BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC ĐÔNG Á

BÀI GIẢNG MÔN HỌC

MÔN: CÔNG NGHỆ PHẦN MỀM

Đà Nẵng 2011

Lời nói đầu

Nhập môn Công Nghệ Phần Mềm là môn học nhằm giúp cho sinh viên có kiến thức cơ bản nhất trong lĩnh vực công nghệ phần mềm. Qua môn học này sinh viên có cái nhìn khái quát về qui trình phát triển phần mềm, hiểu biết và thực hiện các giai đoạn trong qui trình trên một phần mềm cụ thể dựa trên những phương pháp, kỹ thuật trong quá trình thu thập yêu cầu, phân tích, thiết kế và cài đặt, viết tài liệu đã được minh họa cụ thể trong giáo trình. Mục tiêu giáo trình là sinh viên có thể hiểu được những yêu cầu công việc cần phải làm ở mỗi giai đoạn của qui trình, để có thể đảm trách công việc ở một trong các giai đoạn làm phần mềm trong những nhóm dự án.

Mục lục

Chương 1:Tổng quan về Công nghệ Phần mềm

1.1 Mở đầu

Ngày nay, sự phát triển phần mềm ngày càng thực sự khó kiểm soát được; các dự án phần mềm thường kéo dài và vượt quá chi phí cho phép. Những nhà lập trình chuyên nghiệp phải cố gắng hoàn thành các dự án phần mềm một cách có chất lượng, đúng hạn trong chi phí cho phép.

Mục đích của chương này là đưa ra những nhận định cơ bản và tạo nên một bức tranh cơ sở về những phương pháp tiếp cận khác nhau của công việc trong công nghệ phần mềm. Các vấn đề cần làm rõ, chi tiết thêm sẽ được trình bày ở các chương tiếp sau của giáo trình.

1.2 Định nghĩa và đặc tính của sản phẩm phần mềm

1.2.1 Định nghĩa phần mềm

Chương trình máy tính là một trình tự các chỉ thị (lệnh) để hướng dẫn máy tính làm việc nhằm hoàn thành một công việc nào đó do con người yêu cầu.

Phần mềm là một hệ thống các chương trình có thể thực hiện trên máy tính nhằm hỗ trợ các nhà chuyên môn trong từng lĩnh vực chuyên ngành thực hiện tốt nhất các thao tác nghiệp vụ của mình. Nhiệm vụ chính yếu của phần mềm là cho phép các nhà chuyên môn thực hiện các công việc của họ trên máy tính dễ dàng và nhanh chóng hơn so với khi thực hiện cùng công việc đó trong thế giới thực.

Hoạt động của mọi phần mềm là sự mô phỏng lại các hoạt động của thế giới thực trong một góc độ thu hẹp nào đó trên máy tính. Quá trình sử dụng một phần mềm chính là quá trình người dùng thực hiện các công việc trên máy tính để hoàn tất một công việc tương đương trong thế giới thực.

1.2.2 Phân loại và đặc tính của sản phẩm phần mềm.

1.2.2.1 Phân loại sản phẩm phần mềm

Generic Product**:** là sản phẩm đóng gói và bán rộng rãi trên thị trường.

Bespoke Product**:** là sản phẩm được phát triển theo yêu cầu đặc thù của từng khách hàng.

Ngoài ra có thể phân chia phần mềm theo miền ứng dụng như sau:

a. Phần mềm hệ thống

- Là một tập hợp các chương trình được viết để phục vụ cho các chương trình khác
- Xử lý các cấu trúc thông tin phức tạp nhưng xác định (trình biên dịch, trình soạn thảo, tiện ích quản lý tệp)
- Đặc trưng bởi tương tác chủ yếu với phần cứng máy tính
- Phục vụ nhiều người dùng
- Cấu trúc dữ liệu phức tạp và nhiều giao diện ngoài

b. Phần mềm thời gian thực

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

- Thành phần thu thập dữ liệu để thu và định dạng thông tin từ môi trường ngoài
- Thành phần phân tích để biến đổi thông tin theo yêu cầu của ứng dụng
- Thành phần kiểm soát hoặc đưa ra đáp ứng môi trường ngoài
- Thành phần điều phối để điều hòa các thành phần khác sao cho có thể duy trì việc đáp ứng thời gian thực

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

c. Phần mềm nghiệp vụ

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

d. Phần mềm khoa học và công nghệ

- Được đặc trưng bởi các thuật toán (tính toán trên ma trận số, mô phỏng...).
- Thường đòi hỏi phần cứng có năng lực tính toán cao.

e. Phần mềm nhúng

- Nằm trong bộ nhớ chỉ đọc và được dùng để điều khiển các sản phẩm và hệ thống cho người dùng và thị trường công nghiệp.
- Có các đặc trưng của phần mềm thời gian thực và phần mềm hệ thống.

f. Phần mềm máy tính cá nhân

- Bùng nổ từ khi xuất hiện máy tính cá nhân, giải quyết các bài toán nghiệp vụ nhỏ như xử lý văn bản, trang tính, đồ họa, quản trị CSDL nhỏ...
- Yếu tố giao diện người-máy rất được chú trọng.

g. Phần mềm trí tuệ nhân tạo

- Dùng các thuật toán phi số để giải quyết các vấn đề phức tạp mà tính toán hay phân tích trực tiếp không quản lý nổi
- Các ứng dụng chính là: hệ chuyên gia (hệ cơ sở tri thức), nhận dạng (hình ảnh và tiếng nói), chứng minh định lý và chơi trò chơi, mô phỏng.

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

phần mềm nghiệp vụ.

1.2.2.2 **Các đặc tính quan trọng của sản phẩm phần mềm**

Phần mềm thông thường được định nghĩa bao gồm:

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

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

- ✓ *Có thể bảo trì được (Maintainability):* phần mềm tuổi thọ dài phải được viết và được lập tư liệu sao cho việc thay đổi có thể tiến hành được mà không quá tốn kém. Đây được coi là đặc tính chủ chốt nhất của một phần mềm tốt. Để có thể bảo trì được, phần mềm phải có một thiết kế tốt có tính modun hóa cao, được viết bằng ngôn ngữ bậc cao và được lập tài liệu (tài liệu phân tích, thiết kế, chú thích mã nguồn, hướng dẫn người dùng...) đầy đủ.
- ✓ *Đáng tin cậy (Reliablity):* phần mềm phải thực hiện được điều mà người tiêu dùng mong mỏi và không thất bại nhiều hơn những điều đã được đặc tả. Điều này có nghĩa là phần mềm phải thỏa mãn được nhu cầu của người dùng. Để đạt được yếu tố đáng tin cậy, trước tiên người phát triển cần phải hiểu một cách đúng đắn yêu cầu của người dùng và sau đó cần thỏa mãn được các yêu cầu này bằng các thiết kế và cài đặt tốt.
- ✓ *Có hiệu quả (Efficiency):* phần mềm khi hoạt động phải không lãng phí tài nguyên hệ thống như bộ nhớ, bộ xử lý. Nếu phần mềm chạy quá chậm hay đòi hỏi quá nhiều bộ nhớ... thì dù có được cài đặt rất nhiều chức năng cũng sẽ không được đưa vào sử dụng. Tuy nhiên, ngoại trừ các phần mềm nhúng hay thời gian thực đặc biệt, người ta thường không cực đại hóa mức độ hiệu quả vì rằng việc đó có thể phải dùng đếm các kỹ thuật đặc thù và cài đặt bằng ngôn ngữ máy khiến cho chi phí tăng cao và phần mềm rất khó thay đổi (tính bảo trì kém).
- ✓ *Dễ sử dụng (Usability):* giao diện người sử dụng phải phù hợp với khả năng và kiến thức của người dùng, có các tài liệu hướng dẫn và các tiện ích trợ giúp. Đối tượng chính của các phần mềm nghiệp vụ thường là người không am hiểu về máy tính, họ sẽ xa lánh các phần mềm khó học, khó sử dụng.

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

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

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

1.3 Định nghĩa và các đặc trưng của Công nghệ phần mềm

1.3.1 Định nghĩa Công nghệ phần mềm

Công Nghệ Phần Mềm là sự thiết lập và sử dụng các nguyên tắc khoa học nhằm mục đích tạo ra các phần mềm một cách kinh tế mà các phần mềm đó hoạt động hiệu quả và tin cậy trên các máy tính.

Công nghệ phần mềm là một quy trình có hệ thống được sử dụng trong quá trình phân tích, thiết kế, hiện thực, kiểm tra và bảo trì để bảo đảm các sản phẩm phần mềm được sản xuất và hoạt động: hiệu quả, tin cậy, hữu dụng, nâng cấp dễ dàng (modificable), khả chuyển (portable), khả kiểm tra (testable), cộng tác được với các hệ thống khác (interoperable) và vận hành đúng (correct).

1.3.1 Các đặc trưng của Công nghệ phần mềm

- Efficiency: Phần mềm được sản xuất trong thời gian và điều kiện vừa phải. Phần mềm vận hành đúng mức độ yêu cầu về công việc và thời gian.
- Reliablity: Phần mềm vận hành ổn định và tương tác được với các hệ thống ứng dụng
- Usability: Phần mềm có thể dùng được bởi người sử dụng và với môi trường mà người sử dụng đang có. Chú ý tới giao diện, điều kiện hệ thống,…
- Modifiability: Phần mềm có thể được thay đổi dể dàng, nhanh chóng khi yêu cầu của người sử dụng thay đổi.
- Portability: Phần mềm có thể chuyển đổi dễ dàng sang các hệ thống khác mà không cần phải điều chỉnh lớn. Chỉ cần recompile nếu cần thiết là tốt nhất.
- Testability: Phần mềm có thể được kiểm tra dễ dàng. Tốt nhất là được modul hóa.
- Reusability: Phần mềm hay một phần có thể được tái sử dụng cho các ứng dụng khác. Các modul có thiết kế tốt, độc lập và giao tiến đơn giản, cả về tình tương thích công nghệ phát triển
- Maintainability: thiết kế của phần mềm có thể được hiểu dễ dàng cũng như chuyển giao thuận tiện cho người khác trong quá trình điều chỉnh, nâng cấp hay thay đổi theo yêu cầu.
- Interoperability: Phần mềm vận hành ổn định và đúng như mong đợi. Trên hệ thống nhiều người dùng (multi users) phần mềm vẫn hoạt động được với các vận hành khác của hệ thống.

- Correctness: Phần mềm phải tính toán đúng và tạo ra kết quả đúng và đúng với mục tiêu ứng dụng của người dùng.

Một định nghĩa khác của Công nghệ phần mềm

CNPM là các quy trình đúng kỷ luật và có định lượng được áp dụng cho sự phát triển, thực thi và bảo trì các hệ thống thiên về phần mềm

Tập trung vào quy trình, sự đo lường, sản phẩm, tính đúng thời gian và chất lượng

1.3.2 Nội dung công việc của một kỹ sư phần mềm

Kỹ sư phần mềm (software engineer): là một người biết cách áp dụng rộng rãi những kiến thức về cách phát triển ứng dụng vào việc tổ chức phát triển một cách có hệ thống các ứng dụng. Công việc của người kỹ sư phần mềm là: đánh giá, lựa chọn, sử dụng những cách tiếp cận có tính hệ thống, chuyên biệt, rõ ràng trong việc phát triển, đưa vào ứng dụng, bảo trì, và thay thế phần mềm.

Do đặc điểm nghề nghiệp, người kỹ sư phần mềm phải có những kỹ năng cơ bản như:

- Định danh, đánh giá, cài đặt, lựa chọn một phương pháp luận thích hợp và các công cụ CASE.
- Biết cách sử dụng các mẫu phần mềm (prototyping).
- Biết cách lựa chọn ngôn ngữ, phần cứng, phần mềm.
- Quản lý cấu hình, lập sơ đồ và kiểm soát việc phát triển của các tiến trình.
- Lựa chọn ngôn ngữ máy tính và phát triển chương trình máy tính.
- Đánh giá và quyết định khi nào loại bỏ và nâng cấp các ứng dụng.

Mục tiêu của kỹ sư phần mềm là sản xuất ra các sản phẩm có chất lượng cao và phù hợp với các quy trình phát triển chuẩn mực:

- Việc phát triển (development): được bắt đầu từ khi quyết định phát triển sản phẩm phần mềm và kết thúc khi sản phẩm phần mềm được chuyển giao cho người sử dụng.
- Việc sử dụng (operations): là việc xử lý, vận hành hằng ngày sản phẩm phần mềm
- Việc bảo trì (maintenance): thực hiện những thay đổi mang tính logic đối với hệ thống và chương trình để chữa những lỗi cố định, cung cấp những thay đổi về công việc, hoặc làm cho phần mềm được hiệu quả hơn.
- Việc loại bỏ (retirement): thường là việc thay thế các ứng dụng hiện thời bởi các ứng dụng mới.

1.3.3 Lịch sử ngành công nghệ phần mềm

Công nghệ phần mềm có một lịch sử khá sớm. Các công cụ được dùng cũng như các ứng dụng được viết đã tham gia vào kỹ nghệ phần mềm theo thời gian.

– Thập niên 1940: Các chương trình cho máy tính được viết bằng tay.

- Thập niên 1950: Các công cụ đầu tiên xuất hiện như là phần mềm biên dịch Macro Assembler và phần mềm thông dịch đã được tạo ra và sử dụng rộng rãi để nâng cao năng suất và chất lượng. Các trình dịch được tối ưu hoá lần đầu tiên ra đời.
- Thập niên 1960: Các công cụ của thế hệ thứ hai như các trình dịch tối ưu hoá và công việc kiểm tra mẫu đã được dùng để nâng cao sản phẩm và chất lượng. Khái niệm công nghệ phần mềm đã được bàn thảo rộng rãi.
- Thập niên 1970: Các công cụ phần mềm, chẳng hạn trong UNIX các vùng chứa mã, lệnh make, v.v. được kết hợp với nhau. Số lượng doanh nghiệp nhỏ về phần mềm và số lượng máy tính cỡ nhỏ tăng nhanh.
- Thập niên 1980: các PC và máy trạm ra đời. Cùng lúc có sự xuất hiện của mô hình dự toán khả năng. Lượng phần mềm tiêu thụ tăng mạnh.
- Thập niên 1990: Phương pháp lập trình hướng đối tượng ra đời. Các quá trình nhanh như là lập trình cực hạn được chấp nhận rộng rãi. Trong thập niên này, WWW và các thiết bị máy tính cầm tay phổ biến rộng rãi.
- Hiện nay: Các phần mềm biên dịch và quản lý như là .NET, PHP và Java làm cho việc viết phần mềm trở nên dễ dàng hơn nhiều.

1.4 Mô hình phát triển phần mềm

1.4.1 Các công đoạn trong phát triển phần mềm

Các công đoạn chính tổng quát bao gồm 4 giai đoạn

- Giai đoạn đặc tả: xác định các tính năng và điều kiện hoạt động của hệ thống. (thu thập yêu cầu và phân tích)
- Giai đoạn phát triển: Thiết kế phần mềm (software design), viết code (code generation)
- Giai đoạn kiểm tra: kiểm tra phần mềm (software testing), kiểm tra tính hợp lý của phần mềm.
- Giai đoạn bảo trì: Sửa lỗi (correction), thay đổi môi trường thực thi (adaptation), tăng cường (enhancement)

Đặc tả

Đây là bước hình thành bài toán hoặc đề tài. Ở bước này quản trị dự án hoặc phân tích viên hệ thống phải biết được vai trò của phần mềm cần phát triển trong hệ thống, đồng thời phải ước lượng công việc, lập lịch biểu và phân công công việc.

Bên cạnh đó chúng ta phải biết người đặt hàng muốn gì. Các yêu cầu cần phải được thu thập đầy đủ và được phân tích theo chiều ngang (rộng) và chiều dọc (sâu). Công cụ sử dụng chủ yếu ở giai đoạn này là các lược đồ, sơ đồ phản ánh rõ các thành phần của hệ thống và mối liên quan giữa chúng với nhau.

Phát triển

Dựa vào các nội dung đã xác định được, nhóm phát triển phần mềm dùng ngôn ngữ đặc tả hình thức (dựa trên các kiến trúc toán học) hoặc phi hình thức (tựa

ngôn ngữ tự nhiên) hoặc kết hợp cả hai để mô tả những yếu tố sau đây của chương trình:

- Giá trị nhập, giá trị xuất.
- Các phép biến đổi
- Các yêu cầu cần đạt được ở mỗi điểm của chương trình.

Phần đặc tả chỉ quan tâm chủ yếu đến giá trị vào, ra chứ không quan tâm đến cấu trúc và nội dung các thao tác cần thực hiện.

Sau bước thiết kế là bước triển khai các đặc tả chương trình thành một sản phẩm phần mềm dựa trên một ngôn ngữ lập trình cụ thể. Trong giai đoạn này các lập trình viên sẽ tiến hành cài đặt các thao tác cần thiết để thực hiện đúng các yêu cầu đã được đặc tả.

Kiểm tra

Sau giai đoạn phát triển là chúng ta cần phải chứng minh tính đúng đắn của chương trình sau khi đã tiến hành cài đặt. Tuy nhiên thông thường ở bước này chúng ta coi các chương trình như những hộp đen. Vấn đề đặt ra là xây dựng một cách có chủ đích các tập dữ liệu nhập khác nhau để giao cho chương trình thực hiện rồi dựa vào kết quả thu được để đánh giá chương trình. Công việc như trên được gọi là kiểm thử chương trình.

Công việc kiểm thử nhằm vào các mục tiêu sau:

- Kiểm tra để phát hiện lỗi của chương trình. Lưu ý rằng kiểm thử không đảm bảo tuyệt đối tính đúng đắn của chương trình do bản chất quy nạp không hoàn toàn của cách làm.
- Kiểm tra tính ổn định, hiệu quả cũng như khả năng tối đa của chương trình.

Tùy theo mục đích mà người ta thiết kế các tập dữ liệu thử sao cho có thể phủ hết các trường hợp cần quan tâm.

Bảo trì

Công việc quản lý việc triển khai và sử dụng phần mềm cũng là một vấn đề cần được quan tâm trong qui trình phát triển phần mềm. Trong quá trình xây dựng phần mềm, toàn bộ các kết quả phân tích, thiết kế, cài đặt và hồ sơ liên quan cần phải được lưu trữ và quản lý cẩn thận nhằm đảm bảo cho công việc được tiến hành một cách hiệu quả nhất và phục vụ cho công việc bảo trì phần mềm về sau.

