Chương 1: PHẦN MỀM TIẾN TRÌNH VÀ QUẢN LÝ

1.1 Phần mềm và công nghệ phần mềm

1.1.1 Lịch sử và mục tiêu của công nghệ phần mềm

Ngay từ khi xuất hiện các hệ máy tính và ngôn ngữ lập trình đầu tiên thì phần mềm cũng bắt đầu xuất hiện. Tuy nhiên, khi 1 loại phần cứng mới được giới thiệu – phần cứng dựa vào dòng điện tích hợp (integrated circuits) hay còn gọi là chip điện tử - thì việc phát triển phần mềm rơi vào khủng hoảng.

- Các đơn đặt hàng yêu cầu phần mềm có kích thước lớn và độ phức tạp cao ngày càng nhiều, trong khi việc phát triển phần mềm theo cách từ trước ngày càng không phù hợp.
- Nhiều dự án lớn bị trễ 1 thời gian dài.
- Chi phí phần mềm lớn hơn nhiều so với dự đoán.
- Phần mềm ngày càng không đáng tin cậy, khó bảo trì và thực thi 1 cách nghèo nàn (performed poorly).

Do đó, để kiểm soát (control) độ phức tạp này, những kỹ thuật và phương pháp mới cần được sử dụng và công nghệ phần mềm ra đời.

1.1.1.1 Định nghĩa "Công nghệ phần mềm"

Có nhiều định nghĩa về CNPM:

- **Theo Fritz Bauer [1969]:** CNPM là sự thiết lập và sử dụng những nguyên tắc công nghệ hợp lý để đạt được những phần mềm có tính kinh tế mà đáng tin cậy và làm việc hiệu quả trên máy thực.
- **Theo IEEE [1993]:** CNPM là (1) việc áp dụng phương pháp tiếp cận có tính hệ thống, bài bản và số lượng xác định trong việc phát triển, hoạt động và bảo trì phần mềm; đó là việc áp dụng công nghệ vào phần mềm. (2) nghiên cứu các phương pháp tiếp cận ở (1).
- **Theo Roger S. Pressman:** CNPM là bộ môn tích hợp cả các quy trình, các phương pháp, các công cụ để phát triển phần mềm máy tính.

- **Theo Ian Sommerville:** CNPM là 1 lĩnh vực mà liên quan đến tất cả các khía cạnh của sản xuất phần mềm từ những giai đoạn đầu của đặc tả hệ thống đến bảo trì hệ thống sau khi nó đã được đưa vào sử dụng.

1.1.1.2 Lịch sử ra đời

Năm 1968: tại Tây Đức, hội nghị khoa học của NATO đã đưa ra từ "Software Engineering" (Công nghệ phần mềm). Bắt đầu bàn luận về khủng hoảng phần mềm (Software Crisis) và xu hướng hình thành CNPM như 1 chuyên môn riêng.

Nửa cuối 1968: IBM đưa ra chính sách phân biệt giá cả giữa phần cứng và phần mềm. Từ đó, ý thức về phần mềm ngày càng cao. Bắt đầu những nghiên cứu cơ bản về phương pháp luận lập trình.

Nửa đầu những năm 1970: Nhằm nâng cao chất lượng phần mềm, không chỉ có các nghiên cứu về lập trình, kiểm thử, mà còn có cả những nghiên cứu đảm bảo tính tin cậy trong quá trình sản xuất phần mềm.

Năm 1975: Hội nghị quốc tế đầu tiên về CNPM được tổ chức: International Conference on SE (ICSE).

Nửa sau những năm 1970: Quan tâm đến mọi pha trong quá trình phát triển phần mềm, nhưng tập trung chính ở những pha đầu. ICSE tổ chức lần 2, 3 và 4 vào 1976, 1978 và 1979.

- Nhật Bản có "Kế hoạch phát triển kỹ thuật sản xuất phần mềm" từ năm 1981.
- Cuộc "Cách tân sản xuất phần mềm" đã bắt đầu trên phạm vị các nước công nghiệp.

Nửa đầu những năm 1980: Trình độ học vấn và ứng dụng CNPM được nâng cao, các công nghệ được chuyển vào thực tế. Xuất hiện các sản phẩm phần mềm và các công cụ khác nhau làm tăng năng suất sản xuất phần mềm đáng kể.

- ICSE tổ chức lần 5 và 6 vào năm 1981 và 1982 với trên 1000 người tham dự mỗi năm.
- Nhật Bản sang "Kế hoạch phát triển các kỹ thuật bảo trì phần mềm" (1981 – 1985).

Nửa cuối những năm 1980 đến nay: Từ học vấn sang nghiệp vụ. Chất lượng phần mềm tập trung chủ yếu ở tính năng suất, độ tin cậy và tính bảo trì. Nghiên cứu tự động hóa sản xuất phần mềm.

- Nhật Bản có "Kế hoạch hệ thống công nghiệp hóa sản xuất phần mềm" (SIGMA: Software Industrialized Generator & Maintenance Aids, 1985-1990).
- Nhiều trung tâm, viện nghiên cứu CNPM ra đời. Các trường đưa vào giảng dạy SE.

1.1.1.3 Mục tiêu

Là cung cấp 1 cấu trúc cho việc xây dựng phần mềm có chất lượng cao: tính đúng đắn và độ tin cậy cao, dễ sử dụng, thân thiện với người dùng, dễ hiểu, …

1.1.2 Tiêu chuẩn của một sản phẩm phần mềm

Để đánh giá một sản phẩm phần mềm, người ta thường đánh giá theo 2 khía cạnh: chất lượng bên trong (internal qualities) và chất lượng bên ngoài (external qualities).

1.1.2.1 Chất lượng bên ngoài

Người dùng sẽ đánh giá chất lượng 1 phần mềm thông qua những yếu tố của chất lượng bên ngoài, như: tính dễ sử dụng, tính tin cậy, tính chức năng,…. Những yếu tố này sẽ bao gồm cả thuộc tính chức năng (functional attributes) và thuộc tính phi chức năng (non-functional attributes). Những thuộc tính chức năng sẽ miêu tả những chức năng mà sản phẩm phần mềm phải thực hiện (describe WHAT the product MUST do), trong khi những thuộc tính phi chức năng lại miêu tả về cách thức chương trình thực thi (describe HOW the product SHOULD be implemented).

Các yếu tố chất lương bên ngoài sẽ được xem xét thông qua một số câu hỏi đi kèm:

- Tính dễ sử dụng (usability): giao diện có thân thiện không? Các thao tác thực hiện có gần gũi không?,…

- Tính tin cậy (reliability): các chức năng của chương trình đều thực hiện đúng chứ? Các công thức tính toán đều cho ra kết quả đúng như mong muốn?

các dữ liệu được lưu vào trong DB đúng như mong muốn? phần mềm chạy ổn định?

- Tính chức năng (functionality): từng chức năng đều thực hiện đúng? Các công thức tính toán đều cho ra kết quả đúng như mong muốn? Các dữ liệu được lưu vào trong DB đúng như mong muốn?... Tính chức năng tuy cũng trả lời 1 số câu hỏi giống như tính tin cậy nhưng nó chỉ xét trên phạm vi 1 chức năng. Giả sử khi phần mềm có 5 chức năng: tính đúng đắn chỉ quan tân đến từng chức năng riêng rẽ, trong khi tính tin cậy sẽ quan tâm đến sự lien kết, rang buộc giữa các chức năng này với nhau.

- Tính bền vững (stability): phần mềm có thể hoạt động trong những điều kiện khác nhau? Trong những môi trường khác nhau?

- Tính tương thích (compatibility): phần mềm có thể dễ dàng tích hợp với các sản phẩm phần mềm khác?

- Tinh thực thi (performance): phần mềm chạy với tốc độ nhanh hay chậm? khi chạy có sử dụng nhiều tài nguyên của máy tính không: bộ nhớ, bộ xử lý,…?

Do những yếu tố chất lượng bên ngoài là do người dùng đánh giá nên nó là những yếu tố hết sức quan trọng để quyết định đến sự thành công hay thất bại của 1 phần mềm.

1.1.2.2 Chất lượng bên trong

Những yếu tố chất lượng bên trong là những yếu tố "trong suốt" với người dùng, chỉ những người phát triển (developer) mới thấy được.

Những yếu tố này là những tài liệu tham gia vào quá trình phát triển phần mềm, như: tài liệu phân tích yêu cầu, tài liệu thiết kế,…và đoạn code.Tuy yếu tố chất lượng bên ngoài là mục tiêu cuối cùng nhưng yếu tố bên trong lại giúp đỡ những người phát triển đạt được sự cải tiến về chất lượng bên ngoài.

1.1.3 Cái nhìn chung về công nghệ phần mềm

1.1.3.1 Nhân tố con người

Nhân tố con người đóng 1 vai trò hết sức quan trọng trong CNPM, đến nỗi mà viện CNPM đã phát triển "mô hình tính trưởng thành về khả năng quản lý con người" (people management capability maturity model: PM-CMM). Mô

hình này được phát triển để làm tăng tính sẵn sàng của tổ chức phần mềm trong việc đảm trách những ứng dụng phức tạp ngày càng gia tăng bằng cách thu hút, phát triển, thúc đẩy, triển khai, và giữ lại những tài năng cần thiết để cải tiến khả năng phát triển phần mềm.

A. Những người tham gia trong dự án phần mềm

Có thể chia ra thành 5 loại như sau:

A1. Senior Managers: định nghĩa ra những chiến lược kinh doanh và có những ảnh hưởng đáng kể đến dự án.

A2. Project (Technical) Managers: lập kế hoạch, thúc đẩy tinh thần làm việc, tổ chức và kiểm soát những thành viên làm công việc phần mềm.

A3. Practitioners: vận dụng những kỹ năng kỹ thuật của mình để làm sản phẩm hay ứng dụng.

A4. Customers: đưa ra những yêu cầu phần mềm và những stakeholders khác.

A5. End-users: sử dụng phần mềm khi phần mềm đưa ra sử dụng.

Để đạt được hiệu quả, đội dự án phải được tổ chức sau cho có thể sử dụng được tối đa những khả năng và kỹ năng của mỗi người. Và đây chính là công việc của team leader.

B. Project manager (PM), Project leader (PL) và Team leader (TL)

Trong một cuốn sách của mình, Jerry Weinberg đã đề nghị ra mô hình MOI cho bộ phận lãnh đạo.

Motivation (tính thúc đẩy, động lực): đây là khả năng thúc đẩy, động viên những thành viên kỹ thuật để có thể sử dụng những khả năng tốt nhất của họ cho việc sản xuất ứng dụng.

Organization (tính tổ chức): là khả năng sử dụng những tiến trình đang tồn tại để tạo ra sản phẩm cuối cùng.

Ideas or innovation (sáng kiến và sự cải tiến mới): là khả năng thúc đẩy, động viên tính sáng tạo của mọi người, mặc dù họ làm trong những giới hạn đã được thiết lập.

Đối với 1 problem gặp phải, leader và manager phải hiểu rõ được vấn đề để có thể đưa ra hướng giải quyết xác đáng và thích hợp nhất. Chính vì vậy, người leader hay manager có hiệu quả được nhấn mạnh ở 4 điểm chính sau:

Problem solving (giải quyết vấn đề): PM hiệu quả có thể phân tích những vấn đề kỹ thuật và tổ chức, đưa ra giải pháp, áp dụng những bài học từ những dự án trước trong tình trạng mới, đủ linh hoạt để thay đổi hướng giải quyết nếu sự cố gắng ban đầu để giải quyết vấn đề không thu được hiệu quả.

Managerial identity (đặc tính quản lý): 1 PM giỏi phải chịu trách nhiệm đối với dự án, phải biết nắm lấy quyền kiểm soát khi cần thiết và cho phép các thành viên phát huy khả năng kỹ thuật của mình.

Achievement (thành tích): để tối ưu hóa năng suất của đội dự án, người quản lý phải khen thưởng bằng những hành động thiết thực khi đội hoàn thành dự án, và đối với những người gây ra rủi ro cho dự án sẽ không bị trừng phạt nặng.

Influence and team building (ảnh hưởng và xây dựng đội): 1 PM hiệu quả phải hiểu được thành viên trong đội của mình đang muốn gì và có phản ứng trở lại với những nhu cầu đó.

C. Đội phần mềm (software team)

Cấu trúc đội dự án tùy thuộc vào cách quản lý của tổ chức, số lượng người tham gia vào đội, mức độ kỹ năng của từng người và độ khó của vấn đề. Mantei đề xuất hướng tổ chức đội thành 3 loại sau:

Democratic decentralized (DD) (phân quyền dân chủ): đội này sẽ không có leader lâu dài. Các leader sẽ được thay thế nhau theo từng task. Những quyết định về vấn đề và các phương pháp thực thi được đưa ra dựa vào sự nhất trí của nhóm. Việc giao tiếp, truyền thông giữa các thành viên trong nhóm là ngang hàng nhau.

Controlled decentralized (CD) (phân quyền có kiểm soát): đội loại này sẽ có 1 leader chịu trách nhiệm về những task cụ thể và leader thứ hai sẽ chịu trách nhiêm về những task con. Việc giải quyết vấn đề là hoạt động chung của cả nhóm, tuy nhiên việc thực hiện từng giải pháp lại là công việc của từng nhóm con. Việc giao tiếp, truyền thông giữa các nhóm con và các thành viên là

ngang hàng nhau. Việc truyền thông phân cấp cũng xuất hiện theo sự phân cấp quyền kiểm soát.

Controlled centralized (CC) (tập trung kiểm soát): việc giải quyết vấn đề ở mức độ cao nhất và sự điều phối trong nội bộ team được quản lý bởi TL. Việc giao tiếp, truyền thông giữa các leader và các thành viên trong nhóm là theo chiều dọc (có phân cấp).

Tuy nhiên, để lên được cấu trúc của đội thì 7 nhân tố sau nên được xem xét:

- Độ khó của vấn đề cần giải quyết.
- Kích thước của chương trình theo từng dòng code và từng đoạn chức năng.
- Thời gian mà đội sẽ làm việc cùng với nhau.
- Mức độ/ trình độ để mô hình hóa vấn đề.
- Chất lượng đòi hỏi và độ tin cậy của hệ thống xây dựng.
- Tính khắt khe của ngày giao.
- Mức độ truyền thông đòi hỏi trong nhóm.

Bởi cấu trúc tập trung hoàn thành task nhanh hơn, nên nó thích hợp nhất trong việc kiểm soát những vấn đề đơn giản. Những đội được phân quyền tạo ra nhiều giải pháp và những giải pháp này thì tốt hơn là từng cá nhân nghĩ ra. Hơn nữa, những đội này có khả năng thành công hơn trong việc giải quyết những vấn đề khó. Vì vậy, đội CD được tập trung cho việc giải quyết vấn đề, cấu trúc đội CD hay CC có thể được áp dụng thành công đối với những vấn đề đơn giản. Cấu trúc DD thì tốt nhất cho những vấn đề khó.

Cấu trúc CC hay CD thường được giao cho những dự án rất lớn khi những nhóm con có thể hoàn thành task dễ dàng.

Thời gian sống của đội ảnh hưởng đến tinh thành làm việc của đội. Cấu trúc đội DD cho tinh thần làm việc và độ hài lòng công việc cao và vì vậy tốt cho đội là sống trong 1 thời gian dài.

Để đội đạt được sự thực thi cao thì:

- Các thành viên trong đội phải tin tưởng lẫn nhau.
- Sự phân phối các kỹ năng phải thích hợp với vấn đề.

- Những khuyết điểm trong đội phải được loại trừ để duy trì sự gắn kết của đội.

D. Vấn đề điều phối và truyền thông (coordination and communication issue)

Những phần mềm ngày nay có những đặc tính sau:

- **Scale** (tính tỷ lệ): tỷ lệ của sự cố gắng phát triển phần mềm thì lớn, dẫn đến độ phức tạp, sự mơ hồ và độ khó đáng kể trong việc điều hướng những thành viên trong đội.
- **Uncertainty** (tính bất định): sự bất định của phần mềm thì phổ biến, dẫn đến chuỗi thay đổi tiếp diễn mà ảnh hưởng đến đội dự án.
- **Interoperability** (khả năng tương tác): trở thành đặc trưng chính của nhiều hệ thống. Phần mềm mới phải tương tác với phần mềm cũ và phù hợp với những ràng buộc đã được nêu ra trước được chỉ ra bởi hệ thống và sản phẩm.

Để giải quyết những đặc tính này 1 cách hiệu quả, đội CNPM phải xây dựng những phương pháp hiêu quả cho việc điều hướng các thành viên. Để hoàn thành điều đó, cơ cấu truyền thông/ giao tiếp thân mật và lịch sự giữa các thành viên và giữa các đội phải được thiết lập.

Kraul và Streeter xem xét 1 tập hợp những kỹ thuật điều hướng dự án và phân loại thành 5 loại sau:

- **Formal, impersonal approaches** (phương thức lịch sự, khách quan): gồm tài liệu kỹ thuật phần mềm và giao nộp (source code), ghi chú kỹ thuật, các mốc dự án, lịch biểu và những công cụ kiểm soát dự án, những yêu cầu thay đổi và những tài liệu liên quan, những báo cáo lưu vết lỗi, và dữ liệu dự án.
- **Formal, interpersonal procedures** (những thủ tục lịch sự, giữa các cá nhân): tập trung vào những hoạt động đảm bảo chất lượng, áp dụng cho những sản phẩm công việc CNPM, bao gồm review meetings, design and code inspections.

- **Informal, interpersonal procedures** (những thủ tục thân mật, giữa các cá nhân): gồm những cuộc họp nhóm để phổ biến thông tin, giải quyết vấn đề và "sự sắp xếp yêu cầu và đội phát triển".
- **Electronic communication** (truyền thông điện tử): gồm email, bảng tập san điện tử, hội nghị qua video.
- **Interpersonal networking** (mạng giữa các cá nhân): gồm những cuộc thảo luận thân mật giữa các thành viên trong và ngoài dự án (những thành viên có kinh nghiệm và sự hiểu biết có thể hỗ trợ những thành viên trong đội dự án).

1.1.3.2 Các loại phần mềm

Ngày nay, phần mềm được áp dụng ngày càng rộng rãi, trong bất kỳ một hoàn cảnh nào. Nội dung thông tin và sự rõ rang là những nhân tố quan trọng trong việc xác định bản chất của một phần mềm ứng dụng.

Nội dung thông tin đề cập đến ý nghĩa và hình thức thông tin vào và ra. Các phần mềm ứng dụng trong doanh nghiệp sử dụng dữ liệu đầu vào có cấu trúc cao (database) để tạo ra những báo cáo có định dạng.

Trong khi đó, sự rõ ràng của thông tin đề cập đến những dự đoán của trình tự và thời gian của thông tin. Một chương trình phân tích kỹ nghệ chấp nhận dữ liệu có trình tự đã được xác định trước và thực hiện những thuật toán phân tích mà không gián đoạn và đưa ra kết quả trong báo cáo hay những định dạng đồ họa.

Dựa vào mục đích sử dụng và bản chất của phần mềm mà phân mềm được chia ra thành những loại sau:

- System software (phần mềm hệ thống): là một tập hợp các chương trình được viết để phục vụ các chương trình khác, như: drivers, compilers, editors,…
- Real-time software (phần mềm thời gian thực): là phần mềm giám sát, phân tích và kiểm soát những sự kiện xảy ra trong thế giới thực khi nó xảy ra. Các yếu tố của phần mềm thời gian thực là: (1) thu thập thông tin từ bên ngoài (2) phân tích và chuyển đổi thông tin vào trong ứng dụng

(3) kiểm soát dữ liệu đầu ra (4) giám sát các sự kiện bên ngoài để duy trì sự phản ánh một cách chính xác.

- Business software (phần mềm nghiệp vụ): là phần mềm xử lý các thông tin của doanh nghiệp. Các hệ thống rời rạc (tính lương, các khoản thu chi,…) đã phát triển thành phần mềm MIS(Management Information System) truy cập một số lương lớn thông tin của doanh nghiệp.
- Engineering and scientific software (phần mềm khoa học và ứng dụng): là phần mềm sẽ sử dụng các thuật toán để phân tích và chuyển đổi các con số.
- Embedded software (phần mềm nhúng): là phần mềm nằm trong ROM (read-only memory) và dùng để kiểm soát sản phẩm và hệ thống.
- Personal computer software (phần mềm máy tính cá nhân)
- Web-based software (phần mềm ứng dụng web)
- Artificial Intelligence software (phần mềm trí tuệ nhân tạo): là phần mềm sử dụng các thuật toán không số (non-numerical algorithms) để giải quyết các vấn đề phức tạp, không tuân theo quy luật nào, như: hệ thống nhận dạng mô hình (âm thanh, hình ảnh), mạng lưới thần kinh nhân tạo (artificical neural networks).

1.1.3.3 Tiến trình phần mềm (software process)

A. Tổng quan về tiến trình phần mềm

Tiến trình phần mềm là một tập hợp các hoạt động để sản xuất phần mềm. Những hoạt động này liên quan đến việc phát triển phần mềm từ những thứ lộn xộn vào trong ngôn ngữ lập trình chuẩn như java, C. Tuy nhiên, những phần mềm mới đa số được phát triển bằng cách mở rộng hay nâng cấp những phần mềm đang tồn tại.

Những tiến trình phần mềm thì phức tạp và cũng giống như tất cả những tiến trình thông minh và có tính sáng tạo khác, tiến trình phần mềm cũng dựa vào những người đưa ra quyết định và sự chuẩn đoán. Bởi sự cần thiết của việc chuẩn đoán và sự sáng tạo nên họ đang ra sức cố gắng tự động hóa những tiến trình phần mềm đáp ứng được sự thành công có giới hạn. Những công cụ Công Nghệ Phần Mềm Hỗ Trợ Máy Tính (Computer-aided software engineering

CASE) có thể hỗ trợ một số hoạt động tiến trình. Tuy nhiên, ít nhất là trong một vài năm nữa, vẫn không có khả năng cho sự tự động hóa mở rộng hơn, ở đó phần mềm đảm nhận việc thiết kế của những kỹ sư liên quan đến tiến trình phần mềm.

Một lý do, sự hiệu quả của những công cụ CASE bị giới hạn do có quá nhiều tiến trình phần mềm. Không có tiến trình lý tưởng nào và nhiều tổ chức lại phát triển những tiến trình của riêng họ để phát triển phần mềm. Những tiến trình này được đưa ra để khai thác khả năng của con người trong tổ chức và những đặc tính cụ thể của hệ thống mà họ đang phát triển. Đối với một số hệ thống, như những hệ thống quan trọng, một tiến trình phát triển được cấu trúc thì được đòi hỏi. Đối với những hệ thống thương mại, với những yêu cầu bị thay đổi một cách nhanh chóng, một tiến trình nhanh nhẹn và linh hoạt thì có vẻ hiệu quả hơn.

Mặc dù có nhiều tiến trình phần mềm, một số hoạt động nền tảng thì phổ biến chung cho tất cả các tiến trình.

1. Đặc tả phần mềm (Software specification): chức năng của phần mềm và những ràng buộc trong hoạt động của nó phải được định nghĩa.
2. Thiết kế và thực thi phần mềm (Software design and implementation): phần mềm đáp ứng đặc tả phải được tạo ra.
3. Chứng nhận phần mềm: phần mềm phải được chứng nhận để đảm bảo rằng nó làm những gì mà khách hàng muốn.
4. Tiến triển phần mềm: phần mềm phải tiến triển để đáp ứng những nhu cầu thay đổi của khách hàng.

Mặc dù không có một tiến trình phần mềm nào lý tưởng, nhưng lại có phạm vi cho việc cải tiến những tiến trình phần mềm trong các tổ chức. Những tiến trình này có thể bao gồm những kỹ thuật chưa được cập nhật. Thực vậy, nhiều tổ chức vẫn không có được sự thuận lợi nào của những phương pháp công nghệ phần mềm trong sự phát triển phần mềm của họ.

Những tiến trình phần mềm có thể được cải tiến bằng sự chuẩn hóa tiến trình ở nơi mà sự đa dạng trong các tiến trình phần mềm trên tổ chức thì được giảm bớt. Điều này dẫn đến sự truyền thông được cải tiến và sự giảm bớt trong

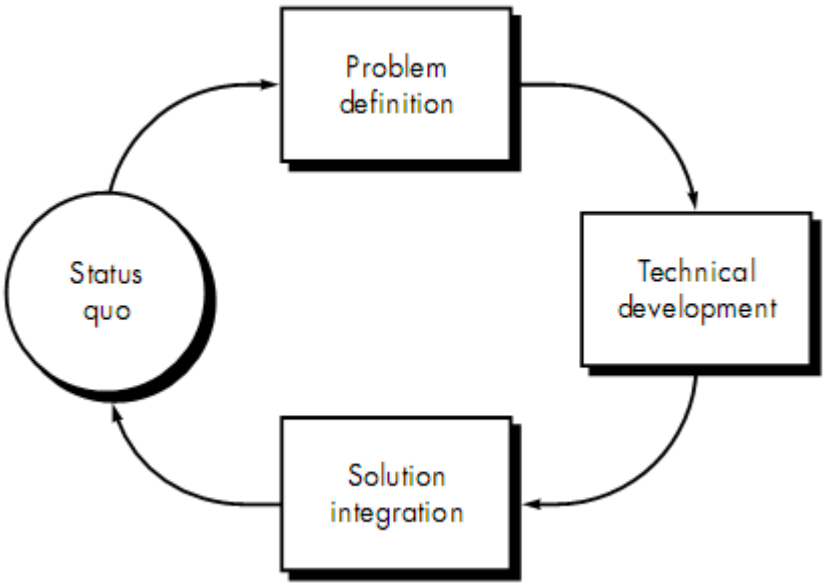
thời gian huấn luyện và làm cho sự hỗ trợ tiến trình được tự động hóa tiết kiệm hơn. Sự chuẩn hóa cũng là một bước đầu quan trọng trong việc giới thiệu những phương pháp và kỹ thuật công nghệ phần mềm mới và áp dụng công nghệ phần mềm được tốt.

B. Tổng quan về những mô hình tiến trình phần mềm (software process models)

Mô hình tiến trình phần mềm là đại diện trừu tượng cho tiến trình phần mềm. Mỗi mô hình tiến trình đại diện cho một tiến trình từ một tiến độ cụ thể, vì vậy chỉ cung cấp cho bạn được môt phần thông tin về tiến trình mà thôi.

Những mô hình chung không phải là sự định nghĩa cuối cùng của những tiến trình phần mềm. Đúng hơn, chúng là sự trừu tượng của tiến trình mà có thể được dùng để giải thích những phương pháp khác nhau của sự phát triển phần mềm. Chúng như một khung tiến trình mà có thể được mở rộng và gắn vào để tạo ra nhiều tiến trình CNPM hơn.

Tất cả sự phát triển phần mềm có thể được đặc trưng như một vòng lặp giải quyết vấn đề gồm 4 giai đoạn riêng biệt sau: status quo (trích dẫn tình trạng), problem definition (định nghĩa vấn đề), technical development (phát triển kỹ thuật), solution integration (tích hợp giải pháp).



Status quo: miêu tả tình trạng hiện tại của công việc.

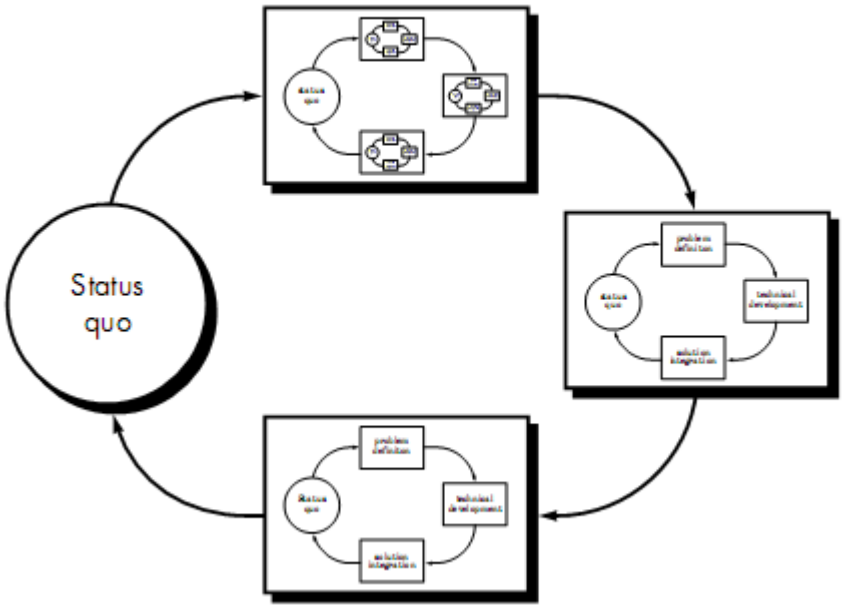
Problem definition: chỉ ra những vấn đề cụ thể cần được giải quyết.

Technical development: giải quyết vấn đề bằng cách áp dụng một số kỹ thuật công nghệ.

