Mô hình xây dựng cơ sở dữ liệu

Lời nói đầu

Trong hơn ba chục năm qua người ta đã chứng kiến sự lớn mạnh về số lượng cũng như mức độ quan trọng trong việc ứng dụng cơ sở dữ liệu. Các cơ sở dữ liệu là thành phần cơ bản trong hệ thống thông tin, dùng trong cả máy lớn lẫn máy nhỏ. Việc thiết kế cơ sở dữ liệu được coi là hoạt động thông dụng, có hiệu quả đối với cán bộ chuyên môn lẫn người dùng không chuyên.

Từ cuối những năm 60, khi các cơ sở dữ liệu xuất hiện lần đầu trên thị trường phần mềm, người thiết kế xoay sở như thợ thủ công, họ dùng sơ đồ khối, các cấu trúc bản ghi và thiết kế cơ sở dữ liệu thường bị lẫn với việc cài đặt cơ sở dữ liệu, tình huống này đã thay đổi, các phương pháp và các mô hình thiết kế cơ sở dữ liệu đã tiến hoá song song với quá trình công nghệ hệ thống cơ sở dữ liệu. Khi làm việc với cơ sở dữ liệu quan hệ người ta sử dụng các ngôn ngữ hỏi mạnh, công cụ phát triển ứng dụng và giao diện người dùng thân thiện. Công nghệ cơ sở dữ liệu đã có nền lý thuyết, gồm lý thuyết quan hệ về dữ liệu, xử lý câu hỏi và tối ưu, điều khiển tương tranh, quản lý giao tác và khôi phục sai xót...

Khi công nghệ cơ sở dữ liệu đã tiến lên, công nghệ thiết kế và các kỹ thuật cũng phát triển.Người ta đã chia quá trình thành các pha, đặt mục đích chính cho mỗi pha, và các kỹ thuật đạt được các pha đó.

Tuy nhiên các phương pháp luận thiết kế cơ sở dữ liệu không thông dụng; hầu hết các tổ chức và các nhà thiết kế cá nhân ít tuân theo các phương pháp luận thiết kế, và điều đó cũng dễ dẫn đến sai lầm trong việc phát triển các hệ thống thông tin. Bên cạnh việc thiếu tiếp cận cấu trúc về thiết kế cơ sở dữ liệu ,thời gian và tài nguyên cần cho đề án cơ sở dữ liệu thường bị đánh giá thấp. Do vậy các cơ sở dữ liệu phát triển là không hoàn thiện và không hiệu quả so với nhu cầu của các ứng dụng; các tư liệu không đầy đủ và bảo trì còn nhiều vấn đề vướng mắc.

Nhiều bài toán đã không rõ ràng và không trong sáng về bản chất chính xác của dữ liệu tại mức khái niệm và mức trìu tượng. Trong nhiều trường hợp, dữ liệu được mô tả từ khi bắt đầu đề án trong cấu trúc dữ liệu lưu trữ; chứ không tập trung vào việc hiểu thuộc tính có cấu trúc của dữ liệu. Các dữ liệu cần độc lập với việc cài đặt.

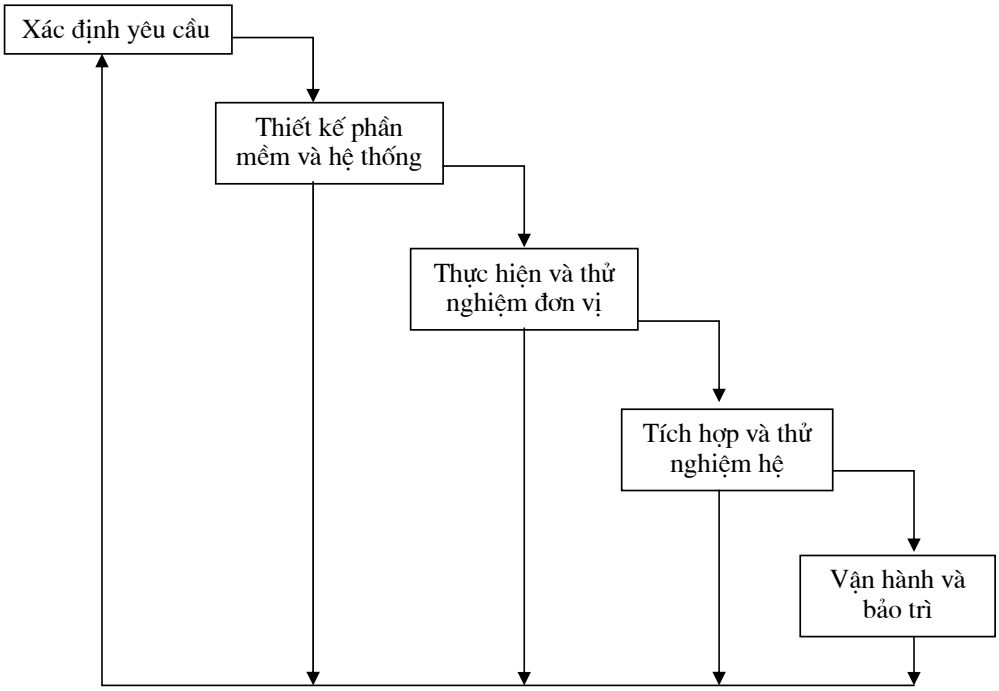
Vì vậy để xây dựng một hệ thống thông tin cần có các mô hình thiết kế cụ thể để tránh những thiếu sót đã nêu trên. Một trong những mô hình được áp dụng rộng rãi và có hiệu quả là mô hình thác nước. Ngày nay, nhiều nghiên cứu đã cho ra đời nhiều mô hình tiến bộ hơn, có thể xây dựng được các hệ thống thông tin lớn như: mô hình phát triển tiến hoá, mô hình xoắn ốc của Bochur... Tuy nhiên mô hình thác nước là một mô hình đơn giản và thích hợp với những bài toán không quá lớn.
Trong khuôn khổ của một buổi **cenema,** tôi muốn giới thiệu qua với các bạn các khái niệm, các bước cơ bản để xây dựng một hệ thống thông tin bằng mô hình thác nước, những thuận lợi và khó khăn trong từng bước xây dựng, cũng như những tiến bộ và hạn chế khi sử dụng mô hình này để xây dựng hệ thống thông tin.

Vì thời gian và khả năng có hạn nên tôi không thể mô tả một cách chi tiết từng bước trong quá trình thiết kế. Rất mong nhận được những ý kiến đóng góp ở các bạn để chương trình ngày càng hoàn thiện hơn. Xin cám ơn tất cả mọi người.

Mô hình thác nước

Mô hình đầu tiên trong việc xử lý và phát triển phần mềm bắt nguồn từ những công nghệ xử lý khác nhau. Mô hình này đã được các nhà hoạch định dự án chấp thuận. Nó bao gồm các tiến trình phát triển rõ ràng và cụ thể, kế cận nhau giống như thác nước. Vì vậy nó có tên gọi là mô hình thác nước

Có 5 tiến trình trong một chu trình của mô hình thác nước:



1. Xác định yêu cầu:

Để xây dựng được một phần mềm mang tính thực tiễn cao, trước hết cần phải trả lời được câu hỏi phần mềm được xây dựng làm gì, ứng dụng vào lĩnh vực nào? Hay ngược lại, các nhà sản xuất phần mềm cần phải biết được các nhà phát triển và người sử dụng muốn gì trong sản phẩm của họ. Nói cách khác cần phải có sự trao đổi giữa các nhà sản xuất với các nhà phát triển và người sử dụng, để từ đó rút ra được một bản danh sách các yêu cầu một cách đầy đủ và chi tiết. Đây là một bước công phu và nhiều khó khăn.

2. Thiết kế phần mềm và hệ thống:

Quy trình thiết kế hệ thống chia các yêu cầu thành hệ thống phần mềmvà phần cứng. Nó đượcc bao trùm bởi một cấu trúc hệ thống. Thiết kế phần mềm liên quan đến việc sử dụng các ngôn ngữ lập trình để viết các hàm phần mềm theo hệ thốngcác yêu cầu từ đó có thể dịch sang một hay nhiều chương trình có tính thực thi.

3. Thực hiện và thử nghiệm đơn vị:

Trong bước này, việc thiết kế phần mềm thực chất là thiết lập một chương trình hay các Modun chương trình riêng lẻ. Việc thử nghiệm từng modun chương trình liên quan đến việc thẩm tra mỗi modun bằng cách đưa vào các chi tiết trong yêu cầu kỹ thuật của nó.

4. Tích hợp và chạy thử cả hệ thống:
Dừng modun chương trình hay các chương trình đơn lẻ được tích hợp lại với nhau và chạy thử như một hệ thống hoàn chỉnh để đảm bảo rằng các yêu cầu phần mềm đã được đáp ứng. Sau khi kiểm thử, hệ thống phần mềm được giao lại cho khách hàng.

5. Vận hành và bảo trì:

Thông thường, đây là bước có chu kì dài nhất, hệ thống đã hoàn tất và được đưa vào sử dụng. Bảo trì là việc sửa chữa lại các sai lầm không được phát hiện sớm ở các bước trước đó, phát hiện các tính năng của từng modun trong hệ thống và nâng cao khả năng phục vụ hệ thống khi các yêu cầu mới được phát triển thêm.

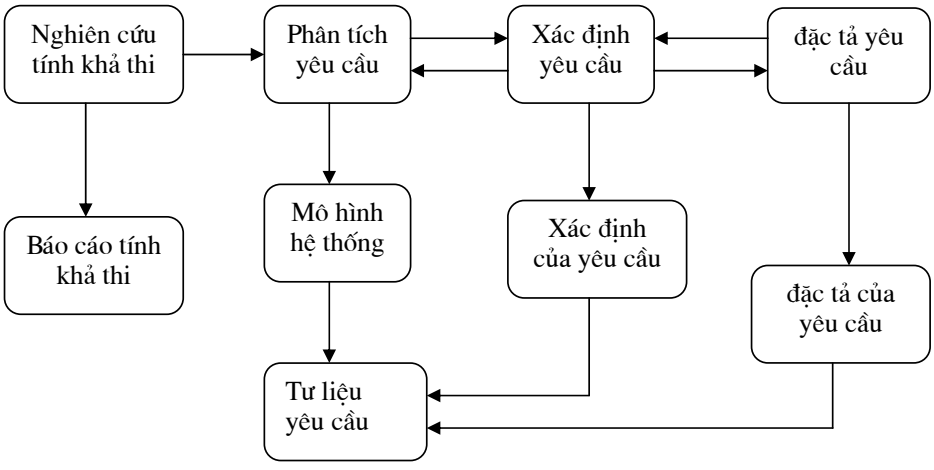
Trong thực tế, các bước này không độc lập với nhau chúng đan xen lẫn nhau và bổ xung thông tin cho nhau trong quá trình thiết kế hệ thống. Xử lí phần mềm không phải là một dạng đường đơn giản mà nó liên quan đến một hệ thống lặp lại tuần tự của các hoạt động phát triển. Trong giai đoạn cuối cùng, phần mềm được đưa vào sử dụng, các lỗi và những thiếu sót trong yêu cầu phần mềm được phát hiện, cần phải định nghĩa thêm một số chức năng mới. Việc sửa đổi trở nên cần thiết để những phần mềm còn sai sót trở nên hữu ích hơn. Thực hiện các cải tiến có thể liên quan đến việc lặp lại các bước trước đó.

Đáng tiếc rằng, một mô hình bao gồm nhiều bước lặp lại tuần tự khó có thể nhận ra một cách rõ ràng và nhanh chóng các sai lầm và thiếu sót cho việc lập kế hoạch và báo cáo. Tuy nhiên sau một số ít bước lặp lại, công việc này có thể được tiến hành một cách dễ dàng hơn và có thể bổ xung thông tin vào bước đặc tả yêu cầu. Sự cố định sớm các yêu cầu có thể làm cho hệ thống không thực hiện được các công việc mà người dùng cần đến. Nó cũng có thể dẫn đến việc xây dựng các hệ thống không tốt, có thể dẫn đến tình trạng hệ thống bị phá vỡ khi thực hiện.

Hạn chế của mô hình thác nước là không linh hoạt trongviệc phân chia các đối tượng thành các bước riêng biệt. Đôi khi việc cứu vãn hệ thống là không thể khi nó không đáp ứng được các yêu cầu mới của khách hàng. Tuy nhiên, mô hình thác nước phản ánh công nghệ theo lối tư duy tự nhiên quen thuộc và nó thích hợp với các hệ thống không quá lớn. Do vậy nó vẫn còn được sử dụng rộng rãi.

Quá trình xây dựng yêu cầu

Muốn xây dựng được hệ thống trước tiên ta phải khảo sát được các yêu cầu của hệ thống. Thông thường để xây dựng một yêu cầu ta cần phải thực hiện các bước như sơ đồ sau:



Yêu cầu là những gì được phát biểu ra, thường là văn bản những gì mà khách hàng muốn có. Trên thực tế giữa yêu cầu thật sự với những gì được phát biểu có sự khác nhau. Nhu cầu là những gì mà người sử dụng muốn, nhưng yêu cầu đôi khi không thoả mãn hết được các nhu cầu.

Yêu cầu hệ thống và mục tiêu của hệ thốngthường được đưa ra bằng những khái niệm mang tính định lượng có thể được kiểm chứng.

Khi có một kí kết hợp đồng người ta thường đưa ra yêu cầu hệ thống chứ không đưa ra nhu cầu và mục tiêu của hệ thống. Bởi hai yếu tố này có thể được kiểm chứng đánh giá. Công việc phân tích và nắm bắt nhu cầu luôn luôn gặp khó khăn. Bởi những những khách hàng nắm bắt được các kiến thức chuyên ngành nhưng khôngg biết chuyên môn tin học nên không thể đưa ra được yêu cầu thích hợp cho hệ thống và ngược lại người làm hệ thống lại không nắm được các kiến thức chuyên ngành. Do vậy, các đặc tả ban đầu thường không tốt.

Có bốn bước quan trọng trong quá trình xây dựng yêu cầu:

- Nghiên cứu tính khả thi :

Công việc ước lượng đánh giá được bắt đầu từ việc nhận biết nhu cầu của người sử dụng. Có thể các phần mềm hiện tại đã làm hài lòng người sử dụng, việc nghiên cứu tính khả thi sẽ quyết định hệ thống được đề xuất có hiệu quả cao hơn theo quan điểm của người kinh doanh hay không và nó có thể được cung cấp ngân sách để tiến hành hay không. Nghiên cứu tính khả thi sẽ làm giảm thời gian và chí phí cho việc sản xuất phần mềm. Kết quả sẽ thông tin đến người ra quyết định với bản phân tích chi tiết hơn.

- Phân tích yêu cầu: Đây là một quá trình xác định các yêu cầu hệ thống thông qua việc theo dõi sự tồn tại của hệ thống, thảo luận với người sử dụng và người ra quyết định. Qui trình này có thể liên quan đến sự phát triển của một hay nhiều mô hình hệ thống khác nhau. Quá trình phân tích giúp ta hiểu được những yêu cầu mà hệ thống đã nêu ra. Những nguyên mẫu hệ thống cũng có thể được phát triển để giúp ta hiểu rõ hơn các yêu cầu.

- Xác định yêu cầu: Qui trình này tập hợp các yêu cầu đã thu thập được thành một tài liệu. Nó phản ánh chính xác điều mà khách hàng muốn, được thể hiện bằng ngôn ngữ tự nhiên cùng với các bảng biểu thể hiện được thông tin chung nhất với người sử dụng. Tài liệu là thông tin do phía khách hàng cung cấp, tài liệu này không dùng để kí kết hợp đồng.

- Đặc tả yêu cầu: Các yêu cầu thường có cấu trúc rõ ràng, tỉ mỉ. Đây là cơ sở cho phía khách hàng và nhà cung cấp kí kết hợp đồng. Việc tạo ra tài liệu này thường được thực hiện song song với việc thiết kế mức cao. Việc thiết kế và các hoạt động yêu cầu ảnh hưởng lẫn nhau trong quá trình thực hiện. Trong quá trình tạo ra tài liệu này các lỗi của bước xác định yêu cầu sẽ được khám phá. Đây là mức mô tả trừu tượng các dịch vụ của hệ thống được sử dụng làm cơ sở cho việc thiết kế và thực hiện phần mềm.

Trên đây là 4 bước quan trọng trong quá trình xây dựng yêu cầu. Bây giờ ta sẽ đi xâu phân tích một số bước cụ thể:

I/ Phân tích yêu cầu:

Sau khi nghiên cứu tính khả thi, bước quan trọng đầu tiên là phân tích hoặc suy luận các yêu cầu. Các chuyên gia phát triển phần mềm làm việc với các khách hàng và người sử dụng cuối

cùng để tìm ra các lĩnh vực ứng dụng mà cấp hệ thống phần mềm phải đáp ứng, việc thực hiện các yêu cầu hệ thống do phần mềm và các yếu tố khác quy định, chẳng hạn như phần cứng, các thiết bị ngoại vi ...

Phân tích yêu cầu là một bước quan trọng, tính khả thi của hệ thống sau này phụ thuộc nhiều vào quá trình trao đổi giữa nhu cầu của khách hàng và nhà cung cấp với các công việc được tự động hoá. Nếu việc phân tích không đáp ứng nhu cầu thực của khách hàng, hệ thống sau khi hình thành sẽ không đáp ứng được các yêu cầu.

Phân tích yêu cầu có thể còn liên quan đến sự đa dạng của các cấp bậc chức vụ khác nhau của các nhân viên trong cùng một tổ chức. Hay nói cách khác, vấn đề bảo mật gây ra rất nhiều khó khăn trong phân tích yêu cầu. Điều này ảnh hưởng tới người tác động cuối cùng vào hệ thống, người tổ chức và thiết đặt hệ thống. Các nhà đầu tư có ảnh hưởng trực tiếp hoặc gián tiếp tới những yêu cầu của hệ thống.

Bước phân tích yêu cầu là khó bởi một số lý do sau:

- Trong hầu hết các trường hợp, các nhà đầu tư không biết thực sự họ muốn gì ở hệ thống máy tính. Thậm chí khi họ có một định hướng rõ ràng họ mới cóthể biết hệ thống cần phải làm gì, và rất khó khăn để kết hợp các yêu cầu lại với nhau. Họ có thể làm sai lệch nhu cầu bởi họ không biết giá trị của câu hỏi.

- Các nhà đầu tư trong một hệ thống thường bộc lộ các yêu cầu với kiến thức ngầm định trong công việc của họ. Còn người kỹ sư không có nhiều kinh nghiệm trong các lĩnh vực của khách hàng, do vậy cần phải hiểu được những yêu cầu và dịch chúng sang một dạng tài liệu chấp nhận được.