Như vậy công việc quản lý không chỉ dừng lại trong quá trình xây dựng phần mềm mà trái lại còn phải được tiến hành liên tục trong suốt quá trình sống của nó.

1.4.2 Các mô hình phát triển phần mềm

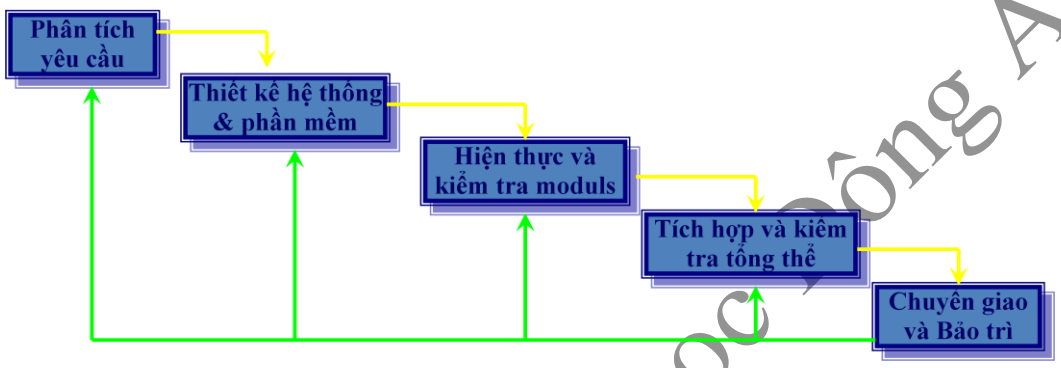
Tùy theo quy mô và công nghệ phát triển, có các mô hình sản xuất khác nhau.

- Mô hình tuần tự tuyến tính- waterfall
- Mô hình Prototyping - Evolutionary Development
- Mô hình xoắn ốc – Boehm’s Spiral Model
- Mô hình RAD – Rapid Application Development

Mỗi mô hình phù hợp với trình độ phát triển, quy mô sản phẩm và yêu cầu ràng buộc cụ thể về thời gian và tính chất của hệ thống.

1.4.3 Mô hình tuần tự tuyến tính WaterFall – Sequency model

Mô hình tuần tự tuyến tính (mô hình thác nước) là một trong những mô hình đầu tiên và phổ biến được áp dụng trong quá trình phát triển phần mềm. Mô hình này chia quá trình phát triển phần mềm thành những giai đoạn tuần tự nối tiếp nhau. Mỗi giai đoạn sẽ có một mục đích nhất định. Kết quả cuả giai đoạn trước sẽ là thông tin đầu vào cho giai đoạn tiếp theo sau. Tùy theo qui mô của phần mềm cần phát triển mà mô hình thác nước sẽ có những biến thể khác nhau.



Hình 1.1: Mô hình tuần tự tuyến tính WaterFall – Sequency model

Mô hình thác nước giúp chúng ta có thể dễ dàng phân chia quá trình xây dựng phần mềm thành những giai đoạn hoàn toàn độc lập nhau. Tuy nhiên, các dự án lớn hiếm khi tuân theo dòng chảy tuần tự của mô hình vì thường phải lặp lại các bước để nâng cao chất lượng. Hơn nữa, khách hàng hiếm khi tuyên bố hết các yêu cầu trong giai đoạn phân tích.

Mô hình này cũng có một hạn chế là chúng ta rất khó thực hiện các thay đổi một khi đã thực hiện xong một giai đoạn nào đó. Điều này làm cho việc xây dựng phần mềm rất khó thay đổi các yêu cầu theo ý muốn của khách hàng. Do đó, phương pháp này chỉ thích hợp cho những trường hợp mà chúng ta đã hiểu rất rõ các yêu cầu của khách hàng.

1.4.4 Mô hình bản mẫu Prototype Model

Tương tự như mô hình thác nước với bổ sung vào các giai đoạn thực hiện phần mềm mẫu ngay khi xác định yêu cầu nhằm mục tiêu phát hiện nhanh các sai sót về yêu cầu. Các giai đoạn trong mô hình bản mẫu phần mềm có thể tiến hành lặp đi lặp lại chứ không nhất thiết phải theo trình tự nhất định.

Ngay sau khi giai đoạn xác định yêu cầu, nhà phát triển phần mềm đưa ra ngay một bản thiết kế sơ bộ và tiến hành cài đặt bản mẫu đầu tiên và chuyển cho người sử dụng. Bản mẫu này chỉ nhằm để miêu tả cách thức phần mềm hoạt động cũng như cách người sử dụng tương tác với hệ thống.

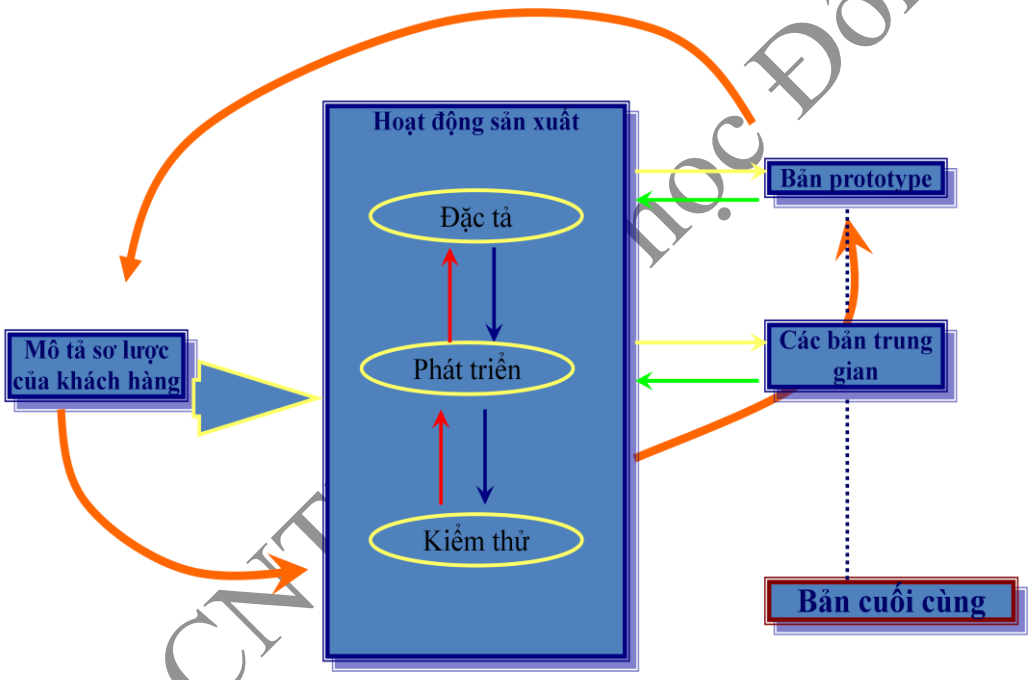
Người sử dụng sau khi xem xét sẽ phản hồi thông tin cần thiết lại cho nhà phát triển. Nếu người sử dụng đồng ý với bản mẫu đã đưa thì người phát triển sẽ

tiến hành cài đặt thực sự. Ngược lại cả hai phải quay lại giai đoạn xác định yêu cầu. Công việc này được lặp lại liên tục cho đến khi người sử dụng đồng ý với các bản mẫu do nhà phát triển đưa ra.

Như vậy đây là một hướng tiếp cận tốt khi các yêu cầu chưa rõ ràng và khó đánh giá được tính hiệu quả của các thuật toán. Tuy nhiên, mô hình này cũng có nhược điểm là tính cấu trúc không cao do đó khách hàng dễ mất tin tưởng.

Các ứng dụng của mô hình bản mẫu:

- Dùng cho các hệ thống nhỏ. Các chi phí khi thay đổi hệ thống là không quá lớn khia cần phải thay đổi sau khi thực hiện prototype.
- Cần sự cấp bách về thời gian triển khai ngắn. Hệ thống cần được đưa vào ứng dụng từng phần trong khoảng thời gian nhất định.
- Trong trường hợp những hệ thống mà việc đặc tả các yêu cầu là rất khó và không rõ ràng ngay từ đầu.



Hình 1.2: Mô hình bản mẫu Prototype Model

Những nhược điểm của mô hình bản mẫu:

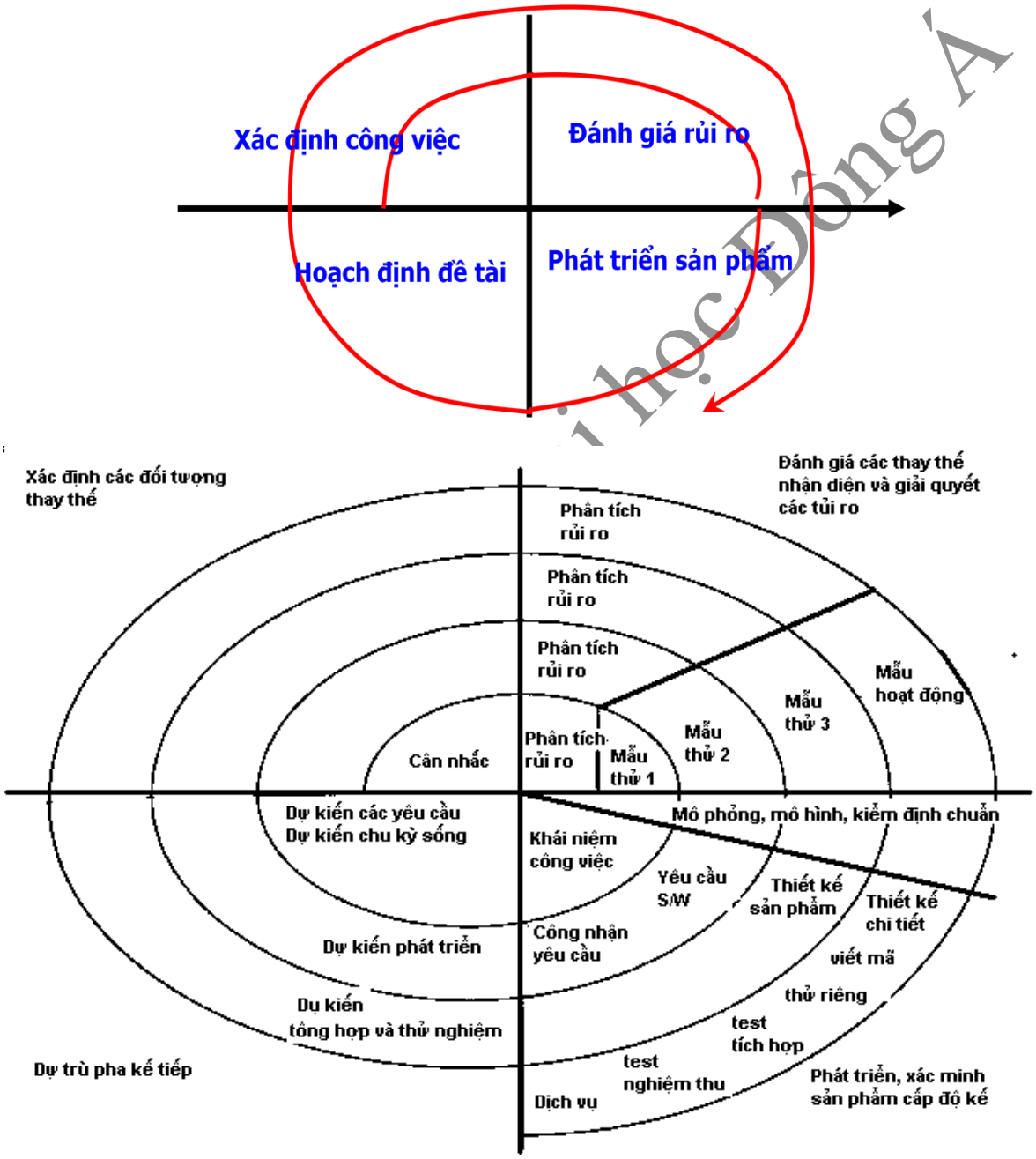
- Các bản mẫu (Prototype) có thể bị "throw-away" gây lãng phí cho dự án.
- Các tiến trình không được phân định rõ ràng
- Hệ thống thông thường có cấu trúc lỏng lẻo
- Cần có những kỹ năng đăc biệt trong quản lý và phát triển
- Khách hàng hối thúc nhà phát triển hoàn thành sản phẩm một khi thấy được các bản mẫu (prototype) đầu tiên

1.4.5 Mô hình xoắn ốc Boehm's Spiral Model

Mô hình này chính là sự kết hợp của mô hình bản mẫu thiết kế (Prototyping) và mô hình tuần tự tuyến tính được lặp lại nhiều lần. Ở lần lặp tiếp theo hệ thống sẽ được tìm hiểu và xây dựng hoàn thiện hơn ở lần lặp trước đó.

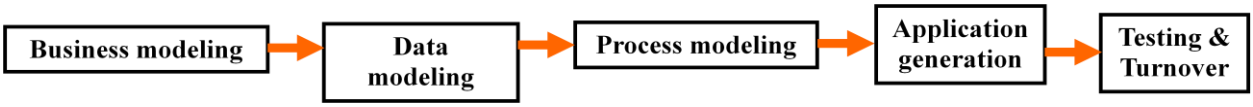
Ở mỗi lần lặp các yêu cầu của người sử dụng sẽ được hiểu ngày càng rõ ràng hơn và các bản mẫu phần mềm cũng ngày một hoàn thiện hơn. Ngoài ra ở cuối mỗi lần lặp sẽ có thêm công đoạn phân tích mức độ rủi ro để quyết định xem có nên đi tiếp theo hướng này nữa hay không.

Mô hình này phù hợp với các hệ thống phần mềm lớn do có khả năng kiểm soát rủi ro ở từng bước tiến hóa. Tuy nhiên vẫn chưa được sử dụng rộng rãi như mô hình thác nước hoặc bản mẫu do đòi hỏi năng lực quản lý, năng lực phân tích rủi ro cao.



Hình 1.3: Mô hình xoắn ốc

1.4.6 Mô hình RAD



Hình 1.4: Mô hình RAD

Là quy trình phát triển phần mềm gia tăng, tăng dần từng bước (Incrimental software development) với mỗi chu trình phát triển rất ngắn (60-90 ngày).

Xây dựng dựa trên hướng thành phần (Component-based construction) với khả năng tái sử dụng (reuse).

Gồm một số nhóm (teams), mỗi nhóm làm 1 RAD theo các pha: Mô hình nghiệp vụ, Mô hình dữ liệu, Mô hình xử lý, Tạo ứng dụng, Kiểm thử và đánh giá (Business, Data, Process, Appl. Generation, Test).

Business modeling: Luồng thông tin được mô hình hóa để trả lời các câu hỏi:

- Thông tin nào điều khiển xử lý nghiệp vụ ?
- Thông tin gì được sinh ra?
- Ai sinh ra nó ?
- Thông tin đi đến đâu ?
- Ai xử lý chúng ?

Data modeling: Các đối tượng dữ liệu cần để hỗ trợ nghiệp vụ (business). Định nghĩa các thuộc tính của từng đối tượng và xác lập quan hệ giữa các đối tượng.

Process modeling: Các đối tượng dữ liệu được chuyển sang luồng thông tin thực hiện chức năng nghiệp vụ. Tạo mô tả xử lý để cập nhật (thêm, sửa, xóa, khôi phục) từng đối tượng dữ liệu.

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

Testing and Turnover: Kiểm thử các thành phần mới và kiểm chứng mọi giao diện (các thành phần cũ đã được kiểm thử và dùng lại)

Các hạn chế của mô hình RAD:

- Cần nguồn nhân lực dồi dào để tạo các nhóm cho các chức năng chính
- Yêu cầu hai bên giao kèo trong thời gian ngắn phải có phần mềm hoàn chỉnh, thiếu trách nhiệm của một bên dễ làm dự án đổ vỡ
- RAD không phải tốt cho mọi ứng dụng, nhất là với ứng dụng không thể môđun hóa hoặc đòi hỏi tính năng cao
- Mạo hiểm kỹ thuật cao thì không nên dùng RAD

1.5 Các tiêu chuẩn dùng trong ngành Công nghiệp phần mềm

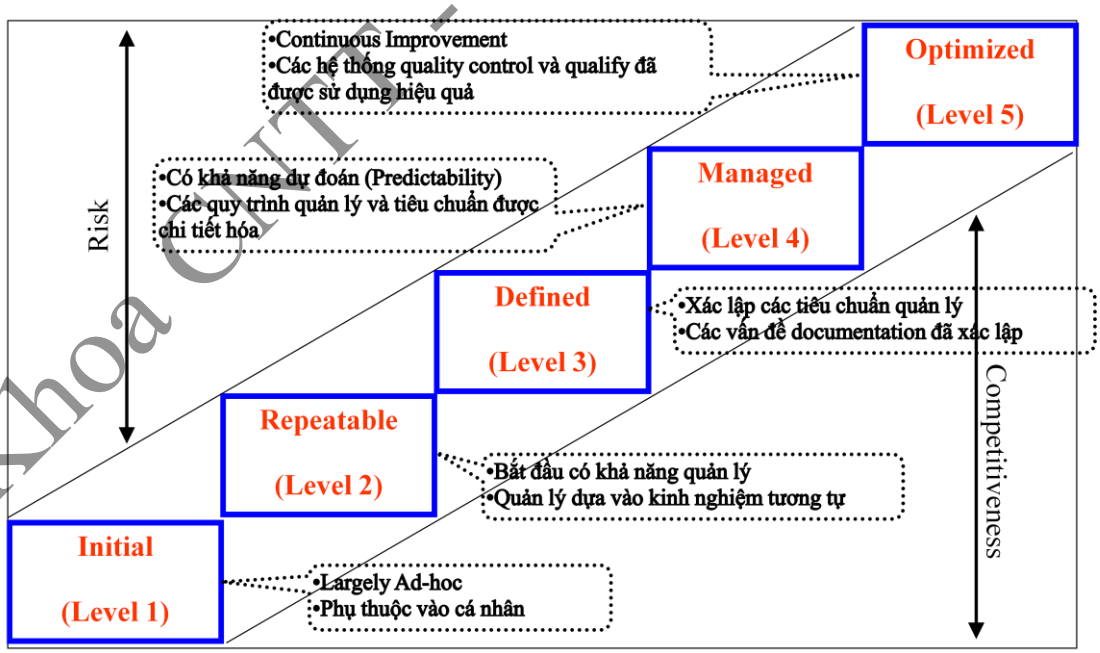
- The capability Maturity Model (CMM) của Software Engineering Institue (SEI) - Đại học Carnegie Mellon.

