số_chỗ, giá_gửi)

NHẬN_XE (số_vé, số_xe, ngày_vào, giờ_vào)

TRẢ_XE (số_phiếu, số_vé, số_xe, ngày_ra, giờ_ra)

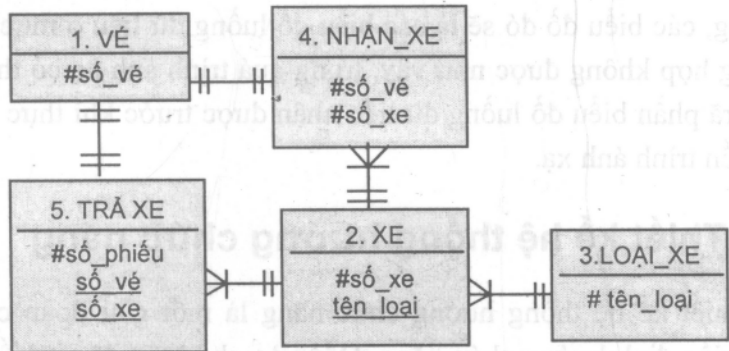
(1)

(2)

(3)

(4)

(5)



Hình 3.23. Biểu đồ liên kết của mô hình dữ liệu quan hệ

3.3.1.2. Thiết kế cơ sở dữ liệu vật lý

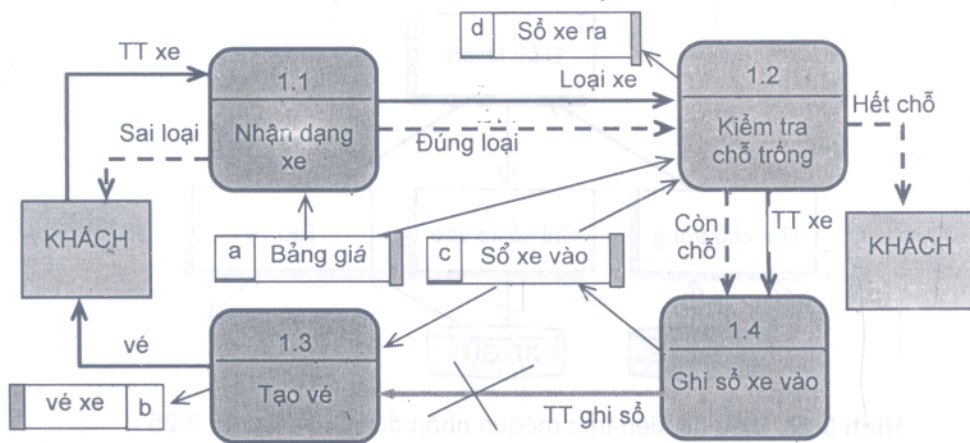
Trên cơ sở của mô hình quan hệ nhận được, lựa chọn hệ quản trị cơ sở dữ liệu, tiến hành phi chuẩn hóa để được số các quan hệ ít nhất, nhưng vẫn đủ thông tin và có cấu trúc tốt. Tiếp theo cần chọn hệ quản trị cơ sở dữ liệu và thiết kế các tệp, trong đó các tệp chính (master file) nhận được từ các quan hệ đã phi chuẩn hóa. Sau khi phi chuẩn hóa các quan hệ trên, ta được hai quan hệ dưới đây.

- LOẠI_XE (loại_xe, đơn_giá, số_chỗ)
- XE_GỬI (số_vé, số_xe, ngày_giờ_vào, ngày_giờ_ra, thành_tiền, số_phiếu)
- Việc chuyển các quan hệ thành tệp vật lý sẽ không trình bày ở đây.

3.3.2. Thiết kế xử lý

3.3.2.1. Thiết kế xử lý logic

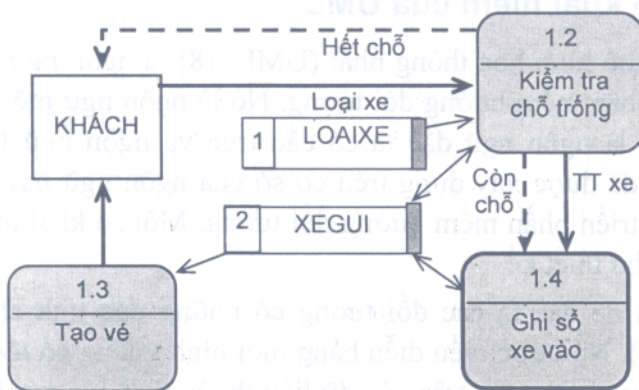
Xuất phát từ luồng dữ liệu vật lý, khi bỏ đi các yếu tố vật lý và cấu trúc lại để mô hình vẫn đảm bảo thực hiện các chức năng. Biểu đồ logic hình 3.24 nhận được từ biểu đồ luồng dữ liệu vật lý hình 2.19.



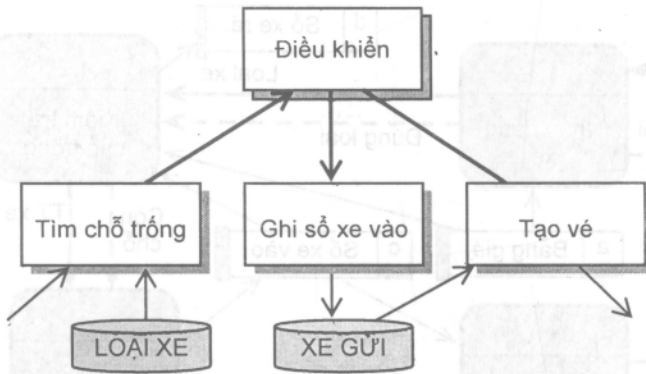
Hình 3.24. Biểu đồ luồng dữ liệu logic

3.3.2.2. Thiết kế kiến trúc mô đun

Trước hết ta cần xác định *luồng hệ thống* từ biểu đồ luồng dữ liệu bằng cách chọn các tiến trình máy thực hiện và phương thức thực hiện (theo lô hay trực tuyến), thay các kho dữ liệu liên quan đến chúng bằng các tệp tương ứng. Hình 3.25 là biểu đồ luồng hệ thống nhận được từ biểu đồ logic hình 3.24. Khi ánh xạ sang kiến trúc mô đun theo phương pháp đã trình bày ở mục 3.2.3.2, ta được kiến trúc mô đun của thành phần chương trình "nhận xe" (hình 3.26).



Hình 3.25. Biểu đồ luồng dữ liệu hệ thống



Hình 3.26. Biểu đồ kiến trúc mô đun nhận được từ biểu đồ 3.25

3.3.3. Các ưu, nhược của thiết kế hướng cấu trúc

Thiết kế hướng cấu trúc đã phát triển hơn ba mươi năm nay, nên các phương pháp thiết kế cũng như những công cụ trợ giúp đã đạt đến mức hoàn thiện, đặc biệt là có nhiều hệ quản trị cơ sở dữ liệu mạnh (như ORACLE, SQLServer,...) trợ giúp việc lưu trữ và khai thác đã đạt mức độ tự động hóa cao. Vì thế, nó rất thích hợp với các bài toán xử lý trên các dữ liệu có thể mô tả ở dạng bảng. Tuy nhiên, nó có hạn chế với các bài toán dữ liệu đa dạng và đòi hỏi sự tương tác của nhiều đối tượng khác nhau. Đặc biệt, do hệ thống sử dụng cơ sở dữ liệu dùng chung nên khó sử dụng lại và sai sót ở một số bộ phận có thể lan truyền sang các phần khác của hệ thống.

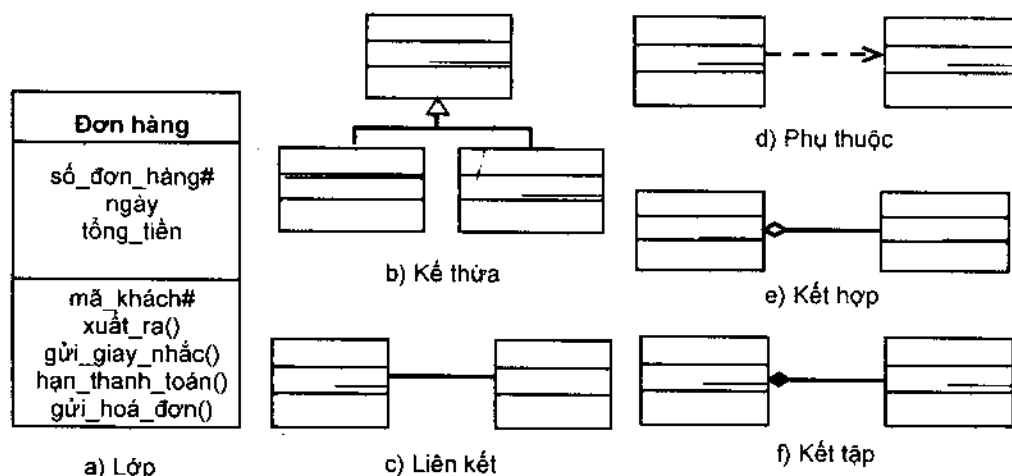
3.4. Thiết kế hệ thống hướng đối tượng

3.4.1. Một số khái niệm của UML

Ngôn ngữ mô hình hóa thống nhất (UML) [8] là ngôn ngữ chuẩn dùng cho phát triển phần mềm hướng đối tượng. Nó là ngôn ngữ mô hình hóa đồ họa, trực quan, là ngôn ngữ đặc tả có cấu trúc và ngôn ngữ làm tài liệu. Nhiều công cụ đã được xây dựng trên cơ sở của ngôn ngữ này để trợ giúp hoạt động phát triển phần mềm hướng đối tượng. Một số khái niệm sau đây của UML cần cho thiết kế.

Lớp (class) để đặc tả các đối tượng có những đặc tính chung (thuộc tính, hành động). Nó được biểu diễn bằng một hình vuông có tên, các thuộc tính, và các toán tử thao tác trên các dữ liệu thuộc tính của nó (hình 3.27a). Một lớp có thể kế thừa từ một hay một vài lớp cha các thuộc tính và toán tử (hình 3.27b).

Các lớp có thể có một số quan hệ với nhau (hình 3.27): *mối quan hệ phụ thuộc*, *mối quan hệ kế thừa*, *mối quan hệ kết hợp* (giữa toàn thể và bộ phận) hay *mối quan hệ kết tập* (đối tượng và thành phần cấu thành nó). UML đã cung cấp các loại biểu đồ lớp với tất cả các mối quan hệ trên cho phép ta có thể đặc tả các mô hình khác nhau của hệ thống hướng đối tượng.



Hình 3.27. Lớp và các mối quan hệ giữa các lớp

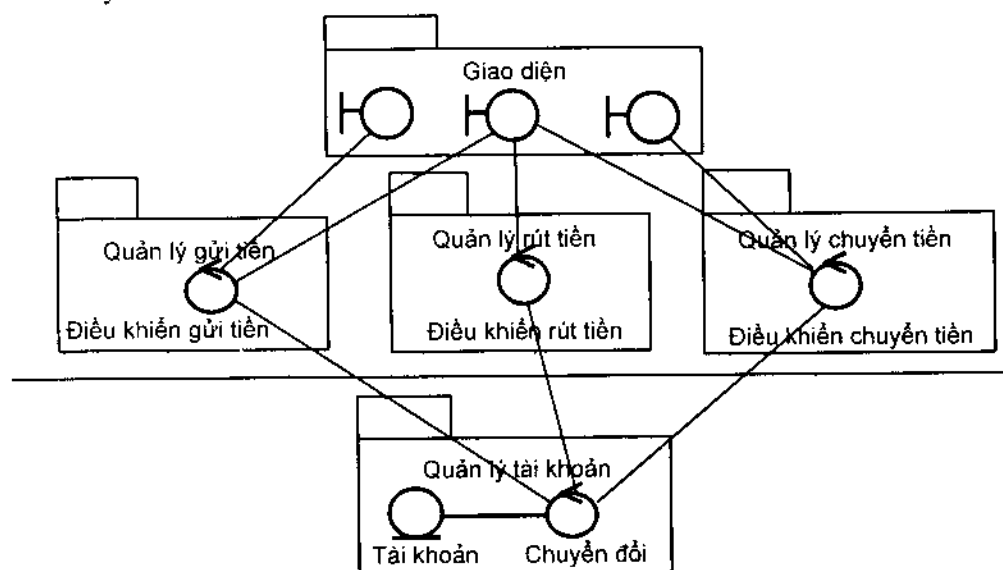
3.4.2. Tiến trình thiết kế hướng đối tượng

Thiết kế hệ thống hướng đối tượng là một phần của chiến lược phát triển định hướng đối tượng. Đầu vào cho hoạt động thiết kế là các mô hình nhận được ở giai đoạn phân tích (xem mục 2.3.3). Tiến trình thiết kế hướng đối tượng gồm các bước tương tự các bước đã nêu ở hình 3.3. Tuy nhiên, đây là quá trình lặp lấy kiến trúc làm trung tâm, cho nên cần sắp thứ tự các gói phân tích để đảm bảo tính chất này. Sau đó mỗi bước lặp được tiến hành cho các ca sử dụng liên quan đến từng gói được chọn. Tiến trình thiết kế có thể mô tả khái quát như sau:

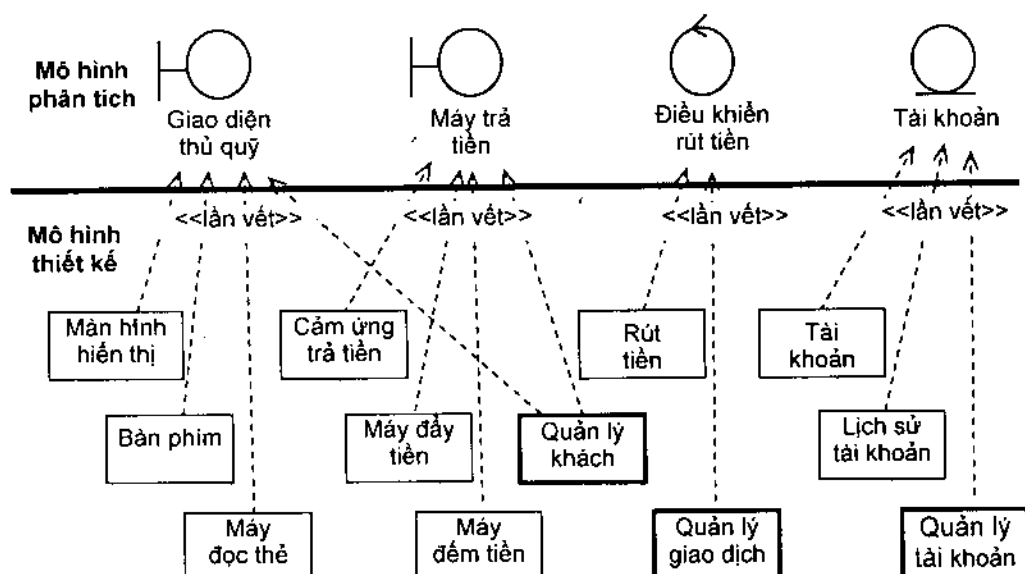
- Xác định kiến trúc của hệ thống (hình 3.28);
- Sắp thứ tự ưu tiên các gói;
- Với mỗi gói, thiết kế cho mỗi ca sử dụng thuộc gói bằng cách xác định các lớp thiết kế tham gia triển khai các lớp phân tích (hình 3.29);
- Xây dựng biểu đồ tương tác giữa các lớp (hình 3.30);
- Thiết kế chi tiết các lớp (hình 3.31).

- Phân tích và hoàn thiện biểu đồ lớp dựa trên các mẫu thiết kế.

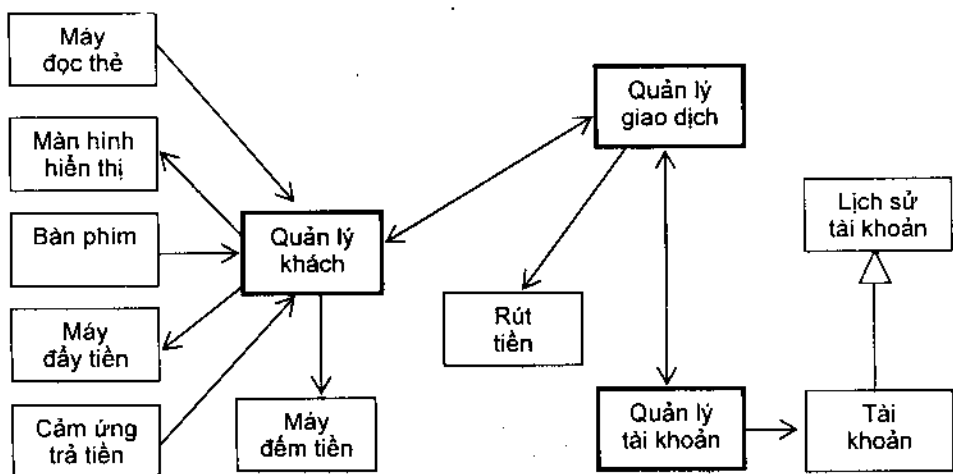
Khi sử dụng mô hình phân tích của ví dụ ở mục 2.3.3.3, thực hiện các bước thiết kế với ca sử dụng *rút tiền*, ta có các kết quả thể hiện qua mô hình dưới đây:



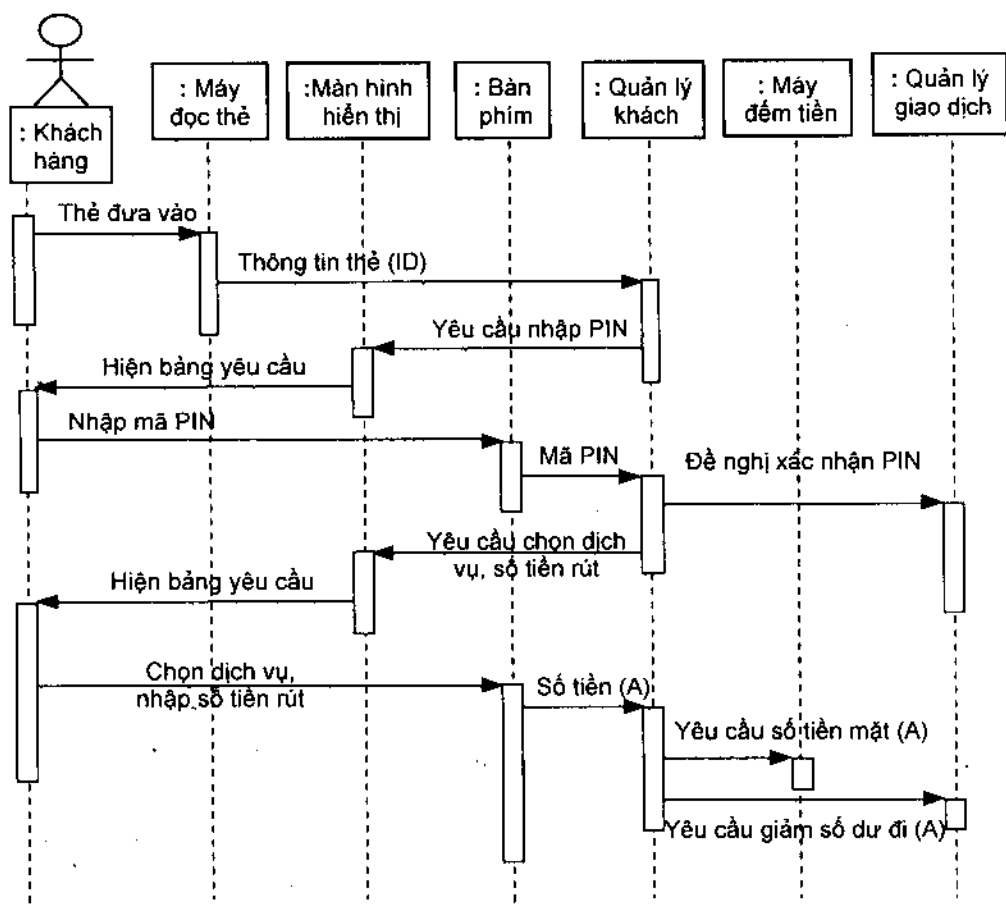
Hình 3.28. Mô hình kiến trúc hệ thống về sự phụ thuộc giữa các gói



Hình 3.29. Các lớp thiết kế nhận được từ các lớp phân tích khi làm mịn



Hình 3.30. Biểu đồ lớp thiết kế tham gia thực hiện ca sử dụng rút tiền



Hình 3.31. Biểu đồ tuần tự lớp thiết kế thực hiện ca sử dụng rút tiền

Tài_khoản
số_tài_khoản: <i>string</i> tên_khách: <i>string</i> số_dư: <i>money</i>
+ 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 3.32. Lớp tài khoản với các thuộc tính và các thao tác của nó

Để đơn giản, ở đây bỏ qua một số bước không quan trọng trong tiến trình, như thiết kế giao diện, thiết kế bố trí phần mềm trong không gian, biểu đồ kiến trúc mô đun các lớp hay các mô tả các ràng buộc phi chức năng mà các lớp phải tuân thủ khi cung cấp dịch vụ.

3.4.3. Các ưu điểm của thiết kế hướng đối tượng

- Thiết kế hướng đối tượng dễ bảo trì vì có thể hiểu và cải biên đối tượng như một thực thể độc lập. Thay đổi trong thực hiện một đối tượng hoặc thêm các dịch vụ cho nó sẽ không làm ảnh hưởng tới các đối tượng khác của hệ thống.
- Các đối tượng là các thành phần có thể dùng lại được.
- Có một vài lớp hệ thống phản ánh một quan hệ rõ ràng giữa các thực thể có thực (chẳng hạn như các thành phần phần cứng) với các đối tượng điều khiển nó trong hệ thống. Điều này làm cho thiết kế có được tính dễ hiểu.

Nhược điểm của thiết kế hướng đối tượng là không dễ dàng nhận ra các đối tượng của một hệ thống. Vì rằng, cách nhìn tự nhiên nhiều hệ thống là cách nhìn chức năng.

3.5. Thiết kế hệ thống thời gian thực

3.5.1. Khái niệm về hệ thời gian thực

Ngày nay, máy tính được sử dụng để điều khiển một số lượng ngày càng lớn các hệ thống, từ các thiết bị gia dụng đơn giản đến toàn bộ một nhà

máy chế tạo tự động. Phần mềm hệ thống này phải phản ứng đủ nhanh để đáp ứng được các sự kiện sinh ra từ phần cứng và đưa ra các tín hiệu điều khiển cho các bộ công tác hoạt động. Phần mềm này được gọi là *hệ thống thời gian thực nhúng*, vì nó được nhúng trong một vài hệ thống phần cứng lớn hơn.

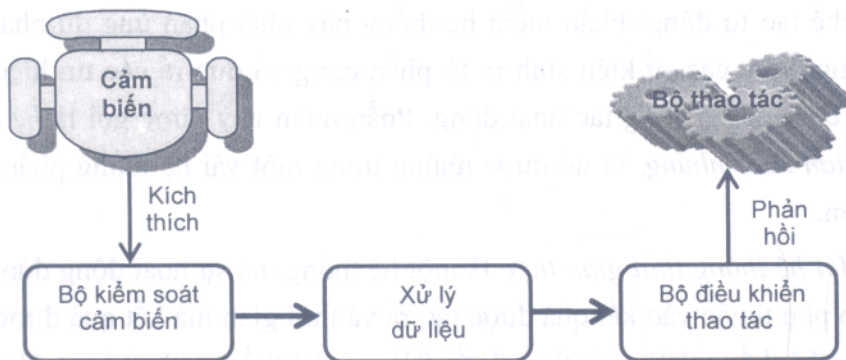
Một hệ thống thời gian thực là một hệ thống mà sự hoạt động đúng đắn của nó phụ thuộc vào kết quả được tạo ra và thời gian mà kết quả được xuất ra. Một *hệ thống thời gian thực "mềm"* là một hệ thống mà sự hoạt động của nó không bị suy biến khi kết quả sinh ra không phù hợp với thời gian yêu cầu. Trong khi đó, một *hệ thống thời gian thực "cứng"* thì hoạt động của nó không còn đúng đắn nếu kết quả xuất ra vượt quá yêu cầu thời gian đã định.

Một hệ thời gian thực có thể xem như một *hệ kích thích/đáp ứng*. Nhận một kích thích nào đó thì hệ phải xuất ra một đáp ứng thích hợp. Có hai loại kích thích:

1. *Kích thích định kỳ*: kích thích sinh ra sau những khoảng thời gian đều.
2. *Kích thích không định kỳ*: kích thích xảy ra bất thường.

Nói chung các kích thích định kỳ của hệ thống thời gian thực được một đồng hồ hay tập các cảm biến (sensor) gắn với hệ thống sinh ra. Các đáp ứng với các kích thích được hướng tới các bộ sinh kích thích, các bộ này làm cho các đơn vị phần cứng thao tác. Các kích thích không định kỳ có thể được các bộ sinh kích thích hoặc cảm biến nhận biết từ điều kiện ngoại lai tạo ra mà hệ thống phải xử lý. Nhìn chung, phần mềm thời gian thực bao gồm:

- *Một thành phần thu thập dữ liệu (cảm biến)* để thu thập và định dạng thông tin từ môi trường ngoài.
- *Một thành phần phân tích* để biến đổi thông tin theo yêu cầu của ứng dụng, và *điều phối* các thành phần khác để duy trì việc đáp ứng thời gian thực.
- *Một thành phần kiểm soát/đáp ứng* để đáp ứng với môi trường ngoài.



Hình 3.33. Tiến trình vận hành hệ thời gian thực

Một dạng kiến trúc khác của hệ thống thời gian thực là xem nó được cấu thành từ những tiến trình đơn, và mỗi tiến trình đơn được gắn với một cặp *kích thích/dáp ứng* (hình 3.33). Bài toán nảy sinh là phải phối hợp các tiến trình đơn như thế nào để mọi cặp *kích thích/dáp ứng* đều được thỏa mãn. Bài toán này được giải quyết ở bộ *quản lý và điều phối các bộ thao tác*.

3.5.2. Tiến trình thiết kế hệ thống thời gian thực

Trong một hệ thống nhúng lớn, giai đoạn đầu tiên của quá trình thiết kế là thiết kế hệ thống: các chức năng hệ thống được phân thành các chức năng cho phần cứng và cho phần mềm. Vai trò của phần mềm là để phối hợp và khống chế các đơn vị phần cứng. Ngày nay, ranh giới giữa phần cứng và phần mềm không còn rõ ràng. Khi quyết định một thực thi bằng phần cứng có nghĩa là thiết kế phần mềm phải được cấu trúc để thích nghi được với tình huống này và có thể tính đến sự thi hành đó bằng một vài cách. Điều đó cũng có nghĩa là, kiến trúc hệ thống phải được cấu trúc từ các thành phần độc lập để có thể lựa chọn các sự thực thi khác nhau. Đó cũng là mục tiêu của nhà thiết kế mong muốn thiết kế một hệ thống bảo trì được, tức là có nhiều giải pháp có thể thay thế nhau. Vì vậy, trị hoãn quyết định phân chia trách nhiệm giữa phần cứng và phần mềm nên tiến hành càng muộn càng tốt.

Quá trình thiết kế một hệ thống nhúng thường bao gồm các bước sau:

1. Xác định các tác nhân kích thích mà hệ phải xử lý và đáp ứng.

2. Đối với mỗi kích thích và mỗi đáp ứng thích hợp, hãy xác định ràng buộc thời gian cho cả việc tiếp nhận kích thích và việc xử lý đáp ứng.
3. Phân tích các kích thích và quá trình xử lý đáp ứng thành một số các tiến trình song song. Một mô hình khái quát tốt cho kiến trúc đó là kết hợp việc xử lý mỗi lớp các kích thích với các đáp ứng.
4. Với mỗi cặp *kích thích/đáp ứng*, hãy thiết kế các thuật toán cho các tính toán yêu cầu. Ở giai đoạn này cần chỉ ra khối lượng xử lý cần thiết và thời gian yêu cầu để hoàn thành xử lý đó.
5. Thiết kế một hệ thống lập lịch bảo đảm các quá trình được bắt đầu ở đúng thời điểm thích hợp để tiến hành việc xử lý được yêu cầu trong điều kiện các ràng buộc về thời gian đã xác định.
6. Tích hợp hệ thống dưới sự điều khiển của một bộ điều phối thời gian thực.

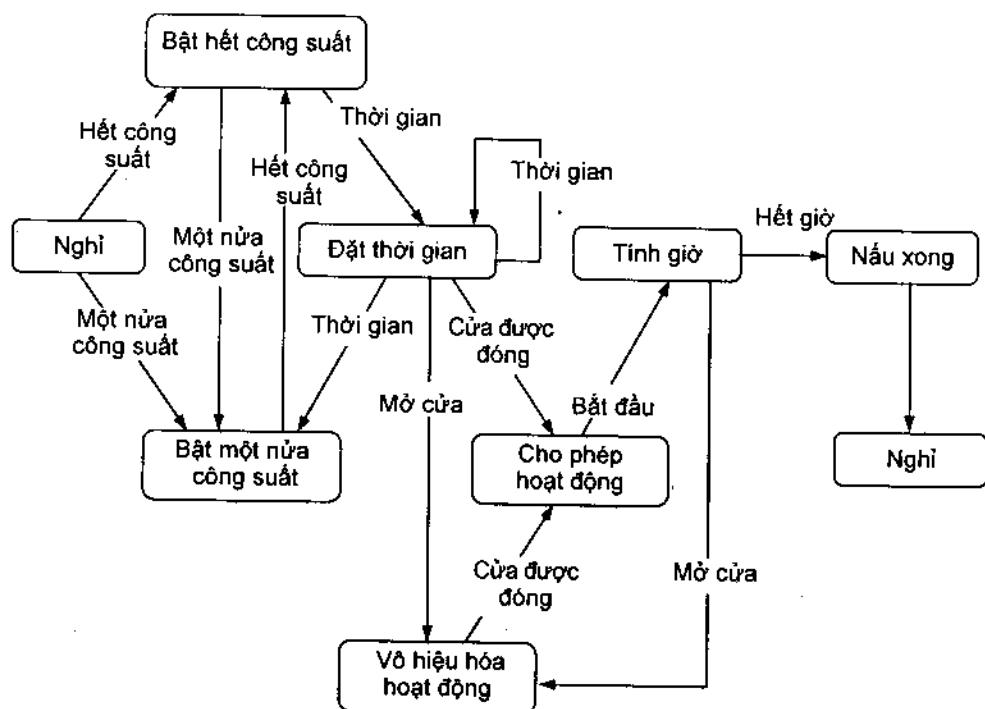
3.5.3. Mô hình hóa máy trạng thái

Các hệ thời gian thực phải đáp ứng với các sự kiện xuất hiện một cách không định kỳ. Các sự kiện này thường là nguyên nhân làm cho hệ thống chuyển tới trạng thái khác. Vì thế, việc mô hình hóa máy trạng thái được sử dụng rộng rãi cho việc thiết kế các hệ thời gian thực.

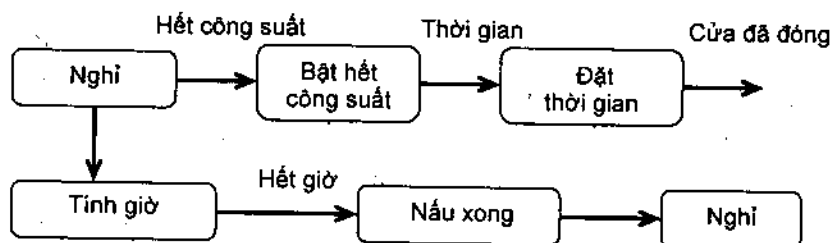
Salter (1976) đã nói tới việc dùng các máy trạng thái hữu hạn để mô hình hóa hệ thống và xét một mô hình hệ thống khái quát là một bộ chức năng của ba phần tử: *khống chế, hàm, dữ liệu*. Cách tiếp cận máy trạng thái vì thế là thích hợp cho việc mô hình hóa các hệ thống thời gian thực khi mà một *cái vào* gây ra một chuỗi các hành vi được kích hoạt. Đối với các hệ lớn, số lượng các trạng thái được nhận biết tăng rất nhanh. Giải pháp cho vấn đề này là không đưa tất cả các trạng thái vào một sơ đồ duy nhất. Điều đó có thể đạt được nhờ sử dụng các biểu đồ chuỗi riêng rẽ chỉ ra một dãy các hành vi được thực hiện xâu chuỗi qua suốt hệ thống như thế nào. Biểu đồ xâu chuỗi được xuất ra cho mọi tổ hợp nhờ thông báo.

Hình 3.34 là mô hình máy trạng thái một lò vi sóng: các hình *chữ nhật góc tròn* (□) biểu diễn các trạng thái của hệ thống, *các mũi tên* (→) biểu

diễn kích thích gây ra một sự chuyển trạng thái. Các tên được chọn trong biểu đồ máy trạng thái mô tả và đưa ra sự chỉ dẫn cho sự hoạt động. Tuy nhiên, trong một thiết kế hoàn thiện, cần cung cấp thêm các chi tiết về cả kích thích và các trạng thái của hệ thống. Mô hình trạng thái lò vi sóng được đơn giản hóa nhưng vẫn là một biểu đồ tương đối phức tạp. Để đơn giản ta đưa ra biểu đồ xâu chuỗi cho mỗi kích thích xuyên suốt hệ thống (hình 3.35). Mô hình máy trạng thái là một cách tốt để biểu diễn thiết kế một hệ thời gian thực và độc lập với ngôn ngữ. Vì thế nó là một phần của các phương pháp thiết kế hệ thời gian thực.



Hình 3.34. Mô hình mẫu máy trạng thái của một lò vi sóng



Hình 3.35. Chuỗi thông điệp từ mô hình hệ thống lò vi sóng.

3.5.4. Bộ điều phối thời gian thực

Các bộ điều phối thời gian thực chịu trách nhiệm quản lý quá trình và định vị tài nguyên trong một hệ thời gian thực. Một bộ điều phối thời gian thực là tương tự với hệ điều hành trong các máy tính với mục đích khái quát. Các thành phần của nó phụ thuộc vào kích cỡ và độ phức tạp của hệ thời gian thực được phát triển.

Đối với các hệ thống cần cung cấp một dịch vụ liên tục với độ tin cậy cao thì bộ thi hành cần phải bao gồm thêm một *bộ quản lý cấu hình* (để thay đổi phần cứng) và một *bộ quản lý lỗi* (để phát hiện và khắc phục lỗi).

Thông thường, các kích thích được xử lý trong hệ thời gian thực tùy mức độ quan trọng của nó mà có mức ưu tiên khác nhau, thường có hai mức ưu tiên:

1. *Mức ngắt*. Đây là mức ưu tiên cao nhất, tại đó một số tiến trình đang hoạt động phải ngừng lại dành cho các tiến trình cần ưu tiên để đáp ứng rất nhanh với một số hoạt động được đòi hỏi (trong số đó có tiến trình bấm giờ).
2. *Mức đồng hồ*. Đây là mức ưu tiên dành cho các tiến trình tuần hoàn sau những thời gian nhất định (được đồng hồ đếm) bắt đầu khởi động lại.

Các loại tiến trình khác nhau có thể được gán các mức ưu tiên khác nhau. Ví dụ, có thể có vài luồng ngắt. Việc chỉ rõ các mức ưu tiên xử lý làm cho tất cả các tiến trình đều được phục vụ đúng lúc. Tất cả các tiến trình này được lập lịch để thực hiện khi khả năng xử lý đã sẵn sàng.

3.5.5. Các hệ giám sát và điều khiển

Các hệ giám sát và khống chế là một lớp quan trọng của các hệ thời gian thực. Nó liên tục kiểm tra các cảm biến là nguồn cung cấp các thông tin về môi trường và thực hiện các hành động tùy thuộc vào bộ đọc cảm biến. Hệ giám sát thực hiện một hành động khi phát hiện một vài giá trị cảm biến là ngoại lệ. Hệ điều khiển liên tục bộ thao tác phản ứng tùy thuộc vào giá trị của các cảm biến gắn kết với nó. Thiết kế hệ này dựa trên sự phân tích cụ thể từng cặp kích thích/đáp ứng của thiết bị giám sát và thiết lập thời gian đáp ứng xuất ra cho phù hợp yêu cầu.

3.5.6. Các hệ thu nhận dữ liệu

Các hệ thu nhận dữ liệu là một lớp khác của các hệ thời gian thực. Các hệ này thu thập dữ liệu từ các cảm biến cho việc phân tích và xử lý nối tiếp nhau.

3.6. Thiết kế giao diện người – máy tính

3.6.1. Vai trò và tiến hóa giao diện người – máy tính

Ngày nay, khi có hàng triệu người trực tiếp sử dụng máy tính, thì giao diện người – máy tính chính là phương tiện thiết lập lên một cơ chế, qua đó người dùng có thể giao tiếp và sử dụng được hệ thống máy tính. Một giao diện khó sử dụng gây nhiều khó khăn và dễ tạo ra sai lầm của người sử dụng. Trong trường hợp bình thường, mức lỗi gặp phải có thể rất cao, làm người dùng không muốn sử dụng hệ thống nữa. Trong trường hợp tồi tệ nhất nó có thể làm cho hệ thống trở nên vô dụng. Chính vì vậy, giao diện của các hệ thống tương tác thường được xem là *tiêu chuẩn để đánh giá hệ thống tốt hay tốt*. Một hệ thống có giao diện tốt là một hệ thống thành công.

Giao diện tương tác người – máy đã tiến hóa qua nhiều thế hệ:

- *Thế hệ thứ nhất*: giao diện chỉ lệnh và hỏi.
- *Thế hệ thứ hai*: giao diện thực đơn (thực đơn) đơn giản.
- *Thế hệ thứ ba*: giao diện đồ họa. Ngày nay giao diện người dùng đồ họa (*graphical user interface – GUI*) đã quen thuộc với nhiều độc giả.

Ưu điểm giao diện người dùng đồ họa: Dễ học, dễ sử dụng, có nhiều cửa sổ cho phép hiển thị đồng thời nhiều kiểu thông tin khác nhau. Người dùng có thể thực hiện đồng thời nhiều công việc, có thể chuyển cảnh mà không mất mối liên hệ trực quan với công việc khác. Việc dùng biểu tượng đồ họa, thực đơn kéo thả, nút và các kỹ thuật khác làm giảm số lượng thao tác, tăng hiệu quả tương tác cho người dùng.

3.6.2. Các nguyên tắc thiết kế giao diện người – máy tính

Giao diện người dùng được xem là yếu tố quyết định sự thành công của một hệ thống. Khi nhân tố con người được tính tới thì đối thoại sẽ trôi chảy và nhịp điệu giao tiếp sẽ được thiết lập giữa người dùng và hệ thống. Vì thế,

thiết kế giao diện người sử dụng phải lấy con người là trung tâm: cần tính đến nhu cầu, kinh nghiệm và khả năng của người sử dụng. Những người dùng tiềm năng nên được tham gia như thành viên chính thức của đội thiết kế giao diện. Bảng 3.1 liệt kê một số nguyên tắc chính cho thiết kế giao diện.

Bảng 3.1. Các nguyên tắc thiết kế giao diện người sử dụng

Nguyên tắc	Mô tả
Tính quen thuộc với người sử dụng (user familiarity)	Giao diện nên sử dụng các thuật ngữ và các khái niệm gần gũi được rút ra từ kinh nghiệm của những người sử dụng nhiều các hệ thống.
Tính nhất quán (consistency)	Giao diện nên nhất quán ở chỗ những so sánh nên được kích hoạt theo cùng một cách.
Làm ngạc nhiên nhỏ nhất (minimal surprise)	Các hoạt động của hệ thống không được làm người sử dụng phải ngạc nhiên.
Có khả năng phục hồi (recoverability)	Giao diện nên có các cơ chế cho phép người dùng phục hồi lại trạng thái trước đó mỗi khi gặp lỗi.
Hướng dẫn người sử dụng (user guidance)	Giao diện cần cung cấp các thông tin phản hồi có ý nghĩa và cung cấp các tiện ích trợ giúp người sử dụng theo từng ngữ cảnh nhạy cảm.
Đa dạng người dùng (user diversity)	Giao diện cần cung cấp các tiện ích tương tác thích hợp với nhiều loại người dùng khác nhau.

3.6.3. Tiến trình thiết kế giao diện

Tiến trình thiết kế bao gồm một số hoạt động:

- Bắt đầu với việc tạo ra các mô hình khác nhau về chức năng hệ thống.
- Phác họa ra các nhiệm vụ hướng con người và máy tính để đạt tới chức năng hệ thống (tiến trình tương tác).
- Xem xét các giải pháp thiết kế được áp dụng cho mọi thiết kế giao diện.
- Sử dụng các công cụ làm bản mẫu.
- Cài đặt cho mô hình thiết kế và đánh giá kết quả về chất lượng.

3.6.3.1. Các mô hình thiết kế giao diện

Có bốn mô hình khác nhau [29] khi thiết kế:

1. *Mô hình thiết kế do kỹ sư phần mềm xây dựng.* Nó tổ hợp các biểu diễn dữ liệu, kiến trúc và thủ tục của phần mềm để thực hiện được chức năng.
2. *Mô hình người dùng do kỹ sư phần mềm/kỹ sư con người xây dựng.* Nó mô tả sơ lược hệ thống cho người dùng cuối.
3. *Mô hình người dùng cảm nhận hệ thống do người dùng cuối cùng xây dựng.* Nó là hình ảnh của hệ thống mà người dùng hình dung trong đầu.
4. *Hình ảnh hệ thống do người cài đặt hệ thống xây dựng.* Nó tổ hợp cách biểu lộ bên ngoài của hệ thống dựa trên máy tính với mọi thông tin trợ giúp mô tả cú pháp và ngữ nghĩa của hệ thống. Khi hình ảnh hệ thống và cảm nhận hệ thống trùng nhau thì nói chung người dùng cảm thấy thoải mái với phần mềm và dùng nó một cách hiệu quả.

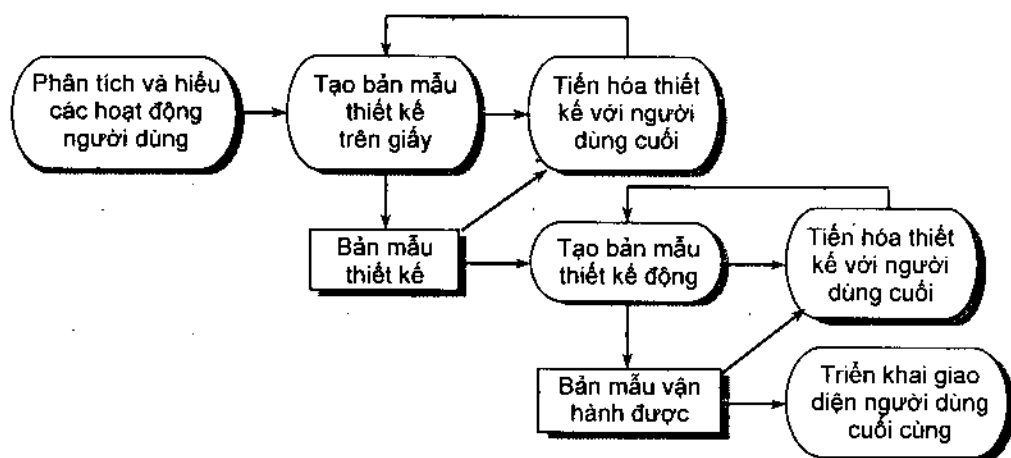
3.6.3.2. Phân tích và mô hình hóa hệ thống

Phân tích nhiệm vụ của giao diện cũng tương tự như phân tích chức năng các đối tượng nội thất, nhưng được áp dụng cho các hoạt động của con người. Các bước của tiến trình thiết kế giao diện có thể được thực hiện theo cách sau:

- Thiết lập các mục tiêu và ý đồ cho nhiệm vụ;
- Ánh xạ thành dãy các hành động xác định;
- Xác định dãy hành động khi nó được thực hiện ở mức giao diện;
- Chỉ ra trạng thái của hệ thống, tức là thể hiện của giao diện vào lúc mà mỗi hành động trong dãy đó được thực hiện;
- Xác định các cơ chế điều khiển, như thiết bị và hành động sẵn có cho người dùng để thay đổi trạng thái hệ thống;
- Chỉ ra cách thức, cơ chế điều khiển này ảnh hưởng tới trạng thái hệ thống;
- Chỉ ra cách thức người dùng diễn giải trạng thái của hệ thống từ thông tin được cung cấp qua giao diện.

3.6.3. Tiến hóa thiết kế và đánh giá

Một khi đã tạo ra bản mẫu giao diện người dùng thì phải đánh giá xem liệu nó có đáp ứng cho nhu cầu của người dùng hay không. Việc đánh giá có thể mở rộng từ chạy "bộ kiểm thử" không hình thức, trong đó người dùng cung cấp phản hồi ứng biến cho tới các nghiên cứu được thiết kế hình thức, dùng các phương pháp thống kê để tính toán, các bảng hỏi cho đa số người dùng điền vào. Tiến trình tiến hóa thiết kế giao diện người dùng theo phương pháp làm mẫu cho ở hình 3.36.



Hình 3.36. Tiến trình thiết kế giao diện bằng cách làm bản mẫu

Khi phát triển giao diện, một số thuộc tính như *khả năng học*, *tốc độ thao tác*, *sự vững chắc*, *khả năng phục hồi*, *khả năng thích nghi* thường được sử dụng để đánh giá. Các tiêu chí này cần được lượng hóa thành các tham số làm cho việc đánh giá trong các cuộc họp xét duyệt thiết kế. Ngoài ra, có thể áp dụng một số phương pháp để thu thập thông tin phục vụ việc phân tích, đánh giá giao diện: *Làm bảng hỏi điều tra*, *quan sát người dùng làm việc với hệ thống*, *khai thác bằng video quay một ca sử dụng hệ thống*, *đưa vào phần mềm đoạn mã để thu thập thông tin về các tiện ích được sử dụng nhiều nhất và các lỗi thường gặp nhất*.

3.6.4. Các hướng dẫn thiết kế giao diện

Các phạm trù của hướng dẫn thiết kế giao diện người máy bao gồm: *Tương tác chung*, *hiển thị thông tin*, *sử dụng màu và kiểm soát vào dữ liệu* ([34] (Chương 15) và [37] (Chương 10)).

3.6.5. Chuẩn giao diện

Việc chuẩn hóa giao diện làm lợi cho cả người phát triển và người dùng giao diện người – máy (*Human Computer Interface – HCI*). Người phát triển có thể dùng lại các mô đun và đối tượng HCI đã có, do đó tạo ra giao diện nhanh chóng hơn và với chất lượng cao hơn đáng kể. Hiện nay chuẩn được dùng phổ biến nhất là chuẩn giao diện đồ họa MS Windows. Có rất nhiều công cụ phát triển phần mềm hỗ trợ chuẩn này, điển hình là bộ công cụ Visual Studio của hãng Microsoft. Một chuẩn khác phổ biến trong hệ điều hành Unix/Linux là chuẩn X – Windows. X – Windows là một chuẩn mở, sử dụng mô hình khách/dịch vụ và có tính hướng mạng cao.