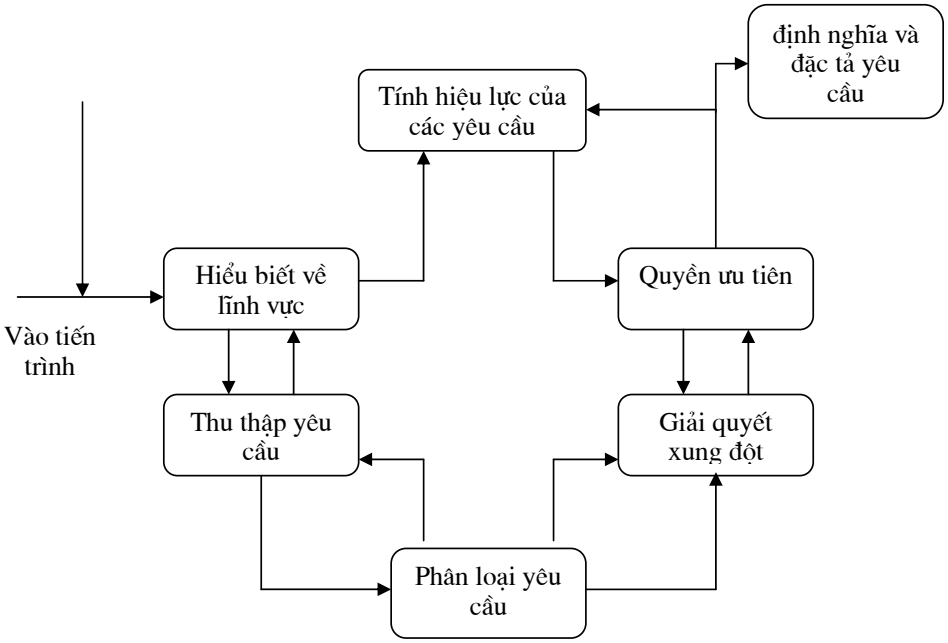
- Các nhà đầu tư khác nhau có những yêu cầu khác nhau và họ có thể tiến hành theo những cách rất khác nhau. Người kỹ sư phải nhận biết được những điểm chung và những điểm khác biệt đó.

- Việc phân tích lấy một không gian trong khung cảnh của tổ chức, những nhân tố chính trị (chẳng hạn như việc phân quyền) có thể ảnh hưởng đến yêu cầu của hệ thống, những nhân tố này có thể không rõ ràng mạch lạc tới người sử dụng cuối cùng. Ví dụ như người lãnh đạo thường có quyền cao hơn đối với hệ thống.

- Việc phân tích lấy khía cạnh thương mại và kinh tế làm động lực. Nó chắc chắn sẽ thay đổi trong quá trình phân tích. Từ đây, những yêu cầu riêng lẻ có thể thay đổi, những yêu cầu mới có thể xuất hiện từ phía các nhà đầu tư mới. Các nhà đầu tư mới sẽ không phải thăm dò lại từ đầu mà chỉ phải bổ xung thêm thông tin hay yêu cầu.

  Để tiến hành phân tích yêu cầu, ta phải dựa vào sự hiểu biết nhất định trong các lĩnh vực cụ thể, khi đó mô hình của tiến trình phân tích chắc chắn sẽ đơn giản hơn.

  Mô hình sau đây nêu lên một số bước quan trọng trong tiến trình phân tích.



Trong sơ đồ trên các bước bổ xung cho nhau, chu kỳ bắt đầu từ hiểu biết về các lĩnh vực và cuối cùng là bản phê chuẩn các yêu cầu. Những hiểu biết của quá trình phân tích sẽ dần được cải thiện sau mỗi chu trình.

- Hiểu biết về lĩnh vực ứng dụng: Phân tích phải được phát triển qua sự hiểu biết về các lĩnh vực ứng dụng. Giả sử có một hệ thống siêu thị, quy trình phân tích phải tìm ra được những điểm chung nhất, khái quát nhất giữa các siêu thị.
- Thu thập yêu cầu: Quy trình này liên quan đến các nhà đầu tư, người xây dựng hệ thống phải thông qua họ để khám phá ra các yêu cầu. Từ đó sẽ nâng cao hiểu biết để phát triển và xây dựng hệ thống.
- Phân loại yêu cầu: Quá trình này lấy các yêu cầu một cách ngẫu nhiên, không có cấu trúc và sắp xếp chúng một cách có hệ thống.
- Giải quyết xung đột: Chắc chắn trong quá trình đưa ra yêu cầu, các nhà đầu tư do không có chuyên môn nên xung đột có thể xảy ra. Công việc của người xây dựng hệ thống là cần phải tìm ra và giải quyết được các xung đột đó.
- Quyền ưu tiên: Trong một tập hợp các yêu cầu, bao giờ cũng phải có những yêu cầu quan trọng, cấp bách hơn yêu cầu khác, vì vậy ta phải tác động đến các nhà đầu tư để khám phá ra các yêu cầu cần thiết nhất, từ đó có kế hoạch xây dựng hệ thống.
- Làm cho các yêu cầu trở nên có hiệu lực: Các yêu cầu phải đảm bảo tính kiên định và phù hợp với nhu cầu thực của các nhà đầu tư. Từ đó mới có thể xây dựng được một hệ thống hữu ích.

  Trong quá trình phân tích yêu cầu, một vài mô hình hệ thống khác nhau có thể được sinh ra, mô hình là hình ảnh trìu tượng của hệ thống, các mô hình khác nhau sẽ có các thông tin khác nhau. Sự khác nhau này xuất phát từ nguồn gốc các yêu cầu phục vụ khác nhau.

II/ Xác định yêu cầu:

Xác định yêu cầu là mô tả trìu tượng các dịch vụ mà hệ thống phải cung cấp cũng như các ràng buộc mà hệ thống phải tuân theo khi thực hiện. Đặc điểm của nó là tư liệu được viết theo kiểu hướng khách hàng, do vậy nó phải đượcviết bằng ngôn ngữ để khách hàng có thể hiểu được. Nó chỉ đặc tả hành vi bên ngoài của hệ thống mà không mô tả chi tiết thiết kế. Hệ thống các yêu cầu được chia làm hai loại:

- Các yêu cầu chức năng: Là các dịch vụ mà hệ thống phải cung cấp. Do vậy các yêu cầu chức năng sẽ tác động trở lại các dữ liệu vào và có ảnh hưởng đến một số tình huống đặc biệt. Trong một số trường hợp, yêu cầu chức năng cũng có thể có các trạng thái không phải làm việc.
- Yêu cầu phi chức năng: Đó là các ràng buộc mà hệ thống phải tuân theo. Nó bao gồm sự ràng buộc về thời gian, các chuẩn công nghệ trong quá trình xử lý...

Trong thực tế, xác định yêu cầu của hệ thống vừa phải hoàn thiện, vừa phải tráng kiện, hoàn thiện có nghĩa là mọi yêu cầu của hệ thống được nêu lên đều phải có trong xác định yêu cầu, tính tráng kiện nghĩa là trong một xác định yêu cầu không thể có các yêu cầu phủ định nhau, mâu thuẫn nhau. Nói cách khác, nó phải đảm bảo được tính thống nhất.

Đối với các hệ thống phức tạp, ta khó có thể tìm được một xác định yêu cầu vừa đầy đủ, vừa tráng kiện. Một phần là do tính phức tạp cố hữu của hệ thống, một phần là do quan điểm khác nhau của nhu cầu người sử dụng. Tính không kiên định sẽ không rõ ràng khi các yêu cầu được đưa ra lần đầu, vấn đề là ta phải phát hiện ra nó ở các bước phân tích sâu hơn.

Xác định yêu cầu được viết bằng ngôn ngữ tự nhiên, từ đó nó sẽ nảy sinh ra các vấn đề sau:

- Tính thiếu rõ ràng: Do viết bằng ngôn ngữ tự nhiên, nên đôi khi mơ hồ dài dòng, khó hiểu.
- Sự lẫn lộn yêu cầu: Không phân biệt được đâu là yêu cầu chức năng và đâu là yêu cầu phi chức năng.
- Sự trộn lẫn trong các yêu cầu: Các yêu khác nhau có thể biểu diễn thành một yêu cầu chung.

Yêu cầu bao gồm cả khái niệm và thông tin chi tiết, nó đưa ra các khái niệm có cấu hình điều khiển thuận lợi được cung cấp như một phần cố định của APSE. Tuy nhiên nó cũng có những thuận lợi trong việc truy nhập tới đối tượng trong một nhóm người sử dụng một tên chưa hoàn chỉnh. Một vài tổ chức cố gắng sản xuất ra một đặc tả đơn để vừa hoạt động như một bản xác định yêu cầu vừa như một bản đặc tả yêu cầu. Khi một bản xác định yêu cầu được kết hợp với một bản đặc tả yêu cầu, ta thường nhầm lẫn giữa khái niệm và chi tiết.

Mô hình đầu tiên trong việc xác định yêu cầu là việc kết hợp ba loại yêu cầu khác nhau:

- Một nhận thức các trạng thái yêu cầu chức năng mà hệ thống soạn thảo sẽ cung cấp một mạng lưới. Nó đưa ra tính hợp lý cho vấn đề này.
- Một yêu cầu phi chức năng đưa ra những thông tin chi tiết về các đơn vị lưới(cm hay inc)
- Một yêu cầu phi chức năng của giao diện người dùng xác định người dùng bật hay tắt lưới.(chiếu)

III/ Đặc tả yêu cầu:

Đặc tả yêu cầu là đưa thêm các thông tin vào bản xác định yêu cầu. Đặc tả yêu cầu thường được dùng với các mô hình hệ thống, phát triển trong suốt quá trình phân tích yêu cầu. Đặc tả cùng và mô hình sẽ được mô tả trong hệ thống để thiết kế và thực hiện. Nó cũng bao gồm tất cả những thông tin cần thiết mà hệ thống phải thực hiện.

Ngôn ngữ tự nhiên thường được dùng để đặc tả yêu cầu. Tuy nhiên, đặc tả bằng ngôn ngữ tự nhiên không phải là cơ sở tốt cho một thiết kế hoặc cho một hợp đồng giữa khách hàng và người phát triển hệ thống. Sau đây là một vài lý do:

- Ngôn ngữ tự nhiên có thể hiểu được là dựa vào việc đọc bản đặc tả và khi viết sử dụng những từ giống nhau cho những khái niệm giống nhau. Điều này rất khó hiểu bởi các từ trong ngôn ngữ tự nhiên đôi khi rất mơ hồ.
- Một bản đặc tả yêu cầu bằng ngôn ngữ tự nhiên rất linh hoạt. bạn có thể nói về cùng một vấn đề bằng những cách khác nhau, nó làm cho người đọc khó tìm ra những yêu cầu giống nhau. đó là lỗi nghiêng về phía tiến trình.
- Các yêu cầu không được phân chia một cách có hiệu quả bằng ngôn ngữ đó, do đó khó tìm ra được các yêu cầu quan hệ. Để khám phá ra một sự thay đổi bạn phải xem xét tất cả các yêu cầu, chứ không chỉ trong nhóm các yêu cầu quan hệ.

Bởi tất cả các lý do trên, đặc tả yêu cầu được viết bằng ngôn ngữ tự nhiên có thể bị hiểu sai, điều này thường không được phát hiện cho đến giai đoạn thiết kế hoặc thực hiện các giai đoạn của tiến trình phần mềm. Vì vậy có mọt cách tốt hơn là dùng luân phiên các kí hiệu để tránh một vài vấn đề về không hạn chế được bằng ngôn ngữ tự nhiên, đó là đưa cấu trúc vào đặt tả. Sau đây là một số phương pháp:

- Ngôn ngữ tự nhiên có cấu trúc: Cách tiếp cận này dựa trên việc định nghĩa các chuẩn mẫu hoặc các chuẩn tạm thời để xây dựng đặc tả yêu cầu.
- Ngôn ngữ mô tả thiết kế: Cách tiếp cận này dựa vào việc sử dụng các ngôn ngữ giống như ngôn ngữ lập trình nhưng có nhiều điểm trìu tượng hơn để phân loại yêu cầu bằng việc đưa ra một mô hình có thể thực hiện được của hệ thống.
- Ngôn ngữ đặc tả yêu cầu: Một số ngôn ngữ được thiết kế cho những mục đích đặc biệt để xây dựng các yêu cầu phần mềm. Ví dụ như: PSL/PSA (Tiechrow và Hershey, 1977) và RSL (Alford, 1977; Bell et at.1977; Alford, 1985). Những tiến bộ của cách tiếp cận này là việc cung cấp các công cụ có mục đích đặc biệt được phát triển.
- Các kí hiệu đồ hoạ: Hệ thống kí hiệu đồ hoạ tốt nhất có lẽ là SADT (Ross, 1977; Schoman và Ross, 1977). SADT có một bộ kí hiệu đồ hoạ quen thuộc, vì vậy chủ yếu được các nhà đầu tư sử dụng. Nó trở nên thân thiện trong việc sử dụng để phân tích và đặc tả yêu cầu.
- Đặc tả toán học: Dùng các kỹ thuật dựa trên các khái niệm toán học có cơ sở như: tập hợp, máy hỗn hợp trạng thái hoặc lưới để đặc tả yêu cầu.
  Ưu điểm: loại bỏ hoàn toàn tính mơ hồ trong đặc tả.
  Nhược điểm: khó khăn cho bên khác hàng bởi họ không hiểu được các kí hiệu toán học.
  Khắc phục: thêm những chú giải bằng ngôn ngữ thử nghiệm ở những chỗ thích hợp.
  Tính dò theo được: khi viết ra các đặc tả yêu cầu, một điểm quan trọng là các yêu cầu có liên quan với nhau phải tham khảo chéo nhau được. Dò theo được là một thuộc tính của đặc tả yêu cầu phản ánh tính dễ tìm ra các yêu cầu có liên quan.

Davis (1990) đã tóm tắt và so sánh một vài cách tiếp cận khác nhau để đặc tả yêu cầu. Trong hai cách tiếp cận đầu, chủ yếu là ngôn ngữ thử nghiệm có cấu trúc và ngôn ngữ mô tả thiết kế. Việc làm thích ứng các ngôn ngữ yêu cầu không được biết hay sử dụng rộng rãi. Các kí hiệu đồ hoạ tương tự như các kí hiệu được sử dụng để xây dựng các mô hình hệ thống.

Khi viết đặc tả yêu cầu một điều rất quan trọng là các yêu cầu quan hệ phải phù hợp khi thay đổi các yêu cầu hoặc các yêu cầu khác được tìm thấy. Một vài mối quan hệ giữa các yêu cầu là rất tế nhị. Đó là lý do mà ta không thể vạch ra cụ thể các yêu cầu. Tuy nhiên, có một vài phương thức đơn giản có thể áp dụng cho việc xác định và đặc tả yêu cầu:

- Tất cả các yêu cầu nên được phân chia
- Các yêu cầu cần phải xác định rõ quan hệ.
- Mỗi tài liệu yêu cầu phải chứa một ma trận để chỉ ra các yêu cầu quan hệ.

Các ma trận khác nhau có thể được phát triển cho các loại quan hệ khác nhau.
Một kỹ thuật đơn giản là các công cụ CASE được sử dụng để cung cấp các phân tích và đặc tả yêu cầu. Một vài công cụ có các tiện ích đơn giản như tìm các yêu cầu sử dụng trong cùng thời kỳ, kết nối các tiện ích từ một yêu cầu đến một yêu cầu có quan hệ khác có thể được cung cấp.

Thiết kế phần mềm

Một bản thiết kế tốt là chìa khoá dẫn đến thành công. Tuy nhiên, ta không thể chính thức hoá tiến trình xử lý trong bất kỳ quy tắc kỹ thuật nào. Thiết kế là sáng tạo ra một tiến trình nhìn thấu các yêu cầu và khả năng trong từng phần của thiết kế. Nó phải thực hành, học hỏi ở những người có kinh nghiệm và nghiên cứu các hệ thống đang tồn tại.

Các vấn đề thiết kế phải được khôi phục trong ba giai đoạn:

1. Nghiên cứu và hiểu được vấn đề: Nếu không hiểu được vấn đề, hiệu quả của công việc thiết kế phần mềm sẽ rất thấp. Chúng ta nên xem xét từ những khía cạnh, những quan điểm khác nhau trong các yêu cầu thiết kế.

2. Xác định toàn bộ các đặc trưng của giải pháp có thể: Điều này rất có lợi cho việc xác định các giải pháp và xem xét đánh giá chúng. Sự lựa chọn giải pháp phụ thuộc vào kinh nghiệm của người thiết kế, tính có giá trị của các thành phần có thể dùng lại được và sự đơn giản của giải pháp. Người thiết kế thường thích dùng các giải pháp quen thuộc mặc dù nó không phải là giải pháp tối ưu vì họ hiểu được những thuận lợi và khó khăn.

3. Các mô tả mang tính trìu tượng hoá được sử dụng trong giải pháp: Trước khi tạo ra một tài liệu chính thứ, người thiết kế có thể viết một bản mô tả các thông tin thiết kế. Điều này có thể được phân tích trong quá trình thiết kế chi tiết. Các lỗi và các thiếu sót trong thiết kế mức cao sẽ được khám phá trong quá trình phân tích. Chúng sẽ được khắc phục khi làm thành tài liệu.

   Tiến trình giải quyết vấn đề được lặp lại sau mỗi lần xác định các thực thể mang tính trìu tượng trong thiết kế. Tiến trình cải tiến còn tiếp tục cho tới khi một bản của mỗi thực thể mang tính trìu tượng được chuẩn bị.

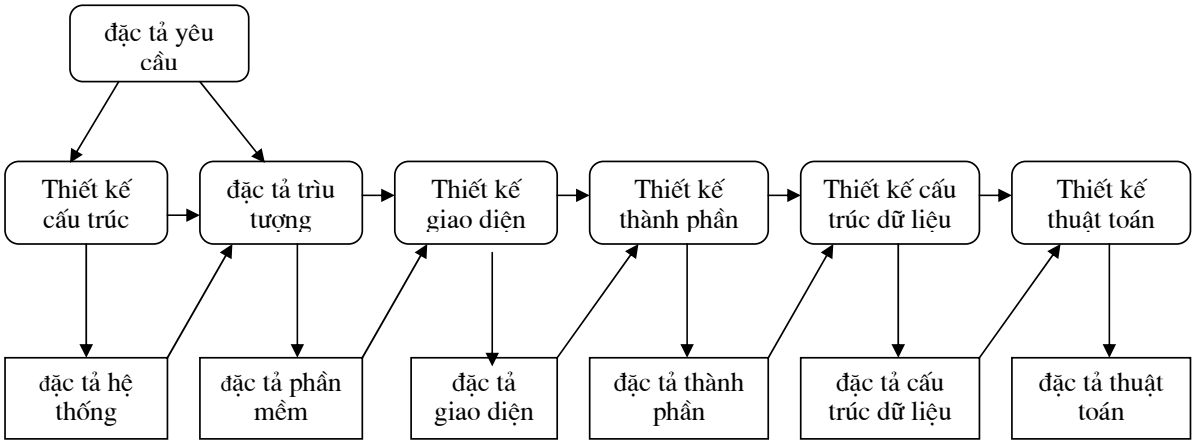
I/ Tiến trình thiết kế:

Một mô hình tổng thể của một bản thiết kế phần mềm là một sơ đồ có hướng. Mục tiêu của tiến trình thiết kế là tạo ra một sơ đồ không có mâu thuẫn. Cần chú ý rằng sơ đồ này được đưa ra những thực thể được thiết kế như các tiến trình, chức năng hoặc kiểu. Việc thiết lập các mối quan hệ giữa các thực thể thường xuyên được thực hiện.