- Chú trọng đến tính hệ thống và khả năng quản trị của các công ty phần mềm hơn là một quy trình (process) cụ thể.

- The process Improvement Paradigm (PIP) của Software Engineering Laboratory (SEL) – NASA's Goddard Space Flight Center
  - Tương tự như CMM, chú trọng đến tính hệ thống và những hướng dẫn để tăng cường tính năng của các quá trình quản lý.
- Các chuẩn khác của Department of Defense
  - MIL – STD 2167A ; MIL-STD 1574A ; MIL-STD 882C
- The electronic Industries Association (EIA) chuẩn SEB-6-A
- The European ESPRIT project
- International Standards Organisation - ISO 9001
- United Kingdom MOD 0055

Mô hình CMM (Capability Maturity Model)

Mô hình CMM cho phần mềm được đưa ra bởi Viện Kỹ nghệ Phần mềm (Software Engineering Institute - SEI) của Đại học tổng hợp Carnegie Mellon, đã được hỗ trợ quốc tế rộng rãi và là một chương trình tài trợ không hoàn lại, công khai cho bất kỳ công ty nào muốn tiếp nhận nó. Mô hình CMM mô tả các nguyên tắc và các thực tiễn nằm bên trong tính "thành thục" quá trình phần mềm và chủ ý giúp đỡ các công ty phần mềm hoàn thiện khả năng thuần thục quá trình sản xuất phần mềm, đi từ tự phát, hỗn độn tới các quá trình phần mềm thành thục, có kỷ luật.



Hình 1.5: Chuẩn CMM

Bằng việc thực hiện CMM các công ty thu được những lợi ích xác thực, giảm được rủi ro trong phát triển phần mềm và tăng được tính khả năng cạnh tranh - do đó trở thành đối tác hay một nhà cung ứng hấp dẫn hơn đối với các khách hàng trên

toàn thế giới. Tuy nhiên, CMM không phải không đòi hỏi chi phí. Những nguồn lực đáng kể của công ty phải được dành cho việc hướng tới các vùng tiến trình then chốt, cần thiết để lên từng bậc thang của chứng nhận CMM. CMM đưa ra một loạt các mức độ để biểu thị mức độ thành thục đã đạt được. Mức 1 ứng với mức độ thành thục thấp nhất và mức 5 ứng với mức độ thành thục cao nhất. Gần đây, SEI đã xúc tiến CMMi, một mô hình kế thừa CMM và các công ty cũng đang bắt đầu triển khai việc sử dụng mô hình này.

Ngoại trừ mức 1, mỗi mức độ thành thục được phân tích thành các vùng tiến trình chủ chốt, biểu thị những khu vực mà một tổ chức nên tập trung vào để cải thiện tiến trình phần mềm của nó.

Những vùng tiến trình chủ chốt ở mức 2 tập trung vào những vấn đề của dự án phần mềm liên quan tới thiết lập kiểm soát cơ bản cho quản trị dự án. Đó là Quản trị yêu cầu (Requirements Management), Hoạch định Dự án phần mềm (Software Project Planning), Giám sát và theo dõi dự án phần mềm (Software Project Tracking and Oversight), Quản trị hợp đồng phụ phần mềm (Software Quality Assurance), Bảo đảm chất lượng phần mềm (Software Quality Assurance), và Quản trị cấu hình phần mềm (Software Configuration Management).

Các vùng tiến trình chủ chốt ở mức 3 nhằm vào cả hai vấn đề về dự án và tổ chức, vì một tổ chức (công ty) tạo nên cấu trúc hạ tầng thể chế các quá trình quản lý và sản xuất phần mềm hiệu quả qua tất cả các dự án. Chúng gồm có Tập trung Tiến trình Tổ chức (Organization Process Focus), Phân định Tiến trình Tổ chức (Organization Process Definition), Chương trình Đào tạo (Training Program), Quản trị Phần mềm Tích hợp (Integrated Software Management), Sản xuất Sản phẩm Phần mềm (Software Product Engineering), Phối hợp nhóm (Intergroup Coordination), và Xét duyệt ngang hàng (Peer Reviews).

Các vùng tiến trình chủ yếu ở mức 4 tập trung vào thiết lập hiểu biết định lượng của cả quá trình sản xuất phần mềm và các sản phẩm phần mềm đang được xây dựng. Đó là Quản lý quá trình định lượng (Quantitative Process Management) và Quản lý chất lượng phần mềm (Software Quality Management)

Các vùng tiến trình chủ yếu ở mức 5 bao trùm các vấn mà cả tổ chức và dự án phải nhắm tới để thực hiện hoàn thiện quá trình sản xuất phần mềm liên tục, đo đếm được. Đó là Phòng ngừa lỗi (Defect Prevention), Quản trị thay đổi công nghệ (Technology Change Management), và Quản trị thay đổi quá trình (Process Change Management.

Mỗi vùng tiến trình chủ yếu được mô tả như các thực hành cốt yếu thỏa mãn các mục tiêu của nó. Các thực hành cốt yếu đó mô tả hạ tầng và các hoạt động có đóng góp chính cho việc thực hiện và thể chế hoá vùng tiến trình chủ yếu một cách hiệu quả.

Các đặc điểm của 5 mức thành thục CMM:

1. Khởi đầu (Initial). Quá trình sản xuất phần mềm được đặc trưng tự phát, và có lúc thậm chí là hỗn độn. Một số quá trình đã được xác lập, và sự thành công phụ thuộc vào nỗ lực và anh hùng cá nhân.
2. Lặp lại (Repeatable). Các quá trình quản lý dự án cơ bản được thiết lập để theo dõi chi phí, thời gian biểu và chức năng. Đã có quy tắc tiến trình cần thiết để lặp lại các thành công trước đây của các dự án có ứng dụng tương tự.
3. Xác lập (Defined). Quá trình sản xuất phần mềm cho cả hoạt động quản lý và kỹ thuật được văn bản hóa, được chuẩn hóa và được hòa nhập vào quá trình sản xuất phần mềm chuẩn đối với tổ chức. Tất cả các dự án sử dụng một phiên bản "may đo" được phê chuẩn của tiêu chuẩn xí nghiệp cho quá trình sản xuất phần mềm để phát triển và bảo trì phần mềm.
4. Quản trị (Managed). Các biện pháp chi tiết của của quá trình sản xuất phần mềm và chất lượng sản phẩm được thu thập lại. Cả quá trình sản xuất phần mềm và các sản phẩm đều được hiểu và được kiểm tra một cách định lượng.
5. Tối ưu hóa (Optimizing). Cải tiến không ngừng quá trình sản xuất qua phản hồi có định lượng từ quá trình sản xuất và từ thử nghiệm các ý tưởng và công nghệ mới.

Chương 2: Quản lý dự án phần mềm

2.1 Dự án phần mềm và sự cần thiết việc quản lý dự án phần mềm

2.1.1 Định nghĩa dự án và quản lý dự án

Dự án là tập hợp các công việc được thực hiện bởi một tập thể (có thể có chuyên môn khác nhau, thực hiện công việc khác nhau, thời gian tham gia dự án khác nhau), nhằm đạt được một kết quả như dự kiến, trong thời gian dự kiến, với một kinh phí dự kiến.

Trong thuật ngữ của chuyên ngành Công nghệ phần mềm, Quản lý dự án phần mềm là các hoạt động trong lập kế hoạch, giám sát và điều khiển tài nguyên dự án (ví dụ như kinh phí, con người), thời gian thực hiện, các rủi ro trong dự án và cả quy trình thực hiện dự án; nhằm đảm bảo thành công cho dự án. Quản lý dự án phần mềm cần đảm bảo cân bằng giữa ba yếu tố: thời gian, tài nguyên (chi phí) và chất lượng. Ba yếu tố này được gọi là tam giác dự án.

Một dự án được coi là thất bại nếu chi phí vượt quá dự tính 20%, thời gian vượt quá dự tính 20% hoặc tỉ lệ lỗi lớn. Tuy vậy nhiều người cho rằng nếu chi phí hoặc thời gian vượt quá 30% nhưng chất lượng tốt và đáp ứng được yêu cầu thì nên coi là thành công.

2.1.2 Sự cần thiết của Quản lý dự án phần mềm.

- Phát triển phần mềm hiện đại làm theo teamworks
- Cần quản lý và kiểm soát được rủi ro (Risk Control & Risk Management) trong quá trình sản xuất
- Các dự án phần mềm đòi hỏi nhiều nguồn nhân lực với chuyên môn khác nhau
- Tính tích hợp công nghệ cao và sự thay đổi nhanh chóng của công nghệ
- Phải bảo đảm tính chuyên nghiệp trong phát triển dự án phần mềm:
    + Bảo đảm lịch trình của dự án
    + Điều phối và khai thác tối đa nguồn nhân lực hiện có
    + Bảo đảm chất lượng của sản phẩm
- Khả năng khắc phục các sự cố xảy ra khách quan
- Các dự án càng lớn càng cần có sự quản lý chặt chẻ và đồng bộ

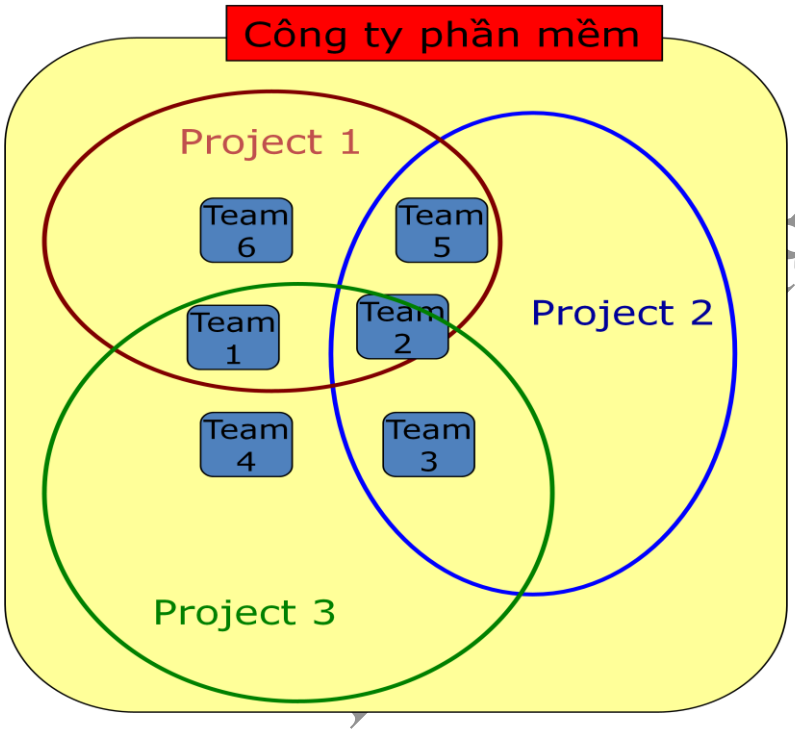
2.2 Các thành phần trong mô hình làm việc của một dự án phần mềm

Mô hình làm việc nhóm (Teamwork) là mô hình hiện tại cho hầu hết các dự án phần mềm:

- Khả năng chuyên nghiệp hóa cao
- Hiệu quả trong quản lý, giao tiếp và điều hành

Một đội dự án phần mềm (software project team) được tạo ra từ nhiều nhóm (sub-teams)

- Các sub-team không nhất thiết là một nhóm người mà có thể là 1 người
- Các sub-team không nhất thiết tồn tại suốt quá trình của một dự án phần mềm



Hình 2.1: Các nhóm trong dự án phần mềm

2.2.1 Vai trò và nhiệm vụ của các nhóm trong dự án phần mềm

- System analysis – Nhóm phân tích hệ thống
- Planning Team – Nhóm lập kế hoạch dự án
- Requirements Team – Nhóm phân tích yêu cầu
- System Design Team – Nhóm thiết kế hệ thống
- Implementation Team – Nhóm lập trình
- Tesing & Intergration Team – Nhóm kiểm thử và tích hợp
- Training Team – Nhóm phụ trách đào tạo
- Delivery & Installation Team – Nhóm phân phối và cài đặt
- Maintenance Team – Nhóm hỗ trợ và bảo trì
- Quality Assurance Team – Nhóm phụ trách chất lượng dự án
- Metrics Team – Nhóm đo lường dự án
- Documentation Team – Nhóm lập tài liệu
- System Administration Team – Nhóm quản trị hệ thống
- Reuse & Reengineering Team – Nhóm quản lý việc sử dụng lại

System Analysis

Xác định tính khả thi của dự án

- Phân tích chi phí (Cost analysis)
- Dự đoán lợi nhận (Estimate revenues)
- Tiên liệu các khó khăn về kỹ thuật và công nghệ
- Sau khi nghiên cứu khả thi, nhóm này sẽ làm việc với Requirement Team để nhận feedbacks
- Nếu dự án được phát triển theo mô hình tương tác cao như Prototype/Spiral model thì tính tương tác và feedback là rất quan trọng kể cả với các nhóm khác.

Planning Team

Nhóm này có nhiệm vụ xây dựng tổng thể tất cả các kế hoạch quản trị dự án và bảo đảm các tiến trình diễn ra đúng tiến độ đã định

- Xây dựng các kế hoạch thực hiện
- Lập các time frame cho các tiến trình
- Kế hoạch sử dụng tài nguyên của hệ thống bao gồm cả nhân lực
- Các kế hoạch dự phòng và điều chỉnh khi có sự cố

Requirement Team

Tiếp xúc khách hàng và xác định đầy đủ, hoàn chỉnh và chính xác các yêu cầu cho dự án

- Dùng các phương thức gặp gở chính thức và bên lề để xác định các yêu cầu của hệ thống
- Nếu không có khách hàng, có thể tiếp xúc với các user tiềm năng

Sau khi xác định các yêu cầu, nhóm này sẽ làm việc với System Design Team để nhận các feedback.

Nếu dự án được phát triển theo mô hình tương tác cao như Prototype/Spiral model thì tính tương tác và feedback là rất quan trọng kể cả với các nhóm khác

System Design Team

Xây dựng thiết kế chi tiết của hệ thống sau khi các yêu cầu đã được xác định.

Nếu sử dụng mô hình Waterfall, nhóm này phải feedback cho nhóm Requirement những khó khăn nếu có.

Sau khi hoàn chỉnh thiết kế, nhóm này phải cộng tác với Implementation Team để nhận feedback.

Nếu dự án được phát triển theo mô hình tương tác cao như Prototype/Spiral model thì tính tương tác và feedback là rất quan trọng kể cả với các nhóm khác

Implementation Team

Phát triển hệ thống theo thiết kế đã có.

- Coding
- Kiểm tra cấp Module

Sau khi hoàn tất chương trình, nhóm này sẽ cộng tác với nhóm Tesing & Integration để kiểm tra các module

Nếu dự án được phát triển theo mô hình tương tác cao như Prototype/Spiral model thì tính tương tác và feedback là rất quan trọng kể cả với các nhóm khác

Testing & Integration Team

Xây dựng thiết kế chi tiết của hệ thống sau khi các yêu cầu đã được xác định.

Nếu sử dụng mô hình Waterfall, nhóm này phải feedback cho nhóm Requirement những khó khăn nếu có.

Sau khi hoàn chỉnh thiết kế, nhóm này phải cộng tác với Implementation Team để nhận feedback.

Nhóm này có thể tiếp nhận các module rời rạc và kiểm tra sau đó tích hợp thành hệ thống hoàn chỉnh.

Nếu dự án được phát triển theo mô hình tương tác cao như Prototype/Spiral model thì tính tương tác và feedback là rất quan trọng kể cả với các nhóm khác

Nhóm này cũng có vai trò trong Interface Control Document để đặc tả các giao diện và giao tiếp giữa các thành phần trong hệ thống

Trainning Team

Chuẩn bị các công cụ và tài liệu cho việc trainning cho người dùng

- Kế hoạch trainning
- Các tài liệu giảng dạy

Delivery & Installation Team

Nhiệm vụ là cài đặt hệ thống cho khách hàng và các hỗ trợ kỹ thuật trong cài đặt vận hành hệ thống.

Maintenance Team

Bảo trì hệ thống sau khi chuyển giao và cài đặt

- Cập nhật sửa chữa
- Nâng cấp mở rộng

Cộng tác chặt chẻ với nhóm implementation để thực hiện việc maintenance

Quality Assurance Team

Nhóm này có 2 nhiệm vụ

1. Thiết lập các tiêu chuẩn cho các quá trình sản xuất cũng như tiêu chuẩn thực hiện của sản phẩm phần mềm
2. Cung cấp các cơ chế kiểm tra, kiểm soát nhằm đánh giá khả năng thỏa mãn các tiêu chuẩn tương ứng của các nhóm làm việc.

Các tiêu chuẩn này dùng trong nội bộ và không chia sẻ với khách hàng.

Các tiêu chuẩn có thể được công bố khi cần thiết, vì vậy cần được lưu trữ và báo cáo cho project manager để hoạt động với bộ phận Q&A

Metrics Team

Lưu trữ các thông tin thống kê về các hoạt động của các TEAm trong dự án.

- Số lượng các yêu cầu maintenance
- Số lượng thực hiện dịch vụ maintenance
- Số dòng code được viết
- Thời gian thực hiện từng công việc

Nhóm này làm việc với hầu hết các nhóm để cung cấp báo cáo về chất lượng, hiệu quả, đồng thời feedback cho các nhóm đó về hiệu quả công việc.