3.7. Tài liệu thiết kế

Dàn bài tài liệu thiết kế cho trong bảng 3.2. Nó có thể được dùng như một hướng dẫn cho đặc tả thiết kế. Mỗi mục đều bao gồm một số đoạn đề cập tới các khía cạnh khác nhau của biểu diễn thiết kế.

Bảng 3.2. Dàn bài hướng dẫn tài liệu đặc tả

I. Phạm vi

- A. Mục tiêu hệ thống
- B. Phần cứng, phần mềm và giao diện con người
- C. Các chức năng phần mềm chính
- D. Cơ sở dữ liệu được xác định bên ngoài
- E. Các ràng buộc, giới hạn thiết kế chính

II. Tài liệu tham khảo

- A. Tài liệu về phần mềm hiện có
- B. Tài liệu hệ thống
- C. Tài liệu người cung cấp (phần cứng, phần mềm)
- D. Tham khảo kỹ thuật

III. Mô tả thiết kế

- A. Mô tả dữ liệu
 - 1. Tổng quan về luồng dữ liệu
 - 2. Tổng quan về cấu trúc dữ liệu
- B. Cấu trúc chương trình suy dẫn
- C. Giao diện bên trong cấu trúc

IV. Môđun: cho mỗi môđun:

- A. Lời thuật xử lý
- B. Mô tả giao diện
- C. Mô tả ngôn ngữ thiết kế (hay những mô tả khác)
- D. Các môđun đã dùng
- E. Tổ chức dữ liệu
- F. Bình luận

V. Cấu trúc tệp và dữ liệu toàn cục

- A. Cấu trúc tệp dữ liệu
 - 1. Cấu trúc logic
 - 2. Mô tả bản ghi logic
 - 3. Phương pháp truy nhập
- B. Dữ liệu toàn cục
- C. Tham khảo chéo tệp và dữ liệu

VI. Tham khảo chéo yếu cầu

VII. Điều khoản kiểm thử

- A. Hướng dẫn kiểm thử
- B. Chiến lược kiểm thử
- C. Xem xét đặc biệt

VIII. Bao gói

- A. Các điều khoản đặc biệt
- B. Xem xét chuyển giao

IX. Lưu ý đặc biệt

X. Phụ lục

Tổng kết

- ◆ Thiết kế phần mềm là quá trình chuyển hóa các đặc tả yêu cầu thành một biểu diễn thiết kế của hệ thống phần mềm cần xây dựng. Nó là cốt lõi của kỹ nghệ phần mềm. Thiết kế là một phần của một chiến lược phát triển: *định hướng cấu trúc* hay *định hướng đối tượng*, hay *các hệ thời gian thực*. Thiết kế là tạo mô hình một hệ thống của thế giới thực. Vì vậy cần mô hình hóa hệ thống với các khái niệm trừu tượng. Nhiều khái niệm được dùng cho thiết kế: *môđun*, *làm mịn*, *thủ tục*, *kết dính*, *ghép nối*, *kế thừa*, *kiến trúc*, *che dấu thông tin*, *tính bảo trì được*, *tính hiểu được*, *tính thích nghi được*.

- ◆ Tiến trình thiết kế phần mềm có thể xem xét theo nhiều cách khác nhau. Theo quan điểm kỹ thuật, thiết kế bao gồm bốn hoạt động chính: *thiết kế kiến trúc*, *thiết kế dữ liệu*, *thiết kế thủ tục* và *thiết kế giao diện*. Theo quan điểm quản lý, thiết kế tiến hóa từ *thiết kế sơ bộ* sang *thiết kế chi tiết*.
- ◆ 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*. Kiến trúc của hệ thống có ảnh hưởng sâu rộng lên quá trình phát triển và đặc trưng chung của hệ thống, như độ tin cậy, an toàn... Thiết kế kiến trúc gồm: *cấu trúc hệ thống*, *mô hình hóa điều khiển* và *phân rã mô đun các hệ con* dựa trên các mô hình kiến trúc tổng quát hay mô hình tham khảo đã có.
- ◆ *Thiết kế hướng cấu trúc* có hai phần tương đối độc lập với nhau: Thiết kế dữ liệu và thiết kế xử lý. Thiết kế dữ liệu đi trước một bước. Hai mô hình được sử dụng là ERM và mô hình dữ liệu quan hệ. Thiết kế xử lý sử dụng mô hình luồng dữ liệu để cấu trúc hệ thống và phương pháp ánh xạ mô đun để thiết kế kiến trúc thành phần.
- ◆ *Thiết kế hướng đối tượng* nhằm xác định các lớp, thuộc tính và mối quan hệ của chúng. Có nhiều phương pháp, nhưng tiến trình RUP là tiên tiến. Khi cấu trúc thành phần với các lớp cần dựa trên các mẫu thiết kế. UML là ngôn ngữ chuẩn cho phát triển hệ thống theo hướng đối tượng.
- ◆ Hệ thống thời gian thực là hệ thống sinh ra đáp ứng cho các sự kiện của môi trường với giới hạn về thời gian chặt chẽ. Hệ thống phần mềm thời gian thực phân thành hệ "mềm" và hệ "cứng". Do đặc trưng thay đổi trạng thái hệ thống để đáp ứng sự kiện thời gian thực nên phương pháp máy trạng thái hữu hạn trở thành một phần trong phương pháp thiết kế thời gian thực.

Câu hỏi ôn tập

1. Tầm quan trọng của thiết kế phần mềm. Nêu các giai đoạn thiết kế cần trải qua.

2. Vẽ sơ đồ các hoạt động thiết kế và sản phẩm thiết kế của nó. Mô tả các hoạt động cốt yếu trong đó.
3. Thiết kế có thể nhìn nhận theo những quan điểm nào? Và nội dung theo mỗi quan điểm đó. Trình bày các khái niệm sử dụng trong thiết kế.
4. Trình bày ý tưởng của các chiến lược phát triển hệ thống hướng cấu trúc và hướng đối tượng.
5. Khái niệm về thiết kế tốt và các hướng dẫn đảm bảo chất lượng thiết kế.
6. Các bước thiết kế kiến trúc một hệ thống. Các loại kiến trúc thường gặp.
7. Các bước thiết kế hướng cấu trúc. Nêu ưu và nhược điểm của nó.
8. Các bước thiết kế hướng đối tượng. Nêu ưu và nhược điểm của nó.
9. Nêu các thể hệ giao diện người – máy đã trải qua. Nêu tiến trình thiết kế giao diện.
10. Trình bày các mô hình thiết kế giao diện. Các nguyên tắc thiết kế giao diện.

Chương 4

LẬP TRÌNH

4.1. Ngôn ngữ lập trình

Ngôn ngữ lập trình là phương tiện để liên lạc giữa con người và máy tính. Tiến trình lập trình – sự chuyển các yêu cầu của con người cho máy tính thực hiện thông qua ngôn ngữ lập trình – là một hoạt động trí tuệ rất cao của con người (lập trình viên) và là một bước cốt lõi trong tiến trình kỹ nghệ phần mềm.

4.1.1. Đặc trưng của ngôn ngữ lập trình

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

- Dễ dịch thiết kế sang chương trình;
- Từ khóa gần với ngôn ngữ tự nhiên;
- Có trình biên dịch hiệu quả;
- Khả chuyển chương trình gốc;
- Có sẵn công cụ phát triển;
- Dễ bảo trì.

Bước lập trình bắt đầu sau khi thiết kế chi tiết đã được xác định, xét duyệt và sửa đổi nếu cần. Về lý thuyết, việc sinh chương trình gốc từ một đặc tả chi tiết nên là một ánh xạ trực tiếp. *Dễ dịch thiết kế sang chương trình* là đặc trưng về việc một ngôn ngữ lập trình có các cấu trúc ánh xạ gần

gửi với các biểu diễn thiết kế. Khi đó, một ngôn ngữ có thể cài đặt trực tiếp các kết cấu có cấu trúc, dữ liệu phức tạp, vào/ra đặc biệt, khả năng thao tác với các bit, và các kết cấu hướng đối tượng sẽ làm cho việc dịch từ thiết kế sang chương trình gốc dễ dàng (nếu các thuộc tính này được xác định trong thiết kế).

Những tiến bộ nhanh chóng về tốc độ xử lý và dung lượng bộ nhớ của máy tính đã làm giảm nhẹ nhu cầu chương trình siêu hiệu quả, nhưng nhiều ứng dụng vẫn còn đòi hỏi các chương trình phải gọn gàng (yêu cầu bộ nhớ thấp) và chạy nhanh. Các ngôn ngữ với *trình biên dịch tối ưu* có thể là rất thích hợp với các phần mềm mà hiệu năng là yêu cầu chủ chốt của nó.

Tính khả chuyển chương trình gốc là một đặc trưng quan trọng khác của ngôn ngữ lập trình. Có thể hiểu tính khả chuyển theo ba cách khác nhau:

- Chương trình gốc có thể được chuyển từ bộ xử lý này sang bộ xử lý khác và từ trình biên dịch nọ sang trình biên dịch kia với rất ít hoặc không phải sửa đổi gì.
- Chương trình gốc vẫn không cần thay đổi ngay cả khi môi trường của nó thay đổi (như việc cài đặt bản mới của hệ điều hành).
- Chương trình gốc có thể được tích hợp vào trong các phần mềm khác nhau với ít hay không cần thay đổi gì nhờ các đặc trưng của ngôn ngữ lập trình.

Trong ba cách hiểu trên thì cách thứ nhất là thông dụng và được quan tâm nhiều nhất. Việc đưa ra các chuẩn (do Tổ chức Tiêu chuẩn quốc tế ISO hay Viện Tiêu chuẩn Quốc gia Mỹ – ANSI) góp phần làm nâng cao tính khả chuyển của ngôn ngữ.

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

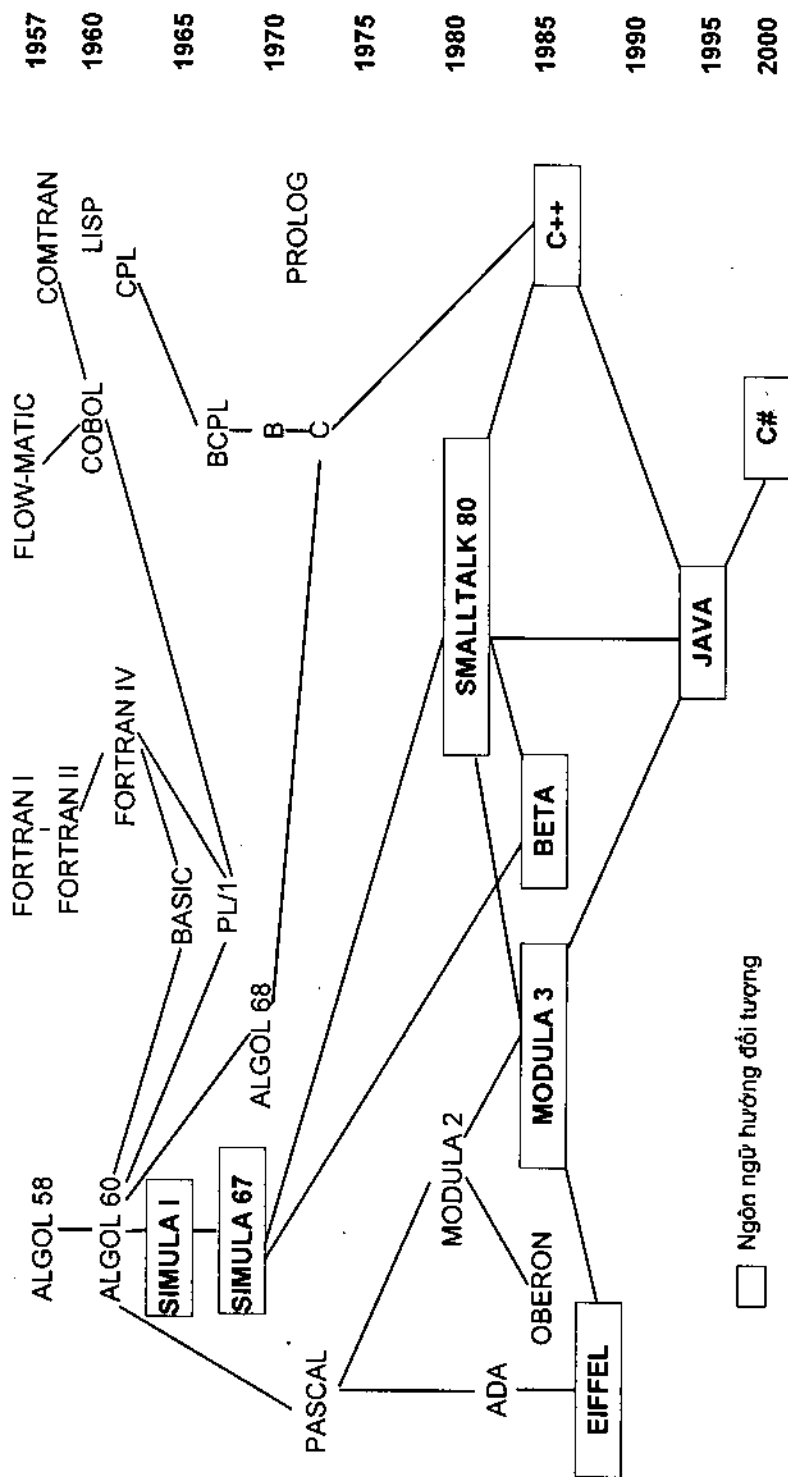
Tính dễ bảo trì của chương trình gốc có tầm quan trọng cốt yếu cho tất cả các nỗ lực phát triển phần mềm. Việc bảo trì không thể được tiến hành chừng nào người ta còn chưa hiểu được phần mềm. Các yếu tố của cấu hình phần mềm (như tài liệu thiết kế) tạo ra một cơ sở tốt cho việc nắm hiểu nhanh chóng phần mềm, nhưng cuối cùng thì vẫn phải đọc và sửa đổi chương trình gốc theo những thay đổi của thiết kế. Vì thế, tính dễ bảo trì trước hết thể hiện ở tính dễ đọc, dễ hiểu của chương trình gốc.

Tính dễ dịch thiết kế sang chương trình là một yếu tố quan trọng để phát triển nhanh và dễ bảo trì chương trình gốc. Bên cạnh đó, các đặc trưng tự làm tài liệu của ngôn ngữ (như chiều dài được phép của tên gọi, định dạng nhãn, định nghĩa kiểu, cấu trúc dữ liệu) có ảnh hưởng mạnh đến tính dễ bảo trì. Một đặc trưng khác liên quan là có các từ khóa gần với ngôn ngữ tự nhiên. Điều này cũng làm cho chương trình dễ hiểu, dễ bảo trì, cũng như dễ dịch thiết kế sang chương trình. Tuy nhiên, mặt trái của đặc trưng này là làm cho chương trình nguồn trở nên rườm rà, đôi khi không thích hợp với môi trường lập trình chuyên nghiệp.

4.1.2. Sự tiến hóa và đa dạng của các ngôn ngữ lập trình

Cùng với sự phát triển của công nghệ chế tạo máy tính, các phần mềm ngày càng trở nên phức tạp và các ngôn ngữ lập trình – công cụ để mô tả phần mềm cũng phát triển không ngừng. Mục tiêu của người thiết kế ngôn ngữ lập trình là tạo ra các ngôn ngữ hiệu quả hơn cho phát triển phần mềm. Tuy nhiên, tùy vào từng giai đoạn và từng kỹ thuật cụ thể, chúng ta có thể nhìn nhận sự tiến hóa này theo một số tiêu chí như sau:

- *Xuất hiện các ngôn ngữ bậc cao hơn*: Các ngôn ngữ bậc cao gắn với tư duy lập trình có cấu trúc (ALGOL, FORTRAN, BASIC, PASCAL, C,...) xuất hiện giúp cho việc lập trình trở nên hiệu quả và khoa học hơn.
- *Các ngôn ngữ lập trình hướng đối tượng*: Thập kỷ 70 của thế kỷ XX đánh dấu sự xuất hiện của tư duy lập trình hướng đối tượng và kéo theo sự xuất hiện của một loạt các ngôn ngữ lập trình hướng đối tượng như MODULA, C ++, Java. Lập trình hướng đối tượng mở ra khả năng sử dụng lại cao hơn, đồng thời cũng nâng cao một bước về chất lượng phần mềm nhờ sự bao gói và che dấu thông tin của nó.



Hình 4.1. Sự phát triển của một số ngôn ngữ lập trình phổ biến

- *Các ngôn ngữ thông dịch*: Tốc độ xử lý ngày càng cao và bộ nhớ ngày càng lớn khiến cho việc ứng dụng các ngôn ngữ thông dịch ngày càng mạnh mẽ. Thập niên vừa qua chứng kiến sự xuất hiện của một loạt các ngôn ngữ thông dịch như ASP, PHP, JavaScript, Python,... Các ngôn ngữ này với độ trừu tượng cao và đặc thù với các miền ứng dụng, giúp cho việc xây dựng ứng dụng vừa và nhỏ (đặc biệt là các ứng dụng cơ sở dữ liệu loại nhỏ, sinh báo cáo, quản lý nghiệp vụ,...) trở nên ngày càng đơn giản và việc kiểm tra trở nên nhanh chóng.
- *Đa dạng hóa và đặc thù với miền ứng dụng*: Ngược với xu hướng thiết kế các ngôn ngữ vạn năng là xu hướng thiết kế các ngôn ngữ đặc thù với các miền ứng dụng. Hầu hết các ngôn ngữ này thuộc lớp các ngôn ngữ thông dịch. Ngoài một số ngôn ngữ kể trên còn phải kể đến LISP được thiết kế chuyên dụng để xử lý ngôn ngữ tự nhiên và thao tác chuỗi, PROLOG chuyên dụng cho lập trình logic, SQL để truy vấn với cơ sở dữ liệu.

Hình 4.1 mô tả lịch sử phát triển của một số ngôn ngữ lập trình chính yếu.

4.1.3. Các loại ngôn ngữ và lựa chọn ngôn ngữ lập trình

4.1.3.1. Phân loại ngôn ngữ

Có nhiều cách để phân loại các ngôn ngữ lập trình. Sau đây là một vài cách phân loại cần biết để giúp lựa chọn cho mỗi ứng dụng cụ thể.

a) Phân loại theo sự tiến hóa

Theo tiến trình phát triển, có các ngôn ngữ với các thế hệ khác nhau:

- *Ngôn ngữ thế hệ đầu tiên*: Bao gồm ngôn ngữ mức mã máy và hợp ngữ. Đây là những lớp ngôn ngữ đầu tiên mà máy có thể hiểu được và vận hành trực tiếp.
- *Ngôn ngữ thế hệ thứ hai*: Các ngôn ngữ bậc cao phát triển vào cuối những năm 50 và đầu những năm 60 của thế kỷ XX. Đặc trưng của chúng là việc sử dụng một số từ vựng và cú pháp của tiếng Anh để cấu trúc câu lệnh có ngữ nghĩa. Các ngôn ngữ này cũng sử dụng rộng rãi các thư viện phần mềm quen thuộc (đặc biệt các hàm tính toán

toán học) đã được chấp nhận (tiêu biểu như FORTRAN, ALGOL, COBOL, BASIC,...). Các ngôn ngữ này cần các bộ dịch để chuyển nó về ngôn ngữ mà máy hiểu được.

- *Ngôn ngữ thế hệ thứ ba*: Chúng còn được gọi là *ngôn ngữ lập trình có cấu trúc*. Chúng là sự hoàn thiện, nâng cấp và mở rộng các ngôn ngữ thế hệ thứ hai. Các ngôn ngữ lớp này được chia thành ba phạm trù lớn theo đặc điểm của chúng: *ngôn ngữ bậc cao vạn năng (ALGOL, PL/I, PASCAL, Modula – 2, C và Ada)*, *ngôn ngữ bậc cao hướng đối tượng (C++, Smalltalk, Eiffel)* và *ngôn ngữ bậc cao chuyên dụng (LISP, PROLOG, APL)*.
- *Ngôn ngữ thế hệ thứ tư (4GL – forth generation language)*: Các ngôn ngữ lớp này nâng mức trừu tượng hóa cao hơn so với thế hệ thứ ba bằng cách giảm thiểu các yêu cầu xác định chi tiết về thủ tục. Ngôn ngữ cho phép người dùng xác định các điều kiện, các ràng buộc của vấn đề (*mang tính thủ tục*) của môi trường và chỉ ra các kết quả mong muốn (*phần phi thủ tục*). Từ đó nó áp dụng các tri thức và các luật của miền lĩnh vực để sinh ra các chi tiết thủ tục của chương trình đầy đủ.
- *Ngôn ngữ truy vấn* thuộc vào 4GL đã được phát triển thường đi với các ứng dụng về cơ sở dữ liệu. Nó đưa ra giao diện ngôn ngữ cho phép người dùng tạo các yêu cầu kết quả có ngữ nghĩa cụ thể, ví dụ: *For the eastern and western region, how did actual sales for last month compare with forecast?* (Với các vùng đông và tây, doanh số bán hàng của tháng trước so với dự kiến như thế nào?)
- *Bộ sinh chương trình* có thể xem là thuộc vào 4GL. Nó cho phép người dùng tạo các chương trình thế hệ thứ ba một cách hoàn chỉnh bằng cách sử dụng một số câu lệnh cỡ lớn để mô tả yêu cầu. Các ngôn ngữ bậc rất cao này dùng nhiều trừu tượng về dữ liệu và thủ tục trợ giúp việc mô tả các ràng buộc và yêu cầu của người dùng một cách dễ dàng. Từ đó nó sinh ra các chương trình đầy đủ. Với một số công cụ CASE hiện nay, người ta có thể mô hình hóa một ứng dụng bằng các ký pháp đồ họa nhờ các công cụ đó, và có những bộ dịch để dịch nó thành chương trình gốc theo một ngôn ngữ lập trình được chọn [32].

Ngoài ra có những ngôn ngữ khác cũng thuộc 4GL như các ngôn ngữ của các hệ trợ giúp quyết định, ngôn ngữ làm bản mẫu, các ngôn ngữ đặc tả hình thức... cũng trợ giúp việc tạo ra chương trình mà không cần phải đặc tả chi tiết các thủ tục.

b) Phân loại theo miền ứng dụng

Mỗi lớp ngôn ngữ lập trình thường có những đặc trưng riêng của nó. Nhờ có các đặc trưng này mà nó sẽ thích hợp với những lớp bài toán trong những lĩnh vực ứng dụng nhất định (sẽ trình bày kỹ ở phần sau). Đây là một cách phân loại quan trọng, giúp việc lựa chọn ngôn ngữ lập trình hiệu quả cho một ứng dụng cụ thể.

c) Phân loại theo môi trường: mạng (network) – không mạng

Các ngôn ngữ có thể chỉ dùng cho các máy trạm hay dùng trong môi trường mạng, đặc biệt là trên mạng internet. Các lớp ngôn ngữ này (ví dụ như Java, Perl, ASP, PHP,...) có các cấu trúc trợ giúp tổ chức việc truyền thông trên mạng một cách đơn giản, dễ dàng.

d) Phân loại theo sự tiện ích: trực quan (visual) – không trực quan

Các ngôn ngữ trực quan như: Visual Basic, Visual Foxpro, Visual C,... Xu hướng chung là các ngôn ngữ được trang bị thêm các công cụ trợ giúp làm trực quan hóa các quá trình lập trình và kiểm thử. Nhờ vậy làm đơn giản các thao tác và yêu cầu ghi nhớ của người lập trình, làm giảm công sức của các lập trình viên.

4.1.3.2. Lựa chọn ngôn ngữ

Các đặc trưng của ngôn ngữ lập trình sẽ quyết định miền ứng dụng của ngôn ngữ. Miền ứng dụng là yếu tố chính để chúng ta lựa chọn ngôn ngữ cho một dự án phần mềm. Dưới đây sẽ phân tích những đặc điểm của ngôn ngữ lập trình thông dụng và miền ứng dụng của chúng.

Ngôn ngữ lập trình C xuất hiện trong quá trình phát triển UNIX, là một ngôn ngữ thường được chọn cho việc phát triển phần mềm hệ thống. Nhân và phần lớn các thành phần của hệ điều hành UNIX và Linux được viết bằng ngôn ngữ này. C là ngôn ngữ súc tích, hiệu quả và có khả năng can thiệp vào mức thấp của hệ thống, vì thế nó thích hợp với những lập trình viên giàu kinh nghiệm.

Pascal là một ngôn ngữ thủ tục, được thiết kế rõ ràng, rất thích hợp cho việc giảng dạy về phương pháp lập trình và mô tả thuật toán. Pascal cùng với chương trình dịch Turbo Pascal đã từng rất phổ biến ở Việt Nam. Tuy nhiên, Pascal chuẩn không thuận tiện cho việc phát triển các ứng dụng lớn, và các ứng dụng thương mại. Hiện nay, một dị bản của Pascal là Delphi với thư viện đồ họa mạnh vẫn được sử dụng rộng rãi trong phát triển các ứng dụng GIS.

FORTRAN (FORMula TRANslation) là một ngôn ngữ thủ tục xuất hiện từ những năm 1960. Tuy cổ điển và có rất nhiều hạn chế về xử lý tệp và cơ sở dữ liệu, nhưng với khả năng tính toán có độ chính xác cao và thư viện toán học phong phú, FORTRAN vẫn còn là ngôn ngữ thống trị trong lĩnh vực tính toán khoa học kỹ thuật. Nhiều phiên bản FORTRAN hỗ trợ tốt cho lập trình song song và được sử dụng trên các siêu máy tính. Tuy vậy, Pascal và C cũng được dùng rộng rãi cho các mục đích tương tự. Môi trường tính toán song song PVM và MPI hỗ trợ tốt cho cả 3 ngôn ngữ này.

COBOL là ngôn ngữ cho ứng dụng kinh doanh và khai thác cơ sở dữ liệu lớn, tuy nhiên các ngôn ngữ thế hệ thứ tư đã dần dần chiếm ưu thế và thay thế ngôn ngữ này. COBOL hầu như không được sử dụng ở Việt Nam. Cho đến trước năm 2000, có thể nói Foxpro là ngôn ngữ phát triển các ứng dụng cơ sở dữ liệu cỡ nhỏ phổ biến nhất ở Việt Nam. Hiện nay, hậu duệ của Foxpro là Foxweb vẫn được nhiều nhóm phát triển sử dụng. Tuy nhiên, xu hướng chủ đạo hiện nay là sử dụng hệ quản trị cơ sở dữ liệu độc lập với ngôn ngữ phát triển ứng dụng.

Do sự đơn giản của cú pháp và tính sẵn có của trình thông dịch, BASIC (Beginner's All – purpose Symbolic Instruction Code) đã có thời là một ngôn ngữ lập trình thống trị. Visual Basic với cú pháp xuất xứ từ BASIC, khả năng xây dựng giao diện đồ họa và truy cập cơ sở dữ liệu dễ dàng đang được đông đảo người phát triển ứng dụng cho máy tính cá nhân sử dụng. Tuy nhiên, ngôn ngữ này rất hiếm khi được những người phát triển các hệ thống lớn sử dụng.

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

Với đặc trưng hướng đối tượng, tính hiệu quả thực hiện, tương thích ngược với C cũng như có nhiều công cụ và thư viện, C++ hiện đang được sử dụng rộng rãi trong phát triển phần mềm. C++ đặc biệt thích hợp với các phần mềm đòi hỏi hiệu năng cao, phần mềm hệ thống, mô phỏng,... Ngôn ngữ Java cũng là một ngôn ngữ hướng đối tượng đang được sử dụng rộng rãi cho phát triển các ứng dụng trên nền Web và phần mềm nhúng vì các lý do độ an toàn cao, tính trong sáng, tính khả chuyển và hướng thành phần. Theo một số thống kê thì tốc độ phát triển một ứng dụng mới bằng Java cao hơn đến 2 lần so với các ngôn ngữ truyền thống như C hay thậm chí C++. Một trong những ưu điểm nổi bật của Java so với các ngôn ngữ vạn năng khác (C, C++,...) là nó giải phóng lập trình viên khỏi việc quản lý bộ nhớ, một công việc phức tạp và là nguyên nhân của rất nhiều lỗi lập trình.

Trong các ứng dụng thời gian thực chúng ta hay gặp các ngôn ngữ như Ada, C, C++ và cả hợp ngữ do tính hiệu quả của chúng. Ada là một ngôn ngữ ban đầu được phát triển để xây dựng các ứng dụng quân sự của Mỹ và được đòi hỏi khắt khe về độ an toàn. Các ngôn ngữ này cùng với Java cũng được dùng cho phát triển phần mềm nhúng.

Các ứng dụng trí tuệ nhân tạo thường dùng các ngôn ngữ như LISP, PROLOG, OPS5 hay các hậu duệ của chúng. Tuy vậy, nhiều ngôn ngữ lập trình (vạn năng) khác như C++ và Java cũng được dùng. LISP (LISt Processing) là ngôn ngữ lập trình hàm được phát triển tại MIT và rất thuận tiện cho xử lý các danh sách. PROLOG (PROgramming in Logic) là ngôn ngữ lập trình logic chuyên dụng, được phát triển ban đầu tại Đại học Marseiller, Pháp. PROLOG chứng minh tự động một kết luận (là đúng hay sai) từ các luật và các mệnh đề (sự thực) đã cho.

Các ngôn ngữ thông dịch (script) với những câu lệnh và thư viện mạnh hiện đang rất được chú ý. Các ngôn ngữ thông dịch như ASP, PHP, JavaScript, PERL với khả năng dễ dàng kết nối với cơ sở dữ liệu đang được sử dụng rộng rãi trong phát triển các ứng dụng trên nền Web. ASP, PHP và PERL được dùng để phát triển các ứng dụng chạy trên máy chủ (server) còn JavaScript được dùng để phát triển các xử lý chạy trên máy khách (client).

Như vậy, khi lựa chọn ngôn ngữ lập trình cho một ứng dụng cụ thể, ta cần xét đến các đặc trưng của nó và sử dụng những kinh nghiệm trên đây để

chọn ngôn ngữ lập trình thích hợp. Trong thực tế, một lập trình viên thường thông thạo từ một đến hai ngôn ngữ lập trình. Xu hướng lựa chọn của họ thường là ngôn ngữ mình thành thạo nhất, bởi vì nó cho phép lập trình viên tốn ít công sức hơn để tạo ra chương trình cho ứng dụng; và đặc biệt là với ngôn ngữ này, họ có khả năng xử lý cả những vấn đề tinh tế nhất trong bài toán mà họ có thể gặp.

4.1.4. Sự ảnh hưởng của ngôn ngữ tới kỹ nghệ phần mềm

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

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

4.2. Phương pháp lập trình

4.2.1. Trừu tượng hóa trong lập trình

Trừu tượng hóa là một quá trình tư duy tự nhiên của con người khi tìm cách nhận thức một sự vật, một vấn đề. Có thể diễn giải một cách đơn giản "trừu tượng hóa" là sự hình thành khái niệm về sự vật hay hiện tượng bằng

cách tập trung vào các đặc trưng cần thiết và loại bỏ các chi tiết cụ thể không cần thiết.

Trong việc thiết kế chương trình, chúng ta phải ánh xạ một bài toán thực thành các thành phần của chương trình như các cấu trúc dữ liệu, cấu trúc điều khiển,... Các đối tượng trong bài toán thực tế có thể chứa rất nhiều các thuộc tính khác nhau, nhưng khi ánh xạ vào thành phần trong chương trình chúng ta chỉ mô tả các thuộc tính cần thiết mà chúng ta quan tâm. Không những thế, các thuộc tính này còn bị rời rạc hóa và được mô tả trong một không gian hữu hạn. Đó là quá trình trừu tượng hóa trong tiến trình phát triển phần mềm. Các kết quả của quá trình trừu tượng hóa này gọi là các mô hình. Với phần mềm, có thể có các mô hình như mô hình dữ liệu, mô hình luồng điều khiển, mô hình giao diện,...

Ví dụ, để giải quyết một bài toán quản lý sinh viên của trường đại học, chúng ta có thể trừu tượng hóa khái niệm sinh viên thực thành mô hình sinh viên, bao gồm các thuộc tính như *mã sinh viên*, *họ tên*, *điểm trung bình*... Với bài toán này chúng ta có thể bỏ qua các thuộc tính như chiều cao, cân nặng,... Tuy nhiên, với một bài toán quản lý sức khỏe cộng đồng thì các thuộc tính này lại có thể là các thuộc tính cơ bản.

Trừu tượng hóa là chìa khóa then chốt trong phát triển phần mềm. Nếu tiến hành trừu tượng hóa tốt sẽ đảm bảo phát triển được phần mềm có chất lượng và dễ bảo trì.

4.2.2. Lập trình thủ tục

Lập trình thủ tục (*procedural programming*) là phương pháp lập trình tiên tiến, bao gồm hai khái niệm chính:

- *Sử dụng các thủ tục*: Chia chương trình lớn thành các đoạn chương trình con (thủ tục) thực hiện các công việc xác định. Các thủ tục có thể được gọi nhiều lần (*sử dụng lại*), gọi lẫn nhau và gọi chính nó (*đệ quy*). Đây chính là ý tưởng chia để trị (*divide and conquer*). Nó góp phần giảm độ phức tạp và tăng tính hiệu được của chương trình.
- *Lập trình có cấu trúc* (*structured programming*): Sử dụng các câu lệnh có cấu trúc hoặc tạo ra các cấu trúc phức tạp từ các cấu trúc đơn giản hơn. Câu lệnh có cấu trúc (ví dụ như while – do, for,...) là các

câu lệnh có mục đích sử dụng rõ ràng, điểm bắt đầu và điểm kết thúc câu lệnh tường minh. Điều đó làm cho chương trình trở nên dễ hiểu và sẽ khiến cho việc lập trình dễ dàng hơn (ít sinh lỗi hơn).

Việc chia đề tài cũng như việc sử dụng các câu lệnh có cấu trúc khiến cho chương trình dễ hiểu, dễ tái sử dụng lại. Đồng thời, nó cũng góp phần loại bỏ khái niệm lập trình cổ điển lạm dụng câu lệnh GOTO, nguyên nhân của một số lượng lớn lỗi lập trình.

4.2.3. Lập trình hướng đối tượng

Phương pháp lập trình hướng đối tượng (*object - oriented programming*) xuất hiện vào thập kỷ 70 của thế kỷ XX và nhanh chóng trở thành phương pháp chủ đạo trong phát triển các phần mềm lớn và chuyên nghiệp. Lập trình hướng đối tượng giúp phát triển các phần mềm có chức năng phức tạp hơn, có hiệu quả hơn và dễ bảo trì hơn.

Lập trình hướng đối tượng dựa trên ba đặc trưng cơ bản là bao gói/che dấu thông tin, kế thừa và đa hình.

Bao gói/che dấu thông tin (*encapsulation/information hiding*) là đặc trưng cơ bản nhất của lập trình hướng đối tượng. Bao gói/che dấu thông tin bao gồm hai công việc liên quan mật thiết với nhau: (i) việc gắn kết dữ liệu và các phương thức xử lý trên các dữ liệu này thành một thể thống nhất gọi là (lớp) đối tượng, và (ii) việc che dấu các cài đặt và cấu trúc dữ liệu cụ thể của đối tượng, và đối tượng chỉ đưa ra kết quả thông qua việc cấp các dịch vụ trên giao diện xác định. Thực hiện tốt việc bao gói/che dấu thông tin sẽ cho phép chúng ta thao tác với đối tượng mà không cần biết cách thức cài đặt cụ thể của nó, tức là chúng ta tách được phần bất biến (giao diện) khỏi phần khả biến (một cách cài đặt cụ thể) của (lớp) đối tượng. Do việc sử dụng đối tượng chỉ cần biết thông tin tối thiểu về giao diện nên chúng ta dễ dàng sử dụng lại, đồng thời việc bảo trì (nâng cấp cài đặt của đối tượng) là dễ dàng. Thêm nữa, do chỉ thay đổi được trạng thái của đối tượng thông qua các giao diện xác định nên có thể đảm bảo được các trạng thái của đối tượng là luôn đúng đắn.

Kế thừa (*inheritance*) là cơ chế đặc biệt của lập trình hướng đối tượng, cho phép xây dựng một lớp đối tượng mới kế thừa (sử dụng lại) lớp đối tượng đã có. Kế thừa thúc đẩy việc sử dụng lại trong bản thân một phần mềm cũng như giữa các phần mềm với nhau.

Đa hình (*polymorphism*) là cơ chế cho phép nhìn nhận một đối tượng theo các góc nhìn khác nhau, đồng thời cho phép các đối tượng khác nhau giải nghĩa cùng một thông điệp (lời gọi phương thức) theo các cách thức khác nhau. Cùng với kế thừa, đa hình tạo nên sự mềm dẻo đặc biệt trong lập trình hướng đối tượng. Đa hình đồng thời là cơ chế nền tảng cho việc thực hiện các thiết kế mở và xây dựng các mẫu thiết kế.

4.2.4. Lập trình hàm

Lập trình hàm (*functional programming*) là phương pháp lập trình dựa trên việc đánh giá các biểu thức toán học và tạo ra các giá trị mới từ các giá trị đã có. Ngược lại, với các phương pháp lập trình thông thường dựa trên các lệnh tuần tự và biến đổi trạng thái, lập trình hàm đánh giá một biểu thức bằng cách thực hiện việc đánh giá các bộ phận (có thể coi là một cách song song) thông qua việc sử dụng các hàm khác hoặc chính nó và tạo ra kết quả mà không thay đổi các trạng thái đã có. Lập trình hàm có một số ưu điểm như dễ dàng song song hóa, không tạo ra các hiệu ứng phụ do các hàm không sử dụng chung dữ liệu. Ngôn ngữ lập trình hàm tiêu biểu là LISP và hậu duệ của nó là Scheme.

4.2.5. Lập trình lôgic

Lập trình lôgic (*logic programming*) là phương pháp lập trình dựa trên việc mô tả tri thức mang tính lôgic về bài toán và mục tiêu cần giải chứ không phải là các bước để giải bài toán. Tri thức ở đây bao gồm các quy tắc về miền bài toán và các dữ kiện đã có. Hệ thống sẽ tìm cách kiểm tra xem từ việc các dữ kiện và các quy tắc đó có cách nào để đạt tới mục tiêu (chứng minh) hay không. Ngôn ngữ lập trình lôgic điển hình là Prolog và miền ứng dụng chủ yếu của phương pháp lập trình này là các hệ chuyên gia và các bài toán chứng minh định lý tự động.

4.3. Phong cách lập trình

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

4.3.1. Tài liệu chương trình

Tài liệu bên trong của chương trình gốc bắt đầu với việc chọn lựa các tên gọi định danh (biểu và nhãn), tiếp tục với vị trí và thành phần của việc chú thích, và kết luận với cách tổ chức cấu trúc một cách trực quan của chương trình.

Việc lựa chọn các tên gọi định danh có nghĩa là điều chủ chốt cho việc hiểu chương trình. Những ngôn ngữ giới hạn độ dài tên biến hay nhãn làm cho các tên không đầy đủ và mang nghĩa mơ hồ. Cho dù một chương trình nhỏ thì một tên gọi có nghĩa cũng làm tăng tính dễ hiểu. Theo ngôn từ của mô hình cú pháp/ngữ nghĩa, tên có ý nghĩa làm "đơn giản hóa việc chuyển đổi từ cú pháp chương trình sang cấu trúc ngữ nghĩa bên trong".

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

Có nhiều hướng dẫn đã được đề nghị cho việc viết lời chú thích. Các chú thích mở đầu và chú thích chức năng là hai phạm trù đòi hỏi cách tiếp cận có hơi khác.

Lời chú thích mở đầu nên xuất hiện ở ngay đầu của mọi môđun. Định dạng cho lời chú thích như thế là:

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

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

Việc viết tài liệu bên trong chương trình ngày càng được chú trọng với sự xuất hiện các công cụ soạn thảo mã nguồn mạnh, hỗ trợ cho việc hiển thị chú thích theo các khuôn dạng thích hợp và hỗ trợ sinh tự động các tài liệu ngoài chú thích. Các tài liệu mô tả ngoài sinh tự động theo kiểu này thường được tạo ra theo định dạng HTML và có các liên kết trỏ thẳng đến các thành phần tương ứng của chương trình.

4.3.2. Khai báo dữ liệu

Độ phức tạp của cấu trúc dữ liệu phụ thuộc vào thiết kế dữ liệu. Vấn đề này được xác định trong pha thiết kế. Tuy nhiên, người lập trình với việc tuân theo những chỉ dẫn đơn giản vẫn có thể cài đặt các cấu trúc dữ liệu dễ hiểu hơn và thuận tiện hơn trong bảo trì. Dưới đây là một số hướng dẫn trong việc khai báo dữ liệu:

- *Các cấu trúc dữ liệu nên được chú giải đầy đủ về cấu trúc và chức năng, và các đặc thù về sử dụng. Đặc biệt là đối với các cấu trúc phức tạp như danh sách móc nối trong các ngôn ngữ như C hay Pascal. Tương tự như vậy, cần phải chú giải về mục đích sử dụng đối với các biến quan trọng, đặc biệt là các biến tổng thể.*
- *Thứ tự khai báo dữ liệu nên được chuẩn hóa cho dù ngôn ngữ lập trình không có yêu cầu bắt buộc nào về điều đó. Điều đó có nghĩa là các biến thuộc cùng kiểu hoặc cùng mục đích sử dụng nên được xếp cạnh nhau.*
- *Quy cách đặt tên cho các biến cần nhất quán. Các tên biến ngoài việc có nghĩa còn cần mang thông tin về kiểu của chúng. Ví dụ, cần thống nhất cách đặt tên biến cho kiểu số nguyên, kiểu số thực, kiểu con trỏ... Nên phân biệt giữa các tên cho các biến cục bộ và các biến sử dụng vào các mục đích đặc biệt như chỉ mục trong vòng lặp.*

4.3.3. Xây dựng câu lệnh

Thiết kế chi tiết các luồng logic của một chức năng phần mềm là công đoạn của pha thiết kế. Tuy nhiên, việc xây dựng từng câu lệnh cụ thể để thể hiện thiết kế này là công việc của pha lập trình. Để tránh lỗi và nâng cao khả

năng bảo trì, việc xây dựng câu lệnh nên tuân theo một quy tắc quan trọng: *mỗi câu lệnh, nên đơn giản và trực tiếp.*

Nhiều ngôn ngữ lập trình cho phép viết nhiều câu lệnh trên một dòng hoặc cho phép viết các câu lệnh phức tạp. Khía cạnh tiết kiệm không gian của tính năng này khó mà biện minh cho tính khó đọc này sinh theo nó. Cấu trúc của khối lệnh cũng như các phép toán điều kiện sẽ dễ dàng bị che lấp bởi cách lập trình như vậy.

Phong cách lập trình có cấu trúc được thể hiện thông qua một giải pháp đơn giản: xây dựng các câu lệnh đơn và viết tụt lề theo các cấp độ ứng với từng khối chức năng.

Các câu lệnh có thể được đơn giản hóa bằng các kỹ thuật:

- *Tránh dùng các phép kiểm tra điều kiện phức tạp:* Phép kiểm tra điều kiện phức tạp sẽ làm tăng khả năng nhầm lẫn khi viết chương trình.
- *Khử bỏ các phép kiểm tra điều kiện phủ định:* Nếu có thể nên thay các điều kiện phủ định bằng điều kiện khẳng định, con người thường tư duy về điều kiện khẳng định dễ hơn nhiều so với chiều ngược lại.
- *Tránh lồng nhau nhiều giữa các điều kiện hay chu trình:* Việc các điều kiện và chu trình lồng nhau sẽ làm tăng độ phức tạp của chương trình và dẫn đến đoạn chương trình khó hiểu, khó bảo trì.
- *Dùng dấu ngoặc để làm sáng tỏ các biểu thức logic hay số học:* Nên đơn giản hóa các biểu thức phức tạp bằng các biểu thức đơn giản, hoặc hạn chế dùng các dấu ngoặc để làm tăng tính tường minh của biểu thức.
- *Dùng dấu cách hoặc các ký hiệu dễ đọc để làm sáng tỏ nội dung câu lệnh:* Nên dùng các dấu cách hoặc các tên biến dễ đọc để làm tăng tính dễ đọc của câu lệnh.
- *Chỉ nên dùng các tính năng chuẩn của ngôn ngữ:* Việc dùng các tính năng chuẩn của ngôn ngữ sẽ làm cho các người lập trình khác dễ dàng hiểu được cũng như tăng tính khả chuyển của chương trình.

Để hướng tới chương trình dễ hiểu luôn nên đặt ra câu hỏi: *Liệu người khác có thể hiểu được đoạn chương trình đó không?* Hay nói tóm lại, cố gắng giữ cho chương trình càng đơn giản càng tốt.

4.3.4. Phong cách vào/ra của các môđun

Vào/ra của các chương trình tương tác với người sử dụng là một phạm trù phức tạp và được bàn đến trong việc thiết kế giao diện người – máy. Ở đây chúng ta quan tâm đến vào/ra của các môđun, hay nói cách khác sự tương tác giữa các môđun phần mềm.

Về cơ bản, thông tin cần trao đổi giữa các môđun được xác định ở pha thiết kế. Tuy nhiên, ở pha lập trình chúng ta cần đảm bảo cho chương trình dễ hiểu và hoạt động tin cậy hơn. Vào/ra giữa các môđun nên tuân thủ theo một số hướng dẫn sau:

- *Kiểm tra tính hợp lệ của dữ liệu vào:* Khi môđun được tích hợp vào một hệ thống phức tạp và trao đổi thông tin với các môđun do người khác thiết kế, chúng ta cần phải kiểm tra tính hợp lệ của thông tin vào và đảm bảo rằng môđun chỉ làm việc với các dữ liệu hợp lệ.
- *Giữ cho định dạng dữ liệu vào đơn giản và thống nhất:* Nên sử dụng các định dạng đơn giản trong việc vào/ra, điều này sẽ làm tăng tính dễ hiểu cũng như giảm chi phí truyền dữ liệu. Đồng thời chúng ta nên thống nhất các quy cách với dữ liệu vào/ra như kiểu và thứ tự của các khoản mục dữ liệu, quy cách của các thông tin về lỗi và ngoại lệ,...
- *Giảm độ ghép nối:* Một trong những phương pháp nâng cao chất lượng phần mềm là giảm độ ghép nối giữa các môđun. Ở pha lập trình, chúng ta có thể giảm độ ghép nối thông qua việc loại bỏ các trường dữ liệu không cần thiết. Ví dụ, có thể dùng các chỉ báo cuối dữ liệu mảng thay vì truyền thêm một tham số "số các khoản mục dữ liệu".