Người thiết kế phần mềm không tới ngay điểm kết thúc của sơ đồ thiết kế ngay lập tức mà phát triển thêm bằng cách lặp lai các thiết kế qua một số phiên bản khác. Tiến trình thiết kế sẽ đưa thêm thông tin và chi tiết hoá khi bản thiết kế được phát triểnvới sự quay lui nhất định để sửa lỗi

sớm hơn. Điểm đầu tiên là một thiết kế không theo qui tắc và được tìm lại bằng cách đưa thêm thông tin để làm cho nó phù hợp và hoàn thiện hơn.

Tiến trình thiết kế liên quan đến việc phát triển mô hình hệ thống ở những mức trìu tượng khác nhau. Khi một bản thiết kế được phân tích, các lỗi và các thiếu sót sẽ được phát hiện sớm hơn. Những thông tin phản hồi này sẽ cải tiến những mô hình thiết kế trước đó. Hình saulà một mô hình chung của tiến trình thiết kế và các mô tả thiết kế sẽ xuất hiện trong những bước khác nhau của tiến trình thiết kế.



Không có một ranh giới rõ ràng giữa các bước nhưng việc xác định các bước rất có lợi để tạo ra những tiến trình thiết kế rõ ràng.

Một bản đặc tả là đầu ra của mỗi hoạt động thiết kế. Bản đặc tả này có thể là một văn bản tóm tắt, nó chắt lọc các yêu cầu hoặc nó có thể là một bản đặc tả các phần của hệ thống được nhận biết. Tiếp theo, tiến trình thiết kế các chi tiết sẽ được đưa thêm vào bản đặc tả. Kết quả cuối cùng của tiến trình là một bản đặc tả chính xác thuật toán và cấu trúc dữ liệu được thực hiện.

Hình vẽ trên dự đoán rằng các bước của quá trình thiết kế là tuần tự. Nhưng thực tế các hoạt động của tiến trình thiết kế được tiến hành song song. Tuy nhiên những hoạt động được chỉ ra là tất cả các phần của tiến trình thiết kế đối với những hệ thống phần mềm lớn. Các hoạt động thiết kế này là:

1. Thiết kế cấu trúc: Các hệ con tạo thành một hệ và những mối quan hệ của chúng được xác định và tài liệu hoá.
2. Đặc tả tóm tắt: Với mỗi hệ con, một bản đặc tả tóm tắt các dịch vụ được cung cấp và những dịch vụ bắt buộc phải được cung cấp.
3. Thiết kế giao diện: Với mỗi hệ con, giao diện của nó với hệ con khác được thiết kế và cung cấp tài liệu bản đặc tả giao diện phải rõ ràng vì nó cho phép các hệ con được sử dụng mà không cần những thao tác của hệ con.
4. Thiết kế thành phần: Các dịch vụ cố định được phân phối trong các thành phần khác nhau và giao diện của các thành phần này được thiết kế.
5. Thiết kế cấu trúc dữ liệu: Cấu trúc dữ liệu được sử dụng khi thực hiện hệ thống được thiết kế chi tiết và được chỉ rõ.
6. Thiết kế thuật toán: Thuật toán được sử dụng để cung cấp các dịch vụ được thiết kế chi tiết và được chỉ rõ.

Tiến trình này được lặp đi lặp lại trong mỗi hệ con cho đến khi các thành phần được nhận biết có thể được sắp xếp trực tiếp vào các thành phần chương trình như các góc, các chương trình hàm. Thiết kế từ trên xuống là một cách để khắc phục một vấn đề trong thiết kế. Vấn đề đó được chia cắt một cách đệ quy thành những vấn đề nhỏ hơn cho đến khi các vấn đề con dễ điều khiển được nhận ra.
Khuôn mẫu chung của thiết kế thường xuất hiện như là một tiến trình có thứ tự Crooss-links trong sơ đồ hiện ra ở mức thấp hơn của công thiết kế khi người thiết kế xác định có thể hiểu tốt một số thành phần của bản thiết kế vì vậy sẽ trì hoãn việc phân tích chúng cho đến khi có những thành phần khác khó hiểu hơn được phát hiện. Hơn nữa việc lập kế hoạch dự án có thể chỉ đòi hỏi sự hiểu biết nhỏ vì những thành phần của hệ thống được khắc phục đầu tiên.

Thiết kế trên xuống dựa trên việc phân tích một cách có hệ thống các đối tượng trìu tượng rõ ràng là một cách tiếp cận có hiệu quả khi các thành phần thiết kế được kết hợp chặt chẽ. Tuy nhiên khi cách tiếp cận hướng đối tượng được thiết kế được chấp nhận và nhiều đối tượng đang tồn tại có thể được dừng lại tiến trình thiết kế không thực sự bắt đầu với một sự trìu tượng hoàn thiện. Trong cách tiếp cận hướng đối tượng, các đối tượng đang tồn tại được sử dụng như một khung thiết kế và bản thiết kế được xây dựng từ chúng. Đó không phải là khái niệm của một đích đơn lẻ hoặc của tất cả các đối tượng đang tồn tại trong một đối tượng có thứ bậc đơn lẻ.

1. Phương thức thiết kế:

Trong nhiều dự án phát triển phần mềm, thiết kế phần mềm vẫn là một tiến trình có mục đích đặc biệt. Bắt đầu từ tập hợp các yêu cầu, những ngôn ngữ thông thường trong tự nhiên, một kế hoạch được chuẩn bị. Công việc mã hoá bắt đầu và bản thiết kế được sửa chữa khi thực hiện hệ thống. Có rất ít hoặc không có những thay đổi hoặc điều khiển thiết kế khi từng bước thực hiện được hoàn thành, bản thiết kế đã thay đổi rất nhiều so với đặc tả ban đầu mà các tài liệu thiết kế gốc là một mô tả chưa đúng và chưa hoàn chỉnh của hệ thống.

Một cách tiếp cận có hệ thống hơn để thiết kế phần mềm được đề xuất bởi phương thức có cấu trúc mà là một tập hợp các kí hiệu và chỉ dẫn trong thiết kế phần mềm. Budgen (1993) mô tả một vài phương thức được sử dụng như thiết kế cấu trúc (Constantine và Yourdon, 1979), phân tích hệ thống có cấu trúc (Gane và Sarson,1979), phát triển hệ thống Jackson (Jackson,1983) và nhiều cách tiếp cận thiết kế hướng đối tượng (Robinson, 1992; Boach,1994).

Việc sử dụng các phương thức có cấu trúc thường liên quan đến việc sản xuất một lượng lớn các tài liệu thiết kế dưới dạng sơ đồ. Các công cụ trợ giúp CASE được phát triển để cung cấp các phương thức riêng biệt. Các phương thức có cấu trúc thường được ứng dụng thành công trong nhiều dự án lớn. Chúng có thể bắt nguồn từ sự giảm bớt giá trị bởi cách sử dụng những kí hiệu chuẩn và chắc chắn rằng các tài liệu thiết kế chuẩn được sản xuất.
Một phương thức mang tính toán học là một chiến lược luôn cho những kết quả giống nhau bất kể ai là người áp dụng phương thức đó. Thuật ngữ “structed method” dự đoán rằng các nhà thiết kế sẽ thiết kế giống như trong đặc tả.

Trên thực tế những chỉ dẫn được đưa ra bởi các phương thức là không bắt buộc vì những tình huống này là không giống nhau. Những phương thức này thực sự là những kí hiệu chuẩn và sự biểu hiện thực hiện tốt. Theo những phương thức này và áp dụng các chỉ dẫn, bản thiết kế hợp lý xuất hiện. Sản phẩm của người thiết kế vẫn cần có để quyết định việc phân tích hệ thống. Do kinh nghiệm nghiên cứu của các nhà thiết kế (Bansler và Bodker, 1993) đã chỉ ra rằng chúng ít khi tuân theo những phương thức mang tính bắt chước thái quá. Chúng lựa và chọn những chỉ dẫn phụ thuộc vào hoàn cảnh nhất định.

Một phương thức có cấu trúc bao gồm một tập các hoạt động, kí hiệu, báo cáo, quy tắc, chỉ dẫn thiết kế. Mặc dù có nhiều phương thức nhưng giữa chúng vẫn có những điểm chung. Phương thức có cấu trúc thường cung cấp một vài hoặc tất cả các mô hình của hệ thống.

- Mô hình dữ liệu nơi hệ thống được mô hình hoá sử dụng các thay đổi dữ liệu trong quá trình sử lý.
- Các mô hình quan hệ thực thể được sử dụng để mô tả các cấu trúc dữ liệu có tính logic đang được sử dụng.
- Một mô hình có cấu trúc, nơi các thành phần hệ thống sự tích hợp chúng được tài liệu hoá.
- Nếu một phương thức là hướng đối tượng nó sẽ bao gồm một mô hình mà các đối tượng được hợp thành từ những đối tượng khác và thông thường một mô hình sử dụng đối tượng chỉ ra những đối tượng sử dụng đối tượng khác.

Các phương thức đặc biệt được bổ xung này với các mô hình hệ thống khác như các sơ đồ của thời kỳ quá độ, lịch sử của các thực thể chỉ ra cách mà thực thể được thay đổi khi nó bị xử lý....

Hầu hết các phương thức gợi ý một nơi lưu trữ tập trung cho hệ thống thông tin hay từ điển dữ liệu sẽ được sử dụng. Không có phương thức nào tốt hơn phương thức nào, những thành công hay thất bại đều phụ thuộc vào khả năng phù hợp với các lĩnh vực ứng dụng.

2. Sự mô tả thiết kế :

Một bản thiết kế phần mềm là một mô hình của thế giới thực có nhiều thực thể quan hệ. Bản thiết kế này được sử dụng trong một số cách khác nhau. Nó hoạt động như cơ sở cho việc thực hiện chi tiết. Nó đóng vai trò quan như một công cụ truyền thông chung gian giữa những người thiết kế các hệ thống con. Nó cung cấp thông tin tới những người bảo trì hệ thống về mục đích gốc của người thiết kế hệ thống.

Các thiết kế được tài liệu hoá trong một tập các tài liệu thiết kế cho người mô tả thiết kế cho người lập trình và những người thiết kế khác. Có ba loại kí hiệu chính được sử dụng trong các tài liệu thiết kế:

- Các kí hiệu đồ hoạ: Chúng được sử dụng để hiển thị những quan hệ giữa những thành phần làm nên bản thiết kế và tạo mối quan hệ giữa bản thiết kế và những thực thể trong thế giới thực. Một hình ảnh đồ hoạ của thiết kế là một hình ảnh trìu tượng. Nó rất có ích trong việc đưa ra một bức tranh toàn cảnh của hệ thống.
- Ngôn ngữ mô tả chương trình: ngôn ngữ được sử dụng điều khiển và xây dựng cấu trúc dựa trên ngôn ngữ lập trình cũng như cho phép các văn bản minh hoạ và đôi khi có thêm các loại báo cáo được sử dụng. Điều này cho phép mục đích của người thiết kế được bày tỏ một cách chi tiết hơn.
- Văn bản không theo thủ tục: nhiều thông tin được kết nối với một thiết kế không được sử lý. Những thông tin về các lý do thiết kế hoặc các sự kiện phi chức năng có thể được tiến hành bằng cách sử dụng các văn bản ngôn ngữ tự nhiên.

Nhìn chung, tất cả các kí hiệu khác nhau có thể được sử dụng trong mô tả thiết kế hệ thống. Cấu trúc và tính logic của dữ liệu thiết kế sẽ được mô tả một cách hình ảnh, được bổ xung các lý do thiết kế và hơn nữa, những văn bản mô tả bắt buộc hoặc không bắt buộc. Thiết kế giao diện và thiết kế cơ sở dữ liệu chi tiết, thiết kế tính logic tốt nhất nên sử dụng một PDL, hoặc một số trường hợp, sử dụng các kí hiệu bắt buộc. Các lý do mô tả có thể bao gồm các chú thích rõ ràng hơn.

II/ Chiến lược thiết kế:

Gần đây người ta thường sử dụng các chiến lược thiết kế dựa trên việc phân tích bản thiết kế thành các thành phần chức năng với hệ thống thông tin quản lý trong một vùng dữ liệu được chia sẻ. Although Parnas (1972) đã gợi ý một chiến lược xen lẫn vào đầu những năm 70 vàchỉ đến cuối năm 80 chiến lược thiết kế hướng đối tượng mới được chấp nhận.

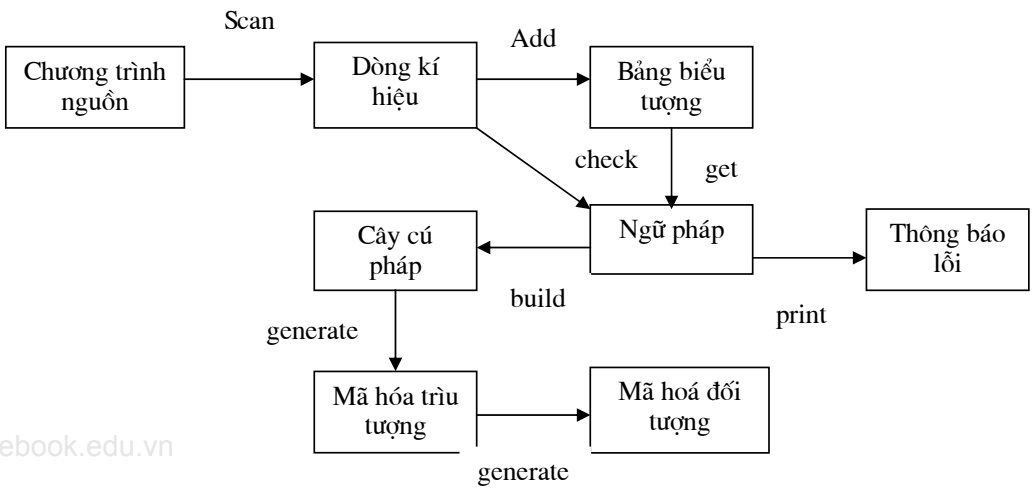
1. Thiết kế chức năng: Hệ thống được thiết kế từ quan điểm hướng chức năng bắt đầu ở tầm nhìn mức cao sau đó được cải tiến dần thành bản thiết kế chi tiết hơn. Cách thức hệ thống được tập trung hay bị chia sẻ giữa các thao tác trong cách thức đó. Chiến lược này được minh hoạ bởi bản thiết kế cấu trúc (của Constantune và Jourdon,1979) SSADM (Cutts,1988; Werver, 1993).... Các phương thức như Jackson Structured Programming (Jacckson, 1975) và Warnier-on methord (Warnier, 1977) là các công nghệ phân tích chức năng, các cấu trúc dữ liệu được sử dụng để quyết định cấu trúc chức năng được sử dụng để xử lý dữ liệu.

2. Thiết kế hướngđối tượng: Hệ thống được coi như tập hợp các đối tượng, là các chức năng. Thiết kế hướng đối tượng dựa trên ý kiến về che dấu thông tin (Parnas, 1972) và được mô tả bởi Muyer (1988), Booch (1994), Jacobsen et at (1993) và nhiều người khác. ISD (Jackson, 1983) là một phương thức thiết kế kết hợp giữa thiết kế hướng chức năng và thiết kế hướng đối tượng.

Trong một bản thiết kế hướng đối tượng cách thức của hệ thống là phân quyền và mỗi đối tượng điều khiển trạng thái thông tin của chính nó. Các đối tượng có một tập các thuộc tính để xác định cách thức và các thao tác của hoạt động dựa trên các đặc tính đó. Các đối tượng thường là thành viên của một lớp đối tượng có chung các thuộc tính và thao tác. Những điều này có thể được kế thừa từ một hoặc nhiều lớp vì vậy một định nghĩa lớp chỉ cần nêu ra sự khác giữa các lớp. Theo quan niệm, các đối tượng truyền các thông điệp trao đổi. Trên thực tế, hầu hết các đối tượng truyền thông được hoàn thành do một đối tượng gọi là một thủ tục kết giao với đối tượng khác. hình vẽ trên đã minh hoạ điều này.

Để minh hoạ cho sự khác nhau giữa cách tiếp cận hướng chức năng và hướng đối tượng trong thiết kế phần mềm, ta hãy xem cấu trúc của một bộ dịch. Nó có thể coi như một tập hợp các truyền thông chung với các thông tin kết nối từ một đơn vị xử lý tới đơn vị khác.

Các thành phần chủ yếu trong khung cảnh chức năng được xem như các hoạt động: "Scan","build", "analyse", "generate".... Một sự xen kẽ các hình ảnh hướng đối tượng của hệ thống được chỉ ra trong hình vẽ sau:



Các đối tượng được kết nối bởi bộ dịch, là trung tâm của các thao tác được kết giao với mỗi đối tượng. Trong trường hợp này, các thành phần chính được định nghĩa như là các thực thể: “Token Stream”, “syntax tree”,..
Những người quan tâm đến kỹ thuật thiết kế đôi khi cho rằng kỹ thuật mà họ yêu thích có thể áp dụng cho tất cả các ứng dụng. Điều đó là nguy hiểm bởi họ đã quá đơn giản hoá các vấn đề trong thiết kế. Qua kinh nghiệm, sự thất vọng của những người sử dụng trong công nghệ riêng lẻ có thể bác bỏ tính hoàn thiện mặc dù nó được áp dụng tốt trong một vài lĩnh vực ứng dụng.

Nó không phải là một chiến lược thiết kế tốt nhất mà nó phù hợp cho tất cả các dự án và cho tất cả các loại ứng dụng. Các cách tiếp cận hướng đối tượng và hướng chức năng thường bổ xung cho nhau chứ không đối lập nhau. Trên thực tế, các kỹ sư phần mềm chọn cách tiếp cận thích hợp nhất cho mỗi bước trong tiến trình thiết kế, các hệ thống phần mềm lớn rất phức tạp nên cần phải sử dụng các cách tiếp cận khác nhau trong thiết kế cho từng phần của hệ thống.

Để minh hoạ cho điều này, ta lấy ví dụ hệ thống phần mềm là một phần của máy bay dân sự hiện đại. Đứng ở mức độ tổng quát, chúng ta xem phần mềm này như là một tập các hệ thống con hoặc các đối tượng. Chúng ta sử dụng các kỹ nghệ khác nhau sao cho phù hợp. Chúng ta nói đó là ”The navigation system”, ”the engine control system”....Tại mức thiết kế trìu tượng này, cách tiếp cận hướng đối tượnglà su thế tự nhiên, khi hệ thống được giám sát chi tiết hơn, sự mô tả tự nhiên của nó được coi như một bộ các chức năng tích hợp hơn là các đối tượng. Một vài chức năng có thể là:

- Hiển thị dấu vết(radar sub-system)
- Đối trọng tốc độ gió(navigation Sub-system)
- Giảm năng lượng(sức mạnh-power) (engine control sub-system)
- Dấu hiệu hỏng máy móc(Instrument sub-system)
- Lock-onta frequency(communication sub-system)

Hình ảnh các chức năng này bắt nguồn từ quá trình xác định yêu cầu. Điều này có thể được chuyển đổi sang một bản thiết kế hướng đối tượng. Tuy nhiên khi đó tính hiện thực của hệ thống không cao, bởi không có một sự phù hợp giữa các thành phần thiết kế và định nghĩa yêu cầu. Một chức năng logic đơn giản trong định nghĩa yêu cầu có thể thực hiện như một thứ tự tích hợp đối tượng.