Documentation Team

Nhóm này thực hiện các hoạt động thiết lập các tài liệu cho hệ thống

- Tài liệu về phân tích, thiết kế, hiện thực, source code,..
- Tài liệu hổ trợ : userguide, manual, support document

System Administrationm Team

Nhóm này có nhiệm vụ cung cấp và bảo đảm các hoạt động của các hệ thống hạ tầng kỹ thuật cần thiết cho dự án

Nhóm này thông thường bao gồm cả Network Administration Team

Reuse & Reengineering Team

Chọn lựa và quyết định việc reuse các module đã có

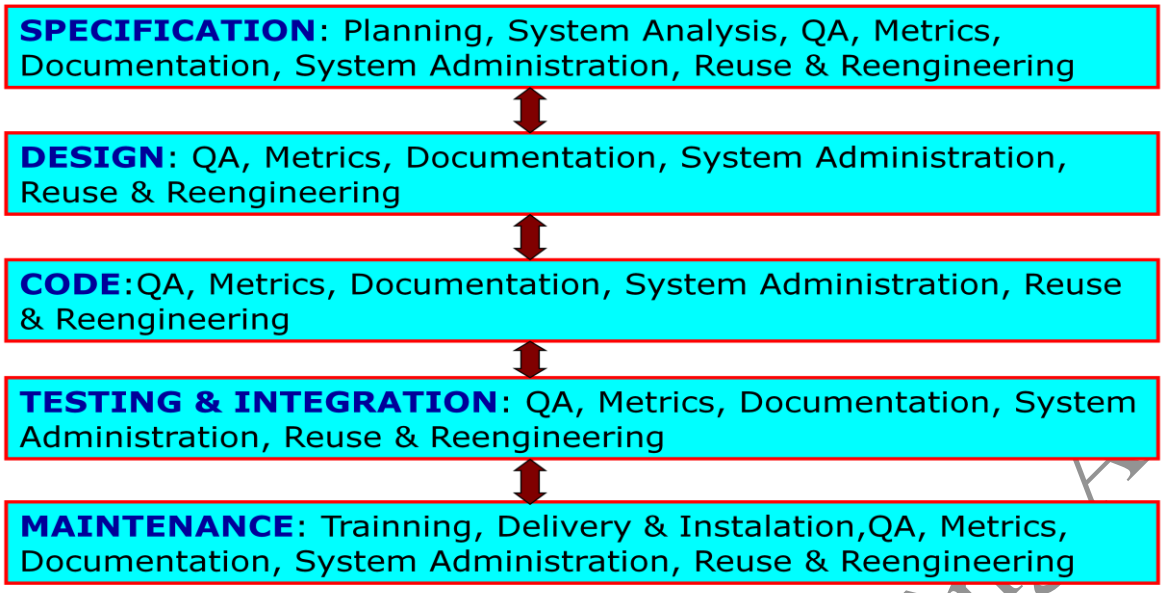
Việc reengineering cũng cần thiết khi mà việc phát triển đòi hỏi phải dùng đến các code cũ khi công nghệ đã thay đổi.

Có rất nhiều việc quản lý trong từng nhóm và từng công đoạn.

Có khá nhiều nhóm nên có nhiều quản lý nhóm (manager), tuy nhiên nếu các nhóm này nhỏ thì thường là một người trong nhóm sẽ là manager của team.

Việc lập lịch trình (scheduling) giữa các team phụ thuộc vào mô hình phát triển cụ thể là gì.

Ví dụ nếu dùng mô hình Waterfall thì các công đoạn có thể được tích hợp thành các bước lớn hơn, các nhóm tham gia vào từng bước theo chức năng của mình.



Hình 2.2: Các nhóm trong mô hình Waterfall

2.2.2 Các nhân sự khác trong dự án

Bên cạnh còn có một số nhân sự khác tham gia vào quá trình phát triển dự án nhưng có thể không được nêu tên một cách chính qui

- **Human-Computer Interface Evaluation**: Đánh giá khả năng thích hợp của giao diện cả như người dùng cấp thấp và cấp cao
- **Tools Support Person**: Người cung cấp và bảo đảo các công cụ cần thiết như tools, software, network vận hành theo yêu cầu của quá trình phát triển
- **Software Economist**: Sử dụng các mô hình đánh giá cần thiết để ước lượng chi phí phần mềm, phần cứng, resource và thời gian cần cho dự án hoàn tất
- **Project Librarian**: Có trách nhiệm lưu trữ và sắp xếp hệ thống tất cả các tài liệu của dự án
- **Chuyên gia hỗ trợ**: Một số dự án cần có những chuyên gia trong lĩnh vực tương ứng hỗ trợ, tư vấn về mặt chuyên môn hay kỹ thuật

➢ Nhân sự của các team có thể thay đổi thường xuyên trong quá trình hoạt động do nhiều yếu tố.

2.2.3 Các yếu tố ảnh hưởng đến các nhóm trong dự án

Nhân sự cần thay đổi theo từng công đoạn: các công đoạn cần nhiều nhân sự và cần thời gian dài như lập trình (coding), kiểm thử (testing) & tích hợp (integration).

Các nguyên nhân khách quan khác:

- Nhân sự thay đổi công việc: chuyên môn thay đổi, công nghệ mới cập nhật
- Nhân sự nghĩ do thay đổi việc, bệnh, về hưu
- Nhân sự mới: mang lại tư duy mới và công nghệ mới tuy nhiên phải cần thời gian tiếp cập

Việc xây dựng các team là linh động theo từng dự án và cần có điều phối giữa các dự án theo từng tiến độ công việc.

Công việc của một quản trị dự án Project Manager

- Quản trị nhân sự: điều phối, quản lý công việc,..
- Phân bổ các tài nguyên của hệ thống theo kế hoạch
- Điều phối nhân sự: trong công ty và bên ngoài
- Xữ lý các phát sinh về thời gian biểu
- Quản lý các thay đổi yêu cầu của dự án
- Giải quyết các sự cố ngoài kế hoạch: máy móc hư hỏng, nhân sự thay đổi,..
- Báo cáo cho lãnh đạo về dự án
- Giao tiếp với khách hàng
- Huấn luyện nhân viên Staffs trainning

2.3 Ước lượng dự án

Ước lượng dự án hiện là khâu yếu nhất hiện nay. Không ước lượng được thì dự án rất dễ vỡ kế hoạch về thời gian và tài chính.

Thực tế không dự án nào có thể ước lượng chính xác, ước lượng cần được thực hiện nhiều vòng. Mức ước lượng trong giai đoạn xác định có thể sai tới 50-100%, nhưng trong giai đoạn thiết kế phải giảm tới 25-50%, còn trong giai đoạn thiết kế chi tiết chỉ còn 10-25%

Ước lượng chỉ có thể chính xác nếu phân rã được các vấn đề nhỏ hơn, đó là kỹ thuật chia để trị (divide and conquer)

Các phương pháp ước lượng

- Ước lượng chuyên gia: các chuyên gia đã có kinh nghiệm triển khai dự án phần mềm, có thể trả lời ngay các ước lượng tuy rằng không phải lúc nào độ chính xác cũng đáng tin cậy
- Đánh giá bằng kinh nghiệm quá khứ. Phải có số liệu quá khứ, phải hiểu được tình hình hiện tại
- Đánh giá bằng các mô hình ước lượng thực nghiệm. Phải có các tham số về dự án (các độ đo)

2.3.1 Độ đo

Khái niệm độ đo: là các chỉ số đặc trưng cho một khía cạnh nào đó. Trong công nghệ phần mềm có độ đo của phần mềm (software metric/software mesure), độ đo của dự án (project metric) và độ đo của quy trình phần mềm (process metric).

Có độ đo trực tiếp và độ đo gián tiếp. Độ đo trực tiếp là độ đo có thể tính đếm trực tiếp không thông qua các độ đo khác (ví dụ độ đo LOC – lines of code), có độ đo gián tiếp là các độ đo tính qua các độ đo khác (ví dụ tỉ lệ lỗi = số lỗi / số dòng mã nguồn

Dự án cũng có độ đo, chi phí cho dự án, năng suất của dự án,

Quy trình phần mềm cũng có độ đo, chẳng hạn tỉ lệ chi phí trung bình cho mỗi giai đoạn phát triển phần mềm đối với quy trình thác nước

2.3.2 Độ đo LOC - Metric hướng quy mô phần mềm

LOC (lines of code) hay KLOC (nghìn dòng lệnh). Độ đo này chỉ có thể chính xác sau khi dự án đã kết thúc. Tuy nhiên bằng kinh nghiệm, hoặc bằng thống kê tương tự có thể ước lượng đượcc khối lượng mã nguồn của một phần mềm trước khi kết thúc dự án.

LOC sau khi kết thúc dự án sẽ được dùng để ước lượng các dự án tương tự sau này.

Các độ đo dẫn xuất: số lỗi trên KLOC, chi phí trên KLOC, số tài liệu trên KLOC, năng suất số KLOC /manmonth.

LOC phụ thuộc vào môi trường lập trình nên khó so sánh giữa các dự án nếu chúng phát triển trên các môi trường lập trình khác nhau

2.3.3 Điểm chức năng Function Point – Metric hướng chức năng

Điểm chức năng (FP) đo độ phức tạp của phần mềm. Quy mô chỉ phản ánh một khía cạnh nhỏ của độ phức tạp, chính chức năng thể hiện độ phức tạp chính xác hơn

FP được tính qua 5 yếu tố chính và 14 yếu tố phụ. Các yếu tố chính là

- Số user input (số các thành phần dữ liệu đưa vào), số các input được dùng trong các câu hỏi khác nhau được tính riêng rẽ.
- Số user output (xuất hiện trong các report, các màn hình, các thông báo). Các output trong các câu hỏi khác nhau được kể riêng rẽ
- Số truy vấn (inquiry) của người sử dụng - số input trong các truy vấn online
- Số lượng file logic (có thể chỉ là một phần của CSDL, có thể tính như một bảng của CSDL) và các file độc lập
- Số lượng các giao tiếp ngoài: ngoại vi, các hệ thống thông tin khác mà nó giao tiếp

Mỗi yếu tố trên được gán một trọng số, tuỳ theo ảnh hưởng của mỗi yếu tố và tuỳ theo mức độ phức tạp: thường tính theo 3 mức là đơn giản, trung bình và phức tạp. Ví dụ

	Tham số đo	Số đo	ĐG	TB	PT	
1	Số input	25	3	4	6	100
2	Số output	30	4	5	7	150
3	Số inquiry	20	3	4	6	120
4	Số file	10	7	10	15	70
5	Số tương tác ngoài	10	5	7	10	70
	Tổng					510

Bảng 2.1: Tham số đo của Function Point

14 yếu tố điều chỉnh phụ của Function Point

1. Hệ thống đòi hỏi backup và hồi phục tin cậy
2. Đòi hỏi dữ liệu truyền thông
3. Có các chức năng phân tán
4. Hiệu năng là điều quan trọng
5. Yêu cầu sử dụng môi trường nặng
6. Hệ thống đòi hỏi dữ liệu on-line
7. Khi đòi hỏi dữ liệu online, cần nhiều màn hình dữ liệu hoặc nhiều xử lý
8. Master file được cập nhật online
9. Input, output, file, và tính toán online phức tạp
10. Quá trình xử lý bên trong phức tạp
11. Mã được thiết kế để dùng lại
12. Việc chuyển đổi và cài đặt được tính ngay trong thiết kế
13. Hệ thống được thiết kế để có thể cài đặt nhiều lần cho các tổ chức khác nhau

14. Ứng dụng được thiết kế để dễ thay đổi và làm dễ dàng sử dụng cho người dùng

2.3.4 Mô hình ước lượng thực nghiệm

Hầu hết các mô hình thực nghiệm đều phải có đầu vào từ một độ đo độ phức tạp của dự án có thể là LOC hay FP

Mô hình chính E= A + B x V^c trong đó E là công sức có thể đo bằng ngườixtháng, B và C là một hằng số đặc trưng cho mô hình và v là LOC hay FP. Mô hình này cho thây công sức không tỉ lệ tuyến tính theo độ phức tạp

Một vài ước lượng

- Waston Felix E = 5.2 + $KLOC^{0.91}$
- Baley Basil E = 5.5 + 0.73 $KLOC^{1.16}$
- Matson Barret E= 585.7+15.12xFP

2.3.5 Mô hình ước lượng thực nghiệm COCOMO

COCOMO : Constructive Cost Model (Barry Boehm- 1981)

COCOMO II (1996) có phân mức đối với các dự án: dự án ở mức định hướng (làm bản mẫu,xem xét công nghệ, giao diện, hiệu năng), dự án ở mức trước khi thiết kế (khi yêu cầu phần mềm đã ổn định và kiến trúc đã được xác lập) và dự án ở tầng sau kiến trúc (khi phần mềm đươc xây dựng)

3 Mô hình cơ bản của COCOMO

- Mô hình 1. Mô hình COCOMO cơ sở tính công sức phát triển phần mềm (và chi phí) xem như hàm của kích cỡ chương trình được diễn đạt theo số dòng mã ước lượng.
- Mô hình 2. COCOMO trung bình tính công sức phát triển phần mềm như một hàm của kích cỡ chương trình và một tập các "hướng dẫn chi phí" bao hàm cả các đánh giá chủ quan về sản phẩm, phần cứng, nhân sự và các thuộc tính dự án.
- Mô hình 3. COCOMO nâng cao tổ hợp của tất cả các đặc trưng của phiên bản trung bình với việc đánh giá của ảnh hưởng của hướng dẫn chi phí lên từng bước (phân tích, thiết kế ...) của tiến trình kĩ nghệ phần mềm.

Theo Boehm, các mô hình 1 và 2 có thể áp dụng cho ba lớp dự án là

1. kiểu tổ chức - các dự án phần mềm tương đối nhỏ, đơn giản trong đó các nhóm nhỏ có kinh nghiệm ứng dụng tốt làm việc với một tập các yêu cầu ít chặt chẽ hơn (như chương trình phân tích nhiệt được phát triển cho nhóm truyền nhiệt);
2. kiểu nửa gắn - một dự án phần mềm trung bình (về kích cỡ và độ phức tạp) trong đó nhóm với nhiều mức độ kinh nghiệm phải đáp ứng cho các yêu cầu chặt chẽ và kém chặt chẽ (như hệ thống xử lý giao tác với yêu cầu cố định cho phần cứng thiết bị đầu cuối và phần mềm cơ sở dữ liệu);

3. kiểu nhúng - một dự án phần mềm phải được phát triển bên trong một tập phần cứng, phần mềm, và các ràng buộc chặt chẽ (như phần mềm kiểm soát bay của các máy bay).

➢ Phương trình COCOMO cơ bản có dạng sau:

$E = aKLOC^b$, $D = cE^d$

Trong đó E là công sức được áp dụng theo người/tháng (man/month), D là thời gian phát triển theo người/tháng, và KLOC là số được ước lượng về số dòng mã phải bàn giao

Mô hình cơ sở được mở rộng để xem xét một tập các "thuộc tính hướng dẫn chi phí" thuộc tính sản phẩm, các thuộc tính phần cứng, các thuộc tính nhân viên và các thuộc tính dự án. Boehm đưa vào nhân tố điều chỉnh công sức (EAF). Giá trị điển hình cho EAF lấy trong miền từ 0.9 đến 1.4.

➢ Phương trình COCOMO trung bình có dạng

$E = a \times KLOC^b \times EAF$

	ai	bi	ci	di
Tổ chức	3.2	1.05	2.5	0.38
Nửa gắn	3.0	1.12	2.5	0.35
Nhúng	2.8	1.20	2.5	0.32

Bảng 2.2: Tham số của phương trình COCOMO

Thực tế triển khai, COCOMO được chi tiết hoá rất nhiều, người ta xây dựng các phiên bản phụ thuộc vào điều kiện cụ thể, một số các tham số của COCOMO phải dựa vào số liệu quá khứ. Đã có một số phần mềm ước lượng

Người ta tính nỗ lực theo từng giai đoạn (thiết kế sơ bộ, thiết kế chi tiết, lập trình và kiểm thử module, kiểm thư). Tham số đầu vào của COCOMO là chi phí hàng tháng cho nhân viên tuỳ theo từng loại như người phân tích, người lập trình, người kiểm thử, nhân viên hành chính, nhân viên làm tài liệu, mỗi loại đó đều có một trọng số để điều chinh

2.4 Lập kế hoạch dự án

Lập kế hoạch thực hiện dự án là hoạt động diễn ra trong suốt quá trình từ khi bắt đầu thực hiện dự án đến khi bàn giao sản phẩm với nhiều loại kế hoạch khác nhau nhằm hỗ trợ kế hoạch chính của dự án phần mềm về lịch trình và ngân sách.

2.4.1 Các loại kế hoạch và nguồn lực thực hiện dự án

Các loại kế hoạch thực hiện dự án:

- Kế hoạch công việc: Mô tả công việc và lịch biểu thực hiện cho sản phẩm dự án.
- Kế hoạch quản lý rủi ro: Xác định các rủi ro và các giải pháp.
- Kế hoạch chất lượng: Mô tả thủ tục và các chuẩn chất lượng được áp dụng.
- Kế hoạch quản lý cấu hình: Mô tả cấu hình, thủ tục và tiến trình quản lý cấu hình và sự thay đổi.
- Kế hoạch ngân sách: Chỉ ra lượng ngân sách cần cho thời gian và các nguồn huy động.
- Kế hoạch nguồn lực: Mô tả số lượng, kỹ năng, kinh nghiệm của thành viên dự án cần và giải pháp.

Nguồn lực thực hiện dự án:

Nhân tố con người

- Nhân tố quan trọng nhất.
- Cần có năng lực nhất định, cơ cấu vị trí làm việc phù hợp.
- Mọi giai đoạn của dự án, nhu cầu về con người là khác nhau.