4.4. Lập trình tránh lỗi

4.4.1. Tránh lỗi

Tránh lỗi và phát triển phần mềm ít lỗi là một sự nỗ lực trong toàn bộ quá trình phát triển phần mềm, không chỉ ở pha lập trình. Một phần mềm tốt cần phải được phát triển dựa trên các yếu tố sau:

1. *Sản phẩm của một đặc tả hệ thống chính xác*: Một phần mềm tốt phải được dựa trên sự phân tích và đặc tả chính xác các yêu cầu.
2. *Tăng cường duyệt lại trong quá trình phát triển và thẩm định hệ thống phần mềm*: Tại mỗi pha phát triển (phân tích, thiết kế, cài đặt,...), cần phải thẩm định kết quả cẩn thận để phát hiện các lỗi cũng như để lỗi không lan truyền xuống pha sau.
3. *Tổ chức thử nghiệm một cách có hệ thống*: Cần lập kế hoạch và tiến hành kiểm thử có hệ thống và đầy đủ để tìm ra các lỗi chưa được phát hiện trong quá trình duyệt lại và để định lượng độ tin cậy của hệ thống.
4. *Áp dụng tư duy thiết kế hiện đại* (mô đun hóa và hướng đối tượng) để nâng cao chất lượng phần mềm.

Ở pha lập trình, hai phương pháp chủ đạo trong việc hạn chế lỗi là lập trình thủ tục (lập trình có cấu trúc) và lập trình hướng đối tượng.

Như đã trình bày, lập trình có cấu trúc sử dụng các câu lệnh có cấu trúc và hạn chế dùng lệnh GOTO. Lập trình có cấu trúc buộc người lập trình phải nghĩ cẩn thận về chương trình của họ, nghĩa là phải có một thiết kế ở một mức nào đó trước khi bắt tay vào lập trình, và vì vậy nó ít tạo ra sai lầm trong khi phát triển. Lập trình có cấu trúc làm cho chương trình có thể được đọc một cách tuần tự và do đó dễ hiểu và dễ kiểm tra. Việc thừa nhận lập trình có cấu trúc là quan trọng, bởi vì nó là bước đầu tiên từ cách tiếp cận không khuôn phép tới phát triển phần mềm.

Lập trình hướng đối tượng ngoài việc hỗ trợ mạnh mẽ cho sử dụng lại còn đảm bảo tính an toàn đối với dữ liệu và tính sáng sủa cao. Tính an toàn ở đây là chúng ta chỉ được phép thay đổi trạng thái của đối tượng thông qua các phương thức xác định, do đó hạn chế được việc sinh ra các trạng thái không hợp lệ như gán giá trị ngoài miền dữ liệu hay các khoản mục dữ liệu mâu thuẫn với nhau.

Một nguyên nhân khác hay dẫn đến lỗi chương trình là gán dữ liệu sai kiểu. Để khắc phục vấn đề này, các ngôn ngữ biên dịch bậc cao hiện nay (C++, Java,...) đều kiểm tra kiểu chặt chẽ. Các ngôn ngữ này đồng thời cho phép lập trình viên định ra các kiểu dữ liệu mới. Hầu hết các vi phạm về kiểu dữ liệu đều có thể phát hiện trong quá trình biên dịch chương trình.

Dưới đây là một ví dụ về định kiểu dữ liệu cho một hệ thống điều khiển đèn giao thông:

```
typedef enum { RED, YELLOW, GREEN } TrafficLightColor;  
TrafficLightColor CurrentColor    RED;
```

Thay vì phải dùng số để biểu diễn màu đèn, chúng ta có thể dùng trực tiếp các hằng số có nghĩa là RED, YELLOW và GREEN. Ngoài ra, do biến `CurrentColor` thuộc kiểu `TrafficLightColor` nên chúng ta không thể gán giá trị không hợp lệ (ví dụ, 4) cho biến này.

Ngoài ra, còn có một số yếu tố khác cũng hay dẫn tới các lỗi phần mềm:

- *Các số thực dấu phẩy động*: Các phép toán số thực được làm tròn khiến cho việc so sánh các kết quả số thực, nhất là so sánh bằng giữa hai số thực là không khả thi. Số thực dấu phẩy động có độ chính xác khác nhau khiến cho kết quả phép tính không theo mong muốn. Ví dụ, trong phép tính tích phân chúng ta cần cộng các giá trị nhỏ trước với nhau nếu không chúng sẽ bị làm tròn.
- *Các con trỏ và bộ nhớ động*: Con trỏ là các cấu trúc bậc thấp khó quản lý và dễ gây ra các lỗi nghiêm trọng đối với hệ thống. Việc cấp phát và thu hồi bộ nhớ động phức tạp và là một trong các nguyên nhân chính gây lỗi phần mềm.
- *Song song*: Lập trình song song đòi hỏi kỹ thuật cao và hiểu biết sâu sắc về hệ thống. Một trong các vấn đề phức tạp của song song là quản lý tương tranh đối với các tài nguyên hệ thống (bộ nhớ, các thiết bị ngoại vi,...).
- *Đệ quy*: Sử dụng cấu trúc đệ quy có thể làm cho chương trình đẹp và chặt chẽ về mặt logic. Tuy nhiên, tư duy đệ quy có thể làm cho chương trình khó hiểu, đồng thời nếu kiểm soát không khéo sẽ dẫn đến các lỗi về điều kiện dừng.
- *Các ngắt*: Sử dụng ngắt có thể xây dựng được các ứng dụng thâm nhập các chức năng của hệ thống, tuy nhiên nó đòi hỏi rất thành thạo về hệ điều hành. Với các ứng dụng thông thường, chúng ta nên tránh xa vấn đề này.

Hầu hết các cấu trúc nêu trên đều có ích, tuy nhiên người lập trình cần hiểu rõ mặt trái của chúng và sử dụng chúng một cách cẩn thận. Việc sử

dụng các ngôn ngữ hiện đại cũng góp phần giải quyết các vấn đề này. Ví dụ, ngôn ngữ Java loại bỏ khái niệm con trỏ truyền thống và giải phóng người lập trình khỏi việc quản lý bộ nhớ động. Do đó, tốc độ phát triển cũng như chất lượng của ứng dụng viết bằng Java thường cao hơn so với một số ngôn ngữ biên dịch khác như C và C++.

4.4.2. Lập trình dung thứ lỗi

Đối với các hệ thống đòi hỏi độ tin cậy đặc biệt cao như hệ thống điều khiển máy bay, hay hệ thống kiểm soát lò phản ứng hạt nhân thì cần phải có khả năng dung thứ lỗi (*fault tolerance*), tức là khả năng đảm bảo cho hệ thống vẫn hoạt động được ngay cả khi có thành phần sinh lỗi.

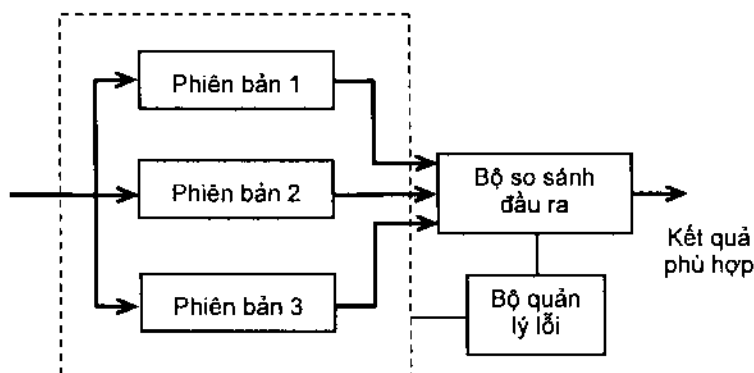
Có bốn hoạt động cần phải tiến hành để hệ thống có khả năng thứ lỗi là:

1. *Phát hiện lỗi*.
2. *Xác định mức độ nghiêm trọng*: Đánh giá mức độ nghiêm trọng của lỗi và xác định xem chương trình có thể tiếp tục thực thi được hay không.
3. *Phục hồi sau khi gặp lỗi*: Hệ thống phải phục hồi về trạng thái mà nó biết là an toàn. Cũng có thể là chỉnh lý trạng thái bị hủy hoại (phục hồi tiến), cũng có thể là lui về một trạng thái trước đó là an toàn (phục hồi lùi).
4. *Chữa lỗi*: Cải tiến hệ thống để cho lỗi đó không xuất hiện nữa. Tuy nhiên, trong nhiều trường hợp phát hiện được đúng nguyên nhân gây lỗi là rất khó khăn vì nó xảy ra bởi một tổ hợp của thông tin vào và trạng thái của hệ thống.

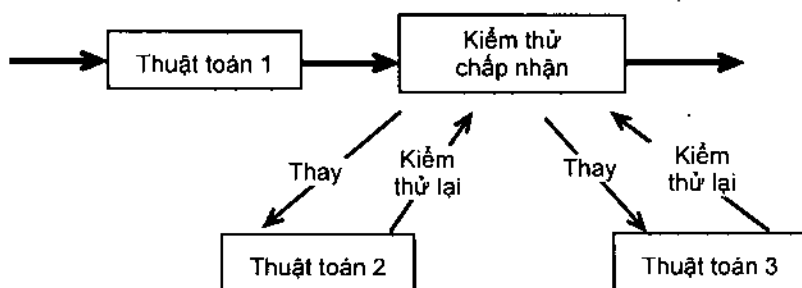
Có hai cách tiếp cận chính để xây dựng một hệ thống dung thứ lỗi là:

1. *Lập trình N – phiên bản* (N – version programming): Phương pháp này lấy ý tưởng từ dung thứ lỗi phần cứng thông qua việc song song hóa các khối chức năng và so sánh kết quả. Chức năng phần mềm được cài đặt bằng nhiều phiên bản có thể được tạo ra bởi các nhóm lập trình khác nhau. Người ta so sánh đầu ra của chúng và dựa trên nguyên tắc số lớn để quyết định đầu ra cuối cùng (hình 4.2). Avizienis [1] đã mô tả cách tiếp cận này.

2. **Khối phục hồi (Recovery blocks):** Phương pháp này ứng dụng trong một số trường hợp mà tồn tại phép thử đơn giản kiểm tra đầu ra có đúng đắn hay không, ví dụ như kiểm tra xem một mảng có được sắp xếp hay không. Khi đó, nếu với một đầu vào đặc thù nào đó mà khối chức năng thông thường bị lỗi thì nó sẽ được chạy lại với khối dự bị (khối phục hồi). Thông thường khối phục hồi sẽ được cài đặt bằng một thuật toán khác để tránh tình trạng các nhóm lập trình khác nhau có cùng sai sót với cùng thuật toán. Có thể có nhiều khối phục hồi khác nhau (hình 4.3). Randel và Xu [30] đã mô tả phương pháp cho khối phục hồi này.



Hình 4.2. Lập trình n – phiên bản



Hình 4.3. Khối phục hồi

4.4.3. Lập trình phòng thủ

Lập trình phòng thủ (defence programming) cũng là một kỹ thuật lập trình dung thứ lỗi nhưng không cần đến các khối chương trình chuyên dụng như trong mục trên. Về cơ bản, việc ngăn ngừa lỗi ở đây vẫn là quá trình phát hiện lỗi, đánh giá lỗi và phục hồi (tìm cách tiếp tục thực thi) sau lỗi.

Khác với hai phương pháp, lập trình thứ lỗi chủ yếu tập trung vào khắc phục các lỗi sai sót chủ quan của người phát triển (lỗi về thuật toán hoặc lỗi về cài đặt), lập trình phòng thủ chú trọng nhiều đến các lỗi mang tính khách quan khi thực hiện chương trình, ví dụ như lỗi vào/ra, lỗi môi trường thực hiện (thiếu bộ nhớ,...), hay một số lỗi đặc thù về số học và logic như lỗi tràn số, lỗi chia cho số không. Về một khía cạnh nào đó, đây là các lỗi (hay các trường hợp bất thường) mà chúng ta có thể dự đoán được.

Thông thường, lập trình phòng thủ dựa trên việc kiểm tra trạng thái hệ thống sau khi biến đổi và tìm cách đảm bảo rằng sự biến đổi trạng thái là đúng đắn. Nếu phát hiện một mâu thuẫn thì hủy bỏ việc biến đổi trạng thái, quay lại trạng thái đúng đắn trước đó (phục hồi lỗi) và tìm cách thực hiện lại, có thể với một bộ dữ liệu khác.

Một trong những nguyên nhân gây ra lỗi trong biến đổi trạng thái là dữ liệu được gán sai kiểu hoặc ngoài miền giá trị. Vấn đề này được giải quyết bằng các kỹ thuật đặc tả miền giá trị (đặc tả kiểu) trong các ngôn ngữ bậc cao, hoặc áp dụng bao gói/che dấu thông tin trong lập trình hướng đối tượng.

Phục hồi lỗi là một quá trình cải biên không gian trạng thái của hệ thống sao cho hiệu ứng của lỗi là nhỏ nhất và hệ thống có thể tiếp tục vận hành, có thể là ở một mức hạn chế. Có hai kỹ thuật phục hồi là: (i) *phục hồi tiến* nhằm điều chỉnh lại trạng thái hệ thống rồi tiếp tục, và (ii) *phục hồi lùi* là quay lại một trạng thái đúng đã biết ngay trước đó rồi tiến hành tiếp.

Phục hồi tiến thường được ứng dụng trong một số trường hợp đặc biệt như:

1. *Khôi phục lại dữ liệu*: Sử dụng kỹ thuật mã hóa thích hợp bằng cách thêm các dữ liệu dư thừa vào dữ liệu cho phép sửa sai khi phát hiện lỗi.
2. *Khôi phục lại cấu trúc*: Nếu các con trỏ tiến và lùi đã có trong cấu trúc dữ liệu thì cấu trúc đó có thể tái tạo ngay cả khi một số con trỏ bị sai giá trị. Kỹ thuật này thường được dùng cho việc sửa chữa hệ thống tệp và cơ sở dữ liệu.

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

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

4.4.4. Xử lý ngoại lệ

Các ngôn ngữ lập trình hướng đối tượng (C++, Java,...) đưa ra kỹ thuật xử lý ngoại lệ để hỗ trợ việc tránh lỗi và phục hồi sau lỗi. Ý tưởng của xử lý ngoại lệ là tách khỏi mã lệnh liên quan đến các trường hợp bất bình thường ra khỏi các mã lệnh làm việc với trường hợp bình thường. Do đó, xử lý ngoại lệ không những hỗ trợ người lập trình xử lý một cách hiệu quả hơn với lỗi mà còn làm cho chương trình sáng sủa hơn và hạn chế các lỗi cấp hai do việc thêm vào các mã xử lý lỗi.

Về cơ bản, xử lý ngoại lệ bao gồm hai bước:

1. *Ném ngoại lệ*: Phát hiện và *ném ngoại lệ* khỏi khối chương trình bình thường.
2. *Bắt ngoại lệ*: *Bắt ngoại lệ* và xử lý chúng tại vị trí mong muốn.

Dưới đây là một ví dụ về xử lý ngoại lệ trong Java. Các lệnh liên quan đến tệp và đọc dữ liệu trong khối lệnh try có thể sinh ra ngoại lệ. Khi đó các ngoại lệ được truyền đến khối lệnh catch và được xử lý một cách thích hợp.

```
try{
    File fi = new File("data.txt");
    Scanner sc = new Scanner(fi);
    while (sc.hasNext())
    {
        System.out.println(sc.nextInt());
    }
}
catch (IOException e)
{
    System.err.println(e);
}
catch (Exception e)
{
    System.err.println(e);
}.
```

4.5. Lập trình hướng hiệu quả thực hiện

4.5.1. Tính hiệu quả chương trình

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

- *Đơn giản hóa các biểu thức số học và logic:* Trong nhiều trường hợp, chúng ta có thể biến đổi (rút gọn) các biểu thức phức tạp thành các biểu thức đơn giản tương đương và sẽ làm giảm đáng kể lượng tính toán.
- *Tối ưu khối lệnh trong các chu trình:* Các chu trình chiếm thời gian đáng kể trong thực hiện chương trình. Chúng ta cần tối ưu các khối lệnh trong chu trình và chuyển ra ngoài tất cả các câu lệnh độc lập với chu trình. Cũng có thể tính sẵn các giá trị hay biểu thức được lặp đi lặp lại nhiều lần.
- *Tránh dùng mảng nhiều chiều:* Mảng nhiều chiều có thể là một nguyên nhân làm giảm tốc độ chương trình do phải tính toán vị trí bộ nhớ thực từ các chỉ số khác nhau.
- *Tránh dùng con trỏ và danh sách phức tạp:* Việc sử dụng con trỏ và các danh sách móc nối phức tạp cũng là một nguyên nhân dẫn đến giảm tốc độ thực hiện do phải tính toán địa chỉ thực thông qua các địa chỉ gián tiếp.
- *Dùng các phép toán nhanh:* Các phép toán số học hoặc logic nhanh (các phép toán một ngôi như tăng, giảm một đơn vị, dịch chuyển bit,...) có thể làm tăng đáng kể tốc độ, đặc biệt là khi chúng được sử dụng nhiều lần trong các vòng lặp.
- *Tránh trộn lẫn các kiểu dữ liệu, nên sử dụng các kiểu đơn giản:* Việc trộn lẫn các kiểu dữ liệu khiến cho chương trình phải đổi kiểu và thực hiện phép toán với các kiểu phức tạp, điều này cũng làm tăng khối lượng tính toán.

- *Tránh gọi các chương trình con*: Lập trình thủ tục dựa trên việc gọi các chương trình con. Tuy nhiên, trong một số trường hợp, cần tối ưu tốc độ, nhất là trong các vòng lặp, người ta tìm cách hạn chế việc gọi các chương trình con, nhất là các chương trình con có nội dung đơn giản như so sánh, tính min, max,... Gọi chương trình con sinh ra các chi phí về truyền/nhận dữ liệu, đồng thời phá vỡ tính cục bộ của khối dữ liệu đệm (cache). Một số ngôn ngữ bậc cao (C/C++) hỗ trợ các kỹ thuật để tránh gọi chương trình con nhỏ, như tạo ra các *macro* hoặc các hàm *inline*.

Nhiều trình biên dịch có tính năng tối ưu tự động và sinh ra chương trình hiệu quả bằng cách nén các biểu thức lặp, thực hiện tính chu trình và áp dụng các thuật toán có hiệu quả liên quan khác. Một số trình biên dịch còn có khả năng tối ưu mã đích với các kiến trúc phần cứng cụ thể. Với những ứng dụng trong đó tính hiệu quả có ý nghĩa quan trọng, những trình biên dịch như thế là công cụ lập trình không thể thiếu được.

4.5.2. Hiệu quả bộ nhớ

Tính hiệu quả trong sử dụng bộ nhớ được tính toán dựa vào đặc trưng sử dụng bộ nhớ đệm của hệ điều hành. Nói chung, tính cục bộ của chương trình và dữ liệu (tính môđun hóa cao) là một phương pháp tuyệt vời làm tăng tính hiệu quả của sử dụng bộ nhớ đệm và tăng tốc độ thực hiện của chương trình.

Hạn chế bộ nhớ trong phát triển phần mềm nhúng là mối quan tâm rất thực tế, mặc dù những tiến bộ trong chế tạo thiết bị vi điện tử ngày càng làm giảm giá thành chế tạo và tăng dung lượng của bộ nhớ một cách đáng kể. Nếu yêu cầu hệ thống sử dụng bộ nhớ tối thiểu (khi sản xuất đại trà sản phẩm với giá thành thấp và tốn ít năng lượng) thì chúng ta cần các trình biên dịch đặc thù với tính năng nén bộ nhớ, hay phải sử dụng các ngôn ngữ bậc thấp gần với hợp ngữ.

Không giống như nhiều đặc trưng của các hệ thống khác phải trả giá bù trừ cho nhau, các kỹ thuật cho hiệu quả về thời gian thực hiện đôi khi có thể dẫn tới hiệu quả sử dụng bộ nhớ. Chẳng hạn, việc sử dụng các mảng một chiều hay các phép toán đơn giản không những làm tăng tốc độ mà còn tăng

hiệu quả sử dụng bộ nhớ. Có thể nói, chìa khóa ở đây là giữ cho chương trình đơn giản.

4.5.3. Hiệu quả vào/ra

Một trong những khâu lãng phí tốc độ là việc vào/ra dữ liệu. Các thiết bị vào/ra thường có tốc độ chậm hơn rất nhiều so với khả năng tính toán của máy tính và tốc độ truy cập bộ nhớ trong. Do đó, việc tối ưu vào/ra có thể làm tăng đáng kể tốc độ thực hiện. Dưới đây là một số hướng dẫn đơn giản để tăng cường hiệu quả vào/ra:

- *Số các yêu cầu vào/ra nên giữ mức tối thiểu:* Do tốc độ của các thiết bị vào/ra chậm hơn nhiều so với tốc độ thực hiện bên trong, nên chúng ta cần tối thiểu hóa các yêu cầu về vào/ra, ví dụ như hạn chế việc đưa các thông báo trạng thái không cần thiết ra màn hình. Một cách thức quan trọng khác là gom góp các yêu cầu vào/ra thành khối để tối thiểu hóa số lần thực hiện các yêu cầu vào/ra. Việc vào/ra theo khối đối với một số thiết bị như đĩa cứng còn có một ý nghĩa quan trọng vì bản thân thiết bị này luôn đọc/ghi theo từng nhóm sector cho dù ta chỉ yêu cầu đọc một byte duy nhất.
- *Mọi việc vào/ra nên qua bộ đệm:* Do các thiết bị vào/ra hoạt động chậm hơn và hoạt động độc lập với các xử lý bên trong, nên thông thường chúng ta cần sử dụng các bộ đệm để giảm chi phí thực hiện vào/ra và tránh lỗi. Việc sử dụng bộ đệm và vào/ra theo khối có thể làm giảm thời gian thực hiện một cách đáng kể.
- *Nên tối ưu thao tác vào/ra với từng chủng loại thiết bị đầu cuối,* ví dụ như màn hình, máy in, mạng hay các loại bộ nhớ ngoài.

4.6. Một số môi trường phát triển

4.6.1. Môi trường VISUAL STUDIO.NET

a) Môi trường tích hợp

Visual Studio và gần đây là Visual Studio.NET là một tập các công cụ phát triển hoàn chỉnh để xây dựng các ứng dụng chạy trên nền Windows (sử dụng Windows Forms, một phần của .NET Framework), Web (sử dụng

ASP.NET và XML Web services), các ứng dụng desktop và các ứng dụng thiết bị thông minh như PDAs, mobile phones, Pocket PC,... (sử dụng .NET Compact Framework). Web ASP, XML Web services, Visual Basic.NET, Visual C++.NET, Visual C#.NET và Visual J#.NET tất cả cùng sử dụng một môi trường phát triển tích hợp (*IDE – Integrated Development Environment*). Nó cho phép chia sẻ các công cụ và tiện ích trong việc tạo ra các giải pháp với nhiều ngôn ngữ. Có thể nói, VS.NET là một môi trường tích hợp nhiều ngôn ngữ lập trình, là sự mở rộng của các ngôn ngữ hiện hành với cú pháp tương đương, chạy trên nền .NET framework.

Visual J# là một công cụ phát triển mà những người lập trình đã quen với ngôn ngữ Java có thể sử dụng, để xây dựng các ứng dụng và dịch vụ trên nền .NET Framework. Nó tích hợp cú pháp của ngôn ngữ Java vào môi trường phát triển tích hợp Visual Studio.NET. Visual J# cũng hỗ trợ hầu hết các chức năng có trong Visual J++ 6.0. Tuy nhiên, Visual J# không phải là một công cụ phát triển chạy trên máy ảo Java, các ứng dụng viết bằng Visual J# chỉ có thể thực thi trên .NET Framework.

Visual C# là một ngôn ngữ lập trình mới được phát triển trong môi trường VS.NET, thích hợp cho việc xây dựng các ứng dụng doanh nghiệp, với cú pháp đơn giản tương tự Visual Basic. C# cũng chứa một số thành phần giống Java, bao gồm cú pháp, bộ thu gom rác và không gian tên phân cấp.

b) NET Framework

Thay đổi lớn nhất của VS.Net là đưa ra môi trường phát triển các mã chương trình được quản lý (managed code), sử dụng .NET Framework. Các chương trình được phát triển sử dụng .NET không được biên dịch sang ngôn ngữ máy (ví dụ như C++) mà thay vào đó chúng được biên dịch sang một dạng ngôn ngữ trung gian là MSIL (*Microsoft Intermediate Language*) hoặc CLI (*Common Language Infrastructure*). Khi một ứng dụng MSIL được thực thi, nó được biên dịch sang ngôn ngữ máy thích hợp với phần nền mà nó đang chạy trên đó. Do đó, mã chương trình có thể được thực thi trên nhiều nền. Có thể chạy MSIL trên Linux hoặc Mac OS X sử dụng các cài đặt non – Microsoft.NET như Mono và DotGNU.

c) Giao diện thân thiện với người dùng

Visual Studio.NET đem đến cho các nhà lập trình cũng như người phát triển một giao diện làm việc thân thiện và sự dính kết tốt hơn các thành phần. Nó tùy biến hơn cho người dùng với các điều khiển, các thanh công cụ kéo thả và các cửa sổ công cụ tự động ẩn đi khi không sử dụng tới để tiết kiệm không gian làm việc. Với việc tích hợp các điều khiển (tệp dll) của người dùng vào thanh công cụ, nó đem lại cho người dùng khả năng thay đổi, bổ sung theo ý muốn các điều khiển phù hợp với ứng dụng của mình.

d) Tích hợp công cụ hỗ trợ phân tích, thiết kế

VS.NET cung cấp các mô hình lập trình phong phú (còn được gọi là các mô hình đối tượng tự động – *Automation object models*) hỗ trợ cho việc mở rộng và tự động hóa môi trường phát triển. Mỗi mô hình tập trung vào một lĩnh vực cụ thể của IDE như tool Windows, code editor, các ấn bản dành cho các nhà phát triển phần mềm, các nhà kiểm thử phần mềm, các chuyên gia cơ sở dữ liệu,...

Bằng cách sử dụng *Visual Paradigm Smart Development Environment – SDE*, có thể chuyển các thiết kế hệ thống (sơ đồ UML) sang mã cài đặt chương trình (mã nguồn của C#, C++ hoặc VB.NET,...). Ví dụ, sinh mã C# từ sơ đồ lớp và ngược lại sinh sơ đồ lớp từ mã C# cũng như tạo ra các báo cáo.

e) Khả năng mở rộng

Visual Studio cung cấp các tiện ích cho phép các nhà lập trình viết các add – ins để mở rộng khả năng của nó. Những thành phần tích hợp này được "gắn" vào Visual Studio và cung cấp các tiện ích mà bản thân Visual Studio không có. Ví dụ, một số thành phần tích hợp cung cấp các công cụ lập trình cải tiến, hoặc các tính năng phát triển ứng dụng nhanh, các công cụ làm báo cáo và thiết kế cơ sở dữ liệu trực quan,...

4.6.2. Môi trường lập trình Java

Java là ngôn ngữ lập trình được sử dụng khá phổ biến hiện nay. Ưu điểm chính của Java là khả năng chạy độc lập nền và tính hướng đối tượng hoàn toàn. Vì thế, độ an toàn cũng như khả năng sử dụng lại mã của Java là rất cao.

Để biên dịch và chạy các chương trình Java, cần phải cài đặt "bộ công cụ phát triển Java" (*Java Development Kit – JDK*). Các hệ điều hành khác nhau sẽ sử dụng các bản JDK tương ứng với nó. Bản JDK dành cho Windows, Linux và Solaris có thể được tải miễn phí tại trang chủ của Sun (<http://java.sun.com>). Hiện nay Sun đã phát hành phiên bản JDK 5.0 với một số thay đổi về ngôn ngữ cũng như thêm vào một số lớp giao diện nhằm hỗ trợ các công việc như lập trình đồng thời (concurrent programming), quản lý framework từ xa (remote management framework),... Bộ công cụ phát triển cơ sở được phần lớn các lập trình viên sử dụng là J2SE (*Java 2 Platform, Standard Edition*). Bên cạnh đó Sun cũng cung cấp thêm các bộ công cụ chuyên biệt khác như J2EE (*Java 2 Platform, Enterprise Edition*), J2ME (*Java 2 Platform, Micro Edition*),...

Không như các môi trường phát triển thường gặp khác như Visual Basic hay Visual C++..., JDK không tích hợp sẵn các trình soạn thảo hỗ trợ dịch, chạy, hay gỡ lỗi từ bảng thực đơn. Các thao tác này đều được thực hiện từ dòng lệnh của các shell windows. Phương pháp sử dụng dòng lệnh hình như sẽ làm giảm hiệu quả lập trình vì đòi hỏi phải nhớ nhiều cú pháp lệnh. Tuy nhiên trong thực tế, các lập trình viên chuyên nghiệp lại thích sử dụng phương pháp này vì nó nhanh hơn, linh hoạt hơn và đòi hỏi ít tài nguyên hơn so với sử dụng các giao diện đồ họa.

Để hỗ trợ cho những người mới dùng tiếp cận với Java cũng như để giúp quản lý các dự án lớn, một số công ty đã phát triển các bộ môi trường phát triển tích hợp (IDE). Các bộ công cụ này có nhiều mức khác nhau và thích hợp với từng công việc cụ thể. Các bộ soạn thảo đơn giản đòi hỏi ít tài nguyên như TextPad, EditPlus, JCreator (môi trường Windows) hay Emacs, Vi (môi trường Linux) thường chỉ hỗ trợ các công việc đơn giản như đổi màu từ khóa, biên dịch hay chạy chương trình từ thực đơn. Trong khi đó, một số các môi trường mạnh hơn, đòi hỏi nhiều tài nguyên hơn như IntelliJ, JBuilder, NetBean, Eclipse,... cho phép thực hiện những công việc phức tạp hơn như xây dựng giao diện đồ họa bằng kéo thả, quản lý các dự án lớn nhiều tệp, quản lý phiên bản (Concurrent Versions System – CVS),...

Một trong số các IDE được nhiều lập trình viên ưa thích hiện nay là Eclipse. Đây là một IDE mã nguồn mở do Công ty IBM phát triển. Eclipse cung cấp sẵn một số công cụ cơ bản như hỗ trợ biên dịch, gỡ lỗi, hoặc hỗ trợ

quản lý, bao gói các dự án lớn. Điểm thú vị nhất ở Eclipse là môi trường này hoạt động với một framework chung, cho phép người dùng mở rộng, tích hợp các plug – in từ bên ngoài. Qua đó người sử dụng có thể dễ dàng xây dựng mới hoặc sử dụng lại các công cụ phần mềm của hãng thứ ba ngay trong môi trường Eclipse. Hiện tại, Eclipse đã có đến phiên bản 3.2 và đã hỗ trợ cho nhiều môi trường khác nhau như Windows, Linux, HP – UX, AIX, Solaris, QNX, Mac OS X,... Toàn bộ chương trình, mã nguồn cũng như tài liệu trợ giúp về Eclipse có thể được tải về miễn phí từ địa chỉ <http://www.eclipse.org>.

Tổng kết

- ◆ Bước lập trình là một tiến trình chuyển hóa thiết kế chi tiết thành chương trình, cuối cùng được biến đổi thành các lệnh mã máy thực hiện được. Lập trình là khâu không thể thiếu trong tiến trình kỹ nghệ phần mềm và có ảnh hưởng lớn đến chất lượng phần mềm.
- ◆ Có rất nhiều ngôn ngữ lập trình với các đặc trưng và miền ứng dụng khác nhau. Các đặc trưng của ngôn ngữ lập trình có ảnh hưởng lớn đến quá trình xây dựng, kiểm thử cũng như bảo trì phần mềm. Lựa chọn ngôn ngữ lập trình cho một ứng dụng cần xem xét đến đặc trưng của mỗi ngôn ngữ.
- ◆ Phong cách lập trình quyết định tính dễ hiểu của chương trình gốc. Các yếu tố của phong cách bao gồm việc lập tài liệu bên trong, phương pháp khai báo dữ liệu, thủ tục, xây dựng câu lệnh và giao diện (vào/ra) của các môđun.
- ◆ Nhiều chương trình cần độ tin cậy cao. Lập trình thử lỗi (*phát hiện lỗi, đánh giá mức thiệt hại, phục hồi, chữa lỗi*), lập trình phòng thủ (như thử lỗi, nhưng bỏ qua sửa lỗi) và xử lý ngoại lệ là những kỹ thuật cơ bản đảm bảo độ tin cậy cho chương trình
- ◆ Lập trình cần hướng tới hiệu quả thực hiện, tức là tiết kiệm tài nguyên phần cứng (mức độ sử dụng CPU, bộ nhớ,...). Mặc dù tính hiệu quả có thể là yêu cầu cực kỳ quan trọng, chúng ta nên nhớ rằng, một chương trình hoạt động hiệu quả mà lại không dễ hiểu, dẫn đến khó bảo trì thì giá trị của nó cũng bị hạn chế.

- ♦ Môi trường phát triển là một phương tiện để trợ giúp đặc lực cho tiến trình phát triển một ứng dụng một cách hiệu quả. Cần hiểu biết các môi trường phát triển khác nhau để tận dụng khi phát triển một hệ thống, đặc biệt trong giai đoạn lập trình.

Câu hỏi ôn tập

1. Nêu các đặc trưng cơ bản của ngôn ngữ lập trình.
2. Phân loại các ngôn ngữ lập trình và cách lựa chọn ngôn ngữ cho ứng dụng.
3. Trình bày về các phương pháp lập trình phổ biến.
4. Nêu các đặc trưng của phương pháp lập trình hướng đối tượng.
5. Nêu các nội dung liên quan đến phong cách lập trình.
6. Trình bày nội dung của lập trình tránh lỗi, lập trình thứ lỗi và xử lý ngoại lệ.
7. Nêu các kỹ thuật dùng để lập trình tránh lỗi và thứ lỗi.
8. Phân biệt hai kỹ thuật lập trình thứ lỗi: lập trình n – phiên bản và khối phục hồi.
9. Những nội dung gì đặt ra liên quan đến lập trình hướng hiệu quả thực hiện?
10. Môi trường phát triển đặc trưng bằng các yếu tố gì? Chúng trợ giúp gì cho giai đoạn lập trình?

Chương 5

XÁC MINH VÀ THẨM ĐỊNH

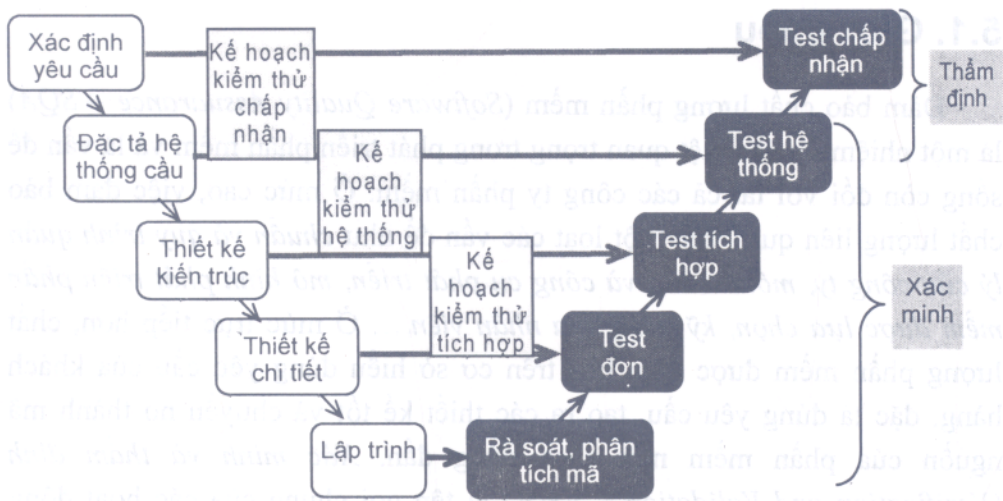
5.1. Giới thiệu

Đảm bảo chất lượng phần mềm (*Software Quality Assurance – SQA*) là một nhiệm vụ đặc biệt quan trọng trong phát triển phần mềm và là vấn đề sống còn đối với tất cả các công ty phần mềm. Ở mức cao, việc đảm bảo chất lượng liên quan đến một loạt các vấn đề như *chuẩn và quy trình quản lý của công ty, môi trường và công cụ phát triển, mô hình phát triển phần mềm được lựa chọn, kỹ năng của nhân viên,...* Ở mức trực tiếp hơn, chất lượng phần mềm được đảm bảo trên cơ sở hiểu đúng yêu cầu của khách hàng, đặc tả đúng yêu cầu, tạo ra các thiết kế tốt và chuyển nó thành mã nguồn của phần mềm một cách đúng đắn. *Xác minh và thẩm định (Verification and Validation – V&V)* là tên gọi chung của các hoạt động SQA, nhằm đảm bảo phần mềm được phát triển đúng theo đặc tả và đáp ứng tốt các yêu cầu người dùng. Tuy nhiên, V&V là hai hoạt động khác nhau. Theo như phát biểu của Boehm [4] có thể giải thích:

- *Xác minh* là sự kiểm tra xem sản phẩm có đúng với đặc tả hay không, tức là chú trọng vào việc phát hiện lỗi của phần mềm qua mỗi bước phát triển.
- *Thẩm định* là sự kiểm tra xem sản phẩm có đáp ứng được yêu cầu người dùng hay không, tức là chú trọng vào việc phát hiện sự khác biệt của sản phẩm làm ra với những gì người dùng mong đợi.

V&V gồm một loạt các hoạt động của SQA, bao gồm rà soát kỹ thuật, kiểm toán cấu hình và kiểm toán chất lượng, giám sát hiệu năng, nghiên cứu khả thi và mô phỏng, rà soát tài liệu, rà soát cơ sở dữ liệu, phân tích thuật

toán, kiểm thử mã nguồn, kiểm thử chất lượng và kiểm thử cài đặt [39]. Chúng được tiến hành ở mọi giai đoạn của tiến trình phát triển phần mềm: ở khâu phân tích, ta cần thẩm định đặc tả yêu cầu; ở khâu thiết kế, ta cần xét duyệt xem thiết kế có đúng với đặc tả không; ở khâu lập mã, ta cần rà soát xem mã nguồn có đúng thiết kế chi tiết không,... Có thể hình dung, V&V bao trùm lên các bước của tiến trình phát triển phần mềm và được lập kế hoạch song song với kế hoạch phát triển phần mềm. Hình 5.1 mô tả mô hình chữ "V" về phạm vi của V&V cũng như mối quan hệ giữa V&V với tiến trình phát triển phần mềm.



Hình 5.1. Mô hình phát triển "V"

Tùy theo thời điểm và đối tượng của hoạt động kiểm tra mà người ta còn phân biệt V&V tĩnh và V&V động.

- *Xác minh và thẩm định tĩnh* là sự kiểm tra phần mềm mà không thực hiện chương trình. Ví dụ như việc *xét duyệt yêu cầu*, *xét duyệt thiết kế*, *thanh tra mã nguồn*, *sử dụng các biến đổi hình thức (suy luận)* để kiểm tra tính đúng đắn của chương trình. Với mỗi loại hình sản phẩm ở mỗi bước, người ta thường lập ra một danh mục cần kiểm tra nó và sử dụng một bảng câu hỏi (*bảng kiểm – check list*) để kiểm tra các đặc tả theo danh mục này. Danh mục này thường thay đổi tùy theo ngôn ngữ được sử dụng ở mỗi bước. Về lý thuyết, thông qua *xác minh và thẩm định tĩnh* có thể phát hiện được hầu hết các lỗi của

từng bước phát triển. Tuy nhiên, phương pháp này không cho phép đánh giá một cách tổng thể tính hiệu quả của phần mềm.

- *Xác minh và thẩm định động* là sự kiểm tra thông qua việc thực hiện chương trình, được tiến hành sau khi đã xây dựng được chương trình (mã nguồn). Đây là kỹ thuật kiểm tra được áp dụng phổ biến nhất và thực hiện từ rất sớm.

Trong chương này, chúng ta đi sâu tìm hiểu về xác minh và thẩm định động, thường được gọi là kiểm thử chương trình (*testing*).

5.2. Các khái niệm về kiểm thử

Theo Glen Myers [21], *kiểm thử phần mềm là quá trình vận hành chương trình để tìm ra lỗi. Một ca kiểm thử tốt là ca kiểm thử có xác suất cao tìm ra một lỗi chưa được phát hiện. Một ca kiểm thử thắng lợi là ca kiểm thử làm lộ ra được ít nhất một lỗi còn chưa được phát hiện.* Với cách nhìn này, *mục tiêu của chúng ta là thiết kế các ca kiểm thử để có thể phát hiện một cách có hệ thống các loại lỗi khác nhau với chi phí thời gian và công sức ít nhất có thể.*

Ở đây, cần phân biệt ba khái niệm: *Hỏng* là hành vi không chấp nhận được mà hệ thống để lộ ra; *Lỗi* là khiếm khuyết trong bất kỳ khía cạnh nào của hệ thống mà đóng góp vào việc xuất hiện của một hay nhiều cái hỏng; còn *Sai* là quyết định không thích hợp do kỹ sư phần mềm làm dẫn tới việc đưa vào lỗi [36]. Phát hiện lỗi là công việc của kiểm thử. Tìm ra cái hỏng và sai để sửa là việc của người lập trình và gỡ lỗi.

Thiết kế các ca kiểm thử thành công sẽ phát hiện ra lỗi của phần mềm. Ngoài ra nó còn mang lại những lợi ích phụ như thuyết minh rằng, các chức năng phần mềm tương ứng với đặc tả, các yêu cầu thực thi là phù hợp, cung cấp thêm các chỉ số độ tin cậy và chỉ số chất lượng phần mềm nói chung. Tuy nhiên, kiểm thử không thể khẳng định được rằng, không còn lỗi trong chương trình (mặc dù có thể nó đã thoả mãn tất cả các phép thử được thực hiện).

Kiểm thử phần mềm không phải là gỡ lỗi: Quá trình gỡ lỗi dùng để loại bỏ các lỗi khi biết chúng tồn tại trong chương trình (nhờ kiểm thử). Quá

trình gỡ lỗi không cần viết tài liệu, không cần đặc tả, theo dõi cũng như viết báo cáo.

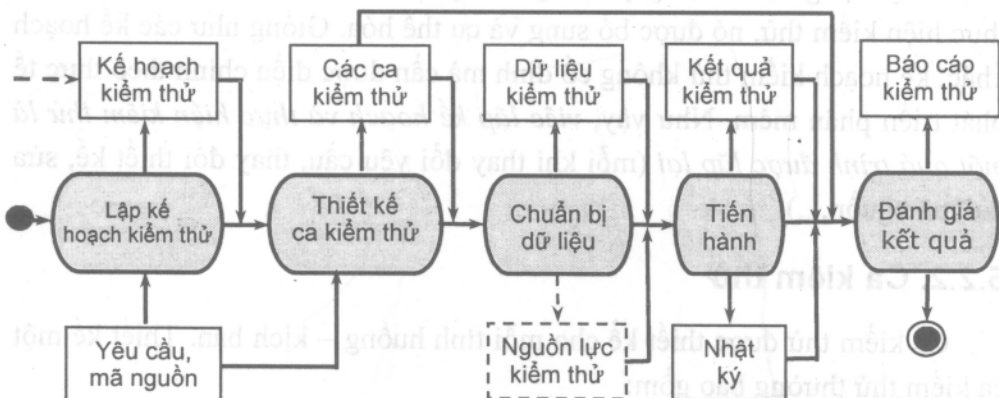
Phần mềm vốn phức tạp và vô hình, khi phát triển nó chịu ảnh hưởng của rất nhiều yếu tố, đặc biệt phụ thuộc vào chủ quan của người tạo ra nó, nên không tránh khỏi sai sót. Mặt khác, phần mềm là phân tử logic (các dòng lệnh), nói chung sẽ không thay đổi chừng nào chưa được nâng cấp hay bị lỗi thời. Cho nên, một khi phần mềm được gửi đi, lỗi thiết kế sẽ bị chôn vùi và vẫn còn tiềm tàng ở đó cho tới khi được kích hoạt. Lỗi phần mềm luôn tồn tại trong bất kỳ mô đun phần mềm nào với kích cỡ vừa phải: không phải vì người lập trình bất cẩn hay thiếu trách nhiệm, mà bởi vì độ phức tạp của phần mềm nói chung là khó nắm bắt và con người chỉ có khả năng giới hạn để quản lý độ phức tạp này. Điều đó cũng đúng với bất kỳ một hệ thống phức tạp nào, lỗi thiết kế có thể chẳng bao giờ được tìm ra khi vận hành chưa nhiều. Vì vậy, khi kiểm thử, nhà phát triển muốn nhìn thấy phần mềm của mình như một sản phẩm thực thụ của một hệ thống vận hành hoàn thiện. Kiểm thử giúp phát hiện ra những sai sót, tạo điều kiện để khắc phục lỗi, hoàn thiện chương trình. Nhờ vậy giảm chi phí phát triển và chi phí bảo trì. Cuối cùng, chính nhờ kiểm thử mà người ta có kế hoạch tốt để nâng cao chất lượng phần mềm trong suốt tiến trình phát triển.

Quá trình kiểm thử là một hoạt động chính thức. Nó thường bao gồm một chiến lược và một phương pháp tiếp cận có hệ thống. Các mức kiểm thử khác nhau được tiến hành theo những chiến lược khác nhau và áp dụng các phương pháp khác nhau, chúng bổ sung cho nhau. Mọi quá trình kiểm thử luôn được chuẩn bị, được đặc tả và ghi chép lại. Đó là một quá trình phức tạp và tốn kém.

Kiểm thử phần mềm có tầm quan trọng rất lớn. Nó chiếm 40% tổng công sức phát triển, chiếm hơn 30% tổng thời gian phát triển. Đặc biệt với các phần mềm ảnh hưởng tới sinh mạng thì chi phí có thể gấp từ 3 đến 5 lần tổng các chi phí khác cộng lại.

Theo Gregg Rothermel [31], *mục đích thiết kế các ca kiểm thử có xác suất tìm ra sai cao, với nỗ lực và thời gian ít nhất*. Trong các thập niên 80 – 90 của thế kỷ XX, người ta đã nghiên cứu nhiều phương pháp thiết kế các ca kiểm thử. Theo đó, một phương pháp tốt phải đảm bảo:

- Tiến trình kiểm thử có thể mô tả như sơ đồ hình 5.2.