Solution integration: đưa ra kết quả của việc giải quyết vấn đề cho những người yêu cầu thông qua tài liệu, chương trình, dữ liệu, chức năng nghiệp vụ mới.

Các pha và các bước chung của CNPM đều có thể dễ dàng nối (map) với những giai đoạn này.

Vòng lặp giải quyết vấn đề này có thể được áp dụng cho các cấp độ khác nhau của công việc CNPM. Nó có thể được sử dụng ở cấp độ marco khi mà ứng dụng chỉ đang trong giai đoạn xem xét, ở cấp độ mid-level khi những thành phần chương trình đang được phác thảo, và thậm chí là ở những dòng code của cấp độ code. Vì vậy, sự đại diện phân dạng có thể cung cấp một cái nhìn lý tưởng hóa của tiên trình. Do đó, trong mỗi giai đoạn của vòng lặp giải quyết vấn đề chứa đựng một vòng lặp giải quyết vấn đề đồng nhất.



Tuy nhiên, thật khó để chia những hoạt động này ra một cách gọn gàng như ở trên trong thực tế.

Raccoon đề nghị một "mô hình hỗn loạn" (Chaos model) mà miêu tả "sự phát triển phần mềm như là một sự liên tục từ người dùng đến các nhà phát triển và đến công nghệ" ("software development [as] a continuum from the user to the developer to the technology"). Khi những tiến trình công việc hướng

đến một hệ thống hoàn chỉnh, các giai đoạn được áp dụng một cách đệ quy với những nhu cầu của người dùng và các đặc tả kỹ thuật của phần mềm của các nhà phát triển.

Hiện nay có nhiều mô hình CNPM khác nhau và mỗi cách được đặc trưng trong một cách mà hỗ trợ một cách lý tưởng trong việc kiểm soát và điều phối một dự án phần mềm.

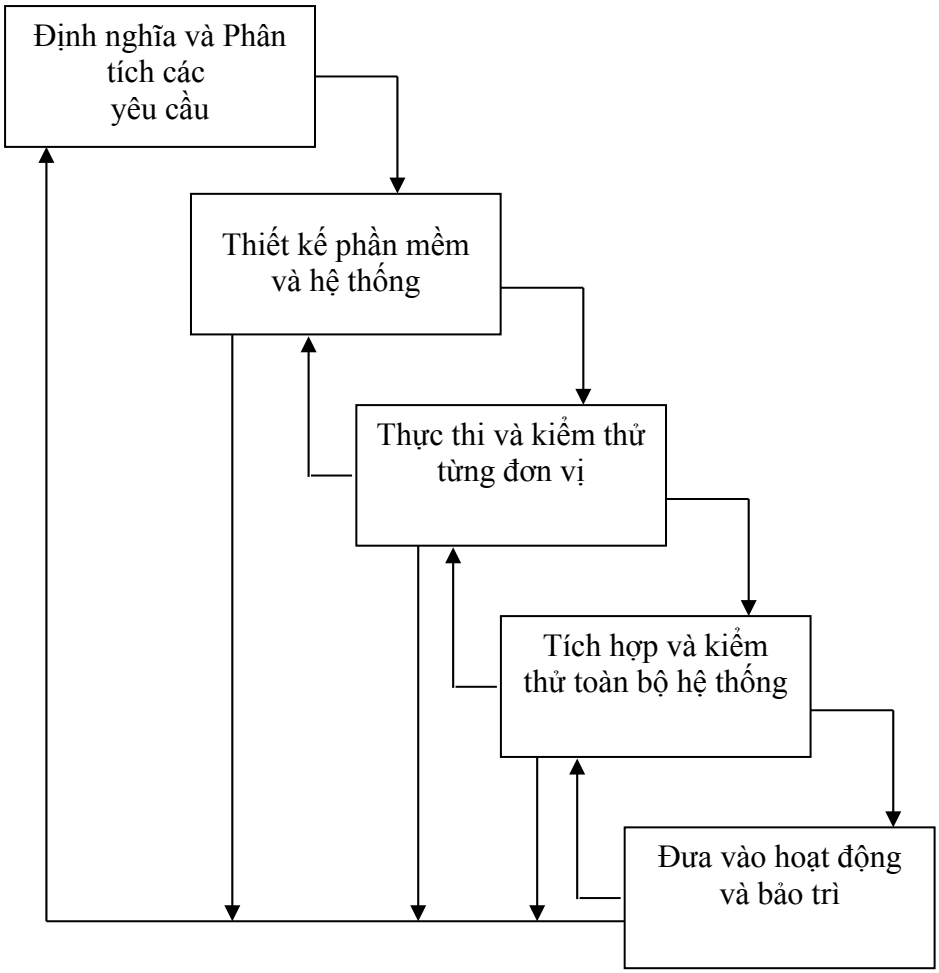
Một số mô hình phần mềm:

- Mô hình thác nước (the waterfall model)
- Mô hình mẫu (the prototyping model)
- Mô hình phát triển ứng dụng nhanh RAD (the rapid application development model)
- Mô hình tiến hóa (evolutionary development model)
    + Mô hình gia tăng (incremental model)
    + Mô hình xoắn ốc (the spiral model)
    + Mô hình xoắn ốc WINWIN (the WINWIN spiral model)

Trong thực tế những mô hình này không loại trừ lẫn nhau mà thường được kết hợp lẫn nhau đặc biệt là cho sự phát triển những hệ thống lớn. Những hệ thống con trong hệ thống lớn có thể được phát triển bằng cách sử dụng những phương pháp khác nhau.

C. Các mô hình phần mềm (software models)

C1. Mô hình thác nước (Waterfall model)



Đây là mô hình đầu tiên của CNPM, nó được lấy từ nhiều tiến trình kỹ nghệ phần cứng. Do tính phân tầng đi từ pha này đến pha khác, nên mô hình này được gọi là mô hình thác nước, mô hình tuyến tính hay chu kỳ sống của phần mềm. Những giai đoạn cơ bản của mô hình đều phù hợp với những hoạt động phát triển chung:

- Định nghĩa và phân tích yêu cầu (Requirements analysis and definition): hệ thống dịch vụ, những rang buộc và những mục tiêu được thiết lập bằng cách thảo luận với người dung hệ thống. Sau đó chúng sẽ được định nghĩa và xem xét như đặc tả hệ thống mà cả nhà phát triển và người dung đều có thể hiểu được.
- Thiết kế phần mềm và hệ thống (System and software design): giai đoạn này sẽ phân yêu cầu ra thành những yêu cầu của hệ thống hay của phần mềm. Nó sẽ xây dựng một cấu trúc hệ thống tổng thể. Thiết kế phần

mềm bao gồm việc định nghĩa và miêu tả sự trừ tượng hóa hệ thống phần mềm và mối quan hệ của chúng.

- Thực thi và kiểm thử từng đơn vị (Implementation and testing unit): trong suốt giai đoạn này, bản thiết kế phần mềm được xem như là một tập hợp các chương trình đơn vị. Kiểm thử đơn vị sẽ chứng nhận rằng từng đơn vị có đúng như với đặc tả không.
- Tích hợp và kiểm thử toàn bộ hệ thống (Integration and system testing): từng đơn vị của chương trình thì được tích hợp và kiểm thử như một hệ thống hoàn chỉnh để đảm bảo chương trình đáp ứng được yêu cầu khách hang. Sau khi kiểm thử, hệ thống phần mềm được giao đến khách hang.
- Đưa vào hoạt động và bảo trì (Operation and maintance): đây là giai đoạn lâu nhất trong chu kỳ sống của phần mềm. Hệ thống được cài đặt và đưa vào sử dụng thực tế. Công viêc bảo trì lien quan đến việc sửa lỗi chưa được phát hiện ở những giai đoạn trước và nâng cấp hệ thống.

Từng giai đoạn đều tạo ra 1 hay nhiều tài liệu cần phải được chấp thuận. Những giai đoạn sau không nên bắt đầu cho đến khi giai đoạn trước nó được hoàn thành. Trong thực tế, những giai đoạn này gối lên nhau và phản hồi thông tin với nhau. Suốt giai đoạn thiết kế, những vấn đề về yêu cầu được chỉ ra. Suốt giai đoạn code, những vấn đề về thiết kế được đưa ra ,…. Tiến trình phần mềm không phải là 1 mô hình tuyến tính đơn giản mà là 1 chuỗi lặp lại của những hoạt động phát triển phần mềm.

Suốt giai đoạn cuối của chu trình sống (operation and maintance), phần mềm được đưa vào sử dụng. Những lỗi và sự thiếu sót trong yêu cầu được phát hiện. Những lỗi chương trình và thiết kế xuất hiện và nhu cầu cần có một chức năng mới được đưa ra.

Điểm thuận lợi của mô hình thác nước là các tài liệu được đưa ra ở từng giai đoạn và các tài liệu này phù hợp với các mô hình tiến trình kỹ nghệ khác. Vấn đề chính yếu của nó đó là sự phân vùng không linh hoạt của nó để đưa vào từng giai đoạn riêng biệt. Sự cam kết phải được làm ở giai đoạn sớm của tiến

trình, điều này gây khó khăn cho sự hồi đáp những thay đổi trong yêu cầu của khách hàng.

Do đó, mô hình thác nước chỉ nên được dùng khi yêu cầu đã được hiểu rõ ràng và sự thay đổi yêu cầu trong suốt giai đoạn phát triển phần mềm là không nghĩ đến. Thực tế, mô hình thác nước vẫn còn được sử dụng nhiều, đặc biệt khi dự án phần mềm là một phần của dự án kỹ nghệ hệ thống lớn hơn.

Tóm lại, những điểm yếu của mô hình thác nước như sau:

- Thực tế các dự án ít khi nào tuân theo dòng tuần tự của mô hình mà thường lặp lại.
- Khách hàng hiếm khi nói rõ ràng các yêu cầu.
- Khách hàng thường phải chờ lâu mới thấy được phiên bản đầu tiên của phần mềm.

C2. Mô hình chữ V (V model)

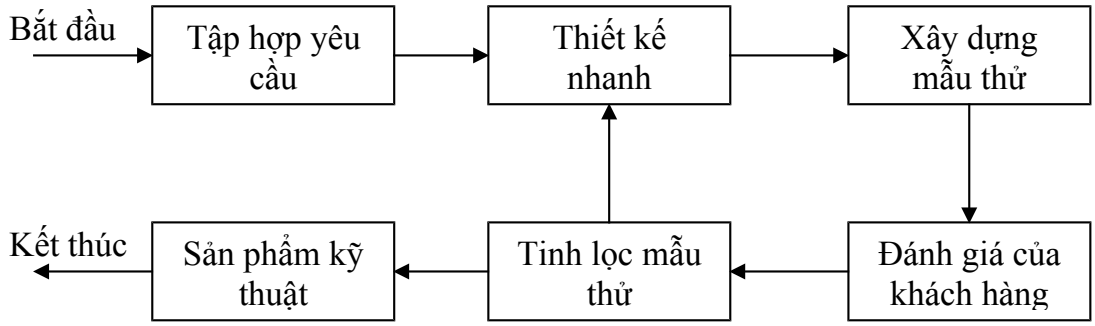
Sinh viên tự nghiên cứu.

C3. Mô hình mẫu (Prototyping model)

Thông thường, khách hàng định nghĩa 1 tập hợp các mục tiêu chung của phần mềm, nhưng lại không chỉ rõ những yêu cầu đầu vào, xử lý, đầu ra. Trong những trường hợp đó, nhà phát triển có thể không đảm bảo được tính hiệu quả của thuật toán, khả năng tương thích của hệ điều hành hay hình thức tương tác giữa con người và máy tính cần có. Trong những tình huống này và nhiều tình huống khác, mô hình mẫu là phương pháp lựa chọn tốt nhất.

Mục tiêu của việc phát triển dựa vào mẫu thử là giải quyết 2 hạn chế đầu tiên của mô hình thác nước. Ý tưởng cơ bản ở đây là thay cho việc chốt lại yêu cầu trước khi thiết kế hay code, một mẫu thử có tính sử dụng 1 lần được xây dựng để hiểu yêu cầu. Mẫu thử được xây dựng dựa vào những yêu cầu được biết hiện tại. Việc phát triển mẫu thử này dĩ nhiên cũng trải qua các giai đoạn design, coding, testing. Nhưng mỗi pha không được thực hiện một cách chu đáo. Bằng việc sử dụng mẫu thử, khách hàng sẽ có được cảm giác một hệ thống thực sự, vì khi tương tác với mẫu thử, khách hàng sẽ biết được rõ hơn những yêu cầu của hệ thống mong muốn.

Mẫu thử là ý tưởng đầy hấp dẫn cho những hệ thống lớn, phức tạp và tự động. Trong những trường hợp này, khách hàng được để cho lập kế hoạch làm việc với mẫu thử, cung cấp những dữ liệu đầu vào, qua đó giúp xác định yêu cầu cho hệ thống.



Tóm lại, mô hình mẫu thử có những ưu, nhược điểm sau:

Ưu điểm:

- Người sử dụng sớm hình dung ra chức năng và đặc điểm của hệ thống thông qua giao diện của ứng dụng.
- Cải thiện sự liên lạc giữa nhà phát triển và người sử dụng.

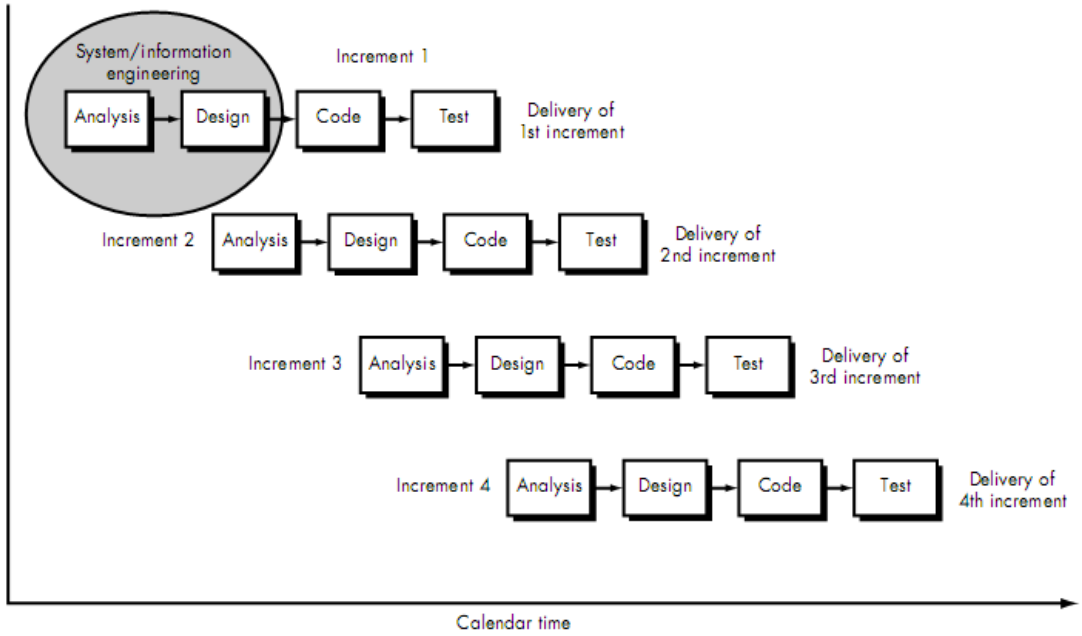
Nhược điểm:

- Khi mẫu thử không chuyển tải được hết các chức năng, đặc điểm của hệ thống phần mềm thì người sử dụng có thể thất vọng và mất đi sự quan tâm đến hệ thống sẽ được phát triển.
- Mẫu thử thường được làm nhanh, thậm chí vội vàng theo kiểu "làm – sửa" và có thể thiếu sự phân tích đánh giá một cách cẩn thận tất cả khía cạnh liên quan đến hệ thống cuối cùng.

C4. Mô hình tăng dần (Incremental model)

Mô hình tăng dần kết hợp những yếu tố của mô hình tuyến tính (mô hình thác nước) và ý tưởng lặp của chế bản mẫu. Mô hình tăng dần phân phối phần mềm ra thành các mảnh (piece) nhỏ nhưng có tính tiện dụng, gọi là "các phần tăng trưởng" (increments). Nhìn chung, mỗi phần tăng trưởng được xây dựng trên những cái đã được phân phối.

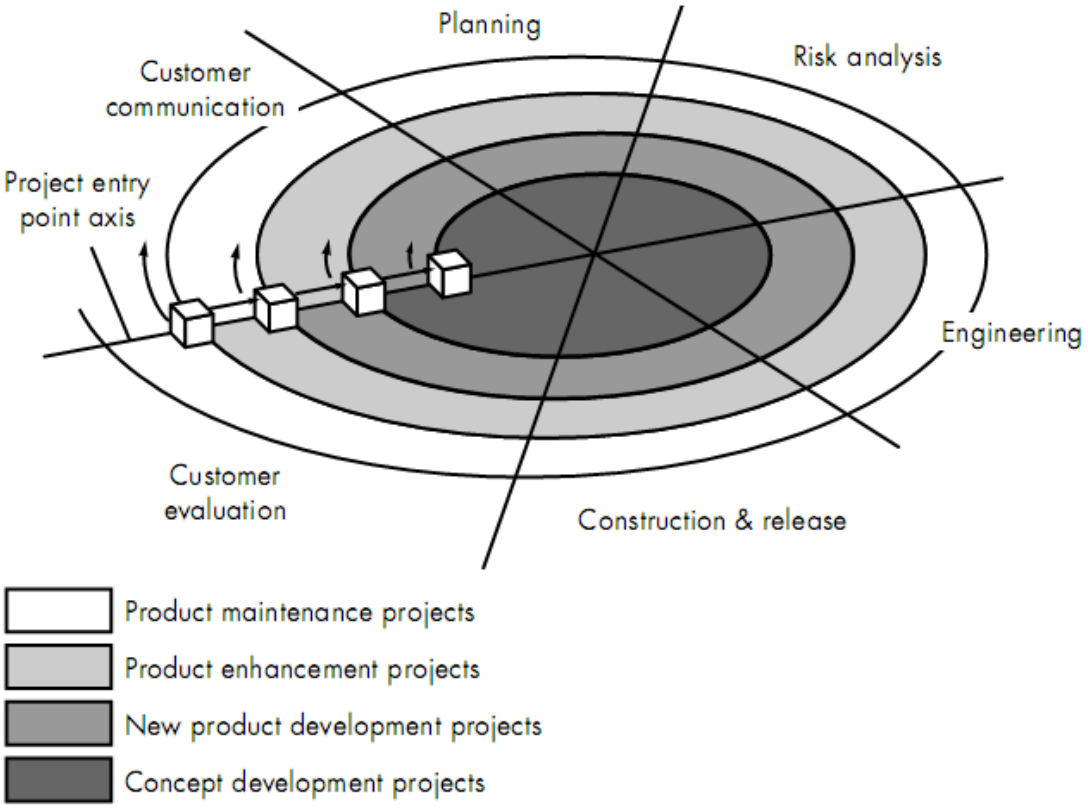
Khi mô hình tăng dần được sử dụng, phần tăng trưởng đầu tiên gọi là sản phẩm lõi. Sản phẩm lõi với những chức năng cơ bản nhất sẽ được lên kế hoạch và thực hiện đầu tiên. Kế hoạch này cũng chỉ ra việc sửa đổi sản phẩm lõi để đáp ứng nhu cầu của khách hàng và giao các chức năng, tính năng thêm vào tốt hơn. Tiến trình này được lặp lại sau việc giao mỗi lần tăng trưởng cho đến khi sản phần hoàn chỉnh được tạo ra.



C5. Mô hình xoắn ốc (Spiral model)

Mô hình xoắn ốc được Boehm đề xuất vào năm 1988. Đây là mô hình kết hợp tính lặp của mô hình mẫu với những khía cạnh được kiểm soát và hệ thống hóa của mô hình tuyến tính.

Mô hình xoắn ốc được chia ra thành nhiều hoạt động chính, điển hình là từ 3 đến 6 hoạt động. Sau đây là mô hình với 6 hoạt động chính:



Customer communication (giao tiếp với khách hàng): thiết lập sự giao tiếp có hiệu quả giữa người phát triển và khách hàng.

Planning (lập kế hoạch): xác định tài nguyên, thời hạn và các thông tin khác.

Risk analysis (phân tích rủi ro): xem xét cả rủi ro kỹ thuật và rủi ro quản lý.

Engineering (kỹ thuật): xây dựng một hay một số biểu diễn của ứng dụng.

Construction & release (xây dựng và phát hành): xây dựng, kiểm thử, cài đặt và cung cấp hỗ trợ người dùng (tư liệu, huấn luyện,…).

Customer evaluation (đánh giá của khách hàng): nhận các phản hồi của người dùng về biểu diễn phần mềm trong giai đoạn kỹ thuật và cài đặt.

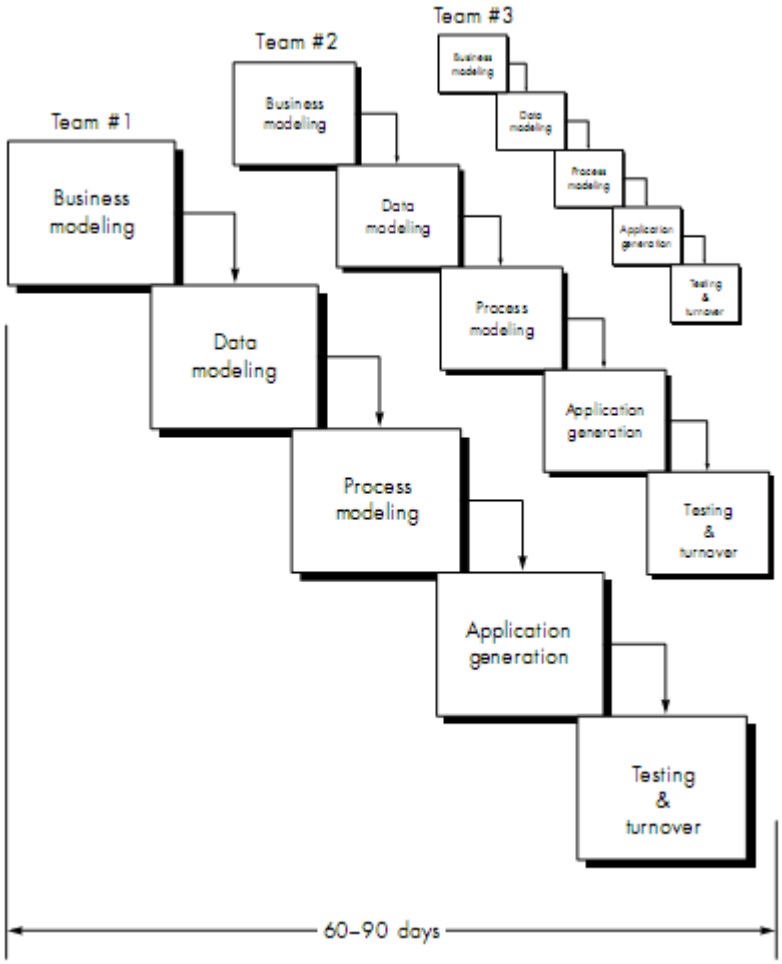
Khi tiến trình bắt đầu, đội CNPM sẽ di chuyển vào trong xoắn ốc theo chiều kim đồng hồ, bắt đầu là ở tâm. Mô hình xoắn ốc được áp dụng xuyên suốt chu kỳ sống của phần mềm. Mỗi khối lập phương đặt tại trục tọa độ đại diện cho điểm bắt đầu của mỗi loại dự án khác nhau. "Concept development projects" bắt đầu ở lõi của xoắn ốc và tiếp tục cho đến khi sự phát triển khái

niệm hoàn thành. Nếu khái niệm được phát triển thành sản phẩm thật, tiến trình sẽ tiến đến hình lập phương tiếp theo, "new development projects" được khởi tạo. Sản phẩm mới sẽ tiến triển suốt một số lượng lớn những lần lặp quanh xoắn ốc cho đến khi nào phần mềm không còn sử dụng nữa. Bất cứ khi nào có sự thay đổi thì tiến trình sẽ bắt đầu ngay tại vị trí hình lập phương thích hợp.

Mô hình xoắn ốc tốt cho các hệ phần mềm quy mô lớn. Nó sử dụng mô hình mẫu như là cơ chế loại trừ lỗi, cho phép nhà phát triển áp dụng mô hình mẫu tại mỗi chu trình phát triển. Tuy nhiên, nó lại khó thuyết phục khách hàng là có thể kiểm soát được sự tiến hóa của phần mềm, vì nó đòi hỏi phải có kỹ năng cao trong việc đánh giá lỗi. Cuối cùng, mô hình xoắn ốc vẫn chưa được sử dụng phổ biến như mô hình thác nước và mô hình mẫu.

C6. Mô hình RAD (Rapid Application Development - RAD model)

Mô hình RAD là mô hình tiến trình phát triển phần mềm tăng dần mà nhấn mạnh vào chu trình phát triển cực ngắn bằng cách sử dụng sự xây dựng dựa trên các thành phần. Nếu các yêu cầu được hiểu tốt và phạm vi dự án được xác định, tiến trình RAD cho phép đội phát triển tạo ra một hệ thống chức năng đầy đủ trong một khoảng thời gian ngắn (60 – 90 ngày).



Được sử dụng chính yếu cho các ứng dụng hệ thống thông tin, RAD bao gồm những pha sau:

Mô hình nghiệp vụ (Business modeling): dòng thông tin trong các chức năng nghiệp vụ được mô hình hóa bằng cách trả lời các câu hỏi:

- Thông tin nào sử dụng trong tiến trình nghiệp vụ?
- Thông tin nào được tạo ra?
- Ai tạo ra thông tin này?
- Thông tin này sẽ chuyển tiếp đến đâu?
- Ai sẽ xử lý những thông tin này?
- …

Mô hình dữ liệu (Data modeling): những thông tin được tập hợp từ mô hình nghiệp vụ được lọc vào trong một tập hợp các đối tượng dữ liệu mà cần thiết để hỗ trợ cho nghiệp vụ (business). Các đặc tính của mỗi đối tượng được đồng nhất và mối quan hệ giữa các đối tượng này được định nghĩa.

Mô hình tiến trình (Process modeling): các đối tượng dữ liệu trong pha mô hình dữ liệu được chuyển tiếp để đạt được dòng dữ liệu cần thiết cho việc thực thi chức năng nghiệp vụ. Sự miêu tả tiến trình được tạo ra để thêm, cập nhật, xóa, và lấy lại một đối tượng dữ liệu.

Thế hệ ứng dụng (Application Generation): dùng các kỹ thuật thế hệ thứ 4 để tạo phần mềm từ các thành phần có sẵn hoặc tạo ra các thành phần có thể tái sử dụng lại sau này. Dùng các công cụ tự động để xây dựng phần mềm.

Kiểm chứng và xoay vòng (Testing and turnover): kiểm thử các thành phần mới và kiểm chứng mọi giao diện (các thành phần cũ đã được kiểm chứng và sử dụng lại).

Giống như tất cả các mô hình khác, mô hình RAD có những ưu, khuyết điểm sau:

Ưu điểm:

- Lược giảm được thời gian phát triển và tái sử dụng các thành phần nên đẩy nhanh được tốc độ phát triển.
- Tất cả các chức năng được mô hình hóa nên dễ dàng làm việc.

Nhược điểm:

- Đối với những dự án lớn, RAD đòi hỏi các thành viên trong đội phải có kỹ năng cao.
- Cả người dùng cuối và các nhà phát triển nên được cam kết hoàn thành hệ thống trong một khung thời gian được rút ngắn. Nếu sự cam kết này thiếu/ không có, RAD sẽ bị thất bại.
- RAD không thích hợp khi những rủi ro kỹ thuật là cao.
- Không phải tất cả các ứng dụng đều thích hợp với RAD. Nếu hệ thống không được mô hình hóa phù hợp, RAD sẽ mơ hồ. Nếu hiệu suất cao là một vấn đề và đạt được xuyên suốt việc điều chỉnh giao diện đến những thành phần hệ thống, RAD có thể không làm việc.

1.2 Quản lý dự án: Đánh giá phần mềm

1.2.1 Khái quát về tiến trình quản lý dự án

Một dự án phần mềm thường gặp phải những vấn đề sau:

- Thời gian thực hiện dự án vượt mức dự kiến. (delivered late)

- Chi phí thực hiện dự án vượt mức dự kiến. (cost more than originally estimated)
- Kết quả của dự án không như yêu cầu của khách hàng. (fail to meet its requirements)

Người quản lý tốt không đảm bảo được thành công của dự án, nhưng người quản lý tồi lại thường gây thất bại cho dự án. Sau đây là một vài trách nhiệm của người quản lý:

- Lên kế hoạch và lập lịch cho việc phát triển phần mềm.
- Giám sát công việc để đảm bảo phần mềm đáp ứng được những tiêu chuẩn đòi hỏi.
- Giám sát tiến trình để kiểm tra việc phát triển có đúng thời hạn và trong ngân sách hay không.