Phần mềm dùng lại được

- Thành phần đóng gói (dùng ngay)
- Thành phần đã kiểm nghiệm tốt (sửa dùng được)
- Thành phần có thể dùng (chi phí sửa lớn)

Phần cứng/công cụ phần mềm chia sẻ

2.4.2 Cấu trúc kế hoạch thực hiện dự án:

- Mở đầu
- Tổ chức thực hiện dự án
- Phân tích các rủi ro
- Yêu cầu về nguồn lực(tài nguyên): Nhân lực, phần cứng, phần mềm
- Bảng phân rã công việc
- Lập lịch dự án
- Cơ chế kiểm soát và báo cáo
- Các kế hoạch phụ trợ

Tổ chức hoạt động dự án:

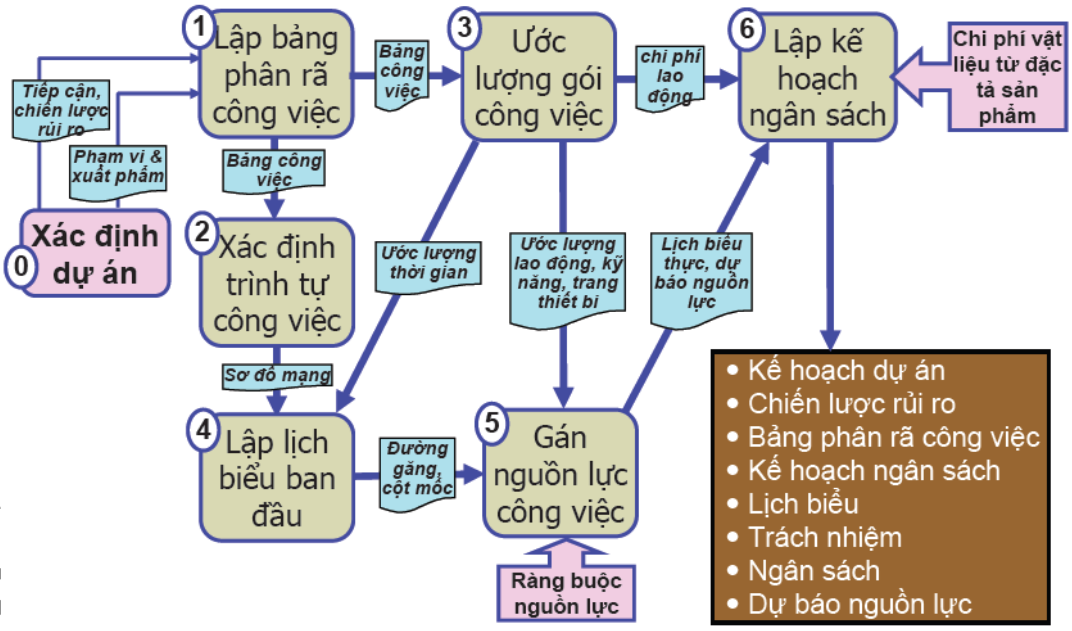
- Tổ chức bộ máy và cơ chế hoạt động: ban quản lý, các nhóm làm việc (team, sub-Team), chơ chế điều hành giám sát, báo cáo.
- Hoạt động dự án cần tổ chức tạo ra các đầu ra thấy được của mỗi tiến trình quản lý.

- Cột mốc (Milestone) là điểm cuối của một tiến trình hoạt động có xuất phẩm và các báo cáo yêu cầu.
- Xuất phẩm (Diliverables) là kế quả dự án gửi đến khách hàng/quản lý dự án giám sát tại mỗi thời điểm.

2.4.3 Quy trình lập kế hoạch thực hiện dự án

Nội dung hoạt động kế hoạch

1. Xây dựng bảng phân rã công việc: Đội dự án và người quản lý dự án xác định các nhiệm vụ (gói công việc) cần thực hiện để tạo ra các sản phẩm.
2. Xác định các mối quan hệ giữa các công việc được phân rã: đặt các gói công việc theo một tiến trình có trình tự trước-sau.
3. Ước lượng các gói công việc: mỗi gói công việc có ước lượng công lao động, số trang thiết bị và thời gian cần thiết để thực hiện.
4. Xây dựng lịch biểu ban đầu: tính toán thời gian thực hiện dự án, thời gian bắt đầu sớm nhất & kết thúc muộn nhất của từng công việc.
5. Gán nguồn lực thực hiện, điều chỉnh lịch: sau khi gán nguồn lực, cần chính xác hoá lịch biểu khi tính đến các ràng buộc về nguồn lực. Các nhiệm vụ được lập lịch sao cho tối ưu hoá việc sử dụng lao động và các nguồn lực khác.



Hình 2.3: Qui trình lập kế hoạch dự án

2.4.4 Lập lịch dự án

2.4.4.1 Bảng phân rã công việc

Cấu trúc bảng phân rã công việc bao gồm:

- Các công việc
- Mối liên hệ (trước-sau) giữa các công việc

- Thời gian thực hiện
- Nguồn lực cần thực hiện công việc

Ý nghĩa của bảng phân rã công việc:

- Hình dung đầy đủ công việc dự án cần làm
- Cơ sở để ước lượng chi phí và thời gian
- Cơ sở cho lập lịch
- Cơ sở để bố trí nguồn lực và phân bổ tài nguyên

Các bước xây dựng bảng phân rã công việc

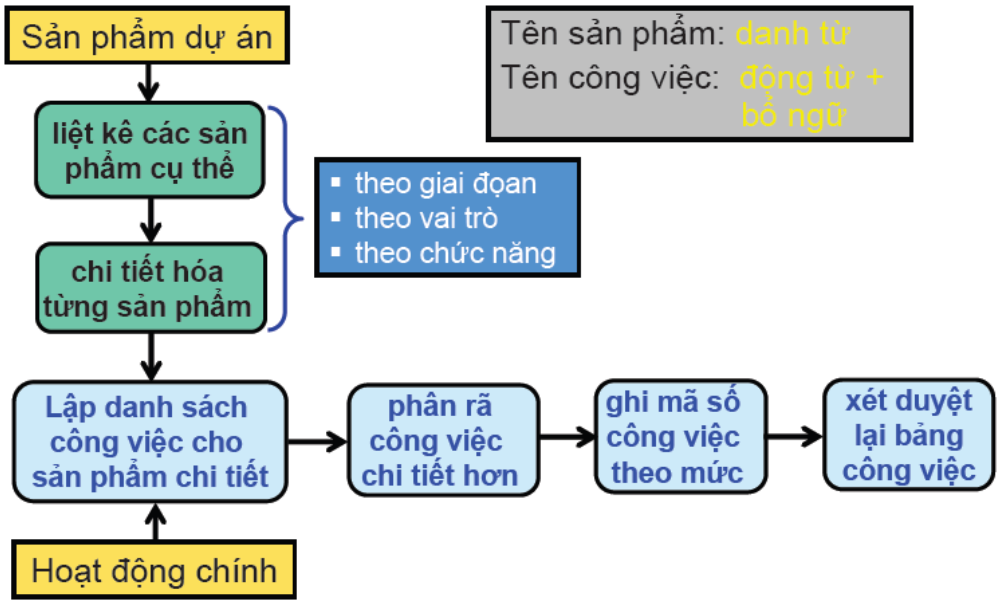
Bước 1: Viết ra sản phẩm chung nhất (lấy ra từ bản dự án cơ sở)

Bước 2: Tạo danh sách các sản phẩm chi tiết ở các mức thấp hơn (khoảng 2,3 mức)

Bước 3: Tạo danh sách các công việc thực hiên sản phẩm ở mức thấp nhất, phân rã các công việc để được các công việc ở mức thấp hơn đến mức đạt yêu cầu.

Bước 4: Đánh mã số cho mỗi công việc, nhóm lại.

Bước 5: Xét duyệt lại bảng công việc.



Hình 2.4: Quy trình xây dựng bảng phân rã công việc

2.4.4.2 **Lập lịch dự án**

Nội dung hoạt động lập lịch dự án

Đầu vào: Bảng phân rã công việc, các nguồn lực thực hiện và tổng số.

Tiến hành lập lịch:

- Lập mạng công việc
- Tính thời gian bắt đầu sớm nhất
- Tính thời gian kết thúc muộn nhất

- Tính thời gian dự phòng
- Tính công việc găng, đường găng (Critical path)
- Tối ưu hoá thực hiện công việc khi tính đến các ràng buộc về nguồn lực

Tính thời gian thực hiện các công việc:

Ước lượng PERT (Program Evaluation and Review Technique)

Thích hợp với các dự án đòi hỏi tính sáng tạo

Quan trọng chất lượng kết quả công việc hơn thời gian hoàn thành dự án

Công thức PERT

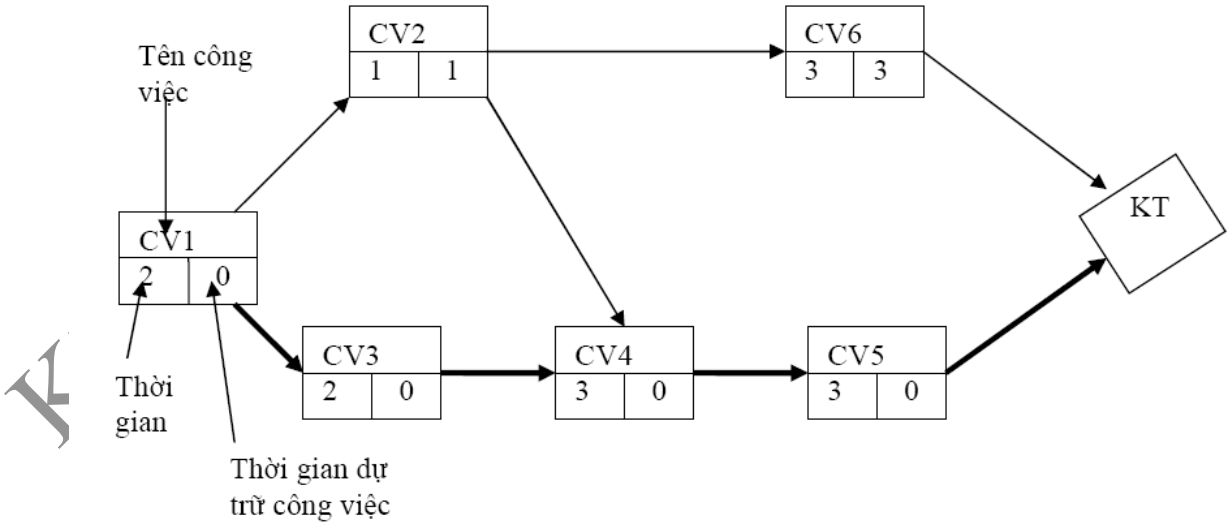
Cần 3 ước lượng thời gian cho mỗi công việc

Kết hợp để cho con số cuối cùng

- Ước lượng khả dĩ (ML – Most Likely): Thời gian cần hoàn thành công việc trong điều kiện bình thường hay hợp lý.
- Ước lượng lạc quan nhất (MO – Most Optimistic): Thời gian cần hoàn thành công việc trong điều kiện tốt nhất hay lý tưởng.
- Ước lượng bi quan nhất (MP – Most Pessimistic): Thời gian cần hoàn thành công việc trong điều kiện xấu nhất.
- Ước lượng thời gian theo công thức: EST = (MO + 4ML + MP)/6

Sau đó tăng một ít thời gian cho các công việc (thời gian lãng phí giữa chừng) thông thường tăng thêm từ 7-10%.

Xây dựng biểu đồ PERT



Hình 2.5: Biểu đồ PERT

Ưu điểm của PERT:

- Buộc tính đến nhiều yếu tố khi xác định các giá trị MO, MP, ML
- Người quản lý phải trao đổi với nhiều người để đạt được sự đồng thuận

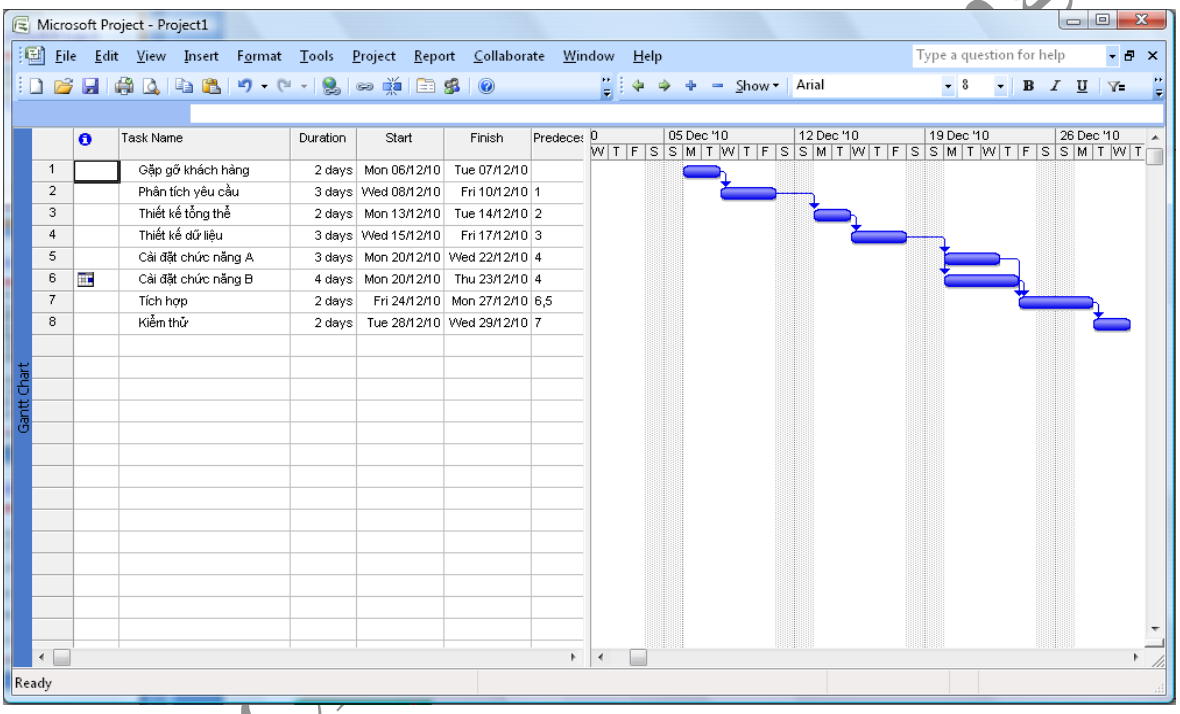
– Giá trị nhận được là cân bằng giữa hai thái cực, có ý nghĩa đáng tin cậy để lập kế hoạch chi tiết hơn. Nếu ước lượng > 2 tuần hoặc 80 giờ thì thực hiện phân rã công việc tiếp.

Nhược điểm:

– Tốn thời gian khi có nhiều công việc
– Có thể xảy ra tranh luận nhiều dẫn tới sự chán nản của các thành viên dự án.
– Có thể dẫn đến tính vụn vặt.

Lập lịch bằng công cụ Microsoft Project:

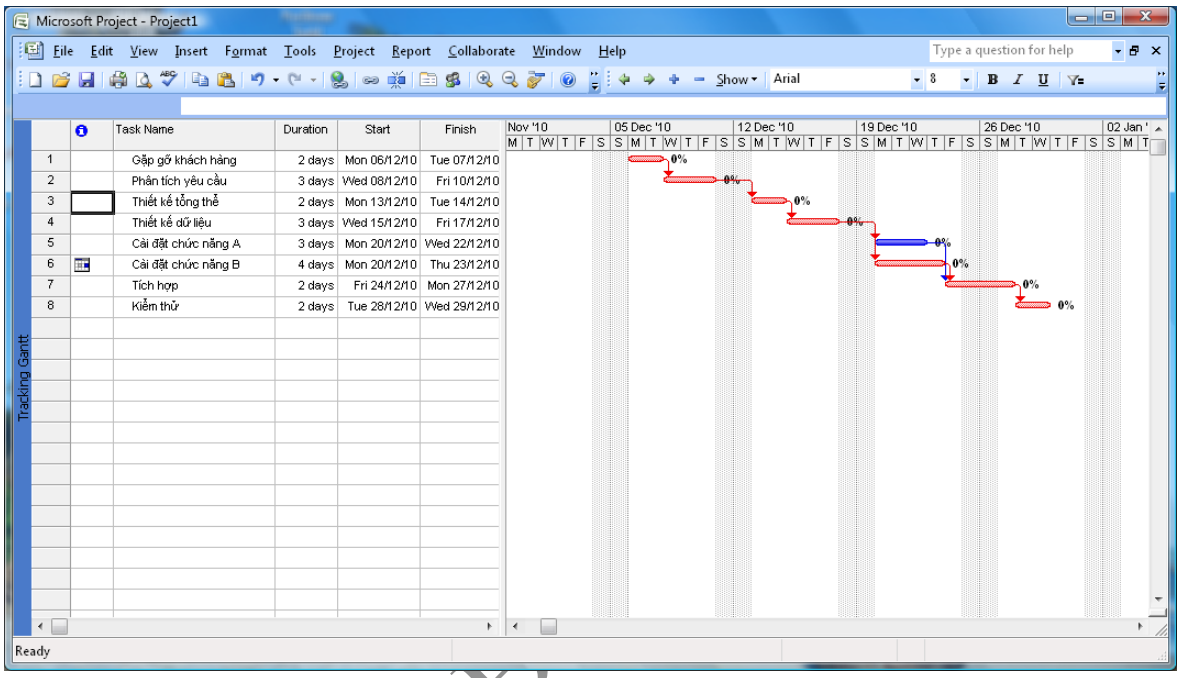
Lập biểu đồ Gantt công việc:



Hình 2.6: Lập biểu đồ Gantt bằng Microsoft Project

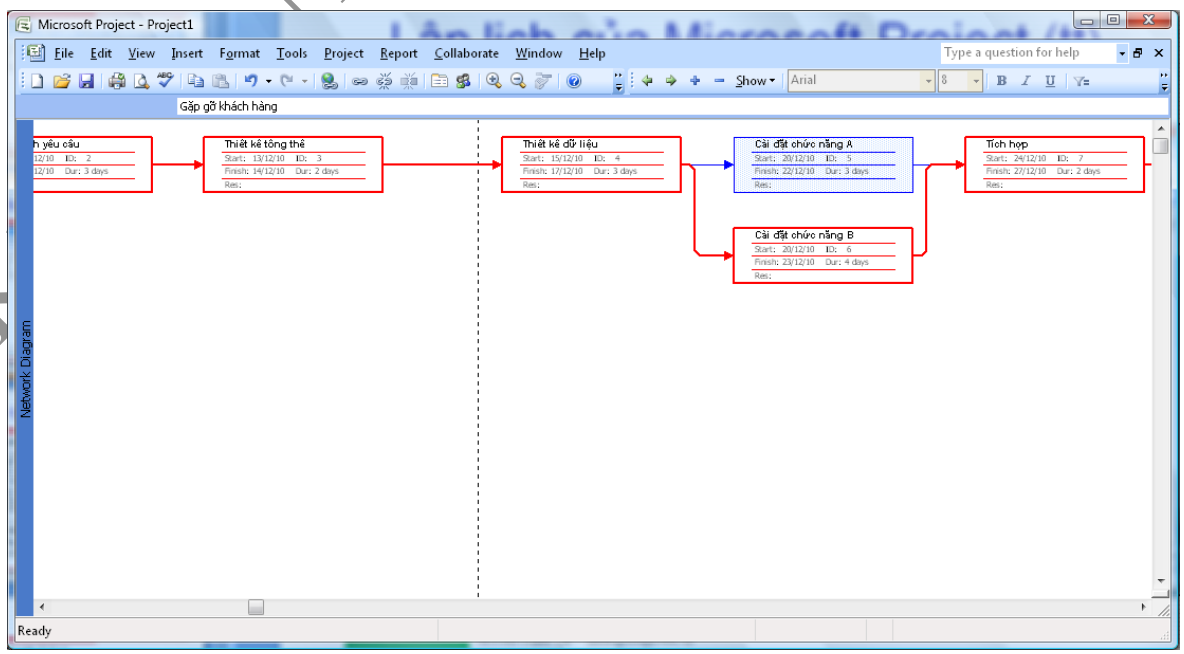
Xác định đường găng trên biểu đồ Gantt:

Đường găng là đường thực hiện công việc có thời gian dài nhất trên sơ đồ mạng công việc. Các công việc trên đường găng không được phép trễ hạn. Chỉ cần một công việc trên đường găng trễ hạn là cả dự án trễ theo. Do đó ưu tiên và quan tâm đến các công việc trên đường găng, phân công nhân sự giỏi, có kinh nghiệm, thiết bị tốt, thường xuyên cập nhật thay đổi trên đường găng.