Hình 5.2. Sơ đồ tiến trình kiểm thử

5.2.1. Kế hoạch kiểm thử

Kế hoạch kiểm thử đã được Frewin và Hatton [12] mô tả, theo đó nó cần có các nội dung sau:

1. *Tiến trình kiểm thử*: Mô tả các pha chính của việc kiểm thử.
2. *Tính lần vết được các yêu cầu*: Người dùng quan tâm đến việc kiểm thử đáp ứng được các yêu cầu của họ. Vì vậy trong kế hoạch, mỗi yêu cầu cần tìm được một kiểm thử tương ứng và ngược lại (nhờ bảng lần vết).
3. *Các khoản mục kiểm thử*: Sản phẩm phần mềm cần kiểm thử phải được đặc tả.
4. *Lịch trình kiểm thử*: Một lịch trình kiểm thử tổng thể và sự phân bổ các nguồn lực cho các hoạt động kiểm thử theo lịch trình.
5. *Các thủ tục ghi lại việc kiểm thử*: Các kết quả kiểm thử phải được ghi lại một cách hệ thống. Nó có thể dùng để kiểm toán tiến trình kiểm thử nhằm biết được kiểm thử nào đã được tiến hành đúng đắn.
6. *Các yêu cầu về phần cứng, phần mềm* cho kiểm thử.

7. *Các ràng buộc*: Các ràng buộc tác động đến tiến trình kiểm thử (như việc thiếu nguồn lực) cần được đưa ra ở đây.

Như đã thấy trong mô hình chữ V (hình 5.1), các kế hoạch kiểm thử được xây dựng từ sớm, ngay trong mỗi pha của tiến trình phát triển. Khi thực hiện kiểm thử, nó được bổ sung và cụ thể hóa. Giống như các kế hoạch khác, kế hoạch kiểm thử không cố định mà cần được điều chỉnh theo thực tế phát triển phần mềm. Như vậy, *việc lập kế hoạch và thực hiện kiểm thử là một quá trình được lặp lại* (mỗi khi thay đổi yêu cầu, thay đổi thiết kế, sửa đổi mã nguồn,...).

5.2.2. Ca kiểm thử

Ca kiểm thử được thiết kế cho mỗi tình huống – kịch bản. Thiết kế một ca kiểm thử thường bao gồm:

- Tên của các môđun được kiểm thử (định danh).
- Mô tả dữ liệu vào.
- Kết quả ra mong muốn (đúng, sai).
- Kết quả ra thực tế (khi tiến hành kiểm thử).

Ví dụ, thiết kế một ca kiểm thử sử dụng ngôn ngữ XML:

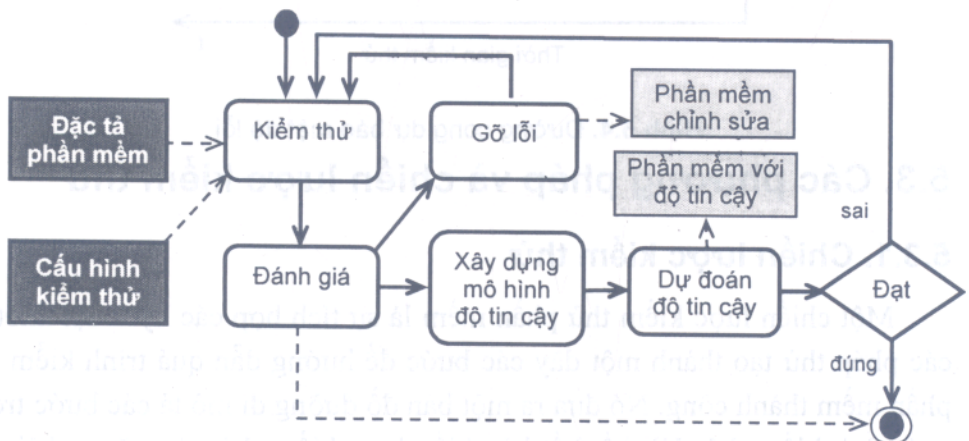
Định danh	Mô tả	Kết quả mong đợi	Kết quả thực tế
1	<pre><TRAIN> <id> Danny </id> <length> 95 </length> <acceleration> 0.7 </acceleration> <startStation> Cary </startStation> <side> L</side> </TRAIN></pre>	sai, quá ngắn	
2	...	...	

Có rất nhiều phương pháp khác nhau được sử dụng để thiết kế các ca kiểm thử phần mềm. Hai phương pháp được dùng nhiều là *phương pháp hộp trắng* và *phương pháp hộp đen*.

5.2.3. Tiến hành kiểm thử

Quá trình kiểm thử cần được tổ chức theo một chiến lược tổng thể với các mức cụ thể. Tiến trình này không thể đảm bảo phát hiện tất cả các khiếm khuyết và lỗi của phần mềm. Càng về sau, việc phát hiện và sửa lỗi còn lại càng trở nên khó khăn và đắt đỏ. Vì thế, người ta cần phải dừng lại ở một thời điểm thích hợp tùy theo cách *tiếp cận bi quan* hay *lạc quan*. Thời điểm dừng này liên quan đến khái niệm về "*tiêu chuẩn đầy đủ của kiểm thử*" [20]. Đối với kiểm thử ở mức mô đun, người ta thường dừng kiểm thử khi đã thực hiện hết các ca kiểm thử được thiết kế. Đối với kiểm thử thẩm định, người ta thường quyết định dừng kiểm thử khi gặp một trong những trường hợp sau:

- Hết thời gian (*tiếp cận bi quan*).
- Hết kinh phí (*tiếp cận bi quan*).
- Đạt đến điểm cân bằng về hiệu quả (*law of diminishing returns*): tại đó đã phát hiện và khắc phục một số lượng đủ lớn các lỗi, phần mềm đã đạt đến một độ tin cậy có thể chấp nhận được (*tiếp cận lạc quan*) và nếu tiếp tục thì chi phí cho việc phát hiện lỗi không thấp hơn chi phí cho việc sửa lỗi nếu nó xảy ra. Đây chính là một tiêu chí quan trọng khi xem xét việc tiếp tục hay dừng kiểm thử. Sơ đồ hình 5.3. cho ta hình dung về tiến trình kiểm thử phần mềm khi nào sẽ dừng.



Hình 5.3. Sơ đồ tiến trình kiểm thử và hoàn thiện chương trình

Khi kiểm thử, người ta thường sử dụng một mô hình thống kê tính độ thất bại của phần mềm theo thời gian thực hiện [29] được cho ở dưới đây:

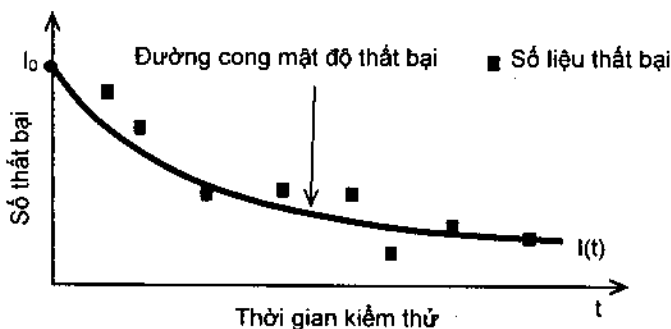
$$f(t) = (1/p) \ln(l_0 p t + 1)$$

Trong đó: $f(t)$ là số thất bại mong đợi xuất hiện sau khi kiểm thử được thực hiện t đơn vị thời gian, p là độ rút gọn của mật độ sai do chúng được phát hiện và sửa, l_0 là mật độ thất bại (= số thất bại/đơn vị thời gian) của phần mềm lúc bắt đầu kiểm thử.

Mật độ thất bại tức thời $l(t)$ được tính bằng cách lấy đạo hàm của $f(t)$ và bằng:

$$l(t) = l_0 / (l_0 p t + 1)$$

Người kiểm thử có thể dùng biểu thức này để dự đoán mật độ lỗi trong quá trình kiểm thử và lấy đó như một độ đo độ tin cậy của phần mềm để quyết định việc dừng kiểm thử hay không. Hình 5.4 cho ta biểu đồ dự đoán mật độ lỗi.



Hình 5.4. Đường cong dự báo mật độ lỗi

5.3. Các phương pháp và chiến lược kiểm thử

5.3.1. Chiến lược kiểm thử

Một chiến lược kiểm thử phần mềm là sự tích hợp các kỹ thuật thiết kế các phép thử tạo thành một dãy các bước để hướng dẫn quá trình kiểm thử phần mềm thành công. Nó đưa ra một bản đồ đường đi mô tả các bước trong quá trình kiểm thử. Vì thế, bất kỳ chiến lược kiểm thử nào cũng phải tích hợp được việc *lập kế hoạch kiểm thử, thiết kế các ca kiểm thử, tiến hành kiểm thử, thu thập và đánh giá các thông tin kết quả*.

Chiến lược kiểm thử phần mềm phải đủ mềm dẻo để cổ vũ các cách tiếp cận sáng tạo của người thực hiện. Nhưng với tư cách là một tiến trình trong dự án, nó cũng phải đủ chặt chẽ để hỗ trợ các kế hoạch và phương thức quản lý.

Một chiến lược kiểm thử phần mềm phải thích ứng với từng mức kiểm thử: các kiểm thử mức thấp (xác minh từng khúc mã nguồn xem có thực thi đúng đắn không), và các kiểm thử mức cao (thẩm định các chức năng hệ thống có đáp ứng yêu cầu khách hàng hay không).

Bên cạnh đó, một chiến lược kiểm thử phần mềm cần *cung cấp các hướng dẫn* cho người thực hành để tiến hành kiểm thử và một tập các *cột mốc* cho nhà quản lý theo dõi, giám sát. Ngoài ra, vì các bước trong chiến lược kiểm thử thực hiện vào các thời điểm khi mà áp lực thời gian tăng lên, do đó quá trình kiểm thử *phải đo được* để có thể nhận ra các vấn đề phát sinh càng sớm càng tốt.

Có nhiều chiến lược kiểm thử phần mềm khác nhau như: *kiểm thử từ trên xuống, kiểm thử từ dưới lên, kiểm thử nhiều bước,...* Các chiến lược kiểm thử khác nhau có thể tìm ra các loại lỗi khác nhau. Vì thế, không thể tập trung vào một chiến lược kiểm thử mà bỏ qua các loại khác. Các mức của quá trình kiểm thử cần được thực hiện tuần tự và bổ sung cho nhau nhằm tìm ra từng loại lỗi với các phương pháp thích hợp.

5.3.2. Kiểm thử hộp trắng

Kiểm thử hộp trắng là việc kiểm tra các đoạn mã chương trình xem nó có vận hành đúng như thiết kế hay không. Kiểm thử hộp trắng dựa trên việc *xem xét cấu trúc bên trong của chương trình* theo cấu trúc điều khiển và sự hoạt động của chúng. Vì thế, nó còn có các tên gọi khác (như: *glass testing, structural testing, open box testing* hay *clear box testing* [3]). Để kiểm thử tốt, người kiểm thử cần có hiểu biết về lập trình, về mã lệnh của ngôn ngữ lập trình, về logic bên trong của chúng và sự vận hành của chương trình.

Đối tượng của kiểm thử hộp trắng là các môđun mã lệnh. Để đảm bảo tính đầy đủ thì kiểm thử hộp trắng cần phải làm sao để: *Mọi lệnh phải được thực hiện, mọi điều kiện phải được kiểm tra, mọi chu trình được duyệt qua, mọi cấu trúc dữ liệu được sử dụng, mọi tiến trình được lần vết.* Từ đó đặt ra yêu cầu cho các ca kiểm thử hộp trắng ít nhất phải đảm bảo:

- Mọi con đường độc lập trong một môđun cần được thực hiện ít nhất một lần.
- Mọi ràng buộc logic được thực hiện cả hai phía đúng (true) và sai (false).
- Tất cả các vòng lặp ở biên của nó và cả các biên vận hành phải được thực hiện.
- Mọi cấu trúc dữ liệu nội tại được sử dụng để bảo đảm tính hiệu lực của chúng.

Tương ứng với các yêu cầu đặt ra ở trên, một số kỹ thuật sau đây đã được sử dụng khi xây dựng các ca kiểm thử theo phương pháp hộp trắng:

- *Đồ thị luồng* (flow graph).
- *Ma trận kiểm thử* (testing matrices).
- *Kiểm thử điều kiện* (condition testing).
- *Kiểm thử vòng lặp* (loop testing).
- *Kiểm thử luồng dữ liệu* (data flow testing).

Kiểm thử hộp trắng được sử dụng ở ba loại kiểm thử là: *kiểm thử đơn vị*, *kiểm thử tích hợp* và *kiểm thử hồi quy*.

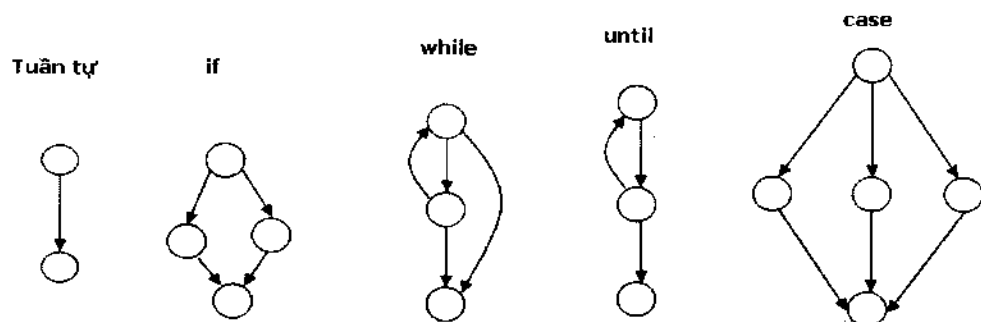
5.3.2.1. Đồ thị luồng

a) Cấu trúc đồ thị luồng

Đồ thị luồng (*flow graph*) là một kỹ thuật kiểm thử hộp trắng được Tom McCabe [19] đề xuất. Đồ thị luồng được xây dựng từ đồ thị luồng điều khiển của chương trình bằng cách gộp các lệnh tuần tự liên tiếp và lệnh rẽ nhánh sau nó thành một nút, thay các lệnh rẽ nhánh (độc lập) hay điểm hợp nhất của các đường rẽ nhánh bằng một nút. Như vậy, cấu trúc của một đồ thị luồng bao gồm:

- Mỗi *nút* (hình tròn) biểu thị một vài (có thể là 0) câu lệnh thủ tục.
- Mỗi *cạnh* biểu diễn dòng điều khiển nối hai nút với nhau.
- *Miền* là phần mặt phẳng giới hạn bởi các cung và nút của đồ thị.
- Những nút biểu thị sự phân nhánh hoặc hội nhập của các cung được gọi là *nút vị từ*.

Các cấu trúc lệnh cơ bản của chương trình tương ứng với các cấu trúc trong đồ thị luồng cho trong hình 5.5.



Hình 5.5. Các cấu trúc cơ bản trong đồ thị luồng

b) Độ phức tạp chu trình

Độ phức tạp chu trình (*cyclomatic complexity*) là số đo sự phức tạp logic của chương trình. Nó là số các đường đi độc lập cơ bản trong tập các con đường độc lập của một chương trình – các đường đi suốt chương trình và khác nhau ít nhất một cung. Đây là số đường độc lập nhỏ nhất phủ hết các cạnh của đồ thị luồng. Số đo này là giới hạn trên của số ca kiểm thử cần phải tiến hành để bảo đảm rằng, tất cả các câu lệnh trong chương trình đều được thực hiện ít nhất một lần.

Độ phức tạp chu trình của đồ thị luồng G được ký hiệu là $V(G)$ và có thể tính theo ba cách sau:

1. $V(G) = \text{số miền phẳng}$
2. $V(G) = E - N + 2$
3. $V(G) = P + 1$

Trong đó: E là số cạnh, N là số nút của đồ thị và P là số nút vị từ.

Ví dụ, xét một đoạn chương trình sau:

```

1  do while records remain
2      read record;
3      if record field 1 = 0
4          then process record;
5          store in buffer;
6          increment counter;

```

```

7   elsif record field 2 = 0
8       then reset record;
9       else process record;
10          store in file;
11      endif;
12  endif;
13 enddo;
14 end;

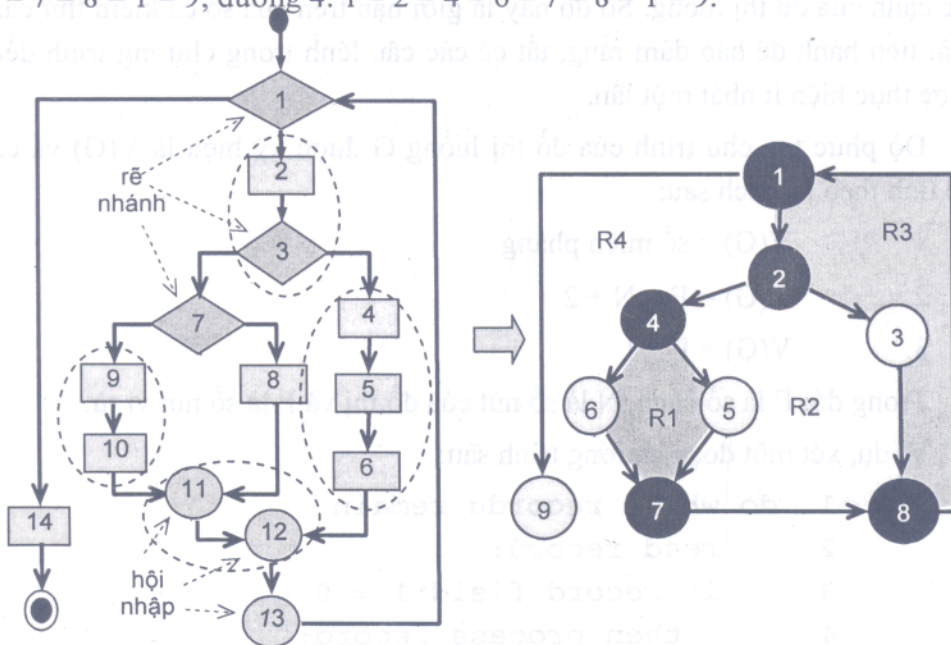
```

Ta có đồ thị luồng điều khiển của chương trình và đồ thị luồng tương ứng của nó cho ở hình 5.6. Độ phức tạp chu trình $V(G)$ của đồ thị này bằng 4, là số miền phẳng của nó, hay có thể tính toán theo các công thức khác dưới đây:

$$V(G) = E - N + 2 = 11 - 9 + 2 = 4$$

$$V(G) = P - 1 = 5 - 1 = 4$$

4 đường độc lập cơ bản tìm được (để lập các ca kiểm thử) từ đồ thị luồng là: đường 1: 1 - 9; đường 2: 1 - 2 - 3 - 8 - 1 - 9; đường 3: 1 - 2 - 4 - 5 - 7 - 8 - 1 - 9; đường 4: 1 - 2 - 4 - 6 - 7 - 8 - 1 - 9.



a) Đồ thị luồng điều khiển

b) Đồ thị luồng của chương trình

Hình 5.6. Đồ thị luồng điều khiển và đồ thị luồng của chương trình

Để thiết lập các ca kiểm thử khi sử dụng kỹ thuật đồ thị luồng, cần tiến hành theo các bước sau:

1. Từ thiết kế hoặc mã nguồn của môđun, ta lập đồ thị luồng điều khiển.
2. Xây dựng đồ thị luồng tương ứng với đồ thị luồng điều khiển.
3. Xác định độ phức tạp chu trình $V(G)$ của đồ thị luồng đó.
4. Xác định tập cơ bản của các con đường độc lập.
5. Chuẩn bị các ca kiểm thử tương ứng mỗi con đường cơ bản.

5.3.2.2. Ma trận kiểm thử

Ma trận kiểm thử được xây dựng dựa trên đồ thị luồng. Nó là một ma trận vuông, có kích thước bằng số các nút của đồ thị luồng và là cách biểu diễn bằng ma trận của đồ thị luồng:

- Mỗi dòng/cột ứng với tên một nút.
- Mỗi ô được ghi số 1 nếu có một cung nối nút dòng đến nút cột.

Ma trận kiểm thử A được dùng để tính độ phức tạp chu trình: cộng số 1 trên mỗi dòng rồi trừ đi 1, sau đó lấy tổng các dòng cộng thêm 1 ta được kết quả. Khi nhân liên tiếp k ma trận A với nhau ta được một ma trận A^k mà con số ghi trong mỗi ô cho số các con đường có k cung đi từ nút dòng tới nút cột. Các ma trận kiểm thử được sử dụng như một dữ liệu có cấu trúc để kiểm tra các đường cơ bản.

Để các ma trận kiểm thử trở thành một công cụ mạnh trong việc đánh giá cấu trúc điều khiển chương trình khi kiểm thử, người ta xác định thêm các trọng số sau đây cho các cung của ma trận kiểm thử:

- Xác suất cung đó được tiến hành.
- Thời gian xử lý của đường đi qua cung đó.
- Bộ nhớ yêu cầu của đường đi qua cung đó.
- Nguồn lực đòi hỏi của đường đi qua cung đó.

Với những tham số này, các ma trận kiểm thử giúp ta lựa chọn những đường đi cơ bản có trọng số quan trọng để thiết lập các ca kiểm thử. Hình 5.7 cho hai ma trận đồ thị kiểm thử của đồ thị luồng cho ở hình 5.6.

A =

Đỉnh	1	2	3	4	5	6	7	8	9	Tính V(G)
1		1							1	$2 - 1 = 1$
2			1	1						$2 - 1 = 1$
3								1		$1 - 1 = 0$
4					1	1				$2 - 1 = 1$
5							1			$1 - 1 = 0$
6							1			$1 - 1 = 0$
7								1		$1 - 1 = 0$
8	1									$1 - 1 = 0$
9										$3 + 1 = 4$

$A^2 =$

Đỉnh	1	2	3	4	5	6	7	8	9
1			1	1					
2					1	1		1	
3	1								
4							2		
5								1	
6								1	
7									
8			1						1
9									

Hình 5.7. Ma trận A và A^2 của đồ thị luồng

5.3.2.3. Kiểm thử điều kiện

Để kiểm thử điều kiện (*condition testing*) trong chương trình, người ta đã đề xuất một số chiến lược kiểm thử:

- *Kiểm thử rẽ nhánh (Branch testing)* là chiến lược đơn giản nhất. Đối với mỗi điều kiện rẽ nhánh phức hợp C, các nhánh true và false của C và mỗi điều kiện đơn giản trong C cần được thực thi ít nhất một lần.

- *Kiểm thử miền (Domain testing)* yêu cầu ba hoặc bốn kiểm thử được thực hiện cho một biểu thức quan hệ. Đối với mỗi biểu thức quan hệ có dạng:

$$E_1 <\text{phép toán quan hệ}> E_2$$

thì cần có ba kiểm thử làm cho giá trị của E_1 lớn hơn ($>$), bằng ($=$), hoặc nhỏ hơn ($<$) E_2 . Nếu $<\text{phép toán quan hệ}>$ là không chính xác, E_1 và E_2 là chính xác, thì ba phép thử này đảm bảo việc phát hiện ra lỗi phép toán quan hệ. Để phát hiện ra lỗi trong E_1 và E_2 , kiểm thử làm cho giá trị của E_1 lớn hơn hoặc nhỏ hơn giá trị của E_2 , nên tạo ra sự khác biệt giữa hai giá trị này nhỏ đến mức có thể.

a) Kiểm thử rẽ nhánh

Không một phần mềm nào có thể viết theo chương trình chỉ gồm các mã lệnh tuần tự liên tục. Tại một vài chỗ, chúng cần phải rẽ nhánh để thực hiện một chức năng nào đó. Kiểm thử rẽ nhánh là chiến lược tập trung vào kiểm thử từng điều kiện rẽ nhánh có trong chương trình. Việc kiểm tra bao gồm, vận hành chương trình với các dữ liệu sao cho đảm bảo kiểm tra mỗi nhánh ít nhất một lần. Kiểm thử rẽ nhánh giúp xác nhận tính đúng đắn tất cả các rẽ nhánh trong mã lệnh và khẳng định không có nhánh nào dẫn tới một hành vi bất thường của ứng dụng.

Điều kiện kiểm thử gồm các dạng cơ bản sau:

- *Điều kiện đơn*: là một biến kiểu bool hoặc một biểu thức quan hệ (có thể có toán tử phủ định ở đầu).
- *Biểu thức quan hệ*: là biểu thức kiểu bool xác định quan hệ giữa hai biểu thức số học bằng một phép so sánh như: $<$, $\leq$ ($\leq$), $=$, $\geq$ ($\geq$), $>$ hay $\neq$.
- *Điều kiện kết hợp*: là điều kiện hợp thành từ nhiều điều kiện đơn nhờ các toán tử bool: *hoặc* (OR), *và* (AND), *phủ định* (NOT).

Nếu một điều kiện bị sai, thì ít nhất một thành phần của điều kiện là không chính xác. Do đó, các loại lỗi của một điều kiện có thể là:

- Sai biến boolean (ví dụ: sử dụng sai biến).
- Sai toán tử boolean (ví dụ: AND thay cho OR).

- Sai số hạng trong biểu thức toán tử boolean.
- Sai toán tử quan hệ (ví dụ: $\geq$ thay cho $>$).
- Sai biểu thức số học (ví dụ: $j + k$ thay cho $j - k$).

Việc kiểm thử tiến hành một cách đơn giản theo nguyên tắc sau:

- Kiểm thử cho từng điều kiện trong chương trình.
- Kiểm thử nhánh: với mỗi điều kiện kết hợp C, thì các nhánh "true" và "false" của C và mỗi một điều kiện đơn trong C phải được kiểm thử ít nhất một lần.

b) Chiến lược kiểm thử BRO

Khi một biểu thức điều kiện phức hợp gồm n điều kiện đơn mà số n đủ lớn, thì việc tiến hành đầy đủ 2^n ca kiểm thử là rất khó thực hiện. Chiến lược BRO (*branch and relation operator*) là một kỹ thuật làm giảm số ca kiểm thử cần tiến hành trong kiểm thử điều kiện. Kỹ thuật này đảm bảo việc phát hiện ra các lỗi quan hệ và nhánh trong điều kiện mà tất cả các biến Bool và toán tử quan hệ chỉ xuất hiện một lần và không có biến chung.

BRO dùng đến "ràng buộc điều kiện cho điều kiện cần thử". Giả sử trong điều kiện C cần thử có n điều kiện đơn, ràng buộc đầu ra của C có n giá trị ($D_1, D_2, \dots, D_n$), trong đó D_i là một đặc tả ràng buộc đầu ra của điều kiện đơn tương ứng của C. Ta nói rằng, ràng buộc D của điều kiện C là được phủ bởi một thi hành của C nếu như trong quá trình thi hành đó đầu ra của mỗi điều kiện đơn D_i trong C thoả mãn các ràng buộc tương ứng.

Tư tưởng của chiến lược BRO là cho phép thử nhảy cảm sai của biểu thức bool. Với một biểu thức bool duy nhất (một biểu thức bool mà mỗi biến bool chỉ xuất hiện 1 lần) với n biến bool ($n > 0$), ta có thể dễ dàng sinh ra một tập kiểm thử ít hơn rất nhiều so với toàn bộ 2^n phép thử mà vẫn đảm bảo phát hiện ra nhiều lỗi của các toán tử bool và cũng là hiệu quả để phát hiện các lỗi khác.

1. Xét điều kiện C là một biến bool, khi đó ràng buộc đầu ra của C là một cặp giá trị t hoặc f.

Ví dụ, xét điều kiện $C = (A \text{ and } B)$ với A và B là 2 biến bool. Chiến lược kiểm thử BRO đòi hỏi rằng, tập ba ràng buộc đầu ra $\{(t, t), (t, f),$

(f, t) đều được phủ bởi các thi hành của C, còn (f, f) là thừa. Nếu C không đúng dẫn do "ít nhất một phép toán bool sai", thì trong 3 cặp trên có ít nhất một cặp làm C thất bại.

2. Xét điều kiện đơn $C = (A = B)$ với A và B là biểu thức số học. Khi đó ràng buộc của C là một trong ba quan hệ: $<, >, =$.
3. Xét điều kiện C là hội của hai biểu thức bool: A và $(B = E)$. Khi đó các ràng buộc của C là các cặp (t, t) , (t, f) và (f, t) ; với $(B = E)$, ta có giá trị t tương ứng với " $=$ ", và giá trị f tương ứng với " $<$ " hoặc " $>$ ". Tương tự như trường hợp ở trên đã xét đối với điều kiện hai biến bool có ràng buộc đầu ra là $\{(t, t), (t, f), (f, t)\}$, ta có tập các ràng buộc của C trong trường hợp này phải gồm bốn phần tử: $(t, =)$, $(t, <)$, $(t, >)$ và $(f, =)$.

Chiến lược kiểm thử điều kiện nhìn chung có hai thuận lợi. Thứ nhất, độ đo mức độ bao phủ của một điều kiện là đơn giản. Thứ hai, bao phủ kiểm thử của điều kiện trong một chương trình cho ta các hướng dẫn để tạo ra các kiểm thử đầy đủ cho chương trình.

5.3.2.4. Kiểm thử luồng dữ liệu

Phương pháp kiểm thử luồng dữ liệu (*data flow testing*) là kiểm thử tuyển chọn các đoạn đường của chương trình tương ứng với việc định vị các biến được xác định và sử dụng trong chương trình. Khi một biến xuất hiện và sử dụng trong chương trình, liên quan đến nó có các khái niệm sau:

- Vị trí xác định biến.
- Vị trí sử dụng biến.
- Vị trí sử dụng biến để tính toán.
- Vị trí sử dụng biến như một vị từ.
- Cặp xác định và sử dụng biến (DEF – USE).

Có hai sự kiện đối với một biến trong một chương trình là khi xác định và khi sử dụng. Vị trí một biến được xác định là nơi mà nó được gán một giá trị cụ thể. Vị trí sử dụng biến là vị trí mà giá trị của biến được sử dụng trong chương trình. Có hai kiểu sử dụng biến là sử dụng để tính toán và sử dụng như một vị từ để xác định đường đi thực thi tiếp tục chương trình. Một

cặp DEF – USE là một cặp vị trí mà tại đó biến được xác định và được sử dụng, mà có thể liên kết với nhau bằng một đường đi thỏa mãn điều kiện là biến được xét không được định nghĩa lại. Mục đích của phương pháp này nhằm kiểm tra tất cả các cặp DEF – USE xuất hiện trong chương trình. Kiểm thử DEF – USE không đảm bảo phủ tất cả các nhánh của chương trình; tuy nhiên, một nhánh không được kiểm thử theo kiểu này là rất hiếm.

Giả sử rằng, mỗi câu lệnh của chương trình được gán với số câu lệnh duy nhất và mỗi hàm không được thay đổi các tham số của nó và các biến toàn cục.

Với mỗi câu lệnh S, ta định nghĩa:

- $DEF(S) = \{X \mid \text{câu lệnh } S \text{ chứa định nghĩa của } X\};$
- $USE(S) = \{X \mid \text{câu lệnh } S \text{ chứa một sử dụng } X\};$
- Nếu S là câu lệnh if hoặc câu lệnh vòng lặp thì DEF(S) của nó là rỗng, còn USE(S) của nó được dựa theo điều kiện trong S.

Ta nói rằng, biến X được xác định ở câu lệnh S là còn sống đến câu lệnh S' nếu có một đường đi từ S tới S', mà trên đó không chứa một định nghĩa nào khác cho X.

Một dây chuyền sử dụng X được ký hiệu là $DU = [X, S, S']$, với X trong DEF(S) và trong USE(S'), đồng thời định nghĩa X trong S vẫn còn đúng trong S'.

Phương pháp kiểm thử luồng dữ liệu yêu cầu rằng, tất cả các dây chuyền DU đều được kiểm thử ít nhất một lần. Trong một số ít trường hợp, ví dụ như cấu trúc lệnh **if – then**, trong phần **then** không có định nghĩa thêm một biến nào và phần **else** không tồn tại. Trong tình huống này thì nhánh **else** của câu lệnh **if** trên không cần thiết phải thực hiện phương pháp này.

Ví dụ:

```

Proc x
  B1;
  Do while C1
    If C2
      Then B1
        If C4
          Then B4;

```

```

Else B5;
Endif;
Else
  If C3
    Then B2;
    Else B3;
  Endif
Endif
Enddo;
B6;
End proc;

```

Giả sử biến X được xác định trong câu lệnh cuối của các khối B_1, B_2, B_3, B_4, B_5 và được dùng trong câu lệnh đầu tiên của các khối B_2, B_3, B_4, B_5 và B_6 . Chiến lược kiểm thử DU đòi hỏi việc thực hiện của đường đi ngắn nhất từ mỗi B_i ($0 < i \leq 5$) tới mỗi B_j ($1 < j \leq 6$) (việc kiểm thử như vậy cũng bao phủ mọi việc dùng biến X trong các điều kiện C_1, C_2, C_3, C_4). Mặc dù có tới 25 dây chuyền DU của biến X , nhưng chỉ cần năm đường đi để bao phủ các dây chuyền DU của X từ B_i ($0 < i \leq 5$) tới B_6 và các dây chuyền DU khác có thể được bao phủ bằng cách làm cho năm đường đi có chứa vòng lặp của chu trình.

5.3.2.5. Kiểm thử vòng lặp

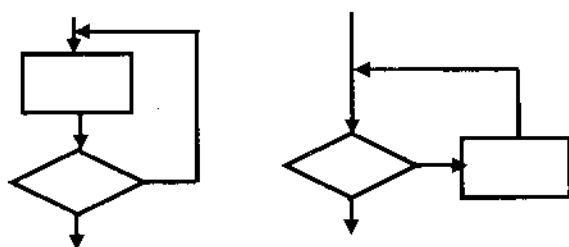
Kiểm thử vòng lặp (*loop testing*) là một kỹ thuật kiểm thử hộp trắng tập trung hoàn toàn vào tính hợp lệ của kết cấu vòng lặp. Có thể xác định bốn lớp vòng lặp khác nhau (hình 5.8): *vòng lặp đơn* (simple loop), *vòng lặp lồng* (nested loop), *vòng lặp nối tiếp* (concatenated loop), *vòng lặp phi cấu trúc* (unstructured loop).

a) Vòng lặp đơn

Tập các kiểm thử được áp dụng cho vòng lặp đơn với n là số tối đa các bước được phép qua vòng lặp:

- Bỏ qua toàn bộ vòng lặp;
- Chỉ qua 1 vòng lặp;
- Qua 2 vòng lặp;
- Qua vòng lặp m với $m < n$;
- Qua vòng lặp $n - 1, n, n + 1$.

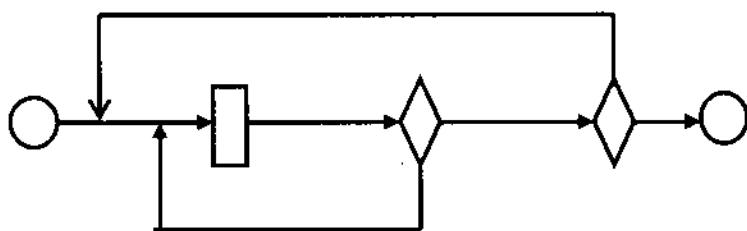
Tóm lại, ta sẽ kiểm tra các trường hợp với các vòng lặp là 0, 1, một giá trị thông thường m nào đó, $n - 1$, n và $n + 1$.



Hình 5.8a. Vòng lặp đơn

b) Vòng lặp lồng

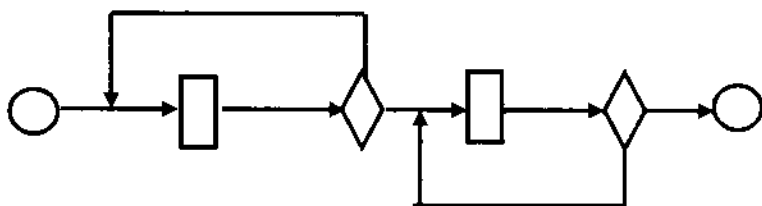
Số phép kiểm thử có thể sẽ tăng theo cấp số nhân khi mức độ lồng nhau tăng lên, phương thức kiểm tra vòng lặp lồng không thể đơn giản là mở rộng cách thức của kiểm tra vòng lặp đơn, bởi vì nó cần tới rất nhiều ca kiểm thử. Có thể dùng phương pháp sau đây:



Hình 5.8b. Vòng lặp lồng

- Bắt đầu từ vòng lặp bên trong nhất. Đặt tất cả các vòng lặp khác vào giá trị tối thiểu.
- Tiến hành kiểm thử vòng lặp đơn cho vòng lặp bên trong nhất, trong khi giữ các vòng lặp bên ngoài ở giá trị tham số lặp tối thiểu. Bổ sung các kiểm thử khác về việc vượt cận hay giá trị bị loại ra.
- Tiến dần ra ngoài, thực hiện phép kiểm thử cho vòng lặp tiếp, nhưng giữ tất cả các vòng lặp bên ngoài hơn ở giá trị tối thiểu và các vòng lặp lồng bên trong khác ở giá trị "điển hình".
- Tiếp tục cho tới khi mọi vòng lặp đã được kiểm thử hết.

c) Vòng lặp nối tiếp

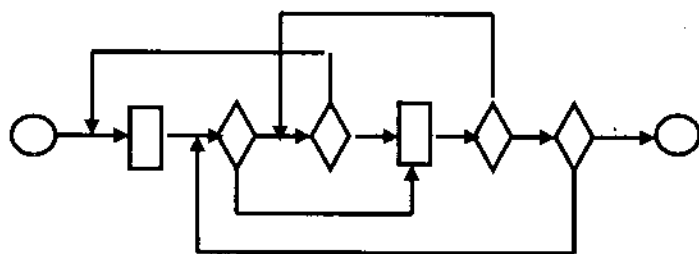


Hình 5.8c. Vòng lặp nối tiếp

Dùng cho các vòng lặp đơn nếu mỗi vòng lặp là độc lập. Tuy nhiên, nếu 2 vòng lặp nối nhau và số đếm của vòng lặp 1 được dùng làm giá trị khởi đầu cho vòng lặp 2 thì các vòng lặp là không độc lập nhau. Khi đó cần áp dụng quy tắc như các vòng lặp lồng nhau.

d) Vòng lặp phi cấu trúc

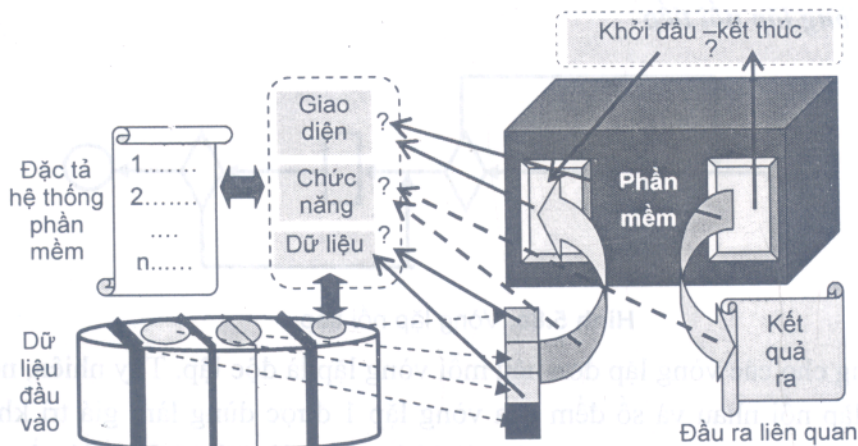
Bất kỳ khi nào gặp phải loại vòng lặp này, thì nên thiết kế lại để phản ánh việc sử dụng các kết cấu lập trình có cấu trúc.



Hình 5.8d. Vòng lặp phi cấu trúc

5.3.3. Kiểm thử hộp đen

Kiểm thử hộp đen (blackbox testing) xem phần mềm như một "hộp đen" thuần túy, người kiểm thử không cần quan tâm đến cấu trúc và hoạt động bên trong của nó. Bằng cách nhập các dữ liệu đầu vào qua giao diện, thực thi chương trình và xem xét kết quả đầu ra qua giao diện để nhận biết việc thực hiện chức năng có phù hợp với sự mong đợi hay không. Kiểm thử hộp đen vì thế còn được gọi là *kiểm thử chức năng (functional testing)* hay *kiểm thử hành vi (behavioral testing)*. Như vậy, cơ sở cho việc kiểm thử hộp đen chính là đặc tả chức năng của phần mềm và các dữ liệu mà nó sử dụng (hình 5.9).



Hình 5.9. Mô hình mô tả kiểm thử hộp đen

Kiểm thử hộp đen nhằm tìm ra các loại sai sau:

- Thiếu chức năng hoặc chức năng không đúng đắn.
- Sai về giao diện.
- Sai trong cấu trúc hoặc trong truy cập dữ liệu ngoài.
- Sai trong thực thi chức năng.
- Sai khởi đầu hoặc kết thúc.

Kiểm thử hộp đen cần tập trung để trả lời các câu hỏi sau đây: Hiệu lực chức năng được kiểm thử đến đâu? Lớp đầu vào nào cho các ca kiểm thử tốt? Chức năng nhạy cảm đối với những giá trị vào nào? Các biên của lớp dữ liệu đã được cô lập như thế nào? Khả năng dung thứ lỗi đối với các nhịp điệu và khối lượng dữ liệu như thế nào? Những tổ hợp dữ liệu đặc biệt ảnh hưởng gì đến hoạt động hệ thống?

Khi kiểm thử hộp đen, căn cứ trên đặc tả của phần mềm, người kiểm thử xây dựng các nhóm giá trị đầu vào cho mỗi yêu cầu chức năng của chương trình để thử. Để tìm được tất cả các lỗi của chương trình thì bắt buộc phải kiểm thử tất cả các giá trị đầu vào. Nhưng điều này là không thể thực hiện được khi dữ liệu đủ lớn. Vì thế, để đạt được mục tiêu kiểm thử, người ta đã áp dụng ba kỹ thuật cho kiểm thử hộp đen là *phân hoạch tương đương*, *phân tích giá trị biên* và *đồ thị nhân quả*.

5.3.3.1. Phân hoạch tương đương

Phân hoạch tương đương (*equivalence partitioning*) là một chiến lược được sử dụng để làm giảm số ca kiểm thử cần thiết và đã được Beizer [3] trình bày. Theo đó chia đầu vào của chương trình thành nhiều vùng khác nhau, gọi là các lớp tương đương. Với mỗi lớp tương đương này, chương trình đối xử với các phân tử của nó là tương tự nhau. Như vậy, đầu vào cần được phân hoạch thuộc vào một trong hai loại lớp: các *lớp tương đương hợp lệ* gồm các dữ liệu nằm trong miền hợp lệ và các *lớp tương đương không hợp lệ* gồm các dữ liệu nằm trong miền không hợp lệ (miền sai). Do đó, đối với mỗi lớp tương đương chỉ cần xây dựng một ca kiểm thử đại diện, và nó tạo ra cùng một trạng thái của chương trình, đó là trạng thái đúng hay sai. Kết quả kiểm thử cho phép phát hiện về sự tồn tại hay không những lớp sai, không phải là sai đối với những giá trị cụ thể nào được kiểm thử.

Ví dụ, trong chương trình điều khiển các đoàn tàu vào ga, người ta xét đến yếu tố chiều dài của tàu mà giới hạn từ 100 – 300m. Do đó, theo đặc trưng của việc xử lý, ta có thể chia loại tàu có độ dài khác nhau thành ba lớp tương đương với một lớp hợp lệ và hai lớp không hợp lệ cho trong bảng sau:

Bảng phân hoạch dữ liệu thành các lớp hợp lệ và không hợp lệ

Lớp không hợp lệ	Lớp hợp lệ	Lớp không hợp lệ
Nhỏ hơn 100	Giữa 100 và 300	Lớn hơn 300

Với cấu trúc dữ liệu đầu vào đã cho, việc xác định các lớp dữ liệu tương đương cho các ca kiểm thử có thể theo quy tắc sau:

1. Nếu trạng thái đầu vào gồm một dải giá trị, thì tạo 1 lớp hợp lệ và 2 lớp không hợp lệ (như ví dụ trên).
2. Nếu trạng thái đầu vào là một số giá trị nào đó (ví dụ R và L cho trường chỉ vị trí), thì tạo 1 lớp tương đương gồm các giá trị hợp lệ (R và L) và 1 lớp gồm các giá trị không hợp lệ (tất cả các chữ khác R và L). Trong trường hợp này cần kiểm thử tất cả các giá trị hợp lệ và một số giá trị không hợp lệ.
3. Nếu trạng thái đầu vào là các phân tử của một tập hợp, tạo một lớp tương đương hợp lệ và một lớp không hợp lệ.

4. Nếu trạng thái đầu vào là giá trị Boolean, xác định một lớp tương đương hợp lệ và một lớp không hợp lệ.

Đối với mỗi lớp này, cho ra một số giá trị đại diện để lập các ca kiểm thử. Việc chọn giá trị đại diện được thực hiện theo nguyên tắc phân tích giá trị biên.

5.3.3.2. Phân tích giá trị biên

Các lập trình viên thường mắc lỗi tại các biên của các lớp tương đương hay một khoảng đầu vào. Vì vậy, cần tập trung việc kiểm thử tại các biên này. Kiểu kiểm thử này được gọi là *phân tích giá trị biên* (*boundary value analysis – BVA*) và hướng dẫn tạo các ca kiểm thử tại biên của các lớp tương đương. Trong ví dụ ở trên, giá các biên của các lớp là 100 và 300. Chúng ta nên tạo các ca kiểm thử với các tàu có chiều dài là 99, 100, 101, 299, 300 và 301.

Khi tạo các ca kiểm thử theo BVA, hãy cân nhắc những điều sau:

1. Nếu trạng thái đầu vào gồm một dải từ **a** đến **b** (ví dụ $a = 100$ tới $b = 300$), tạo các ca kiểm thử:
 - Ngay dưới **a** (99);
 - Bằng **a** (100);
 - Ngay trên **a** (101);
 - Ngay dưới **b** (299);
 - Bằng **b** (300);
 - Ngay trên **b** (301).
2. Nếu trạng thái đầu vào là một số các giá trị được phép, kiểm thử các giá trị này. Ví dụ, chỉ một tàu được phép bắt đầu tại mỗi hướng trên mỗi ga. Thử thêm tàu thứ hai vào cùng ga/cùng hướng. Nếu có 3 tàu có thể bắt đầu trên cùng một ga/hướng, thử thêm 2 tàu (pass), 3 tàu (pass), rồi 4 tàu (fail).

5.3.3.3. Đồ thị nhân quả

Kỹ thuật *lập đồ thị nhân quả* (*Cause – Effect Graphing*) là một kỹ thuật thiết kế ca kiểm thử hộp đen [3]. Nó cung cấp một biểu diễn chính xác giữa

các điều kiện logic và các hành động tương ứng. Phương pháp này có hiệu quả khi việc phân hoạch thành lớp tương đương gặp khó khăn, mà ở đó một cái ra có thể do một số đầu vào xác định.