Tuy nhiên, để có thể miêu tả công việc chuẩn của người quản lý lại là điều không thể vì nó phụ thuộc vào tổ chức và sản phẩm phần mềm được phát triển. Sau đây là các hoạt động của người quản lý:

- Viết đề xuất (Proposal writing).
- Lên kế hoạch và lập lịch biểu cho dự án (Project planning and scheduling).
- Chi phí cho dự án (Project cost).
- Giám sát và xem xét dự án (Project monitoring and reviews).
- Lựa chọn và đánh giá nhân viên (Personnel selection and evaluation).
- Viết và trình diễn báo cáo (Report writing and presentations).

1.2.2 Các hoạt động chính trong quản lý dự án phần mềm

1.2.2.1 Xác định dự án phần mềm cần thực hiện

Xác định yêu cầu phần mềm

- Trước tiên cần xác định các yêu cầu chức năng và phi chức năng của phần mềm. Chỉ ra mục đích và cách thức thực hiện của phần mềm.
- Xác định tài nguyên cần thiết để xây dựng phần mềm, gồm: nhân tố con người, các thành phần, phần mềm có thể sử dụng lại, các phần cứng hoặc công cụ có sẵn cần dùng đến.
- Ước lượng chi phí và thời gian để thực hiện dự án.

- Xem xét tính khả thi của dự án.

Viết tài liệu dự án

Viết đề án là một kỹ năng mà không phải ai cũng có, được tính lũy trong thực tiễn và kinh nghiệm.

Đây là quá trình xây dựng tài liệu mô tả dự án để xác định phạm vi của dự án, trách nhiệm của những người tham gia dự án; là cam kết giữa người quản lý dự án, người tài trợ và khách hàng. Nội dung của tài liệu thường gồm những phần sau:

- Mục đích và mục tiêu của dự án: tin học hóa hoạt động nào trong quy trình nghiệp vụ của khách hàng, lợi ích phần mềm đem lại,…
- Phạm vi dự án: các hoạt động nghiệp vụ cấn tin học hóa,…
- Nguồn nhân lực tham gia dự án: những người liên quan tới dự án.
- Thời gian thực hiện dự án: ngày nghiệm thu, ngày bàn giao,…
- Kinh phí: kinh phí thực hiện trong từng giai đoạn của dự án.
- Ràng buộc công nghệ phát triển: công nghệ nào được phép sử dụng để thực hiện dự án.
- Xác nhận của các bên liên quan tới dự án.

1.2.2.2 Lập kế hoạch dự án

Kế hoạch dự án chính là sơ đồ các nhiệm vụ, thời gian và các mối quan hệ giữa chúng. Việc lên kế hoạch, nói chung, thường gồm các bước sau:

- Liệt kê các nhiệm vụ: gồm các nhiệm vụ phát triển ứng dụng, các nhiệm vụ đặc trưng của dự án, các nhiệm vụ về tổ chức giao diện, sự xem xét lại và các việc phê chuẩn.
- Xác định nhân viên dựa vào kỹ năng và kinh nghiệm.
- Ấn định thời gian hoàn thành cho mỗi công việc bằng các tính toán thời gian hợp lý nhất cho mỗi công việc.
- Thương lượng, thỏa thuận và cam kết ngày bắt đầu và kết thúc công việc.
- Xác định các giao diện của ứng dụng, đặt kế hoạch cho việc thiết kế giao diện chi tiết.

Các loại kế hoạch trong dự án:

Loại kế hoạch	Mô tả
Kế hoạch chất lượng	Miêu tả những phương thức và tiêu chuẩn chất lượng được sử dụng trong dự án.
Kế hoạch xác nhận	Miêu tả cách thức, tài nguyên và lịch biểu cho việc xác nhận dự án.
Kế hoạch quản lý cấu hình	Miêu tả cách thức và cấu trúc quản lý cấu hình được sử dụng.
Kế hoạch bảo trì	Dự đoán những yêu cầu bảo trì của hệ thống, chi phí bảo trì và công sức yêu cầu.
Kế hoạch phát triển nhân lực	Miêu tả cách mà kỹ năng và kinh nghiệm của các thành viên trong đội được phát triển.

1.2.2.3 Giám sát quá trình thực hiện dự án

Người quản lý phải theo dõi tiến độ thực hiện dự án và so sánh tiến độ, chi phí trong thực tế với trong kế hoạch.

Giám sát thân mật thường có thể dự đoán những vấn đề tiềm năng của dự án bằng cách đưa ra những khó khăn khi chúng xuất hiện. Ví dụ, những cuộc thảo luận hàng ngày với đội dự án có thể đưa ra những vấn đề thực tế trong việc tìm lỗi phần mềm. Điều này thì tốt hơn nhiều so với việc chờ báo cáo trễ tiến đội, thông qua đó người quản lý có thể giao vấn đề gặp phải cho một vài chuyên gia hay quyết định rằng dự án nên được lập trình lại.

Trong suốt dự án, nhiều cuộc review tiến độ tổng thể và tiến độ phát triển từng phần của dự án sẽ diễn ra, kiểm tra xem dự án và mục đích của tổ chức chi trả cho phần mềm có đi đúng hướng không. Thông qua các cuộc review này mà quyết định hoãn dự án có thể được đưa ra. Thời gian thực hiện 1 dự án lớn có thể là vài năm. Trong suốt thời gian này mục đích của tổ chức có thể thay đổi, dẫn đến phần mềm có thể không còn được yêu cầu hay những yêu cầu ban đầu không còn đúng nữa. Người quản lý có thể quyết định ngừng phát triển dự án hay thay đổi dự án để thích hợp với những thay đổi về mục đích của tổ chức.

1.2.2 Đo hiệu năng và chất lượng phần mềm

1.2.2.1 Các nhân tố tác động đến chất lượng

Cách đây nhiều năm, McCall và Cavano đã định nghĩa 1 tập hợp những nhân tố chất lượng mà đánh giá chất lượng phần mềm từ 3 quan điểm:

- Hoạt động (việc sử dụng phần mềm).
- Xem xét lại (việc thay đổi phần mềm).
- Chuyển đổi (cập nhật phần mềm để làm việc trong các môi trường khác nhau).

1.2.2.2 Đo lường chất lượng

Mặc dù có nhiều cách đo chất lượng phần mềm, tính đúng đắn, tính bảo trì, tính tích hợp và tính tiện dụng cung cấp những số liệu hữu ích cho đội dự án. Gilb đề nghị những định nghĩa và đo lường sau cho từng cái:

Tính đúng đắn: một chương trình phải hoạt động một cách đúng đắn, chính xác. Tính đúng đắn là mức độ mà phần mềm thực hiện chức năng yêu cầu. Phương thức đo lường tính đúng đắn phổ biến nhất là lỗi trên 1 KLOC, nơi mà 1 lỗi được định nghĩa như là 1 sự thiếu kiểm chứng của sự phù hợp đối với yêu cầu. Khi xem xét chất lượng tổng quát của sản phẩm phần mềm, lỗi là những vấn đề được báo cáo với người dùng chương trình sau khi chương trình được đưa vào sử dụng. Đối với những mục đính đánh giá chất lượng, lỗi được đếm trên 1 thời kỳ chuẩn, điển hình là 1 năm.

NOTE: KLOC (kilo lines of code): one thousand lines of programming source code.

Tính bảo trì: Bảo trì phần mềm giải thích cho sự nỗ lực hơn là bất cứ hoạt động CNPM nào khác. Tính bảo trì thì dễ với chương trình có thể được sửa chính xác nếu gặp lỗi, thích hợp nếu môi trường thay đổi hay nâng cấp nếu khách hàng mong muốn 1 sự thay đổi trong yêu cầu. Không cách nào để đo trực tiếp tính bảo trì nên chúng ta phải sử dụng cách đo lường gián tiếp. Số liệu hướng thời gian đơn giản (time-oriented metric) là cách lấy thời gian để thực hiện việc thay đổi (mean-time-to-change MTTC). Thời gian được dùng để phần tích yêu cầu thay đổi, thiết kế sự cập nhật thích hợp, thực thi sự thay đổi, kiểm chứng và phân phối những thay đổi đến người dùng. Nhìn chung, những chương trình có tính bảo trì sẽ có MTTC (cho những loại thay đổi tương đương) thấp hơn những chương trình mà không có tính bảo trì.

Tính tích hợp: Tính tích hợp phần mềm trở nên ngày càng quan trọng trong thời đại của những tin tặc và các bức tường lửa. Thuộc tính này đo khả năng chịu đựng của hệ thống trước các cuộc tấn công (cả tình cờ hay cố ý) đến tính bảo mật của phần mềm. Các cuộc tấn công có thể nhắm vào 3 thành phần của phần mềm: chương trình, dữ liệu và tài liệu. Để đo tính tích hợp, 2 thuộc tính thêm vào phải được định nghĩa: mối đe dọa và độ bảo mật (threat and security). Mối đe dọa là xác suất (có thể được ước lượng hay thu thập từ những chứng cứ thực nghiệm) mà 1 cuộc tấn công của 1 loại đặc biệt sẽ xuất hiện trong 1 thời gian đưa ra. Độ bảo mật là xác suất (có thể được ước lượng hay được thu thập từ những chứng cứ thực nghiệm) mà 1 cuộc tất công của 1 loại đặc biệt sẽ bị đẩy lùi. Đối với từng loại tấn công, tính tích hợp của hệ thống có thể được định nghĩa:

Integrity = summation [(1 - threat) X (1 - security)]

Tính tiện dụng: nếu 1 chương trình không thân thiện với người dùng, nó thường sẽ bị thất bại, cho dù các chức năng thực thi thì có hiệu quả. Tính tiện dụng được dùng để định lượng sự thân thiện với người dùng và có thể được đo lường trong 4 đặc tính sau:

(1): thể chất hoặc/ và kỹ năng trí tuệ đòi hỏi để học hệ thống.

(2): thời gian đòi hỏi để trở nên có hiệu quả 1 cách vừa phải trong việc sử dụng hệ thống.

(3): sự gia tăng ròng về năng suất (qua cách tiếp cận mà hệ thống thay thế) được đo khi hệ thống được sử dụng bởi 1 người có hiệu quả vừa phải.

(4): một đánh giá chủ quan (đôi khi đạt được thông qua 1 bảng các câu hỏi) của thái độ người dùng đối với hệ thống.

1.2.2.3 Hiệu quả loại bỏ lỗi (Defect Removal Efficiency – DRE)

Một thước đo chất lượng mà cung cấp lợi ích ở cả dự án và cấp tiến độ là hiệu quả loại bỏ lỗi (DRE). Về bản chất, DRE là thước đo khả năng lọc của các hoạt động kiểm soát và đảm bảo chất lượng khi chúng được áp dụng trong tất cả các hoạt động khung tiến trình.

Khi xem xét tổng thể 1 dự án, DRE được định nghĩa như sau:

DRE = E/(E + D)

Trong đó

E: số lượng lỗi (error) được tìm thấy trước khi đưa phần mềm cho người dùng cuối.

D: số lượng lỗi (defect) được tìm thấy sau khi giao.

Lý tưởng là DRE = 1, tức không có lỗi (defect) nào được tìm thấy trong phần mềm. Thực tế, D > 0 nhưng DRE vẫn có thể tiến đến 1. Khi E tăng (đối với giá trị D được đưa ra), giá trị tổng thể của DRE bắt đầu tiến đến 1. Trong thực tế, khi E tăng, có vẻ như là giá trị của D sẽ giảm (error được lọc trước khi trở thành defect). DRE khuyến khích đội dự án tìm càng nhiều lỗi trước khi giao càng tốt.

DRE có thể được sử dụng để đánh giá khả năng của đội trong việc tìm kiếm lỗi tại 1 task trước khi chuyển qua task khác. Trong trường hợp này, DRE được định nghĩa như sau:

$DRE_i = E_i/(E_i + E_{i+1})$

E_i : số lượng lỗi được tìm thấy trong task i

E_{i+1} : số lượng lỗi được tìm thấy trong task i+1 mà không được tìm thấy trong task i

Mục tiêu chất lượng là DRE tiến đến 1, tức lỗi nên được lọc ra trước khi chuyển qua task khác.

Tham khảo thêm về đo hiệu năng phần mềm và công cụ đo tại: http://www.pcworld.com.vn/articles/cong-nghe/ung-dung/2007/09/1191121/kiem-tra-hieu-nang-phan-mem-voi-loadrunner-8-1/

1.3 Quản lý dự án: Ước lượng phần mềm

1.3.1 Quan sát về ước lượng

Ước lượng tài nguyên, chi phí và lịch biểu đòi hỏi kinh nghiệm, truy cập thông tin lịch sử tốt và can đảm cam kết những dự đoán có tính định lượng khi chỉ tồn tại thông tin định lượng. Việc ước lượng gồm những rủi ro sẵn có và rủi ro này dẫn đến sự không chắc chắn.

Độ phức tạp của dự án có tác động mạnh mẽ trong việc lập kế hoạch. Nó là thước đo chịu ảnh hưởng bởi những nỗ lực cho những điều tương tự trong quá khứ.

Kích thước dự án là 1 yếu tố quan trọng khác mà có thể tác động đến sự chính xác và hiệu quả của việc ước lượng. Khi kích thước tăng, sự phụ thuộc qua lại giữa các nhân tố khác nhau trong dự án cũng phát triển nhanh chóng. Việc phân hoạch vấn đề, 1 yếu tố quan trong để ước lượng, trở nên khó khăn hơn bởi những nhân tố bị phân hoạch vẫn còn rất dữ dội.

Mức độ không chắc chắn về cấu trúc cũng tác động đến rủi ro ước lượng.

Tính sẵn có của thông tin lịch sử có ảnh hưởng mạnh mẽ đến rủi ro ước lượng. Bằng việc xem xét lại, chúng ta có thể bắt chước những thứ đã làm và cải tiến những chỗ mà vấn đề xuất hiện.

Rủi ro được đo bằng mức độ không chắc chắn trong ước lượng tài nguyên, chi phí và lịch biểu. Nếu phạm vi dự án được hiểu 1 cách nghèo nàn và những yêu cầu dự án bị thay đổi thì sự không chắc chắn và rủi ro trở nên cao 1 cách nguy hiểm. Người lập kế hoạch nên đòi hỏi sự đầy đủ những định nghĩa về chức năng, sự thực thi và giao diện. Hơn nữa, khách hàng cũng nên nhận biết rằng sự thay đổi trong yêu cầu có nghĩa là sự bất ổn trong chi phí và lịch biểu.

1.3.2 Phạm vi phần mềm

Được định nghĩa bằng cách trả lời các câu hỏi sau:

- Ngữ cảnh: phần mềm được xây dựng để thích hợp trong ngữ cảnh như thế nào, trong những ngữ cảnh đó thì có những ràng buộc nào,…
- Mục đích thông tin: dữ liệu đầu vào, đầu ra là gì?,…
- Chức năng và sự thực thi: chức năng nào thực hiện chuyển đổi dữ liệu đầu vào thành dữ liệu đầu ra? Những đặc tính thực thi nào được chỉ ra?

Phạm vi dự án phần mềm phải rõ ràng và dễ hiểu ở mức độ quản lý và kỹ thuật. Dữ liệu định lượng (như: số lượng người dùng đồng thời, kích thức mailing list, thời gian tối đa cho phép trả lời) được nêu rõ ràng, những ràng buộc và/ hay những giới hạn (như: chi phí sản phẩm hạn chế kích thước bộ

nhớ) được ghi chú, các yếu tố giảm nhẹ (như: những thuật toán mong muốn được hiểu rõ và có sẵn trong C++) được miêu tả.

1.3.3 Ước lượng dự án phần mềm

1 lỗi ước lượng chi phí lớn có thể làm nên sự khác biết giữa lợi nhuận và sự mất mát. Việc vượt chi phí có thể gây nên sự bất hạnh cho những nhà phát triển.

Ước lượng chi phí và nỗ lực cho phần mềm sẽ không bao giờ là khoa học chính xác. Nhiều yếu tố - con người, kỹ thuật, môi trường, chính trị- có thể ảnh hưởng đến chi phí sau cùng của phần mềm và nỗ lực để phát triển nó. Để đạt được những ước lượng chi phí và nỗ lực đáng tin cậy, những lựa chọn sau được đưa ra:

(1): Ước lượng sự chậm trễ được làm từ đầu cho đến cuối dự án (hiển nhiên, chúng ta có thể đạt được những ước lượng chính xác 100% sau khi dự án hoàn thành).

(2): Những ước lượng dựa trên những dự án tương tự mà đã được hoàn thành.

(3): Sử dụng những kỹ thuật phân hoạch đơn giản có liên quan để tạo ra những ước lượng chi phí và nỗ lực cho phần mềm.

(4): Sử dụng một hay nhiều mô hình thực nghiệm cho việc ước lượng chi phí và nỗ lực cho phần mềm.

Lựa chọn (1) thì không thực tế. Những ước lượng chi phí phải được đưa ra từ đầu. Tuy nhiên, chúng ta nên nhân thấy rằng chúng ta càng chờ lâu bao nhiêu thì chúng ta càng biết nhiều bấy nhiêu, và chúng ta càng biết nhiều bao nhiêu thì chúng ta càng ít có khả năng tạo ra những lỗi nghiêm trọng trong ước lượng.

Lựa chọn (2) có thể làm việc khá tốt nếu dự án hiện tại thì khả giống với những nỗ lực trong quá khứ và những ảnh hưởng đến dự án khác (như: khách hàng, điều kiện kinh doanh, SEE, hạn chót) thì tương đượng nhau. Thật không may, kinh nghiệm trong quá khứ không phải luôn luôn là chỉ báo tốt cho những kết quả trong tương lai.

Những lựa chọn còn lại là những phương pháp hữu hiệu cho việc ước lượng dự án phần mềm. 1 cách lý tưởng, các kỹ thuật được ghi nhận cho mỗi lựa chọn nên được áp dụng song song và dùng kiểm tra chéo lần nhau.

Những kỹ thuật phân hoạch dùng phương pháp "phân chia và chiếm đoạt" để ước lượng dự án phần mềm. Bằng cách phân hoạch dự án thành những chức năng chính và những hoạt động CNPM liên quan, việc ước lượng chi phí và nỗ lực cho phần mềm có thể được thực thi trong hình dạng bậc thang.

Những mô hình ước lượng dựa trên kinh nghiệm có thể được dùng để bổ sung cho những kỹ thuật phân hoạch và cung cấp 1 phương pháp ước lượng có giá trị tiềm năng trong phạm vi của nó. 1 mô hình thì dựa vào kinh nghiệm (những dữ liệu có tính lịch sử) và theo định dạng: $d = f(v_i)$ với

d: một trong những giá trị được ước lượng (như: nỗ lực, chi phí, thời gian dự án)

v_i: những biến độc lập được chọn (như: LOC hay FP (function points) được ước lượng)

Những công cụ ước lượng tự động thực hiện 1 hay nhiều kỹ thuật phân hoạch hay nhưng mô hình dựa trên kinh nghiệm. Khi kết hợp với giao diện đồ họa người dụng (GUI), những công cụ này cung cấp những lựa chọn đầy hấp dẫn cho việc ước lượng.

Mỗi lựa chọn đều tốt như những dữ liệu có tính lịch sử dùng để ước lượng. Nếu không có những dữ liệu này, việc định giá sẽ nằm trên 1 nền tảng không vững chắc.

Những loại ước lượng thuộc về kỹ thuật phân hoạch:

- Ước lượng dựa vào vấn đề (problem-based estimation): gồm LOC-based estimation và FP-based estimation.
- Ước lượng dựa vào tiến trình (process-based estimation)

Những mô hình ước lượng dựa trên kinh nghiệm:

- Mô hình COCOMO.
- Mô hình cân bằng phần mềm (Software equation model).

1.4 Quản lý dự án: Lập kế hoạch

1.4.1 Lập kế hoạch dự án phần mềm

Mục đích của việc lập kế hoạch dự án phần mềm là cung cấp 1 khung (framework) mà cho phép người quản lý làm những ước lượng hợp lý về tài nguyên, chi phí và lịch biểu. Những ước lượng này được làm trong 1 khung thời gian giới hạn tại thời điểm bắt đầu của dự án phần mềm và nên được cập nhật thường xuyên như là sự tiến triển của dự án. Thêm vào, những ước lượng nên cố gắng chỉ ra những kịch bản cho trường hợp tốt nhất và xấu nhất để kết quả của dự án có thể được tốt đẹp hơn. Khung (framework) này gồm những giai đoạn sau:

- Thiết kế.
- Code.
- Test.
- Giao sản phẩm (gồm thời gian giao và chờ khách hàng thực hiện acceptance test, phản hồi, sửa lỗi).
- Bảo trì.

Mục đích của việc lập kế hoạch đạt được thông qua 1 tiến trình của việc khai thác thông tin mà đưa đến những ước lượng hợp lý.

Khi thực hiện việc lập kế hoạch dự án phải xem xét đến những thông tin sau:

- Phạm vi của dự án.
- Tài nguyên: bao gồm nhân lực, những phần mềm sử dụng, môi trường.

1.4.2 Lập lịch biểu tổ chức

Lập lịch biểu phần mềm là 1 hoạt động phân phối những nỗ lực (effort) ước lượng xuyên suốt khoảng thời gian của dự án bằng cách ấn định nỗ lực với task cụ thể. Sau đây là những nguyên tắc cơ bản hướng dẫn việc lập lịch biểu phần mềm dự án.:

- Phân chia công việc ra thành các hoạt động mang tính quản lý và task.
- Sự phụ thuộc qua lại giữa các task.
- Ấn định thời gian.
- Thừa nhận nỗ lực (Effort validation).
- Xác định trách nhiệm cho mỗi task.

- Xác định kết quả của mỗi task.
- Xác định cột mốc cho mỗi task. Khi một cột mốc được hoàn thành, công việc sẽ được review về mặt chất lượng và approve (chấp nhận) nếu tốt.

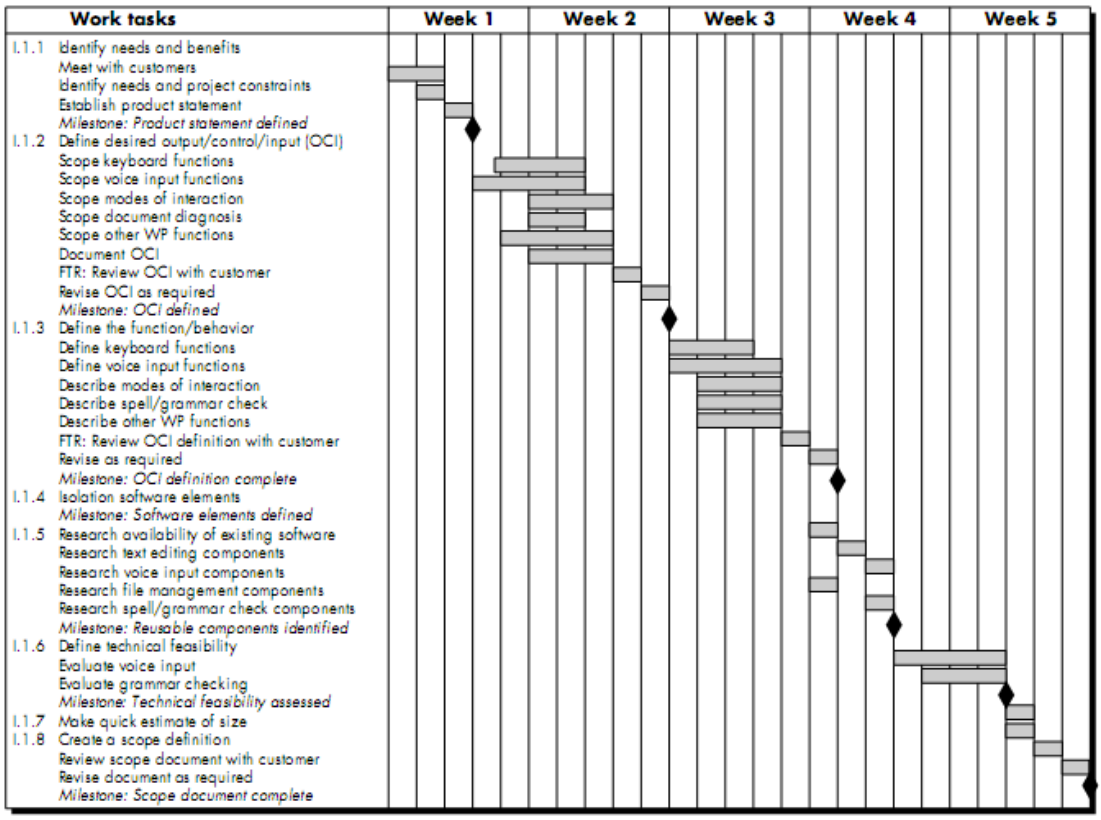
Đối với 1 dự án nhỏ, từng cá nhân có thể thực hiện : phân tích yêu cầu, thiết kế, code, test. Khi kích thước dự án tăng lên, nhiều người hơn sẽ tham gia vào dự án. Việc thêm ngươi vào dự án khi dự án bị trễ tiến độ có thể gây hiệu quả không tốt đến dự án, làm lịch biểu bị giãn ra. Chính vì vậy, thông thường trong trường hợp này người quản lý sẽ thương lượng lại với khách hàng về ngày giao.

Dựa vào yêu cầu phần mềm mà người dự án phải phân chia công việc cho hợp lý, xác định task 1 cách đúng nhất. Sau đó, dùng tool để tạo lịch biểu.

Cách tạo lịch biểu :

- Xác định task.
- Xác định effort, ngày bắt đầu và khoảng thời gian thực hiện cho từng task.
- Xác định người thực hiện task.

Tool thường được sử dụng để tạo lịch biểu hiện nay là Microsoft Project.



Hình 1.4.2 : ví dụ về 1 lịch biểu

Người quản lý phải luôn theo dõi tiến độ thực hiện để đảm bảo schedule được thực hiện đúng và hành động, đưa ra những quyết định đúng đắn khi không đúng tiến độ.

1.4.3 Kế hoạch dự án phần mềm

Kế hoạch dự án phần mềm được tạo ra tại planning task. Nó cung cấp thông tin về chi phí và lịch biểu mà sẽ được sử dụng xuyên suốt tiến trình phần mềm. Kế hoạch dự án phần mềm là 1 tài liệu gắn ngọn mà được đưa ra cho nhóm người liên quan. Nó phải

(1): truyền tải thông tin về phạm vi và nguồn lực cho những người quản lý, đội ngũ kỹ thuật và khách hàng.

(2): định nghĩa những rủi ro và đề xuất những kỹ thuật làm giảm rủi ro.

(3): định nghĩa chi phí và lịch biểu cho management review (xem xét việc quản lý).

(4): cung cấp 1 phương thức tổng thể cho việc phát triển phần mềm cho tất cả những người có liên quan.

(5): phác thảo số lượng chất lượng sẽ được đảm bảo và sự thay đổi sẽ được quản lý.

Bảng kế hoạch không phải là 1 tài liệu tĩnh, đội dự án có thể thực hiện việc cập nhật những rủi ro, ước lượng, lịch biểu và những thông tin trong bảng kế hoạch.

Chương 2: PHÂN TÍCH YÊU CẦU HỆ THỐNG

2.1 Công nghệ hệ thống máy tính

2.1.1 Công nghệ hệ thống máy tính (System engineering)

2.1.1.1 Tổng quan

Công nghệ phần mềm xuất hiện như là kết quả của công nghệ hệ thống máy tính. Thay vì chỉ tập trung vào phần mềm, công nghệ hệ thống tập trung vào những nhân tố khác, phân tích, thiết kế và tổ chức những nhân tố này vào 1 hệ thống mà có thể là sản phẩm, dịch vụ hay công nghệ. Các nhân tố này là:

- Phần mềm: những chương trình máy tính, cấu trúc dữ liệu và những tài liệu liên quan mà tác động đến phương pháp hợp lệ, thủ tục hay những điều khiển được yêu cầu.
- Phần cứng: những thiết bị điện tử cung cấp khả năng tính toán, những thiết bị liên kết nối (như: thiết bị chuyển mạch mạng network switches, thiết bị viễn thông telecommunications devices) cho phép luồng dữ liệu, những thiết bị cơ điện electromechanical devices (như: cảm biến, động cơ, máy bơm) cung cấp chức năng thế giới bên ngoài.
- Con người: người dùng và người vận hành phần cứng, phần mềm.
- Cơ sở dữ liệu: 1 tập hợp thông tin lớn và có tổ chức được truy cập thông qua phần mềm.
- Tài liệu: thông tin miêu tả (như: sách hướng dẫn sử dụng, tập tin trợ giúp trực tuyến, các trang web) mà miêu tả cách sử dụng và/ hay cách hoạt động của hệ thống.
- Thủ tục: các bước xác định cách sử dụng cụ thể của mỗi nhân tố hệ thống hay ngữ cảnh thủ tục mà hệ thống thuộc về.