Hình 2.7: Xác định đường găng trên biểu đồ Gantt

Sơ đồ mạng công việc:



Hình 2.8: Sơ đồ mạng công việc

2.5 Quản lý rủi ro

2.5.1 Định nghĩa rủi ro và quản lý rủi ro

Định nghĩa rủi ro

Những sự kiện có thể làm phá vỡ một dự án.

Những điều không chắc chắn, những khoản nợ hay những điểm yếu có thể làm cho dự án không đi theo đúng kế hoạch đã định.

Có thể quản lý được.

Định nghĩa quản lý rủi ro

Quy trình quản lý rủi ro nhằm giảm tối thiểu ảnh hưởng của những sự cố không biết trước cho dự án bằng cách xác định và đưa ra những giải pháp tình huống trước khi có những hậu quả xấu xảy ra

Lý do cần quản lý rủi ro

- Tất cả các dự án đều phụ thuộc vào rủi ro
- Tiến trình sẽ không đúng theo kế hoạch trong một số giai đoạn của dự án
- Rủi ro không thể được loại trừ triệt để

Giá trị của quản lý rủi ro

- Giảm thiểu ảnh hưởng của các sự cố không biết trước cho dự án
- Nâng cao xác suất thực hiện thành công dự án
- Tạo ra ý thức kiểm soát
- Có được các giải pháp hiệu quả và kịp thời

2.5.2 Nhận diện rủi ro

- Nhận diện các rủi ro tiềm tàng
- Ai tham gia nhận diện và giải quyết
- Tầm quan trọng: được chuẩn bị. Phần mềm là một lĩnh vực khó. Nhiều thứ có thể sai, đó là lý do mà phải chuẩn bị - việc hiểu rủi ro và làm những công việc ước lượng trước để tránh hay quản lý chúng – là một thành phần cơ bản của hoạt động quản lý dự án phần mềm
- Những bước nào sẽ phải làm việc để quản lý rủi ro
- Sản phẩm của người làm quản lý: RMMM (risk mitigation, monitoring & management)
  - ➢ Cần đưa ra một chiến lược phòng, chống rủi ro tổng quát trong CNPM

Ba loại rủi ro

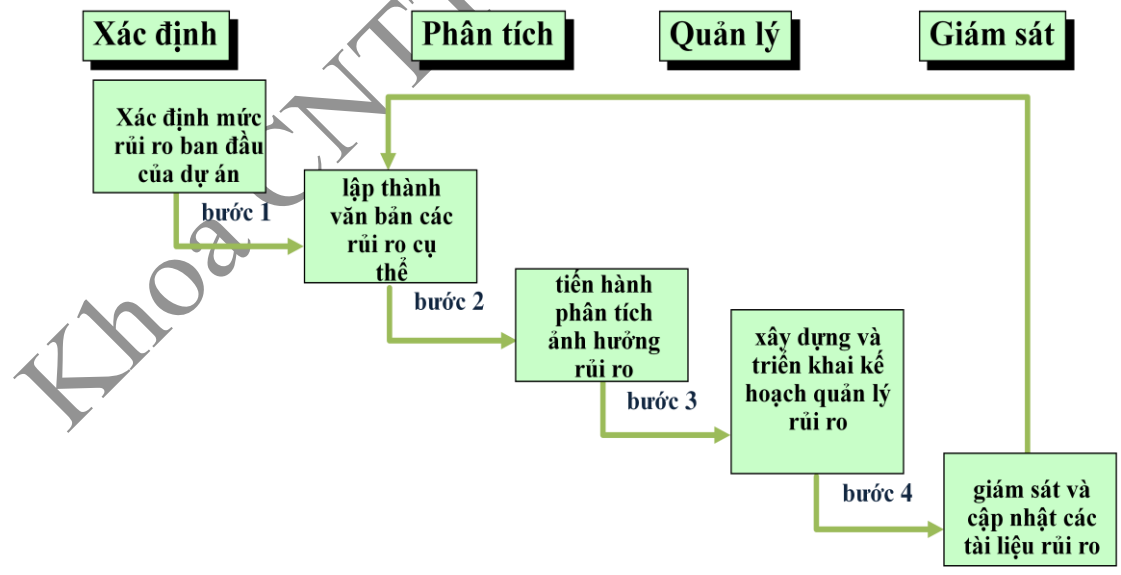
- **Rủi ro dự án** là mối đe doạ cho kế hoạch dự án: Kế hoạch lịch trình sai sẽ làm tăng chi phí. Có thể sai trong dự tính ngân sách, kế hoạch, cá nhân(nhân viên, tổ chức), tài nguyên, khách hàng, và những yêu cầu và ảnh hưởng của chúng

- **Rủi ro kĩ thuật** là mối đe doạ chất lượng và tính đúng đắn của phần mềm được sản xuất. Nếu một lỗi kĩ thuật trở thành hiện thực, sự cài đặt có thể trở lên khó khăn hoặc không thể. Rủi ro kĩ thuật được tìm ra trong thiết kế, cài đặt, giao diện, sự kiểm tra, và vấn đề bảo trì. Thêm vào đó, sự tối nghĩa, kĩ thuật không vững chắc, kĩ thuật lỗi thời, và công nghệ "giới hạn sự hướng dẫn" cũng là tác nhân của rủi ro. Rủi ro kĩ thuật xảy ra vì vấn đề khó giải quyết hơn chúng ta nghĩ nó sẽ xảy ra.
- **Rủi ro nghiệp vụ** là mối đe doạ khả năng tồn tại của phần mềm được xây dựng. Rủi ro nghiệp vụ thường gây nguy hiểm cho dự án hoặc sản phẩm. Dự tính có 5 loại rủi ro nghiệp vụ có thể là (1) rủi ro thị trường, (2) rủi ro chiến lược), (3) xây dựng một sản phẩm với nỗ lực để bán nhưng không hiểu phải bán như thế nào, (4) rủi ro quản lý), và (5) rủi ro ngân sách).

Nhận diện rủi ro

- Quy mô sản phẩm (product size)
- Ảnh hưởng của thị trường (Businees Impact)
- Đặc tính của khách hàng (Customer Characteristics)
- Xác định quy trình (Process Definition)
- Môi trường phát triển
- Công nghệ để xây dựng phần mềm
- Quy mô và kinh nghiệm của nhân viên

2.5.3 Quy trình quản lý rủi ro



Hình 2.5: Quy trình quản lý rủi ro

Chương 3: Lập trình

3.1 Ngôn ngữ lập trình

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

3.1.1 Đặc trưng của ngôn ngữ lập trình

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

(1) dễ dịch thiết kế sang chương trình,

(2) có trình biên dịch hiệu quả,

(3) khả chuyển chương trình gốc,

(4) có sẵn công cụ phát triển,

(5) dễ bảo trì.

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

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

Tính khả chuyển chương trình gốc là một đặc trưng ngôn ngữ lập trình có thể được hiểu theo ba cách khác nhau:

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

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

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

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

Tính sẵn có của công cụ phát triển có thể làm ngắn bớt thời gian cần để sinh ra chương trình gốc và có thể cải thiện chất lượng của chương trình. Nhiều ngôn ngữ lập trình có thể cần tới một loạt công cụ kể cả trình biên dịch gỡ lỗi, trợ giúp định dạng chương trình gốc, các tiện nghi soạn thảo có sẵn, các công cụ kiểm soát chương trình gốc, thư viện chương trình con mở rộng trong nhiều lĩnh vực ứng dụng, các trình duyệt, trình biên dịch chéo cho phát triển bộ vi xử lý, khả năng bộ xử lý

macro, công cụ kỹ nghệ ngược và những công cụ khác. Trong thực tế, khái niệm về môi trường phát triển phần mềm tốt (bao hàm cả các công cụ) đã được thừa nhận như nhân tố đóng góp chính cho kỹ nghệ phần mềm thành công.

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

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

3.1.2 Lựa chọn ngôn ngữ lập trình

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

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

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

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

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

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

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

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

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

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

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

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

3.2 Phong cách lập trình

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

3.2.1 Tài liệu chương trình

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

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

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

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

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

1. Một phát biểu về mục đích chỉ rõ chức năng mô đun.
2. Mô tả giao diện bao gồm:
   - Một mẫu cách gọi
   - Mô tả về dữ liệu
   - Danh sách tất cả các mô đun thuộc cấp

3. Thảo luận về dữ liệu thích hợp (như các biến quan trọng và những hạn chế, giới hạn về cách dùng chúng) và các thông tin quan trọng khác.

4. Lịch sử phát triển bao gồm:

- Tên người thiết kế modul (tác giả).
- Tên người xét duyệt và ngày tháng.
- Ngày tháng sửa đổi và mô tả sửa đổi.

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

3.2.2 Khai báo dữ liệu

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

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

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

3.2.3 Xây dựng câu lệnh

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

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

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

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

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

3.2.4 Vào/ra

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

- Làm hợp lệ mọi cái vào.
- Kiểm tra sự tin cậy của các tổ hợp khoản mục vào quan trọng.
- Giữ cho định dạng cái vào đơn giản.

- Dùng các chỉ báo cuối dữ liệu thay vì yêu cầu người dùng xác định "số các khoản mục".

- Giữ cho định dạng cái vào thống nhất khi một ngôn ngữ lập trình có các yêu cầu định dạng nghiêm ngặt.

3.3 Lập trình tránh lỗi

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

i) Sản phẩm của một đặc tả hệ thống chính xác.
ii) Chấp nhận một cách tiếp cận thiết kế phần mềm dựa trên việc bao gói dữ liệu và che dấu thông tin.
iii) Tăng cường duyệt lại trong quá trình phát triển và thẩm định hệ thống phần mềm.
iv) Chấp nhận triết lý chất lượng tổ chức: chất lượng là bánh lái của quá trình phần mềm.
v) Việc lập kế hoạch cẩn thận cho việc thử nghiệm hệ thống để tìm ra các lỗi chưa được phát hiện trong quá trình duyệt lại và để định lượng độ tin cậy của hệ thống.

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

Lập trình có cấu trúc:

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

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

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

i) Các số thực dấu chấm động: các phép toán số thực được làm tròn khiến cho việc so sánh các kết quả số thực, nhất là so sánh bằng giữa hai số thực là không khả thi. Số thực dấu phẩy động có độ chính xác khác nhau khiến cho kết quả phép tính không theo mong muốn. Ví dụ, trong phép tính tính phân chúng ta cần cộng các giá trị nhỏ trước với nhau nếu không chúng sẽ bị làm tròn.
ii) Các con trỏ và bộ nhớ động: con trỏ là các cấu trúc bậc thấp khó quản lý và dễ gây ra các lỗi nghiệm trọng đối với hệ thống. Việc cấp phát và thu hồi bộ nhớ động phức tạp và là một trong các nguyên nhân chính gây lỗi phần mềm.
iii) Song song: lập trình song song đòi hỏi kỹ thuật cao và hiểu biết sâu sắc về hệ thống. Một trong các vấn đề phức tạp của song song là quản lý tương tranh.
iv) Đệ quy.

v) Các ngắt.

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

Phân quyền truy cập dữ liệu:

Nguyên lý an ninh trong quân đội là các cá nhân chỉ được biết các thông tin có liên quan trực tiếp đến nhiện vụ của họ. Khi lập trình người ta cũng tuân theo một nguyên lý tương tự cho việc truy cập dữ liệu hệ thống. Mỗi thành phần chương trình chỉ được phép truy cập đến dữ liệu nào cần thiết để thực hiện chức năng của nó. Ưu điểm của việc che dấu thông tin là các thông tin bị che dấu không thể bị sập đổ (thao tác trái phép) bởi các thành phần chương trình mà được xem rằng không dùng thông tin đó.

Tiến hóa của sự phân quyền truy cập là che dấu thông tin, hay nói chính xác hơn là che dấu cấu trúc thông tin. Khi đó, chúng ta có thể thay đổi cấu trúc thông tin mà không phải thay đổi các thành phần khác có sử dụng thông tin đó.

3.3.1 Lập trình thứ lỗi

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

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

i) Phát hiện lỗi.

ii) Định ra mức độ thiệt hại.

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

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

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

3.3.2 Lập trình phòng thủ

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

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

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

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

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

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

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

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

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

3.4 Lập trình hướng hiệu quả thực hiện

3.4.1 Tính hiệu quả chương trình

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

- Đơn giản hóa các biểu thức số học và lôgic trước khi đi vào lập trình.
- Tính cẩn thận từng chu kỳ lồng nhau để xác định liệu các câu lệnh hay biểu thức có thể được chuyển ra ngoài hay không
- Khi có thể, hãy tránh dùng mảng nhiều chiều
- Khi có thể hãy tránh việc dùng con trỏ và danh sách phức tạp
- Dùng các phép toán số học "nhanh"
- Không trộn lẫn các kiểu dữ liệu, cho dù ngôn ngữ có cho phép điều đó
- Dùng các biểu thức số học và logic bất kì khi nào có thể được

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

3.4.2 Hiệu quả bộ nhớ

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

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

3.4.3 Hiệu quả vào/ra

Các thiết bị vào ra thường có tốc độ chậm hơn rất nhiều so với khả năng tính toán của

máy tính và tốc độ truy cập bộ nhớ trong. Việc tối ưu vào ra có thể làm tăng đáng kể tốc độ thực hiện. Một số hướng dẫn đơn giản để tăng cường hiệu quả vào/ra:

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

3.5 Tổng kết

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

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

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

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

Chương 4: Kiểm nghiệm và bảo trì phần mềm

4.1 Kiểm nghiệm phần mềm

4.1.1 Khái niệm kiểm nghiệm phần mềm

Lý do phải kiểm nghiệm phần mềm

Mặc dù được tự động hoá một phần bởi các công cụ CASE, rất nhiều công đoạn trong quá trình sản xuất phần mềm vẫn được thực hiện bởi con người

- vẫn có khả năng xảy ra lỗi.

Lỗi có thể xảy ra trong tất cả các giai đoạn: phân tích yêu cầu, thiết kế, mã hoá

- Do đó phải kiểm nghiệm chương trình trước khi chính thức sử dụng

Khái niệm kiểm nghiệm phần mềm

Kiểm nghiệm phần mềm là hoạt động thực thi chương trình với mục đích tìm ra lỗi.

Phân loại:

- Kiểm nghiệm **black-box**: kiểm tra các chức năng cụ thể của phần mềm, không quan tâm cấu trúc bên trong, thường áp dụng cho những module lớn.
- Kiểm nghiệm **white-box**: kiểm tra cấu trúc điều khiển bên trong chương trình, thường dùng cho những những module nhỏ.

Mỗi loại kiểm nghiệm có khả năng tìm ra những nhóm lỗi khác nhau → nên kết hợp cả hai

Mục tiêu của kiểm nghiệm phần mềm

Mục tiêu của kiểm nghiệm phần mềm là tìm ra lỗi (nếu có) với chi phí thấp nhất.

Kiểm nghiệm phần mềm giúp:

- Phát hiện được lỗi trong chương trình (nếu có).
- Chứng minh được phần mềm hoạt động đúng như đã thiết kế.

  Chứng minh phần mềm được viết đúng
- Chứng minh được phần mềm đáp ứng yêu cầu của user

  Góp phần chứng minh chất lượng của phần mềm.

Quá trình kiểm nghiệm phần mềm là tốt khi

- Có khả năng tìm ra lỗi cao.
- Không dư thừa.
- Biết chọn lọc: chỉ kiểm nghiệm những phần nào có khả năng tìm ra lỗi đặc trưng.
- Không quá phức tạp cũng không quá đơn giản.

Chú ý: Kiểm nghiệm phần mềm không khẳng định được phần mềm không còn khiếm khuyết, chỉ khẳng định được phần mềm có lỗi và giúp giảm thiểu lỗi.

4.1.2 Các nguyên lý kiểm nghiệm phần mềm

- Việc kiểm nghiệm nên hướng về yêu cầu của khách hàng
- Nên được hoạch định trước một thời gian dài.
- Áp dụng nguyên lý Pareto (nguyên tắc 80-20):
- 80% lỗi có nguyên nhân từ 20% các module → cô lập và kiểm tra những module khả nghi nhất.
- Nên tiến hành từ nhỏ đến lớn: bắt đầu từ những module riêng biệt rồi sau đó tích hợp các module lại.
- Không thể kiểm nghiệm triệt để một phần mềm.
- Nên được thực hiện bởi những đối tượng KHÔNG tham gia vào quá trình phát triển phần mềm.