Nội dung chuẩn bị cho các ca kiểm thử với đồ thị nhân quả là:

1. Các nguyên nhân (cái vào) và kết quả (cái ra) được lấy ra từ đặc tả.
2. Mỗi quan hệ giữa nguyên nhân và kết quả được biểu diễn bằng đồ thị làm cho mỗi quan hệ logic giữa chúng được rõ ràng.
3. Bảng quyết định được chuẩn bị dựa trên các đồ thị. Từ đó, tạo ra dữ liệu cho các ca kiểm thử.

Các bước sử dụng kỹ thuật đồ thị nhân quả bao gồm:

1. Lập danh sách các nguyên nhân (điều kiện vào) và kết quả (hành động) cho từng mô đun và gán giá trị định danh cho chúng.
2. Phát triển đồ thị nhân quả.
3. Chuyển đồ thị đó thành bảng quyết định.
4. Các quy luật của bảng quyết định được sử dụng xây dựng các ca kiểm thử.

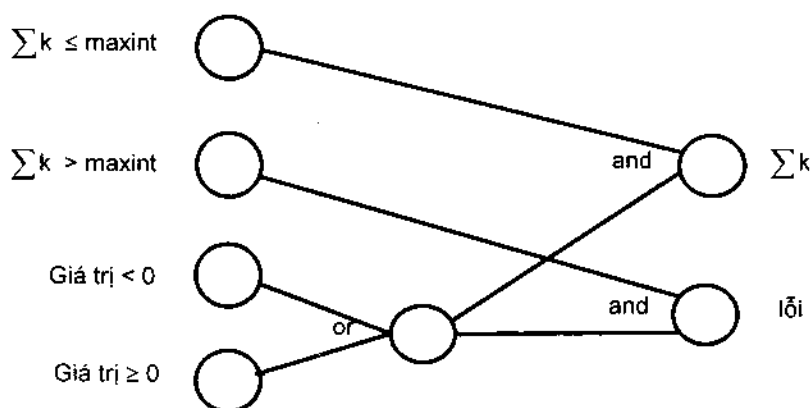
Ví dụ: Sau khi nhập vào số nguyên maxint và một giá trị, cần tính toán giá trị kết quả theo điều kiện sau:

$$\text{Result} = \sum_{k=0}^{\text{Value}} k \text{ nếu } \sum_{k=0}^{\text{Value}} k \leq \text{maxint} \text{ và báo lỗi nếu } \sum_{k=0}^{\text{Value}} k > \text{maxint}.$$

a) Lập bảng các điều kiện kiểm thử

	Maxint	Giá trị nhập	Kết quả
đúng	100	10	55
	100	-10	"55"
sai	10	10	lỗi
	10	-	lỗi
	10	30	lỗi
	"XYZ"	10	lỗi
	100	9.1E4	lỗi

b) Phát triển đồ thị nhân quả



c) Xây dựng bảng quyết định

Cái vào – nguyên nhân	$\Sigma k \leq \text{maxint}$	1	1	0	0
	$\Sigma k > \text{maxint}$	0	0	1	1
	giá trị < 0	1	0	1	0
	giá trị ≥ 0	0	1	0	1
Cái ra – hiệu ứng	Σk	1	1	0	0
	lỗi	0	0	1	1

d) Lập các ca kiểm thử

Căn cứ vào bảng quyết định trên, ta sẽ tiến hành lập các ca kiểm thử. Với bài toán này cần tiến hành bốn ca kiểm thử tương ứng với bốn cột có các giá trị của cái vào – nguyên nhân và cái ra – kết quả tương ứng.

5.3.3.4. Những hạn chế của kiểm thử hộp đen

Mặc dù kiểm thử hộp đen có nhiều ưu điểm, nhưng sử dụng riêng bản thân nó là chưa đủ. Trước tiên, khi hệ thống có nhiều loại dữ liệu đầu vào khác nhau, điều này có thể dẫn tới sự bùng nổ các ca kiểm thử. Thứ hai, do không thể biết được mô đun nào của chương trình đã hay chưa được kiểm thử, khi đó phải kiểm thử lại hay bỏ qua những lỗi tiềm ẩn trong gói phần

mềm. Tiếp nữa, nếu không có bản đặc tả rõ ràng và súc tích, rất khó thiết kế các ca kiểm thử đúng đắn.

5.4. Các loại hình kiểm thử

Chiến lược kiểm thử tổng thể một hệ thống được sử dụng rộng rãi nhất hiện nay là *chiến lược từ mức thấp đến mức cao*, bao gồm bốn mức như được thể hiện trong mô hình chữ "V" (hình 5.1):

- Kiểm thử đơn vị (*unit testing*);
- Kiểm thử tích hợp (*integration testing*);
- Kiểm thử hệ thống (*systems testing*);
- Kiểm thử chấp nhận (*acceptance testing*).

Khi kiểm thử, nếu các lỗi được phát hiện tại bất kỳ mức nào, chúng đòi hỏi chương trình phải gỡ lỗi và hiệu chỉnh cho chính xác, và sau đó cần kiểm thử lại, tức là cần được lặp lại những kiểm thử nào đó ở mức trước. Trong quy trình này, thông tin lỗi gặp phải hay mức độ tin cậy đạt được của chương trình ở mỗi giai đoạn sau đều được phản hồi về giai đoạn trước của tiến trình (xem hình 5.2). Như vậy, tiến trình kiểm thử phải là tiến trình lặp.

Bảng tổng hợp các mức kiểm thử cho ở bảng 5.1

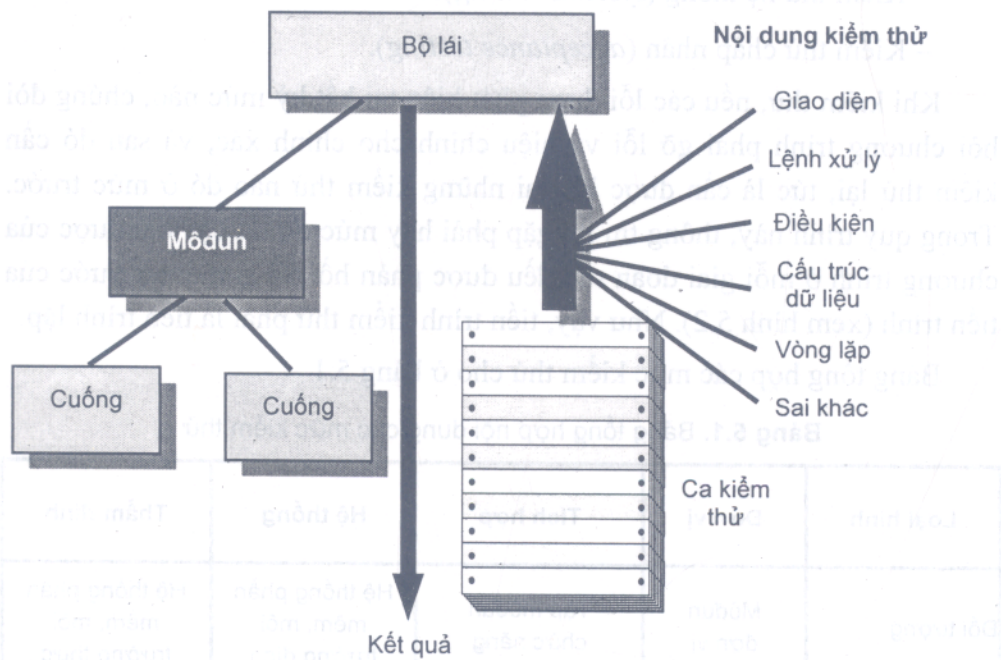
Bảng 5.1. Bảng tổng hợp nội dung các mức kiểm thử

Loại hình	Đơn vị	Tích hợp	Hệ thống	Thẩm định
Đối tượng	Mô đun đơn vị	Tập mô đun chức năng	Hệ thống phần mềm, môi trường đích	Hệ thống phần mềm, môi trường thực
Cơ sở đối sánh	Thiết kế chi tiết	Thiết kế chi tiết, kiến trúc, đặc tả chức năng	Thiết kế kiến trúc, đặc tả phần mềm	Đặc tả yêu cầu phần mềm
Phương pháp	Hộp trắng (hộp đen)	Hộp đen, hộp trắng	Hộp đen, mô hình	Hộp đen
Người tiến hành	Lập trình viên	Nhóm kiểm thử độc lập	Nhóm kiểm thử chuyên	Người dùng

5.4.1. Kiểm thử đơn vị

Mô hình kiểm thử đơn vị được mô tả như ở hình 5.10. Nó nhằm vào kiểm tra đơn vị thiết kế nhỏ nhất – một môđun phần mềm. Một môđun hoạt động thường có trao đổi thông tin với môđun thứ cấp (mức dưới) hay môđun thượng cấp (mức trên) của nó. Do đó phạm vi phát hiện lỗi liên quan chặt chẽ đến các môđun này.

Người tiến hành kiểm thử đơn vị thường là người lập trình môđun đó hoặc lập trình viên cùng nhóm. Kỹ thuật kiểm thử đơn vị chủ yếu là kiểm thử hộp trắng, trong trường hợp cần thiết có thể sử dụng thêm kiểm thử hộp đen.



Hình 5.10. Sơ đồ mô tả kiểm thử đơn vị

Nội dung kiểm thử đơn vị bao gồm: *kiểm thử giao diện* (các tham số vào/ra qua giao diện), *kiểm thử vào/ra* (các file, bộ đệm và các lệnh đóng mở), *kiểm thử cấu trúc dữ liệu cục bộ* (khai báo và sử dụng biến), *kiểm thử xử lý* (các phép toán và tính đúng đắn của kết quả), *kiểm thử điều kiện logic*, *kiểm thử sai tiềm ẩn* (về ngoại lệ, mô tả), *kiểm thử các giá trị biên* (các chu trình).

Thông thường, các môđun thứ cấp của môđun cần kiểm thử chưa được phát triển thì sẽ được thay bằng *cuồng* (*stub*). *Cuồng* là một chương trình máy tính dùng để thay thế cho một môđun phần mềm sẽ được xác định sau (IEEE, 1990) hoặc là một thành phần giả, có thể là một đối tượng thay thế các hoạt động của thành phần thực [3]. Đơn giản, *cuồng* là đoạn mã dùng để mô phỏng hoạt động của thành phần còn thiếu. Môđun thượng cấp chưa có được thay bằng *bộ lái* (*driver*). *Bộ lái* là một môđun phần mềm được sử dụng để gọi một môđun cần kiểm thử bên dưới. Nó cung cấp dữ liệu đầu vào để kiểm tra, điều khiển, thi hành và đưa ra kết quả kiểm tra (IEEE, 1990) hoặc đơn giản là một dòng lệnh dùng để gọi một phương thức và truyền giá trị cho phương thức đó. *Bộ lái* có nhiệm vụ đọc dữ liệu kiểm thử, kích hoạt môđun cần kiểm thử và so sánh kết quả đầu ra so với kết quả mong muốn:

Giả sử chúng ta cần kiểm thử môđun tính lịch Calendar(), trong đó môđun này gọi môđun tính ngày trong tuần calc_day() chưa được phát triển. Khi đó calc_day() có thể được thay bằng một *cuồng* đơn giản như sau:

```
String calc_day(Date d)
{
    return "Sunday";
}
```

Để tăng độ thích nghi cho các *cuồng*, một kỹ thuật thường dùng là yêu cầu người kiểm tra nhập kết quả trả lại từ bàn phím và sau đó *cuồng* trả lại cho môđun gọi nó. Ví dụ, *cuồng* trên có thể được cải tiến thành dạng sau:

```
String calc_day(Date d)
{
    String s;
    cout << "Enter day_of_week of "<< d;
    cin >> s;
    return s;
}
```

Ngược lại, giả sử chúng ta cần kiểm thử chính môđun tính ngày trong tuần calc_day(). Dưới đây là một ví dụ đơn giản về *bộ lái* của nó, có nhiệm vụ đọc dữ liệu kiểm thử từ bàn phím và in kết quả ra màn hình.

```

void calc_day_test_drive()
{
    Date d;
    String s;
    while (1)
    {
        cout << "Enter date: ";
        cin >> d;
        s = calc_day(d);
        cout << s << endl;
    }
}

```

Viết ra bộ lái và cuồng cần các chi phí bổ sung, nhưng được tính vì nó không được phân phối kèm với sản phẩm phần mềm. Chi phí này thấp nếu nó đơn giản, nhưng trong thực tế thường là cao. Khi đó người ta đành hoãn kiểm thử đầy đủ môđun, các yêu cầu còn lại để dành cho kiểm thử tích hợp sau này.

Một số công cụ trợ giúp kiểm thử đơn vị như *Junit*, là phần mềm miễn phí do JUnit.org (<http://www.junit.org/>) cung cấp; *CppUnit* hỗ trợ cho môi trường C++ do SourceForge (<http://cppunit.sourceforge.net/>) cung cấp miễn phí; và *HtmlUnit* là công cụ kiểm thử đơn vị với Java trong ứng dụng web của Gargoyl Software (<http://htmlunit.sourceforge.net/>).

5.4.2. Kiểm thử tích hợp

Sau khi từng môđun được kiểm thử riêng rẽ, công việc quan trọng để tạo nên những phần chức năng của hệ thống phần mềm hoàn chỉnh là tích hợp các môđun và kiểm thử như một thể thống nhất. Các đơn vị phần mềm được tích hợp dần thành các môđun chức năng, các hệ con và cuối cùng là hệ thống hoàn chỉnh. Kiểm thử này thường do các nhóm kiểm thử độc lập thực hiện. Phương pháp sử dụng chính là hộp đen. Phương pháp hộp trắng được sử dụng bổ sung khi cần kiểm thử lại môđun.

Những lỗi cần được phát hiện khi kiểm thử tích hợp là lỗi dữ liệu qua giao diện, hiệu ứng của một môđun gây lỗi cho môđun khác, lỗi xung đột về bộ nhớ hay các thiết bị ngoại vi,... Dưới đây là một số lỗi giao diện (môđun) điển hình:

- *Sử dụng sai giao diện*: Tham số được truyền giữa các môđun có thể không tương thích về kiểu hay về thứ tự. Đây là một lỗi rất phổ biến.
- *Hiểu nhầm về giao diện*: Môđun gọi có thể hiểu nhầm về chức năng của môđun được gọi, do đó dẫn đến các kết quả sai.
- *Xung đột*: Có nhiều kiểu xung đột có thể xảy ra, điển hình là trong các hệ thống thời gian thực hay các hệ thống cơ sở dữ liệu khi các môđun tương tác với thiết bị nhưng không đồng bộ với nhau.

Khi tích hợp các môđun, người kiểm thử phải dựa trên các đặc tả chức năng để thiết kế các ca kiểm thử phát hiện các lỗi giao diện nêu trên. Có nhiều cách (chiến lược) tích hợp các môđun. Hai cách tiếp cận tự nhiên để tích hợp các môđun là:

- *Tích hợp từng bước*. Theo cách này có ba chiến lược:
 - + Tích hợp từ dưới lên (*bottom – up testing*);
 - + Tích hợp từ trên xuống (*top – down testing*);
 - + Kết hợp cả dưới lên và trên xuống (*sandwich testing*).
- *Tích hợp đồng thời*: Kiểm thử vụ nổ lớn (*big bang testing*).

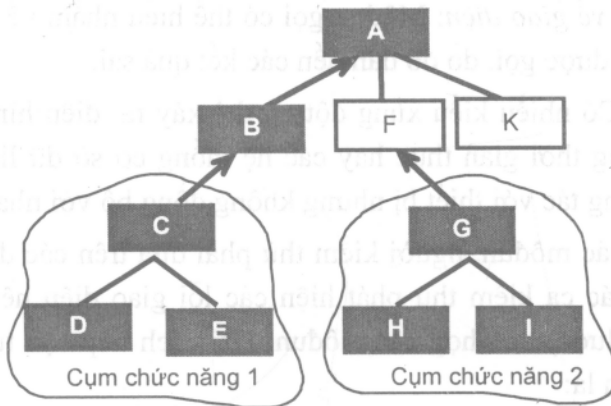
Ngoài ra, cần áp dụng một loại kiểm thử bổ sung: *kiểm thử quay lui* được sử dụng mỗi khi tích hợp thêm môđun mới hay cập nhật lại chương trình.

a) Kiểm thử từ dưới lên

Kiểm thử từ dưới lên là quá trình tích hợp và kiểm thử bắt đầu với các môđun ở mức thấp. Để kiểm thử các môđun cấp dưới, lúc đầu người ta dùng các bộ lái, sau đó thay thế dần các bộ lái bằng các môđun thượng cấp đã được phát triển.

Hình 5.11 mô tả chiến lược kiểm thử từ dưới lên. Thông thường, người ta không tuân tủy kiểm thử tất cả các môđun ở tầng dưới, mà nhóm chúng thành các nhóm chức năng, tích hợp và kiểm thử chúng theo từng cụm. Đồng thời để sớm có phiên bản thực hiện, người ta cũng không tiến hành kiểm thử hết các cụm ngang hàng mà tiến hành tích hợp ngược lên một số môđun cấp trên. Ví dụ như trong hình, sau khi tích hợp và kiểm thử cụm chức năng C, D, E, người ta sẽ tiến hành tích hợp với môđun B, sau đó là A

để có một phiên bản làm việc hạn chế, mà không tích hợp với cụm **G, H, I**. Khi đó các môđun **F** và **K** còn chưa phát triển sẽ được thay bằng các *cuồng*.



Hình 5.11. Tích hợp từ dưới lên

Kiểm thử từ dưới lên có một số ưu điểm sau:

- Tránh phải tạo các *cuồng* phức tạp hay tạo các kết quả nhân tạo: Do tích hợp từ dưới lên nên chỉ cần tạo ra các bộ lái, các môđun mức dưới đã được phát triển và kiểm thử.
- Thuận tiện cho phát triển các môđun thứ cấp dùng lại được: Nhờ phát triển từ dưới lên, người thiết kế có thể thiết kế các môđun dịch vụ dùng chung cho nhiều chức năng của hệ thống.

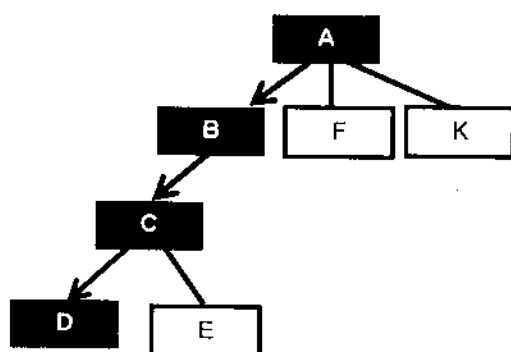
Tuy nhiên, một số nhược điểm của phương pháp từ dưới lên là:

- Chậm phát hiện các lỗi thiết kế: Do phát triển từ dưới lên, nên các lỗi tổng thể như phát triển sai chức năng hay hệ thống kém hiệu quả thường bị phát hiện muộn. Do lỗi phát hiện muộn nên chi phí và thời gian sửa lỗi tăng cao.
- Chậm có phiên bản của hệ thống thực hiện được: Các môđun mức trên được phát triển sau và tích hợp sau cùng, nên chậm có phiên bản thực hiện. Trong nhiều trường hợp có thể làm khách hàng mất niềm tin vì họ không được thử nghiệm sớm.

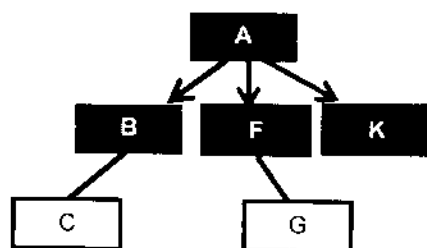
b) Kiểm thử từ trên xuống

Kiểm thử từ trên xuống tiến hành kiểm thử các môđun bắt đầu từ mức cao, các môđun mức thấp tạm sử dụng các chức năng hạn chế, có giao diện giống như trong đặc tả. Môđun mức thấp có thể chỉ đơn giản là các *cuồng*

trả lại kết quả với một vài đầu vào được xác định trước. Sau đó các cuống được thay dần bằng các môđun thực đã được phát triển.



a) Theo chiều sâu



b) Theo chiều rộng

Hình 5.12. Tích hợp từ trên xuống

Hình 5.12 mô tả một quá trình tích hợp và kiểm thử từ trên xuống. Với chiến lược này ta lại có thể thực hiện *chiến lược theo chiều sâu* hay *theo chiều rộng*. Thông thường, để sớm có phiên bản thực hiện được, người ta tích hợp theo một nhánh cho đến các môđun cấp thấp nhất (*theo chiều sâu*). Như trong hình 5.12a, sau khi tích hợp môđun B thì ta tích hợp tiếp các môđun cấp dưới của nó là C, rồi D, thay vì việc tích hợp tiếp các môđun F rồi đến K (theo chiều rộng – hình 5.12b).

Kiểm thử từ trên xuống có các ưu điểm như:

- *Phát hiện sớm các lỗi thiết kế:* Cách tiếp này dễ dàng phát hiện các lỗi như phát triển nhầm, thiếu chức năng so với đặc tả, do đó có thể giảm chi phí cho việc thiết kế và cài đặt lại.
- *Có phiên bản hoạt động sớm:* Kiểm thử từ trên xuống luôn đảm bảo có phiên bản hoạt động sớm, do đó có thể thẩm định tính đúng đắn được của sản phẩm và dùng nó để huấn luyện người dùng.

Nhược điểm của chiến lược này là các chức năng của môđun cấp thấp nhiều khi rất phức tạp, có thể khó mô phỏng được (hoặc khó tạo cuống), có thể dẫn đến việc không kiểm thử đầy đủ các chức năng của môđun.

Có thể nói, mỗi chiến lược kiểm thử trên đều có các ưu, nhược điểm riêng và nó thường tương ứng với chiến lược phát triển hệ thống. Trên thực

tế, người ta có thể phối hợp hai chiến lược này và ta có chiến lược kiểm thử lai (*sandwich testing*).

c) *Kiểm thử quay lui*

Một công đoạn kiểm thử đặc biệt khi tích hợp hệ thống là *kiểm thử quay lui* hay *kiểm thử hồi quy* (*regression testing*), tức là tiến hành lại các phép thử đã thành công mỗi khi tích hợp thêm môđun hoặc khi cập nhật mã nguồn chương trình.

Khi chúng ta tích hợp thêm môđun vào hệ thống, với một tổ hợp trạng thái mới, rất có thể lỗi sẽ xuất hiện ở môđun trước đây chưa gây lỗi. Một mặt khi kiểm thử, chúng ta sửa mã nguồn chương trình để khắc phục một lỗi E nào đó thì rất có thể gây ra một lỗi R ở môđun đã sửa. Trường hợp xấu hơn là nó sinh ra lỗi mới mà trước đây chưa có. Điều tương tự cũng xảy ra khi tiến hành nâng cấp chương trình. Vì vậy, trong những trường hợp này cần tiến hành lại các kiểm thử đã thực hiện thành công để đề phòng các hiệu ứng phụ không mong đợi.

Các phép thử hồi quy được chia ra làm ba loại khác nhau:

- *Các phép thử đại diện* sẽ thực hiện tất cả các chức năng của phần mềm.
- *Các phép thử bổ sung* tập trung vào những chức năng dễ bị ảnh hưởng nhất khi phần mềm có sự thay đổi.
- *Các phép thử tập trung* vào những thành phần phần mềm bị thay đổi.

Kiểm thử hồi quy có thể được tiến hành thủ công, nhưng với hàng trăm, hàng nghìn ca kiểm thử thì cách tiến hành thủ công là không thực tế, vì chi phí thời gian và công sức quá lớn. Giải pháp cho trường hợp này là sử dụng công cụ kiểm thử tự động (như Junit). Các công cụ này sẽ chạy một cách định kỳ (thường là hàng đêm) để phát hiện các lỗi phát sinh khi hệ thống phần mềm có sự thay đổi.

Mặt khác, khi số các phép thử rất lớn, chi phí cho chúng nhiều khi đến mức lớn không thể chấp nhận và không hiệu quả. Trong trường hợp này cần có những kỹ thuật để giảm chi phí và nâng cao hiệu quả của quá trình kiểm thử hồi quy.

Có nhiều kỹ thuật đã được đề xuất nhằm giảm chi phí cho quá trình kiểm thử hồi quy như: *kỹ thuật rút gọn quy mô kiểm thử (test suite reduction)*, *kỹ thuật ca kiểm thử ưu tiên (test cases prioritization)*, *kỹ thuật kiểm thử có lựa chọn (regression test selection)*. Trong các kỹ thuật trên, không có kỹ thuật nào là tuyệt đối hoàn hảo. Mỗi kỹ thuật đều phải "trả giá" nào đó để có được kết quả mong muốn.

d) Kiểm thử big bang

Kiểm thử vụ nổ lớn (big bang testing) là một chiến lược kiểm thử tích hợp hệ thống một lần duy nhất để được môđun chức năng (hay hệ thống) hoàn chỉnh. Phương pháp này vẫn thường được tiến hành khi phát triển các phần mềm có kích thước nhỏ. Tuy nhiên, đối với các dự án phần mềm lớn thì ta nên tránh sử dụng phương pháp này, vì rất khó định vị chính xác được môđun hay vị trí gây lỗi.

5.4.3. Kiểm thử hệ thống

Sau khi đã tiến hành tích hợp tất cả các môđun thành một hệ thống hoàn thiện, ta chuyển sang kiểm thử khả năng hoạt động của cả hệ thống. Ở giai đoạn này, thay vì phát hiện các sai sót về đặc tả hay giao diện giữa các môđun, người ta chuyển sang kiểm tra các đặc trưng của toàn hệ thống, như *hiệu năng của hệ thống, khả năng phục hồi khi gặp sự cố, độ tin cậy, ...* Việc kiểm thử hệ thống do nhóm kiểm thử độc lập – chuyên về hệ thống – thực hiện. Phương pháp chính sử dụng ở đây là phương pháp hộp đen. Ngoài ra còn nhiều phương pháp khác được sử dụng, trong đó phải kể đến phương pháp mô phỏng. Dưới đây là một số cách kiểm thử hệ thống.

a) Kiểm thử phục hồi

Nhiều hệ thống thông tin cần phải phục hồi (tiếp tục cung cấp dịch vụ) sau khi gặp sự cố trong một khoảng thời gian nhất định. Trong một số trường hợp, hệ thống cần phải có khả năng dung thứ lỗi, tức là dù có xảy ra lỗi như thế nào thì toàn bộ hệ thống cũng không được ngừng hoạt động.

Kiểm thử phục hồi (recovery testing) là các kiểm thử được tiến hành nhằm gây lỗi làm cho hệ thống ngừng hoạt động và đánh giá khả năng phục hồi sau đó. Với các hệ thống có khả năng phục hồi tự động, ta cần đánh giá khả năng thiết lập lại các tham số hệ thống, khôi phục lại dữ liệu và tái khởi

động. Với các trường hợp đòi hỏi khởi động lại thủ công, ta cần đánh giá thời gian ngừng để sửa chữa (*MTTR – mean time to repair*) và trong một số trường hợp cần đánh giá cả chi phí cho việc sửa chữa khôi phục này có chấp nhận được hay không.

b) Kiểm thử áp lực

Đối với một số hệ thống quan trọng, người ta cần tiến hành *kiểm thử áp lực (stress testing)*. Đây là loại kiểm thử được tiến hành khi đã có phiên bản làm việc, nhằm đánh giá hoạt động của hệ thống trong các trường hợp bất thường (dữ liệu lớn, số người sử dụng nhiều, tài nguyên hạn chế,...). Mục đích của kiểm thử áp lực là:

- Đánh giá giới hạn chịu tải của hệ thống;
- Khảo sát các đặc trưng của hệ thống khi đạt và vượt giới hạn chịu tải (khi bị sụp đổ).

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

c) Kiểm thử hiệu năng

Đối với rất nhiều phần mềm (phần mềm thời gian thực, phần mềm nhúng), việc đánh giá hiệu năng hoạt động là đặc biệt quan trọng.

Kiểm thử hiệu năng (performance testing) được thiết kế để đánh giá hiệu suất hoạt động có hiệu quả của phần mềm trong một ngữ cảnh cụ thể, thông thường là trong một môi trường tích hợp với các phần mềm và phần cứng cụ thể.

Kiểm thử hiệu năng nên được tiến hành ở tất cả các công đoạn kiểm thử. Ví dụ, ở mức kiểm thử đơn vị, chúng ta có thể đánh giá hiệu năng của từng đơn vị thông qua kiểm thử hộp trắng. Tuy nhiên, việc đánh giá hiệu năng thực sự của toàn bộ hệ thống chỉ có ý nghĩa khi hệ thống đã được tích hợp. Kiểm thử hiệu năng liên quan chặt chẽ đến ngữ cảnh sử dụng bao gồm cả các phần mềm khác (hệ điều hành, cơ sở dữ liệu,...) và môi trường phần cứng (CPU, bộ nhớ, mạng). Kiểm thử hiệu năng thường được tiến hành cùng với kiểm thử áp lực.

d) Kiểm thử an ninh

Kiểm thử an ninh (security testing) nhằm làm cho phần mềm có thể hoạt động được trước các cuộc tấn công hiểm độc: trước mặt, sau lưng hay

Có nhiều kỹ thuật đã được đề xuất nhằm giảm chi phí cho quá trình kiểm thử hồi quy như: *kỹ thuật rút gọn quy mô kiểm thử (test suite reduction)*, *kỹ thuật ca kiểm thử ưu tiên (test cases prioritization)*, *kỹ thuật kiểm thử có lựa chọn (regression test selection)*. Trong các kỹ thuật trên, không có kỹ thuật nào là tuyệt đối hoàn hảo. Mỗi kỹ thuật đều phải "trả giá" nào đó để có được kết quả mong muốn.

d) Kiểm thử big bang

Kiểm thử vụ nổ lớn (big bang testing) là một chiến lược kiểm thử tích hợp hệ thống một lần duy nhất để được mô đun chức năng (hay hệ thống) hoàn chỉnh. Phương pháp này vẫn thường được tiến hành khi phát triển các phần mềm có kích thước nhỏ. Tuy nhiên, đối với các dự án phần mềm lớn thì ta nên tránh sử dụng phương pháp này, vì rất khó định vị chính xác được mô đun hay vị trí gây lỗi.

5.4.3. Kiểm thử hệ thống

Sau khi đã tiến hành tích hợp tất cả các mô đun thành một hệ thống hoàn thiện, ta chuyển sang kiểm thử khả năng hoạt động của cả hệ thống. Ở giai đoạn này, thay vì phát hiện các sai sót về đặc tả hay giao diện giữa các mô đun, người ta chuyển sang kiểm tra các đặc trưng của toàn hệ thống, như *hiệu năng của hệ thống, khả năng phục hồi khi gặp sự cố, độ tin cậy,...* Việc kiểm thử hệ thống do nhóm kiểm thử độc lập – chuyên về hệ thống – thực hiện. Phương pháp chính sử dụng ở đây là phương pháp hộp đen. Ngoài ra còn nhiều phương pháp khác được sử dụng, trong đó phải kể đến phương pháp mô phỏng. Dưới đây là một số cách kiểm thử hệ thống.

a) Kiểm thử phục hồi

Nhiều hệ thống thông tin cần phải phục hồi (tiếp tục cung cấp dịch vụ) sau khi gặp sự cố trong một khoảng thời gian nhất định. Trong một số trường hợp, hệ thống cần phải có khả năng dung thứ lỗi, tức là dù có xảy ra lỗi như thế nào thì toàn bộ hệ thống cũng không được ngừng hoạt động.

Kiểm thử phục hồi (recovery testing) là các kiểm thử được tiến hành nhằm gây lỗi làm cho hệ thống ngừng hoạt động và đánh giá khả năng phục hồi sau đó. Với các hệ thống có khả năng phục hồi tự động, ta cần đánh giá khả năng thiết lập lại các tham số hệ thống, khôi phục lại dữ liệu và tái khởi

bên sườn thông qua các lỗ hổng. Những lỗ hổng an ninh trong phần mềm được tạo ra từ hai nguồn: *những lỗi trong trình lập trình*, hoặc *những khiếm khuyết của thiết kế phần mềm*. Có rất nhiều loại lỗ hổng mà các nhà nghiên cứu an ninh máy tính đã chỉ ra và phân loại. Những lỗi trong lập trình có thể dẫn đến lỗ hổng như: *lỗi tràn bộ đệm, dùng biến môi trường không an toàn, dùng lời gọi hàm hệ thống không an toàn, truy nhập cơ sở dữ liệu không an toàn*. Những khiếm khuyết trong thiết kế như: *thiếu sót trong áp dụng thuật toán mã hóa, thiếu sót trong kiến trúc các tầng bảo vệ và thiết kế phân quyền*.

Thuật ngữ "an ninh phần mềm không phải là phần mềm an ninh" cung cấp một chỉ dẫn quan trọng cho kiểm thử an ninh. Mặc dù các công cụ an ninh như mã hóa, xác thực và điều khiển truy cập đóng một vai trò thiết yếu trong an ninh phần mềm, nhưng không thể coi an ninh như những đặc trưng hay cơ chế, mà là một thuộc tính quan trọng của toàn bộ hệ thống. Bởi vậy, kiểm thử an ninh phần mềm nhất thiết phải bao gồm hai cách tiếp cận riêng biệt:

- Kiểm thử các cơ chế an ninh để đảm bảo những chức năng được thực thi đúng đắn.
- Thực hiện kiểm thử an ninh dựa trên việc hiểu biết và mô phỏng cách tiếp cận của những kẻ tấn công thực sự, những tin tặc.

Như vậy, để kiểm thử an ninh đòi hỏi kiểm thử viên phải có kiến thức sâu rộng trong lĩnh vực an ninh thông tin, thậm chí là kinh nghiệm tấn công của tin tặc.

Từ trên cho thấy, tất cả mọi người từ các kiến trúc sư hệ thống tới nhân viên lập trình, nhân viên đảm bảo chất lượng phần mềm đều cần tham gia vào quá trình kiểm thử an ninh phần mềm trong suốt vòng đời phát triển phần mềm. Khi kiểm thử an ninh, nhân viên kiểm thử đóng vai trò của kẻ xâm nhập tìm mọi cách tấn công vào hàng rào phòng thủ được dựng lên: tìm kiếm mật khẩu, bẻ gãy khóa, làm ngập lụt hệ thống khi tạo ra việc từ chối dịch vụ; gây ra các lỗi hệ thống; sử dụng các dữ liệu không an toàn,...

Việc đặc tả yêu cầu an ninh là rất khó khăn. Bằng việc phân tích rủi ro, đặc biệt ở tầng thiết kế, có thể giúp ta xác định các nguy cơ an ninh thiết yếu

và tác động của chúng. Thông qua đó xác định được các lỗ hổng và đánh giá mức độ hiểm họa từ chúng để phát triển các ca kiểm thử an ninh hiệu quả.

An ninh liên quan tới các thông tin và các dịch vụ được bảo vệ, các kỹ năng và nguồn lực của những kẻ tấn công, chi phí tạo công cụ bảo vệ và việc phục hồi hệ thống sau khi bị tấn công. Về nguyên tắc, mọi đột nhập đều có thể thực hiện nếu có đủ thời gian và nguồn lực. Bài toán thiết kế hệ thống an ninh đặt ra là:

- Phí đột nhập thành công phải cao hơn giá trị thu được từ đột nhập.
- Phí xây dựng hàng rào bảo vệ phải nhỏ hơn giá trị bị mất nếu bị đột nhập.

Các cuộc tấn công luôn xảy ra, cho dù phần mềm được bảo vệ bằng bất kỳ kỹ thuật an ninh tốt nào. Do vậy, việc ứng dụng các hiểu biết về tấn công và khai thác lỗ hổng cần được áp dụng liên tục trong các tổ chức phát triển phần mềm. Cuối cùng, các tình huống lạm dụng nên được kiểm thử sớm trong vòng đời phát triển phần mềm để làm tăng hiệu quả của một kế hoạch kiểm thử. Kiểm thử an ninh rất cần đến cách suy nghĩ của kẻ tấn công cũng như cách làm của những người bảo vệ.

e) Kiểm thử so sánh

Đối với một số phần mềm (như cho máy bay, lò hạt nhân,...), độ tin cậy của nó đóng vai trò tuyệt đối quan trọng, người ta thường gọi là *phần mềm tuyệt đối đúng*. Trong các trường hợp này, phần cứng và phần mềm thường được sử dụng để làm cực tiểu khả năng gây lỗi. Khi phát triển, các đội kỹ sư sử dụng cùng một đặc tả để tạo ra các phiên bản phần mềm độc lập nhau. Mỗi phiên bản được kiểm thử với cùng một ca kiểm thử, cùng dữ liệu đầu vào để đảm bảo rằng tất cả các đầu ra là như nhau. Nếu đầu ra khác nhau, chắc chắn trong các phiên bản phần mềm còn có lỗi. Việc so sánh các đầu ra có thể được thực hiện bằng các công cụ tự động.

Kỹ thuật kiểm thử hộp đen với các phiên bản độc lập được gọi là *kiểm thử so sánh*. Nó còn được gọi là *kiểm thử dựa lưng vào nhau (back to back testing)*.

5.4.4. Kiểm thử chấp nhận

Thông thường, sau khi kiểm thử hệ thống là *kiểm thử chấp nhận (acceptance testing)* hay *kiểm thử thẩm định (validation testing)*, được

khách hàng thực hiện. Mục đích của kiểm thử này là để khách hàng đánh giá xem phần mềm có thỏa mãn tất cả yêu cầu của họ không để chấp nhận sản phẩm.

Đối với những sản phẩm bán rộng rãi trên thị trường cho nhiều người sử dụng, quá trình kiểm thử trải qua hai bước gọi là *kiểm thử Alpha* và *kiểm thử Beta*. Với kiểm thử Alpha, người sử dụng thực hiện kiểm thử trong sự quản lý của nhà phát triển, lập trình viên sẽ ghi nhận các lỗi hoặc phản hồi và lên kế hoạch sửa chữa. Với kiểm thử Beta, phần mềm sẽ được gửi tới cho người sử dụng tiềm năng để kiểm thử trong môi trường thực. Lỗi hoặc phản hồi cũng sẽ gửi ngược lại cho nhà phát triển để sửa chữa.

5.4.5. Kiểm thử hệ thời gian thực

Hệ thời gian thực đòi hỏi phải đáp ứng đúng, chính xác các sự kiện của môi trường. Nó thường bao gồm các thành phần phần cứng và phần mềm kết thành các cụm thực hiện các chức năng chính sau: *tiếp nhận dữ liệu* (cảm biến), *kiểm soát dữ liệu vào*, *xử lý dữ liệu*, *điều khiển thao tác* đáp ứng lại môi trường qua bộ thao tác (hình 3.33).

Kiểm thử hệ thời gian thực được thực hiện theo chiến lược 4 bước:

Bước 1. Kiểm thử tác vụ: Kiểm thử từng tác vụ một cách độc lập với nhau bằng các kỹ thuật khác nhau. Bước này cho phép phát hiện sai về logic và chức năng, nhưng chưa phát hiện sai về thời gian và ứng xử của phần mềm.

Bước 2. Kiểm thử ứng xử: Sử dụng công cụ CASE tạo mô hình hệ thống để mô phỏng ứng xử của hệ thời gian thực, xem ứng xử của nó như hậu quả của việc đáp ứng nguyên nhân – các sự kiện từ bên ngoài. Dùng kết quả hoạt động phân tích này để thiết kế các ca kiểm thử (tương tự đồ thị nhân quả):

- *Phân lớp sự kiện tương đương*, kiểm thử từng lớp sự kiện và nhận ứng xử của hệ thi hành để phát hiện các sai khi xử lý đáp ứng các sự kiện đó.
- *Kiểm thử mọi lớp sự kiện*. Các sự kiện được đưa vào trong hệ thống theo thứ tự ngẫu nhiên, với tần suất ngẫu nhiên nhằm phát hiện các sai ứng xử trong tình huống khác thường.

Bước 3. Kiểm thử liên tác: Tìm các sai liên quan đến đáp ứng thời gian do không đồng bộ:

- Các tác vụ không đồng bộ liên tác với các tác vụ khác. Vì thế cần kiểm thử nhịp điệu dữ liệu và mức tải với các xử lý khác nhau.
- Các tác vụ không đồng bộ do giao tiếp phụ thuộc hàng đợi thông điệp, hoặc truy nhập kho dữ liệu cũng được thử để bộc lộ các sai về sự phù hợp kích cỡ dữ liệu.

Bước 4. Kiểm thử hệ thống: Sau khi tích hợp phần cứng và phần mềm, cần tiến hành kiểm thử hệ thống để tìm ra các sai:

- *Giao diện*, lỗi giữa phần cứng và phần mềm.
- *Môi trường*, đánh giá tác động từ môi trường đến các sự kiện.
- *Các ngắt*, sự tác động của các loại ngắt đối với toàn bộ chương trình.

Ở đây sử dụng chiến lược *kiểm thử luồng sợi (thread testing)*, hay *kiểm thử luồng giao tác (transaction – flow testing)* [2]. Nó gồm việc xác định và thực hiện chuỗi các xử lý tương ứng với lớp các sự kiện đầu vào. Tất nhiên, việc kiểm thử đầy đủ các luồng là không thể. Khi đó, các luồng sử dụng chung nhất có thể được xác định và được lựa chọn để kiểm thử.

5.4.6. Kiểm thử phần mềm phát triển theo hướng đối tượng

5.4.6.1. Các đặc trưng của hệ thống phần mềm hướng đối tượng

Kiến trúc của phần mềm hướng đối tượng được đặc trưng bởi tính *bao gói, tính kế thừa, tính đa hình và tương tác bằng truyền thông*. Vì thế, chúng khác với hệ thống phần mềm thông thường. Thêm vào đó, nó có thể được phát triển theo những tiến trình khác nhau, nên những quy tắc liên quan đến kiểm thử phần mềm hướng đối tượng cũng khác so với phần mềm thông thường.

Do *tính bao gói* thông tin của một lớp, các phương thức của lớp tiến hành xử lý chỉ trên các dữ liệu của lớp đó. Vì thế, một lớp là một thành phần độc lập và là đơn vị hoàn chỉnh có thể tiến hành kiểm thử như một môđun

đơn vị. Do tính kế thừa, một lớp con được tạo thành từ một lớp cha có sẵn, nên khi kiểm thử lớp con đòi hỏi phải kiểm thử lại cả lớp cha của nó. Ngoài ra, các phương thức của một lớp có thể được thực hiện theo những chuỗi khác nhau có thể có.

Tính đa hình cho phép cài đặt sử dụng cùng tên một giao diện cho nhiều hơn một phương thức thực hiện. Sự ràng buộc đối với phương thức có thể là tĩnh hoặc động. Khi ràng buộc là động thì các phương thức thực thi phụ thuộc vào lời gọi hàm tại thời điểm thực thi này. Điều này đòi hỏi phải thiết kế ca kiểm thử phương thức xác định tại mỗi thời điểm có lời gọi hàm xảy ra.

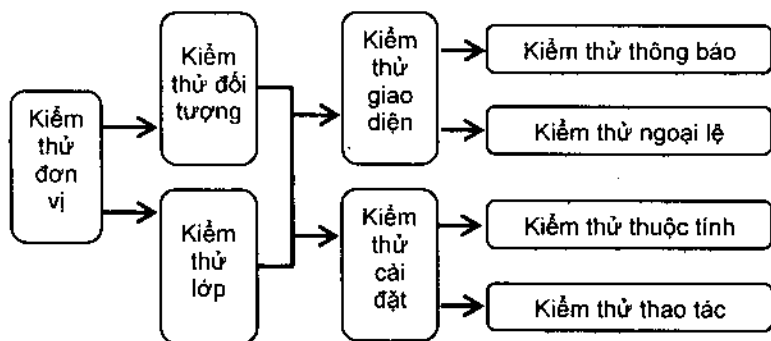
5.4.6.2. Tiến trình kiểm thử phần mềm hướng đối tượng

Kiểm thử phần mềm hướng đối tượng được thực hiện ở ba mức: *mức đơn vị*, *mức tích hợp* và *mức hệ thống*. Tại mức đơn vị, lớp là đối tượng kiểm thử. Các lớp được tích hợp dựa trên mối quan hệ như kế thừa và kết hợp để hình thành một đơn vị chức năng lớn hơn. Kiểm thử tích hợp cho phép nhận biết những lỗi trong quá trình tích hợp các đơn vị là các lớp. Kiểm thử hệ thống bao gồm kiểm thử toàn bộ phần mềm.

a) Kiểm thử đơn vị

Kiểm thử lớp bao gồm kiểm thử các phương thức được xác định và sự tác động của chúng trên trạng thái của lớp hoặc các đối tượng. Trạng thái của lớp hoặc các đối tượng cần phải được theo dõi cả trước và sau khi lớp thực thi xong. Kiểm thử lớp hướng đến phát hiện lỗi liên quan đến thao tác của phương thức, đến sự thay đổi trạng thái và đến điều khiển các phương thức theo các chuỗi khác nhau. Ba phương pháp được sử dụng cho kiểm thử đơn vị phần mềm hướng đối tượng là:

- Kiểm thử hộp trắng.
- Kiểm thử hộp đen.
- Kiểm thử hộp xám (*gray – box testing*). Kiểm thử hộp xám dựa trên kiểm tra đặc tả phần mềm nhằm giải quyết những vấn đề đặt ra của phần mềm hướng đối tượng mà hai phương pháp trên chưa phát hiện được. Chẳng hạn, những vấn đề liên quan đến tính đa hình. Sơ đồ kiểm thử đơn vị cho ở hình 5.13.



Hình 5.13. Sơ đồ các kiểm thử đơn vị

b) Kiểm thử tích hợp

Kiểm thử tích hợp kiểm tra các lỗi trong sự tương tác giữa các lớp của những phần chức năng nhỏ. Một cách tiếp cận chung của kiểm thử tích hợp là sử dụng các cụm, nó đơn giản là một nhóm các lớp và các phương thức của chúng được yêu cầu để thực hiện một số chức năng chung nào đó. Kiến trúc của phần mềm hướng đối tượng dựa trên sự tương tác "truyền thông trực tiếp", không phải là điều khiển phân cấp, nên các chiến lược tích hợp từ trên xuống hay từ dưới lên là không có ý nghĩa với nó. Kiểm thử được tích hợp liên quan đến việc kiểm tra các lỗi thông qua các mối quan hệ khác nhau, theo mỗi trình tự đúng của việc gọi phương thức và cập nhật trạng thái chính xác cho việc cài đặt mỗi chức năng cụ thể. Các mối quan hệ kế thừa đòi hỏi kiểm thử lại các phương thức dẫn xuất khi chúng được sử dụng trong lớp con.