2.1.1.2 Phân tầng công nghệ hệ thống

Công nghệ hệ thống bao gồm 1 tập hợp những phương pháp top-down, bottom-up để định hướng sự phân tầng như hình bên dưới.

Bắt đầu với "world view", đó là toàn bộ phạm vi nghiệp vụ hay sản phẩm được xem xét để đảm bảo rằng ngữ cảnh nghiệp vụ hay công nghệ có thể được thiết lập. Quan điểm thế giới (World view) được tinh chế để tập trung đầy

đủ hơn vào phạm vi quan tâm cụ thể (domain of interest). Trong 1 phạm vi cụ thể, nhu cầu cho những nhân tố hệ thống mục tiêu (như: dữ liệu, phần mềm, phần cứng, con người) được phân tích. Cuối cùng, phân tích, thiết kế và cấu tạo của 1 nhân tố hệ thống mục tiêu được thiết lập. Ở đầu sự phân tầng, 1 ngữ cảnh chung được thiết lập và ở cuối, những hoạt động kỹ thuật chi tiết được chỉ ra.

Theo hình bên dưới, world view (WV) gồm nhiều domain (Di) –có thể là 1 hệ thống hay hệ thống con.

WV = {D1, D2,…, Dn}

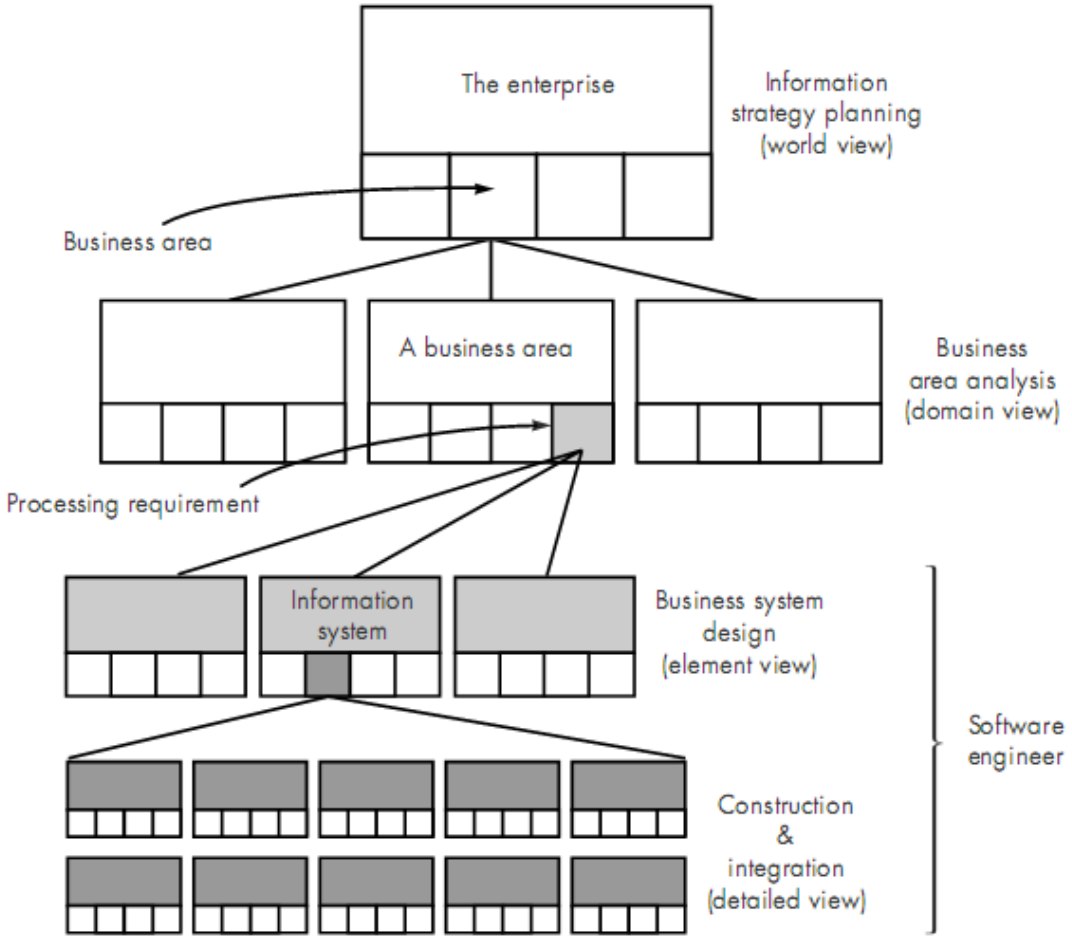
Mỗi domain gồm những nhân tố cụ thể (Ej), mỗi Ej phục vụ cho vài vai trò cho việc hoàn thành mục tiểu của domain hay component.

Di = {E1, E2, …, En}

Mõi nhân tố được thực hiện bằng cách cụ thể những thành phần (component) (Ck) kỹ thuật mà thực hiện chức năng cần thiết cho 1 nhân tố.

Ej = {C1, C2, …, Cn}

Trong ngữ cảnh phần mềm, 1 thành phần có thể là 1 chương trinhg máy tính, 1 thành phần chương trình có tính tái sử dụng, 1 module, 1 class hay object hay thậm chí là 1 câu lệnh ngôn ngữ lập trình.



2.1.2 Phân tích hệ thống

Hoạt động phân tích phân loại yêu cầu và tổ chức chúng vào những tập con liên quan, tìm hiểu mối quan hệ giữa các yêu cầu với nhau, xem xét các yêu cầu cho tính nhất quán, sự thiếu sót và sự mơ hồ; và xếp hạng những yêu cầu dựa vào nhu cầu của khách hàng/ người sử dụng.

Trong hoạt động phân tích yêu cầu, những câu hỏi sau được hỏi và trả lời:

- Mỗi yêu cầu có phù hợp với mục tiêu tổng thể của hệ thống/ sản phẩm?
- Tất cả các yêu cầu có được chỉ rõ ở mức độ trừu tượng thích hợp không? Đó là, có phải một số yêu cầu cung cấp 1 mức độ chi tiết kỹ thuật mà không thích hợp ở giai đoạn này không?
- Yêu cầu có thật sự cần thiết? hay nó có đại diện cho 1 tính năng thêm vào nào mà không cần thiết đối với mục tiêu của hệ thống không?
- Mỗi yêu cầu có bị giới hạn hay rõ ràng không?

- Có yêu cầu nào mâu thuẫn với những yêu cầu khác không?
- Mỗi yêu cầu có tính kiểm chứng một khi được thực thi không?

…

2.1.3 Đặc tả hệ thống

Trong ngữ cảnh hệ thống máy tính, thuật ngữ "đặc tả" (specification) có nghĩa là những điều khác nhau đối với những người khác nhau. Một đặc tả có thể là 1 tài liệu được viết ra, 1 mô hình đồ họa, 1 mô hình toán học hình thức, 1 tập hợp những kịch bản sử dụng, 1 mẫu thử, hay sự kết hợp những thứ này.

Một số người đề nghị rằng 1 "mẫu chuẩn" (standard template) nên được phát triển và sử dụng cho đặc tả hệ thống, cho rằng đều này dẫn đến những yêu cầu được trình bày nhất quán và do đó mà dễ hiểu hơn. Tuy nhiên, đôi khi cần linh hoạt khi một đặc tả được phát triển. Đối với những hệ thống lớn, 1 tài liệu được viết ra, kết hợp với những miêu tả ngôn ngữ tự nhiên và những mô hình đồ họa có thể là cách tiếp cận tốt nhất. Tuy nhiên, những kịch bản có tính sử dụng có thể là tất cả những gì được đòi hỏi cho những sản phẩm nhỏ hơn hay những hệ thống mà nằm bên trong những môi trường kỹ thuật được hiểu rõ.

Đặc tả hệ thống là sản phẩm công việc cuối cùng được tạo ra bởi kỹ sư hệ thống và yêu cầu. Nó phục vụ như nền tảng cho công nghệ phần cứng, công nghệ phần mềm, công nghệ cơ sở dữ liệu và công nghệ nhân lực (human engineering). Nó miêu tả chức năng và hiệu năng của hệ thống máy tính và những ràng buộc mà quản lý sự phát triển. Đặc tả giới hạn mỗi nhân tố hệ thống được chỉ định. Đặc tả cũng miêu tả thông tin (dữ liệu và điều khiển) mà được nhập vào hay xuất ra từ hệ thống.

Sinh viên tìm hiểu và nghiên cứu thêm về một số mô hình và kỹ thuật đặc tả.

2.2 Nền tảng của phân tích yêu cầu

2.3.1 Các nguyên lý phân tích

Trên hai thập kỉ qua, người ta đã xây dựng ra một số phương pháp phân tích và đặc tả phần mềm. Những người nghiên cứu đã xác định ra các vấn đề và nguyên nhân của chúng, và đã xây dựng ra các qui tắc và thủ tục để vượt qua

chúng. Mỗi phương pháp đều có kí pháp và quan điểm riêng. Tuy nhiên, tất cả các phương pháp này đều có quan hệ với một tập hợp các nguyên lý cơ bản:

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

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

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

4. Tiến trình phân tích phải đi từ thông tin bản chất hướng tới chi tiết cài đặt. Bằng cách áp dụng những nguyên lý này, người phân tích tiếp cận tới vấn đề một cách hệ thống.

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

2.3.2 Mô hình hóa

Chúng ta tạo ra các mô hình để thu được hiểu biết rõ hơn về thực thể thực tế cần xây dựng. Khi thực thể là một vật vật lý (như toà nhà, máy bay, máy móc) thì ta có thể xây dựng một mô hình giống hệt về hình dạng, nhưng nhỏ hơn về qui mô. Tuy nhiên, khi thực thể cần xây dựng là phần mềm, thì mô hình của chúng ta phải mang dạng khác. Nó phải có khả năng mô hình hóa thông tin mà phần mềm biến đổi, các chức năng (và chức năng con) làm cho phép biến đổi đó thực hiện được, và hành vi của hệ thống khi phép biến đổi xảy ra.

Trong khi phân tích các yêu cầu phần mềm, chúng ta tạo ra các mô hình về hệ thống cần xây dựng. Các mô hình tập trung vào điều hệ thống phải thực hiện, không chú ý đến cách thức nó thực hiện. Trong nhiều trường hợp, các mô hình chúng ta tạo ra có dùng kí pháp đồ hoạ mô tả cho thông tin, xử lý, hành vi hệ thống, và các đặc trưng khác thông qua các biểu tượng phân biệt và dễ nhận

diện. Những phần khác của mô hình có thể thuần túy văn bản. Thông tin mô tả có thể được cung cấp bằng cách dùng một ngôn ngữ tự nhiên hay một ngôn ngữ chuyên dụng cho mô tả yêu cầu. Các mô hình được tạo ra trong khi phân tích yêu cầu còn đóng một số vai trò quan trọng:

• Mô hình trợ giúp cho người phân tích trong việc hiểu về thông tin, chức năng và hành vi của hệ thống, do đó làm cho nhiệm vụ phân tích yêu cầu được dễ dàng và hệ thống hơn.

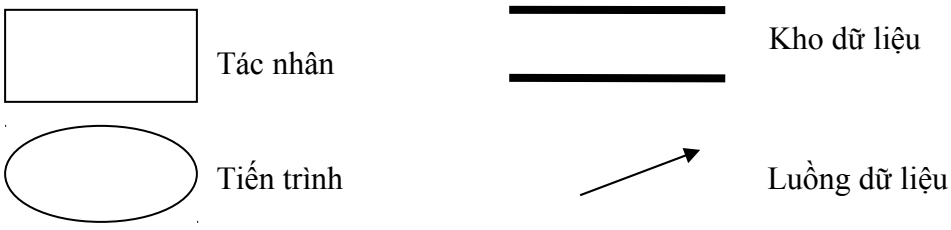
• Mô hình trở thành tiêu điểm cho việc xem xét và do đó, trở thành phần mấu chốt cho việc xác định tính đầy đủ, nhất quán và chính xác của đặc tả.

• Mô hình trở thành nền tảng cho thiết kế, cung cấp cho người thiết kế một cách biểu diễn chủ yếu về phần mềm có thể được "ánh xạ" vào hoàn cảnh cài đặt.

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

2.3.2.1 Biểu đồ luồng dữ liệu

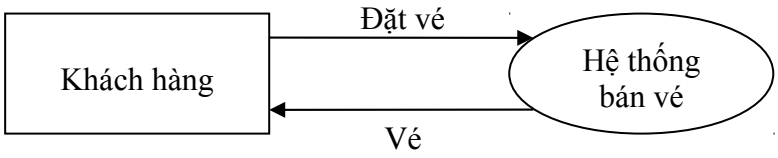
Khi thông tin đi qua phần mềm nó bị thay đổi bởi một loạt các phép biến đổi. Biểu đồ luồng dữ liệu (Data Flow Diagram - DFD) là một kỹ thuật vẽ ra luồng dữ liệu di chuyển trong hệ thống và những phép biến đổi được áp dụng lên dữ liệu. Ký pháp cơ bản của biểu đồ luồng dữ liệu được minh họa trên hình 2.3.2.1.



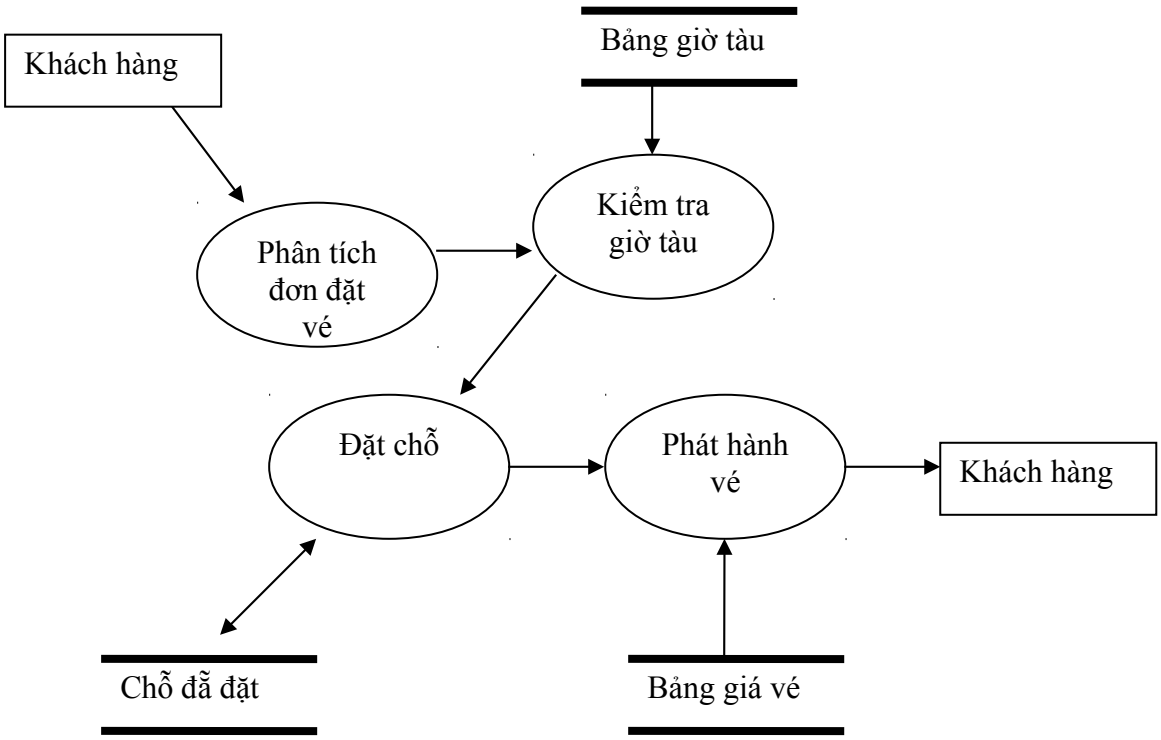
Hình 2.3.2.1(a): Ký pháp DFD.

Biểu đồ luồng dữ liệu có thể được dùng để biểu diễn cho một hệ thống hay phần mềm ở bất kì mức trừu tượng nào. Trong thực tế, DFD có thể được phân hoạch thành nhiều mức biểu diễn cho chi tiết chức năng và luồng thông tin ngày càng tăng. Do đó phương pháp dùng DFD còn được gọi là phân tích có cấu trúc. Một DFD mức 0, cũng còn được gọi là biểu đồ nền tảng hay biểu đồ

ngữ cảnh hệ thống, biểu diễn cho toàn bộ phần tử phần mềm như một hình tròn với dữ liệu vào và ra được chỉ ra bởi các mũi tên tới và đi tương ứng. Một DFD mức 1 cụ thể hóa của DFD mức 0 và có thể chứa nhiều hình tròn (chức năng) với các mũi tên (luồng dữ liệu) nối lẫn nhau. Mỗi một trong các tiến trình được biểu diễn ở mức 1 đều là chức năng con của toàn bộ hệ thống được mô tả trong biểu đồ ngữ cảnh. Hình 2.3 minh họa một DFD cho hệ thống bán vé tàu.



DFD mức 0



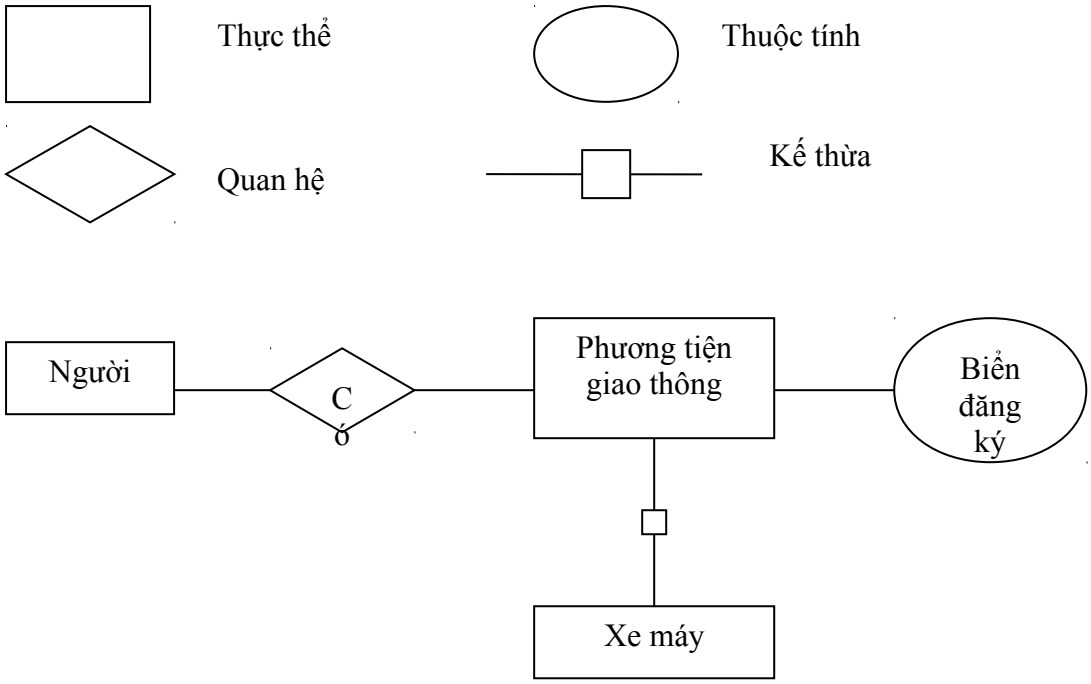
DFD mức 1

Hình 2.3.2.1(b): Biểu đồ luồng dữ liệu của một hệ thống bán vé tầu.

2.3.2.2 Biểu đồ thực thể quan hệ

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

Ký pháp của biểu đồ ER cũng tương đối đơn giản. Các thực thể được biểu diễn bằng các hình chữ nhật có nhãn. Mối quan hệ được chỉ ra bằng hình thoi. Các mối nối giữa sự vật dữ liệu và mối quan hệ được thiết lập bằng cách dùng nhiều đường nối đặc biệt (hình 2.3.2.2).



Hình 2.3.2.2: Mô hình thực thể quan hệ người - phương tiện giao thông.

2.3.3 Người phân tích

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

- Khả năng hiểu thấu các khái niệm trừu tượng, có khả năng tổ chức lại thành các phân tích logic và tổng hợp các giải pháp dựa trên từng dải phân chia.
- Khả năng rút ra các sự kiện thích đáng từ các nguồn xung khắc và lẫn lộn.
- Khả năng hiểu được môi trường người dùng/khách hàng.
- Khả năng áp dụng các phần tử hệ thống phần cứng và/hoặc phần mềm vào môi trường người sử dụng/khách hàng.
- Khả năng giao tiếp tốt ở dạng viết và nói.
- Khả năng trừu tượng hóa/tổng hợp vấn đề từ các sự kiện riêng lẻ.

2.3 Phân tích có cấu trúc

2.3.1 Tạo biểu đồ ER

Biểu đồ ER giúp cho kỹ sư phần mềm chỉ ra đầy đủ những đối tượng dữ liệu đầu vào, đầu ra của hệ thống, những thuộc tính xác định tính chất của các đối tượng và mối quan hệ. Những cách tiếp cận sau nên được thực hiện trong việc tạo biểu đồ ER:

1. Yêu cầu khách hàng liệt kê những thứ có trong ứng dụng.

2. Xét 1 đối tượng, xác định sự liên kết giữa đối tượng này với các đối tượng dữ liệu khác.

3. Tạo ra các mối quan hệ dựa trên sự liên kết giữa các đối tượng.

4. Đối với mỗi đối tượng/ cặp quan hệ, kiểu quan hệ (1-1,1-n,n-n) và phương thức quan hệ được xem xét.

5. Lặp lại bước 2 đến bước 4 cho đến khi tất cả các đối tượng/ cặp quan hệ được xác định.

6. Các thuộc tính của mỗi thực thể được xác định.

7. Biểu đồ ER được hình thức hóa và xem xét lại.

8. Bước 1 đến bước 7 được lặp lại cho đến khi mô hình dữ liệu được hoàn thành.

Để minh họa cho cách sử dụng của những hướng dẫn cơ bản này, ta xem xét ví dụ về hệ thống Safehome.

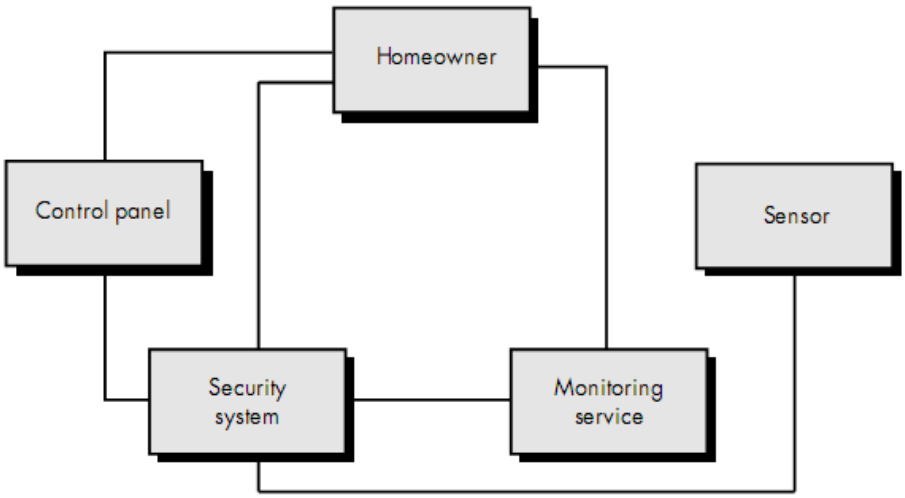
Bài toán hệ thống Safehome:

SafeHome software *enables* the homeowner to *configure* the security system when it is *installed, monitors* all sensors *connected* to the security system, and *interacts* with the homeowner through a keypad and function keys *contained* in the SafeHome control panel shown in Figure 11.2.

During installation, the SafeHome control panel is *used* to "*program*" and *configure* the system. Each sensor is *assigned* a number and type, a master password is programmed for *arming* and *disarming* the system, and telephone number(s) are *input* for *dialing* when a sensor event occurs.

When a sensor event is *recognized*, the software *invokes* an audible alarm attached to the system. After a delay time that is *specified* by the homeowner during system configuration activities, the software dials a telephone number of a monitoring service, *provides* information about the location, *reporting* the nature of the event that has been detected. The telephone number will be *redialed* every 20 seconds until telephone connection is *obtained*.

All interaction with SafeHome is *managed* by a user-interaction subsystem that *reads* input provided through the keypad and function keys, *displays* prompting messages on the LCD display, displays system status information on the LCD display. Keyboard interaction takes the following form . . .



Hình 2.3.1(a): hệ thống safehome

Bước 1: hệ thống gồm:

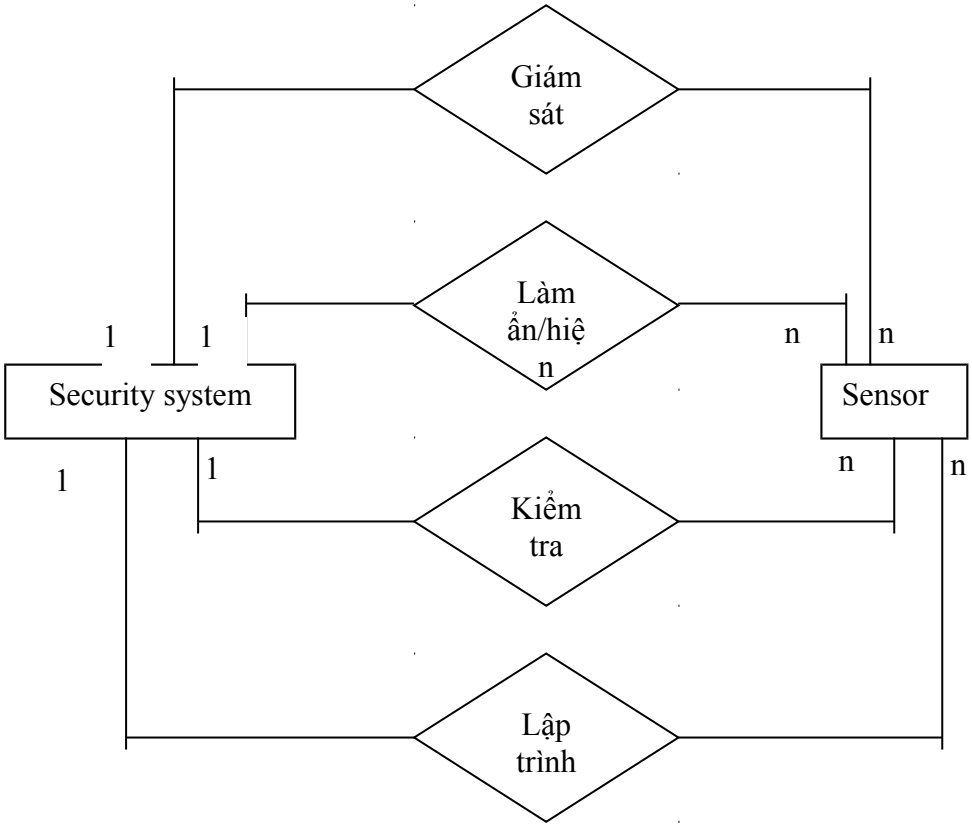
- Homeowner
- Control panel
- Sensor
- Security system
- Monitoring service

Bước 2: sự liên kết trực tiếp giữa homeowner với control panel, security system và monitoring service; sự liên kết đơn giữa sensor với sercurity system.

Bước 3: khi tất cả các liên kết được xác định, một hay nhiều cặp quan hệ được chỉ ra cho mỗi liên kết. Ví dụ, sự liên kết giữa sensor và security system có những mối quan hệ sau:

- Security system giám sát sensor.
- Security system làm ẩn/hiện sensor
- Security system kiểm tra sensor
- Security system lập trình sensor

Bước 4: Mỗi cặp quan hệ sẽ được đem ra phân tích để xác định kiểu quan hệ và phương thức quan hệ. Ví dụ quan hệ security system giám sát sensor, kiểu quan hệ : 1-n, phương thức quan hệ: 1 security system giám sát 1 hay nhiều sensor.



Bước 6: các thuộc tính nên tập trung vào những dữ liệu phải được lưu trữ để giúp cho hệ thống hoạt động được. Ví dụ, đối tượng sensor có thể có những thuộc tính sau: loại sensor, mã số nội bộ, vị trí khu vực và mức độ báo động.

2.3.2 Tạo mô hình luồng dữ liệu

DFD giúp kỹ sư phần mềm có thể phát triển những mô hình phạm vi thông tin và phạm vi chức năng ở cùng 1 thời điểm. Khi DFD được tinh chế vào trong những mức độ chi tiết lớn hơn, nhà phân tích thực hiện 1 phân tích chức năng ẩn của hệ thống, qua đó hoàn thành nguyên tắc phân tích hoạt động thứ tư cho chức năng. Tại 1 thời điểm, việc tinh chế DFD đưa ra kết quả tinh chế tương tự của dữ liệu khi nó di chuyển thông qua các quá trình mà gồm cả ứng dụng.