Khi thiết kế hệ thống, điều quan trọng là phải phân tích, một hình ảnh hướng đối tượng lại có thể biểu thị một cách tự nhiên. Tại bước thiết kế chi tiết, các đối tượng được kết nối có thể là: trạng thái máy móc (the engine status); Vị trí máy bay (the aircraft-position), đồng hồ đo độ cao (the-artimetar), sóng báo hiệu (the radio beacen).... Theo cách này, một phương thức tiếp cận hướng đối tượng được áp dụng để giảm các mức trong thiết kế hệ thống là có hiệu quả.

Tóm lại, cách tiếp cận hướng đối tượng để thiết kế phần mềm dường như là tự nhiên trong thiết kế hệ thống ở mức cao nhất và thấp nhất. Trên thực tế, tất nhiên sử dụng các cách tiếp cận khác nhau để thiết kế có thể đòi hỏi người thiết kế chuyển đổi bản thiết kế của họ từ một mô hình này sang mô hình khác. Nhiều người thiết kế không kết hợp các cách tiếp cận mà thích sử dụng thiết kế hướng đối tượng hoặc hướng chức năng.

III/ Chất lượng thiết kế:

Không có một sự chấp nhận chung trên giả thuyết của một bản thiết kế tốt. Một phần là do những tiêu chuẩn mơ hồ mà một bản thiết kế thực hiện đúng theo bản đặc tả. Một bản thiết kế

tốt là một bản thiết kế cho phép tạo ra mã có hiệu lực, nó có thể là bản thiết kế hầu như có thể duy trì.

Một bản thiết kế duy trì được có thể thích hợp để sửa đổi các chức năng duy trì và thêm vào các chức năng mới. Các thành phần thiết kế phải dễ hiểu và những thay đổi nên được xác định trong tác động. Các thành phần thiết kế phải có độ dính kết, điều đó có nghĩa là chúng phải có sự tích hợp chặt chẽ. Tính ghép nối là thước đo sự phụ thuộc giữa các thành phần.

Hệ thống đo chất lượng thiết kế có thể được sử dụng để đánh giá xem bản thiết kế có tốt không. Mục đích của hệ thống đo hầu như được phát triển trong sự kết nối với phương pháp tiếp cận chức năng hình ảnh là thiết kế hệ thống của Yourdon.

Các đặc tính chất lượng chia đều giữa thiết kế hướng đối tượng và hướng chức năng. Bởi vì thông thường trong thiết kế hướng đối tượng, khuyến khích các thàh phần phụ thuộc phát triển. Nó dễ thành công trong duy trì thiết kế vì các thông tin ẩn trong đối tượng.

1. Độ dính kết:

Độ dính kết của một thành phần là thước đo sự gần gũi trong mối quan hệ giữa các thành phần thực hiện một chức năng logic đơn lẻ hoặc thực hiện một thực thể logic đơn lẻ. Tất cả các phần của thành phần tạo nên sự thực hiện đó. Nếu một thành phần bao gồm những phần không có liên kết chặt chẽ để thực hiện các chức năng logic thì có nghĩa nó có độ kết dính thấp.
Độ kết dính là một đặc tính cần thiết bởi nó là một đơn vị đưa ra một phần đơn lẻ của giải pháp giải quyết những vấn đề phức tạp. Nó trở nên cần thiết để thay đổi hệ thống mà một phần đang tồn tại trong một nơi riêng biệt và mọi điều cần làm được tóm lược trong một đơn vị riêng lẻ. Không cần thiết phải sửa đổi nhiều thành phần nếu như có một sự thay đổi được thực hiện. Constantine và Jourdon (1979) đã đưa ra 7 mức độ dính kết:

- Độ dính kết trùng hợp: Các phần của một thành phần không quan hệ nhưng nó được bó buộc vào một thành phần đơn lẻ.
- Sự kết hợp logic: Các thành phần thực hiện các chức năng tương tự nhau như nhập, phát hiện lỗi.... được đưa vào một thành phần đơn lẻ cùng nhau.
- Độ dính kết tạm thời: Tất cả các thành phần hoạt động tại một thời điểm riêng lẻ như khởi động, tắt máy được mang cùng nhau.
- Độ dính kết thủ tục: Các bộ phận của một thành phần tạo ra một thứ tự điều khiển đơn lẻ.
- Độ dính kết truyền thông: Tất cả các bộ phận của một thành phần thực hiện trên cùng một dữ liệu vào hoặc đưa ra dữ liệu giống nhau.
- Độ dính kết có thứ tự: dữ liệu ra từ một bộ phận trong một thành phần là dữ liệu vào cho một bộ phận khác.
- Độ dính kết chức năng: Mỗi phần của một thành phần cần cho việc tiến hành các chức năng đơn lẻ.

  Các lớp dính kết này không được rõ ràng. Constantine và Jourdon minh hoạ mà lớp bằng một ví dụ. Nó không dễ để quyết định dùng loại dính kết nào cho một đơn vị được phân lớp. Phương thác của Constantine và Jourdon là chức năng tự hiên. Vì vậy hầu hết các mẫu dính kết của thành phần là chức năng. Tuy nhiên một độ dính kết có độ sâu cao cũng là khuôn mẫu tương lai của một hệ thống hướng đối tượng. Cách tiếp cận này để chỉ đọc thiết kế một cách tự nhiên, để các thành phần được dính kết.
  Một đối tượng dính kết là một trong các thực thể đơn đưa ra và tất cả các thao tác trên thực thể đó có liên quan đến đối tượng. Ví dụ như, một đối tượng đưa ra một bảng biểu tượng của

bộ dịch được dính kết nếu tất cả các chức năng cần thiết như " Addasiymbol", "Search table'.... được bao gồm với đối tượng bảng biểu tượng.
Do đó, một lớp xa hơn của độ dính kết có thể được định nghĩa như sau:

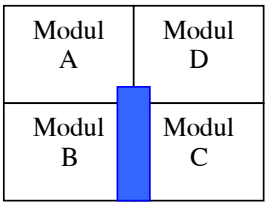
- Đối tượng dính kết: Mã thao tác cung cấp chức năng cho phép các đặc trưng của đối tượng được sửa đổi được xem xét hoặc sử dụng như cơ sở để cung cấp các dịch vụ.

Nếu một lớp hệ thống hướng đối tượng được kế thừa các đặc tính và các thao tác từ một lớp cao hơn, độ dính kết của lớp đó sẽ giảm, không mất nhiều thời gian để cân nhắc xem một lớp đối tượng được chứa trong một đơn vị. Tất cả các lớp trên cũng phải được xem xét nếu chức năng của các đối tượng được hiểu hoàn toàn. Hệ thống (browsers) hiển thị lớp đối tượng và duy trì để hiểu một thành phần kế thừa các đặc tính từ một lớp là rất phức tạp.

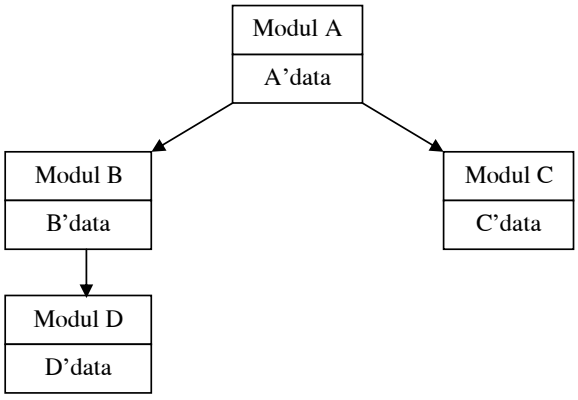
2. Sự ghép nối:

Sự ghép nối có quan hệ tới độ dính kết. Nó thể hiện sức mạnh của mối quan hệ nối liền giữa các thành phần trong thiết kế. Hệ thống ghép nối có mối quan hệ nối liền mạng, các đơn vị cũng phụ thuộc lẫn nhau. Hệ thống ghép nối chặt chẽ được làm từ những thành phần độc lập hoặc gần như độc lập.

Một tập các modun được ghép nối chặt chẽ nếu chúng sử dụng để chia sẻ các biến hoặc chúng thay đổi các thông tin điều khiển. Constantine và Jourdon gọi những điều này là ghép nối chung và ghép nối điều khiển. Sự ghép nối được hoàn thành do các chi tiết của việc đưa dữ liệu không được điều khiển bởi một thành phần thông qua giao diện của nó với các thành phần khác thông qua một danh sách biến. Nếu việc chia sẻ thông tin là cần thiết, sự chia sẻ cũng phải được giới hạn.
Hình 12.7 và12.8 minh hoạ các modun được ghép nối chặt chẽ và lỏng lẻo.



Ghép nối chặt chẽ



Ghép nối lỏng lẻo

Một mẫu khác của bộ phận hỗ trợ ghép nối khi các tên sớm bị buộc lại thành giá trị. Ví dụ nếu một chương trình liên quan tới sự tính toán thuế và tỉ lệ thuế là 30%, để mã hoá một số chương trình, chương trình đó được ghép nối chặt chẽ với tỉ lệ thuế. Những thay đổi về tỉ lệ thuế cũng đòi hỏi những thay đổi về chương trình. Nếu chương trình đọc trong tỉ lệ thuế tại thời điểm đang chạy, sự ghép nối sẽ bị giảm bớt. Sự thay đổi tỉ lệ thuế có thể không làm thay đổi chương trình. Các đối tượng điều khiển tạo ra những hệ thống có độ ghép nối không cao. Nó chủ yếu cho thiết

kế hướng đối tượng mà sự thực hiện của một bước chia sẻ và bất kỳ đối tượng nào có thể thay thế bởi đối tượng khác với giao diện.

Sự kế thừa trong một hệ thống hướng đối tượng tạo ra sự khác nhau của sự ghép nối. Các lớp đối tượng kế thừa các đặc tính và thao tác được ghép nối với lớp trên của nó (Super-class) thay đổi lớp trên phải được làm cẩn thận vì những thay đổi sinh sản ra tất cả các lớp kế thừa các đặc tính của chúng.

3. Tính hiểu được:

Tính hiểu được của thiết kế là rất quan trọng bởi và bất cứ một sự thay đổi nào trong thiết kế đầu tiên cũng phải hiểu nó. Một số đặc tính ảnh hưởng đến tính hiểu được, đó là:

- Độ dính kết và sự ghép nối: Một thành phần có thể hiểu được mà không cần các thành phần hay không?
- Việc đặt tên: Các tên của thành phần được sử dụng có ý nghĩa gì không? Các tên có ý nghĩa là các tên phản ánh được các thực thể trong thế giới thực được đem đặt cho các thành phần.
- Tài liệu: Các thành phần được tài liệu hoá có hình thành một ánh xạ giữa các thực thể trong thế giới thực và các thành phần một cách rõ ràng không?
- Sự phức tạp: Các thuật toán được sử dụng để thực hiện các thành phần phức tạp như thế nào?

Thuật ngữ ”sự phức tạp” được dùng ở đây là một cách thuộc về trực giác. Độ phức tạp cao hàm ý nhiều mối quan hệ giữa các thành phần thiết kế và độ sâu trong tập hợp câu lệnh
If- then- else. Các thành phần phức tạp thì khó hiểu, người thiết kế luôn phải cố gắng để thiết kế càng đơn giản càng tốt.

Hầu hết các công việc trong đo lường chất lượng thiết kế tập trung trong việc kiểm tra độ phức tạp của một thành phần. Điều này xem như tính phức tạp liên quan trực tiếp đến tính hiểu được. Tính phức tạp ảnh hưởng đến tính hiểu được, nhưng cũng có những nhân tố khác ảnh hưởng đến tính hiểu được như: tổ chức dữ liệu và kiểu dữ liệu được sử dụng để mô tả thiết kế.

Đo độ phức tạp có thể chỉ cung cấp một chỉ dẫn để có thể hiểu được một thành phần.
Tính kế thừa trong thiết kế hướng đối tượng ảnh hưởng đến tính hiểu được của nó trong cả cách định vị và tính chất. Tính kế thừa có thể được sử dụng để che dấu các chi tiết thiết kế, bản thiết kế có thể cũng được giới thiệu một cách trìu tượng, dễ hiểu. Mặt khác, sử dụng tính kế thừa để mở rộng thông tin trong thiết kế. Người đọc phải nhìn vào nhiều lớp đối tượng khác nhau trong hệ thống có thứ bậc sự kế thừa, do đó có thêm nhiều tính năng đượclàm để hiểu thiết kế.

4. Tính thích nghi được:

Tính thích nghi được của thiết kế là một sự đánh giá chung về tính dễ dàng để thay đổi thiết kế. Tất nhiên điều này ngụ ý rằng các thành phần của nó được ghép nối chặt chẽ. Tính thích nghi được cũng có nghĩa là thiết kế sẽ lập tư liệu tốt và các thành phần của tài liệu sẽ dễ hiểu. Nó bao gồm cả sự thực hiện, việc thực hiện một chương trình của hệ thống được viết bằng cách có thể đọc được.

Một bản thiết kế thích hợp sẽ có một tính vạch ra được mức độ cao. Điều này có ý nghĩa là có những mối quan hệ rõ ràng giữa các mức khác nhau trong thiết kế. Từ một mô hình thiết kế ta sẽ dễ dàng tìm ra mối quan hệ với mô hình khác.

Một bản thiết kế thích nghi được bao gồm việc vạch ra sự kết nối giữa các bản thiết kế khác nhau trong các tài liệu khác nhau. Tính có thể chỉ ra các nối kết được củng cố giữa các thông tin phụ thuộc lẫn nhau. Khi thực hiện một thay đổi tất cả các phần của thiết kế và các tài liệu của nó cũng bị ảnh hưởng. Nếu đây không phải là một trường hợp những thay đổi tạo ra một bản mô tả thiết kế không phổ biến tới tất cả các mô tả có quan hệ.

Với các điều kiện thuận lợi thích hợp, một thành phần có thể chứa chính nó. Các thành phần chứa chính nó là các thành phần không bị phụ thuộc vào những thành phần bên ngoài khác. Tuy nhiên điều này không có lợi để có thể dùng lại các thành phần. Người thiết kế phải đưa ra được sự cân bằng giữa những thuận lợi khi dùng lại các bộ phận và việc mất đi tính thích hợp của các thành phần.

Tự chứa mình không phải là được ghép nối một cách chặt chẽ. Một thành phần có thể được ghép nối một cách lỏng lẻo trong trường hợp nó chỉ hợp tác với các thành phần khác khi có thông điệp. Tuy nhiên các thành phần có tính kết nối lỏng lẻo có thể dựa trên các thành phần khác như là các chức năng hệ thống hoặc chức năng điều khiển lỗi. Tính thích nghi được của các thành phần có thể liên quan đến sự thay dổi các bộ phận của thành phần dựa vào các chức năng bên ngoài. Bản đặc tả những chức năng bên ngoài này cũng phải được những người thiết kế hiểu. Một trong những thuận lợil chủ yếu của tính kế thừa trong hệ thống hướng đối tượng là các thành phần có thể thích hợp được. Tính thích nghi của các bộ phận có khi không dựa trên việc sửa chữa các thành phần đang tồn tại. Hơn nữa, các thành phần mới được tạo ra kế thừa các thuộc tính và thao tác của thành phần gốc. Chỉ những thuộc tính và thao tác cần thiết mới bị sửa đổi.

Tính thích nghi được đơn giản là một lý do mà các ngôn ngữ hướng đối tượng có ảnh hưởng đến tốc độ của nguyên mẫu. Tuy nhiên một vấn đề trong tính kế thừa là những thay đổi càng được thực hiện nhiều thì càng tăng tính phức tạp. Chức năng được sao chép tại các điểm khác nhau trong mạng và trong các thành phần sẽ khó hiểu hơn. Kinh nghiệm lập trình hướng đối tượng chỉ ra rằng tính kế thừa mạng luôn phải được xem xét trước và tính phi cấu trúc (Restructured) phải giảm tính phức tạp của nó và các chức năng lặp lại.

Tích hợp các modul chương trình

Mỗi môdul riêng lẻ có những nhiệm vụ chuyên biệt, tuy nhiên nó chỉ phát huy được tác dụng khi các môdul làm việc cùng nhau trên một đường tích hợp. Điều đó cũng có lợi cho chính quá trình tích hợp, đó là các modul chuyên biệt được kết hợp với nhau để cung cấp một cách đầy đủ hơn các chức năng cho các tiến trình hoạt động. Một hệ thống được tích hợp có hiệu quả giúp ta có thể tích hợp thêm các hệ thống mới mà không làm đảo lộn các hệ thống đang tồn tại. Với một hệ thống được tích hợp, các chi phí đào tạo có khả năng giảm xuống bởi các phần mềm đang hoạt động có khả năng được tái sử dụng khi các hệ thống mới được thêm vào. Nếu các hệ thống được tích hợp có sự tương đồng thì thời gian học tập và tỷ lệ các sai sót giảm xuống đáng kể .

Sự tích hợp bao gồm các tiến trình sau:

1. Sự tích hợp của một thiết kế và một tài liệu : tài liệu được tự động sinh ra bởi các công cụ thiết kế và có thể được định dạng lại, sau đó được hợp nhất trong một tài liệu hệ thống.
2. Sự tích hợp của bản ghi chi tiết, thiết kế và các công cụ lập trình với một mô hình điều khiển. Đầu ra của các công cụ có thể điều khiển quá trình hoạt động của hệ thống. Tổ chức có thể quyết định các thay đổi về phiên bản khác nhau.

Wasserman (1990) đã đưa ra một mô hình năm mức cho việc tích hợp trong các môi trường công nghệ phần mềm:

- Tích hợp nền : Các công cụ chạy trên cùng một môi trường phần cứng và các thao tác hệ thống giống nhau.
- Tích hợp dữ liệu: Các công cụ thao tác sử dụng một mô hình dữ liệu được chia nhỏ .
- Tích hợp bản trình bày: Các công cụ đưa ra một giao diện người dùng chung.
- Tích hợp điều khiển: Các công cụ có thể hoạt động và điều khiển các thao tác của những công cụ khác.
- Sử lý tiến trình: Các công cụ chuyên biệt được giới thiệu bởi một tiến trình cụ thể và một công nghệ xử lý tương đồng.

Từ nhu cầu của người sử dụng, tích hợp có nghĩa là các môđul đồng bộ được kết hợp với nhau. Mức độ đồng bộ cũng rất đa dạng, những hệ thống tự do có thể cung cấp dữ liệu một cách hạn chế. Ngược lại trong các hệ thống được tích hợp một cách chặt chẽ thì các công cụ hoạt động một cách riêng lẻ. Các thành phần tích hợp có một ranh giới thích hợp và có sự chuyển tiếp quá độ giữa các công cụ.