Kiểm thử tích hợp của việc cài đặt ràng buộc động là kiểm tra ràng buộc của các phương thức liên quan tại thời điểm chạy chương trình. Vì mô hình lập và tăng dần đối tượng được áp dụng để phát triển phần mềm hướng đối tượng, nên các hệ thống con trong phần mềm hướng đối tượng cũng kiểm thử tăng dần.

Một số kỹ thuật được dùng cho kiểm thử tích hợp là:

1. *Kiểm thử một phân lớp*: Kiểm thử mối quan hệ giữa các thể hiện của các lớp.
2. *Kiểm thử theo kịch bản*: Kiểm thử sự tương tác của các đối tượng khác nhau mà đáp ứng được sự mong đợi của một ca kiểm thử đã được đặc tả.

3. *Kiểm thử chấp nhận*: Tìm ra các lỗi và các mâu thuẫn trong sự tích hợp của các thành phần so với yêu cầu đề ra (thường cho trong đặc tả ca sử dụng).
4. *Kiểm thử dựa trên sự kiện*: Tìm ra các luồng điều khiển mà nó có thể không được thực hiện.
5. *Kiểm thử dựa trên UML*: Kết hợp đặc tả phân tích thiết kế phần mềm bằng ngôn ngữ mô hình hóa thống nhất UML theo các biểu đồ: biểu đồ ca sử dụng, biểu đồ lớp, biểu đồ tuần tự,... với công cụ kiểm thử Rational Rose để sinh mã xuôi và ngược.

c) Kiểm thử hệ thống

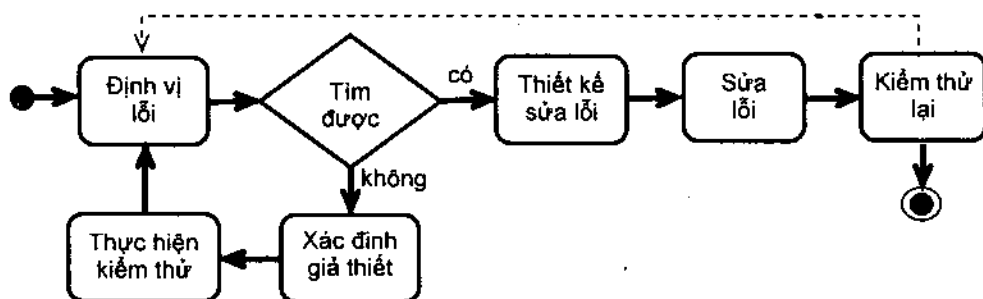
Trong kiểm thử hệ thống, toàn bộ phần mềm được kiểm tra xem nó có làm việc theo đúng đặc tả và đáp ứng đúng yêu cầu người dùng không. Cách kiểm thử ở đây cũng giống như kiểm thử hệ thống của phần mềm thông thường. Do vậy, các phương pháp kiểm thử hộp đen và phương pháp mô hình được sử dụng rộng rãi.

5.5. Gỡ lỗi

5.5.1. Tiến trình gỡ lỗi

Khi các lỗi được tìm ra trong chương trình, chúng phải được phân tích và gỡ bỏ. Hành động này gọi là *gỡ lỗi (debug)*. Phát hiện lỗi và gỡ lỗi đôi khi được coi như là các phần của tiến trình hoàn thiện chương trình. Thực tế chúng hoàn toàn khác nhau. Kiểm thử nhằm phát hiện sự tồn tại của các lỗi, còn gỡ lỗi quan tâm đến việc định vị lỗi và sửa các lỗi đó của chương trình.

Hình 5.14. minh họa một tiến trình gỡ lỗi chung nhất. Các lỗi trong mã nguồn phải được định vị và chương trình được sửa đổi để phù hợp các yêu cầu của nó. Sau đó phải kiểm thử lại để đảm bảo việc sửa lỗi là chính xác.



Hình 5.14. Tiến trình gỡ lỗi

Người gỡ lỗi phải đưa ra các giả thiết về hành động có thể quan sát của chương trình, sau đó kiểm tra các giả thiết có hy vọng tìm ra sai sót là nguyên nhân gây ra kết quả bất thường. Thực hiện kiểm thử các giả thiết có thể thực hiện chương trình từng bước bằng tay, cũng có thể đòi hỏi đặc tả ca kiểm thử mới để khoanh vùng vấn đề. Các công cụ thực hiện gỡ lỗi tương tác hiển thị các giá trị trung gian của các biến chương trình và vết của các câu lệnh thực hiện có thể được sử dụng để trợ giúp cho quá trình gỡ lỗi. Nguyên tắc Pareto được áp dụng cho kiểm thử và gỡ lỗi phần mềm: *80% tất cả sai sót tìm ra có thể thuộc về 20% nguyên nhân*. Như vậy, nếu định vị được 20% nguyên nhân này ta sẽ khắc phục được phần lớn lỗi với chi phí nhỏ.

5.5.2. Phương cách gỡ lỗi

Không thể đưa ra những chỉ dẫn cụ thể cho việc gỡ lỗi chương trình. Đôi khi người ta cho rằng, khả năng gỡ lỗi là bẩm sinh. Người gỡ lỗi cần tìm kiếm các mô hình cho các kết quả kiểm thử có xuất hiện lỗi và sử dụng hiểu biết về lỗi, mô hình và tiến trình thực hiện chương trình để định vị lỗi. Người gỡ lỗi phải biết các lỗi thường xảy ra của lập trình viên và so sánh ngược lại với các mô hình cần quan sát. Sau khi phát hiện lỗi trong chương trình, cần phải sửa lỗi và hệ thống cần được kiểm tra lại bằng kiểm thử hồi quy.

Gỡ lỗi là công việc rất khó khăn, vì triệu chứng có thể rất xa nguyên nhân, có thể tạm thời biến mất khi có một sai khác được sửa, có thể do người dùng sai mà không dễ theo dõi được, có thể lỗi là kết quả của vấn đề thời gian chứ không phải là vấn đề xử lý. Mặt khác, khó tái hiện lại các điều kiện đầu vào (trong ứng dụng thời gian thực, thứ tự đầu vào không xác định). Triệu chứng có thể là bị gián đoạn (các hệ nhúng liên kết giữa phần cứng và phần mềm không tháo rời được). Triệu chứng có thể do các nguyên nhân được phân tán trong một số các nhiệm vụ chạy trên các bộ xử lý khác nhau. Mặc dù gỡ lỗi là công việc khó quy trình hóa, người ta cũng đã nêu ra ba lựa chọn:

- Vũ dũng vô mưu.
- Lăn theo vết cũ.
- Loại trừ dần các nguyên nhân.

Phương cách vũ dũng vô mưu là cách dễ nhất, nhưng lại ít hiệu quả nhất. Chỉ áp dụng phương sách này khi không còn cách nào khác. Triết lý của cách này là "hãy để cho máy tính tìm ra sai": xem xét tất cả, ghi lại tất cả với hy vọng tìm ra trong đồng thông tin không lồ đó cái nguyên nhân của sai! Đây là cách nghĩ và thực hành thô thiển, ít khả thi.

Phương cách lần theo vết cũ cũng ít thông dụng và có thể cho kết quả với các chương trình nhỏ. Ta bắt đầu lần ngược từ chỗ có triệu chứng sai trong mã nguồn theo mỗi con đường có thể cho tới khi định vị được sai. Khi số dòng lệnh tăng lên thì số con đường lần ngược có thể nhiều đến mức không thể thực hiện được.

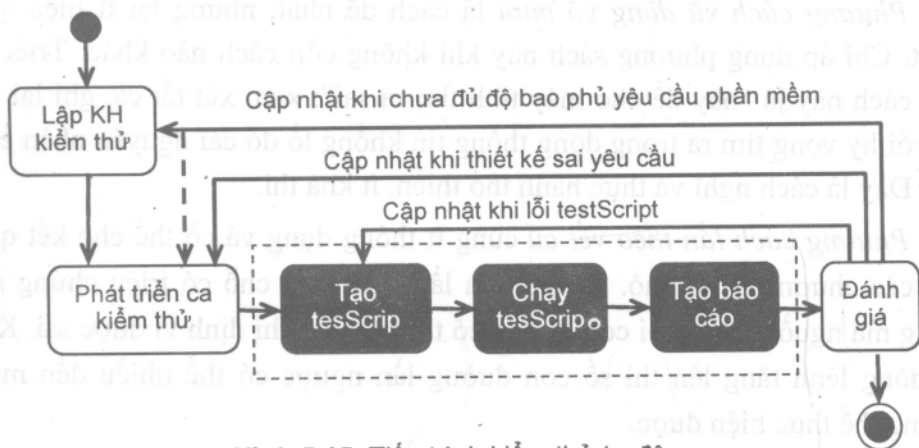
Phương cách loại trừ dần các nguyên nhân: Bằng cách tổ chức lại dữ liệu liên quan đến xuất hiện sai để cô lập nguyên nhân tiềm ẩn, Một số giả thiết về nguyên nhân được đưa ra. Khi sử dụng các dữ liệu có được, ta có thể minh chứng hoặc phản chứng các giả thiết đó. Từ đây ta có một danh sách các nguyên nhân tiềm ẩn và lập các ca kiểm thử tương ứng để loại trừ dần các nguyên nhân này. Nếu kiểm thử chỉ ra một nguyên nhân nào đó có hứa hẹn thì tinh chế tiếp dữ liệu liên quan để cô lập lỗi. Với những vùng dữ liệu sai, ta thường phân nhỏ hơn để tiếp tục tiến hành kiểm thử loại để nguyên nhân.

Mỗi khi xác định được nguyên nhân lỗi thì phải chỉnh sửa. Nhưng sửa một sai có thể gây ra sai khác. Vì thế, trước khi sửa cần đặt câu hỏi để tìm giải pháp thích hợp: Nguyên nhân này có thể sinh ra sai ở phần khác của chương trình hay không? Lỗi nào có thể được sinh ra tiếp? Cần làm thế nào để có thể ngăn cản các lỗi này ngay từ chỗ đầu tiên? (thường có nhiều giải pháp để nhà thiết kế lựa chọn).

5.6. Kiểm thử tự động

5.6.1. Khái niệm về kiểm thử tự động

Kiểm thử tự động phần mềm là tiến trình thực hiện một cách tự động một số bước của các ca kiểm thử, bằng cách sử dụng một công cụ nhằm rút ngắn thời gian và công sức kiểm thử. Tiến trình chung cho kiểm thử tự động phần mềm cũng giống như phát triển một dự án (hình 5.15).



Hình 5.15. Tiến trình kiểm thử tự động

Kiểm thử tự động thường được áp dụng khi thiếu tài nguyên, khi kiểm thử hồi quy hay khi phần mềm phải vận hành trong môi trường đặc biệt. Để kiểm thử tự động, cần thực hiện các công việc sau:

- Thu thập các đặc tả yêu cầu hoặc các ca kiểm thử, lựa chọn những phần cần thực hiện kiểm thử tự động.
- Phân tích và thiết kế mô hình cho kiểm thử tự động.
- Sử dụng công cụ để tiến hành kiểm thử, với các bước cho ở bảng sau:

Bước	Nội dung thực hiện	Mô tả
1	Tạo testScript	Sử dụng công cụ kiểm thử để ghi lại các thao tác lên phần mềm và sinh tự động testScript.
2	Chỉnh sửa testScript	Chỉnh testScript để thực hiện kiểm tra đúng yêu cầu đặt ra, tức là làm theo ca kiểm thử cần thực hiện.
3	Chạy testScript	Vận hành và giám sát hoạt động kiểm tra phần mềm bằng testScript.
4	Đánh giá kết quả	Kiểm tra kết quả sau khi thực hiện kiểm thử tự động. Bổ sung, chỉnh sửa những sai sót nếu cần.

5.6.2. Lợi ích và hạn chế của kiểm thử tự động

Kiểm thử tự động có một số ưu điểm:

- Không cần sự can thiệp của kiểm thử viên khi tiến hành kiểm thử.
- Giảm chi phí khi thực hiện số lượng lớn ca kiểm thử hoặc lặp lại ca kiểm thử nhiều lần.
- Giả lập tình huống khó có thể thực hiện bằng tay.

Bên cạnh đó cũng có những khó khăn cơ bản sau:

- Cần chi phí để tạo và bảo trì các đặc tả kiểm thử (script).
- Đòi hỏi kỹ thuật. Cần phải có kỹ năng sử dụng công cụ và tạo script.
- Không áp dụng được trong việc tìm lỗi mới của phần mềm.

5.6.3. Phân loại công cụ kiểm thử tự động

Chi phí kiểm thử phần mềm thường chiếm tới 40% tổng các nỗ lực dành cho một dự án phát triển phần mềm. Vì thế, các nhà nghiên cứu và người thực hành đã cố gắng phát triển một số thể hệ các công cụ kiểm thử tự động khác nhau để giúp giảm thiểu các chi phí này. Chúng bao gồm:

- *Bộ phân tích tĩnh.* Các hệ thống phân tích chương trình hỗ trợ cho "việc chứng minh" các lý lẽ tĩnh để phát hiện những vấn đề yếu kém về cấu trúc và định dạng của chương trình.
- *Bộ kiểm toán mã.* Những bộ lọc chuyên dụng dùng để kiểm tra chất lượng chương trình để đảm bảo rằng, nó đáp ứng các chuẩn mã hóa tối thiểu.
- *Bộ xử lý khẳng định.* Những hệ thống tiền xử lý/hậu xử lý này được sử dụng để cho biết liệu những mệnh đề do người lập trình tạo ra, được gọi là các khẳng định về hành vi của chương trình có thực sự đáp ứng được yêu cầu khi thực hiện chương trình hay không.
- *Bộ sinh tệp kiểm thử.* Những bộ xử lý sinh ra và điền các giá trị đã xác định vào các tệp để chương trình đọc vào khi kiểm thử.
- *Bộ sinh dữ liệu kiểm thử.* Những hệ thống phân tích hỗ trợ cho người dùng trong việc lựa chọn dữ liệu kiểm thử làm cho chương trình hành xử theo một cách cụ thể.
- *Bộ kiểm chứng kiểm thử.* Chúng đo mức bao quát của việc kiểm thử bên trong, thường liên quan tới cấu trúc điều khiển của sự vật kiểm thử và báo cáo về giá trị bao quát cho chuyên gia đảm bảo chất lượng.
- *Dụng cụ kiểm thử.* Lớp các công cụ này hỗ trợ cho việc thực thi các hoạt động kiểm thử như một người trợ giúp để: (1) xác lập một chương trình ứng cử viên trong môi trường kiểm thử; (2) nạp dữ liệu vào và (3) mô phỏng các cường thay cho hành vi của các mô đun phụ.

- *Bộ so sánh cái ra.* Nó so sánh một tập cái ra từ một chương trình này với một tập cái ra khác (đã được lưu giữ trước) để xác định sự khác biệt giữa chúng.
- *Hệ thống thực hiện ký hiệu.* Công cụ này thực hiện việc kiểm thử chương trình bằng cách dùng cái vào đại số, thay vì giá trị dữ liệu số. Phần mềm được kiểm thử với các lớp dữ liệu, thay vì chỉ là một trường hợp kiểm thử cụ thể. Cái ra là đại số và có thể được so sánh với kết quả trông đợi cũng được xác định dưới dạng đại số.
- *Bộ mô phỏng môi trường.* Nó là một hệ thống dựa trên máy tính giúp người kiểm thử mô hình hóa môi trường bên ngoài của hệ thời gian thực để mô phỏng các điều kiện vận hành thực một cách động.
- *Bộ phân tích luồng dữ liệu.* Công cụ này theo dõi dấu vết luồng dữ liệu đi qua hệ thống (tương tự như bộ phân tích đường đi) và cố gắng tìm ra những tham chiếu dữ liệu không xác định, đặt chỉ số sai và các lỗi khác có liên quan tới dữ liệu.

Một số công cụ kiểm thử tự động đã biết như:

- Cactus (<http://jakarSE.apache.org/cactus>).
- ESC/Java (<http://www.research.compaq.com/SRC/esc/Esc.html>).
- FlawFinder (<http://www.dwheeler.com/flawfinder/>).
- HttpUnit (<http://www.httpunit.org>).
- JTest, Junit (<http://www.junit.org>).
- Nunit, QuickTest – Professional (<http://www.mercury.com>), Rational Rôbot, Rational Performance tester (RPT), SilkTest, WinRunner.

Và các công cụ trợ giúp kiểm thử an ninh như:

- ITS4 (<http://www.cigital.com/its4/>).
- MOPS (<http://www.cs.berkeley.edu/~daw/mops/>).
- RATS (http://www.securesoftware.com/download_form_rats.htm).
- Splint (<http://splint.org/>),...

Việc dùng công cụ để tự động hóa kiểm thử phần mềm đang phát triển, và có thể sẽ phát triển rất nhanh trong thập kỷ tới. Các công cụ kiểm thử tạo ra những thay đổi lớn trong cách chúng ta kiểm thử phần mềm, đem lại hiệu

quả kinh tế cũng như cái tiến độ tin cậy của các hệ thống dựa trên máy tính một cách đáng kể.

Tổng kết

- ◆ *Xác minh và thẩm định (V&V) là các hoạt động thuộc đảm bảo chất lượng phần mềm. Xác minh nhằm kiểm tra tính chính xác của các sản phẩm phần mềm so với đặc tả, còn thẩm định nhằm kiểm tra khả năng đáp ứng của phần mềm đối với yêu cầu người dùng. Xác minh và thẩm định được tiến hành xuyên suốt và song song với quá trình phát triển phần mềm.*
- ◆ *V&V gồm các hoạt động: rà soát kỹ thuật, kiểm toán cấu hình và kiểm toán chất lượng, giám sát hiệu năng, nghiên cứu khả thi và mô phỏng, rà soát tài liệu, rà soát cơ sở dữ liệu, phân tích thuật toán, kiểm thử mã nguồn, kiểm thử chất lượng và kiểm thử cài đặt. Có hai loại V&V là: V&V tĩnh và V&V động. V&V động là các tiến trình kiểm thử.*
- ◆ *Kiểm thử là vận hành chương trình để phát hiện lỗi. Tiến trình kiểm thử bao gồm: lập kế hoạch, thiết kế các ca kiểm thử, chuẩn bị và tiến hành kiểm thử, thu thập và đánh giá kết quả. Mục tiêu của kiểm thử là lập các ca kiểm thử sao cho có khả năng phát hiện lỗi cao với chi phí ít nhất. Để đạt được mục tiêu này, nhiều chiến lược và phương pháp đã được nghiên cứu, đề xuất. Hai phương pháp phổ biến là phương pháp hộp trắng và phương pháp hộp đen.*
- ◆ *Phương pháp hộp trắng có đối tượng là mã nguồn và dựa trên cấu trúc bên trong của chương trình. Năm kỹ thuật dùng ở đây là: đồ thị luồng, ma trận kiểm thử, kiểm thử điều kiện, kiểm thử vòng lặp, kiểm thử luồng dữ liệu.*
- ◆ *Phương pháp hộp đen có đối tượng là các mô đun tạo thành chức năng của phần mềm. Nó xem phần mềm như một "hộp đen". Thông qua giao diện và dữ liệu vào/ra để nhận biết xem chương trình thực hiện chức năng có phù hợp với sự mong đợi hay không. Ba kỹ thuật dùng kiểm thử hộp đen là: phân hoạch tương đương, phân tích giá trị biên và đồ thị nhân quả.*

- ◆ Chiến lược kiểm thử tổng thể là từ mức thấp đến mức cao, đó là các mức: *kiểm thử đơn vị, kiểm thử tích hợp, kiểm thử hệ thống, kiểm thử chấp nhận*.
- ◆ *Kiểm thử đơn vị* nhằm vào kiểm tra đơn vị thiết kế nhỏ nhất – một môđun đơn vị phần mềm. Phương pháp sử dụng là *hộp trắng*. Các lỗi cần kiểm tra là: *lỗi giao diện, lỗi vào/ra, lỗi cấu trúc và sử dụng dữ liệu cục bộ, lỗi xử lý, lỗi điều kiện logic, lỗi tiềm ẩn khác*.
- ◆ *Kiểm thử tích hợp* có đối tượng là tập các môđun tạo thành chức năng của phần mềm. Phương pháp sử dụng là *hộp trắng* và *hộp đen*. Ở đây có thể triển khai các chiến lược tích hợp: *tích hợp từng bước* (từ dưới lên, từ trên xuống, kết hợp cả hai) hay *tích hợp đồng thời* (vụ nổ lớn). Ngoài ra còn sử dụng *kiểm thử quay lui* mỗi lần thực hiện tích hợp thêm môđun mới.
- ◆ *Kiểm thử hệ thống* có đối tượng là toàn bộ hệ thống phần mềm. Phương pháp sử dụng là *hộp đen* và các *phương pháp khác* (như *mô phỏng*). Mục tiêu ở đây là kiểm tra và đánh giá các thuộc tính của toàn hệ thống phần mềm. Các kiểm thử tiến hành bao gồm: *kiểm thử hiệu năng, kiểm thử phục hồi, kiểm thử áp lực* và *kiểm thử an ninh*. Ở đây cũng có thể sử dụng *kiểm thử so sánh*.
- ◆ *Kiểm thử chấp nhận* được người dùng thực hiện. Mục đích của kiểm thử này là khách hàng đánh giá xem phần mềm có thỏa mãn các yêu cầu của họ không và chấp nhận sản phẩm. Kiểm thử này có thể qua hai bước gọi là *kiểm thử Alpha* và *kiểm thử Beta*. Kiểm thử Alpha thực hiện trong môi trường quản lý của nhà phát triển. Kiểm thử Beta thực hiện trong môi trường thực nơi người dùng.
- ◆ *Kiểm thử hệ thống phần mềm phát triển theo hướng đối tượng* có một số khác biệt do đặc trưng của các hệ thống đối tượng xác định. Có ba mức: kiểm thử đơn vị, tích hợp và hệ thống. Ngoài phương pháp chung có thể vận dụng còn cần thêm những kỹ thuật mới để thiết kế ca sử dụng.
- ◆ *Gỡ lỗi* không phải là kiểm thử, nhưng khi gỡ lỗi có thể phải tiến hành kiểm thử. Đó là công việc khó khăn và có ba phương cách để thực hiện gỡ lỗi. Mỗi phương cách có những ưu, nhược điểm của nó.

- ♦ *Kiểm thử tự động* được thực hiện bằng cách sử dụng các công cụ kiểm thử. Công cụ kiểm thử có nhiều loại khác nhau, trợ giúp cho nhiều hoạt động kiểm thử khác nhau. Nó giúp giảm thiểu các chi phí kiểm thử, tăng hiệu quả và cũng chỉ áp dụng trong những phạm vi và điều kiện kiểm thử nhất định.

Câu hỏi ôn tập

1. Thẩm định và xác minh (V&V) là gì? Tầm quan trọng của V&V.
2. Có những loại V&V nào? Mô tả nội dung mỗi loại.
3. Vẽ mô hình chữ V, trình bày mối quan hệ giữa V&V và tiến trình phát triển phần mềm.
4. Nêu các bước của tiến trình kiểm thử.
5. Trình bày nội dung của một kế hoạch kiểm thử. Của một ca kiểm thử.
6. Phần mềm còn lỗi hay không khi thực hiện hết các ca kiểm thử được thiết kế. Khi nào thì dừng kiểm thử?
7. Nêu các phương pháp kiểm thử thường dùng.
8. Trình bày phương pháp kiểm thử hộp trắng: cơ sở của phương pháp; các yêu cầu cần phải kiểm tra; các kỹ thuật được sử dụng.
9. Trình bày nội dung các kỹ thuật: đồ thị luồng; ma trận kiểm thử; kiểm thử điều kiện; kiểm thử vòng lặp; kiểm thử luồng dữ liệu.
10. Trình bày phương pháp kiểm thử hộp đen: cơ sở để lập các ca sử dụng; mục tiêu đặt ra; các kỹ thuật được sử dụng; ưu và nhược điểm của hộp đen.
11. Trình bày nội dung các kỹ thuật: phân hoạch tương đương; phân tích giá trị biên; đồ thị nhân quả.
12. Có những loại hình (mức) kiểm thử nào? Chúng được tiến hành theo trình tự nào?
13. Kiểm thử đơn vị có đối tượng là gì? Ai thực hiện. Các phương pháp và kỹ thuật nào được sử dụng? Kiểm tra những loại sai nào?
14. Các chiến lược nào dùng trong kiểm thử tích hợp? Ưu điểm và hạn chế mỗi loại.

15. Những kiểm thử nào dùng trong kiểm thử tích hợp? Tại sao cần nó?
16. Giải thích khái niệm *cuồng* và *bộ lái*? Nó được sử dụng ở đâu? Vì sao?
17. Kiểm thử hệ thống nhằm kiểm tra cái gì? Ai thực hiện? Các phương pháp.
18. Trình bày các kiểm thử được thực hiện trong kiểm thử hệ thống.
19. Kiểm thử chấp nhận là gì? Có những kiểm thử gì được thực hiện ở đây? Phân biệt sự khác nhau của chúng.
20. Gỡ lỗi được tiến hành khi nào? Trình bày tiến trình gỡ lỗi. Có mấy phương cách gỡ lỗi? Nội dung và ưu nhược điểm của mỗi cách.
21. Kiểm thử tự động là gì? Trình bày tiến trình kiểm thử tự động. Kiểm thử tự động có lợi khi nào? Có những loại công cụ kiểm thử tự động nào?
22. Hệ thống phần mềm hướng đối tượng có những đặc trưng gì? Đó đó khi kiểm thử cần nhắm đến các lỗi liên quan đến các yếu tố nào? Có các mức và các kỹ thuật kiểm thử nào được áp dụng?

Chương 6

QUẢN LÝ DỰ ÁN PHẦN MỀM

Thất bại của nhiều dự án phần mềm lớn trong những năm 1960 và đầu năm 1970 là một chỉ số về những khó khăn của quản lý các dự án phần mềm: *phần mềm chuyển giao trễ hạn, không tin cậy, chi phí vượt nhiều lần dự toán, sản phẩm thể hiện sự nghèo nàn về chức năng*. Theo số liệu của nhóm Standish⁽¹⁾ cho thấy:

- 16% dự án hoàn thành đúng lịch, trong phạm vi ngân sách, đủ tính năng.
- 31% dự án bị ngừng khi đang phát triển.
- 53% dự án đã hoàn thành, có thể hoạt động nhưng vượt quá ngân sách, quá thời gian dự kiến và chỉ cung cấp một số tính năng đề ra.

Năm 1995, các công ty 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 SeaGames 22 của VN" dự kiến kinh phí là 15 tỷ VND, nhưng mới đến tháng 6/2003 đã chi tới 90 tỷ VND⁽²⁾. Những dữ liệu trên cho thấy rằng, các dự án phần mềm cần được quản lý để tránh những thất bại, mặc dù quản lý tốt chưa đảm bảo dự án thành công, nhưng quản lý tồi luôn dẫn đến sự thất bại của dự án.

Người quản lý dự án phần mềm phải làm các công việc như một người quản lý dự án bất kỳ. Tuy nhiên, kỹ nghệ phần mềm có nhiều điểm khác với các loại kỹ nghệ khác, vì thế việc quản lý các dự án phần mềm đặc biệt khó khăn:

⁽¹⁾ Standish Group. *The CHAOS Report*, 1995 http://www.pm2go.com/sample_research/chaos_1994.PHP

⁽²⁾ *Tạp chí PC World* 7/2003

- *Phần mềm là sản phẩm không nhìn thấy được*: Người quản lý dự án không thể nhìn thấy, sờ mó được phần mềm mà chỉ nhận biết nó qua tiến trình dự án diễn ra: những công việc nào đã và đang thực hiện, những tài liệu cần và đã xét duyệt.
- *Không có một tiến trình phần mềm chuẩn*: Ta không thể có được một sự hiểu biết đầy đủ, rõ ràng về mối quan hệ giữa tiến trình phần mềm và sản phẩm. Kinh nghiệm qua một thời gian dài cho thấy, những tiến trình đó cần phải kiểm tra và xét duyệt. Mặc dù sự hiểu biết của chúng ta về tiến trình phần mềm đã được nâng lên đáng kể trong vài năm gần đây, nhưng vẫn không thể dự kiến trước mọi bất định. Mỗi tiến trình cụ thể luôn nảy sinh những vấn đề, những rủi ro riêng.
- *Các dự án phần mềm lớn thường là dự án mới*: Các dự án phần mềm lớn luôn khác với các dự án trước đó. Bởi vậy, người quản lý khó sử dụng kinh nghiệm đã có để làm giảm bất định khi lập kế hoạch. Họ rất khó khăn để có thể dự kiến hết các vấn đề nảy sinh. Hơn nữa, sự thay đổi nhanh chóng trong kỹ thuật máy tính và truyền thông làm cho nhiều vấn đề vượt ra khỏi tầm kiến thức và kinh nghiệm của người quản lý tích lũy được, khả năng đáp ứng cho các dự án mới là rất hạn chế. Phần lớn các hệ thống phần mềm luôn là mới và đòi hỏi đổi mới kỹ thuật. Vì vậy, nhiều vấn đề trở nên nan giải, đặc biệt trong hoạt động lập kế hoạch và lập lịch.
- *Sự thay đổi là cố hữu đối với các dự án phần mềm*: Sự thay đổi về yêu cầu khách hàng từ nhu cầu nghiệp vụ của họ, sự thay đổi về công nghệ do công nghệ thông tin tiến bộ nhanh chóng và thường xuyên, sự thay đổi về môi trường đích mà phần mềm được xây dựng sẽ hoạt động,... Tất cả luôn tác động lên quá trình phát triển mà người quản lý dự án buộc phải đối mặt.

6.1. Dự án và quản lý dự án

6.1.1. Khái niệm về dự án

Dự án là một tập hợp các công việc được một số người thực hiện nhằm đạt đến một kết quả dự kiến là *một sản phẩm xác định, trong thời gian nhất*

định, với kinh phí dự kiến. Như vậy, khái niệm dự án liên quan đến ba yếu tố sau đây:

- Kết quả dự án là *một sản phẩm hay dịch vụ hoàn toàn xác định.* Vì vậy, sản phẩm cần được mô tả về các đặc trưng: tính năng tác dụng, giá trị sử dụng, hiệu năng mang lại, tiêu chí đánh giá,...
- *Do một số người thực hiện.* Tất nhiên cũng có dự án do một người thực hiện nhưng là hạn hữu. Vì có nhiều người thực hiện, nên trước hết những người thực hiện phải có những kỹ năng và nghiệp vụ phù hợp với yêu cầu tạo ra sản phẩm. Sau nữa cần có cách tổ chức quản lý làm việc sao cho sử dụng tốt năng lực của mỗi người và các tài nguyên cần thiết để tạo ra sản phẩm một cách hiệu quả.
- *Có những giới hạn đặt lên các hoạt động dự án,* đó là thời gian (bao gồm thời gian bắt đầu, thời gian kết thúc và một số mốc khác cần để kiểm soát) và kinh phí cần thiết được sử dụng.

Các hoạt động dự án có các đặc trưng sau:

- 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 – dự án thành công, ngược lại – dự án là thất bại.

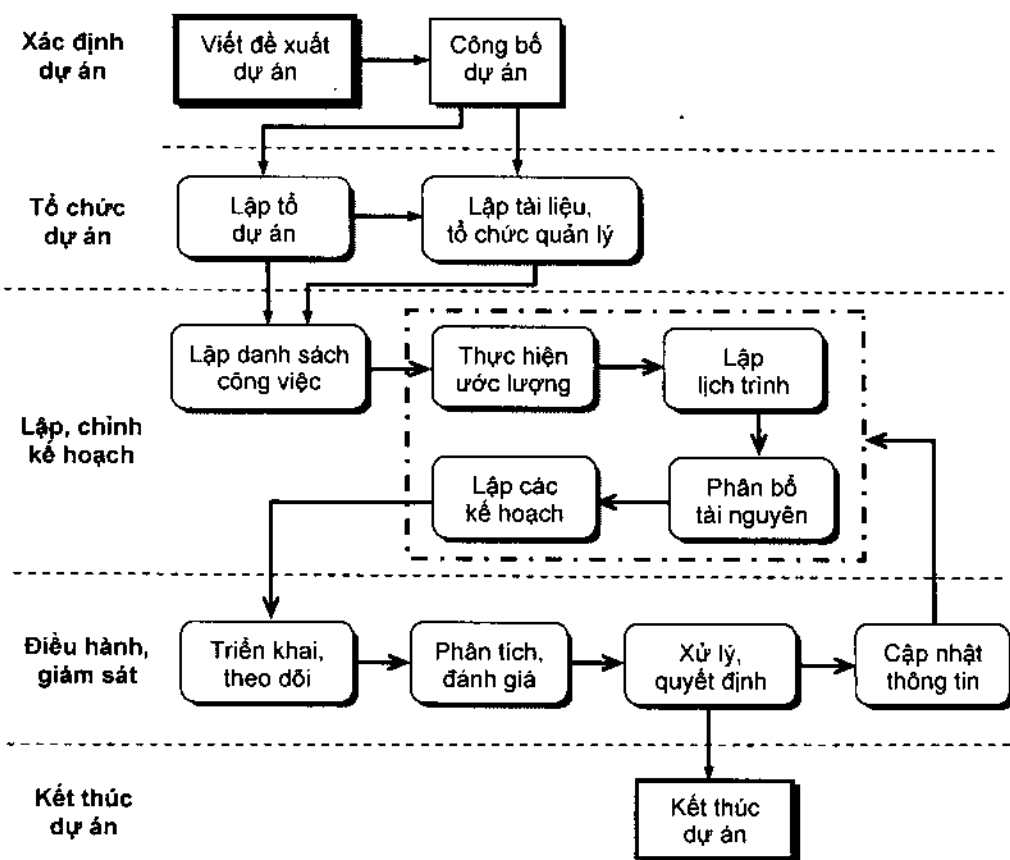
Các đặc trưng này đặt ra nhiều vấn đề cần được nghiên cứu và giải quyết trong hoạt động triển khai dự án. Mặt khác, một dự án thất bại thường do một số nguyên nhân sau:

- *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 sự kiện bất ngờ thường từ môi trường: như công nghệ thay đổi, người cung cấp nguồn lực sai hẹn,...

Ba lý do sau cùng đều liên quan với 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.

6.1.2. Quản lý dự án

Cho đến nay, chưa có một định nghĩa nào về quản lý dự án là đầy đủ và chính xác. Quản lý dự án được xem là việc vận dụng các kiến thức, phương pháp, công cụ và kỹ thuật để triển khai các hoạt động của dự án, nhằm tạo ra sản phẩm theo đúng các yêu cầu được đặt ra. 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 cho ở hình 6.1.



Hình 6.1. Bức tranh tổng thể tiến trình triển khai một dự án

Nhìn một cách bao quát, các hoạt động chính của quản lý dự án gồm:

– Khởi thảo, xác định dự án:

- + Viết bản đề xuất.
- + Thiết lập quan hệ với khách hàng.
- + Xây dựng dự án sơ bộ.
- + Lập tổ dự án.
- + Thiết lập thủ tục quản lý.
- + Thiết lập môi trường quản lý và tài liệu dự án.

– Lập kế hoạch dự án.

– Triển khai thực hiện, giám sát.

– Kết thúc dự án.

Viết bản đề xuất: Giai đoạn đầu tiên của một dự án phần mềm là viết đề xuất thực hiện dự án. Bản đề xuất cần mô tả các đối tượng của dự án, phạm vi dự án, nguồn lực và cách thức triển khai dự án. Thường thì nội dung còn phải bao gồm các ước lượng sơ bộ chi phí và lịch trình thực hiện dự án với các mốc chính. Có thể phải trình bày rõ vì sao hợp đồng dự án được trình ra cho một tổ chức hay một đội cụ thể.

Viết bản đề xuất là một công việc quan trọng. Nó phụ thuộc vào việc có đủ cơ sở hay không để dự án được chấp nhận và hợp đồng đưa ra được ký kết. Không có một hướng dẫn cụ thể và đầy đủ nào cho nhiệm vụ này. Công việc này đòi hỏi những kỹ năng thu nhận được từ kinh nghiệm của người viết.

Những hoạt động khác trong giai đoạn xác định dự án đơn thuần là những hoạt động quản lý có tính nghiệp vụ.

Quản lý dự án chủ yếu liên quan đến ba hoạt động cuối cùng.

6.2. Các hoạt động quản lý dự án

Sự biến động trong công việc quản lý dự án phụ thuộc rất lớn vào tổ chức, vào sản phẩm phần mềm được phát triển, vào hiện trạng công nghệ được sử dụng. Tuy nhiên, những hoạt động phổ biến cho hoạt động quản lý dự án thường bao gồm:

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

Lựa chọn, đánh giá và tổ chức sử dụng các nguồn lực: Người quản lý dự án cũng phải lựa chọn những người làm việc cho dự án của mình, bố trí họ vào những công việc thích hợp ở mỗi thời điểm. Lý tưởng nhất là có được những người làm việc với kỹ năng cao, kinh nghiệm nhiều. Nhưng trong đa số trường hợp, người quản lý phải tổ chức được các đội ngũ làm việc của mình trong những điều kiện cho phép. Lý do có sự kiện đó là:

- Ngân sách dự án không cho phép chọn được những người có kỹ năng cao.
- Đội ngũ người tham gia dự án thích hợp không phải lúc nào cũng sẵn có.
- Bản thân tổ chức phát triển dự án cũng muốn đào tạo một đội ngũ cán bộ có tay nghề của mình.

Lập kế hoạch dự án: Công việc này liên quan đến việc xác định các nhiệm vụ, các cột mốc và sản phẩm xuất ra của dự án. Một kế hoạch cần được lập ra để hướng dẫn quá trình phát triển hướng đến các mục tiêu dự án. Để có một kế hoạch tốt cần phải ước lượng các chi phí liên quan đến các hoạt động dự án như các nguồn lực về người, về thời gian thực hiện, về phần cứng và phần mềm cần chia sẻ, và ngân sách cho mọi hoạt động của dự án. Sau đó phải lập lịch.

Triển khai và kiểm soát dự án: Đây là một hoạt động liên tục của dự án. Người quản lý phải theo dõi tiến trình thực hiện dự án, so sánh chi phí, lịch trình thực tế với các dự kiến kế hoạch để điều hành và có giải pháp thích hợp. Mặc dù phần lớn các tổ chức đã có những cơ chế chính thức cho hoạt động này, nhưng một người quản lý có kỹ năng vẫn cần hình thành lên một

bức tranh rõ ràng về những gì đang diễn ra bằng cách trao đổi thường xuyên một cách không hình thức với đội ngũ quản lý. Việc thúc đẩy dự án thường phải phát hiện những rủi ro tiềm tàng có thể xảy ra và áp dụng hay đưa ra các giải pháp thích hợp cho tình huống tương ứng. Hằng ngày, người quản lý phải trao đổi với các nhân viên dự án nhằm nắm được những vấn đề cụ thể để giải quyết hay thu thập các dữ liệu cần thiết cho các báo cáo theo lịch trình. Trong quá trình dự án, thường có những cuộc họp xem xét về quản lý (tiến trình chung) hay rà soát kỹ thuật chính thức (đối với phát triển kỹ thuật của dự án). Tổ chức đội làm việc hiệu quả cũng là vấn đề cực kỳ quan trọng trong giai đoạn này.

Đối với những dự án lớn, thời gian phát triển dài (một vài năm), các đối tượng của tổ chức không tránh khỏi có những thay đổi, nó đòi hỏi dự án phải có những thay đổi thích hợp hay quyết định dừng lại.

Viết báo cáo và trình bày: Người quản lý dự án thường có trách nhiệm báo cáo cho cả khách hàng và tổ chức chủ hợp đồng về dự án. Vì vậy, họ cần viết báo cáo dự án sao cho rõ ràng, dễ hiểu, khái quát được các thông tin từ các báo cáo dự án chi tiết. Họ cũng phải trình bày các thông tin đó trong các cuộc họp xét duyệt hay trình bày cả bằng lời và văn bản cho các đối tượng liên quan.

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.

6.3. Lập kế hoạch dự án

Việc quản lý hiệu quả một dự án phần mềm phụ thuộc trước hết vào việc lập kế hoạch tiến độ của dự án. Người quản lý dự án phải thấy trước mọi công việc cần làm, những vấn đề có thể phát sinh và chuẩn bị những giải pháp thích hợp cho những vấn đề đó. Một kế hoạch nên được sử dụng như một công cụ đắc lực cho quản lý dự án. Cấu trúc cho một kế hoạch phát triển phần mềm được mô tả trong mục 6.3.2. Ngoài ra, người quản lý còn cần lập các kế hoạch cho những phần khác của dự án. Các kế hoạch được mô tả trong bảng 6.1.

Bảng 6.1. Các kiểu kế hoạch cần cho dự án

Kế hoạch	Mô tả
Kế hoạch về chất lượng	Mô tả các thủ tục và chuẩn về chất lượng được sử dụng trong dự án.
Kế hoạch thẩm định	Mô tả cách thức, nguồn lực và tiến độ sử dụng cho việc thẩm định hệ thống.
Kế hoạch quản lý cấu hình	Mô tả các thủ tục và cấu trúc quản lý cấu hình được sử dụng.
Kế hoạch bảo trì	Dự đoán các yêu cầu bảo trì của hệ thống, chi phí bảo trì và các cố gắng cần thiết.
Kế hoạch phát triển nhân lực	Mô tả cách phát triển các kỹ năng và kinh nghiệm của các thành viên của dự án.

6.3.1. Tiến trình lập kế hoạch quản lý dự án

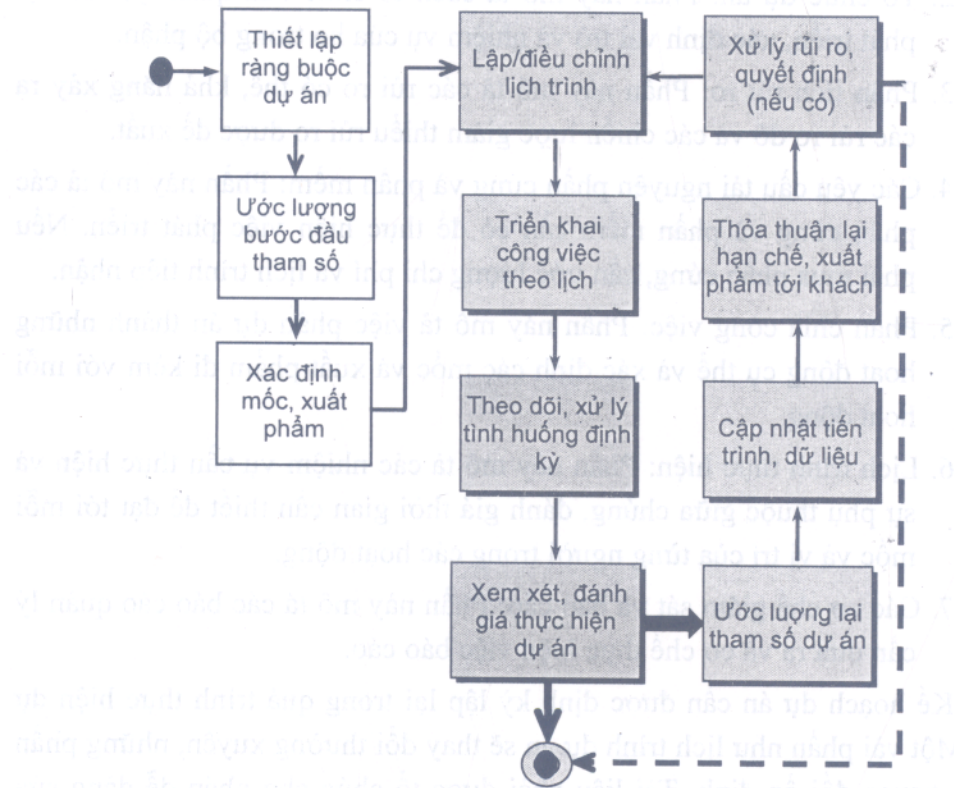
Biểu đồ hình 6.2 mô tả tiến trình lập kế hoạch dự án cho việc phát triển phần mềm. Nó cho thấy, việc lập kế hoạch là một quá trình lặp và chi kết thúc khi chính dự án đó hoàn thành. Mỗi khi đã có các thông tin cụ thể về dự án, cần xem xét lại dự án một cách định kỳ. Mục tiêu tổng thể của doanh nghiệp luôn là nhân tố quan trọng cần được xét đến khi hình thành kế hoạch dự án.

Quá trình lập kế hoạch bắt đầu bằng việc đánh giá những ràng buộc (thời hạn giao sản phẩm được yêu cầu, số lượng nhân viên, tổng ngân sách,...) ảnh hưởng đến dự án. Nó được vạch ra phù hợp với các tham số ước lượng của dự án như cấu trúc, kích cỡ và việc phân bổ các chức năng của dự án.

Các mốc tiến độ và chuyển giao sản phẩm được xác định. Tiến trình sau đó sẽ đi vào vòng lặp. Một lịch trình cho dự án được lập ra và các hoạt động được xác định trong lịch được khởi động hoặc được tiếp tục triển khai. Sau một khoảng thời gian (từ 2 – 3 tuần), tiến độ được xem xét và ghi chú các điểm không nhất quán. Do các tham số của dự án đánh giá ban đầu chỉ là dự kiến, nên kế hoạch cần phải được bổ sung và điều chỉnh thường xuyên.

Khi kế hoạch bị trễ, nhà quản lý phải đàm phán lại về ràng buộc, về thời hạn chuyển giao sản phẩm với khách hàng. Nếu việc đàm phán không thành công và lịch trình không thể đáp ứng thì cần phải tổ chức một buổi xem xét

lại về mặt kỹ thuật của dự án. Mục tiêu của buổi họp này là tìm ra những cách tiếp cận khác để phát triển dự án đúng yêu cầu. Các vấn đề gần như chỉ xuất hiện trong quá trình thực hiện dự án. Do đó, cần phải tính đến cả những yếu tố bất ngờ để không phải đàm phán lại với khách hàng ở các vòng lặp của dự án.