Một vài hướng dẫn đơn giản:

(1) mức 0 DFD nên miêu tả phần mềm/ hệ thống như 1 hình tròn.

(2) đầu vào, đầu ra căn bản nên được ghi chú cẩn thận.

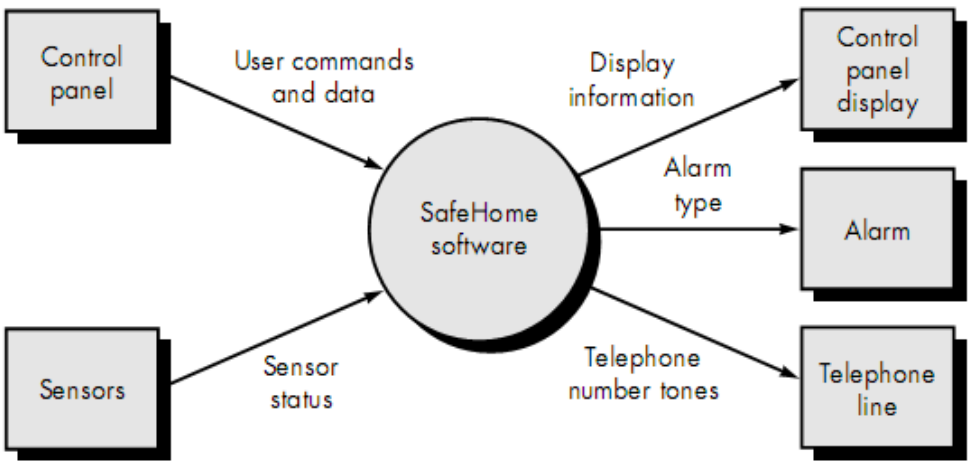
(3) việc tinh chế nên bắt đầu bằng việc cô lập những tiến trình, những đối tượng dữ liệu, những nơi lưu trữ mà được thể hiện ở mức độ tiếp theo.

(4) tất cả dấu mũi tên và hình tròn nên được gắn nhãn với những tên có nghĩa.

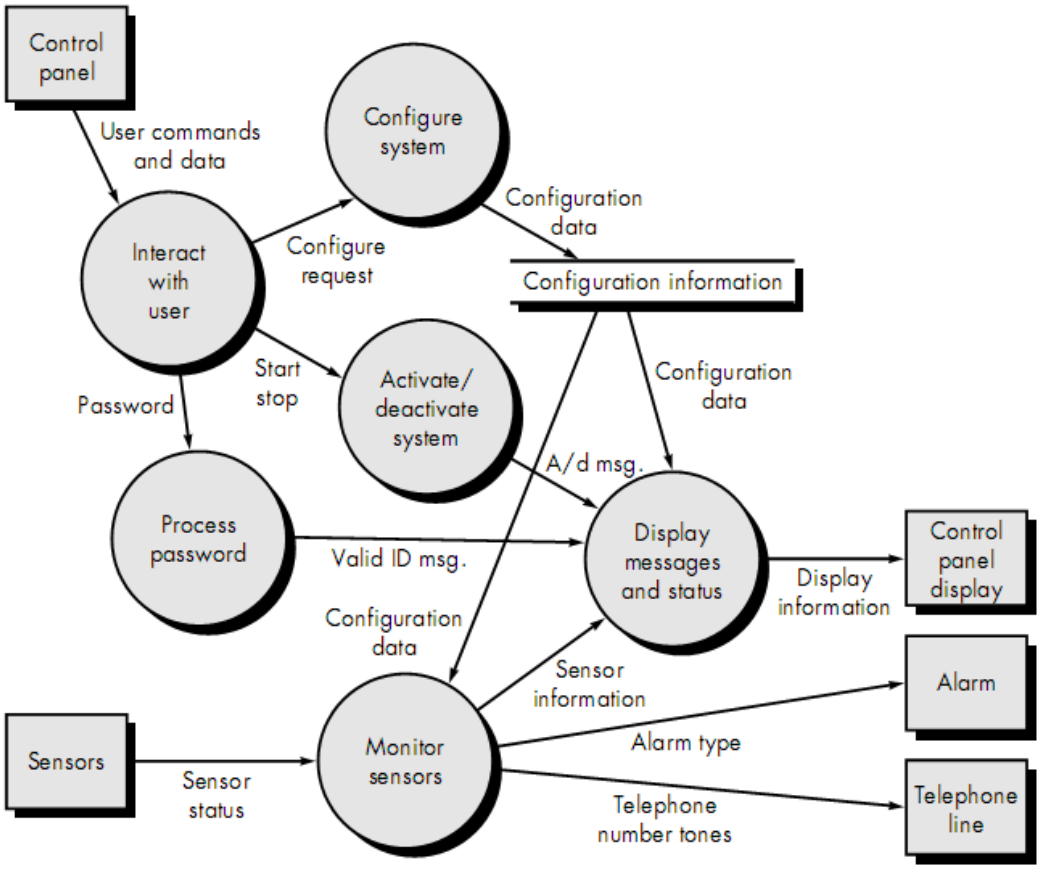
(5) sự liên tục của luồng thông tin nên được duy trì từ cấp độ này đến cấp độ khác.

(6) tại 1 thời điểm, 1 hình tròn nên được tinh chế.

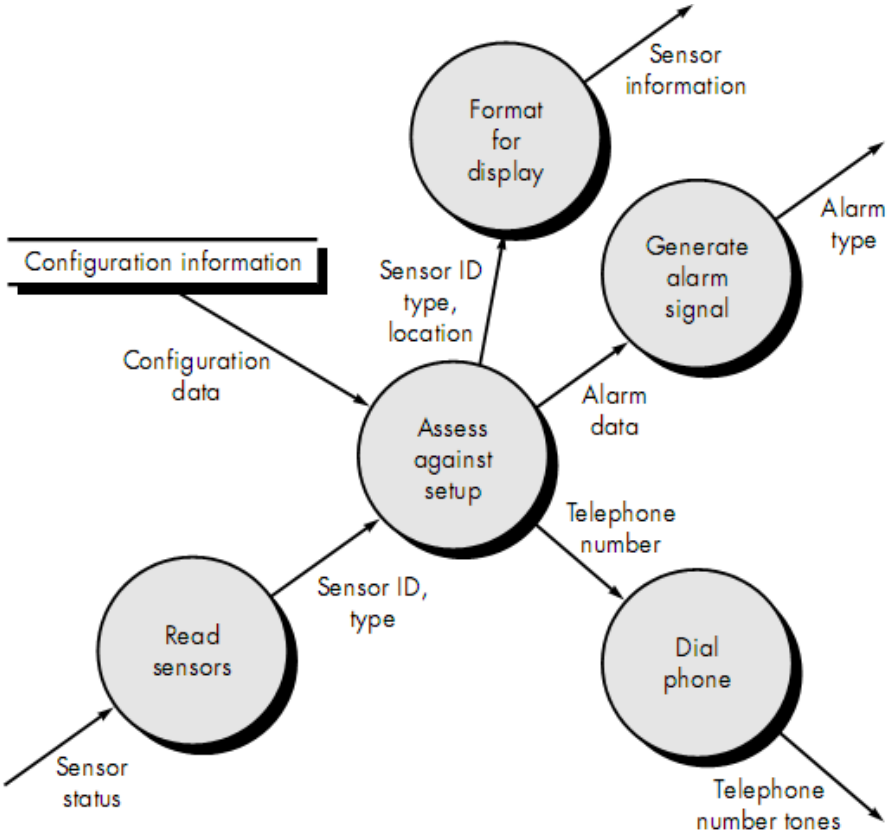
Xét ví dụ hệ thống safehome



Hình 2.3.2(a): mức 0 DFD



Hình 2.3.2(b): mức 1 DFD



Hình 2.3.2(c): Mức 2: tinh chế tiến trình monitor sensor

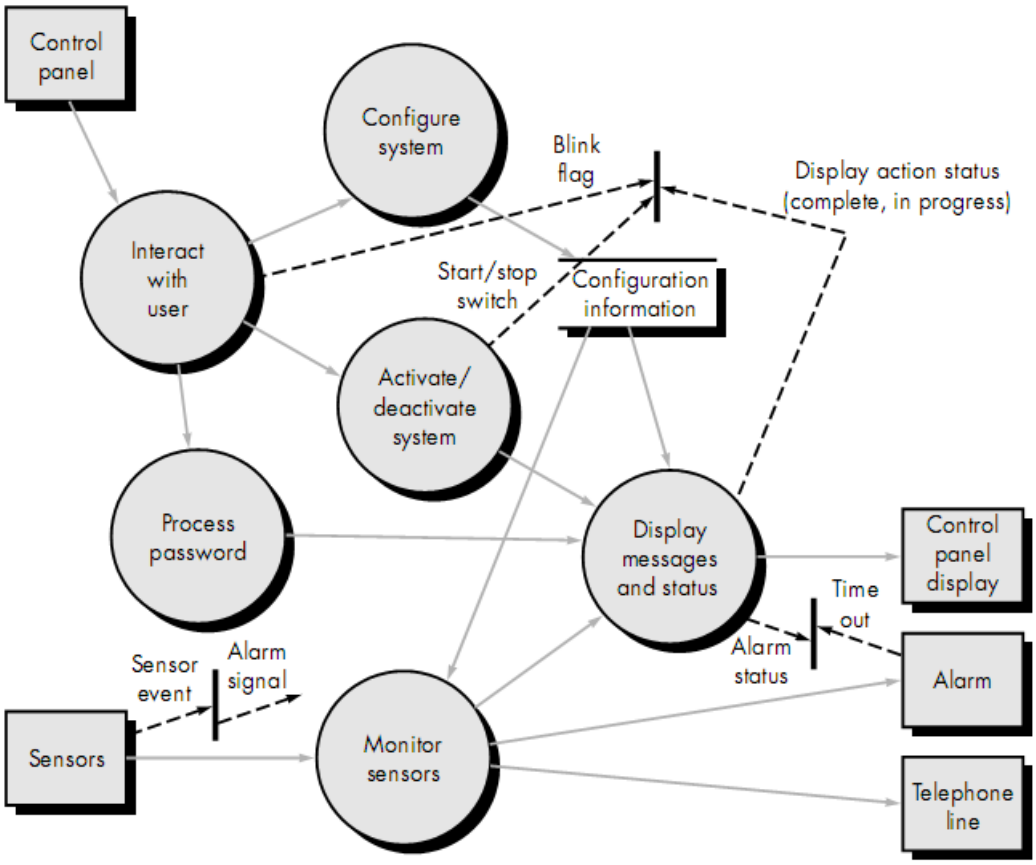
Việc tinh chế DFD tiếp tục cho đến khi mỗi hình tròn thực hiện 1 chức năng. Đó là, cho đến khi tiến trình đại diện với hình tròn thực hiện 1 chức năng mà sẽ dễ dàng được thực hiện như 1 thành phần chương trình.

2.3.3 Tạo mô hình luồng điều khiển (Control Flow Diagram CFD)

Đối với những ứng dụng xử lý dữ liệu, mô hình dữ liệu và DFD là cần thiết để có cái nhìn sâu về yêu cầu phần mềm. Tuy nhiên, đối với những ứng dụng lớn - được kiểm soát bởi sự kiện chứ không phải dữ liệu, tạo ra những thông tin điều khiển chứ không chỉ là báo cáo, hiển thị lên màn hình, xử lý những thông tin với thời gian lớn và hiệu suất cao- đòi hỏi sử dụng mô hình luồng điều khiển bên cạnh mô hình luồng dữ liệu.

Một vài hướng dẫn để chọn sự kiện:

- Liệt kê tất cả các sensor được đọc bởi phần mềm.
- Liệt kê tất cả các điều kiện ngắt.
- Liệt kê tất cả các điều kiện dữ liệu.
- …

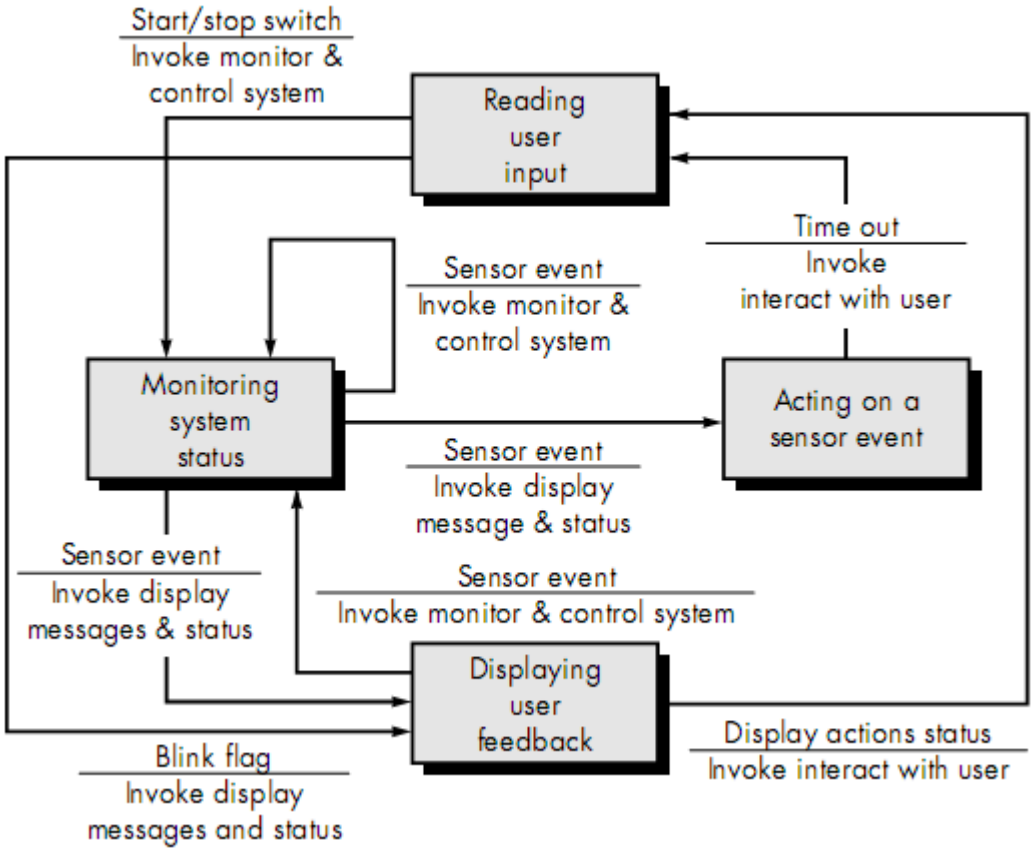


Hình 2.3.3: Mức 1 CFD

Trong số những sự kiện và item điều kiện được chỉ trong hình, chú ý đến sensor event (như: 1 sensor bị nhả ra), blink flag (một tín hiệu nhấp nháy trên màn hình LCD), và start/ stop switch (1 tín hiệu bật/ tắt hệ thống). Khi sự kiện đưa vào cửa sổ CSPEC (Control Specification) từ thế giới bên ngoài, nó ngụ ý rằng CSPEC sẽ kích hoạt 1 hay nhiều tiến trình chỉ ra trong CFD.

2.3.4 Đặc tả điều khiển (Control Specification CSPEC)

CSPEC miêu tả cách xử lý của hệ thống (ở mức độ mà nó được tham chiếu đến) trong 2 cách khác nhau: biểu đồ chuyển đổi trạng thái (state transition diagram STD) (1 đặc tả liên tiếp của cách xử lý), và bảng kích hoạt chương trình (program activation table PAT) – 1 đặc tả tổ hợp của cách xử lý.



Hình 2.3.4: Biểu đồ chuyển đổi trạng thái cho mức 1 CFD

Các mũi tên chuyển đổi được gán nhãn chỉ ra cách hệ thống hồi đáp lại các sự kiện khi nó đi ngang qua 4 trạng thái được xác định ở mức độ này. Thông qua STD, kỹ sư phần mềm có thể xác định cách xử lý của hệ thống và quan trọng hơn, có thể biết chắc được là có lỗ hổng nào trong cách xử lý được chỉ ra không.

Trong hình 2.3.4, khi gặp phải *start/ stop switch*, trạng thái *reading user input* chỉ chuyển thành 1 trạng thái là *monitoring system status.*

PAT thể hiện thông tin chứa trong STD trong ngữ cảnh của tiến trình, chứ không phải trạng thái. PAT thể hiện sự liên tục thỉnh nguyện (invocation sequence) của những hình tròn trong DFD. PAT chỉ ra tiến trình nào sẽ được gọi lên khi sự kiện xuất hiện. PAT cũng có thể được dùng như 1 hướng dẫn cho người xây dựng quyền hành để kiểm soát các tiến trình thể hiện ở mức độ này.

Input events						
Sensor event	0	0	0	0	1	0
Blink flag	0	0	1	1	0	0
Start/stop switch	0	1	0	0	0	0
Display action status						
Complete	0	0	0	1	0	0
In progress	0	0	1	0	0	0
Time out	0	0	0	0	0	1
Output						
Alarm signal	0	0	0	0	1	0
Process activation						
Monitor and control system	0	1	0	0	1	1
Activate/deactivate system	0	1	0	0	0	0
Display messages and status	1	0	1	1	1	1
Interact with user	1	0	0	1	0	1

Hình 2.3.4(b): PAT cho mức 1 DFD

2.3.5 Đặc tả quá trình (Process Specification PSPEC)

PSPEC được dùng để miêu tả tất cả những tiến trình mô hình luồng xuất hiện ở mức độ cuối cùng của việc tinh chế. Nội dung của PSPEC có thể bao gồm văn bản tường thuật, miêu tả ngôn ngữ thiết kế chương trình (program design language PDL) của thuật toán quá trình, phương trình toán học, bảng biểu, biểu đồ hay bảng xếp hạng. Bằng cách cung cấp PSPEC để hoàn thành mỗi hình tròn trong mô hình luồng, kỹ sư phần mềm tạo ra 1 đặc tả nhỏ mà có thể phục vụ như bước đầu tiên trong việc tạo ra đặc tả yêu cầu phần mềm và như là 1 hướng dẫn cho các thiết kế của thành phần phần mềm mà sẽ thực hiện quá trình.

Xem xét *process password* trong hình 2.3.2(b)

PSPEC: process password

The *process password* transform performs all password validation for the *SafeHome* system. *Process password* receives a four-digit password from the *interact with user* function. The password is first compared to the master password stored within the system. If the master password matches, <valid id message = true> is passed to the *message and status display* function. If the master password does not match, the four digits are compared to a table of secondary passwords (these may be assigned to house guests and/or workers who require entry to the home when the owner is not present). If the password matches an entry within the table, <valid id message = true> is passed to the *message and status display* function. If there is no match, <valid id message = false> is passed to the *message and status display function*.

2.4 Phân tích hướng sự vật và mô hình hóa dữ liệu

Phần này hướng đến cách sử dụng UML, sinh viên tự tìm hiểu.

2.5 Các kỹ thuật phân tích và phương pháp hình thức khác

Có nhiều phương pháp phân tích được sử dụng, nhưng trong đó có 3 phương thức quan trọng là:

- Data Structured Systems Development (DSSD).
- Jackson System Development (JSD).
- Structured Analysis and Design Technique (SADT).

Sinh viên tự tìm hiểu thêm về những phương pháp phân tích này.

Chương 3: THIẾT KẾ VÀ CÀI ĐẶT PHẦN MỀM

3.1 Các nền tảng thiết kế phần mềm

3.1.1 Khái niệm

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

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

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

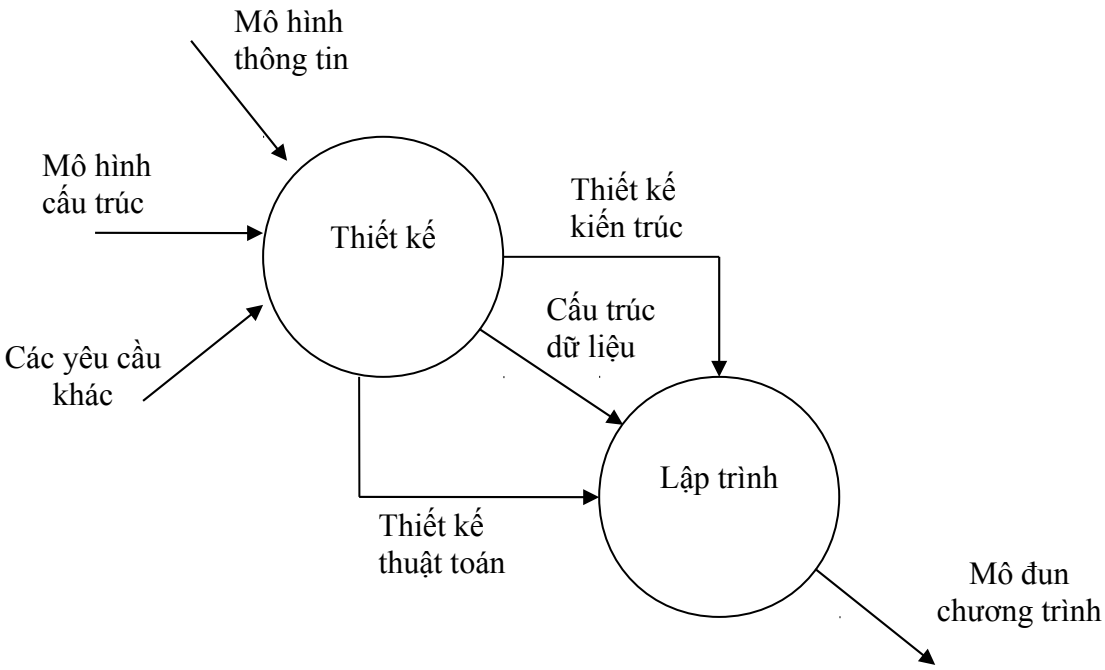
Mô hình chung của một thiết kế phần mềm là một đồ thị có hướng, các nút biểu diễn các thực thể có trong thiết kế, các liên kết biểu diễn các mỗi quan hệ giữa các thực thể đó. Hoạt động thiết kế là một loại hoạt động đặc biệt:

- Là một quá trình sáng tạo, đòi hỏi có kinh nghiệm và sự nhanh nhạy và sáng tạo
- Cần phải được thực hành và học bằng kinh nghiệm, bằng khảo sát các hệ đang tồn tại, chỉ học bằng sách vở là không đủ.

3.1.2 Tầm quan trọng

Tầm quan trọng của thiết kế phần mềm có thể được phát biểu bằng một từ "chất lượng". Thiết kế là nơi chất lượng phần mềm được nuôi dưỡng trong quá trình phát triển: cung cấp cách biểu diễn phần mềm có thể được xác nhận về chất lượng, là cách duy nhất mà chúng ta có thể chuyển hóa một cách chính xác các yêu cầu của khách hàng thành sản phẩm hay hệ thống phần mềm cuối cùng. Thiết kế phần mềm là công cụ giao tiếp làm cơ sở để có thể mô tả một cách đầy đủ các dịch vụ của hệ thống, để quản lý các rủi ro và lựa chọn giải pháp thích hợp. Thiết kế phần mềm phục vụ như một nền tảng cho mọi bước kỹ nghệ phần mềm và bảo trì. Không có thiết kế có nguy cơ sản sinh một hệ thống

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



Hình 3.1.2: Vai trò của thiết kế phần mềm trong quá trình kỹ nghệ.

3.1.3 Quá trình thiết kế

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

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

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

3. Mô tả trừu tượng cho mỗi nội dung trong giải pháp. Trước khi tạo ra các tư liệu chính thức người thiết kế cần phải xây dựng một mô tả ban đầu sơ

khai rồi chi tiết hóa nó. Các sai sót và khiếm khuyết trong mỗi mức thiết kế trước đó được phát hiện và phải được chỉnh sửa trước khi lập tư liệu thiết kế.

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

Các nội dung chính của thiết kế là:

- Thiết kế kiến trúc: Xác định hệ tổng thể phần mềm bao gồm các hệ con và các quan hệ giữa chúng và ghi thành tài liệu
- Đặc tả trừu tượng: các đặc tả trừu tượng cho mỗi hệ con về các dịch vụ mà nó cung cấp cũng như các ràng buộc chúng phải tuân thủ.
- Thiết kế giao diện: giao diện của từng hệ con với các hệ con khác được thiết kế và ghi thành tài liệu; đặc tả giao diện không được mơ hồ và cho phép sử dụng hệ con đó mà không cần biết về thiết kế nội tại của nó.
- Thiết kế các thành phần: các dịch vụ mà một hệ con cung cấp được phân chia cho các thành phần hợp thành của nó.
- Thiết kế cấu trúc dữ liệu: thiết kế chi tiết và đặc tả các cấu trúc dữ liệu (các mô hình về thế giới thực cần xử lý) được dùng trong việc thực hiện hệ thống.
- Thiết kế thuật toán: các thuật toán được dùng cho các dịch vụ được thiết kế chi tiết và được đặc tả.

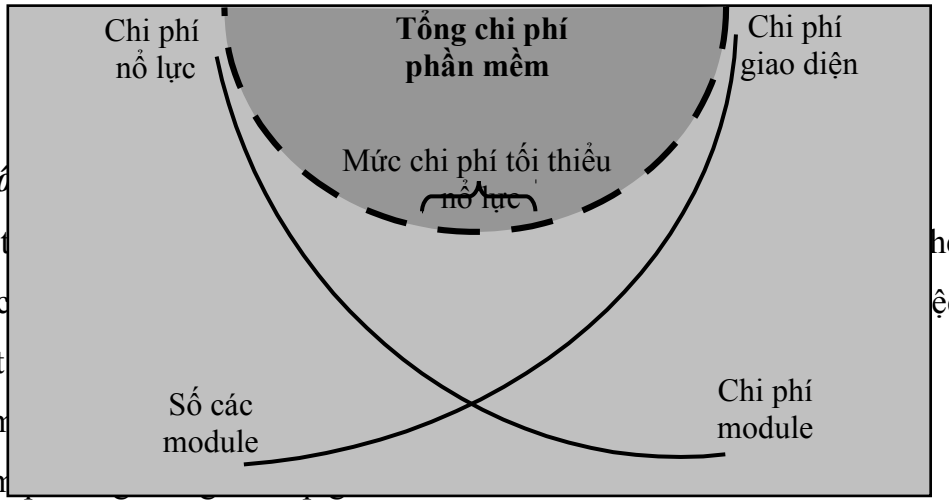
Quá trình này được lặp lại cho đến khi các thành phần hợp thành của mỗi hệ con được xác định đều có thể ánh xạ trực tiếp vào các thành phần ngôn ngữ lập trình, chẳng hạn như các gói, các thủ tục và các hàm.

3.1.4 Cơ sở của thiết kế

Phần mềm được chia thành các thành phần có tên riêng biệt và xác định được địa chỉ, gọi là các mô đun, được tích hợp để thỏa mãn yêu cầu của vấn đề. Người ta nói rằng: tính môđun là thuộc tính riêng của phần mềm cho phép một

chương trình trở nên quản lý được theo cách thông minh. Người đọc không thể nào hiểu thấu phần mềm nguyên khối (như một chương trình lớn chỉ gồm một môđun). Điều này dẫn đến kết luận "chia để trị" sẽ dễ giải quyết một vấn đề phức tạp hơn khi chia nó thành những phần quản lý được. Với cùng một tập hợp các yêu cầu, nhiều môđun hơn có nghĩa là kích cỡ từng môđun nhỏ; độ phức tạp giảm và chi phí cho phát triển môđun giảm. Nhưng khi số các mô đun tăng lên thì nỗ lực liên kết chúng bằng việc làm giao diện cho các môđun cũng tăng lên. Đặc trưng này dẫn đến đường cong tổng chi phí (nỗ lực) như trong hình 3.1.4.

Chúng ta nên mô đun hóa nhưng cần phải duy trì chi phí trong vùng lân cận của chi phí tối thiểu. Môđun hóa còn chưa đủ hay quá mức đều nên tránh. Một gợi ý cho kích cỡ của các môđun cơ sở là mỗi môđun đảm nhận một chức năng cơ bản.



3.1.5 Mô

Một hế
giới thực c ệc
mô tả thiết

- Làm
- Làm
- Cung cấp đủ thông tin cho những người bảo trì hệ thống

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

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

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

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

- Cấu trúc tuần tự
- Cấu trúc rẽ nhánh
- Cấu trúc lặp

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

- Dạng văn bản phi hình thức: Mô tả bằng ngôn ngữ tự nhiên các thông tin không thể hình thức hóa được như các thông tin phi chức năng. Bên cạnh các cách mô tả khác, mô tả văn bản thường được bổ sung để làm cho thiết kế được đầy đủ và dễ hiểu hơn.
- Các biểu đồ: Các biểu đồ được dùng để thể hiện các mối quan hệ giữa các thành phần lập lên hệ thống và là mô hình mô tả thế giới thực. Việc mô tả đồ thị của các thiết kế là rất có lợi vì tính trực quan và cho một bức tranh tổng thể về hệ thống. Trong thời gian gần đây, người ta đã xây dựng được một ngôn ngữ đồ thị dành riêng cho các thiết kế phần mềm với tên gọi: ngôn ngữ mô hình hóa thống nhất (Unified Modeling Model - UML). Tại mức thiết kế chi tiết, có một số các dạng biểu đồ hay được sử dụng là flow chart, JSP, NassiuShneiderman diagrams.