4.1.3 Phương pháp kiểm nghiệm – Test Case

- Thiết lập các test case – vận hành thử - so sánh kết quả
- Khái niệm test-case
  + Dữ liệu input
  + Thao tác kiểm nghiệm
  + Dữ liệu output hay đáp ứng mong đợi của chương trình
- Test-case cho kiểm nghiệm black-box: chủ yếu dựa vào các yêu cầu cụ thể của chức năng phần mềm.
- Test-case cho kiểm nghiệm white-box: chủ yếu dựa vào cấu trúc điều khiển của phần mềm → vấn đề đặt ra: số lượng test-case cần thiết là quá lớn

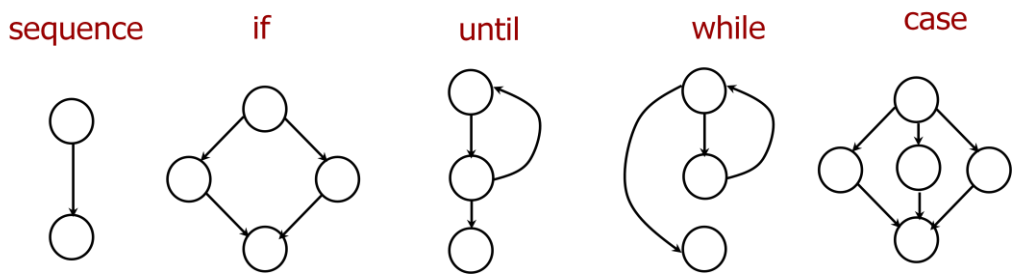
Kiểm nghiệm các đường độc lập cơ bản

Kiểm nghiệm white-box dựa vào cấu trúc điều khiển của thiết kế thủ tục để sinh các test-case với tiêu chí

- Kiểm nghiệm các đường độc lập cơ bản là một trong những phương cách kiểm nghiệm white-box
- Bảo đảm số phép thử là ít nhất đủ để phát hiện các lỗi
- Tất cả các đường thực thi độc lập được thử qua ít nhất một lần
- Thử các điều kiện rẽ nhánh ở cả 2 nhánh true và false
- Thử qua vòng lặp tại biên cũng như bên trong
- Thử qua cấu trúc dữ liệu để đảm bảo tính toàn vẹn của nó

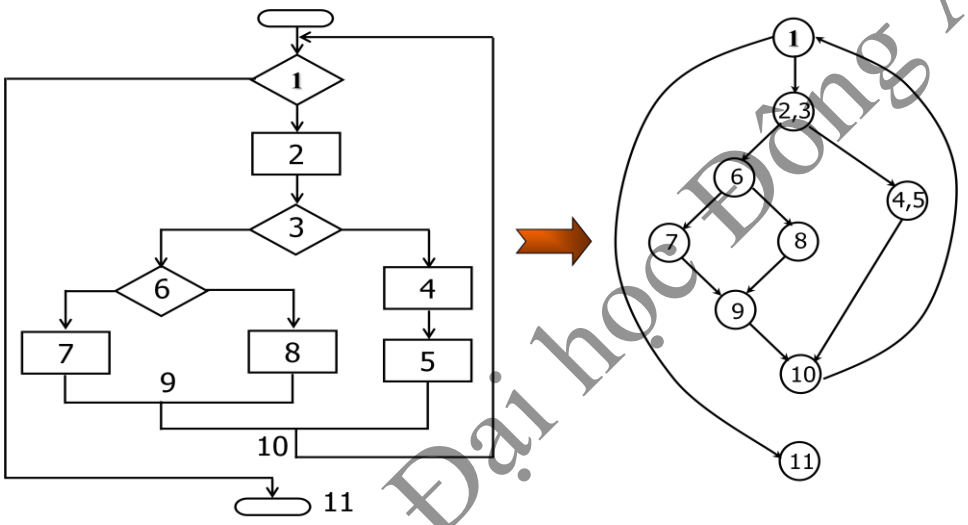
4.1.3.1 **Đồ thị dòng chảy**

- Mỗi node hình tròn biểu diễn một hoặc một vài tác vụ (hơi khác so với lưu đồ thuật giải)
- Cạnh có hướng miêu tả đường thực thi
- Đồ thị dòng chảy được xây dựng từ lưu đồ thuật giải



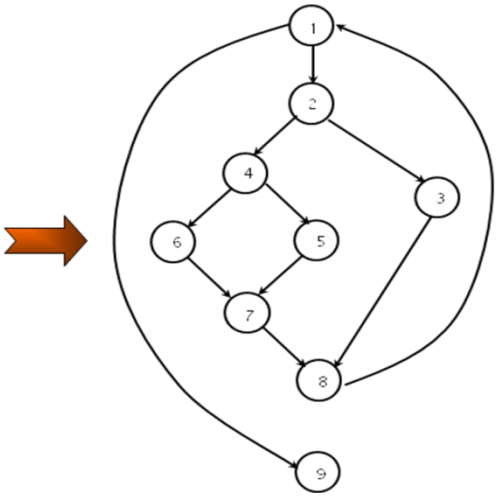
Hình 4.1: Lưu đồ thuật giải

Xây dựng đồ thị dòng chảy



Hình 4.2: Xây dựng đồ thị dòng chảy

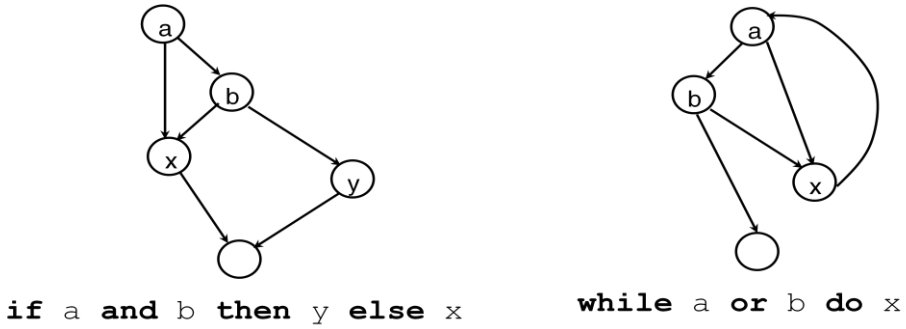
procedure: DoSomething
1: do while x=0
2: if y=0 then
3: z=0;
4: elseif k=0 then
5: z=1;
6: else x=1;
7: endif;
endif;
8: enddo
9: end



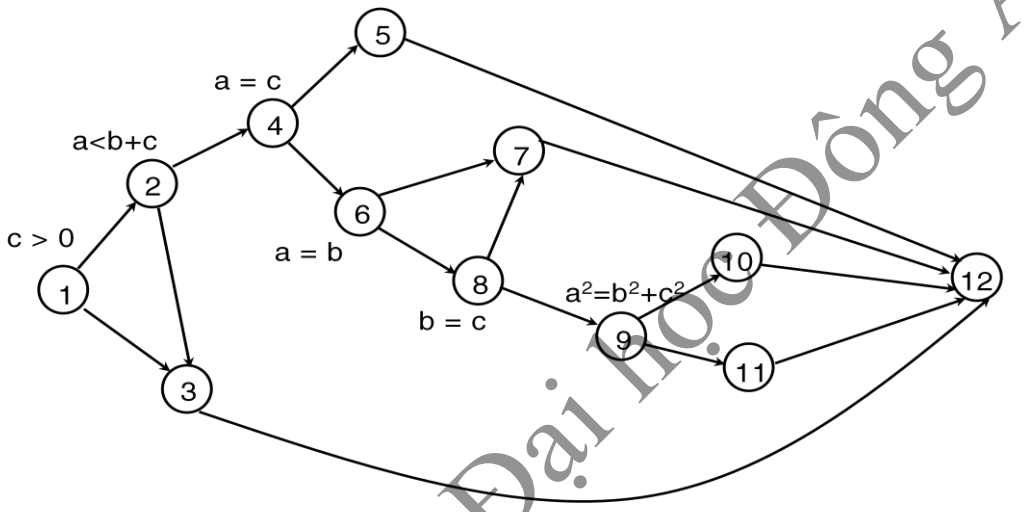
Hình 4.3: Ví dụ xây dựng đồ thị dòng chảy

Phải phân rã tất cả các điều kiện phức trở thành các điều kiện đơn

Mỗi node mô tả một điều kiện đơn được gọi là predicate



Hình 4.4: Phân rã điều kiện phức thành điều kiện đơn



Hình 4.5: Ví dụ Procedure AnalyzeTriangle

Các đường độc lập cơ bản

- Đường thực thi
- Đường thực thi cơ bản
- Các đường thực thi độc lập cơ bản

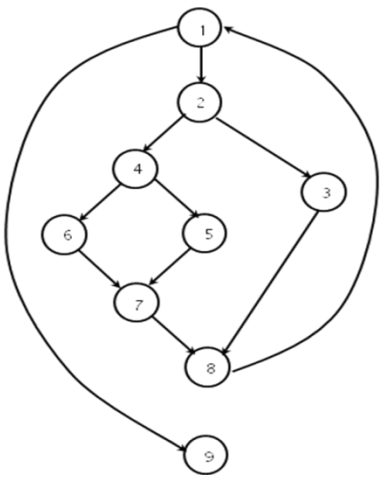
  Từ node bắt đầu đến node kết thúc, các đường thực thi cơ bản được liệt kê theo một thứ tự nào đó để đảm bảo rằng: đường đang liệt kê ít nhất đi qua một cạnh chưa được duyệt qua bởi các đường đã liệt kê trước đó
- Tổng số đường thực thi cơ bản độc lập nhau được tính bằng

  V = P + 1; trong đó **P** là số node phân nhánh (predicate)

Đối với chương trình con DoSomething

- Tổng số đường :

  V = 3 + 1 = 4
- Đường 1: 1-9
- Đường 2: 1-2-3-8-1…
- Đường 3: 1-2-4-5-7-8-1…



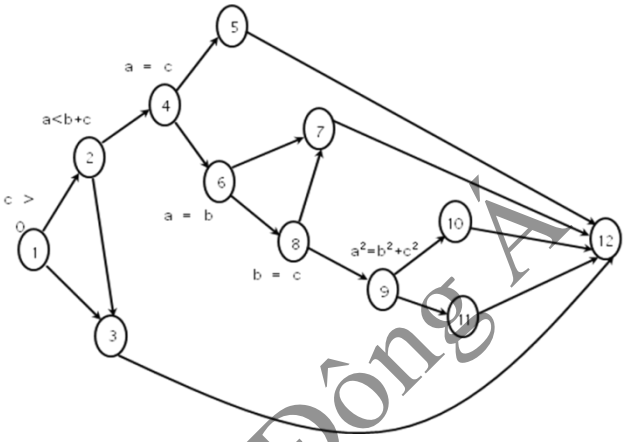
- Đường 4: 1-2-4-6-7-8-1…

Chú ý: dấu 3 chấm (…) mang ý nghĩa "không quan tâm", từ đó có thể đi theo bất kỳ cạnh nào bởi vì các cạnh sau đó đã được duyệt qua rồi

Đối với chương trình con AnalyzeTriangle

- Tổng số đường :

  V = 6 + 1 = 7



- Đường 1: 1-3-12
- Đường 2: 1-2-3-12
- Đường 3: 1-2-4-5-12
- Đường 4: 1-2-4-6-7-12
- Đường 5: 1-2-4-6-8-7-12
- Đường 6: 1-2-4-6-8-9-10-12
- Đường 7: 1-2-4-6-8-9-11-12

4.1.3.2 **Thiết lập các Test Case**

- Thiết lập một test-case cho mỗi đường thực thi cơ bản
- Dựa vào thuật giải để tìm ra một dữ liệu input, sau đó tính ra dữ liệu output hay đáp ứng mong đợi của thuật giải
- Chú ý: có thể không tạo ra được test-case cho một đường thực thi nào đó
- Ví dụ Sinh test-case cho chương trình con AnalyzeTriangle
- Test-case cho đường 1:

  Input: a = 3, b = 2, c = 0

  Output mong đợi: type = "Error"
- Test-case cho đường 2:

  Input: a = 17, b = 5, c = 4

  Output mong đợi: type = "Error"
- Test-case cho đường 3:

  Input: a = 6, b = 6, c = 6

  Output mong đợi: type = "Equilateral"

4.2 Chiến thuật kiểm nghiệm phần mềm

4.2.1 Khái niệm

Chiến thuật kiểm tra phần mềm tích hợp các phương pháp tạo ra test-case trở thành một chuỗi các bước có thứ tự để có thể kiểm nghiệm phần mềm thành công với chi phí thấp.

Bao gồm các công việc

- Lập kế hoạch kiểm nghiệm
- Sinh test-case

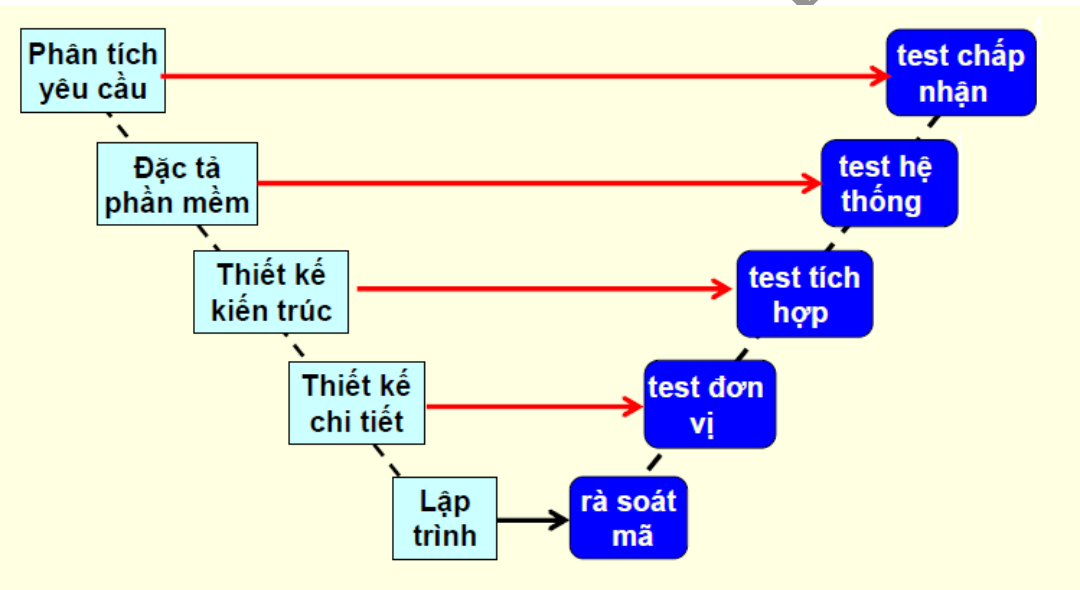
– Thực hiện kiểm nghiệm, thu thập kết qủa và đánh giá

Verification: các hành động để đảm bảo cho phần mềm được hiện thực đúng theo một chức năng cụ thể nào đó → **"Are we building the product right?"**

Validation: các hành động để đảm bảo cho phần mềm được xây dựng theo đúng yêu cầu của khách hàng → **"Are we building the right product?"**

4.2.2 Chiến thuật kiểm nghiệm phần mềm phổ biến

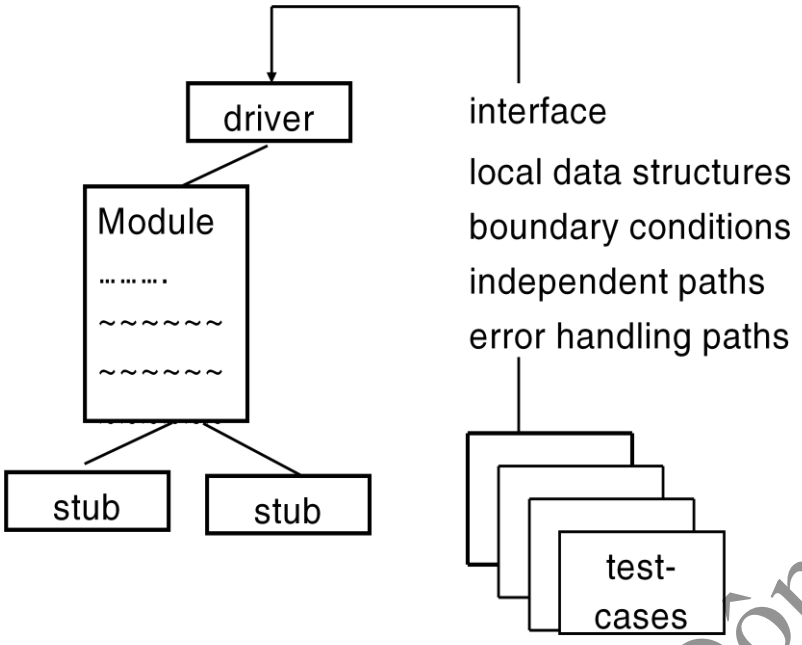
- Bắt đầu tại từng module rồi tích hợp lớn dần đến toàn bộ hệ thống.
- Các kỹ thuật khác nhau được sử dụng thích hợp tại các giai đoạn khác nhau.
- Kiểm nghiệm có thể được tiến hành bởi người phát triển phần mềm, nhưng đối với các dự án lớn thì việc kiểm nghiệm phải được tiến hành bởi một nhóm độc lập.
- Kiểm nghiệm và sửa lỗi là các hoạt động độc lập nhưng việc sửa lỗi phải phù hợp với các chiến thuật kiểm nghiệm.



Hình 4.6: Chiến thuật kiểm nghiệm phần mềm phổ biến

4.2.3 Kiểm nghiệm từng modul – Unit test

- Tiến hành kiểm nghiệm trên từng đơn vị nhỏ nhất của phần mềm, đó là module mã nguồn, sau khi đã thiết kế, mã hoá và biên dịch thành công
- Thường dùng kỹ thuật kiểm nghiệm white-box
- Có thể tiến hành kiểm nghiệm cùng lúc nhiều module.
- Một số vấn đề trong việc xây dựng các test case
  - + Test case nào?
  - + Dữ liệu đầu vào và đầu ra có tử đâu?
  - + Tính độc lập/phụ thuộc hoạt động của các module



Hình 4.7: Kiểm nghiệm module

Mỗi module mã nguồn không phải là một chương trình hoàn chỉnh và đôi khi phải gọi các module chưa được kiểm nghiệm khác → có thể phải thiết lập **driver** và/hoặc **stub**: phí tổn khá lớn (70%)

Driver là một chương trình chính có nhiệm vụ nhận dữ liệu kiểm nghiệm, chuyển dữ liệu đó xuống cho module để kiểm tra và in ra các kết quả kiểm tra tương ứng.