Hình 6.2. Tiến trình lập kế hoạch và thực hiện dự án

6.3.2. Kế hoạch quản lý dự án

Kế hoạch dự án thiết lập 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, vào tổ chức thực hiện. Tuy nhiên, hầu hết các kế hoạch đều bao gồm những phần sau:

1. Giới thiệu: Phần này mô tả các mục đích của dự án và đưa ra các ràng buộc (như ngân sách, thời gian,...) tác động lên dự án.
2. Tổ chức dự án: Phần này mô tả cách tổ chức ban quản lý, các đội phát triển, xác định vai trò và nhiệm vụ của họ trong bộ phận.
3. Phân tích rủi ro: Phần này mô tả các rủi ro có thể, khả năng xảy ra các rủi ro đó và các chiến lược giảm thiểu rủi ro được đề xuất.
4. Các yêu cầu tài nguyên phần cứng và phần mềm: Phần này mô tả các phần cứng và phần mềm cần có để thực hiện việc phát triển. Nếu phải mua phần cứng, cần ước lượng chi phí và lịch trình tiếp nhận.
5. Phân chia công việc: Phần này mô tả việc phân dự án thành những hoạt động cụ thể và xác định các mốc và xuất phẩm đi kèm với mỗi hoạt động.
6. Lịch trình thực hiện: Phần này mô tả các nhiệm vụ cần thực hiện và sự phụ thuộc giữa chúng, đánh giá thời gian cần thiết để đạt tới mỗi mốc và vị trí của từng người trong các hoạt động.
7. Các cơ chế giám sát và báo cáo: Phần này mô tả các báo cáo quản lý cần đưa ra và cơ chế thực hiện việc báo cáo.

Kế hoạch dự án cần được định kỳ lập lại trong quá trình thực hiện dự án. Một vài 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.

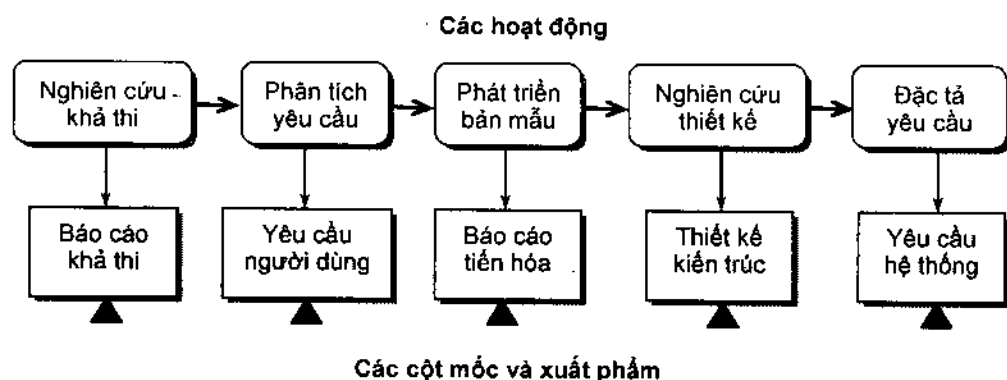
6.3.3. Các mốc và các xuất phẩm

Phần mềm là "vô hình", các thông tin về nó chỉ có thể được cung cấp ở dạng tài liệu mô tả trạng thái của phần mềm đang được phát triển. Không có những thông tin này, sẽ không thể đánh giá tiến độ, chi phí ước lượng và không thể cập nhật được lịch dự án.

Khi lập kế hoạch một dự án, các mốc cần được thiết lập, mỗi mốc là một điểm cuối của một loạt các hoạt động trong tiến trình phần mềm. Tại mỗi mốc, cần có một đầu ra chính thức, rõ ràng và cụ thể như một báo cáo

cho người quản lý, một xuất phẩm chuyên cho khách hàng. Một báo cáo tại mỗi mốc không nhất thiết phải là một tài liệu lớn. Chúng có thể đơn giản chỉ là một báo cáo ngắn về những điều đã đạt được trong mỗi hoạt động. Các mốc cần thể hiện điểm cuối mỗi giai đoạn logic và khác biệt của dự án (hình 6.3).

Một xuất phẩm là kết quả của dự án được chuyển tới khách hàng. Nó thường được tạo ra tại cuối mỗi pha lớn của dự án như pha *đặc tả, thiết kế, ...* Xuất phẩm thường nhận được ở tại các mốc, nhưng ở mỗi mốc không nhất thiết phải có xuất phẩm. Tại một mốc có thể chỉ cho kết quả nội bộ bên trong dự án và được người quản lý sử dụng để kiểm soát dự án. Để thiết lập các mốc, tiến trình phần mềm cần được phân ra thành những hoạt động cơ bản với các đầu ra cụ thể đi kèm (hình 6.3).



Hình 6.3. Các cột mốc trong tiến trình xác định yêu cầu

6.4. Lập lịch dự án

6.4.1. Tiến trình lập lịch

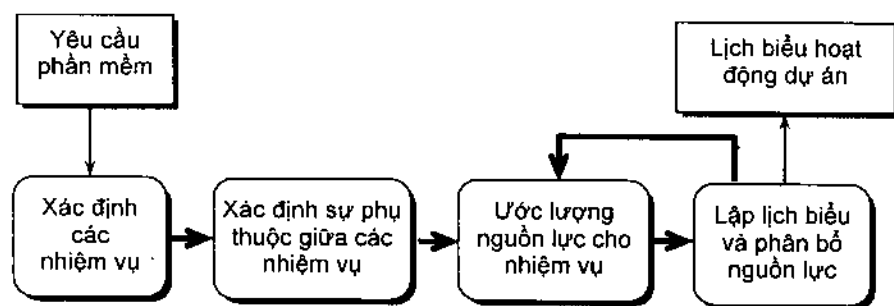
Lập lịch dự án là một công việc đòi hỏi khắt khe và chính xác, một hoạt động đặc trưng của quản lý dự án. Những nhà quản lý phải làm chi tiết các công việc cần thực hiện, ước lượng thời gian và các tài nguyên cần thiết để hoàn thành chúng, tổ chức chúng theo một chuỗi tuần tự liên kết chặt chẽ với nhau theo đúng logic.

Việc xác định lịch trình dự án tốt là một việc rất phức tạp, do các dự án phần mềm khác nhau có thể sử dụng các mô hình tiến trình (mô hình thác

nước, xoắn ốc,...), các phương pháp phân tích và thiết kế khác nhau (cấu trúc hay đối tượng), ngôn ngữ thực thi khác nhau. Nếu dự án thiên về kỹ thuật, các đánh giá ban đầu hầu như là khá tốt, ngay cả khi người quản lý cần phải xem xét đến những tình huống có thể xảy ra. Lúc này, lập lịch phần mềm không khác so với việc lập lịch các kiểu dự án lớn khác. Các dự án phần mềm lớn thường bị trễ vì các vấn đề không thể dự kiến hết được. Do đó, lịch trình cần được cập nhật liên tục khi có thêm các thông tin chi tiết về thực hiện các hoạt động và tiến độ của dự án.

Tiến trình lập kế hoạch lịch cho hoạt động dự án (hình 6.4) bao gồm việc phân chia hoạt động của dự án thành các công việc riêng rẽ và đánh giá thời gian cần thiết để hoàn thành chúng. Người lập lịch dự án cần phải phối hợp sao cho các hoạt động có thể thực hiện song song và tổ chức công việc sao cho có thể sử dụng tối đa các nguồn lực.

Xây dựng bảng công việc chi tiết để hoàn thành dự án là một việc khó khăn: Người quản lý phải nghĩ ra được tất cả các công việc cần làm để hoàn thành dự án. Bảng công việc được xác định tốt là cơ sở để xác định các bước hoàn thành dự án, làm cơ sở cho việc phân bổ các nguồn lực, cho việc xác định trách nhiệm cá nhân và xây dựng lịch biểu thực hiện dự án.



Hình 6.4. Tiến trình lập kế hoạch 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 ước lượng thời gian là 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 phương pháp kinh nghiệm. Trong phương pháp PERT (*Program Evaluation and Review Techniques*), 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 được tính theo công thức:

$$t_{cv} = (MO + 4.ML + MP)/6$$

Các hoạt động của dự án thường diễn ra ít nhất một tuần. Khoảng thời gian lớn nhất cho một hoạt động khoảng từ tám đến mười tuần. Nếu nó diễn ra dài hơn thì nên tiếp tục phân chia hoạt động thành những công việc nhỏ hơn.

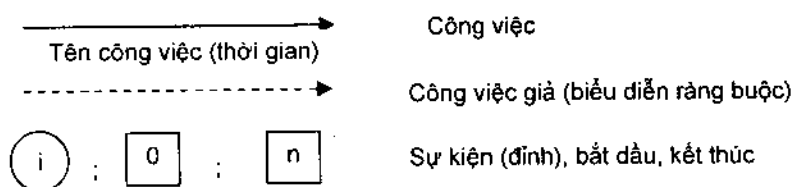
Người quản lý cũng phải đánh giá các tài nguyên cần thiết để hoàn thành mỗi công việc. Tài nguyên có thể là con người, dung lượng đĩa cần thiết trên máy chủ, thời gian cần thiết cho những phần cứng đặc biệt, như bộ giả lập và chi phí đi lại cần thiết cho các nhân viên dự án... Việc dự toán tài nguyên 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 được 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 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). Công cụ trợ giúp việc tạo sinh tự động các biểu đồ, tính toán các tham số, lập báo cáo.

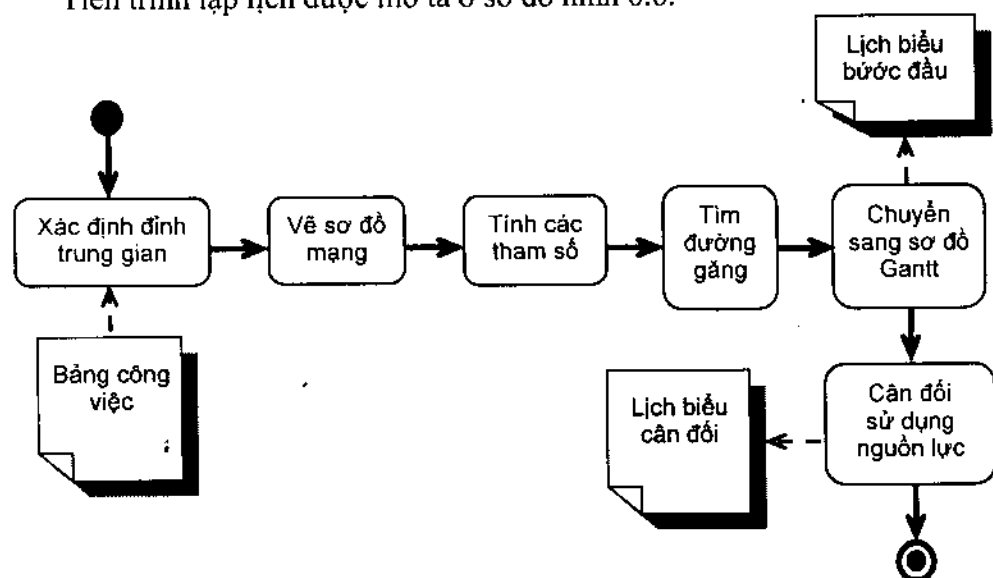
6.4.2. Phương pháp và công cụ lập lịch

Phương pháp đường găng (*CPM – 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 đầu tiên của dự án đi ra và một *đỉnh kết thúc*, có các công việc cuối cùng của dự án đi vào. Trên mỗi cung – công việc có ghi tên công việc và thời gian thực hiện nó (hình 6.5 và hình 6.7).



Hình 6.5. Các ký pháp của sơ đồ mạng

Tiến trình lập lịch được mô tả ở sơ đồ hình 6.6.



Hình 6.6. Tiến trình lập lịch biểu bằng sơ đồ mạng

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

Xuất phát từ bảng liệt kê công việc (bảng 6.2) của dự án, 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 các chữ (công việc) là duy nhất (cấp 2 / cấp 3) trên dòng mà 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 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 ở bảng 6.2.

Bảng 6.2. 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
b	7	–	m	3	i
c	8	–	l	3	i
d	4	–	n	2	k
e	4	a	o	1	l, n
f	3	b	p	2	g , i , h
g	3	c	q	3	g , i , h
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 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 tại 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 đã bị 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 ở bảng 6.2 ta được mạng cho trong hình 6.7a.

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: $t_s(j)$

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

- $t_s(0) = 0$
- đỉnh j ($j = 1, 2, \dots, n$):

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

(mọi cv đi vào đỉnh j)

• Thời gian muộn nhất của một đỉnh $t_m(i)$

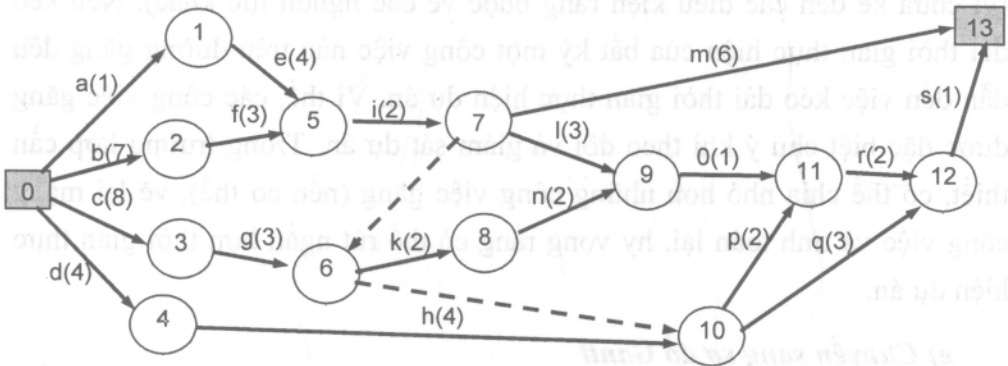
Quá trình tính ngược lần lượt từ đỉnh lớn đến đỉnh nhỏ, bắt đầu từ đỉnh n:

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

– đỉnh i ($i = n - 1, \dots, 0$):

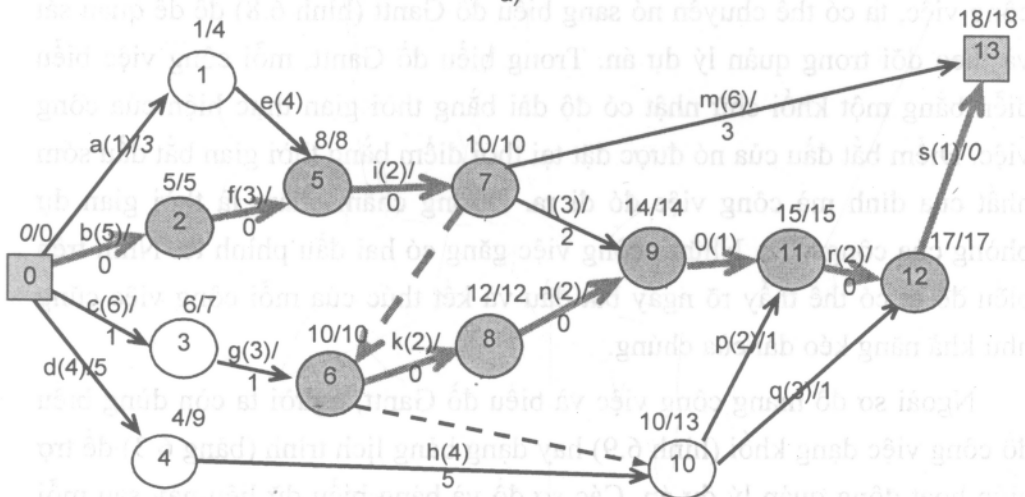
$$t_m(i) = \text{Min} \{ t_m(\text{đỉnh cuối cv}) - t_{cv} \}$$

(mọi cv đi khỏi đỉnh i)



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

a)



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)

Hình 6.7. a) Sơ đồ mạng được vẽ từ bảng công việc (bảng 3)

b) Sơ đồ mạng với các tham số thời gian và đường găng

- Thời gian dự phòng của mỗi công việc: $t_{df}(cv)$

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

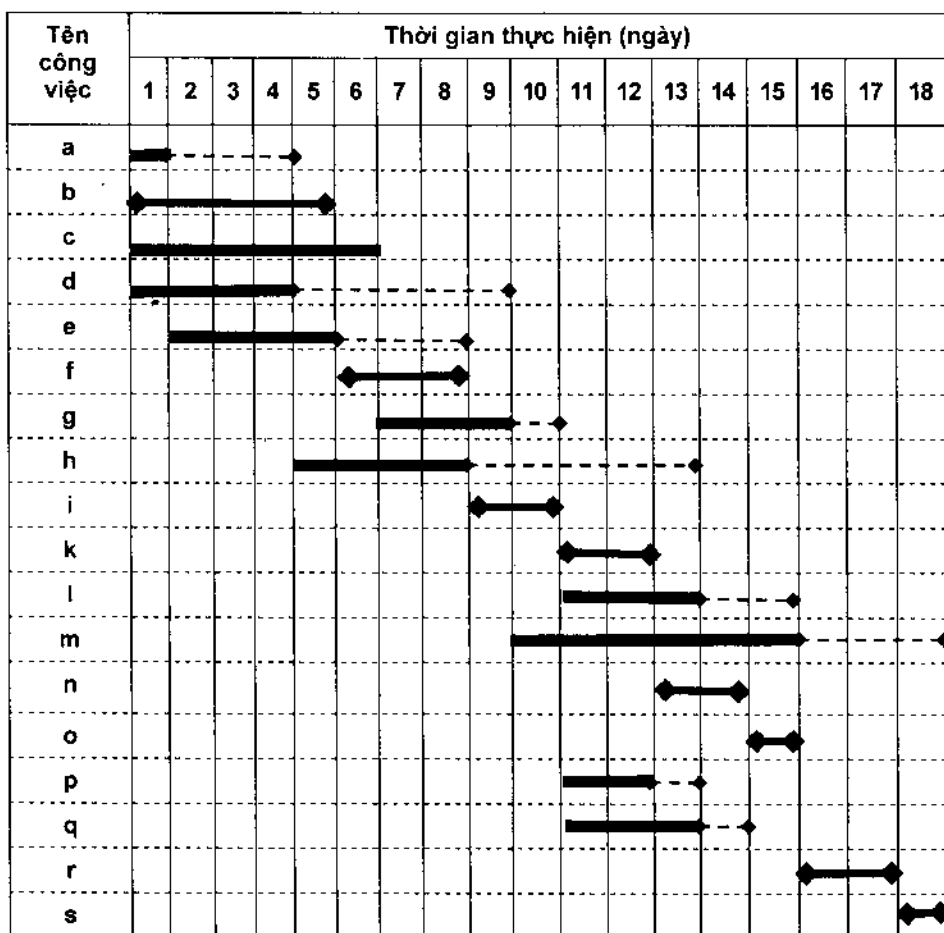
d) Tìm công việc găng và đường găng

Những công việc găng (nét đậm trên hình 6.8) là những công việc có thời gian dự phòng bằng 0 ($t_{df}(cv) = 0$). Khi đó *đường găng* 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 6.7b) 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 găng 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 găng (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.

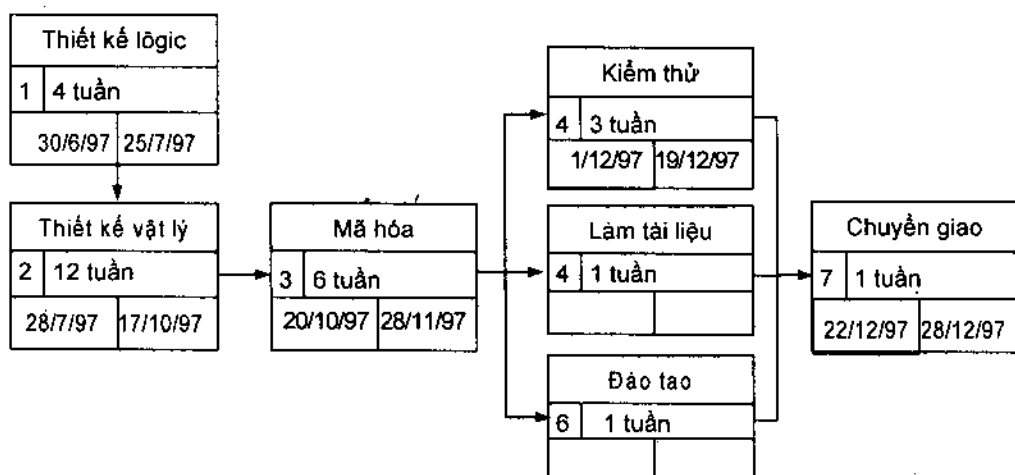
e) Chuyển sang sơ đồ Gantt

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 6.8) 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 của 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 chấm chấm 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 kết thúc của mỗi công việc cũng như khả năng kéo dài của chúng.

Ngoài sơ đồ mạng công việc và biểu đồ Gantt, người ta còn dùng biểu đồ công việc dạng khối (hình 6.9) hay dạng bảng lịch trình (bảng 6.3) 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 6.8. Biểu đồ Gantt lịch trình dự án



Hình 6.9. Biểu đồ khối lịch trình công việc

Bảng 6.3. Bảng lịch trình thực hiện công việc

TT	Tên công việc	Thời gian thực hiện	Ngày bắt đầu	Ngày kết thúc	Người thực hiện
1	Thiết kế logic	4	30/6/97	25/7/97	Trần Văn An
2	Thiết kế vật lý	12	28/7/97	17/10/97	Nguyễn Toàn
3	Mã hoá	6	20/10/97	28/11/97	Hoàng Lan
4	Kiểm thử	3	1/12/97	19/12/97	Vũ Văn Thạch
5	Làm tài liệu	1			Trần Văn Chung
6	Đào tạo	1			Đào Chí Hùng
7	Chuyển giao	1	22/12/97	28/12/97	Nguyễn Trung

6.5. Quản lý rủi ro

Một nhiệm vụ quan trọng của người quản lý dự án là nhận biết các rủi ro có thể ảnh hưởng đến lịch trình của dự án, hoặc chất lượng của phần mềm đang được phát triển, thực hiện các hoạt động nhằm tránh hay giảm thiểu các thiệt hại do các rủi ro này gây ra. Các kết quả phân tích rủi ro được làm thành tài liệu trong kế hoạch dự án cùng với phân tích về hậu quả nếu rủi ro xảy ra. Việc xác định các rủi ro và lập kế hoạch để kiểm soát ảnh hưởng của chúng lên dự án được gọi là *quản lý rủi ro*.

Các loại rủi ro có thể có được phân định như sau:

1. Các rủi ro dự án là rủi ro ảnh hưởng đến lịch trình hoặc tài nguyên dự án.
2. Các rủi ro sản phẩm là các rủi ro ảnh hưởng đến chất lượng hoặc hiệu suất của phần mềm đang được phát triển.
3. Các rủi ro nghiệp vụ là các rủi ro ảnh hưởng đến tổ chức phát triển hoặc thuê phần mềm.

Tất nhiên, không thể có một sự phân loại hoàn toàn loại trừ nhau. Nếu một lập trình viên có kinh nghiệm rời khỏi dự án, đây có thể là rủi ro dự án (vì việc triển khai hệ thống có thể bị trễ), cũng là rủi ro sản phẩm (vì người thay thế có thể không có kinh nghiệm và có thể gây lỗi) và cũng là rủi ro nghiệp vụ (vì kinh nghiệm là thứ không thể mua được cho các công việc trong tương lai). Các rủi ro có thể tác động lên một dự án tùy thuộc vào dự án và môi trường mà phần mềm được phát triển. Tuy vậy, các rủi ro là đa dạng và nhiều khả năng có thể xảy ra (bảng 6.4).

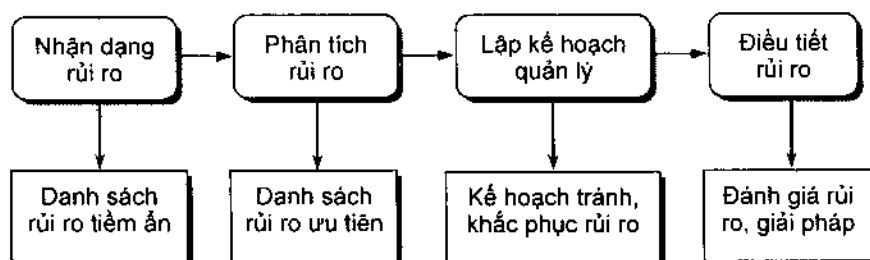
Bảng 6.4. Các rủi ro phần mềm có thể có

Rủi ro	Loại rủi ro	Mô tả
Thay đổi nhân viên	Dự án	Nhân viên có kinh nghiệm rời khỏi dự án trước khi nó hoàn tất.
Thay đổi quản lý	Dự án	Có thay đổi về sự ưu tiên trong quản lý của tổ chức.
Thay đổi yêu cầu	Dự án và sản phẩm	Số lượng các thay đổi trong các yêu cầu lớn hơn dự đoán.
Trễ đặc tả	Dự án và sản phẩm	Các đặc tả giao diện cần thiết không có sẵn theo đúng lịch.
Đánh giá thấp kích thước	Dự án và sản phẩm	Kích cỡ của hệ thống đã ước lượng quá thấp.
Công cụ CASE có hiệu suất kém	Sản phẩm	Các công cụ CASE hỗ trợ dự án không hoạt động như mong đợi.
Thay đổi công nghệ	Kinh doanh	Công nghệ mà hệ thống đang sử dụng bị thay thế bởi công nghệ mới hơn.
Cạnh tranh sản phẩm	Kinh doanh	Một sản phẩm cạnh tranh được đưa ra thị trường trước khi hệ thống hoàn tất.

Quản lý rủi ro rất quan trọng đối với các dự án phần mềm vì tình trạng không rõ ràng cố hữu mà hầu hết các dự án phải đối mặt. Những điều này xuất phát từ các yêu cầu không được xác định chặt chẽ, các khó khăn trong việc đánh giá thời gian và tài nguyên cần thiết để phát triển phần mềm, sự phụ thuộc rất lớn vào các kỹ năng cá nhân, vào sự thay đổi trong các yêu cầu do nhu cầu của khách hàng. Người quản lý dự án cần thấy trước các rủi ro, hiểu các ảnh hưởng của các rủi ro này lên dự án, sản phẩm và kinh doanh để có kế hoạch, bước đi xử lý thích hợp.

Tiến trình quản lý rủi ro bao gồm một số giai đoạn (hình 6.10):

1. Nhận dạng rủi ro: Xác định công việc dự án, sản phẩm và phát hiện các rủi ro kinh doanh có thể.
2. Phân tích rủi ro: Đánh giá khả năng xảy ra và hậu quả của các rủi ro.
3. Lập kế hoạch rủi ro: Lập kế hoạch định vị rủi ro bằng cách tránh nó hoặc giảm thiểu ảnh hưởng của nó đến hệ thống.
4. Giám sát rủi ro: Rủi ro được đánh giá định kỳ và kế hoạch cho việc giảm thiểu rủi ro được xem xét lại mỗi khi có các thông tin cụ thể về rủi ro.



Hình 6.10. Tiến trình quản lý rủi ro

Tiến trình quản lý rủi ro giống như tất cả các kế hoạch dự án khác, là một tiến trình lặp diễn ra trong suốt quá trình thực hiện dự án. Mỗi khi một tập khởi tạo các kế hoạch được đưa ra, tình huống sẽ được giám sát. Khi thông tin về các rủi ro trở nên rõ ràng, chúng có thể được phân tích và phân loại lại, mức độ mới của rủi ro được thiết lập, các giải pháp tránh rủi ro và các kế hoạch cho các trường hợp bất ngờ có thể được thay đổi.

6.5.1. Nhận dạng rủi ro

Nhận dạng rủi ro là bước đầu tiên của quản lý rủi ro. Nó liên quan đến việc phát hiện ra các rủi ro có thể ảnh hưởng đến dự án. Về mặt khái niệm, chúng không được đánh giá hay phân cấp vào thời điểm này; về mặt thực hành, các rủi ro có ảnh hưởng nhỏ hoặc nguy cơ xảy ra thấp thường không được xem xét.

Nhận dạng rủi ro có thể được thực hiện như một tiến trình của đội bằng cách suy nghĩ hoặc đơn giản là dựa trên kinh nghiệm quản lý. Để hỗ trợ tiến trình, ta sử dụng một danh sách các loại rủi ro có thể. Các loại này (bảng 6.4) bao gồm:

1. Các rủi ro công nghệ: Các rủi ro xuất phát từ công nghệ phần cứng hoặc phần mềm đang được sử dụng trong phát triển hệ thống.
2. Các rủi ro con người: Các rủi ro liên quan đến con người trong các nhóm phát triển.
3. Các rủi ro tổ chức: Các rủi ro xuất phát từ môi trường của tổ chức mà phần mềm đang được phát triển.
4. Các rủi ro công cụ: Các rủi ro xuất phát từ các công cụ CASE và các phần mềm hỗ trợ khác được sử dụng để phát triển hệ thống.

5. Các rủi ro yêu cầu: Các rủi ro xuất phát từ những thay đổi trong yêu cầu của khách hàng và tiến trình quản lý các yêu cầu thay đổi.
6. Rủi ro đánh giá: Các rủi ro xuất phát từ những đánh giá quản lý các đặc tính của hệ thống và các tài nguyên cần thiết để xây dựng hệ thống.

6.5.2. Phân tích rủi ro

Trong suốt quá trình phân tích rủi ro, mỗi rủi ro đã nhận dạng, được theo dõi và được đánh giá lại về khả năng có thể xảy ra cũng như mức độ nghiêm trọng của nó. Thực hiện những công việc này là không dễ dàng, người quản lý dự án cần có kinh nghiệm, khả năng phân tích và đánh giá. Không nên đánh giá chúng bằng những con số cụ thể mà nên đưa ra những khoảng giá trị:

1. Khả năng xảy ra rủi ro có thể được đánh giá với các mức sau: rất thấp ($<10\%$), thấp ($10 - 25\%$), trung bình ($25 - 50\%$), cao ($50 - 75\%$), hoặc rất cao ($>75\%$). Thông thường sử dụng ba mức: *thấp*, *trung bình* và *cao*.
2. Ảnh hưởng của rủi ro có thể được đánh giá như sau: *rất tồi tệ*, *ng nghiêm trọng*, *có thể thứ lỗi* hoặc *không quan trọng*.

Các kết quả của tiến trình phân tích nên được lập bảng và sắp xếp theo mức độ nghiêm trọng của rủi ro. Việc đánh giá khả năng xảy ra và độ nghiêm trọng là tùy thuộc vào từng cá nhân. Trong thực tế, ta cần thông tin chi tiết về dự án, tiến trình, nhóm phát triển và tổ chức xem xét để đưa ra những đánh giá thích hợp.

Cả khả năng xảy ra và việc đánh giá mức độ ảnh hưởng của rủi ro có thể thay đổi khi có thêm thông tin về rủi ro và khi các kế hoạch quản lý rủi ro được thực thi. Do đó, bảng này cần được cập nhật trong suốt tiến trình quản lý rủi ro.

Mỗi khi rủi ro được phân tích và xếp hạng, cần thực hiện đánh giá rủi ro nào là quan trọng nhất và cần quan tâm nhất trong suốt dự án. Sự đánh giá này phải dựa trên sự kết hợp khả năng xảy ra của rủi ro và ảnh hưởng của rủi ro đó. Nói chung, các rủi ro rất tồi tệ luôn phải được chú ý, cũng như các rủi ro nghiêm trọng cần được quan tâm hơn các rủi ro xảy ra ở mức trung bình.

Bảng 6.5. Rủi ro và các loại rủi ro

Kiểu rủi ro	Những rủi ro có thể xảy ra
Công nghệ	– Cơ sở dữ liệu được sử dụng trong hệ thống không thể xử lý được số giao dịch/giây như trông đợi. – Các thành phần phần mềm được sử dụng lại chứa khiếm khuyết làm giới hạn các chức năng của chúng.
Con người	– Không tuyển được nhân viên với các kỹ năng được yêu cầu. – Các nhân viên chủ chốt bị ốm và không sẵn sàng tại những thời điểm mang tính quyết định. – Việc đào tạo các nhân viên không sẵn sàng.
Thuộc về tổ chức	– Tổ chức được cơ cấu lại dẫn tới thay đổi những người có trách nhiệm đối với dự án. – Các vấn đề về tài chính của tổ chức dẫn đến việc giảm ngân sách cho dự án.
Công cụ	– Các mã do công cụ CASE tạo ra không hiệu quả. – Các công cụ CASE không thể tích hợp được.
Các yêu cầu	– Thay đổi trong yêu cầu dẫn đến thiết kế lại là rất nhiều. – Các khách hàng không thể hiểu được sự ảnh hưởng của các thay đổi trong các yêu cầu.
Sự đánh giá	– Thời gian cần để phát triển phần mềm vượt dự kiến. – Tỷ lệ sửa khiếm khuyết là thấp hơn so với đánh giá. – Kích cỡ của phần mềm là lớn hơn so với đánh giá.

Boehm (1988) khuyến nghị xác định và giám sát tới 10 rủi ro, nhưng đây chỉ là ý kiến cá nhân. Số lượng thực sự các rủi ro cần giám sát phụ thuộc vào từng dự án. Nó có thể là 5 nhưng cũng có thể là 16. Tuy nhiên, cần phải quản lý được số lượng các rủi ro đã chọn để giám sát. Một số lượng lớn các rủi ro có thể yêu cầu thu thập quá nhiều thông tin. Ta xem xét các loại rủi ro có thể gây ra hậu quả rất tồi tệ và nghiêm trọng (bảng 6.6).

Bảng 6.6. Phân tích rủi ro

Rủi ro	Khả năng xảy ra	Ảnh hưởng
Các vấn đề tài chính của tổ chức dẫn tới giảm ngân sách dành cho dự án	Thấp	Rất tồi tệ
Không thể tuyển dụng thêm nhân viên với các kỹ năng cần thiết cho dự án	Cao	Rất tồi tệ
Nhân viên chính bị ốm vào thời điểm quan trọng của dự án	Trung bình	Nghiêm trọng
Những thay đổi trong các yêu cầu đòi hỏi thực hiện việc tái thiết kế lớn	Trung bình	Nghiêm trọng
Tổ chức được cơ cấu lại dẫn tới người quản lý khác chịu trách nhiệm cho dự án	Cao	Nghiêm trọng
Cơ sở dữ liệu trong hệ thống không thể xử lý số lượng giao dịch / giấy nhiều như mong đợi	Trung bình	Nghiêm trọng
Thời gian cần thiết để phát triển phần mềm nhiều hơn dự kiến	Cao	Nghiêm trọng
Không thể tích hợp các công cụ CASE	Cao	Có thể thứ lỗi
Khách hàng không hiểu được ảnh hưởng của những thay đổi trong các yêu cầu	Trung bình	Có thể thứ lỗi
Không thực hiện được việc đào tạo cần thiết cho nhân viên	Trung bình	Có thể thứ lỗi
Kích cỡ của phần mềm lớn hơn dự kiến	Cao	Có thể thứ lỗi
Mã sinh ra bởi các công cụ CASE không hiệu quả	Trung bình	Không quan trọng

6.5.3. Lập kế hoạch phòng ngừa, hạn chế rủi ro

Quá trình lập kế hoạch phòng ngừa, hạn chế rủi ro cần xem xét mỗi rủi ro chính đã được xác định chưa và đưa ra các chiến lược để quản lý. Không có một tiến trình đơn giản nào có thể thiết lập các kế hoạch quản lý rủi ro. Có năm chiến lược xử lý rủi ro:

1. *Chấp nhận rủi ro*: Không làm gì cả.

Chọn chiến lược này khi xác suất xảy ra rủi ro và tác động là tối thiểu. Nếu chúng xảy ra cũng dễ dàng xử lý.

2. 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, có thể làm thay đổi phạm vi nghiệp vụ. Ở trường hợp này, những thay đổi này cần được chủ dự án và khách hàng chấp nhận, và tất nhiên thu nhập cùng chi phí thường giảm đi.

3. Giám sát và chuẩn bị dự phòng

Chọn một chỉ số để quan sát xem rủi ro đến hay chưa? Ví dụ, nếu 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. Một cách chung nhất là dự trữ một số tiền.

Khi sử dụng chiến lược này, cần có 2 nhân tố: sự kiện phát hiện rủi ro và sự kiện kích hoạt giải pháp. Sử dụng một chỉ tiêu mà quy mô của nó cho thấy rủi ro có thể sẽ xảy ra. Một chỉ tiêu khác cho mức mà phương án đáp ứng cần thực hiện. Với mỗi loại rủi ro khác nhau, quy mô và mức độ này là khác nhau.

4. Chuyển giao 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 với giá cố định. Khi đó đã chuyển rủi ro cho thầu phụ. Chuyển giao là một lợi thế. Tuy nhiên có thể nảy sinh rủi ro mới, cần tính toán một cách cụ thể. Phần quan trọng trong chiến lược này là ký được hợp đồng hiệu quả và quản lý tốt các nhà thầu phụ.

5. 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.

6.5.4. Giám sát rủi ro

Giám sát rủi ro bao gồm việc đánh giá định kỳ các rủi ro đã xác định để xem khả năng xảy ra của nó tăng lên hay giảm đi, hay mức ảnh hưởng của rủi ro có thay đổi hay không. Tất nhiên, điều này không thể nhận biết một cách trực tiếp mà cần phải xem xét các yếu tố khác liên quan đến khả năng

xảy ra và mức độ ảnh hưởng của chúng. Các yếu tố này phụ thuộc vào các loại rủi ro. Bảng 6.7 là một số ví dụ về các yếu tố có thể sử dụng khi đánh giá các loại rủi ro.

Giám sát rủi ro là một quá trình liên tục, tại mỗi lần xem xét lại tiến độ quản lý, các rủi ro chính cần được quan tâm riêng và được thảo luận trong cuộc họp.

Bảng 6.7. Các yếu tố rủi ro

Loại rủi ro	Chỉ số tiềm ẩn có thể
Công nghệ	Việc triển khai phần cứng hay hỗ trợ phần mềm bị chậm; có nhiều báo cáo về các vấn đề công nghệ.
Con người	Tinh thần của nhân viên thấp, mối quan hệ giữa những thành viên trong nhóm thấp; có các công việc khác.
Về tổ chức	Những chuyện ngoài lề của tổ chức; thiếu những hoạt động của người quản lý cấp cao.
Công cụ	Sự bất đắc dĩ trong việc sử dụng các công cụ của các thành viên trong nhóm; những phản nản về các công cụ CASE; yêu cầu các trạm làm việc mạnh hơn.
Yêu cầu	Nhiều yêu cầu thay đổi; phản nản của khách hàng.
Đánh giá	Không đúng lịch; không sửa được các báo lỗi khiếm khuyết.

6.6. Kết thúc dự án

Mục tiêu kết thúc dự án là hoàn tất dự án. Kết thúc dự án xảy ra sau pha triển khai thực hiện. Pha bảo trì là đặc trưng cho việc tiếp tục các dự án khác mà mỗi dự án đó được quản lý riêng (không đề cập đến ở đây). Kết thúc dự án bao gồm các công việc sau:

- *Đóng dự án lại:* Để đóng lại dự án, cần thực hiện một số hoạt động như đánh giá các thành viên và kiến nghị các lợi ích cho họ, hoàn thiện các tài liệu và các 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 là xác định được mặt yếu, mặt mạnh của sản phẩm dự án, của quá trình hình thành lên nó và 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 với khách hàng:* Ký kết các bản thanh lý hợp đồng với khách hàng.

6.7. Mô hình trưởng thành khả năng của tổ chức – CMM

6.7.1. Phát triển khả năng quản lý tiến trình phần mềm

Chất lượng quản lý dự án không chỉ phụ thuộc vào kỹ năng của các nhà phát triển, vào việc áp dụng công nghệ hiện đại và vào kinh nghiệm của các nhà quản lý, mà còn phụ thuộc rất lớn vào năng lực của toàn tổ chức phát triển phần mềm. Vì vậy, xây dựng năng lực quản lý của toàn bộ tổ chức phát triển phần mềm có ý nghĩa quyết định cho việc triển khai thành công các dự án phần mềm.

Dựa vào những quan sát về các công ty hoạt động trên thực tế, người ta thấy rằng: cần phải xây dựng một khung làm việc giúp cho việc gia tăng sự trưởng thành năng lực của tổ chức trong việc đảm bảo tiến trình phần mềm và kết quả là nâng cao chất lượng của phần mềm làm ra.

6.7.2. Sự ra đời của mô hình trưởng thành khả năng

Năm 1986, SEI phát triển mô hình CMM (*Capability Maturity Model*) – mô hình tăng trưởng khả năng (của một tổ chức). Mô hình này là một nỗ lực giúp cho các tổ chức làm phần mềm hoàn thiện tiến trình phần mềm. Ngoài ra còn phải kể đến một loạt chuẩn của tổ chức quốc tế (*International Standards Organization – ISO*) trong ISO 9000 và SPICE (*Software Process Improvement Capability dEtermination*) cũng được phát triển nhằm cùng một mục tiêu như mô hình CMM.

CMM là một nhóm chuẩn, giúp cho việc hoàn thiện tiến trình phần mềm (không tương ứng với mô hình vòng đời thực tế). CMM là độ đo về mức tốt của chính tiến trình phát triển phần mềm của một tổ chức. Hiện nay đã có các CMM:

- Cho phần mềm: SW – CMM.
- Cho quản lý nguồn nhân lực: P – CMM (people – CMM).
- Cho quản lý kỹ nghệ hệ thống: SE – CMM (Systems Engineering – CMM).
- Cho phát triển tích hợp sản phẩm: IPD – CMM (Intergred Pruduct Development – CMM).

- Cho nắm bắt yêu cầu phần mềm: SA – CMM (Software Acquisition – CMM).

ISO 9000 là một loạt chuẩn áp dụng cho hoạt động công nghiệp, gồm: *thiết kế, phát triển, sản xuất, lắp đặt và dịch vụ*. Nó không đơn thuần là các chuẩn dùng cho tiến trình phần mềm. Trong ISO 9000 có ISO.9001 (1987) dành cho hệ thống chất lượng, là thích hợp nhất với phát triển phần mềm và ISO đã công bố hướng dẫn cụ thể để hỗ trợ việc áp dụng ISO.9001 cho phần mềm (ISO-9000-3, 1991).

Cũng như ISO 9000, SPICE được Bộ Quốc phòng Anh khởi xướng mong muốn hoàn thiện nó như một chuẩn quốc tế. Phiên bản đầu tiên hoàn thiện vào năm 1996. SPICE đi xa hơn CMM và ISO 9000 là thiết lập một khuôn khổ đo các phương pháp đánh giá phù hợp với SPICE nhưng không đưa ra phương pháp cụ thể (SW – CMM và ISO 9000 là những phương pháp được làm phù hợp).

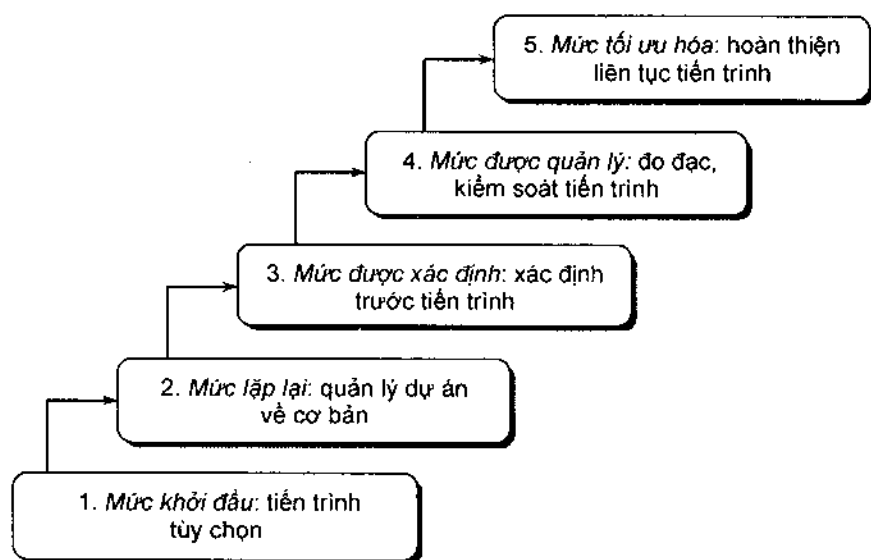
Theo SW – CMM, người ta tin rằng: việc sử dụng kỹ thuật phần mềm mới không tự nó đem đến sự tăng năng suất và lợi nhuận. Vấn đề là ở chỗ, giờ đây chúng ta quản lý các tiến trình phần mềm như thế nào. Chiến lược của SW – CMM là: *cải tiến tiến trình phần mềm sẽ dẫn đến việc cải tiến kỹ thuật như một hệ quả*.

CMM đưa ra một khung làm việc để tổ chức những bước tiến hóa nhỏ của tổ chức thành năm mức trưởng thành, tạo nên một lộ trình làm nền tảng cho việc cải tiến liên tục tiến trình phần mềm. Năm mức trưởng thành này cũng xác định một thước đo để đo độ trưởng thành của tổ chức trong việc quản lý tiến trình phần mềm.

Một mức trưởng thành là một trạng thái tương đối ổn định trong quá trình cải tiến hướng tới một tiến trình phần mềm hoàn thiện. Nó là một lớp cơ sở cho việc cải tiến liên tục tiến trình. Mỗi mức bao gồm một tập các mục tiêu mà khi đạt được chúng là đảm bảo đạt được mức trưởng thành. Việc đạt được một mức trưởng thành sẽ tạo nên một thành phần cấu thành trong tiến trình phần mềm, và kết quả là làm tăng khả năng quản lý của ông ty. Hình 6.11 mô tả năm mức trưởng thành khả năng quản lý tiến trình:

- *Trưởng thành mức 1: Mức khởi đầu.* Mọi hoạt động phát triển được thực hiện một cách tùy biến (không định trước). Phần lớn các tổ chức phần mềm trên thế giới thuộc loại này.

- *Trưởng thành mức 2: Mức được lặp lại.* Việc quản lý dự án phần mềm theo tiến trình về cơ bản được áp dụng để theo dõi chi phí, lịch trình và tính năng của phần mềm. Các quy tắc của tiến trình được thiết lập sao cho kinh nghiệm và sự thành công của mỗi dự án có thể được lặp lại ở dự án tương tự.
- *Trưởng thành mức 3: Mức được xác định.* Quy trình phần mềm cho cả các hoạt động quản lý và các hoạt động sản xuất đều được ghi thành tài liệu, chuẩn hóa và tích hợp thành một quy trình phần mềm chuẩn cho công ty. Tất cả các dự án đều phải sử dụng quy trình phần mềm chuẩn đã được cải biên và công nhận.