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

Procedure Write Name
if sex = male
write "Mr."
else
write "Ms."
endif
write name
end Procedure

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

3.1.6 Chất lượng thiết kế

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

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

*1) Sự kết dính (Cohesion) :*Sự kết dính của một môđun là độ đo về tính khớp lại với nhau của các phần trong môđun đó. Nếu một môđun chỉ thực hiện một chức năng logic hoặc là một thực thể logic, tức là tất cả các bộ phận của

môđun đó đều tham gia vào việc thực hiện một công việc thì độ kết dính là cao. Nếu một hoặc nhiều bộ phận không tham gia trực tiếp vào việc chức năng logic đó thì mức độ kết dính của nó là thấp. Thiết kế là tốt khi độ kết dính cao. Khi đó chúng ta sẽ dễ dàng hiểu được từng môđun và việc sửa chữa một môđun sẽ không (ít) ảnh hưởng tới các môđun khác. Constantine và Yourdon định ra 7 mức kết dính theo thứ tự tăng dần sau đây:

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

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

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

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

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

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

g. Kết dính chức năng: Mỗi phần của môđun đều là cần thiết để thi hành cùng một chức năng nào đó. Các lớp kết dính này không được định nghĩa chặt chẽ và cũng không phải luôn luôn xác định được. Một đối tượng kết dính nếu nó thể hiện như một thực thể đơn: tất cả các phép toán trên thực thể đó đều nằm trong thực thể đó. Vậy có thể xác định một lớp kết dính nữa là:

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

2) Sự ghép nối (Coupling):Ghép nối là độ đo sự nối ghép với nhau giữa các đơn vị (môđun) của hệ thống. Hệ thống có nối ghép cao thì các môđun phụ thuộc lẫn nhau lớn. Hệ thống nối ghép lỏng lẻo thì các môđun là độc lập hoặc là tương đối độc lập với nhau và chúng ta sẽ dễ bảo trì nó. Các mô đun được ghép nối chặt chẽ nếu chúng dùng các biến chung và nếu chúng trao đổi các

thông tin điều khiển (ghép nối chung nhau và ghép nối điều khiển). Ghép nối lỏng lẻo đạt được khi bảo đảm rằng các thông tin cục bộ được che dấu trong các môđun và các môđun trao đổi thông tin thông qua danh sách tham số (giao diện) xác định. Có thể chia ghép nối thành các mức từ chặt chẽ đến lỏng lẻo như sau:

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

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

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

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

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

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

Ưu việt của thiết kế hướng đối tượng là do bản chất che dấu thông tin của đối tượng dẫn tới việc tạo ra các hệ ghép nối lỏng lẻo. Việc thừa kế trong hệ thống hướng đối tượng lại dẫn tới một dạng khác của ghép nối, ghép nối giữa đối tượng mức cao và đối tượng kế thừa nó.

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

a. Tính kết dính: có thể hiểu được thành phần đó mà không cần tham khảo tới một thành phần nào khác hay không?

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

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

d. Độ phức tạp: độ phức tạp của các thuật toán được dùng để thực hiện thành phần đó như thế nào? Độ phức tạp cao ám chỉ nhiều quan hệ giữa các thành phần khác nhau của thành phần thiết kế đó và một cấu trúc logic phức tạp mà nó dính líu đến độ sâu lồng nhau của cấu trúc ifưthenưelsse. Các thành phần phức tạp là khó hiểu, vì thế người thiết kế nên làm cho thiết kế thành phần càng đơn giản càng tốt. Đa số công việc về đo chất lượng thiết kế được tập trung vào cố gắng đo độ phức tạp của thành phần và từ đó thu được một vài độ đo về sự dễ hiểu của thành phần. Độ phức tạp phản ánh độ dễ hiểu, nhưng cũng có một số nhân tố khác ảnh hưởng đến độ dễ hiểu, chẳng hạn như tổ chức dữ liệu và kiểu cách mô tả thiết kế. Các số đo độ phức tạp có thể chỉ cung cấp một chỉ số cho độ dễ hiểu của một thành phần.

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

Để có độ thích nghi thì hệ thống còn cần phải phải tự chứa. Muốn là tự chứa một cách hoàn toàn thì một hệ thống không nên dùng các thành phần khác được xác định ngoại lai. Tuy nhiên, điều đó lại mâu thuẫn với kinh nghiệm nói rằng các thành phần hiện có nên là dùng lại được. Vậy là cần có một cân bằng giữa tính ưu việt của sự dùng lại các thành phần và sự mất mát tính thích nghi được của hệ thống. Một trong những ưu việt chính của kế thừa trong thiết kế hướng đối tượng là các thành phần này có thể sẵn sàng thích nghi được. Cơ cấu thích nghi được này không dựa trên việc cải biên thành phần đã có mà dựa trên việc tạo ra một thành phần mới thừa kế các thuộc tính và các chức năng của thành phần đó. Chúng ta chỉ cần thêm các thuộc tính và chức năng cần thiết cho thành phần mới. Các thành phần khác dựa trên thành phần cơ bản đó sẽ không bị ảnh hưởng gì.

3.2 Các phương pháp thiết kế

3.2.1 Thiết kế hướng dòng dữ liệu (hướng chức năng)

3.2.1.1 Cách tiếp cận hướng chức năng

Thiết kế hướng chức năng là một cách tiếp cận thiết kế phần mềm trong đó bản thiết kế được phân giải thành một bộ các đơn thể tác động lẫn nhau, mà mỗi đơn thể có một chức năng được xác định rõ ràng. Các chức năng có các trạng thái cục bộ nhưng chúng chia sẻ với nhau trạng thái hệ thống, trạng thái này là tập trung và mọi chức năng đều có thể truy cập được. Nhiều tổ chức đã phát triển các chuẩn và các phương pháp dựa trên sự phân giải chức năng. Nhiều phương pháp thiết kế kết hợp với công cụ CASE đều là hướng chức năng. Vô khối các hệ thống đã được phát triển bằng cách sử dụng phương pháp tiếp cận hướng chức năng. Các hệ thống đó sẽ được bảo trì cho một tương lai xa xôi. Bởi vậy thiết kế hướng chức năng vẫn sẽ còn được tiếp tục sử dụng rộng rãi.

Trong thiết kế hướng chức năng, người ta dùng các biểu đồ luồng dữ liệu (mô tả việc xử lý dữ liệu), các lược đồ cấu trúc (nó chỉ ra cấu trúc của phần mềm), và các mô tả thiết kế chi tiết. Thiết kế hướng chức năng gắn với các chi tiết của một thuật toán của chức năng đó nhưng các thông tin trạng thái hệ thống là không bị che dấu. Việc thay đổi một chức năng và cách nó sử dụng trạng thái của hệ thống có thể gây ra những tương tác bất ngờ đối với các chức năng khác. Cách tiếp cận chức năng để thiết kế là tốt nhất khi mà khối lượng thông tin trạng thái hệ thống được làm nhỏ nhất và thông tin dùng chung nhau là rõ ràng.

3.2.1.2 Biểu đồ luồng dữ liệu

Biểu đồ luồng dữ liệu chỉ ra cách thức biến đổi dữ liệu vào thành dữ liệu ra thông qua một dãy các phép biến đổi. Bước thứ nhất của thiết kế hướng chức năng là phát triển một biểu đồ luồng dữ liệu hệ thống. Biểu đồ này không nhất thiết bao gồm các thông tin điều khiển nhưng nên lập tư liệu các phép biến đổi dữ liệu. Biểu đồ luồng dữ liệu là một phần hợp nhất của một số các phương pháp thiết kế và các công cụ CASE thường trợ giúp cho việc tạo ra biểu đồ luồng dữ liệu.

3.2.1.3 Lược đồ cấu trúc

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

3.2.1.4 Các từ điển dữ liệu

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

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

3.2.2 Thiết kế hướng đối tượng

3.2.2.1 Cách tiếp cận hướng đối tượng

Thiết kế hướng đối tượng dựa trên chiến lược che dấu thông tin cấu trúc vào bên trong các thành phần. Cái đó ngầm hiểu rằng việc kết hợp điều khiển logic và cấu trúc dữ liệu được thực hiện trong thiết kế càng chậm càng tốt. Liên lạc thông qua các thông tin trạng thái dùng chung (các biến tổng thể) là ít nhất, nhờ vậy khả năng hiểu được nâng lên. Thiết kế là tương đối dễ thay đổi vì sự thay đổi cấu trúc một thành phần có thể không cần quan tâm tới các hiệu ứng phụ trên các thành phần khác.

Việc che dấu thông tin trong thiết kế hướng đối tượng là dựa trên sự nhìn hệ phần mềm như là một bộ các đối tượng tương tác với nhau chứ không phải là bộ các chức năng như cách tiếp cận chức năng. Các đối tượng có một trạng thái riêng được che dấu và các phép toán trên trạng thái đó. Thiết kế biểu thị các dịch vụ được yêu cầu cùng với những hỗ trợ mà các đối tượng có tương tác với nó cung cấp.

3.2.2.2 Ba đặc trưng của thiết kế hướng đối tượng

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

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

2. Các đối tượng là các thực thể độc lập, dễ thay đổi vì rằng tất cả các trạng thái và các thông tin biểu diễn chỉ ảnh hưởng trong phạm vi chính đối tượng đó thôi. Các thay đổi về biểu diễn thông tin có thể được thực hiện không cần sự tham khảo tới các đối tượng khác.

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

3.2.2.3 Cơ sở của thiết kế hướng đối tượng

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

- Cấu trúc dữ liệu (thuộc tính)
- Hàm, thủ tục (chức năng)
- Giao diện (cung cấp khả năng trao đổi dữ liệu đối với các lớp khác, về bản chất là các chức năng của đối tượng)
- Sự kế thừa

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

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

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

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

2.) Hoạt động: đối tượng nhận các thông báo và thực hiện các chức năng được yêu cầu.

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

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

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

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

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

3.2.2.4 Các bước thiết kế

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

1. Xác định lớp đối tượng.

2. Xác định thuộc tính cho lớp: các biến của lớp

3. Xác định hành vi (chức năng): các hàm

4. Xác định tương tác giữa các lớp đối tượng: giao diện (thông báo).

5. Áp dụng tính kế thừa: xây dựng các lớp trừu tượng có các thuộc tính chung, đây là một khâu đặc trưng của thiết kế hướng đối tượng.

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

Phương pháp xác định đối tượng Xác định đối tượng là một trong nhưng công đoạn thiết kế quan trọng, phụ thuộc nhiều vào kinh nghiệm và bài toán cụ

thể. Có một số phương pháp được đề xuất, một trong các phương pháp này là phân tích từ vựng của "câu yêu cầu". Cụ thể là danh từ trong câu yêu cầu sẽ được coi là đối tượng và động từ sẽ được coi là chức năng.

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

3.2.2.5 Ưu nhược điểm của thiết kế hướng đối tượng

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

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

• Các đối tượng là các thành phần dùng lại được thích hợp do tính độc lập của chúng và khả năng kế thừa cao.

• Có một vài lớp hệ thống thể hiện phản ánh quan hệ rõ ràng giữa các thực thể có thực (chẳng hạn như các thành phần phần cứng) với các đối tượng điều khiển nó trong hệ thống. Điều này đạt được tính dễ hiểu của thiết kế.

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

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

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

Một thiết kế hướng đối tượng cũng có thể được thực hiện trong một ngôn ngữ thủ tục kiểu như PASCAL hoặc C (không có các đặc điểm bao gói như vậy). Ada không phải là ngôn ngữ lập trình hướng đối tượng vì nó không trợ giúp sự thừa kế của các lớp, nhưng lại có thể thực hiện các đối tượng trong Ada bằng cách sử dụng các gói hoặc các nhiệm vụ (tasks), do đó Ada cũng được dùng để mô tả các thiết kế hướng đối tượng.

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

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

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

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

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

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

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

3.2.3 Thiết kế hướng dữ liệu

3.2.4 Thiết kế giao diện

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

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

- Khía cạnh nghiệp vụ: người dùng thông qua giao diện để tương tác với hệ thống, đây là khâu nghiệp vụ thủ công duy nhất do đó nếu được thiết kế tốt sẽ nâng cao tốc độ xử lý công việc và dẫn tới hiệu quả kinh tế cao.
- Khía cạnh thương mại: đối với các sản phẩm bán hàng loạt, giao diện được thiết kế tốt (dễ sử dụng, đẹp) sẽ gây ấn tượng với khách hàng và là yếu tố chính khi khách hàng chọn mua sản phẩm.

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

Có hai dòng giao diện chính là:

- Giao diện dòng lệnh: là loại giao diện đơn giản nhất, thường được thiết kế gắn chặt với chương trình và có tính di chuyển cao (tương đương với chương trình). Giao diện dòng lệnh phù hợp với các ứng dụng thuần túy xử lý dữ liệu, nhất là đối với các chương trình mà đầu ra là đầu vào của chương trình khác. Giao diện dòng lệnh gọn nhẹ, dễ xây dựng nhưng thường khó học, khó sử dụng và chỉ phù hợp với người dùng chuyên nghiệp trong các ứng dụng đặc thù.
- Giao diện đồ họa: sử dụng các cửa sổ, menu, icon... cho phép người dùng có thể truy cập song song đến nhiều thông tin khác nhau; người dùng thường tương tác bằng cách phối hợp cả bàn phím và con chuột; giao diện đồ họa dễ học, dễ sử dụng và trở nên rất thông dụng và có độ chuẩn hóa cao.

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

- Giao diện đồ họa (GUI) truyền thống: là giao diện đồ họa được thiết kế có độ liên kết cao với chương trình (được xây dựng cùng ngôn ngữ, cùng bộ công cụ...), hầu hết các chương trình trên máy tính cá nhân sử dụng loại giao diện này.
- X protocol: giao diện đồ họa sử dụng giao thức X protocol, phổ biến trên các máy Unix/Linux. Loại giao diện này có ưu diểm là có thể hoạt động độc lập với khối chương trình còn lại, tức là ta có thể chạy giao diện trên một máy tính trong khi đó phần xử lý bên trong lại hoạt động trên một máy khác. Đáng tiếc là phương thức này vẫn chưa phổ biến trên các máy tính cá nhân (chạy hệ điều hành MS Windows).
- Client/server: một cách tiếp cận để hướng tới tính độc lập và khả chuyển của giao diện là xây dựng giao diện như là một chương trình client, tương tác với khối chương trình xử lý (server) thông qua các giao thức trao đổi thông tin trên mạng (TCP/IP).
- Web based: một trong các cách thức xây dựng giao diện phổ biến hiện nay là dựa trên nền web, sử dụng các trình duyệt web để trao dổi thông tin với server. Tuy có một số nhược điểm về an toàn thông tin và tốc độ nhưng với tính độc lập hoàn toàn với phần xử lý, độ chuẩn hóa cao và khả năng sẵn có trên hầu hết các thiết bị nối mạng, phương thức này đang được ứng dụng rộng rãi.

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

3.2.4.1 Một số vấn đề thiết kế

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

1. Thời gian phản hồi. Chúng ta cần quan tâm tới hai loại thời gian là

• Thời gian đáp ứng trung bình: là thời gian trung bình mà hệ thống phản hồi đối với một yêu cầu của người dùng. Thời gian để sinh

ra "kết quả thực sự" của yêu cầu sẽ phụ thuộc vào bản chất yêu cầu, thuật toán, tốc độ của máy tính, tuy nhiên chúng ta cần quan tâm khía cạnh tâm lý là nếu người dùng đợi quá lâu mà không nhận được thông tin gì thì họ sẽ nghĩ là có vấn đề và có thể sẽ tiến hành các thao tác ngoài mong đợi như lặp lại thao tác hay dừng hệ thống.

• Độ biến thiên của thời gian: độ biến thiên của thời gian cũng là đại lượng cần quan tâm. Nếu độ biến thiên lớn, ví dụ một thao tác thường được đáp ứng trong 1 giây mà có trường hợp phải mất 5 giây mới hoàn thành thì cũng có thể làm cho người dùng đưa ra các thao tác sai.

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

• Tích hợp: là tiện ích được tích hợp vào giao diện như nút Help cung cấp các thuyết minh về thao tác.

• Phụ thêm: là các tiện ích phụ thêm như các tài liệu trực tuyến.

• Macro: một số chương trình còn cho phép người dùng tự động hóa một số thao tác bằng các lệnh kiểu macro.

3. Thông báo. Các thông báo do hệ thống đưa ra cần

• Có nghĩa: mọi thông báo cần có nghĩa đối với người dùng.

• Ngắn gọn: các thông báo cần ngắn gọn đi vào bản chất vấn đề, đặc biệt là đối với kiểu giao diện dòng lệnh.

• Có tính xây dựng: thông báo nên có tính xây dựng như đưa ra các nguyên nhân và các hướng khắc phục.

3.2.4.2 Một số hướng dẫn thiết kế

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

• Hướng người dùng: đối tượng người dùng phải rõ ràng, giao diện nên được thiết kế có tính đến năng lực, thói quen... của loại đối tượng đó.

• Có khả năng tùy biến cao: giao diện nên có khả năng tùy biến cao để phục vụ cho các cá nhân có cách sử dụng khác nhau, các môi trường hoạt động khác nhau.

Các phần mềm trên hệ UNIX với giao diện theo chuẩn X protocol thường được thiết kế có độ tùy biến rất cao.

• Nhất quán: các biểu tượng, thông báo, cách thức nhập dữ liệu phải nhất quán và nên tuân theo các chuẩn thông thường.

• An toàn: nên có chế độ xác nhận lại đối với các thao tác nguy hiểm (như xóa dữ liệu) và nên có khả năng phục hồi trạng thái cũ (undo).

• Dễ học, dễ sử dụng: giao diện luôn cần được thiết kế hướng tới tính dễ học, dễ sử dụng, tức là không đòi hỏi người dùng phải có các năng lực đặc biệt. Ví dụ như không cần nhớ nhiều thao tác, không đòi hỏi phải thao tác nhanh, các thông tin trên màn hình dễ đọc... Một cách tốt nhất để xây dựng giao diện dễ học dễ sử dụng là tuân theo các chuẩn giao diện thông dụng.

3.3 Các ngôn ngữ lập trình

3.3.1 Nền tảng của ngôn ngữ lập trình

3.3.1.1 Kiểu dữ liệu, định nghĩa kiểu dữ liệu và kiểm tra kiểu dữ liệu

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

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

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

Các cấu trúc dữ liệu như mảng, bảng, danh sách tuyến tính,... là loại thứ ba của cấu trúc dữ liệu của ngôn ngữ. Các ngôn ngữ có thể hỗ trợ hoặc không

hỗ trợ kiểu này. Tuy nhiên, các kiểu dữ liệu đơn giản như mảng, danh sách tuyến tính,... thường được hầu hết các ngôn ngữ hỗ trợ.

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

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

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

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

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

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

Ví dụ, trong Fortran, trộn lẫn số thực và số nguyên trong toán tử toán học dẫn đến các kết quả không thể dự đoán được bởi vì kiểu đích (target type)

được quyết định bởi việc định nghĩa trường kết quả. Nếu trường kết quả được định nghĩa là thực, kết quả tính toán là số thực. Nếu trường kết quả được định nghĩa là integer, tiến trình sẽ làm tròn câu trả lời (số thực) và đưa ra kết quả là integer.

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

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

3.3.1.2 Chương trình con

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

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

Nói chung, khi sử dụng chương trình con, module chính gọi chương trình con làm những việc của nó và trả lại kết quả cho module chính. Khả năng hỗ trợ xử lý chương trình con đòi hỏi một hoặc nhiều hơn một mục vào hoặc điểm thoát. Xử lý Exit và Return cũng quan trọng khi chuyển quyền điều khiển giữa các module. Trong các trường hợp, càng nhiều cơ hội để vào và thoát khỏi module đã xác định trước, thì lập trình viên càng cần sự thành thạo, đảm bảo khả năng xử lý thành thạo, đảm bảo khả năng xử lý hoàn hảo. Theo các nhà lập trình cấu trúc, một module được thiết kế tốt nên có một điểm vào và một điểm ra. Module một vào và một ra ít gây lỗi hơn so với các module có nhiều mục vào, điểm ra.

3.3.1.3 Cấu trúc điều khiển

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

Sự tuần tự có hai dạng: giữa các dòng lệnh và trong dòng lệnh. Lập trình viên điều khiển sự tuần tự giữa các dòng lệnh (between-command sequencing) như là một trật tự của các lệnh, còn sự tuần tự trong dòng lệnh đó chính là thứ tự ưu tiên của các phép toán -operator precendence- dùng trong thao tác dữ liệu, nó được các ngôn ngữ quy định sẵn.

Với hai khối lệnh A, B tuân theo phương thức xử lý tuần tự thì với R là số lần thực hiện của khối lệnh ta có $R_A=R_B$. Cấu trúc tuần tự trong các ngôn ngữ lập trình thường tuân theo trật tự từ trái sang phải và từ trên xuống dưới.

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

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

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

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

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

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

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

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

3.3.1.4 Vào và ra dữ liệu

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

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

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

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

3.3.1.5 Quản lý bộ nhớ

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

3.3.1.6 Quản lý lỗi

Quản lý lỗi là mức chương trình được cài đặt để phát hiện và quản lý lỗi mà không phải dừng chương trình. Khả năng này sẽ làm tăng độ phức tạp và mở rộng phạm vi hữu ích của ngôn ngữ. Ví dụ Cobol cho phép ta chặn đứng lỗi dữ liệu như tràn, chia cho 0, nhưng lại không chặn được lỗi như định nghĩa dữ

liệu không hợp lệ, đọc quá cuối file,.... Ngược lại Smalltalk cho phép chặn được bất kỳ lỗi nào.

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

3.3.2 Các đặc trưng của ngôn ngữ cài đặt

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

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

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

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

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

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

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

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

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

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

3.3.3 Phân lớp và đánh giá về ngôn ngữ cài đặt

3.3.3.1 Các lớp ngôn ngữ

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

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

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

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

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

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

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

1. SQL- Structured Query Language

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

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

2. Focus

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

3. Basic - Beginers All purpose Symbolic Interchange Code

Được đánh giá một ngôn ngữ mạnh, cơ bản, trong ngôn ngữ không có những kỹ thuật phức tạp nhưng có toàn bộ các thành phần sơ đẳng. Basic là một ngôn ngữ dễ học, dễ viết, có tính thống nhất, chặt chẽ và các hệ thống trợ giúp kiểm tra tự động tốt. Các đặc trưng ngôn ngữ còn lại thay đổi tuỳ thuộc

vào các phiên bản Basic khác nhau. Khả năng khả chuyển của Basic kém bởi các lệnh vào ra thường phải thay đổi để phù hợp với môi trường.

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

4. Cobol- Common Business Oriented Language

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

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

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

5. Fortran - Formula Tranlastion

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

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

quá trình xử lý số sẽ cho kết quả khác nhau tuỳ thuộc vào định nghĩa những trường dữ liệu được xử lý.

6. C

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

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

7. Pascal

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

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

8. Prolog - Programming in Logic

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

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

9. Smalltalk

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

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

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

10. Ada

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

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

Khái niệm file trong Ada giống như trong Smalltalk được định nghĩa là một kiểu trong cấu trúc của ngôn ngữ và mọi quá trình xử lý hiện trên các kiểu. Giống như Smalltalk, sức mạnh của Ada là khả năng của nó trong việc hỗ

trợ xử lý hướng sự kiện như tên lửa dẫn đường trong hệ thống phòng thủ quốc gia.

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

3.3.4 Ngôn ngữ lập trình và và sự ảnh hưởng tới công nghệ phần mềm

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

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

Chương 4: ĐẢM BẢO, KIỂM CHỨNG VÀ BẢO TRÌ PHẦN MỀM

4.1 Tính đúng đắn của chương trình phần mềm

4.1.1 Khái niệm chung

Như ta đã biết, chương trình P là một bộ biến đổi tuần tự P để chuyển cái vào x thành ra cái y; ở đây x và y hoàn toàn được xác định trước.

Như vậy, một chương trình P được gọi là đúng nếu nó thực hiện chính xác những mục tiêu do người thiết kế đặc ra. Ta gọi:

+ Giả thiết {A} là mệnh đề được phát biểu để thể hiện tính chất của cái vào, gọi tắt là mệnh đề dữ liệu vào.

+ Kết luận {B} là mệnh đề được phát biểu để tính chất cần có của dữ liệu ra, gọi tắt là mệnh đề dữ liệu ra.

Do P có tính tuần tự và hữu hạn nên có thể biểu diễn P là một dãy liên tiếp các cấu trúc điều khiển P_1, P_2,....,P_n. Do vậy, bằng cách nào đó mà ta khẳng định được:

P_1 biến đổi {A} thành {A_1}

P_2 biến đổi {A_1} thành {A2}

....

P_n biến đổi {A_{n-1}} thành {A_n}

Và dựa vào quy tắc toán học, {A_n} có thể suy ra {B} thì ta có thể nói rằng P là đúng với cái vào {A} và cái ra {B}. Lúc này ký hiệu {A}P{B} hay

$$\{A\} \overset{P}{\Rightarrow} \{B\}.$$

Cần chú ý rằng $\{A\} \overset{P}{\Rightarrow} \{B\}$ là khác với $\{A\} \overset{L_O}{\Rightarrow} \{B\}$:mệnh đề {A} suy diễn ra mệnh đề {B} dựa vào các quy tắc toán học.

Nói cách khác, để chứng minh P là đúng,
ta chứng minh theo sơ đồ sau:

$\{A\}\ P_1\ \{A_1\}$

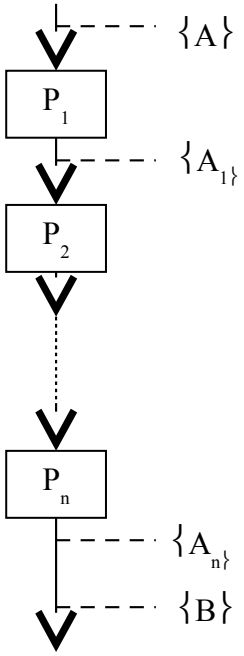
$\{A_1\}\ P_2\ \{A_2\}$

.....

.....

$\{A_{n-1}\}P_n\{A_n\}$

$\{A_n\} \overset{L}{\Rightarrow} \{B\}$



Ở đây, cần để ý là tính chất $\{A\}$ và tính chất $\{B\}$ có thể không liên quan đến nhau.

Ví dụ 1: Cho mệnh đề dữ liệu vào $\{A: x,y \in R; 0<x<1\}$

Đoạn trình $P = P_1 \cup P_2 \cup P_3 \cup P_4$ như sau:

x:=1/x+1; (P_1)

y:=y+1; (P_2)

x:=x+2; (P_3)

x:=x+y; (P_4)

và mệnh đề dữ liệu ra $\{B: x,y \in R; x>y+3\}$

Lúc này ta có dãy biến đổi tính chất dữ liệu vào/ ra như sau:

$\{A\}\ P_1\{A_1: x,y \in R; x>2\}$

$\{A_1\}P_1\{A_2: x,y \in R; x>2\}$

$\{A_2\}P_1\{A_3: x,y \in R; x>4\}$

$\{A_3\}P_1\{A_4: x,y \in R; x>y+4\}$

$\{A_4\} \overset{L}{\Rightarrow} \{B\}$

và

Vậy ta có kết luận $\{A\}P\{B\}$ hay nói cách khác là P đúng với dữ liệu vào $\{A\}$ và dữ liệu ra $\{B\}$.

Cần để ý rằng khí ta có dãy biến đổi tính chất dữ liệu vào và ra như sau:

$\{A\}\ P_1\ \{A_1\}$

$\{A_1\}\ P_2\ \{A_2\}$

.....

.....

$\{A_{n-1}\}P_n\{A_n\}$

$\{A_n\} \overset{L}{\Rightarrow} \{B\}$

Thì chưa thể kết luận được điều gì vì còn tuỳ thuộc vào các mệnh đề trung gian thu được $\{A_1\},\{A_2\},....\{A_n\}$ là đã "mạnh nhất" hay chưa.

4.1.2 Hệ tiên đề Hoare

4.1.2.1 Tiên đề 1: Tiên đề tuần tự

Nếu mệnh đề $\{A\}$ sau khi chịu tác động của khối cấu trúc điều khiển P ta được $\{B\}$ và mệnh đề $\{B\}$ sau khi chịu tác động của cấu trúc điều khiển Q ta được $\{C\}$ thì $\{A\}$ chịu tác động tuần tự P,Q sẽ thu được $\{C\}$.

Hay nói cách khác, đây chính là tiên đề về dãy thao tác: Nếu $\{A\}\ P\ \{B\}$ và $\{B\}\ Q\ \{C\}$ thì $\{A\}\ P,Q\ \{C\}$.