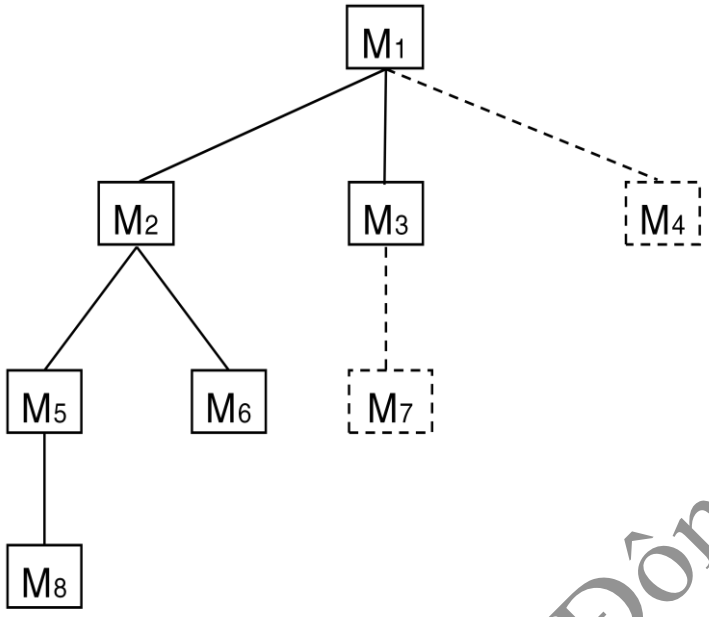
Stub thay thế các module được gọi bởi module đang kiểm tra.

- Làm thế nào để giảm các chi phí tạo driver hay stub

4.2.4 Kiểm nghiệm tích hợp

- Từng module mã nguồn đã hoạt động đúng. Liệu khi kết hợp chúng lại thành một nhóm lớn chúng có hoạt động đúng không ?
- Phải tiến hành kiểm nghiệm tích hợp để phát hiện lỗi liên quan đến giao tiếp giữa các module.
- Tránh tích hợp kiểu big-bang: tất cả các module được kết hợp lại, và toàn bộ chương trình sẽ được kiểm nghiệm một lúc
- Nên tích hợp tăng dần: từ trên xuống hoặc từ dưới lên

4.2.4.1 **Kiểm nghiệm tích hợp từ trên xuống**

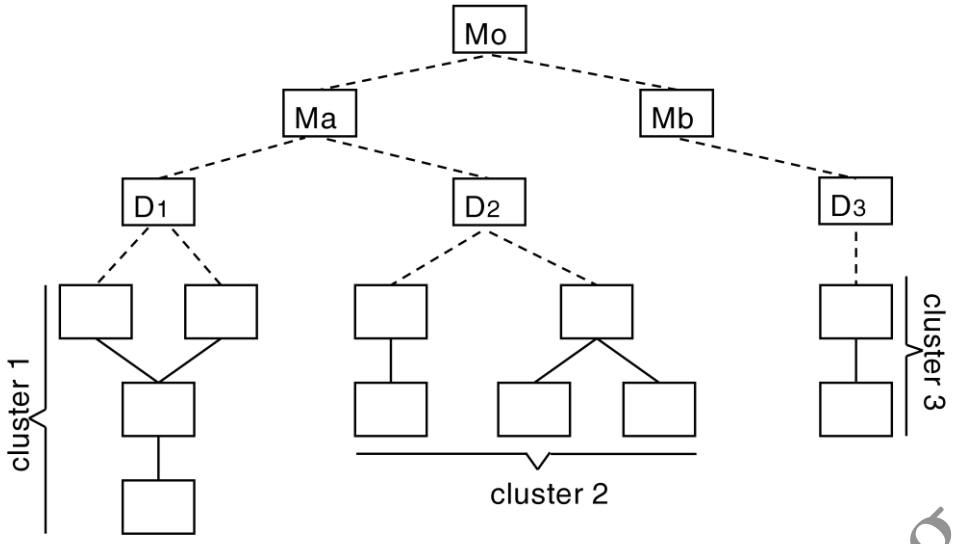


Hình 4.8: Kiểm nghiệm tích hợp top-down

- Module chính được dùng như là driver, và stub được thay thế bởi các module con trực tiếp của của module chính này.
- Tuỳ thuộc vào cách tích hợp theo chiều sâu (depth-first) hoặc chiều ngang(breath-first), mỗi stub con được thay thế một lần bởi module tương ứng đã kiểm nghiệm.
- Tiến hành kiểm nghiệm khi có sự thay thế mới
- Tiến hành kiểm nghiệm hồi quy để phát hiện các lỗi khác trong từng module
- Tích hợp kiểu từ trên xuống theo hình thức depth-first
- Tiết kiệm được chi phí tạo các driver

4.2.4.2 **Kiểm nghiệm tích hợp từ dưới lên**

- Các module mức thấp nhất được kết hợp thành các nhóm thể hiện một chức năng con đặc biệt của phần mềm.
- Một driver được tạo ra để thao tác các test-case
- Các module được kiểm nghiệm theo từng nhóm (Cluster): là nhóm các module mà module phía trên cần đến khi kiểm nghiệm
- Driver được bỏ đi và các nhóm module được kết hợp dần lên phía trên trong sơ đồ phân cấp của chương trình.
- Tiết kiệm được chi phí tạo các stub



Hình 4.9: Kiểm nghiệm tích hợp bottom-up

4.2.4.3 **Kiểm nghiệm hồi quy**

- Việc kết hợp các module lại với nhau có thể ảnh hưởng đến vòng lặp điều khiển, cấu trúc dữ liệu hay I/O chia sẻ trong một số module
- Điều đó làm lộ ra một số lỗi không thể phát hiện được khi tiến hành kiểm nghiệm theo đơn vị

Phải kiểm nghiệm hồi quy khi tích hợp

- Kiểm nghiệm hồi quy có thể được tiến hành thủ công bằng cách thực hiện lại các test-case đã tạo ra. Hoặc có thể dùng một công cụ capture-playback để thực hiện tự động

4.2.4.4 **Kiểm nghiệm tính năng**

Kiểm nghiệm tính năng hiểu theo cách đơn giản nhất là: kiểm tra các chức năng của phần mềm đáp ứng được nhu cầu của khách hàng đã được xác định trong văn bản đặc tả yêu cầu của phần mềm

Áp dụng kỹ thuật black-box

Kiểm nghiệm tính năng bao gồm

- Xem xét lại cấu hình phần mềm theo lược đồ triển khai
- Kiểm nghiệm alpha
- Kiểm nghiệm beta

Kiểm nghiệm alpha

- Được tiến hành ngay tại nơi sản xuất phần mềm.
- Nhà phát triển phần mềm sẽ quan sát người sử dụng dùng sản phẩm và ghi nhận lại những lỗi phát sinh để sửa chữa.

Kiểm nghiệm beta

- Phần mềm được kiểm tra bên ngoài phạm vi của đơn vị sản xuất.

- Khách hành trực tiếp sử dụng và ghi nhận lỗi để báo lại cho nhà phát triển sửa chữa.

4.2.4.5 **Kiểm nghiệm hướng đối tượng**

Về cơ bản chiến thuật kiểm nghiệm hướng đối tượng cũng theo thứ tự giống như kiểm nghiệm cổ điển:

Hình 4.10: Kiểm nghiệm hướng đối tượng

Không thể tách rời từng tác vụ của đối tượng/lớp để kiểm nghiệm

- Tác vụ được đóng bao trong lớp
- Các lớp con có thể override một tác vụ nào đó

Kiểm nghiệm đơn vị hướng đối tượng tập trung vào các lớp → kiểm nghiệm hành vi của lớp

4.2.4.6 **Kiểm nghiệm tích hợp hướng đối tượng**

Khái niệm sơ đồ phân cấp không còn nhiều ý nghĩa trong chương trình hướng đối tượng → kiểm nghiệm tích hợp theo cách khác

Hai hình thức kiểm nghiệm tích hợp hướng đối tượng

- Kiểm nghiệm trên cơ sở thread: tích hợp các lớp tạo thành một thread để phục vụ cho một input nào đó của chương trình
- Kiểm nghiệm trên cơ sở sử dụng: các lớp client sẽ được tích hợp để sử dụng dịch vụ nào đó cung cấp bởi các lớp server

4.2.4.7 **Kiểm nghiệm theo kịch bản**

Dựa vào các use-case để soạn ra các kịch bản

Ví dụ: một kịch bản cho hệ thống đăng ký môn học qua WEB

1. Login với username = “e59306547”, password = “6547”
2. Chọn chức năng đăng ký môn học
3. Chọn 5 nhóm môn học của 5 môn: CNPM, AI, XLTHS, PTTK, XLSS trong đó có 2 nhóm trùng thời khoá biểu
4. Nhấn nút Submit

➢ Chương trình phải báo lỗi và liệt kê 2 nhóm bị trùng thời khoá biểu

4.3 Bảo trì phần mềm

4.3.1 Khái niệm và phân loại bảo trì

Bảo trì là công việc tu sửa, thay đổi phần mềm đã được phát triển (chương trình, dữ liệu, JCL, các loại tư liệu đặc tả, . . .) theo những lý do nào đó

Các hình thái bảo trì: bảo trì để

- Tu chỉnh
- Thích hợp
- Cải tiến
- Phòng ngừa

Bảo trì để tu sửa

Là bảo trì khắc phục những khiếm khuyết có trong phần mềm.

Một số nguyên nhân điển hình:

- Kỹ sư phần mềm và khách hiểu nhầm nhau
- Lỗi tiềm ẩn của phần mềm do sơ ý của lập trình hoặc khi kiểm thử chưa bao quát hết
- Vấn đề tính năng của phần mềm: không đáp ứng được yêu cầu về bộ nhớ, tệp, . . . Thiết kế sai, biên tập sai . . .
- Thiếu chuẩn hóa trong phát triển phần mềm (trước đó)

Kỹ nghệ ngược (Reverse Engineering): dò lại thiết kế để tu sửa

Những lưu ý

- Mức trừu tượng
- Tính đầy đủ
- Tính tương tác
- Tính định hướng

Bảo trì để thích hợp

Là tu chỉnh phần mềm theo thay đổi của môi trường bên ngoài nhằm duy trì và quản lý phần mềm theo vòng đời của nó.

Thay đổi phần mềm thích nghi với môi trường: công nghệ phần cứng, môi trường phần mềm.

Những nguyên nhân chính:

- Thay đổi về phần cứng (ngoại vi, máy chủ,. . .)
- Thay đổi về phần mềm (môi trường): đổi OS
- Thay đổi cấu trúc tệp hoặc mở rộng CSDL

Bảo trì để cải tiến

Là việc tu chỉnh hệ phần mềm theo các yêu cầu ngày càng hoàn thiện hơn, đầy đủ hơn, hợp lý hơn.

Những nguyên nhân chính:

- Do muốn nâng cao hiệu suất nên thường hay cải tiến phương thức truy cập tệp.
- Mở rộng thêm chức năng mới cho hệ thống.
- Cải tiến quản lý kéo theo cải tiến tư liệu vận hành và trình tự công việc.
- Thay đổi người dùng hoặc thay đổi thao tác.

Còn gọi là tái kỹ nghệ (re-engineering).

Mục đích: đưa ra một thiết kế cùng chức năng nhưng có chất lượng cao hơn.

Các bước thực hiện:

- Xây dựng lưu đồ phần mềm
- Suy dẫn ra biểu thức Bun cho từng dãy xử lý
- Biên dịch bảng chân lí
- Tái cấu trúc phần mềm

Bảo trì để phòng ngừa

Là công việc tu chỉnh chương trình có tính đến tương lai của phần mềm đó sẽ mở rộng và thay đổi như thế nào.

Thực ra trong khi thiết kế phần mềm đã phải tính đến tính mở rộng của nó, nên thực tế ít khi ta gặp bảo trì phòng ngừa nếu như phần mềm được thiết kế tốt .

Mục đích:

- Sửa đổi để thích hợp với yêu cầu thay đổi sẽ có của người dùng
- Thực hiện những thay đổi trên thiết kế không tường minh
- Hiểu hoạt động bên trong chương trình
- Thiết kế / lập trình lại
- Sử dụng công cụ CASE

4.3.2 Trình tự nghiệp vụ bảo trì

Quy trình bảo trì là gì ? Đó là quá trình trong vòng đời của phần mềm, cũng tuân theo các pha phân tích, thiết kế, phát triển và kiểm thử từ khi phát sinh vấn đề cho đến khi giải quyết xong.

Thao tác bảo trì: Gồm 2 loại

- Tu chỉnh cái đã có (loại 1)
- Thêm cái mới (loại 2)

Hiểu phần mềm đã có

- Theo tài liệu nắm chắc các chức năng
- Theo tài liệu chi tiết hãy nắm vững đặc tả chi tiết, điều kiện kiểm thử,...
- Dò đọc chương trình nguồn, hiểu trình tự xử lý chi tiết của hệ thống

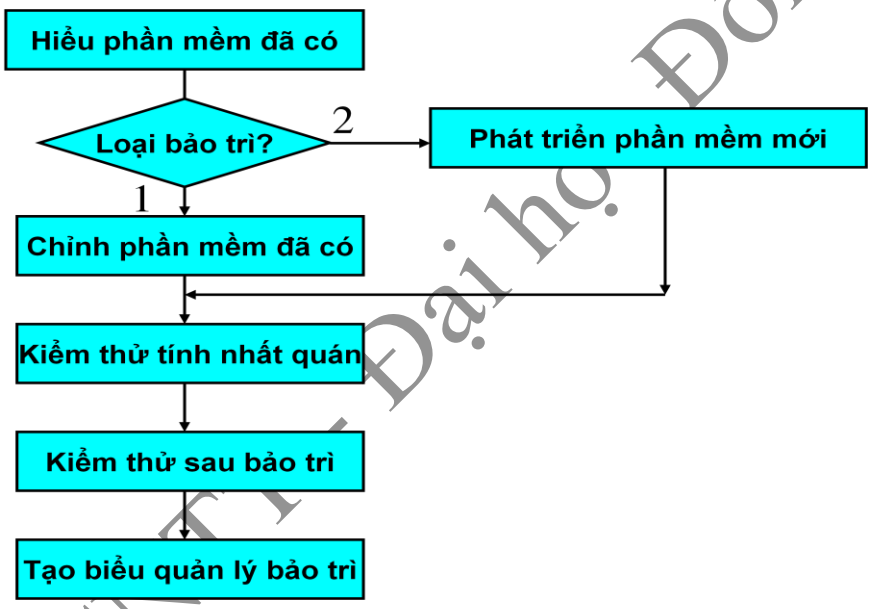
3 việc trên đều là công việc thực thi trên bàn

Tu sửa phần mềm đã có

- Bảo trì chương trình nguồn, tạo các môđun mới và dịch lại
- Thực hiện kiểm thử unit và tu chỉnh những mục liên quan có trong tư liệu đặc tả
- Chú ý theo sát tác động của môđun được sửa đến các thành phần khác trong hệ thống

Phát triển phần mềm mới

- Khi thêm chức năng mới phải phát triển chương trình cho phù hợp với yêu cầu
- Cần tiến hành từ thiết kế, lập trình, gỡ lỗi và kiểm thử unit
- Phản ảnh vào giao diện của phần mềm (thông báo, phiên bản,...)



Hình 4.11: Sơ đồ bảo trì

Lập biểu quản lý bảo trì

Cần quản lý tình trạng bảo trì

Lập biểu quản lý tình trạng bảo trì

- Ngày tháng, giờ
- Nguyên nhân
- Tóm tắt cách khắc phục
- Chi tiết khắc phục, hiệu ứng làn sóng
- Người làm bảo trì
- Số công

Những vấn đề lưu ý khi bảo trì phần mềm

Phương pháp cải tiến thao tác bảo trì:

- Sáng kiến trong quy trình phát triển phần mềm
- Sáng kiến trong quy trình bảo trì phần mềm
- Phát triển những kỹ thuật mới cho bảo trì

Sáng kiến trong quy trình phát triển phần mềm

1. Chuẩn hóa mọi khâu trong phát triển phần mềm
2. Người bảo trì chủ chốt tham gia vào giai đoạn phân tích và thiết kế
3. Thiết kế để dễ bảo trì

Sáng kiến trong quy trình bảo trì phần mềm

1. Sử dụng các công cụ hỗ trợ phát triển phần mềm
2. Chuẩn hóa thao tác bảo trì và thiết bị môi trường bảo trì
3. Lưu lại những thông tin sử bảo trì
4. Dự án nên cử một người chủ chốt của mình làm công việc bảo trì sau khi dự án kết thưc giai đoạn phát triển

Phát triển những kỹ thuật mới cho bảo trì

- Công cụ phần mềm hỗ trợ bảo trì
- Cơ sở dữ liệu cho bảo trì
- Quản lý tài liệu, quản lý dữ liệu, quản lý chương trình nguồn, quản lý dữ liệu thử, quản lý sử bảo trì
- Trạm bảo trì tính năng cao trong hệ thống mạng lưới bảo trì với máy chủ thông minh

Tài liệu tham khảo

1. Bài giảng Công nghệ Phần mềm – Nguyễn Cao Trí, Đại học Bách Khoa Tp Hồ Chí Minh
2. Bài giảng Công nghệ Phần mềm – Bộ môn Công nghệ Phần mềm, Viện Công nghệ Thông tin và Truyền thông, Đại học Bách Khoa Hà Nội.
3. Bài giảng Công nghệ Phần mềm – Nguyễn Việt Hà, Đại học Bách Khoa Hà Nội
4. Bài giảng Công nghệ Phần mềm – Nguyễn Văn Vy, Đại học Công Nghệ, Đại học Quốc gia Hà Nội
5. Giáo trình Phân tích và thiết kế hệ thống thông tin với UML – Ts Dương Kiều Hoa, Tôn Thất Hoà An
6. Giáo trình Phân tích thiết kế hệ thống thông tin – Học viện Công nghệ Bưu chính Viễn thông.
7. Introduction to Software Engineering – Ronald J. Leach, CRC Press
8. Software Engineering – Ian Sommerville, Fifth edition
9. Software Engineering – A practitioner's approach, R.S.Pressman, McGraw-Hill, 1997
10. OMG Unified Modeling Language Specification, version 1.3, Object Management Group (www.omg.org), 1999
11. UML Toolkit, Hans – Erik Eriksson & Magnus Penker, 1998
12. Object–Oriented Software Engineering, A Use-Case Driven Approach, I. Jacobson, ACM Press/Addison-Wesley, 1992
13. Object– Oriented Analysis and Design with Applications, G. Booch, The Benjamin Cummings Publishing Company, 1994
14. Website: www.dayhoctructuyen.com, http://edu.net.vn