I/ Tích hợp nền:

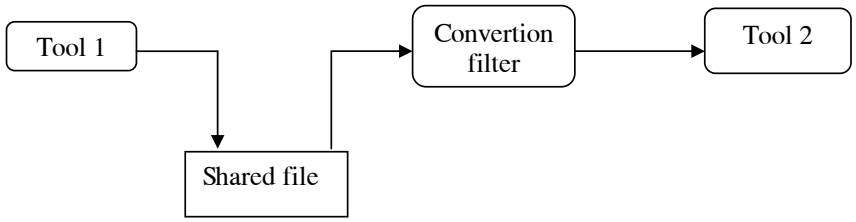
Các công cụ được tiến hành trên cùng một nền, đó có thể là một sự kết hợp hệ thống máy tính đơn, cũng có thể là sự nối mạng của các hệ thống. Sự tích hợp nền tiến hành trên một hệ thống đơn sẽ không gặp khó khăn trừ khi có một nhu cầu để tích hợp các công cụ PC và Unix
Một số vấn đề còn tồn tại với sự tích hợp nền là khi một tổ chức điều hành có sự kết hợp hỗn tạp giữa các máy tính khác nhau đang chạy trên các môi trường khác nhau. Các máy mới có thể nối mạng với các máy cũ của hệ thống, trong một vài trường hợp các hệ thống đang tồn tại không lập tức tiếp nhận các hệ thống mới, những hệ thống mới có thể không làm việc với các bản dịch cũ của hệ thống đang hoạt động. Các vấn đề của sự tích hợp nền phải được giải quyết một cách lâu dài. Nó yêu cầu hệ thống được sử dụng rộng rãi và có một tiêu chuẩn nhất định được chấp nhận rộng rãi. Điều này khó có thể xảy ra bởi một mạng lưới hỗn tạp trong tương lai là điều mà ta có thể nhìn thấy trước được.

II/ Tích hợp dữ liệu:

Hình thức đơn giản nhất của tích hợp dữ liệu là tích hợp xung quanh một file được chia khi được trợ giúp bởi hệ thống Unix, bất cứ công cụ nào cũng có thể đọc và bổ xung thêm các tính chất mới của file được sinh ra bởi các công cụ khác. Một file đơn cho phép các hình vẽ được xử lý như các file. Nó dễ dàng cung cấp các đường truyền riêng cho việc nối các tiến trình trực tiếp. Khi các giai đoạn của tiến trình được nối bởi một đường truyền, các dòng đặc tính (hoặc các nét chữ) được bỏ qua một cách trực tiếp từ tiến trình này tới một tiến trình khác mà không cần tạo file lúc đó. Các file là một quy trình vật lý đơn giản tiếp cận sự thay đổi thông tin. Tuy nhiên nếu chúng sử dụng các thông tin đã sinh ra bởi các công cụ khác thì muốn thay đổi thông tin, ta cần phải biết được về cấu trúc logic của một file. Cấu trúc logic này được đưa vào một chương trình viết riêng dành cho file, hơn nữa tất cả các công cụ đều phải tuân theo cấu trúc sau đó chúng sẽ trở thành tiêu chuẩn hoặc một công cụ sử dụng các thông tin mà nó phải biết được công cụ nào sinh ra thông tin đó.

Một chiến lược tích hợp file cơ bản dẫn đến hai lối tiếp cận điển điểm để tích hợp. Để tích hợp được mỗi thành phần của các công cụ cũng phải đồng bộ trên phương diện sắp xếp các file hay

bị thay đổi và cũng phải cung cấp một tiền đề để thay đổi file đã chia từ một file đại diện đến file khác. Sơ đồ sau chỉ ra quá trình tích hợp file:



Theo nguyên tắc, tích hợp file đã chia cần có sự thay đổi các chương trình lựa chọn được viết cho mỗi phần của các công cụ mà nó phải được tích hợp.

Các quy ước sắp xếp file có thể được chấp nhận và các công cụ được lập kế hoạch để làm việc với các quy ước này. Tuy nhiên, với các công cụ mới thì quy ước này có thể là bị hạn chế và không tương xứng. Trước đó người ta có thể bỏ qua và sử dụng cách sắp đặt của chính nó. Vì vậy rất khó có thể xây dựng được sự phù hợp giữa công cụ mới và công cụ đang hoạt động trong hệ thống.

Một sự lựa chọn tiến đến sự tích hợp dữ liệu mà nó kết hợp chặt chẽ với các thông tin có ý nghĩa về ngôn ngữ và cú pháp được chia và được dựa trên các cấu trúc dữ liệu được chia. Tích hợp xung quanh các cấu trúc dữ liệu được chia để dấu đi sự khác nhau giữa các công cụ cá nhân và người sử dụng để tạo nên một hệ thống hoàn thiện. Tuy nhiên nhờ có thể kết hợp thêm các công cụ ở trong hoàn cảnh này và bởi sự phức tạp của tập hợp dữ liệu được chọn nên bất cứ công cụ mới nào cũng phải biết được cấu trúc chi tiết của nó.

Từ nay trở đi các **workbench** đã dựa trên hình thức tích hợp có thể trở thành các hệ thống độc lập. Hình thức tích hợp này phải xuyên qua mã nguồn của các ngôn ngữ lập trình được chia.

Tích hợp thành một kho hoặc thành một hệ thống điều khiển dữ liệu là một hệ thống cơ sở dữ liệu có nhiều thuận lợi trong việc phân loại các thực thể của hệ thống, kết hợp với các thuộc tínhcủa thực thể này và thiết lập nên mối quan hệ giữa chúng.

Bản chất của hình thức tích hợp này là một giản đồ cơ sở dữ liệu chung được xác định các loại và mối quan hệ của thực thể được sử dụng. Công cụ đọc và viết dữ liệu tuỳ thuộc vào giản đồ. Nếu một công cụ muốn sử dụng dữ liệu được sản xuất bởi công cụ khác, giản đồ được sử dụng để nghiên cứu ra cấu trúc dữ liệu của công cụ đó.

Có hai khó khăn chủ yếu trong phương thức này để tích hợp dữ liệu là:

1. Các công cụ phải được viết một cách đặc biệt để sử dụng hệ thống điều khiển đối tượng các công cụ đang tồn tại không thể thường xuyên thích nghi mà không có những thiếu sót đáng kể.

2. Như một công cụ hoặc **workbench**, người sử dụng cũng phải mua hệ thống điều khiển đối tượng. Việc mua thêm rất tốn kém và khó khăn.

Nhìn lại các vấn đề này, từ đầu những năm 80 và môi trường làm việc thống nhất như PCTE (Thomas, 1980; Workman và Sowen, 1998)đã tìm ra và thực hiện. Tuy nhiên, người bán công cụ

buộc phải đưa ra một chuẩn thống nhất. Chúng hầu như thích cách tiếp cận của chính nó, điều này có nghĩa là tại một thời điểm đang thiết kế, thị trường của hệ thống CASE dựa vào hình thức tích hợp này là bị giới hạn và đáng tin cậy trong một vài tổ chức đặc biệt.

III/ Tích hợp trình bày:

Tích hợp trình bày hay tích hợp giao diện người sử dụng có nghĩa là các công cụ trong một hệ thống sử dụng phải đưa ra một bộ chuẩn chung cho giao diện người sử dụng. Các công cụ xuất hiện như nhau nên người sử dụng có thể giảm được thời gian học khi một công cụ nếu được giới thiệu như là một vài giao diện đã quen thuộc.

Có ba mức sử dụng khác nhau của tích hợp trình bày:

1. Tích hợp cửa sổ hê thống: Các công cụ được tích hợp ở mức này sử dụng các hệ thống cửa sổ mức dưới giống nhau và đưa ra một giao diện chung cho dưới tập cửa sổ lệnh. Các cửa sổ xuất hiện giống nhau và các lệnh giống nhau khi di chuyển cửa sổ hay đưa trả lại kích thước cũ của cửa sổ....

2. Tích hợp câu lệnh: Các công cụ được tích hợp ở mức này sử dụng các dạng câu lệnh giống nhau đối với cách làm có thể so sánh được. Nếu một giao diện nguyên bản được sử dụng, cú pháp của các dòng lệnh và các tham số là giống nhau cho tất cả các câu lệnh. Nếu một giao diện đồ hoạ có hệ thống menu nút được sử dụng. Các câu lệnh có thể so sánh được có cùng tên. Các danh mục của menu được đặt cùng nơi trong mỗi ứng dụng. Những lời giới thiệu giống nhau được sử dụng trong tất cả các hệ con cho các nút, các menu...

3. Tích hợp các tác động: Các ứng dụng này trong hệ thống cùng với một tác động lôi kéo trực tiếp giao diện người dùng với một đồ thị hoặc hình ảnh nguyên bản của một thực thể, tích hợp tác động có ý nghĩa là các thao tác trực tiếp giống nhau như: lựa chọn, xoá.... Được cung cấp trong một hệ thống con. Tuy nhiên nếu văn bản được lựa chọn bằng cách nhấn đúp, nó có thể lựa chọn các thực thể trong một sơ đồ bằng các cách giống nhau.

4. Tích hợp câu lênh có ý nghĩa là các ứng dụng và các chức năng điều khiển môi trường được cung cấp trong một cách đồng nhất. Ví dụ: tất cả các ứng dụng đòi hỏi các cơ cấu tuân theo người sử dụng để dừng lại và sau đó thực hiện. Ngay lập tức tất cả các ứng dụng có thể có cùng một loại nút ”quit”. Nếu các công cụ được điều khiển bằng những câu lệnh nguyên bản, các câu lệnh có thể có các định dạng tương tự hoặc giống nhau hoàn toàn và trùng tên các tham số. Tích hợp câu lệnh có thể được hoàn thành nếu sự thực hiện tuân theo một tập các quy tắc, định nghĩa các tác động lại của các hoạt động giao tiếp của người sử dụng. Những tác động này có thể bao gồm một phần của một lựa chọn từ một tập các hoạt động xen kẽ nhau, những thông tin bằng chữ hoặc bằng số được hiển thị.

Hầu hết các công cụ phần mềm mô tả các đối tượng dưới dạng sơ đồ hoặc văn bản. Tích hợp tương tác có nghĩa là các cơ cấu được sử dụng để tác động đến sơ đồ hoặc đối tượng nguyên bản. Ví dụ như: nếu văn bản được lựa chọnmột cách thông thường bằng cách đưa khoá điều khiển con trỏ ngang qua, tất cả các công cụ mà văn bản lựa chọn yêu cầu sẽ sử dụng cách tiến hành như nhau.

Cung cấp các quy tắc cho việc tích hợp tương tác là rất khó bởi số lượng các tác động xảy ra và các tiềm năng có cấu trúc sự trưng bày các đối tượng đồ hoạ. Nó là quan hệ trực tiếp để xác định vị trí tương tác giữa các văn bản không cấu trúc và các đối tượng đồ hoạ không phân loại. Điều

đó sẽ khó hơn khi văn bản hoặc sơ đồ đưa ra một thực thể có cấu trúc với các thao tác (phép toán) đầu tiên qua màn hình.

Trong các hệ thống mở, tích hợp giao diện người dùng ở trên mức hệ thống của cửa sổ sẽ khó hơn. Trong các môi trường hoặc **workbench** các công cụ được phát triển tại các thời điểm khác nhau và bằng các cách phát triển khác nhau. Một hệ thống phát triển các công cụ mới được đưa ra sẽ khó khăn hơn để duy trì sự đồng bộ trong giao diện người sử dụng. Mặc dù tất cả các quy tắc đối với việc tích hợp giao diện người dùng có thể được đưa ra, những người thiết kế quy tắc cũng không thể biết trước được mọi môi trường có thể sử dụng. Các thuận lợi mới như là sử dụng âm thanh có thể không nằm trong nguyên tắc.Vì vậy các thoả thuận giao diện người dùng được phát minh để cung cấp một cách chắc chắn, khi đó sự đồng bộ sẽ bị mất đi.

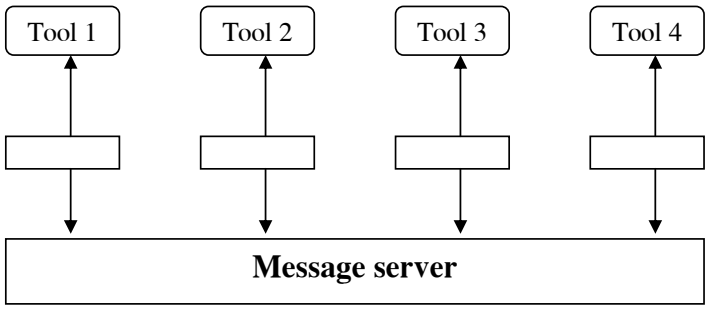
IV/ Tích hợp điều khiển:

Tích hợp điều khiển liên quan đến các cơ cấu được cung cấp cho một công cụ trong một **workbench** hoặc môi trường để điều khiển các hoạt động của các công cụ khác. Một công cụ có thể gọi các dịch vụ được cung cấp bởi công cụ khác trong hệ thống.

Một vài workbench đã phát triển các cơ cấu mục đích đặc biệt cho việc tích hợp điều khiển. Tuy nhiên một phương thức chung dựa trên thông điệp đã được chấp nhận. Nó đã được thực hiện trong các hệ thống như: Hewleft Packard's Sytbench, DEC'S FUSE và Sun's ToolTalk Brown (1993).

Trong phương thức tiếp cận bằng thông điệp, các hệ thống thay đổi thông tin. Các thông điệp này có thể cung cấp các trạng thái thông tin, báo cho các công cụ khác về những gì đang xảy ra hoặc các yêu cầu của các dịch vụ được cung cấp bởi hệ thống. Một dịch vụ thông điệp chung điều khiển truyên thông giữa các hệ thống.

Hình vẽ sau minh hoạ mô hình chung này:



Mỗi công cụ được tích hợp cung cấp một giao diện điều khiển cho phép truy nhập tới các phương tiện của công cụ. Dịch vụ thông điệp bao gồm các thông tin về giao diện của tất cả các công cụ có thể và các vùng chứa những công cụ này. Nó lấy các thông tin đã mã hoá và giải mã cho mạng chuyển tiếp.

Khi một công cụ muốn truyền thông tới một công cụ khác, nó phân tích một thông điệp sử dụng một định dạng, địa chỉ đã biết và gửi tới dịch vụ thông điệp. Công cụ không cần biết nơi đã gọi

công cụ. Tuy nhiên mô hình cung cấp một hệ thống được phân cấp nơi mà các thành phần khác nhau của hệ thống tiến hành trên các môi trường khác nhau.

Một cách logic, các công cụ phát ra các thông điệp được các công cụ khác tiếp nhận. Trong thực tế, đây là một cách tiếp cận không hiệu quả. Dịch vụ thông điệp biết các thông điệp có thể được sử lý bởi mỗi hệ thống vì vậy chỉ bỏ qua các thông điệp thích hợp.

Một minh hoạ cho cách tiếp cận này, ta xem như một hệ thống bao gồm một người biên tập thiết kế, một bộ mã và một bộ dịch. Chúng được coi như các thành phần bị chia sẻ. Khi kết thúc một phiên biên tập, các hoạt động phải tuân theo là:

- Người biên tập thiết kế gửi một thông điệp tới bộ mã để xử lý các yêu cầu thiết kế.
- Sau khi sinh mã, bộ mã gửi một thông điệp tới cả biên tập thiết kế và bộ dịch. Biên tập thiết kế kết nối file mã tới thiết kế, bộ dịch và bộ mã.
- Sau khi mã đã được sinh, bộ dịch gửi một thông điệp tới biên tập thiết kế. Nó báo cho người sử dụng biết công việc biên soạn đã hoàn thành, tích hợp điều khiển cũng đòi hỏi một vài mức tích hợp dữ liệu, vì vậy các tham số của các phép toán có thể bị thay đổi. Khuôn dạng dữ liệu cũng bị thay đổi trong khi tiến hành một thao tác mã hoá thông thường trong một ngôn ngữ định nghĩa giao tiếp (IDL). Ngôn ngữ định nghĩa giao tiếp này kết hợp thành một tập các chuẩn mà tất cả các công cụ sử dụng. Mỗi công cụ phải định dạng lại dữ liệu để chuyển thành các dạng khác.

Tuy nhiên điều này không giải quyết được các vấn đề về tích hợp dữ liệu khi một đối tượng lớn dữ liệu được chia sẻ. Nó chỉ phù hợp cho những thông điệp ngắn. Thay đổi dữ liệu có kích thước lớn được tổ chức thông qua file hoặc đối tượng. Do đó các thông điệp muốn truyền qua lại các hệ thống thông thường phải bao gồm cả việc kết nối các file mà dữ liệu được chia sẻ.

V/ Tích hợp tiến trình:

Tích hợp tiến trình nghĩa là hệ thống phải ghi nhận được các hoạt động của tiến trình, sự định pha, sự bắt buộc và các công cụ cần để cung cấp cho những hoạt động này. Hệ thống chia hoạt động theo thời gian và trong quá trình kiểm tra đòi hỏi các hoạt động có thứ tự được duy trì.

Tích hợp tiến trình đòi hỏi hệ thống duy trì một mô hình tiến trình phần mềm và sử dụng những mô hình này để điều khiển các hoạt động của tiến trình. Thực ra, các hoạt động và việc phân phát được nhận dạng, một chiến lược tích hợp được định nghĩa và các công cụ được yêu cầu để cung cấp các hoạt động đặc biệt, tất cả những điều này được ghi nhận trong mô hình và một tiến trình thông dịch hay"engine". Sau đó thông qua mô hình này để điều khiển tiến trình phần mềm.

Điều đang tồn tại hiện thời là quy tắc xử lý này không được đề ra nhưng nó cung cấp một sự hướng dẫn để làm việc trên các hoạt động tiến trình. Nó cũng thừa nhận là không phải tất cả các hoạt động đều được mô hình hoá hay cung cấp. Cung cấp các tiến trình hệ thống có "opt-outs" tuân theo một phần của tiến trình để mô tả kỹ nghệ này.

Rất nhiều hoạt động xảy ra đồng thời và điều này phải được phản ánh lại trong mô hình tiến trình, các hoạt động và sơ đồ phải phụ thuộc lẫn nhau. Mô hình tiến trình phải được hoạt động và thay đổi như thêm thông tin về việc xử lý các hoạt động đang tồn tại.