4.1.2.2 Tiên đề gán: tính chất của phép gán

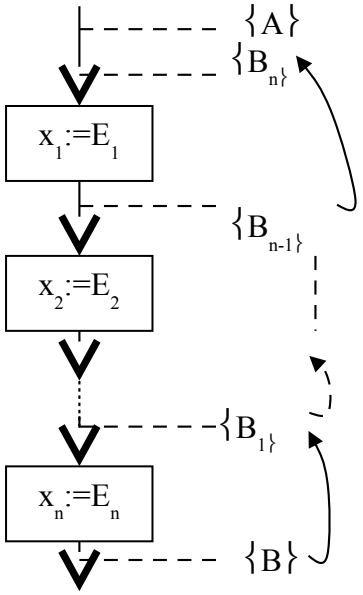
Điều kiện để có mệnh đề $\{B\}$ sau khi thực hiện lệnh gán x: = E (với E là một biểu thức) từ mệnh đề {A} thì trước đó ta phải có {A} suy dẫn được ra {B[x|E]}.

Mệnh đề {B[x|E]} là mệnh đề thu được từ {B} bằng phép thay thế mọi xuất hiện của x trong {B} bởi E. Tức là: $\{A\}$ x: = E $\{B\}$ thì $\{A\} \overset{L}{\Rightarrow} \{B[x|E]\}$

- *Kỹ thuật lần ngược của tiên đề gán*

Cho đoạn trình P gồm n phép gán $x_1:=E_1;\ x_2:=E_2;.....x_n:=E_n;$ để {A}P{B} thì

ta phải có $\{A\} \overset{L}{\Rightarrow} \{B_n\}$. Trong đó $\{B_n\}$ được xác định như sau



Trong đó các mệnh đề $\{B_i\}$ được xác định như sau:

$\{B_1\}$ là mệnh đề $\{B[x_n|E_n]\}$

$\{B_{n-1}\}$ là mệnh đề $\{B_{n-2}[x_2|E_2]\}$

$\{B_n\}$ là mệnh đề $\{B_{n-1}[x_1|E_1]\}$

Trong trường hợp $\{A\} \overset{L}{\not\Rightarrow} \{B_n\}$ thì ta nói P là có lỗi.

Ví dụ 2: (Xét ví dụ 1) Cho mệnh đề dữ liệu vào {A: x,y∈R; 0<x<1},

Đoạn trình P =$P_1 \cup P_2 \cup P_3 \cup P_4$ như sau:

x:=1/x+1; (P_1)

y:=y+1; (P_2)

x:=x+2; (P_3)

x:=x+y; (P_4)

và mệnh đề dữ liệu ra {B: x,y∈R; x>y+3}. Hãy khảo sát {A}P{B} hay không?

Ta có

{B[x|x+y]} ≡{B_1 : x+y,y∈R; x+y>y+3}

$\{B_1[x|x+2]\} \equiv \{B_2 : (x+2)+y, y \in R;\ (x+2)+y > y+3\}$

$\{B_2[y|y+1]\} \equiv \{B_3 : (x+2)+(y+1), (y+1) \in R;\ (x+2)+(y+1) > (y+1)+3\}$

$\{B_3[x|1/x+1]\} \equiv \{B_4 : ((1/x+1)+2)+(y+1), (y+1) \in R;\ ((1/x+1)+2)+(y+1) > (y+1)+3\}$

Rõ ràng ta có $\{A\} \xRightarrow{L} \{B_4\}$, nên {A}P{B}.

4.1.2.3 Tiên đề rẽ nhánh

i. Với mệnh đề dữ liệu vào {A}, mệnh đề dữ liệu ra {B}, biểu thức logic E,

$\{A, !E\} \xRightarrow{L} \{B\}$

và đoạn trình P. Nếu ta có {A, E}P{B} và thì ta nói rằng mệnh đề {A} và {B} tuân theo cấu trúc rẽ nhánh dạng khuyết với cấu trúc P và điều kiện lựa chọn E; tức là: {A} *if* E *then* P*;* {B}.

ii. Với mệnh đề dữ liệu vào {A}, mệnh đề dữ liệu ra {B}, biểu thức logic E,
và các đoạn trình P, Q. Nếu ta có {A, E}P{B} và {A,!E}Q{B} thì ta nói rằng mệnh đề {A} và {B} tuân theo cấu trúc rẽ nhánh dạng đủ với cấu trúc P, Q và điều kiện lựa chọn E; tức là: {A} *if* E *then* P *else* Q*;* {B}.

Ví dụ 3: Cho mệnh đề dữ liệu vào {A: x,y,q,r∈R, x=qy+r, 0≤r<2y}, đoạn trình P như sau:

```
If y≤r then
    Begin
        q:=q+1;
        r:=r-y;
    End;
```

Và mệnh đề dữ liệu ra {B: x,y,q,r∈R, x=qy+r, 0≤r<y}. Hãy xem {A}P{B}?

Áp dụng tính chất của phép gán, ta có:

i. {A,E: x,y,q,r∈R, x=qy+r, 0≤ r<2y, y≤ r}q:=q+1;r:=r-y;{B}

ii. {A,!E: x,y,q,r∈R, x=qy+r, 0≤ r<2y, y>r}=>{B}

do đó suy ra {A}P{B}.

4.1.2.4 Tính bất biến của chương trình

Cho mệnh đề dữ liệu vào {A} và đoạn trình P. Nếu ta có {A}P{A} thì ta nói rằng tính chất dữ liệu của mệnh đề {A} không thay đổi khi chịu sự tác động của đoạn trình P và lúc này người ta nói rằng mệnh đề {A} là bất biến đối với P, tức ta có: {A}P {A}.

Ví dụ 4: Ta có mệnh đề {A: x∈R, x>0} là bất biến đối với đoạn trình P: x:=x*x; vì ta có {A}P{A}.

4.1.2.5 Tiên đề lặp

Cho mệnh đề dữ liệu vào {A}, biểu thức logic E và đoạn trình P. Nếu mệnh đề {A} tuân theo cấu trúc lặp P với điều kiện lặp E thì mệnh đề {A} sẽ bất biến đối với P trong điều kiện E, tức là {A,E}P{A}, kết thúc vòng lặp ta có mệnh đề {A,!E}. Lúc này ta viết: {A} *while* E *do* P; {A,!E}.

Ví dụ 5: Cho x,y,z là 3 số nguyên không âm. Hãy viết chương trình để tính z=xy, biết rằng x,y được nhập từ bàn phím. Hãy khẳng định tính đúng của chương trình.

Ta có đoạn trình như sau:

Vào: x,y,z∈N; x=a; y=b;

Ra: x,y,z∈N; z=ab;

Chương trình P được viết:

z:=0;

```
while x>0 do
        Begin
                If (x mod 2)≠0 then z:=z+y;
                x=x div 2;
                y:=y*2;
        End;
Return z;
```

Ta cần phải khẳng định chương trình trên đúng với yêu cầu đặt ra.

Thật vậy, gọi mệnh đề thể hiện tính chất dữ liệu vào của chương trình {A} và mệnh đề thể hiện tính chất dữ liệu ra cần có {B}, ta có

{A: x,y,z∈ N; x=a; y=b;} và {B: x,y,z∈ N; z=ab;}

Ta cần chứng tỏ {A}P {B}.

+ Xét mệnh đề {C: x,y,z∈ N; ab=z+xy;}

+ Ta có {A} z:=0;{C}

+ Để chứng tỏ {C} là bất biến của đoạn trình

```
while x>0 do
        Begin
                If (x mod 2)≠0 then z:=z+y;
                x=x div 2;
                y:=y*2;
        End;
```

Ta cần có: {C,E: x,y,z∈ N; ab=z+xy;x>0}Q{C}, với đoạn trình Q như sau:

```
If (x mod 2)=0 then z:=z+y;
x=x div 2;
y:=y*2;
```

Theo tính chất của phép gán, ta có:

$\{C_1\}\equiv\{C[y|y*2]: x,y*2,z\in N; ab=z+x(y*2);\}$

$\{C_2\} \equiv \{C_1[x|(x \text{ div } 2)]: (x \text{ div } 2), y*2, z \in N; ab=z+(x \text{ div } 2)(y*2);\}$

Nên cần chứng tỏ:

$\{C,E: x,y,z \in N; ab=z+xy; x>0\}$ *If* $(x \bmod 2) \neq 0$ *then* $z:=z+y; \{C_2\}$

Dễ dàng ta có

i. $\{C,E,F: x,y,z \in N; ab=z+xy; x>0, (x \bmod 2) \neq 0\}$ $z:=z+y$ $\{C_2\}$; và

ii..$\{C,E,!F: x,y,z \in N; ab=z+xy; x>0, (x \bmod 2)=0\} \overset{L}{\Rightarrow} \{C_2\}$;

Vậy $\{C\}$ là bất biến của Q. Nên kết thúc Q, ta có mệnh đề $\{C,!E\}$.

+ Dễ dàng chứng tỏ: $\{C,!E\} \overset{L}{\Rightarrow} \{B\}$

Vậy ta có $\{A\}P\{B\}$, hay chương trình trên là đúng.

Để ý rằng: do $\{A,E\}P\{A\}$ nên trong trường hợp $\{A\} \overset{L}{\Rightarrow} E$ thì vòng lặp là vô hạn và không tồn tại mệnh đề $\{A, !E\}$.

4.2 Đảm bảo chất lượng phần mềm

Đảm bảo chất lượng phần mềm (Software Quality Assurance SQA) là 1 hoạt động được áp dụng trong suốt tiến trình phần mềm. SQA bao gồm:

(1): phương thức quản lý chất lượng.

(2): kỹ thuật CNPM hiệu quả (phương pháp và công cụ).

(3): review về mặt kỹ thuật trong suốt tiến trình phần mềm.

(4): chiến lược kiểm chứng.

(5): kiểm soát những tài liệu của phần mềm.

(6): thủ tục để đảm bảo những tiêu chuẩn phần mềm đươc đáp ứng.

(7): cơ chế đo lường và báo cáo.

Một số khái niệm về đảm bảo chất lượng phần mềm:

4.2.1 Chất lượng (Quality)

Theo "American Heritage Dictionary", chất lượng là 1 đặc tính hay thuộc tính của cái gì đó. Là thuộc tính của 1 item, chất lượng tham khảo đến những đặc tính có tính đo lường được- những thứ có thể so sánh được với những tiêu chuẩn đã biết như độ dài, màu sắc, tính uốn dẻo,…

Khi chúng ta xem xét 1 item trên cơ sở những đặc tính có tính đo lường được, 2 loại chất lượng có thể gặp phải là: chất lượng của thiết kế (quality of design) và chất lượng của sự phù hợp (quality of conformance).

Chất lượng của thiết kế: tham khảo đến những đặc tính mà người thiết kế đặc tả cho item. Lớp vật liệu, sức chịu đựng và chi tiết hiệu suất, tất cả góp phần tạo nên chất lượng thiết kế.

Chất lượng của sự phù hợp: là mức độ theo đúng đặc tả thiết kế trong quá trình sản xuất. Mức độ của sự phù hợp càng lớn thì mức độ chất lượng của sự phù hợp cũng cao hơn.

Trong quá trình phát triển phần mềm, chất lượng của thiết kế gồm yêu cầu, đặc tả, và thiết kế của hệ thống. Chất lượng của sự phù hợp là vấn đề được đưa ra tập trung vào sự thực thi. Nếu sự thực thi theo đúng thiết kế và hệ thống kết quả đáp ứng đúng yêu cầu và mục tiêu hiệu suất, chất lượng sự phù hợp sẽ cao.

4.2.2 Kiểm soát chất lượng (Quality Control QC)

QC bao gồm 1 loạt những sự kiểm tra, phê duyệt (inspection, review, test) trong suốt tiến trình phần mềm để đảm bảo mỗi phần công việc đáp ứng được yêu cầu tại phần đó.

QC có thể được thực hiện hoàn toàn tự động, bằng tay hay kết hợp cả hai.

4.2.3 Đảm bảo chất lượng (Quality Assurance QA)

QA bao gồm việc kiểm tra (audit) và báo cáo những chức năng của việc quản lý. Mục tiêu của QA là cung cấp sự quản lý với những dữ liệu cần thiết để được thông tin về chất lương sản phẩm, từ đó đạt được cái nhìn sâu sắc và sự tự tin rằng sản phẩm đáp ứng được mục tiêu của nó.

4.2.4 Chi phí của chất lượng (Cost of Quality)

Chi phí của chất lượng bao gồm tất cả các chi phí phát sinh trong việc theo đuổi chất lượng hoặc trong việc thực hiện các hoạt động liên quan đến chất lượng.

Chi phí chất lượng có thể được chia thành:

Chi phí phòng ngừa (Prevention cost): gồm

- quality planning
- formal technical reviews
- test equipment
- training

Chi phí định giá (Appraisal cost): gồm những hoạt động để đạt được cái nhìn sâu sắc trong điều kiện sản phẩm "lần đầu tiên" trong mỗi tiến trình.

Chi phí lỗi (Failure cost): sẽ không xuất hiện nếu không có defect nào xuất hiện trước khi giao sản phẩm cho khách hàng. Chi phí lỗi có thể được chia ra thành chi phí lỗi nội bộ (internal failure cost) và chi phí lỗi bên ngoài (external failure cost).

Chi phí lỗi nội bộ liên quan đến những defect được tìm thấy trước khi giao sản phẩm. Gồm: rework, repair, failure mode analysis.

Chi phí lỗi bên ngoài liên quan đến những defect được tìm thấy sau khi giao sản phẩm. Gồm: complaint resolution, product return and replacement, help line support, warranty work.

4.3 Kiểm chứng phần mềm

4.3.1 Đặc điểm của kiểm tra phần mềm

Xác minh và thẩm định một hệ phần mềm là một quá trình liên tục xuyên suốt mọi giai đoạn của quá trình phần mềm. Xác minh và thẩm định là từ chung cho các quá trình kiểm tra để đảm bảo rằng phần mềm thỏa mãn các yêu cầu của chúng và các yêu cầu đó thỏa mãn các nhu cầu của người sắm phần mềm.

Xác minh và thẩm định là một quá trình kéo dài suốt vòng đời. Nó bắt đầu khi duyệt xét yêu cầu. Xác minh và thẩm định có hai mục tiêu:

i) Phát hiện các khuyết tật trong hệ thống.

ii) Đánh giá xem hệ thống liệu có dùng được hay không?

Sự khác nhau giữa xác minh và thẩm định là:

i) Thẩm định: Xem xét cái được xây dựng có *là sản phẩm đúng* không? Tức là kiểm tra xem chương trình có được như mong đợi của người dùng hay không.

ii) Xác minh: Xem xét cái được xây dựng có *đúng là sản phẩm* không? Như thế, xác minh là kiểm tra chương trình có phù hợp với đặc tả hay không.

Như trên, kiểm tra là một quá trình của tìm kiếm lỗi. Một kiểm tra tốt có khả năng cao tìm kiếm các lỗi chưa được phát hiện. Một kiểm tra thành công là kiểm tra tìm ra các lỗi mới, một kiểm tra tồi là kiểm tra mà không tìm được lỗi.

Có hai kiểu lỗi trong ứng dụng và các kiểm tra tốt sẽ xác định cả hai loại đó. Cụ thể:

- Không làm những điều cần phải làm: lỗi bỏ sót thường xuất hiện đối với ứng dụng mới được phát triển.
- Làm những điều không cần làm: lỗi của lệnh đã cư trú trước trong các ứng dụng bảo trì.

Các kiểm tra có mặt tại các mức khác nhau và được tiến hành bởi các cá nhân khác nhau trong quá trình phát triển ứng dụng. Các kiểm tra được tiến hành bởi đội ngũ dự án và kiểm tra được tiến hành bởi các cơ quan bên ngoài để chấp nhận ứng dụng.

Các kiểm tra của đội dự án được gọi là kiểm tra phát triển (Development test). Kiểm tra phát triển bao gồm kiểm tra đơn vị, kiểm tra hệ thống con, kiểm tra tích hợp và các kiểm tra hệ thống.

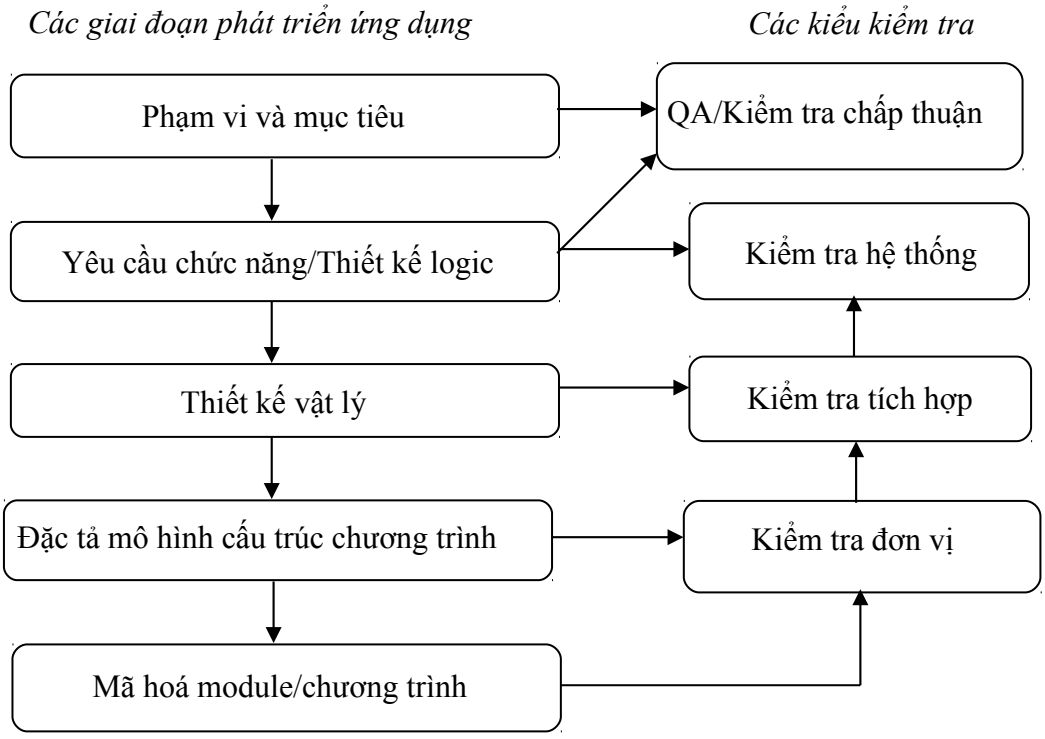
- Kiểm tra đơn vị (Unit test) được tiến hành cho mỗi đơn vị mã nhỏ nhất.
- Kiểm tra tích hợp (Subsystem integration test) kiểm tra mặt logic và xử lý cho phù hợp của các khối, kiểm tra việc truyền tin giữa chúng.
- Kiểm tra hệ thống (System test) đánh giá các đặc tả chức năng có được đáp ứng, các thao tác giao diện người có giống thiết kế không, và các công việc của ứng dụng trong môi trường thao tác mong muốn, đối với các ràng buộc.

Các kiểm tra bởi các cơ quan bên ngoài được gọi là đảm bảo chất lượng (Quality assurance-QA) và kiểm tra chấp nhận (Acceptance test). Người ngoài

có thể là người sử dụng hoặc đại diện người dùng, nó tạo một mục đích, đánh giá khách quan về ứng dụng và cơ quan bên ngoài được coi là có mục đích hơn các thành viên nhóm.

Kiểm tra đảm bảo chất lượng tương tự các kiểm tra hệ thống về mặt mục đích và tiến hành, nhưng nó khác là họ nằm ngoài sự điều khiển của dự án trưởng. Các báo cáo kiểm tra đảm bảo chất lượng được gửi thường xuyên tới nhóm phát triển và quản lý dự án. Các kiểm tra viên đảm bảo chất lượng lập kế hoạch cho chiến lược của mình và tiến hành kiểm tra để đảm bảo các ứng dụng thực hiện tất cả các chức năng cần thiết. Kiểm tra đảm bảo chất lượng là kiểm tra cuối cùng được làm trước khi ứng dụng được đưa sản xuất đại trà.

Quan hệ giữa các mức kiểm tra và các giai đoạn trong vòng đời của chương trình được trình bày như sau:



Mỗi mức kiểm tra đòi hỏi xác định chiến lược kiểm tra. Chiến lược kiểm tra hộp trắng hoặc kiểm tra hộp đen.

- Dữ liệu vào được tạo theo thiết kế để sinh ra các dữ liệu ra khác nhau mà không chú ý tới các chức năng logic thực hiện thế nào. Các kết

quả được dự đoán và so sánh với các kết quả thực tế để đánh giá mức độ thành công.

- Chiến lược White-box mở hộp và nhìn vào các logic đặc tả của ứng dụng để kiểm tra nó làm thế nào. Kiểm tra sử dụng các đặc tả logic để tạo ra các xử lý khác nhau và dự đoán các kết quả ra. Các kết quả trung gian và đầu ra cuối cùng có thể dự đoán và định lượng nhờ kiểm tra white-box.

Chiến lược kiểm tra top–down hay bottom–up: xác định các kiểm tra và phát triển mã sẽ được tiến hành như thế nào:

- Kiểm tra top–down cho rằng mã điều khiển tới hạn và các chức năng sẽ được phát triển và kiểm tra đầu tiên. Tiếp theo là các chức năng thứ cấp và các hàm hỗ trợ. Lý thuyết là càng có nhiều module tới hạn được kiểm tra, thì càng ổn định về chương trình.
- Kiểm tra bottom–up cho rằng càng ít thay đổi trong các module khả năng sinh lỗi càng ít. Toàn bộ các module được mã và đơn vị được kiểm tra. Sau đó kiểm tra được tiến hành ở mức độ tích hợp.

Các chiến lược kiểm tra không loại trừ lẫn nhau, chúng có thể được sử dụng đơn lẻ hoặc đồng thời. Lý tưởng là kiểm tra cho một ứng dụng bao gồm nhiều chiến lược để phát hiện được hết các lỗi.

Sau khi chiến lược đã được xác định, nó được áp dụng để tạo các trường hợp kiểm tra cụ thể. Test case: là các giao dịch riêng biệt hoặc các bản ghi dữ liệu tạo các logic được kiểm tra. Cho mọi trường hợp kiểm tra tất cả các kết quả của xử lý được dự đoán. Đối với ứng dụng on-line và off-line tài liệu hoá, test scripts các hội thoại tạo ra giữa người dùng và ứng dụng và các thay đổi từ hội thoại. Test plan: tư liệu hoá các chiến lược, kiểu, các trường hợp và mô tả cho kiểm tra vài khối ứng dụng. Tất cả các kế hoạch tạo thành kế hoạch tổng thể cho ứng dụng.

Kiểm tra được lặp lại cho đến khi không còn lỗi, hoặc đạt được mức độ chấp nhận.

- Bước đầu tiên của xử lý kiểm tra, đầu vào kiểm tra, cấu hình và mã ứng dụng được đòi hỏi để tạo các kiểm tra.
- Bước thứ hai là so sánh các kết quả kiểm tra với kết quả dự tính và định lượng sự khác biệt.
- Bước tiếp theo là loại trừ các lỗi, hoặc gỡ rối mã. Khi việc mã hoá lại hoàn thành, kiểm tra lại được tiến hành.

Quá trình tiến hành kiểm tra bắt đầu từ khi thiết kế. Cộng tác viên thực hiện việc kiểm tra nên có khả năng của phân tích viên và lập trình viên để có thể hiểu các đòi hỏi của ứng dụng và kiểu cách tiến hành kiểm tra. Chương trình càng lớn và phức tạp thì càng cần người có kinh nghiệm, đôi khi cần có một đội ngũ kiểm tra. Đội ngũ kiểm tra sử dụng yêu cầu chức năng từ giai đoạn phân tích và đặc tả chương trình, đặc tả thiết kế từ giai đoạn thiết kế như là đầu vào để phát triển chiến lược kiểm tra hệ thống. Khi chiến lược đã được hoàn tất, các buổi thảo luận được tiến hành để kiểm tra lại chiến lược và thông báo tới toàn thể đội. Các nhiệm vụ tại các mức sẽ được ấn định. Thời gian được ước lượng. Đội ngũ kiểm tra làm việc độc lập và song song với đội ngũ lập trình. Họ làm việc với quản trị CSDL để phát triển cơ sở dữ liệu mà có thể hỗ trợ tất cả các mức kiểm tra. Đối với kiểm tra đơn vị, đội kiểm tra kiểm tra kết quả, các chấp nhận các module và chương trình cho kiểm tra tích hợp (*integration test*). Đội ngũ kiểm tra tiến hành và định lượng các kiểm tra tích hợp và kiểm tra hệ thống.

4.3.2 Chiến lược kiểm tra phần mềm

Như đã nói ở trên, có hai kiểu chiến lược kiểm tra. Kiểu thứ nhất liên quan logic được kiểm tra thế nào trong ứng dụng. Chiến lược kiểm tra logic có thể là black-box hoặc white-box. Chiến lược kiểm tra black-box cho ràng module của chương trình hoặc hệ thống liên quan tới đầu vào và đầu ra. Các chi tiết logic chi tiết được che dấu và không cần phân tích. Chiến lược black-box có tính hướng dữ liệu. White-box hướng tới việc cho rằng logic đặc trưng là quan trọng và cần phải kiểm tra. White-box đánh giá một vài hoặc tất cả mặt

logic để kiểm tra được tính đúng đắn của chức năng. White-box hướng về logic - giải thuật.

Kiểu thứ hai liên quan tới việc kiểm tra được tiến hành thế nào, không quan tâm chiến lược kiểm tra logic. Nó là top-down hoặc bottom-up. Top-down coi chương trình chính là quan trọng nhất nên cần phải phát triển và kiểm tra trước và tiếp tục trong quá trình phát triển. Bottom-up cho rằng các module và chương trình riêng sẽ được phát triển hoàn toàn như standalone. Vậy chúng được kiểm tra riêng rẽ sau đó được kết hợp thành kiểm tra tổ hợp.

4.3.2.1 Kiểm tra Black-box

Kiểm tra hộp đen, black-box, coi xử lý kết quả như là minh chứng bởi dữ liệu. Khái niệm kiểm tra là black-box không quan tâm logic. Khuynh hướng này hiệu quả đối với các modul chức năng đơn và các hệ thống cấp cao. Ba phương pháp chính là:

- *Phân hoạch cân bằng:* Mục đích của phương pháp này là tối thiểu các trường hợp kiểm tra cho trước, các mục dữ liệu vào được chia thành các nhóm dữ liệu có vai trò cân bằng nhau, mỗi nhóm đại diện cho một tập dữ liệu. Nguyên tắc là bằng cách kiểm tra triệt để một mục của mỗi tập hợp, chúng ta có thể chấp nhận tất cả các mục tương đương khác của tập hợp đó cũng sẽ được kiểm tra một cách kỹ càng.
- *Phân tích cực biên:* Phân tích giá trị cực biên là một dạng nghiêm ngặt của phân hoạch cân bằng có sử dụng các giá trị biên hơn là bất cứ giá trị nào trong tập cân bằng. Ví dụ: miền giá trị của tháng là 1 – 12 và ngoài là 0 và 13. Thì 4 kiểm tra ứng với bốn trường hợp sẽ được kiểm tra phân tích cực biên thường xuyên được dùng tại các mức modul để xác định các mục dữ liệu đặc trưng cho testing.
- *Đoán lỗi:* Dựa trên cơ sở trực giác và kinh nghiệm, các chuyên gia có thể dễ dàng kiểm tra các điều kiện lỗi bằng cách đoán cái nào dễ xảy ra nhất. Ví dụ, chia 0, nếu modul có phép chia, nên dùng phép chia 0. Vì dựa trên trực giác, nên phép thử này khó tìm hết các lỗi.

Một nhược điểm của phân hoạch cân bằng và phân tích cực biên là tổ hợp của các miền hợp không được xác định. Để bổ sung, người ta dùng sơ đồ nguyên nhân – kết quả (Cause – Effect Graphing). Sơ đồ nguyên nhân – kết quả chỉ ra các đầu ra và thông tin di chuyển như là hệ quả và các đầu vào gây ra hệ quả trên. Các ký hiệu graphic xác định tương tác, lựa chọn, logic và các điều kiện tương đương. Một vòng đại diện một dãy các thao tác không có điểm quyết định hoặc điều khiển. Mỗi dòng biểu diễn một lớp cân bằng của dữ liệu và các điều kiện sử dụng nó. Mỗi lần graph được thực hiện thì ít nhất một giá trị có nghĩa và một không có nghĩa cho mỗi tập được thử.

4.3.2.2 White-box testing

Có ba loại kiểm tra hộp trắng là kiểm tra logic -logic test, chứng minh toán học -mathematical proof và Cleanroom testing.