Hình 6.11. Năm mức trưởng thành khả năng quản lý tiến trình

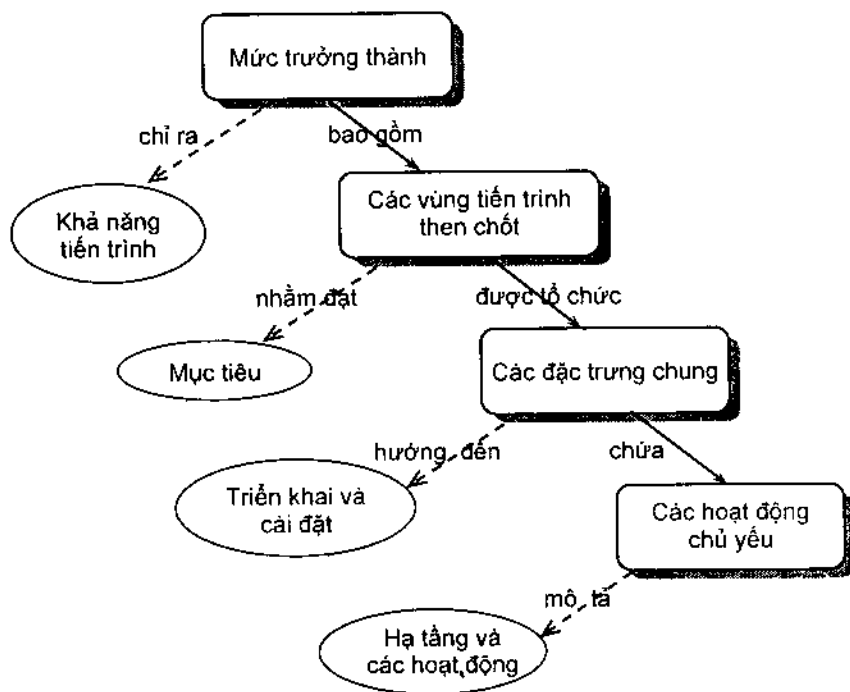
- *Trưởng thành mức 4: Mức được quản lý.* Tổ chức thiết lập được mục tiêu, chất lượng và năng suất cho mỗi dự án. Về mặt số lượng, tiếp tục được đo và điều chỉnh khi mục tiêu đã được chấp nhận. Kiểm tra thống kê chất lượng được thực hiện để quản lý sự thay đổi hay vi phạm về chuẩn chất lượng và năng suất (sử dụng độ đo thống kê lỗi trên 1000 dòng lệnh). Cả tiến trình và sản phẩm đều được định lượng và kiểm soát trong quá trình phát triển.
- *Trưởng thành mức 5: Mức tối ưu hóa.* Tổ chức tiếp tục hoàn thiện tiến trình và kỹ thuật. Thông tin phản hồi từ quản lý thống kê chất

lượng và tiến trình phần mềm được sử dụng để hướng dẫn tổ chức hoàn thiện liên tục tiến trình. Các kiến thức thu được từ dự án, các tư tưởng sáng tạo, công nghệ tiên tiến được áp dụng.

6.7.3. Cấu trúc bên trong của các mức trưởng thành

a) Nội dung của một mức

Mỗi mức trưởng thành, trừ mức 1, đều gồm bốn nội dung: mức trưởng thành, các vùng tiến trình then chốt, các đặc trưng chung và những thực hành chủ yếu (hình 6.12).

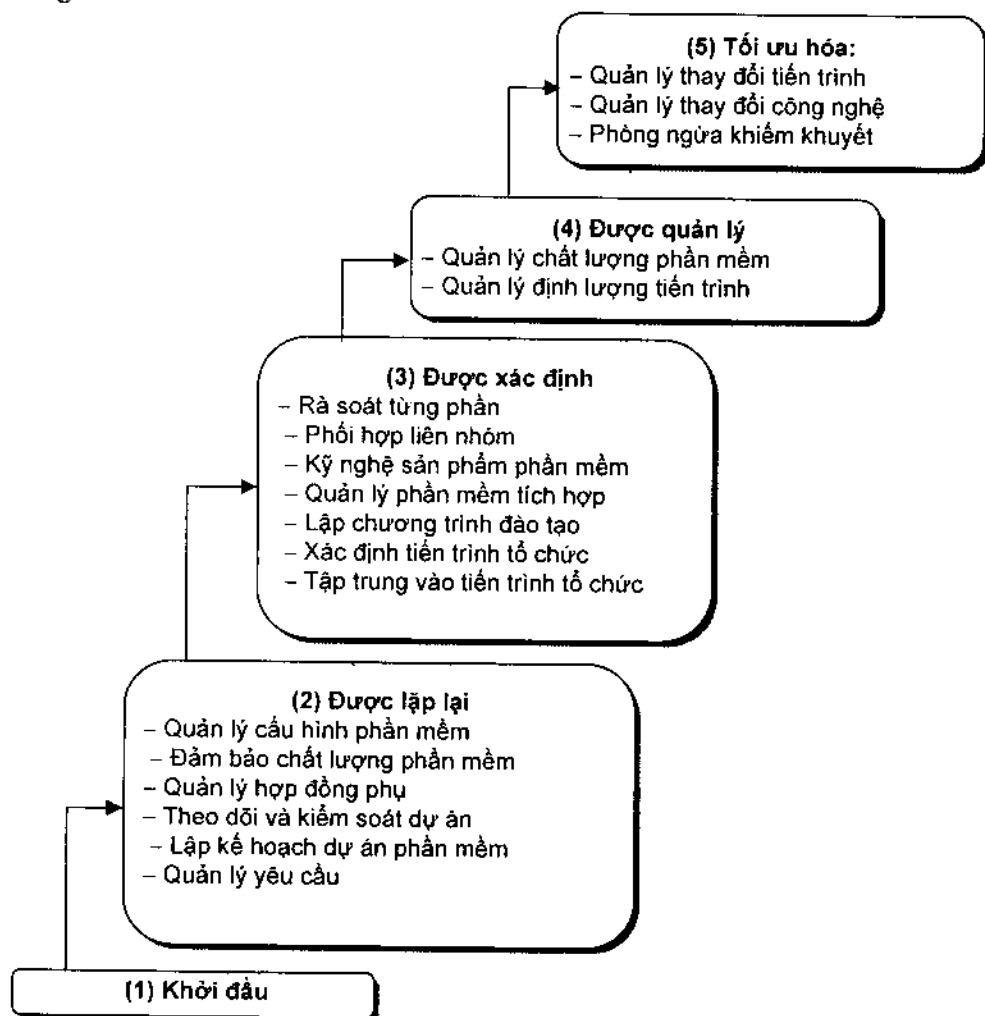


Hình 6.12. Cấu trúc của CMM

Vùng tiến trình then chốt được tổ chức thành 5 phần gọi là những đặc trưng chung. Các đặc trưng mô tả những thực hành then chốt mà khi tập hợp lại chúng hướng đến đạt mục tiêu của vùng tiến trình then chốt. Mỗi vùng tiến trình then chốt thể hiện một lĩnh vực mà công ty phải tập trung vào để cải thiện tiến trình phần mềm của mình, nhằm đạt được mức trưởng thành đó.

b) Vùng tiến trình then chốt

Một vùng tiến trình then chốt là tập các hoạt động liên quan với nhau, nếu được thực hiện một cách đồng thời thì sẽ đạt được một số mục tiêu quan trọng trong việc tăng cường khả năng quản lý tiến trình. Chúng được xác định sao cho chỉ nằm ở một mức trưởng thành. Con đường để đạt được những mục tiêu của một vùng tiến trình then chốt có thể khác nhau đối với mỗi dự án. Khi những mục tiêu của một vùng tiến trình then chốt nào đó được hoàn thành trong nhiều dự án liên tiếp của một công ty, thì họ có thể tuyên bố là "đã thể chế hóa được khả năng tiến trình" được mô tả bởi vùng tiến trình then chốt. Hình 6.13 mô tả nội dung các tập hoạt động chủ yếu của vùng then chốt.



Hình 6.13. Vùng tiến trình then chốt của các mức trưởng thành

CMM không mô tả chi tiết tất cả các vùng tiến trình. Chỉ những tiến trình được coi là then chốt mới được mô tả trong CMM. Các vùng tiến trình then chốt của CMM chỉ ra một công ty cần trưởng thành như thế nào. Những vùng tiến trình then chốt này được xác định dựa trên nhiều năm kinh nghiệm trong lĩnh vực quản lý sản xuất phần mềm, trong việc thẩm định tiến trình phần mềm và phát triển khả năng tiến trình phần mềm.

c) Các đặc trưng chung

Các vùng tiến trình then chốt được tổ chức thành các đặc trưng chung. Các đặc trưng chung là các thuộc tính mô tả sự thực thi và thể chế hóa của một vùng tiến trình then chốt có kết quả hay không, có thể lặp lại được và có kéo dài được hay không. Có năm đặc trưng chung sau đây:

1. *Cam kết thực hiện*: Mô tả các hành động tổ chức cần thực hiện để bảo đảm rằng, tiến trình được thiết lập và khả hiện. Cam kết thực hiện thường liên quan tới việc thiết lập các chính sách của tổ chức và trách nhiệm của các cấp quản lý mức cao.
2. *Khả năng thực hiện*: Mô tả các tiền đề cần có trong dự án và trong tổ chức để thực thi tiến trình phần mềm một cách thành thạo. Khả năng thực hiện thường liên quan tới tài nguyên, cấu trúc của tổ chức và đào tạo.
3. *Các hành động được thực hiện*: Các hành động được thực hiện mô tả vai trò và các thủ tục cần thiết để thực thi một vùng tiến trình then chốt. Các hành động được thực hiện thường liên quan tới việc thiết lập các kế hoạch và các thủ tục, triển khai công việc, theo dõi nó và có những hành động sửa sai khi cần thiết.
4. *Đo đạc và phân tích*: Mô tả nhu cầu đo đạc tiến trình và phân tích kết quả đo được. Nó thường bao gồm các ví dụ đo đạc mà có thể được thực hiện để quyết định trạng thái và tính hiệu lực của các hành động được thực hiện.
5. *Thanh tra thực thi*: Thẩm tra thực thi mô tả các bước để bảo đảm rằng, các hoạt động được thực hiện tuân theo tiến trình đã được thiết lập. Nó thường bao gồm các rà soát và kiểm toán do cấp quản lý và bảo đảm chất lượng phần mềm thực hiện.

d) Các thực hành then chốt

Mỗi vùng tiến trình then chốt có các thực hành then chốt để đáp ứng mục tiêu của vùng tiến trình đó. Nó mô tả cơ sở hạ tầng và các hoạt động góp phần vào việc thực thi và thể chế hóa các vùng tiến trình then chốt. *Các thực hành* của các đặc trưng chung mô tả những gì cần được thực thi để tạo ra khả năng tiến trình. Các thực hành khác, khi được thực hiện sẽ tạo ra nền tảng, từ đó tổ chức có thể thể chế hóa các thực hành mô tả trong *thực hành đặc trưng chung*.

Mỗi thực hành then chốt bao gồm một phát biểu đơn ngắn gọn, kèm theo là mô tả chi tiết hơn, và có thể bao gồm những ví dụ và phần giải thích chi tiết. Những thực hành này cũng bao gồm cả những thực hành then chốt mức cao, phát biểu các quy tắc cơ bản, các thủ tục và các hoạt động cho vùng tiến trình then chốt. Các thành phần của mô tả chi tiết thường được nói đến như là các thực hành phụ.

6.7.4. Vận dụng mô hình CMM

CMM là một khung làm việc, chỉ ra lộ trình hoàn thiện tiến trình cho các công ty phần mềm tham khảo để tăng cường khả năng tiến trình phần mềm của mình. Những mô tả chi tiết của CMM được thiết kế để có thể sử dụng cho nhiều mục đích khác nhau. Có ít nhất bốn cách khai thác mô hình CMM:

- Sử dụng CMM để tìm ra điểm mạnh và điểm yếu của công ty.
- Sử dụng CMM để xác định những rủi ro khi lựa chọn những đối tác.
- Các nhà quản lý và kỹ thuật viên sử dụng CMM để hiểu được các việc cần làm, rồi lên kế hoạch và triển khai chương trình cải tiến tiến trình phần mềm.
- Với các nhóm cải tiến tiến trình, CMM hướng dẫn họ cách xác định và cải tiến các tiến trình phần mềm của công ty.

Để hoàn thiện tiến trình, một tổ chức trước hết cần nhận biết được thực trạng tiến trình hiện tại của mình, sau đó hình thành lên bức tranh cho tiến trình dự kiến, sự hoàn thiện được xác định và sắp xếp theo thứ tự ưu tiên. Một khi kế hoạch và nội dung hoàn thiện được vạch ra để tổ chức thực hiện nó (thực tế cho thấy: để hoàn thiện các mức từ 1 đến 5 cần 18 tháng đến 5

năm. Tuy nhiên, từ mức 1 đến mức 2 có thể đã mất từ 1 đến 3 năm). Quá trình triển khai CMM có thể mô tả như sau:

Bước đầu tiên là lựa chọn nhóm làm việc. Các thành viên của nhóm này phải được học về các khái niệm cơ bản của CMM cũng như các đặc trưng của phương pháp đánh giá và ước lượng. Họ cũng phải có những kiến thức nghiệp vụ về kỹ nghệ phần mềm và quản lý.

Bước thứ hai là đề nghị một số đại diện của công ty mà nhóm muốn đánh giá hoàn thành một bảng câu hỏi về độ trưởng thành và các tài liệu thăm dò khác.

Bước thứ ba: Nhóm đánh giá hoặc nhóm ước lượng sẽ tiến hành phân tích kết quả bằng cách kiểm tra các câu trả lời để xác định xem những vùng nào cần phải tiếp tục thăm dò. Các vùng cần thăm dò tương ứng với các vùng tiến trình then chốt của mô hình CMM. Đến đây thì nhóm đánh giá đã sẵn sàng để đến trực tiếp công ty.

Bước thứ tư: Sử dụng những kết quả của bước phân tích kết quả, nhóm tiến hành các cuộc phỏng vấn và xem xét các tài liệu để nắm bắt được tiến trình đang được công ty sử dụng. Các vùng tiến trình then chốt và các thực hành then chốt trong CMM sẽ giúp họ cách đặt câu hỏi, cách thu thập thông tin và tổng hợp các thông tin thu được. Sau đó nhóm vận dụng những đánh giá nghiệp vụ để quyết định xem các vùng tiến trình then chốt mà công ty thực hiện có thoả mãn các mục tiêu tương ứng hay không. Những đánh giá này có thể phải thực hiện dựa trên những thông tin không đầy đủ, trong đó có các vấn đề liên quan tới bảo mật dữ liệu và bí mật của công ty. Khi mà các thực hành then chốt của công ty khác với các thực hành then chốt của CMM, nhóm đánh giá phải ghi lại lý do dẫn đến việc phải thay đổi vùng tiến trình then chốt đó.

Bước thứ năm: Nhóm đánh giá phải lập danh sách các vấn đề còn tồn tại để chỉ ra những điểm mạnh và điểm yếu trong tiến trình phần mềm của công ty. Nó là cơ sở để đưa ra những kiến nghị để cải tiến tiến trình; trong ước lượng khả năng phần mềm, những tồn tại này trở thành một phần của việc phân tích rủi ro.

Cuối cùng, nhóm sẽ chuẩn bị một hồ sơ vùng tiến trình then chốt, chỉ ra các vùng mà công ty đã thoả mãn hoặc chưa thoả mãn những mục tiêu của vùng tiến trình then chốt.

SEI đã phát triển hai phương pháp để đánh giá độ trưởng thành của một công ty phần mềm:

1. Các phương pháp *Đánh giá tiến trình phần mềm* được sử dụng để xem xét trạng thái hiện tại của tiến trình phần mềm của một công ty, các vấn đề quan trọng khác có liên quan tới tiến trình phần mềm mà công ty đang gặp phải và tìm kiếm sự trợ giúp công ty trong việc cải tiến tiến trình phần mềm.
2. Các phương pháp *Ước lượng khả năng phần mềm* được sử dụng để đánh giá các đối tác, xem họ có đủ năng lực để thực hiện các công việc về phần mềm hay không và để theo dõi trạng thái của một tiến trình phần mềm sử dụng trong dự án.

Mặc dù có những điểm tương đồng, nhưng kết quả của đánh giá tiến trình phần mềm và ước lượng khả năng phần mềm có thể khác nhau. Một lý do là phạm vi của việc đánh giá và việc ước lượng có thể rất đa dạng. Việc đánh giá tiến trình phần mềm và ước lượng khác nhau cả về động cơ, mục đích, kết quả và người sở hữu kết quả đánh giá.

Đánh giá tiến trình phần mềm được thực hiện trong môi trường mở và hợp tác. Sự thành công phụ thuộc vào cam kết từ cả nhà quản lý và lập trình viên cùng nhau xây dựng công ty. Mục đích là giúp công ty tìm ra những vấn đề còn tồn tại để khắc phục. Trong khi bảng câu hỏi giúp nhóm đánh giá tập trung vào các vấn đề liên quan tới mức trưởng thành của công ty thì các cuộc phỏng vấn có cấu trúc và những cuộc phỏng vấn mang tính ngẫu hứng lại giúp họ hiểu được tiến trình phần mềm hiện có của công ty.

Ước lượng khả năng phần mềm thì lại khác, được thực hiện trong môi trường kiểm toán. Mục đích của việc ước lượng gắn với yếu tố tài chính vì những gợi ý của nhóm đánh giá có thể giúp công ty lựa chọn được đối tác phù hợp, hoặc đặt những mức thưởng, phạt hợp lý. Sự chú ý tập trung vào những tài liệu báo cáo, những bước mà một tiến trình phần mềm thực hiện trong thực tế. Tuy nhiên, điều đó không có nghĩa là kết quả của đánh giá tiến trình phần mềm và ước lượng khả năng phần mềm là không thể so sánh với nhau. Vì cả hai phương pháp đều dựa trên CMM, nên những việc so sánh giống và khác nhau đều phải có bằng chứng và phải giải thích rõ ràng.

6.7.5. Mô hình CMMI và RMM

Từ năm 1991, các mô hình CMM đã phát triển đến hàng chục nghìn luật cho rất nhiều lĩnh vực khác nhau, từ tiến trình phát triển phần mềm đến tiến trình phát triển hệ thống. Mặc dù nó được sử dụng rất tốt cho nhiều công ty, nhưng việc có quá nhiều mô hình lại là một trở ngại lớn cho việc vận dụng. Sự khác nhau giữa các mô hình CMM về mặt cấu trúc, nội dung, cũng như cách tiếp cận, làm hạn chế khả năng có thể tập trung cải tiến một cách hiệu quả tiến trình của công ty. Ngoài ra, việc áp dụng nhiều mô hình riêng rẽ cho một công ty là rất tốn kém.

Dự án CMMI (*Capability Maturity Model, Integration*) được thực hiện để giải quyết các vấn đề trên. Nhóm phát triển đã kết hợp CMMI từ ba mô hình nguồn là:

- Mô hình trưởng thành khả năng cho phát triển phần mềm (SW – CMM) version 2.0 draft C.
- Mô hình tăng trưởng khả năng kỹ nghệ hệ thống (SECM).
- Mô hình trưởng thành khả năng cho phát triển sản phẩm tích hợp (IPD – CMM).

Sự kết hợp của 3 mô hình trên vào một khung làm việc duy nhất, làm cho các công ty có thể áp dụng cho phạm vi toàn công ty. Ba mô hình trên được lựa chọn do sự áp dụng rộng rãi chúng trong kỹ nghệ phần mềm và kỹ nghệ hệ thống nói chung ở nhiều công ty. Vì thế, CMMI có thể áp dụng ở các công ty đã triển khai một trong ba mô hình thành phần cũng như cho các công ty mới bắt đầu áp dụng CMM.

Trong những năm gần đây, tiến trình phát triển phần mềm theo hướng sử dụng lại phát triển rất mạnh. Cùng với nó là sự ra đời của mô hình RMM (*Reuse Maturity Model*), nhằm tăng cường năng lực của tổ chức trong việc phát triển phần mềm theo hướng sử dụng lại.

Tổng kết

- ♦ Quản lý dự án tốt là yếu tố cơ bản đảm bảo một dự án phần mềm được phát triển thành công, đúng thời hạn, trong giới hạn ngân sách cho phép.

- ◆ Phần mềm là không nhìn thấy, mọi dự án phần mềm đều khác với các dự án trước đó. Do đó không có kinh nghiệm, hiểu biết nào có thể hướng dẫn đầy đủ, chính xác việc quản lý triển khai nó.
- ◆ Người quản lý dự án phần mềm có nhiều vai trò. Các hoạt động của họ là lập kế hoạch, ước lượng chi phí và lập lịch, thúc đẩy và kiểm soát tiến trình thực hiện dự án. Tiến trình quản lý dự án là một quá trình lặp. Các kế hoạch chính của dự án gồm kế hoạch công việc, kế hoạch nguồn lực (nhân lực, tài chính), kế hoạch đảm bảo chất lượng và tương ứng với chúng là các hoạt động quản lý.
- ◆ Các cột mốc dự án là thời điểm được dự kiến mà tại đó các đầu ra của một số hoạt động dự án được ấn định, bao gồm sản phẩm gửi đến khách hàng, một số báo cáo chính thức được trình lên các nhà quản lý để xét duyệt.
- ◆ Việc lập lịch dự án nhằm tạo ra các biểu diễn đồ thị các hoạt động của dự án, trợ giúp cho việc quản lý. Nó được thực hiện nhờ các công cụ phần mềm chuyên dùng trên cơ sở của các phương pháp như phương pháp PERT, vẽ sơ đồ.
- ◆ Mọi dự án đều có thể có rủi ro, cần được xác định, đánh giá xác suất chúng xảy ra cùng hậu quả chúng đem đến. Từ đó lập kế hoạch để hạn chế, ngăn ngừa nó.
- ◆ Chất lượng quản lý không chỉ phụ thuộc vào việc vận dụng mô hình, phương pháp mà cả năng lực quản lý tiến trình phát triển mềm của tổ chức. Mô hình CMM là một khung làm việc hướng dẫn và đo mức độ tăng trưởng khả năng quản lý tiến trình của tổ chức. Có năm mức: Mức 1 – mức khởi đầu (quản lý tùy biến). Mức 2 – mức được lặp lại (quản lý dự án về cơ bản theo kinh nghiệm dự án đã có). Mức 3 – mức được xác định (mọi hoạt động quản lý tiến trình và sản phẩm đều được xác định và tuân thủ). Mức 4 – mức được quản lý (mục tiêu, chất lượng sản phẩm, năng suất dự án đều được xác định, đo đạc và kiểm soát). Mức 5 – mức tối ưu (tiến trình liên tục được hoàn thiện cho mọi dự án).

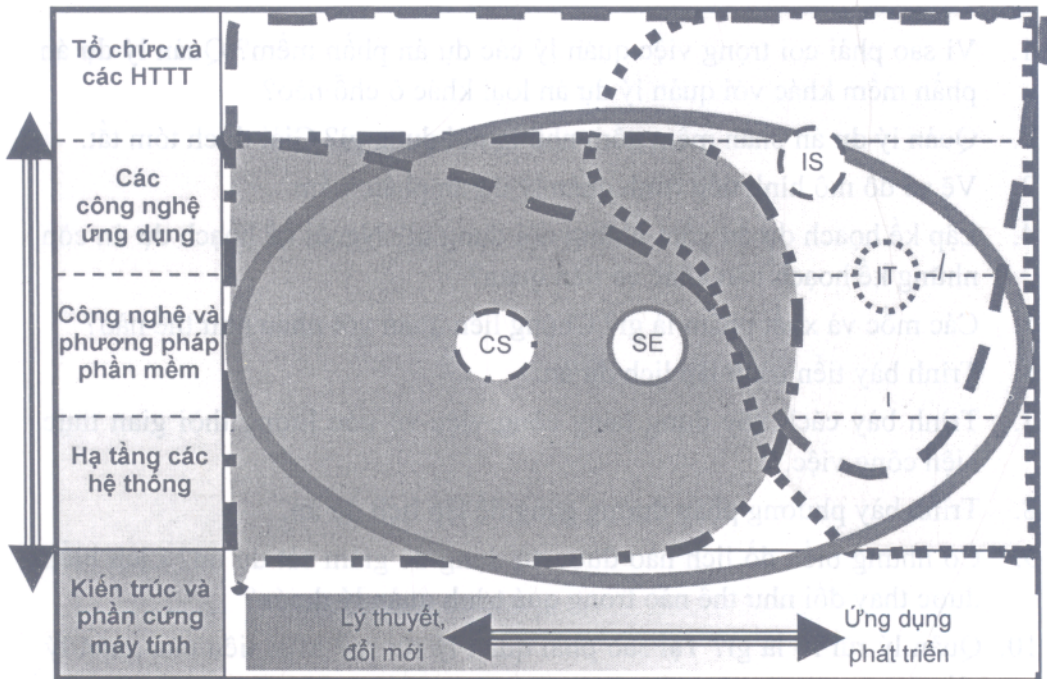
- ◆ Mỗi mức trưởng thành có những vùng tiến trình then chốt, chỉ ra những hoạt động chính cần thực hiện để đạt được mức tăng trưởng tương ứng.
- ◆ Ngoài CMM còn có các chuẩn và mô hình khác hỗ trợ nâng cao năng lực tổ chức để quản lý tiến trình phát triển phần mềm một cách hiệu quả như CMMI, RMM.

Câu hỏi ôn tập

1. Vì sao phải coi trọng việc quản lý các dự án phần mềm? Quản lý dự án phần mềm khác với quản lý dự án loại khác ở chỗ nào?
2. Quản lý dự án phần mềm gồm những nội dung gì? Giải thích tóm tắt.
3. Vẽ sơ đồ mô hình tiến trình quản lý dự án phần mềm.
4. Lập kế hoạch dự án gồm những nội dung gì? Ngoài kế hoạch dự án còn những kế hoạch nào khác có liên quan?
5. Các mốc và xuất phẩm là gì? Chúng liên quan với nhau như thế nào?
6. Trình bày tiến trình lập lịch dự án.
7. Trình bày cách xây dựng bảng công việc và ước lượng thời gian thực hiện công việc.
8. Trình bày phương pháp đường găng để lập lịch dự án.
9. Có những biểu đồ lịch nào được sử dụng để giám sát dự án? Lịch biểu được thay đổi như thế nào trong quá trình quản lý dự án?
10. Quản lý rủi ro là gì? Tại sao phải quản lý rủi ro? Mục tiêu của quản lý rủi ro.
11. Nội dung hoạt động quản lý rủi ro gồm những gì? Vẽ sơ đồ tiến trình quản lý.
12. Nêu các loại rủi ro và giải pháp tương ứng có thể áp dụng cho nó.
13. Mô hình CMM là cái gì? Có những mức tăng trưởng khả năng nào trong mô hình CMM? Nội dung của mỗi mức là gì?
14. Làm thế nào để một tổ chức đạt được các mức tăng trưởng đó? Cái gì là giải pháp và thước đo về các mức tăng trưởng?
15. Nêu các chuẩn quốc tế khác về phần mềm. Trình bày sự khác nhau giữa mô hình CMM và các chuẩn đó.
16. Nêu các bước vận dụng mô hình CMM vào tổ chức.

PHỤ LỤC

A. Phân biệt kỹ nghệ phần mềm với khoa học máy tính



CS – Computer Science, IS – Information Systems,
IT – Information Technology, SE – Software Engineering

Hình A. Phạm vi các lĩnh vực xét theo các chiều khác nhau
(Theo số liệu của tài liệu Computing Curricula – dự án của ACM, AIS và IEEE, 2004)

Nhà khoa học xây dựng cái để học, người kỹ sư học để xây dựng (*A scientist builds in order to learn; an engineer learns in order to build – Fred Brooks*). Kỹ nghệ phần mềm không phải khoa học máy tính.

Khoa học máy tính (computer science) liên quan đến những vấn đề lý thuyết và phương pháp làm cơ sở cho việc xây dựng và hoạt động của các hệ thống máy tính và phần mềm. Trong khi đó, *kỹ nghệ phần mềm (software*

engineering) lại liên quan đến những vấn đề thực tế của việc sản xuất ra các phần mềm. Một cách lý tưởng thì mọi vấn đề của kỹ nghệ phần mềm đều dựa trên những cơ sở lý thuyết của khoa học máy tính. Nhưng trên thực tế, nhiều vấn đề của kỹ nghệ phần mềm chưa có những cơ sở lý thuyết, nên những người phát triển thường phải sử dụng một cách tiếp cận tùy biến dựa trên kinh nghiệm và thực hành. Hình A cho ta hình dung phạm vi của kỹ nghệ phần mềm và các lĩnh vực khác thuộc ngành Tin học (*computing*). Rõ ràng kỹ nghệ phần mềm nằm ở trung tâm và giao cắt với mọi ngành.

B. Chuẩn ISO 9126 về chất lượng phần mềm

Chuẩn ISO /IEC 9126 được đưa ra vào năm 1991, quy định các đặc tính chất lượng của phần mềm. Có 6 đặc tính lớn và được thể hiện thông qua 21 đặc tính con.

1. Đặc tính chức năng (*Functionality*)

Đặc tính này liên quan đến khả năng thực hiện một loạt các thao tác để đạt được các yêu cầu hiện hay ẩn của người sử dụng. Đặc tính này được chi tiết hóa thành 5 đặc tính con sau:

- *Tính phù hợp (Suitability)*: Đặc tính của phần mềm chỉ ra các chức năng phải đầy đủ và phù hợp đối với các nhiệm vụ đã được xác định.
- *Tính chính xác (Accurateness)*: Phần mềm phải cho kết quả hoặc hiệu quả chính xác như dự tính.
- *Khả năng liên tác (Interoperability)*: Phần mềm có khả năng hoạt động tương tác với các hệ thống được xác định.
- *Tính tuân thủ chuẩn (Compliance)*: Phần mềm tham gia được với các ứng dụng liên quan tới các chuẩn, các quy ước, luật lệ.
- *Tính an toàn (Security)*: Khả năng phòng, chống lại các truy cập không được phép dù là vô tình hay cố ý đối với chương trình hoặc dữ liệu.

2. Độ tin cậy (*Reability*)

Độ tin cậy là thuộc tính liên quan đến khả năng bảo toàn hiệu quả của phần mềm dưới những tác động của các điều kiện hay ở mức giới hạn nào đó. Đặc tính này gồm các đặc tính con sau:

- *Độ chín – trưởng thành (Maturity)*, thể hiện qua tỷ lệ trục trặc do lỗi của phần mềm.
- *Khả năng thứ lỗi (Fault Tolerance)*, thể hiện khả năng bền vững của phần mềm, cho phép phần mềm vẫn duy trì được hiệu quả xác định dưới tác động của các yếu tố không thuận lợi như lỗi hay sự vi phạm về giao tiếp với phần mềm.
- *Khả năng khôi phục (Recoverability)*, khả năng và nỗ lực cần thiết để khôi phục lại được hoạt động của hệ-thống và khôi phục dữ liệu đạt được hiệu quả ở mức độ cho phép và kịp thời khi hệ thống gặp sự cố.

3. Tính sử dụng được (*Usability*)

Sử dụng được liên quan đến khả năng sử dụng phần mềm theo hướng giảm thiểu nỗ lực của người sử dụng. Đặc tính này gồm các đặc tính con sau:

- *Dễ hiểu (Understandability)*: Thuộc tính liên quan tới nỗ lực cần thiết của người dùng để hiểu được quan niệm logic của phần mềm cũng như khả năng áp dụng được những quan niệm đó.
- *Dễ học (Learnability)*: Tính năng liên quan đến nỗ lực cần thiết của người sử dụng để học cách sử dụng phần mềm.
- *Dễ thao tác (Operability)*: Thuộc tính liên quan đến nỗ lực cần thiết của người sử dụng để thao tác và kiểm soát các thao tác đối với phần mềm.

4. Tính hiệu quả (*Efficiency*)

Đặc tính này gồm các đặc tính con sau:

- *Hiệu quả thời gian (Time Behavior)*: Thời gian đáp ứng, thời gian xử lý và tần suất thông qua có hiệu quả (số lần thực hiện có tính chu kỳ trong một khoảng thời gian của các chức năng phần mềm).
- *Hiệu quả tài nguyên (Resource Behavior)*: Khối lượng tài nguyên được sử dụng và khoảng thời gian sử dụng tài nguyên có hiệu quả.

5. Tính bảo trì được (*Maintenability*)

Đặc tính này gồm các đặc tính con sau:

- *Phân tích được (Analysability)*: Đặc tính liên quan đến nỗ lực cần thiết để chẩn đoán khiếm khuyết, hoặc nhận diện được phần cần phải thay đổi.
- *Thay đổi được (Changeability)*: Đặc tính liên quan đến nỗ lực cần thiết để sửa đổi, huỷ bỏ phần có lỗi hoặc thay đổi môi trường.
- *Ổn định (Stability)*: Đặc tính liên quan đến các rủi ro hoặc các hiệu quả không mong muốn khi sửa đổi phần mềm.
- *Kiểm tra được (Testability)*: Đặc tính liên quan đến các nỗ lực cần thiết để kiểm chứng tính hợp thức của phần mềm được thay đổi.

6. Tính khả chuyển (Portability)

Đặc tính này gồm các đặc tính con sau:

- *Thích nghi được (Adaptability)*: Đặc tính thể hiện khả năng phần mềm thích ứng được với nhiều môi trường khác nhau mà không cần phải có những điều chỉnh đặc biệt.
- *Cài đặt được (Installability)*: Đặc tính liên quan đến các nỗ lực cần thiết để cài đặt chương trình trong một môi trường xác định.
- *Tương hợp (Conformance)*: Đặc tính thể hiện khả năng kết hợp được với các chuẩn và quy ước có liên quan khi thay đổi môi trường.
- *Thay thế được (Replaceability)*: Khả năng và các nỗ lực cần thiết để sử dụng phần mềm thay thế một phần mềm khác trong cùng môi trường của phần mềm.

TÀI LIỆU THAM KHẢO

- [1] Avizienis, A.A. *A Methodology of programming*. In Software fault Tolerance (M.R. Lyu ed.) Chichester: John Wiley and sons, 23 – 46 (Ch. 18).
- [2] Beizer, B. *Software Testing Techniques*. 2nd Edition, New York, Van Nostrand Rheinhold, 1990, (Ch. 20).
- [3] Beizer, B. *Black – box Testing*. Wiley, 1995.
- [4] Boehm, B. *Software Engineering Economics*. Prentice Hall, 1981.
- [5] Boehm, B. *A Spiral Model for Software Development and Enhancement*. Computer, vol. 21, No. 5, pp. 61 – 72, 1988.
- [6] Boehm B. *Using the WINWIN Spiral Model: A case Study*. Computer, vol. 31, No. 7, July 1998, pp. 33 – 34.
- [7] C. Bolun and I. Jacopini. *Flow Diagrams, Turing Machines, and Languages with on Two Formation Rules*. Communication of the ACM 9 (may), 1966.
- [8] Grady Booch, James Rumbaugh, Ivar Jacobson. *The Unified Modeling -language User Guid*. Addison – Wesley, 1998.
- [9] Chen, P. P – S. *The Entity – Relationships Model – Toward a Unified View of Data*. AMC Transaction on Database Systems. 1 (mach), 1976.
- [10] Ivica CrnKovic, Magnus Larsson. *Building Reliable Component – Based Software Systems*. Artech House Publishers, 1st Edition, July 15, 2002.

- [11] Computing Curricula 2004. *Overview Report / Including: A guide to Undergraduate Degree Programs in Computing* (A cooperative project of The Associate for Computing, The Associate for Information Systems, The Computing Society).
- [12] Frewin, G.D. and Hatton B.J. *Testing Software Using Multiple Versions*. Software Productivity Consortium, Report No 89029N, Reston, VA, June, 1989.
- [13] Gamma, E., Richard Helm, Ralph Johnson and John Vlissides. *Design Patterns: Elements of Reusable Object – Oriented Software*. Reading, MA: Addison Wesley, 1995, 1998.
- [14] C. Gane and T. Sarson. *Structured Systems Analysis*. Englewood Cliffs, NJ: Prentice – Hall, 1979.
- [15] Heniger, K.L. *Specifying Software Requirements for complex Systems*. New Technique and their Applications. IEEE Trans on Software Engineering, SE – 6(1), 2 – 13 (Ch. 5).
- [16] Kampen, G.R. *An Electric Approach to Specification*. "Proceeding of fourth International Workshop Specification and Design, Monterey, CA, April 1987, pp. 178 – 182 (Ch. 10).
- [17] Kotonya G. and Sommeville. I. *Requirements Engineering with Viewpoints*. BCS/IEE Software Engineering J. 11(1), pp. 5 – 8.
- [18] Larsen G. *Designing Component – Based Frameworks using Patterns in the UML*. Communications of the ACM, 42(10): 38 – 45, October 1999.
- [19] McCabe T.J. *A Software Complexity Measure*. IEEE Trans. on Software Engineering, Vol 1. SE – 2. December, 1976, pp. 308 – 320.
- [20] Musa, J.D. and Ackerman, A.F. *Quantifying Software Validation. When to Stop Testing?* IEEE Software, May, 1989, pp. 19 – 27.
- [21] Myers, G. *The Art of Software Testing*. Wiley, 1979.
- [22] Myer, B. *Object – oriented Software construction*. Prentice – Hall, 1988
- [23] Naur, P. et al (ed.). *Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee*. NATO, 1969.

- [24] Ngô Trung Việt, Nguyễn Kim Ánh. *Nhập môn Công nghệ phần mềm*, NXB Khoa học và Kỹ thuật, 2003.
- [25] Norman. *The Invisible Computer*. MIT Press, 1998.
- [26] Ould, M. *Managing Software Quality and Business Risk*. John Wiley and Sons, 1999.
- [27] Page – Jones M. *The Practical Guide to Structured Systems Design*. NY: Yourdon Press, 1980.
- [28] Parnas, D.L. *On Criteria to be used in Decomposing Systems into Modules*. CAM, Vol. 14, No. 1, April 1972, pp. 221 – 227.
- [29] Pressman R.S. *Software Engineering, a Practitioner's Approach*. Fifth Edition, McGraw Hill, 2001.
- [30] Randel, B. Xu, J. *The Evolution of Recover block concept*. In Software fault Tolerance (M.R. Lyu ed.) Chichester: John Wiley and sons, 1 – 22 (Ch. 18).
- [31] Gregg Rothermel, Mary Jean Harrold. *A Framework for Evaluating Regression Test Selection Techniques*. Proceedings of the Sixteenth International Conference on Software Engineering, Sorrento, Italy, May 1994, p. 201 – 210.
- [32] Rational Rose Corp. *Rational Rose 2002*. <http://www.rational.com/uml/>.
- [33] Walker Royce. *Software Project Management. A Unified Framework*, Addison – Wesley, 1998.
- [34] Ian Sommerville. *Software Engineering*. 6th Edition, Addison – Wasley, 2001.
- [35] Stevens W., Myer G. and Constantine L. *Structured Design*. IBM System Journal vol. 13, No. 2, 1974.
- [36] John Vu. *Nhập môn kỹ nghệ phần mềm*. <http://www.segvn.org/forum/mvnforum/search?member=JohnVu&offset=60>.
- [37] Nguyễn Văn Vy. *Phân tích thiết kế hệ thống thông tin hiện đại – hướng cấu trúc và hướng đối tượng*, NXB Thống kê, Hà Nội, 2002.
- [38] Nguyễn Văn Vy. *Phân tích thiết kế hệ thống thông tin quản lý*. TT Khoa học tự nhiên và Công nghệ, 2007.

- [39] Wallace D.R and Fujii, R.U. *Software Verification and Validation. An Review*. IEEE Software, May, 1989, pp. 10 – 17.
- [40] Niklaus Wirth. *Program Development by Steptwice Refinement*, CACM, vol. 14, No. 4, 1971, pp. 221 – 227.
- [41] Yourdon E. and Constantine L. *Structured Design*. Prientice Hall, 1979.
- [42] Yourdon, E. *Formal Methods*. Guerrilla Programmer, Cutter Information Corp., October, 1994.
- [43] Yourdon E. *Software Reuse*. Application Development Strategies, vol. 6, No. 12, 10/1994, 1 – 16 pp.
- [44] Zimmermann, H. *OSI Reference Model – The OSI model of Arthitecture for Open Systems Interconnection*. IEEE Trans. on Communication, COM – 28(4), pp. 425 – 432.
<http://www.rationalcom/pst/products/rosefamily.htm>

MỤC LỤC

LỜI NÓI ĐẦU	3
CHỮ VIẾT TẮT.....	5
Chương 1. PHẦN MỀM VÀ KỸ NGHỆ PHẦN MỀM	7
1.1. Khái niệm về phần mềm	8
1.2. Sự phát triển của phần mềm và thách thức	16
1.3. Kỹ nghệ phần mềm	19
1.4. Chất lượng phần mềm	39
1.5. Kỹ nghệ hệ thống dựa trên máy tính	40
Tổng kết	45
Câu hỏi ôn tập	47
Chương 2. XÁC ĐỊNH VÀ ĐẶC TẢ YÊU CẦU PHẦN MỀM	48
2.1. Tổng quan về yêu cầu phần mềm	48
2.2. Xác định yêu cầu phần mềm	57
2.3. Đặc tả yêu cầu	72
2.4. Thẩm định yêu cầu	87
2.5. Đặc tả hệ thống và làm bản mẫu	90
Tổng kết	95
Câu hỏi ôn tập	96
Chương 3. THIẾT KẾ PHẦN MỀM	98
3.1. Tổng quan về thiết kế phần mềm	98
3.2. Thiết kế kiến trúc	116
3.3. Thiết kế hệ thống hướng chức năng	129
3.4. Thiết kế hệ thống hướng đối tượng	132
3.5. Thiết kế hệ thống thời gian thực	136
3.6. Thiết kế giao diện người – máy tính	142
3.7. Tài liệu thiết kế	146

Tổng kết	147
Câu hỏi ôn tập	148
Chương 4. LẬP TRÌNH	150
4.1. Ngôn ngữ lập trình	150
4.2. Phương pháp lập trình	159
4.3. Phong cách lập trình	162
4.4. Lập trình tránh lỗi	166
4.5. Lập trình hướng hiệu quả thực hiện	173
4.6. Một số môi trường phát triển	175
Tổng kết	179
Câu hỏi ôn tập	180
Chương 5. XÁC MINH VÀ THẨM ĐỊNH	181
5.1. Giới thiệu	181
5.2. Các khái niệm về kiểm thử	183
5.3. Các phương pháp và chiến lược kiểm thử	188
5.4. Các loại hình kiểm thử	207
5.5. Gỡ lỗi	223
5.6. Kiểm thử tự động	225
Tổng kết	229
Câu hỏi ôn tập	231
Chương 6. QUẢN LÝ DỰ ÁN PHẦN MỀM	233
6.1. Dự án và quản lý dự án	234
6.2. Các hoạt động quản lý dự án	237
6.3. Lập kế hoạch dự án	239
6.4. Lập lịch dự án	243
6.5. Quản lý rủi ro	252
6.6. Kết thúc dự án	259
6.7. Mô hình trưởng thành khả năng của tổ chức – CMM	260
Tổng kết	269
Câu hỏi ôn tập	271

PHỤ LỤC	272
A. Phân biệt kỹ nghệ phần mềm với khoa học máy tính	272
B. Chuẩn ISO 9126 về chất lượng phần mềm	273
1. Đặc tính chức năng (<i>Functionality</i>)	273
2. Độ tin cậy (<i>Reability</i>)	273
3. Tính sử dụng được (<i>Usability</i>)	274
4. Tính hiệu quả (<i>Efficiency</i>)	274
5. Tính bảo trì được (<i>Maintenability</i>)	274
6. Tính khả chuyển (<i>Portability</i>)	275
TÀI LIỆU THAM KHẢO	276

Chịu trách nhiệm xuất bản :

Chủ tịch HĐQT kiêm Tổng Giám đốc NGÔ TRẦN ÁI

Phó Tổng Giám đốc kiêm Tổng biên tập NGUYỄN QUỲ THAO

Biên tập nội dung và sửa bản in :

ĐỖ HỮU PHÚ

Thiết kế mỹ thuật và trình bày bìa :

BÍCH LA

Thiết kế sách và chế bản :

ĐỖ PHÚ

GIÁO TRÌNH KỸ NGHỆ PHẦN MỀM

Mã số: 7B737Y9 – DAI

In 1.000 bản (QĐ : 30), khổ 16 x 24 cm. In tại Nhà in Đại học Quốc Gia Hà Nội.

Địa chỉ : 16 Hàng Chuối, Hà Nội.

Số ĐKKH xuất bản : 161 - 2009/CXB/14 - 208/GD.

In xong và nộp lưu chiểu tháng 5 năm 2009.



CÔNG TY CỔ PHẦN SÁCH ĐẠI HỌC - DẠY NGHỀ
HEVOBCO
25 HÀN THUYỀN - HÀ NỘI
Website : www.hevobco.com.vn



VƯƠNG MIỆN KIM CƯƠNG
CHẤT LƯỢNG QUỐC TẾ

TÌM ĐỌC

SÁCH THAM KHẢO VỀ TOÁN - TIN

CỦA NHÀ XUẤT BẢN GIÁO DỤC VIỆT NAM

- | | |
|---|----------------------------|
| 1. Toán học cao cấp (tập 1, 2, 3) | Nguyễn Đình Trí (Chủ biên) |
| 2. Bài tập Toán học cao cấp (tập 1, 2, 3) | Nguyễn Đình Trí (Chủ biên) |
| 3. Đồ thị và các thuật toán | Hoàng Chí Thành |
| 4. Giáo trình Nhập môn Hệ cơ sở dữ liệu | Nguyễn Tuệ |
| 5. Giáo trình Hệ điều hành Unix - Linux | Hà Quang Thụy (Chủ biên) |
| 6. Giáo trình Khai phá dữ liệu Web | Hà Quang Thụy (Chủ biên) |
| 7. Toán rời rạc ứng dụng trong tin học | Đỗ Đức Giáo |
| 8. Hướng dẫn giải Bài tập Toán rời rạc | Đỗ Đức Giáo |
| 9. Giáo trình Nguyên lý hệ điều hành | Hồ Đắc Phương |
| 10. Giáo trình Nhập môn mạng máy tính | Hồ Đắc Phương |

Bạn đọc có thể mua sách tại các Công ty Sách – Thiết bị trường học ở các địa phương hoặc các cửa hàng sách của Nhà xuất bản Giáo dục Việt Nam :

- Tại TP. Hà Nội : 25 Hàn Thuyên ; 187B Giảng Võ ; 232 Tây Sơn ; 23 Tràng Tiền ;
- Tại TP. Đà Nẵng : Số 15 Nguyễn Chí Thanh ; Số 62 Nguyễn Chí Thanh.
- Tại TP. Hồ Chí Minh : Cửa hàng 451B – 453 ; Hai Bà Trưng – Quận 3 ;
240 Trần Bình Trọng – Quận 5.
- Tại TP. Cần Thơ : Số 5/5, đường 30/4.

Website : www.hevobco.com.vn



89349801984239



Giá: 36.000đ