Cung cấp các tiến trình tích hợp với công nghệ CASE dựa vào việc xây dựng các mô hình tiến trình, một vài vấn đề trong quá trình tạo ra các mô hình thực tế:

- Mô hình tiến trình phần mềm được thoả thuận ở trên là một mô hình chung, chúng dựa vào việc giải thích của con người để phân thành các bộ tình huống riêng biệt. Các hoạt động và sự thực hiện chúng không được xác định một cách chi tiết trong các mô hình này. Nó là một mô hình được sử dụng để tích hợp các hoạt động.
- Đó không phải là một phương pháp thích hợp để chuẩn bị cho việc phát triển phần mềm và cả cho điều khiển đối tượng cũng như việc phát triển kỹ nghệ thay đổi tiến trình. Con người có thể dịch chuyển giữa các đối tượng quan hệ một cách nhanh chóng nếu những tình huống bất ngờ nảy sinh. Đưa vào sự linh động trong mô hình là rất khó.
- Mô hình tiến trình phân loại các sản phẩm của tiến trình phần mềm và truyền thông giữa các nhà phát triển. Việc truyền đạt các đặc tả có thể cho các nhiệm vụ có cấu trúc tốt. Tuy nhiên, sự phối hợp các mô hình trong cấu trúc lỏng lẻo, vấn đề giải quyết các nhiệm vụ thông thường trong phát triển phần mềm là rất khó xác định.

Tại thời điểm đang viết, một số **workbench** của mô hình tiến trình có thể dùng được như là tiến trình Weaver (Fernsfrom, 1992) ở đó các mô hình tiến trình có thể được tích hợp. Một số ví dụ về tiến trình thử nghiệm tập trung trong các môi trường được xây dựng (Taylor, 1988; Ferfrom, 1992) vẫn chưa phát triển trong các lĩnh vực thương mại. Sau khi nghiên cứu, Rader (1993) đã không tìm ra một ứng dụng thương mại quan trọng cho loại tích hợp này. Nó không giống tích hợp hệ thống CASE qua mô hình tiến trình sẽ được sử dụng rộng rãi trong những năm cuối thế kỷ 20.

Kiểm chứng và thẩm định

Kiểm chứng và thẩm định là tên chung cho quá trình thử nghiệm, nó đảm bảo rằng phần mềm thích hợp với các chi tiết kĩ thuật của nó và đáp ứng được yêu cầu sử dụng. Hệ thống kiểm chứng và thẩm định trong mỗi tiến trình của bộ sử lý phần mềm đang sử dụng dữ liệu trong quá trình cài đặt. Kiểm chứng và thẩm định đôi khi không tách bạch rõ ràng, chúng là những hoạt động khác nhau trong tiến trình thẩm định.

Thẩm định: Phải trả lời được câu hỏi có phải chúng ta đang tạo dựng một sản phẩm tốt? Quá trình này đòi hỏi được thử nghiệm khi chương trình đang thực hiện, thể hiện khả năng tính toán của phần mềm tương ứng.

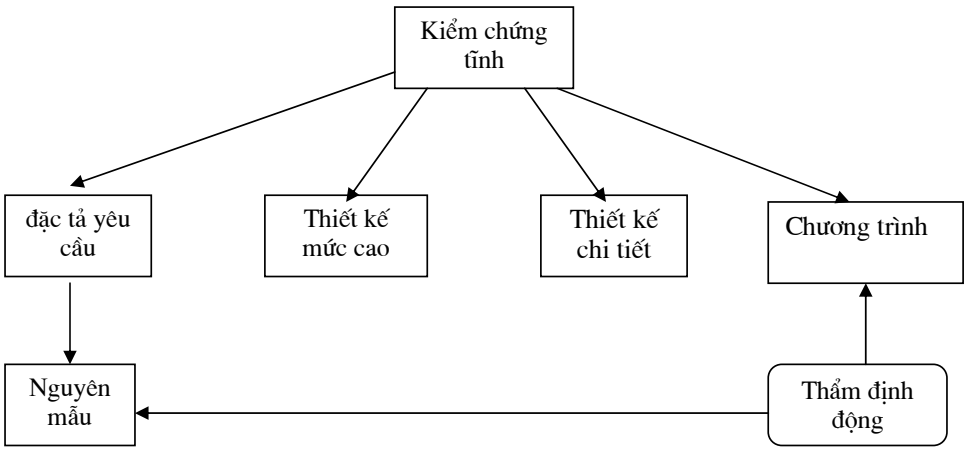
Kiểm chứng:Trả lời câu hỏi phải chăng chúng ta đang tạo dựng một sản phẩm mang tính nhất thời? Quá trình này đòi hỏi thử nghiệm xem chương trình có thích ứng với các chi tiết kỹ thuật của bản thân nó không.Tuy nhiên, những sai sót và thiếu hụt các nhu cầu đôi lúc chỉ được phát hiện khi hệ thống đã thực hiện đầy đủ.

Để đáp ứng được yêu cầu của quá trình kiểm chứng và thẩm định, người ta đưa ra hai khái niệm thử nghiệm thử nghiệm tĩnh và thử nghiệm thử nghiệm động.

Thử nghiệm tĩnh: Phân tích và thử nghiệm về hệ thống trên các mặt: đặc tả yêu cầu, biểu đồ thiết kế, mã chương trình. Phương thức này được sử dụng trong hầu hết các tiến trình xây dựng phần mềm

Thử nghiệm động: Thử nghiệm tất cả các tiến trình của hệ thống hoặc thử nghiệm diễn biến của sự thực hiện một yêu cầu. Phương thức này được sử dụng khi một nguyên mẫu hoặc một chương trình thực hiện có giá trị.

Mô hình thử nghiệm:



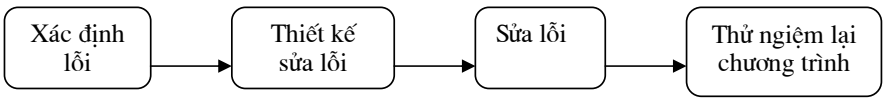
Một số người có nhận xét: Kỹ thuật thử nghiệm tĩnh có thể thay thế kỹ thuật thử nghiệm động trong kiểm chứng và thẩm định. Điều này là không thể vì thử nghiệm tĩnh chỉ có thể kiểm tra mối tương xứng giữa một chương trình và các chi tiết kỹ thuật của nó, mà chúng ta không thể chứng minh rằng phần mềm hoạt động hữu ích.

Quá trình thử nghiệm luôn cần thiết, nó bao gồm việc chương trình sử dụng các dữ liệu thực để thực hiện một yêu cầu của hệ thống. Việc xác định các thiếu sót cũng như sự không tương thích chỉ có thể nhận biết được ở sản phẩm đầu ra. Quá trình thử nghiệm có thể được tiến hành trong quá trình thực hiện chương trình, nó xác minh được rằng phần mềm đang thực hiện các công việc của giai đoạn thiết kế trước đó và cuối cùng là thử nghiệm sự thích ứng với các yêu cầu và xác định tính trung thực của hệ thống. Các giai đoạn thử nghiệm khác nhau thì sử dụng các kiểu dữ liệu khác nhau:

1. Thử nghiệm tĩnh có thể được sử dụng để thử nghiệm việc thực hiện các yêu cầu và tính trung thực của công việc đó. Các quy trình thử nghiệm thường được thiết kế để phản ánh tính thường xuyên của việc sử dụng dữ liệu vào. Sau khi tiến hành thử nghiệm, một kế hoạch về chu trình hoạt động của cả hệ thống có thể được xác định thông qua diễn biến của quá trình thử nghiệm tĩnh.

2. Kiểm tra các thiếu sót được sắp xếp để tìm ra những đoạn chương trình không thích ứng với các chi tiết kỹ thuật của nó, từ đó làm giảm các thiếu sót thực tại của hệ thống. Khi thiếu sót được tìm thấy trong chương trình, nó cần phải được phát hiện và sửa đổi, quá trình đó được gọi là gỡ lỗi (debugging). Việc phát hiện và gỡ lỗi đôi lúc được cài đặt trong quá trình giống nhau.

   Hình vẽ sau minh hoạ một tiến trình bóc lỗi:



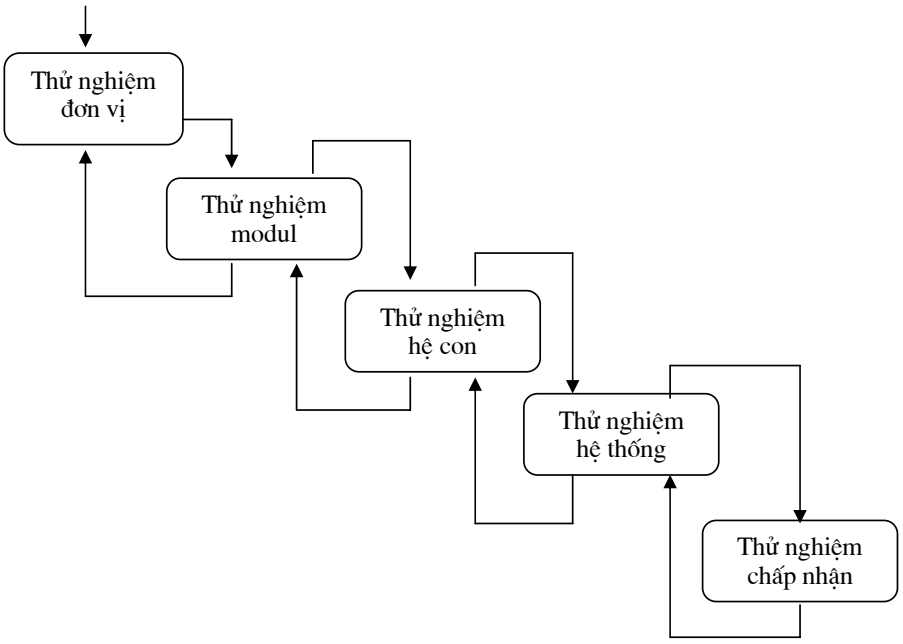
Những thiếu sót trong mã nguồn phải được xác định đúng vị trí và chương trình phải được sửa đổi để đáp ứng yêu cầu của nó. Sự thử nghiệm sau đó phải đươc thay thế để đảm bảo rằng sự thay đổi đã và đang được làm triệt để chính xác. Quá trình gỡ lỗi phải sinh ra giả thuyết về sử lý quan sát của chương trình sau đó thử nghiệm giả thuyết này để hy vọng tìm ra nguyên nhân gây lỗi ở đầu ra. Quá trình thử nghiệm giả thuyết có thể liên quan đến mã nguồn của chương trình, khi đó có thể đưa ra các hình thức thử nghiệm mới.

Chúng ta không thể đưa ra các khuôn mẫu cố định cho chương trình gỡ lỗi. Kỹ năng gỡ lỗi tìm ra những mô hình trong phần thử nghiệm đường ra, nơi mà thiếu sót được bộc lộ và sử dụng những điều nhận biết là rất quan trọng, người gỡ lỗi phải biết những thiếu sót chung của người lập trình và kết hợp những điều này dựa trên những mô hình quan sát.

Sau khi một thiếu sót của chương trình được phát hiện, nó cần phải được sửa chữa và toàn bộ hệ thống nên được thử nghiệm lại. Hình thức thử nghiệm này được gọi là " thử nghiệm lại", nó được sử dụng để thử nghiệm những thay đổi đã làm đối với một chương trình mà không đề cập đến những lỗi mới trong cả hệ thống. Theo nguyên tắc, tất cả các quá trình thử nghiệm nên được lặp lại sau mỗi lần sửa lỗi. Trong thực tế, điều này quá tốn kém. Khi thực hiện một thay đổi, ta chỉ cần thay một phần phụ của toàn bộ hệ thống để thử nghiệm dữ liệu, thử nghiệm tổ hợp và giảm sự phụ thuộc.

I/ Quá trình thử nghiệm :

Ngoài những chương trình nhỏ, ta không nên thử nghiệm các hệ thống một cách đơn lẻ, kể cả những đơn vị nguyên khối. Quá trình thử nghiệm được sử dụng rộng rãi nhất là quá trình thử nghiệm hồi quy, bao gồm năm giai đoạn:



Khi các thiếu sót được phát hiện trong bất kỳ giai đoạn nào, đều cần phải được sửa đổi và kết nối lại, đều này có thể dẫn đến việc lặp lại các giai đoạn khác của quá trình thử nghiệm, những lỗi trong tổ hợp chương trình có thể được dịch chuyển để làm rõ bước tiếp theo của quá trình thử nghiệm (hình vẽ trên) các mũi tên từ trên đỉnh các ô chỉ ra diễn biến thường

xuyên của việc thử nghiệm, những mũi tên quay vòng trở lại ô trước chỉ ra rằng quá trình thử nghiệm trước đó có thể được lặp lại. Năm giai đoạn trong quá trình thử nghiệm là:

1. Thử nghiệm đơn vị: một đơn vị chương trình cần phải được thử nghiệm để đảm bảo rằng chúng hoạt động chính xác. Mỗi đơn vị được thử nghiệm độc lập, không có các hệ thống tổ hợp khác.
2. Thử nghiệm modul: Một môdul là một tập hợp các đơn vị phụ thuộc nhau như một lớp thể, một kiểu dữ liệu chung, một vài loại tập hợp của các chức năng và quy tắc. Sau khi thống nhất các đơn vị chương trình thành một môdul, ta cần thử nghiệm cả môdul để đảm bảo rằngsự kết hợp đó được thực hiện đúng, nói cách khác, môdul vừa hoàn thành là hữu ích.
3. Thử nghiệm hệ con: các môdul lại phải kết hợp với nhau thành một hệ con, trong giai đoạn này, ta phải tiến hành kiểm tra sự hoạt động của hệ con. Hệ con có thể được thiết kế độc lập và hoàn chỉnh. Những vấn đề chung nhất được tăng lên trong hệ thống phần mềm lớn và từ đó cũng nảy sinh sự bất hoà giữa các hệ thống phụ. Chu trình thử nghiệm hệ thống phụ trước đó nên tập trung vào sự dò tìm các lỗi bề mặt phân giải bởi những diễn biến chính xác của hệ thống tương tác đó.
4. Thử nghiệm hệ thống: Hệ thống con lại được hội nhập để cấu tạo nên toàn bộ hệ thống hoàn chỉnh. Tiến trình thử nghiệm có liên quan tới việc tìm các lỗi mà nó bắt nguồn từ những tác động qua lại được thấy trước giữa những hệ con và tổ hợp hệ thống con. Nó cũng liên quan tới các yêu cầu mà hệ thống phải đáp ứng trong các bộ phận, các yêu cầu đó có thể là chức năng hoặc phi chức năng.
5. Thử nghiệm chấp nhận: Đây là giai đoạn cuối cùng trong tiến trình thử nghiệm trước khi hệ thống được đưa vào sử dụng. Hệ thống được thử nghiệm với các dữ liệu được cung cấp sẽ tìm ra những hệ thống thích hợp hơn khi sử dụng dữ liệu thử nghiệm giả, thử nghiệm chấp nhận có thể giảm các vấn đề yêu cầu. Thử nghiệm chấp nhận có hai dạng:

- Thử nghiệm α: Trước khi các hệ thống được phân phối tới khách hàng đơn lẻ, ta giao hệ thống cho người sử dụng làm nhưng có sự giám sát của các nhà cung cấp để tiếp nhận các thông tin và phát hiện sai sót xảy ra.
- Thử nghiệm β: Khi một hệ thống được đưa ra thị trường như một sản phẩm phần mềm, một tiến trình thử nghiệm thường được gọi là thử nghiệm β được sử dụng. Thử nghiệm β liên quan đến một hệ thống phân phối và đến số lượng các khách hàng tiềm năng mà họ đồng ý sử dụng hệ thống đó. Hệ thống được giao cho người sử dụng nhưng không có sự giám sát trực tiếp của nhà cung cấp, nhà cung cấp chỉ nhận được thông tin phản hồi từ phía khách hàng nếu khách hàng đó gửi các nhận xét của mình cho nhà sản xuất.

II/ Thử nghiệm hệ thống hướng đối tượng:

Sơ đồ trên dựa trên quan niệm tích hợp hệ thống, các đơn vị riêng lẻ được tích hợp để hình thành nên các modun. Các modun được tích hợp vào các hệ con và cuối cùng các hệ con lại được tích hợp vào hệ thống hoàn chỉnh. Điều cơ bản là chúng ta nên hoàn thành quá trình thử nghiệm trên từng mức độ trước khi chuyển đến cấp cao hơn.

Khi các hệ thống hướng đối tượng được phát triển, các mức độ của sự tích hợp kém độc lập và rõ ràng. Các dữ liệu và sự hoạt động được tích hợp để hình thành nên các mục tiêu và các lớp mục tiêu. Thử nghiệm các lớp mục tiêu này tương ứng với việc thử nghiệm đơn vị. Không có sự tương quan trực tiếp giữa việc thử nghiệm các modul trong hệ thống hướng đối tượng. Tuy nhiên, Musphyetal (1994) cho rằng các lớp của từng nhóm nên được kết hợp để thử nghiệm đồng bộ. Nó được gọi là thử nghiệm khối ở mức cao hơn của quá trình tích hợp, việc

thử nghiệm chuỗi thường được sử dụng. Nó dựa trên việc thử nghiệm sự tương ứng của hệ thống với một đường vào đặc biệt hoặc các tập đầu vào. Các hệ thống hướng đối tượng thường bị biến động vì vậy đây là một hình thức thử nghiệm đặc biệt để phù hợp với nhu cầu sử dụng.

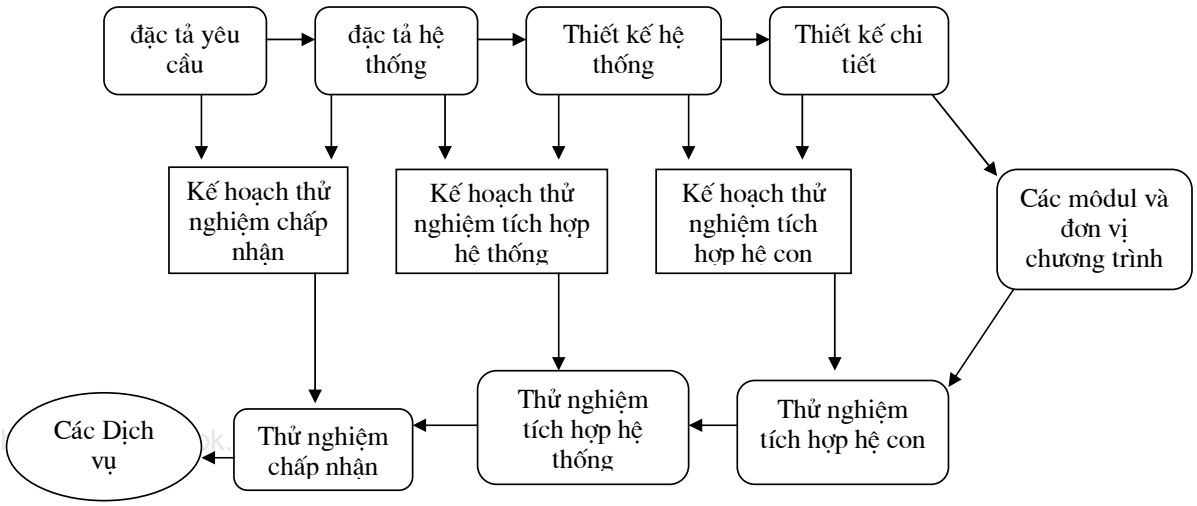
Một phương pháp liên kết các nhóm kiểm tra và mục tiêu tác động qua lại được Sorgensen và Ericleson (1994) nhận xét như sau: cần phải có một đối tượng trung gian của quá trình kiểm tra sự tích hợp được gọi là các cổng M-M(method-message). Đây là những dấu hiệu tồn tại qua một diễn biến liên tục của những sự tác động qua lại của yêu cầu. Nó dừng lại khi một hoạt động không nối tiếp hoạt động khác. Người ta cũng nhận dạng một cấu trúc liên kết mà họ gọi là Atonic System Fuctur (asf) Một ASP gồm một vài hoạt động đầu vào và tiếp đó là những diễn biến thường xuyên của cổng M-M, hoạt động được chấm dứt bằng các hoạt động đầu ra.