1. Logic test

Logic test có thể được chi tiết đến mức lệnh. Trong khi thực hiện của mọi lệnh là mục đích đáng khen ngợi, nó có thể không kiểm tra tất cả các điều kiện thuộc chương trình. Ví dụ, ít nhất 2 kiểm tra cho một kiểm tra 2 điều kiện (1 đúng và 1 sai). Cố gắng kiểm tra tất cả các điều kiện lẽ dĩ nhiên là không thể thực hiện được về thực tiễn. Ví dụ trong 1 module có 10 thao tác qua 4 vòng lặp thì có 5,5 triệu trường hợp kiểm tra. Do đó phải có phương pháp lựa chọn.

Logic kiểm tra nhìn vào mỗi quyết định trong module và sản sinh các dữ liệu để tạo tất cả các kết quả ra có thể. Có một vấn đề với logic test tại mức này là chúng không test tình trạng của module đối với đặc tả. Nếu kiểm tra được phát triển dựa trên đặc tả, mà đặc tả được dịch khác nhau bởi người lập trình thì kiểm tra sẽ không đúng. Giải pháp là đòi hỏi đặc tả chương trình đủ chi tiết và logic. Điều này có thể phù hợp với ngôn ngữ thế hệ 1 và 2.

Các kiểm tra nhiều điều kiện tạo mỗi kết quả ra của tiêu chuẩn đa quyết định và nhiều điểm vào module. Các kiểm tra đòi hỏi việc phân tích xác định được bên quyết định đa tiêu chuẩn. Nếu các biên được xác định không chính xác, thì kiểm tra không hiệu quả. Khi được thiết kế phù hợp, test logic đa điều kiện có thể tối thiểu hoá các trường hợp kiểm tra để kiểm tra nhiều nhất các điều kiện. Sự sử dụng kỹ thuật này đòi hỏi luyện tập và kỹ năng.

2. Chứng minh bằng toán học

Một phương pháp theo cách tiếp cận giảm thiếu sót về 0 là áp dụng suy diễn toán học cho đòi hỏi logic, chứng minh tính đúng đắn của chương trình. Phương pháp này đòi hỏi đặc tả ngôn ngữ dạng hình thức. Kỹ thuật chứng minh tính đúng đắn của chương trình được đề cập ở 6.4.

3. Cleanroom testing

Cleanroom testing là mở rộng của phương pháp chứng minh bằng toán học. Lý thuyết của kỹ thuật này là bằng cách ngăn chặn các lỗi tại các đầu vào của quá trình xử lý, giá thành sẽ giảm, độ tin cậy tăng lên và tiệm cận tới không có lỗi.

Phương pháp này được phát triển tại IBM đầu những năm 1980 bởi Mills, Dyer, Linger. Các đặc tả hình thức được dùng và việc kiểm tra thủ công được tiến hành bởi các đội kiểm tra. Các chương trình khó đọc sẽ được viết lại và kiểm tra hoàn toàn trên giấy.

Cleanroom testing là kỹ thuật kiểm tra toán học hình thức và hội ý (walkthrough). Mục đích là phân tích mọi module thành các chức năng và liên kết. Các phép kiểm tra chức năng dùng kỹ thuật toán học, các kiểm tra liên kết sử dụng thuyết tập hợp.

Sau khi thực hiện kiểm tra (verification), đội kiểm tra độc lập sẽ dịch và thực hiện mã. Dữ liệu kiểm tra được dịch bởi các phân tích đặc tả chức năng và được thực hiện thể hiện các tỷ lệ xác suất của dữ liệu được giả định cho hệ

thống thực. Thêm vào các dữ liệu chuẩn thì các loại lỗi nghiêm trọng được tạo để kiểm tra ứng dụng.

Bất lợi của phương pháp này bắt buộc là đòi hỏi kỹ năng cao về: toán, thống kê, logic và ngôn ngữ đặc tả hình thức.

4.3.2.3 Kiểm tra top-down

Phương pháp kiểm tra top-down cần một mã ngoài, được hiểu như là một bộ khung để gắn các chức năng gốc, các modul, và các phần khác của ứng dụng. Bộ khung này thường bắt đầu với ngôn ngữ điều khiển công việc và logic chính của ứng dụng. Logic chính được kiểm tra và lập khung theo các hệ thống phân rã. Đầu tiên chỉ có các thủ tục thiết yếu và các logic điều khiển được kiểm tra.

Khi các module thiết yếu nhất đã được kiểm tra và chạy tốt thì mã của các modul ít quan trọng hơn sẽ được gắn vào khung và tiếp tục kiểm tra. Về lý thuyết thì, top-down sẽ tìm thấy các lỗi thiết kế sớm hơn trong kiểm tra thao tác (testing process) hơn các khuynh hướng khác. Phương pháp này ít hiệu quả trong việc cải thiện chất lượng của các phần mềm chuyển giao vì bản chất tương tác của kiểm tra.

Kiểm tra top-down dễ dàng hỗ trợ giao diện người dùng và thiết kế màn hình. Trong các ứng dụng tương tác, thường là bộ điều khiển màn hình được kiểm tra đầu tiên. Người dùng có thể kiểm tra sự điều khiển thông qua màn hình với các phát triển tạo mẫu.

4.3.2.4 Kiểm tra bottom-up

Nguyên tắc của bottom-up là kiểm tra mọi thay đổi tại module có thể ảnh hưởng tới chức năng của nó. Trong kiểm tra bottom-up, toàn bộ khối là đơn vị để đánh giá. Tất cả các module được mã hoá và kiểm tra riêng rẽ.

Các trường hợp kiểm tra: các trường hợp kiểm tra là dữ liệu vào được tạo để thể hiện từng khối và toàn bộ hệ thống thoả mãn tất cả các yêu cầu thiết kế.

Mỗi trường hợp kiểm tra nên được phát triển để kiểm tra nghiệm các đòi hỏi thiết kế đặc trưng, thiết kế chức năng, hoặc mã đã được thoả mãn. Hơn nữa các trường hợp kiểm tra cần dự đoán các output.

Mỗi đơn nguyên của ứng dụng (Ví dụ: module, subroutine, program, utility,...) phải được kiểm tra với ít nhất hai trường hợp kiểm tra: một trường hợp chạy tốt và một trường hợp không chạy. Trong trường hợp chạy sai hệ phải đưa được thông báo, quay lại (rollback) được trạng thái ban đầu của giao dịch.

Để chắc chắn rằng các trường hợp được toàn diện nhất, người ta thường dùng ma trận. Chúng được dùng cho:

- Kiểm tra đơn khối để định nhánh logic, điều kiện logic, các phần dữ liệu hoặc biên dữ liệu để kiểm tra trên cơ sở đặc tả chương trình.
- Kiểm tra tổ hợp để định ra yêu cầu về dữ liệu và quan hệ trong số các tương tác.
- Kiểm tra hệ thống để xác định yêu cầu về người dùng và hệ thống từ các yêu cầu chức năng và các yêu cầu chấp nhận.

Phối hợp các kiểm tra trong một chiến lược: mục đích của các nhà kiểm tra là tìm ra sự cân bằng của các chiến lược cho phép họ chứng minh được ứng dụng chạy tốt mà tối thiểu hoá chi phí máy tính và nhân lực. Sử dụng duy nhất một chiến lược là rất nguy hiểm. Không có cái nào là duy nhất đúng. Nếu chỉ có white-box thì tài nguyên nhân lực và máy là rất tốn kém, nếu chỉ có black-box các vấn đề logic đặc trưng có thể chưa được khám phá.

4.4 Bảo trì phần mềm

4.4.1 Hoạt động bảo trì phần mềm và phân loại

Bảo trì phần mềm là phức tạp và chúng ta có thể chia hoạt động bảo trì ra làm bốn hoạt động như sau:

4.4.1.1 Bảo trì hiệu chỉnh

Công việc bảo trì đầu tiên cần phải thực hiện là do việc kiểm tra chương trình không thể tránh được mội lỗi ẩn chứa bên trong một hệ phần mềm lớn.

Trong khi sử dụng bất kỹ một chương trình lớn nào, các lỗi sẽ được báo về lại cho người phát triển.

Bảo trì hiệu chỉnh chính là quá trình phân tích và hiệu chỉnh một hay nhiều lỗi của chương trình.

4.4.1.2 Bảo trì tiếp hợp

Hoạt động thứ hai diễn ra bởi sự thay đổi thường xuyên môi trường. Những thế hệ phần cứng mới dường như được công bố theo chu trình 24 tháng một lần. Những hệ điều hành mới hay phiên bản mới của các hệ cũ đều đặn xuất hiện; thiết bị ngoại vi và các thành phần hệ thống khác liên tục được nâng cấp và thay đổi. Thời gian hữu dụng của một phần mềm ứng dụng mặt khác lại dễ dàng vượt qua thời hạn mười năm, lâu hơn môi trường hệ thống đã phát triển nó đầu tiên.

Bảo trì tiếp hợp là hoạt động sửa đổi phần mềm để thích ứng được với những thay đổi của môi trường.

4.4.1.3 Bảo trì hoàn thiện

Hoạt động thứ ba diễn ra khi một phần mềm đã được hoàn tất thành công. Hoạt động này chiếm hầu hết các công sức tiêu tốn cho việc bảo trì phần mềm. Lúc sử dụng, các yêu cầu về những khả năng mới, các thay đổi những chức năng đã có, và các mở rộng tổng quát được người dùng gửi đến.

Để thỏa mãn những yêu cầu phát triển của người sử dụng, ta tiến hành bảo trì hoàn thiện.

4.4.1.4 Bảo trì phòng ngừa

Bảo trì phòng ngừa là hoạt động bảo trì diễn ra khi phần mềm được thay đổi để cải thiện tính năng bảo trì hay độ tin cậy trong tương lai hoặc để cung cấp một nền tảng tốt hơn cho những mở rộng sau này.

Bảo trì phòng ngừa, hoạt động này vẫn còn nhiều xa lạ trong thế giới phần mềm hiện nay.

Các thuật ngữ dùng để mô tả ba hoạt động bảo trì đầu tiên là do Swanson đề xướng. Thuật ngữ thứ tư thường được dùng trong việc bảo trì phần cứng hay các hệ thống vật lý khác. Tuy nhiên cần chú ý rằng những điểm tương tự giữa bảo trì phần mềm và bảo trì phần cứng có thể gây nhầm lẫn.

Phần mềm khác với phần cứng, không thể tận dụng được, vì vậy hoạt động bảo trì phần cứng chủ yếu - thay thế các bộ phận bị hỏng hóc hay gãy vỡ - không được kể đến.

Trong thực tế của hoạt động bảo trì, các nhiệm vụ được làm như một phần của bảo trì tiếp hợp và bảo trì hoàn thiện cũng giống như các nhiệm vụ cần làm trong giai đoạn phát triển của một chu trình phần mềm. Để tiếp hợp hay hoàn thiện, chúng ta đều phải xác định yêu cầu, thiết kế lại, tạo mã và kiểm tra phần mềm có được. Thông thường các nhiệm vụ đó đã được gọi là bảo trì rồi.

4.4.2 Đặc điểm của bảo trì phần mềm

Bảo trì phần mềm cho tới gần đây vẫn còn là một giai đoạn bị coi nhẹ của chu trình phần mềm. Kiến thức về bảo trì có được rất ít khi so sánh với các giai đoạn hoạch định và phát triển. Có rất ít các số liệu nghiên cứu và chế tạo tập trung vào đề tài này, vẫ rất ít các phương pháp kỹ thuật được đưa ra. Để hiểu được điểm bản chất của bảo trì, chúng ta sẽ xem xét các vấn đề từ ba góc độ khác nhau:

- Các hoạt động cần thiết để hoàn thành giai đoạn bảo trì và tính toàn vẹn của một cách tiếp cận theo công nghệ phần mềm đối với hiệu quả của những hoạt động đó, hay sự thiếu hụt nó.
- Chi phí kèm theo giai đoạn bảo trì.
- Các vấn đề thường gặp phải khi tiến hành bảo trì phần mềm.

4.4.2.1Bảo trì có cấu trúc đối với bảo trì không cấu trúc

Nếu thành phần có được duy nhất của một cấu hình phần mềm là mã nguồn, hoạt động bảo trì bắt đầu với việc đánh giá chi tiết mã nguồn thường là khá phức tạp với những tài liệu nghèo nàn bên trong. Những đặc điểm tế nhị như cấu trúc phần mềm, các cấu trúc dữ liệu toàn cục, giao diên hệ thống,hoạt động và các ràng buộc thiết kế thường rất khó sáng tỏ và hay bị hiểu lầm. Các thay đổi lặt vặt thường xuyên làm cho các mã rất khó đánh giá. Các kiểm tra hồi quy (lặp lại các kiểm tra trước kia để đảm bảo rằng những thay đổi không

tạo ra lỗi trong phần mềm đã hoạt động) là không thể thực hiện được bởi không hề có các bản lưu về các kiểm tra đó. Chúng ta đang tiến hành phép bảo trì không cấu trúc và đang phải trả giá (khi lãng phí công sức và gây tâm trạng thất vọng). Sự trả giá này luôn đi kèm với các phần mềm đã không được phát triển theo những phương pháp đúng đắn.

Nếu có một cấu hình phần mềm hoàn thiện, nhiệm vụ bảo trì bắt đầu bằng việc đánh giá các tài liệu thiết kế. Sau đó là xác định các đặc điểm thuộc cấu trúc quan trọng, các đặc điểm hoạt động và giao diện. Tính toàn vẹn của những sửa đổi và hiệu chỉnh cần thiết sẽ được đánh giá và một kế hoạch sửa đổi sẽ được thiết lập. Thiết kế được thay đổi (sử dụng những kỹ thuật phù hợp với những điều đã bàn luận ở ácc chương trước) rồi nhận xét đánh giá. Mã nguồn được phát triển, sau đó tiến hành các kiểm tra hồi quy sử dụng thông tin chứa trong phần "đặc tả kiểm tra" và rồi phần mềm lại được phát hành.

Các mô tả trên đây là phép bảo trì cấu trúc và được tiến hành như là kết quả của những ứng dụng trước đây trong khoa học về công nghệ phần mềm. Mặc dù sự có mặt của một cấu hình phần mềm không đảm bảo được các vấn đề bảo trì nảy sinh, nhưng khối lượng công việc đã được giảm bớt và chất lượng chung của những thay đổi và hiệu chỉnh đã được cải thiện.

4.4.2.2 Giá thành bảo trì

Theo thống kê, giá thành cho việc bảo trì các phần mềm tăng lên một cách đều đặn trong suốt 20 năm qua. Trong những năm 1970, bảo trì chiếm đến 35 đến 40 phần trăm kinh phí phần mềm dành cho tổ chức hệ thống thông tin.Tỷ lệ này đã nhảy tới con số 60 vào giữa những năm 1980. Và nhiều công ty đã chi 80% kinh phí cho việc bảo trì vào giữa những năm 1990.

Chi phí cho việc bảo trì là rõ ràng nhất. Tuy nhiên những chi phí khác khó thấy hơn có thể gây ra mối quan tâm lớn hơn:

- Một chi phí khó xác định của việc bảo trì phần mềm là các cơ hội phát triển bị bỏ qua vì các tài nguyên có sẵn đều dành cho nhiệm vụ bảo trì.

- Sự không hài lòng của người dùng khi các yêu cầu có vẻ hợp lý cho việc sửa chữa hay sửa đổi không được chú ý một cách hợp lý.
- Việc suy giảm chất lượng nói chung do lỗi tạo ra bởi sự thay đổi trong các phần mềm được bảo trì.
- Một yêu cầu bất chợt làm ngắt quãng quá trình phát triển của một nhân viên buộc anh ta tiến hành công việc bảo trì.

Chi phí cuối cùng cho việc bảo trì là sự giảm sút kinh khủng về năng suất lao động (được đo theo số dòng lệnh -LOC- của một người trong một tháng hay số tiền chi phí cho dòng lệnh). Sự giảm sút này xuất hiện khi tiến hành bảo trì đối với các phần mềm cũ. Người ta đã ghi nhận sự giảm sút năng suất lao động theo tỷ lệ 40:1, có nghĩa là chi phí cho việc phát triển trị giá $25.00 trên một dòng lệnh sẽ có thể trị giá tới $1000.00 cho việc bảo trì mỗi dòng lệnh.

Công sức cho việc bảo trì có thể được phân chia thành các thao tác làm việc: phân tích, ước lượng, thay đổi thiết kế, và sửa chữa chương trình nguồn và các thao tác lặp lại: việc cố gắng hiểu chương trình nguồn làm gì, cố gắng sáng tỏ cấu trúc dữ liệu, các thuộc tính giao diện, giới hạn của việc thực hiện,... Biểu thức dưới đây cung cấp một mô hình cho công việc bảo trì:

$$M = p + K^* exp(c-d),$$

với M = toàn bộ các công việc cho việc bảo trì;

p = công việc làm (như miêu tả ở trên);

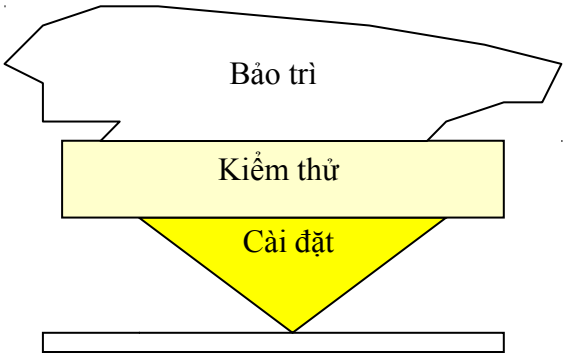
K = hằng số kinh nghiệm;

c = đánh giá mức độ phức tạp được tính cho việc thiếu thiết kế về cấu trúc và dữ liệu;

d = đánh giá mức độ hiểu biết về phần mềm.

Mô hình trên đây cho thấy công việc và giá thành có thể tăng lên theo cấp số mũ nếu một phương pháp phát triển phần mềm kém cỏi được sử dụng -tức là thiếu sót của công nghệ phần mềm, và nếu một người hay một nhóm dùng các phương pháp không có giá trị để bảo trì phần mềm. Chi phí cho bảo

trì khi phần mềm được phát triển không đúng phương pháp được minh hoạ ở hình sau:



Chi phí của việc phát triển phần mềm không có phương pháp

4.4.2.3 Một số vấn đề khác

Hầu hết các vấn đề liên quan tới việc bảo trì phần mềm đều liên quan tới các sai lệch trong cách xây dựng và phát triển phần mềm. Sự thiếu sót trong việc điều khiển và tổ chức trong hai giai đoạn đầu tiên của một chu trình phần mềm gần như luôn luôn tạo ra các vấn đề giai đoạn cuối.

Nhiều vấn đề kinh điển có thể liên quan tới việc bảo trì phần mềm được trình bày dưới đây:

- Rất khó hoặc không thể theo dõi sự tiến hóa của phần mềm qua các phiên bản. Các thay đổi không được tư liệu hóa.
- Khó theo dõi được các quá trình xử lý được tạo bởi các phần mềm.
- Thường xuyên gặp rất nhiều khó khăn trong việc tìm hiểu chương trình của người khác viết. Những khó khăn này tăng lên khi số thành phần các cấu hình của phần mềm giảm đi. Nếu chỉ có các chương trình nguồn không có tài liệu hướng dẫn thì không nên tìm hiểu phần mềm đó.
- Những người viết phần mềm thường không có mặt để giải thích. Chúng ta không thể trông cậy vào những giải thích cá nhân của các nhà phát triển phần mềm khi việc bảo trì được yêu cầu.

- Các tài liệu chính xác không có hay thiếu trầm trọng. Phải thừa nhận rằng phải có tài liệu về phần mềm là bước đầu tiên, nhưng tài liệu phải hiểu được và phù hợp với chương trình lại là chuyện khác.
- Phần lớn các phần mềm không thiết kế để thay đổi, trừ phi sử dụng phương pháp thiết kế dùng các khái niệm về phân tách chương trình thành các module độc lập. Việc thay đổi phần mềm sẽ rất khó khăn và dẫn đến xu hướng sai.
- Việc bảo trì phần mềm không được coi là một công việc dễ dàng mà công việc bảo trì phần mềm luôn liên quan tới các sai lệch ở mức độ cao.

4.4.3 Công việc bảo trì phần mềm

4.4.3.1 Khả năng bảo trì

Khả năng bảo trì của phần mềm có thể coi như các khả năng hiểu, hiệu chỉnh, tiếp hợp hoặc có thể thêm vào khả năng phát triển. Khả năng bảo trì là chìa khóa để dẫn đến các phương pháp thiết kế xây dựng phần mềm.

A. Yếu tố kiểm soát

Khả năng bảo trì cơ bản bao gồm nhiều yếu tố. Sự thiếu cẩn trọng trong việc thiết kế, viết chương trình nguồn, kiểm tra có ảnh hưởng tiêu cực đến việc bảo trì có kết quả một phần mềm. Cấu hình yếu kém cũng có các tác động tương tự, thậm chí cả khi từng bước của kỹ thuật xây dựng phần mềm được xem xét một cách cẩn thận. Thêm vào đó nhiều yếu tố khác liên quan tới phương pháp phát triển phần mềm, như:

- Chất lượng hiệu quả của đội ngũ phần mềm.
- Cấu trúc của hệ thống dễ hiểu.
- Dễ dàng kiểm soát hệ thống.
- Dùng các ngôn ngữ lập trình chuẩn.
- Dùng các hệ điều hành chuẩn.
- Dùng các cấu trúc chuẩn tài liệu.
- Dùng được các tài liệu kiểm tra.

- Phương tiện gỡ rối đi kèm.
- Dùng được các máy tính tốt để thực hiện việc bảo trì.

Các yếu tố được đưa ra ở trên đã phản ánh những đặc điểm về phần cứng cũng như chương trình nguồn được dùng trong việc phát triển phần mềm. Những yếu tố khác chỉ ra sự cần thiết để có được phương pháp chuẩn, chương trình nguồn chuẩn. Có thể, yếu tố quan trọng nhất tác động tới khả năng bảo trì là kế hoạch cho khả năng bảo trì. Nếu coi phần mềm như là một hệ thống các thành phần sẽ phải chịu những thay đổi không tránh được, thì cơ hội tạo những phần mềm có khả năng bảo trì sẽ tăng thực sự.

B. Đánh giá định lượng

Khả năng bảo trì, như chất lượng hay độ tin cậy là hết sức khó xác định.

Tuy nhiên chúng ta có thể đánh giá khả năng bảo trì gián tiếp bằng việc xem xét các thuộc tính của các công việc bảo trì có thể đánh giá được:

- Thời gian nhận biết vấn đề.
- Thời gian trễ do các công việc hành chính.
- Thời gian lựa chọn công cụ bảo trì.
- Thời gian phân tích vấn đề.
- Thời gian xác định sự thay đổi.
- Thời gian hiệu chỉnh (hay sửa đổi) thực sự.
- Thời gian chạy thử cục bộ.
- Thời gian chạy thử tổng thể.
- Thời gian tổng kết bảo trì.
- Tổng thời gian của công việc bảo trì.

Mỗi đánh giá trên thực tế là các dữ liệu đó có thể cung cấp cho người quản lý cùng với chỉ số về hiệu quả của công việc.

4.4.3.2 Các công việc bảo trì

Những nhiệm vụ liên quan tới việc bảo trì gồm: các tổ chức bảo trì được thiết lập; các thủ tục ghi nhận và đánh giá được miêu tả; và một loạt thứ tự

chuẩn của các bước cho mỗi yêu cầu bảo trì phải được định nghĩa. Thêm vào đó, một thủ tục lưu trữ các hồ sơ cho các hoạt động bảo trì được thiết lập và bản tổng kết những tiêu chuẩn đánh giá được vạch rõ.

A. Cơ cấu bảo trì

Mặc dù những tổ chức bảo trì chuẩn không cần được thiết lập, nhưng sự ủy thác trách nhiệm rất là cần thiết kể cả cho các tổ chức phát triển phần mềm nhỏ. Những yêu cầu bảo trì được chuyển qua cho người kiểm soát công việc bảo trì và từ đây chuyển tiếp yêu cầu tới người quản lý hệ thống để đánh giá. người quản lý hệ thống là thành viên của nhóm nhân viên kỹ thuật. Những nhân viên này có trách nhiệm về một phần nhỏ của chương trình sản phẩm. Khi môth yêu cầu được đánh giá, người được ủy quyền quản lý việc thay đổi phải quyết định hhành động nào được thực hiện tiếp.

Cơ cấu được miêu tả ở trên phục vụ cho việc thiết lập phạm vi trách nhiệm đối với công việc bảo trì. Người kiểm soát và người ủy quyền quản lý việc thay đổi có thể là một người hay là một nhóm quản lý và chuyên gia kỹ thuật cao cấp.

B. Báo cáo

Tất cả các yêu cầu về việc bảo trì phần mềm cần được trình bày theo một tiêu chuẩn. Người phát triển phần mềm thường cung cấp một đơn yêu cầu bảo trì còn được gọi là báo cáo các lỗi phần mềm. Báo cáo này được người sử dụng điền vào khi yêu cầu công việc bảo trì. Nếu xuất hiện một lỗi, bản mô tả đầy đủ tình huống dẫn đến lỗi bao gồm dữ liệu, đoạn chương trình và các yêu cầu khác phải được điền đầy đủ vào bản báo cáo. Nếu yêu cầu bảo trì là bảo trì tiếp hợp hay bảo trì hoàn thiện thì một yêu cầu chi tiết sẽ được thảo ra. Đơn yêu cầu bảo trì sẽ được người kiểm soát bảo trì và người quản lý hệ thống xem xét như phần trước đã nêu.

Đơn yêu cầu bảo trì được thiết lập từ bên ngoài và được coi như một cơ sở để đề ra kế hoạch của công việc bảo trì. Ngoài ra trong nội bộ của cơ quan phần mềm một báo cáo thay đổi phần mềm cũng được tạo ra. Nó chỉ ra:các

công sức đòi hỏi để thỏa mãn một đơn yêu cầu bảo trì, trạng thái ban đầu của yêu cầu sửa đổi, mức ưu tiên của yêu cầu, các dữ liệu cần cho việc sửa đổi,...

c. Lưu giữ các hồ sơ

Thường chúng ta không có đầy đủ các hồ sơ cho tất cả các giai đoạn trong chu kỳ sống của một phần mềm. Thêm nữa không có các hồ sơ về việc bảo trì phần mềm. Do đó chúng ta thường khó có thể tiến hành các công việc bảo trì có hiệu quả, không có khả năng xác định tính chất của các chương trình và khó xác định được giá bảo trì phần mềm.

Các thông tin cần phải lưu giữ trong hồ sơ bảo trì thường:

- Dấu hiệu nhận biết chương trình.
- Số lượng các câu lệnh trong chương trình nguồn.
- Số lượng các lệnh mã máy.
- Ngôn ngữ lập trình được sử dụng.
- Ngày cài đặt chương trình.
- Số các chương trình chạy từ khi cài đặt.
- Số các lỗi xử lý xảy ra.
- Mức và dấu hiệu thay đổi chương trình.
- Số các câu lệnh được thêm vào chương trình nguồn khi chương trình thay đổi.
- Số các câu lệnh được xóa khỏi chương trình nguồn khi chương trình thay đổi.
- Số giờ mỗi người sử dụng cho mỗi lần sửa đổi.
- Ngày thay đổi chương trình.
- Dấu hiệu của kỹ sư phần mềm.
- Dấu hiệu của đơn yêu cầu bảo trì.
- Kiểu bảo trì.
- Ngày bắt đầu và kết thúc bảo trì.
- Tổng số giờ của mỗi người dùng cho việc bảo trì.

D. Xác định giá bảo trì

Việc xác định giá trị bảo trì thường phức tạp do thiếu thông tin. Nếu tiến hành việc lưu giữ các hồ sơ có thể tiến hành một số cách đánh giá về việc thực hiện bảo trì. Theo các chuyên gia thì đánh giá về việc thực hiện bảo trì dựa vào:

- Số lượng trung bình các lỗi xử lý cho một lần chạy chương trình.
- Tổng số giờ của mỗi người dùng cho mỗi loại bảo trì.
- Số lượng trung bình các thay đổi theo chương trình, theo ngôn ngữ lập trình, theo kiểu bảo trì.
- Số giờ trung bình của mỗi người cho một dòng lệnh được thêm vào và xóa đi.
- Số giờ trung bình của mỗi người cho một ngôn ngữ lập trình.
- Thời gian trung bình cho việc bảo trì một đơn yêu cầu bảo trì.
- Tỷ lệ phần trăm của mỗi kiểu bảo trì.

TÀI LIỆU THAM KHẢO

1. Software Engineering – A practitioner's approach fifth edition : Roger S. Pressman, Ph.D
2. Software Engineering eighth edition : Ian Sommerville
3. Bài giảng Kỹ thuật phần mềm: Nguyễn Việt Hà, trường Đại học Công nghệ - Đại học Quốc gia Hà Nội.
4. Bài giảng Công nghệ phần mềm: ThS Trương Vạn Trình, trường Đại học Phạm Văn Đồng.

MỤC LỤC