III/ Kế hoạch kiểm tra :

Việc kiểm tra hệ thống đòi hỏi chi phí cao, bởi một vài hệ thống lớn như hệ thống tương thích với những yêu cầu phi chức năng rất phức tạp. Một phần hai tổng chi phí có thể phải dùng để thực hiện quá trình kiểm tra. Để việc kiểm tra được tiến hành một cách hiệu quả và dễ kiểm soát chi phí, ta nên có một kế hoạch kiểm tra. Kế hoạch kiểm tra sẽ làm cho quá trình kiểm tra phù hợp hơn với mo hình sản phẩm. Nó cho phép các kỹ thuật viên lấy một bức tranh toàn cảnh của việc kiểm tra hệ thống thay thế cho công việc của chính họ. Kế hoạch kiểm tra không chỉ quản lý các tài liệu, mà chúng còn được dành cho các kỹ sư phần mềm trong việc thiết kế và thực hiện tiến trình kiểm tra hệ thống. Kế hoạch kiểm tra cũng cung cấp thông tin cho nhân viên, những người phải chịu trách nhiệm về nguồn gốc của phần cứng cũng như phần mềm.

Như các dự án khác, kế hoạch kiểm tra không phải là một tài liệu cố định, nó thường xuyên bị thay đổi bởi kiểm tra là một hoạt động phụ thuộc vào sự thực hiện có thành công hay không. Nếu một bộ phậncủa hệ thống không hoàn tất thì tiến trình kiểm tra hệ thống không thể bắt đầu được.

Sự chuẩn bị cho kế hoạch kiểm tra nên bắt đầu khi các yêu cầu của hệ thống được lập thành một hệ thống công thức và nó cần phải được phát triển một cách chi tiết. Sơ đồ sau chỉ ra mối quan hệ giữa các hoạt động của phần mềm, nó là một bình diện phẳng của cái đôi khi được gọi là mô hình thẩm định (kiểm chứng). Mô hình này là một phần kéo dài của mô hình thác đơn. Mỗi giai đoạn liên quan đến sự phát triển và có sự liên kết giữa giai đoạn xác định và làm rõ giá trị.



Các kế hoạch kiểm tra riêng lẻ nối giữa các hoạt động này. Người lập trình làm nhiệm vụ tích hợp có thể phải kiểm tra các modul và các đơn vị chương trình. Kiểm tra đơn vị là một phần của tiến trình thực hiện và nó được dự kiến tích hợp thành một hệ con cùng chức năng. Người lập trình làm việc với hệ thống khi các yêu cầu được thực hiện và họ cảm thấy việc kiểm tra đe doạ tới sự sáng tạo của họ. Về mặt tâm lý, người lập trình không muốn phá huỷ công việc của mình; bằng cách này hay cách khác, các quá trình kiểm tra được chọn sẽ không giải thích sự xuất hiện các lỗi trong hệ thống. Nếu người làm nhiệm vụ tích hợp lại phải nhận cả nhiệm vụ kiểm tra đơn vị thì nó phải được giám sát một cách chặt chẽ để đảm bảo rằng các tổ hợp được kiểm tra một cách hợp lý. Một vài tổ hợp nên được kiểm tra bởi một người kiểm tra độc lập, họ sử dụng các cách kiểm tra khác nhau mà cùng đi đến một kết quả thì có thể khẳng định rằng những phương pháp kiểm tra của người lập trình là tương đồng

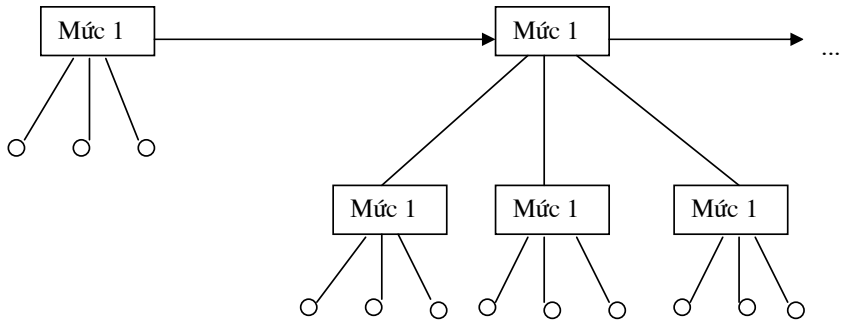
Các tiến trình tiếp theo của việc kiểm tra là người lập trình phải lập kế hoạch trước bởi nó liên quan đến số lượng các môdule được tích hợp. Một vài người kiểm tra nên đảm nhận công việc đó. Việc kiểm tra các hệ thống phụ và các môdul nên được tiến hành độc lập vì hệ thống phụ được thiết kế giống nhau. Quá trình kiểm tra sự tích hợp nên được tiến hành kết hợp với thiết kế hệ thống. Các cuộc kiểm tra sự tiếp nhận nên được thiết kế một cách chi tiết. Chúng có thể được viết vào hợp đồng cho sự phát triển hệ thống.

IV/ Các chiến lược kiểm tra

Một tiến trình kiểm tra hợp lý là một cách tiếp cận dễ dàng. Các chiến lược kiểm tra khác nhau có thể tiến hành dựa cách thức hệ thống đã được thử nghiệm và quá trình thử nghiệm đã được tiến hành. Các chiến lược kiểm tra được thảo luận trong phần này là:

1. Thử nghiệm từ trên xuống:

   Chiến lược này kiểm tra mức cao của hệ thống trước khi kiểm tra các thành phần chi tiết. Sau khi thành phần ở mức cao nhất được kiểm tra, các thành phần con của nó được tiến hành kiểm tra như trên. Quá trình xử lý này tiếp tục cho tới khi thành phần ở mức cuối cùng được kiểm tra , toàn bộ hệ thống được kiểm tra một cách hoàn thiện. Hình vẽ sau minh hoạ chiến lược kiểm tra này:



   Quá trình thử nghiệm dừng lại khi chạm mức đơn vị. Sử dụng chiến lược thử nghiệm này có một số ưu điểm và nhược điểm sau:

- Ưu điểm:

_ Các thành phần hệ thống được thử nghiệm ngay sau khi mã hoá. Vì vậy lỗi có thể sớm được phát hiện, tiết kiệm thời gian và chi phí.

_ Có thể coi một hệ chưa hoàn chỉnh nhưng đã có những chức năng chính. Từ đó có thể minh hoạ tính khả thi của hệ thống. Mức độ đưa ra có thể chỉ là mô hình mức cao (các cuống vẫn chưa được hoàn thiện).

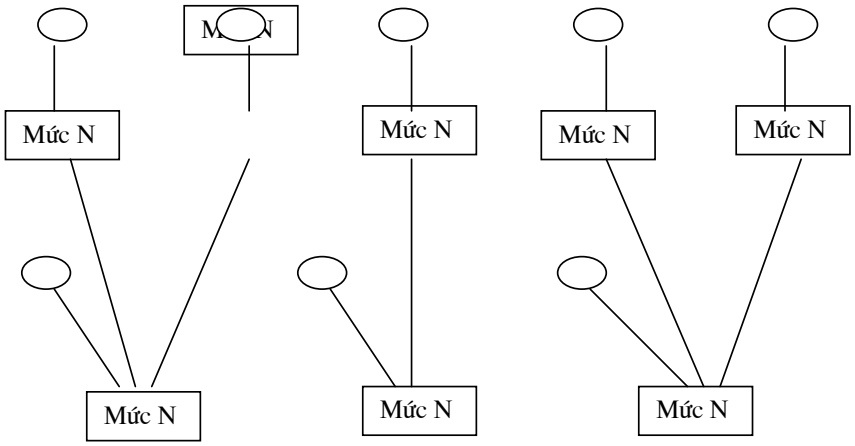
- Nhược điểm:

_ Cách thức xây dựng các cuống và chọn chức năng nào để phát triển trước là các vấn đề mà chúng ta cần quan tâm. Bên cạnh đó mô hình mức cao chưa có đầu ra rõ ràng.

2. Thử nghiệm dưới lên:

Chiến lược thử nghiệm này trái ngược với chiến lược thử nghiệm trên xuống. Nó liên quan đến việc kiểm tra các môdul tại các mức thấp hơn trong hệ thốngvà sau đó là các môdul ở mức cao hơn.

Hình vẽ sau minh hoạ chiến lược này :

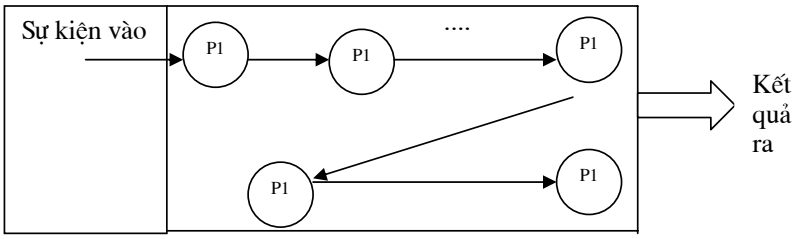


Sự thuận lợi trong chiến lược này lại chính là bất lợi của chiến lược kiểm tra trên xuống, do vậy nó có một số thuận lợi và khó khăn sau:

- Ưu điểm : không phải lo kiểm tra các cuống, các cuống thử nghiệm hoàn chỉnh và có thể phát triển được.
- Nhược điểm: những lỗi trong thiết kế cuối cùng mới được phát hiện, thường là lỗi cấu trúc, do đó tăng chi phí.

3. Chiến lươc thử nghiêm luồn sợi:

Thử nghiệm luồn sợi là một chiến lược thường được sử dụng trong các hệ thống thời gian thực. Đây là cách tiếp cận dựa trên các sự kiện khởi đầu các hoạt động của hệ thống, một cách tiếp cận có thể so sánh được sử dụng để thử nghiệm hệ thống hướng đối tượng khi chúng có mô hình hoạt động như một hệ thống điều khiển sự kiện. Bezier(1990) đã thảo luận về cách tiếp cận này một cách chi tiết nhưng ông gọi là thử nghiệm đơn tác và sau đó gọi là thử nghiệm luồn sợi.



Thử nghiệm luồn sợi là một chiến lược thử nghiệm sau khi các quá trình hoặc các đối tượng đã được thử nghiệm và tích hợp vào hệ con. Việc xử lý mới sự kiện bên ngoài "sợi" thông qua xử lý hệ thống hoặc đối tượng và theo từng bước. Thử nghiệm luồn sợi liên quan tới việc xác định và tiến hành xử lý mỗi sợi nếu được. Tất nhiên có thể chiến lược thử nghiệm luồn sợi không hoàn thành bởi số lượng các phối hợp có thể giữa dữ liệu vào và ra.

Như vậy, trong trường hợp lý tưởng ta phải xét được hầu hết các sự kiện ở đầu vào và đi qua hầu hết các quy trình bên trong hệ thống. Trên thực tế, người ta xác định các sự kiện và sợi chính để thử nghiệm .

4. Thử nghiệm áp lực:

Một vài lớp của hệ thống được thiết kế để quản lý một đối tượng. Ví dụ như, một giao tác tiến trình hệ thống có thể được thiết kế để xử lý 100 giao tác mỗi giây. Một hệ thống điều khiển có thể được thiết kế để quản lý 200 thiết bị đầu cuối. Thử nghiệm phải được thiết kế sao cho nó có thể xử lý việc tải các thông tin được mong đợi. Điều này thường liên quan đến việc lập kế hoạch, một tập các thử nghiệm được tạo ra một cách đều đặn. Loại thử nghiệm này có hai chức năng:

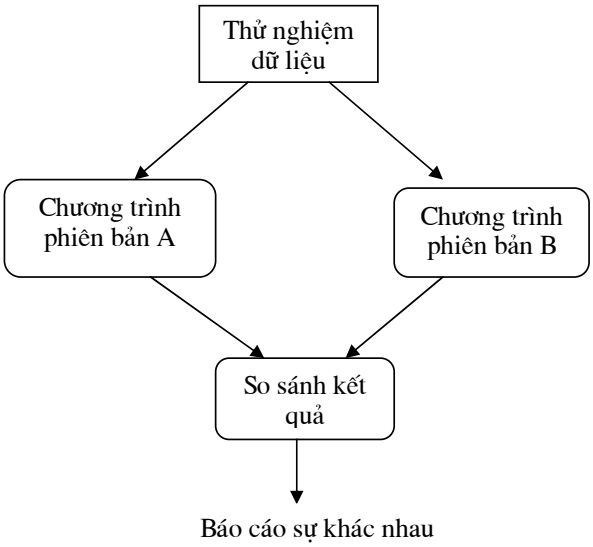
- Kiểm tra những hỏng hóc của hệ thống, các tình huống không được mong đợi hay sự quá tải của hệ thống. Trong các tình huống này, một điều quan trọng là các lỗi hệ thống không bắt nguồn từ những sai lạc về dữ liệu hoặc những mất mát của các dịch vụ người dùng.

- Áp đặt hệ thống và có thể tạo nên những ảnh hưởng không rõ ràng. Mặc dù có thể chứng tỏ rằng những điều này ảnh hưởng không giống như nguyên nhân hệ thống bị hỏng trong khi sử dụng bình thường, đó có thể là một sự hợp nhất không bình thường.

Thử nghiệm áp lực thích hợp để xây dựng hệ thống dựa trên mạng các tiến trình. Các hệ thống này thường đưa ra một vài sự thoái hoá khi chúng bị tải nặng, mạng trở nên mất tác dụng đối với các dữ liệu trong các tiến trình khác nhau phải được thay đổi

5. Thử nghiệm bach-to-bach:

Thử nghiệm bach-to-bach được sử dụng khi có nhiều hơn một phiên bản của một hệ thống được thử nghiệm. Quá trình thử nghiệm như nhau được đưa ra cho cả các phiên bản của hệ thống, sau đó đem kết quả ra so sánh. Sự khác nhau giữa các kết quả thử nghiệm chỉ rõ hơn trong hình vẽ.



Thử nghiệm back-to-back được áp dụng trong trường hợp:

- Khi một nguyên mẫu hệ thống được sử dụng
- Khi nhận ra hệ thống được phát triển sử dụng những phiên bản chương trình.
- Khi các phiên bản khác nhau của một hệ thống được phát triển cho những loại môi trường khác nhau. Một cách lần lượt, nơi một phiên bản của một hệ thống được sản xuất với một vài chức năng chung giống với các phiên bản trước đó. Thử nghiệm trên những phiên bản mới có thể được so sánh với những phiên bản cũ.

Các bước có liên quan tới chiến lược thử nghiệm bach-to-bach là:

- Chẩn bị một tập các thử nghiệm có mục đích chung.
- Chạy một phiên bản của chương trình và lưu trữ kết quả vào một vài files.
- Chạy một phiên bản chương trình khác với cùng một cách thức thử nghiệm, lưu trữ kết quả vào một file khác.

So sánh một cách tự động các file được sinh ra. Nếu các chương trình tiến hành cùng một cách, các file sosánh chỉ ra các file đầu ra. Mặc dù điều này không đảm bảo rằng chúng là hợp lệ. Nó có thể là các chương trình được sửa chữa trực tiếp. Sự khác nhau giữa những các nghiên cứu đầu vào có thể được nghiên cứu chi tiết hơn.

Thử nghiệm là một quá trình không thể thiếu trong tiến trình thiết kế phần mềm. Một sản phẩm muốn lưu hành được trên thị trường thì cần phải khẳng định rằng nó là một sản phẩm hữu ích, không thể tồn tại những sai sót do quá trình thiết kế. Tiến trình kiểm tra đòi hỏi công phu và nhiều tốn kém, vì vậy ta phải tiến hành một cách hợp lý. Các phương pháp thử nghiệm và cách thức xây dựng một hệ thống thử nghiệm được giới thiệu ở trên sẽ gúp bạn có cách tiếp cận tốt để xây dựng một tiến trình kiểm tra.

Bảo trì phần mềm

Không có một hệ thống nào không cần có những thay đổi trong quá trình thực hiện. Qua thời gian sống của hệ thống, những yêu cầu nguyên bản sẽ được sử đổi để phản ánh những thay đổi

trong nhu cầu của khách hàng và người sử dụng. Môi trường hệ thống sẽ thay đổi khi các phần cứng bổ xung. Các lỗi không được phát hiện khi thử nghiệm có thể bị phát hiện và cần được sửa chữa. Tiến trình thay đổi một hệ thống sau khi nó đã được vận hành và đưa vào sử dụng được gọi là "bảo trì phần mềm". Sự thay đổi này có thể là những thay đổi đơn giản để sửa chữa các lỗi mã hoá, sự thay đổi lớn hơn là sửa chữa các sai lầm hệ thống hoặc sự tăng cường đáng kể để sửa chữa các lỗi đặc tả hay làm thích nghi các yêu cầu mới. "Bảo trì" trong trường hợp này có nghĩa thực sự là "làm tiến triển". Nó là một tiến trình thay đổi hệ thống để đảm bảo rằng hệ thống có thể tiếp tục tồn tại.

Có ba loại khác nhau trong bảo trì phần mềm với sự khác biệt rất nhỏ:

1. Bảo trì mang tính sửa chữa: Liên quan tới các lỗi được phát hiện trong phần mềm. Các lỗi mã hoá thường đơn giản và ít tốn kém khi sửa chữa. Sửa chữa các lỗi có thể tốn kém hơn nếu nó phải viết lại các chương trình con. Sửa chữa các lỗi yêu cầu sẽ đắt hơn bởi nó cần phải thiết kế lại hệ thống một cách lớn hơn.

2. Bảo trì mang tính làm thích nghi: Có nghĩa là sự thay đổi phần mềm từ sự bổ xung thêm phần cứng hay sử dụng các hệ thống điều khiển khác nhau. Các chức năng phần mềm không thay đổi hoàn toàn.

3. Bảo trì mang tính hoàn thiện: Liên quan tới việc thực hiện các chức năng hoặc các yêu cầu phi chức mới của hệ thống. Nó được tiến hành bởi các khách hàng phần mềm cũng như sự thay đổi của tổ chức hay các nhà kinh doanh.

Rất khó có thể tìm ra được mức độ quan trọng giữa các loại bảo trì. Năm 1980 Lienz và Swanson đã khái quát rằng có khoảng 65% công việc bảo trì mang tính hoàn thiện,18% mang tính làm thích nghi và 17% mang tính sửa chữa. Con số tương tự được nhắc lại khoảng mười năm sau đó (1990) và hiện tại cũng vẫn đúng.

Lientz và Swanson đã chứng minh rằng các tổ chức lớn dành 50% ngân sách cho việc bảo trì hệ thống. Mekee (1984) đã tìm ra sự phân phối tương tự trong nỗ lực bảo trì qua các loại bảo trì khác nhau nhưng ông dự đoán rằng chi phí cho việc bảo trì trong tương lai có thể tăng từ 65% đến 75% ngân sách.

Giá thành cho việc tăng chức năng của hệ thống sau khi nó đã được đưa vào sử dụng thường tốn kém hơn khi phần mềm đang được xây dựng bởi một vài lý do sau:

1. Bảo trì thường liên quan đến vấn đề thiếu kinh nghiệm và không quen thuộc với các lĩnh vực ứng dụng. Bảo trì là một hình ảnh ít hấp dẫn trong số các kỹ nghệ phần mềm. Nó được xem như là một tiến trình kém kỹ năng hơn phát triển hệ thống
2. Chương trình đang duy trì có thể đươc phát triển từ hệ thống cũ đã được sử dụng từ nhiều năm trước mà không cần những kỹ nghệ phần mềm hiện đại. Chúng có thể không có cấu trúc và thích hợp với các yêu cầu hơn là để hiểu.
3. Những thay đổi có thể làm cho chương trình xuất hiện các lỗi mới. Những lỗi mới có thể xuất hiện bởi vì sự phức tạp của hệ thống và làm cho ta khó tiến hành những thay đổi có hiệu quả.
4. Khi hệ thống bị thay đổi, tính có cấu trúc bị suy giảm. Điều này làm cho hệ thống khó hiểu hơn và làm cho những thay đổi khó hơn khi chương trình bị giảm bớt tính gắn kết.
5. Kết nối giữa một chương trình và một tài liệu đôi khi bị mất trong quá trình bảo trì.

Tài liệu hỗ trợ việc nắm bắt được chương trình. Vấn đề đầu tiên có thể được khắc phục bởi các tổ chức chấp nhận làm sáng tỏ sự kiểm soát, điều khiển các vấn đề bảo trì. Người điều hành phải

giải thích cho các kỹ sư hiểu rằng bảo trì có giá trị ngang bằng với các giai đoạn khác của tiến trình phần mềm và thừa nhận rằng việc phát triển nó cũng như việc phát triển phần mềm. Các nhà lập trình và thiết kế giỏi nên thay đổi quan niệm và thúc đẩy việc bảo trì hệ thống.

Boelum (1983) đã gợi ý một số phương pháp có thể thúc đẩy quá trình bảo trì:

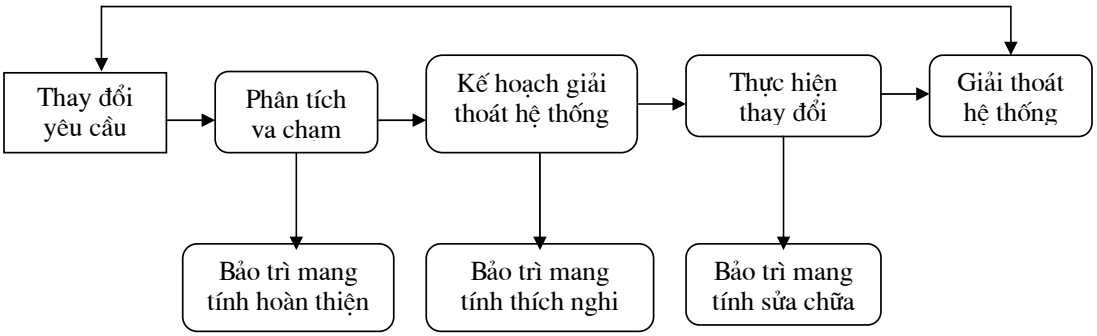
1. Kết hợp các mục tiêu của phần mềm thành mục tiêu của tổ chức.
2. Kết hợp các vấn đề bảo trì phần mềm cho việc thực hiện của tổ chức.
3. Tập hợp các nhân viên bảo trì phần mềm thành một nhóm thực hiện.
4. Tạo ra một tuỳ chọn, sử dụng ngân sách cho bảo trì một cách thích hợp. Cho nhóm bảo trì tự quyết định khi cần sửa chữa các bộ phận phần mềm. Ngăn ngừa bảo trì có nghĩa là làm các thay đổi phần mềm có tính cấu trúc nhằm làm đơn giản hoá quá trình bảo trì.
5. Cần phải tiến hành bảo trì sớm trong tiến trình phần mềm .

Điều thứ hai trong các vấn đề trên, chủ yếu là mã hoá không có cấu trúc, có thể khắc phục bằng việc sử dụng lại các kỹ nghệ và thiết kế các công nghệ khôi phục.
Các vấn đề bảo trì khác là các vấn đề về tiến trình. Cấu trúc bị suy thoái cùng với những thay đổi. Các tổ chức phải lập kế hoạch để đầu tư thêm năng lực và kinh phí trong bảo trì và phòng ngừa trong việc hỗ trợ duy trì các cấu trúc kỹ nghệ phần mềm tốt như sử dụng các thông tin hoặc sự phát triển hướng đối tượng giúp làm giảm thiểu những suy thoái về cấu trúc, nhưng năng lực của bảo trì cấu trúc vẫn được đòi hỏi. Công nghệ này cũng làm giảm tỉ lệ lỗi khi thay đổi xảy ra.

Sự sai sót từ thiết kế các tài liệu có thể là do thứ tự các điều khiển hình thức. Sự luân phiên có thể giảm thông qua cách tiếp cận ”quick fix”, bảo trì ”quick fix” có nghĩa là các chương trình được sửa đổi khi thay đổi một yêu cầu sẽ không làm thay đổi các tài liệu khác.

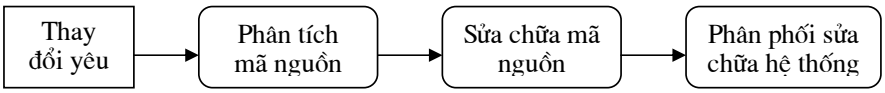
I/ Tiến trình bảo trì:

Tiến trình bảo trì được phát động bởi một tập các thay đổi yêu cầu từ phía người sử dụng hệ thống hoặc khách hàng. Giá cả và những xung đột của sự thay đổi này mang tính ước lượng, quy ước. Nếu các thay đổi đề ra được chấp thuận, một giải pháp mới của hệ thống sẽ được lập kế hoạch. Giải pháp này thường xuyên liên quan tới tiến trình bảo trì mang tính thích nghi, sửa chữa và hoàn thành. Những thay đổi này được thực hiện và phê chuẩn, một phiên bản mới của hệ thống được đưa vào khai thác. Tiến trình sau đó là lặp lại một loạt những thay đổi mới cho những thành phần mới được đưa vào sử dụng. Arthur (1988) đã chỉ ra mô hình khái quát của tiến trình này như sau:



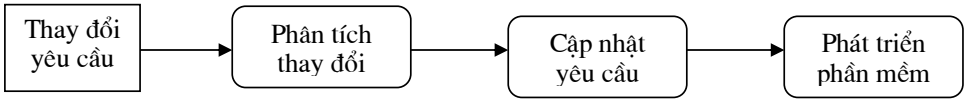
Hình vẽ 1.1

Đôi khi thay đổi yêu cầu có liên quan đến các vấn đề của hệ thống và phải được khắc phục ngay. Ví dụ như, một lỗi trong hệ thống của khách hàng phải được sửa chữa nhanh chóng để đáp ứng được các yêu cầu kinh doanh thông thường. Khuynh hướng tự nhiên trong tình huống này là thông qua một tiến trình như mô hình sau:



Việc sửa chữa ngay đôi khi rất cần thiết nếu như vấn đề đó ảnh hưởng đến hiệu lực của hệ thống. Tuy nhiên, sự nguy hiểm trong cách tiếp cận này là các tài liệu thiết kế không được cập nhật. Mã và thiết kế dần dần trở nên xa rời hệ thống. Khó có thể tránh được vấn đề này bởi các nhân viên bảo trì buộc phải giải quyết những tình thế khẩn cấp mới cho phần mềm. Nếu một kỹ sư làm những thay đổi trong mã nguồn tạm nghỉ công việc trước khi bản thiết kế được cập nhật thì sau đó anh ta sẽ rât khó khăn trong việc nắm bắt những thay đổi mới trong thiết kế.

Một vấn đề mang tính khẩn cấp hơn trong việc sửa chữa hệ thống là phải hoàn thành một cách nhanh nhất có thể được. Cách giải quyết được sử dụng đôi khi thích hợp hơn giải pháp tối ưu. Điều này thúc đẩy tiến trình già hoá phần mềm, vì vậy những thay đổi trong tương lai ngày càng khó hơn. Nếu coi bảo trì là một tiến trình độc lập, nó thường được xem như là một sự lặp lại các tiến trình phát triển. Những yêu cầu mới phải được phát triển một cách có hệ thống và được chấp thuận. Các thành phần của hệ thống phải được thiết kế lại và thực hiện. Từng phần hay toàn bộ hệ thống phải được kiểm tra lại. Mô hình được minh hoạ trong hình 1.3 sau:



Tiến trình phát triển được các tổ chức coi như những tiến trình phần mềm thông thường. Nếu sự thay đổi là cần thiết, nó sẽ được thay đổi như một phần của tiến trình phần mềm.

Việc lặp lại tiến trình phát triển có hiệu quả khi những thay đổi không lớn và có thể thực hiện đồng loạt. Nếu phải thực hiện việc sửa chữa mã nguồn, các vấn đề của tài liệu trở nên mâu thuẫn và suy giảm tính cấu trúc, điều đó có thể tránh nếu các yêu cầu thay cho việc sửa chữa những vấn đề còn tồn tại chưa được giải quyết sau khi các lỗi mã hoá đã bị cố định. Nó có thể được thực hiện lại một cách cẩn thận sau khi phân tích kỹ. Tất nhiên việc sửa chữa mã có khi bị dừng lại. Tuy nhiên một cách lựa chọn tốt hơn để giải quyết vấn đề này sẽ được tìm ra sau vài lần phân tích.

II/ Tài liệu hệ thống:

Sản xuất và bảo trì tài liệu hệ thống là một sự hỗ trợ đáng kể cho kỹ nghệ bảo trì. Tài liệu hệ thống bao gồm tất cả các tài liệu mô tả sự thực hiện của hệ thống, từ những đặc tả yêu cầu rồi

cuối cùng là chấp nhận kế hoạch kiểm tra. Tài liệu có thể được sinh ra để hỗ trợ tiến trình bảo trì bao gồm:

- Tài liệu yêu cầu và một lý do kết hợp.
- Một tài liệu mô tả các cấu trúc hệ thống.
- Đối với mỗi chương trình trong hệ thống, một bản mô tả kiến trúc là một chương trình.
- Mỗi thành phần có một đặc tả và một mô tả thiết kế riêng.
- Danh sách các chương trình mã nguồn cần được chú thích. Nếu các tên có ý nghĩa được sử dụng sẽ tránh được những vấn đề còn mơ hồ. Nhiều mã chương ttrình có thể lập tài liệu mà không cần có những bình luận mang tính chú thích. Các chương trình chú giải chỉ cần giải thích những phần mã phức tạp và cung cấp các mối quan hệ giữa các phương thức mã hoá được sử dụng.
- Tài liệu kiểm chứng mô tả mỗi chương trình được kiểm chứng như thế nào, thông tin có quan hệ với yêu cầu được kiểm chứng.
- Một hướng dẫn bảo trì hệ thống mô tả các vấn đề hiểu biết về hệ thống và mô tả những thành phần của hệ thống phụ thuộc vào phần cứng và phần mềm. Bản hướng dẫn cũng giải thích sự tiến triển của hệ thống như thế nào để thích hợp cho việc tính toán. Trong thiết kế tài liệu hệ thống cũng cần có tính cấu trúc và việc hướng dẫn tổng quát người đọc một cách quy củ và mô tả chi tiết hơn các khía cạnh của hệ thống. Một điều rất quan trọng là tài liệu phải rõ ràng và dễ đọc. Trong đó các chuẩn trình bày cần không mâu thuẫn với manul người sử dụng.

III/ Chương trình tiến triển động:

Chương trình tiến triển động nghiên cứu về những thay đổi của hệ thống. Đa số những nghiên cứu trong lĩnh vực này được thực hiện bởi Lehman vàBelady (1985). Từ những nghiên cứu này họ đã đề xuất một tập các quy tắc liên quan đến sự thay đổi hệ thống. Họ cho rằng các quy tắc này không thay đổi và cần được ứng dụng rộng rãi.

Các quy tắc của Lehman là một trong những ví dụ trong kỹ nghệ phần mềm của những học thuyết được hình thành từ quan sát thực tiễn. Đó lấy thực tiễn của các nghành khoa học khác để làm nền tảng cho những học thuyết hình thành từ quan sát, để thực hiện những quan sát một cách khách quan trong kỹ nghệ phần mềm là rất khó khăn và tốn kém.

Lehman và Belady đã khảo sát quá trình phát triển của nhiều phần mềm. Những quy tắc được đưa ra đều xuất phát từ hình thức này. Những quy tắc được chỉ ra trong bảng sau:

Quy tắc	Mô tả
Tiếp tục thay đổi	Một chương trình được sử dụng trong môi trường thế giới thực cần thiết phải có những thay đổi để có ích hơn trong môi trường ứng dụng
Tăng sự phức tạp	Khi một chương trình thay đổi, tính cấu trúc phải được duy trì và nó sẽ trở nên phức tạp hơn. Các phương tiện có sẵn bên ngoài phải được dùng cho việc duy trì và đơn giản hoá cấu trúc.
Mở rộng chương trình lớn	Tiến hoá chương trình là một tiến trình tự điều khiển. Các thuộc tính của hệ thống như kích cỡ, thời gian giữa những lần giải

Sự gắn chặt tính tổ chức Sự duy trì tính thân thiện	Thoát, và số các lỗi được phát hiện là gần như bất biến trong mỗi lần hệ thống được đưa ra sử dụng. Qua thời gian sống của chương trình, tỉ lệ phát triển xấp xỉ một hằng số và phụ thuộc vào các phương tiện có sẵn dùng cho việc phát triển hệ thống. Qua thời gian sống của hệ thống, những thay đổi sau mỗi lần đưa vào sử dụng xấp xỉ một hằng số

Quy tắc đầu tiên nói rằng bảo trì hệ thống là một tiến trình không thể tránh được. Sửa chữa lỗi chỉ là một phần trong hoạt động bảo trì. Hệ thống các yêu cầu luôn luôn thay đổi. Vì vậy một hệ thống phải phát triển nếu nó muốn duy trì tính hữu ích. Lý do cho vấn đề này là môi trường hệ thống thay đổi theo thời gian và các yêu cầu của khách hàng cũng thay đổi.

Quy tắc thứ hai là khi một hệ thống bị thay đổi, tính cấu trúc cũng giảm dần. Việc thêm các mã phải được chấp nhận, nếu không tính cấu trúc bị giảm sút và bị đảo lộn. Tiến trình bảo trì có thể bao gồm các hoạt động xây dựng lại cấu trúc một cách rõ ràng nhằm tăng khả năng thích nghi của hệ thống.

Quy tắc thứ ba có lẽ là điều thú vị nhất và đáng quan tâm nhất trong các quy tắc của Lehman. Nó dự đoán rằng mỗi hệ thống lớn đều có một động lực của chính nó, được thành lập tại giai đoạn trước của tiến trình phát triển. Điều khiển bảo trì có thể không được làm, tuy nhiên nó muốnnhững thay đổi của hệ thống được đề cập. Lehman và Belady đã dự đoán rằng quy tắc này là một kết quả của cấu trúc cơ bản và các nhân tố có tính tổ chức.

Một hệ thống vượt quá kích thước nhỏ nhất, nó tác động giống nhau lên một khối cố định, với kích cỡ hạn chế và có nhiều thay đổi lớn hơn bởi vì các thay đổi hình thành nên các lỗi mới làm giảm các chức năng của hệ thống. Nếu một thay đổi lớn được đề xuất sẽ mang lại nhiều lỗi mới và làm giảm tính hữu ích của những thay đổi bắt nguồn trong một phiên bản mới của hệ thống

Những hệ thống lớn thường được sử dụng trong các tổ chức lớn. Những tổ chức này thường có thói quan liêu, làm giảm tiến trình thay đổi và là nhân tố quyết định xem có chi ngân sách cho những hệ thống riêng lẻ hay không. Những thay đổi hệ thống lớn hơn đòi hỏi đưa ra những quyết định mang tính tổ chức và những thay đổi để dự kiến ngân sách.

Cần phải có thời gian để đưa ra quyết định. Trong thời gian đó, những vấn đề về thay đổi hệ thống được ưu tiên hơn. Ta cần phải sắp xếp những thay đổi để tiến trình phê chuẩn được bắt đầu lại thuận lợi hơn. Hơn nữa, tỷ lệ những thay dổi của hệ thống được quản lý một cách cục bộ bởi các tiến trình làm nên những quyết định của tổ chức.

Quy tắc thứ tư của Lehman cho rằng hầu hết các chương trình lớn dự kiến làm việc trong một trạng thái được gọi là “bão hoà”. Điều này cũng được dự đoán qua quy tắc thứ năm, nó gợi ý rằng sự tiến triển của chương trình phụ thuộc nhiều vào quyết định của người điều khiển. Quy tắc này xác nhận rằng sự phát triển cao độ phần mềm lớn là không thực hiện được nếu như tổng phí truyền thông giữa những người thiết kế chiếm ưu thế trong công việc của họ. Quy tắc thứ

năm của Lehman đề cập đến sự gia tăng các thay đổi trong mỗi lần hệ thống được đưa vào sử dụng.

Những quan sát của Lehman chỉ có thể nhận biết một cách chung chung. Chúng nên được đưa vào những tính toán trong việc lập kế hoạch bảo trì. Nó có thể bị bỏ qua trong một lần nào đó vì những lý do thương mại.

Nó có thể xuất hiện theo những cách hoàn toàn khác nhau và rõ ràng giữa những lần đưa các sản phẩm chương trình vào thực hiện theo các quy tắc của Lehman. Ví dụ như trong vòng mười năm Microsoft word đã thay đổi từ một bộ sử lý từ đơn giản thao tác trong bộ nhớ 256k tới một khối khổng lồ. Bây giờ nó cần nhiều Gigabit bộ nhớ và tốc độ xử lý cao. Điều này dường như trái với quy tắc thứ ba và thứ tư của Lehman.

Tuy nhiên tôi nghi ngờ rằngchương trình này không thực sự là những phiên bản có thứ tự. Hơn nữa những cái tên giống nhau còn tồn tại là những lý do thương mại nhưng chương trình cũng có những kế thừa lớn giữa những lần phát hành.

Tóm lại, bảo trì là một phần tất yếu trong tiến trình xây dựng phần mềm. Những người làm thiết kế luôn phải chú trọng tới bước này, nó đòi hỏi nhiều thời gian và công sức.