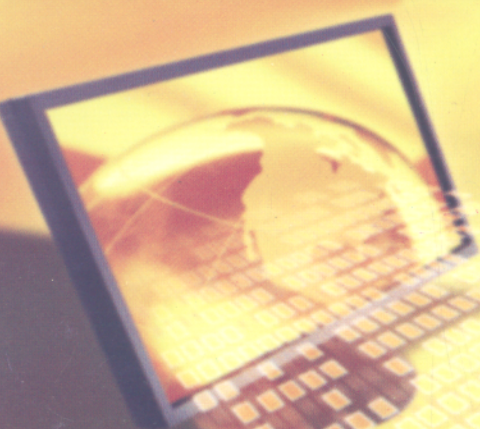


THẠC BÌNH CƯỜNG

Nhập môn CÔNG NGHỆ PHẦN MỀM



NHÀ XUẤT BẢN GIÁO DỤC

THẠC BÌNH CƯỜNG

NHẬP MÔN CÔNG NGHỆ PHẦN MỀM

NHÀ XUẤT BẢN GIÁO DỤC

Bản quyền thuộc HEVOBCO – Nhà xuất bản Giáo dục

183 – 2008/CXB/14–363/GD

Mã số : 7K754Y8 – DAI

Chương 1

GIỚI THIỆU CHUNG VỀ PHẦN MỀM

1.1. Định nghĩa chung về phần mềm

Trong ba thập kỷ đầu tiên của thời đại tính toán, thách thức chủ yếu là phải phát triển ứng dụng phần cứng máy tính để làm giảm bớt giá thành xử lý và lưu trữ dữ liệu. Trong suốt thập kỷ 80, tiến bộ trong vi điện tử đã làm phát sinh năng lực tính toán mạnh hơn với giá thành thấp đáng kể. Ngày nay vấn đề đã khác đi. Thách thức chủ yếu là phải cải tiến chất lượng và giảm giá thành của các giải pháp dựa trên máy tính – giải pháp được cài đặt bằng phần mềm.

Vậy phần mềm là gì?

Định nghĩa 1:

Phần mềm là tập hợp:

1. Các lệnh (chương trình máy tính) khi được thực hiện sẽ cung cấp những chức năng và kết quả mong muốn.
2. Cấu trúc dữ liệu làm cho chương trình thao tác thông tin thích hợp.
3. Các tư liệu mô tả thao tác và cách sử dụng chương trình.

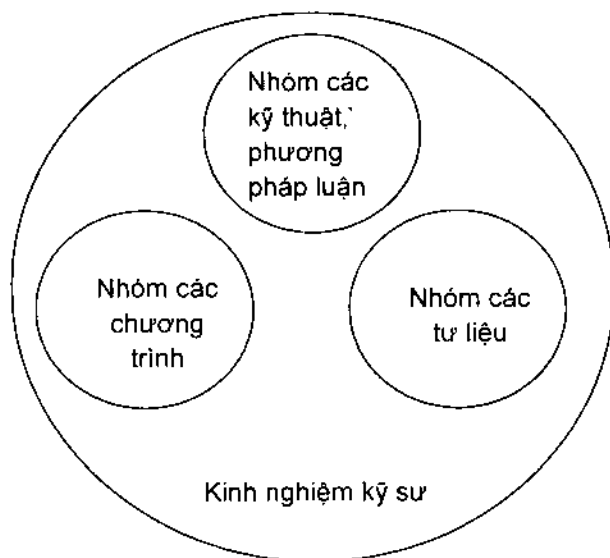
Định nghĩa 2:

Trong một hệ thống máy tính, nếu trừ bỏ đi các thiết bị và các loại phụ kiện thì phần còn lại chính là các phần mềm.

Nghĩa hẹp: phần mềm là dịch vụ chương trình để tăng khả năng xử lý của phần cứng máy tính.

Nghĩa rộng: phần mềm là tập tất cả các kỹ thuật ứng dụng để thực hiện những dịch vụ chức năng cho các mục đích nào đó bằng phần cứng.

Tóm lại : Phần mềm được biểu diễn như hình 1.1.



Hình 1.1. Phần mềm là gì?

Trong đó:

- Nhóm các kỹ thuật, phương pháp luận bao gồm:
 - Các khái niệm và trình tự cụ thể hoá của một hệ thống;
 - Các phương pháp tiếp cận và giải quyết vấn đề;
 - Các trình tự thiết kế và phát triển được chuẩn hoá (quy trình);
 - Các phương pháp đặc tả, yêu cầu, thiết kế hệ thống, thiết kế chương trình, kiểm thử, toàn bộ quy trình quản lý phát triển phần mềm.
- Nhóm các chương trình là phần giao diện với phần cứng, con người từ các nhóm lệnh chỉ thị cho máy tính biết trình tự thao tác xử lý dữ liệu.
- Nhóm các tư liệu:
 - Những tài liệu hữu ích, có giá trị cao và rất cần thiết để phát triển, vận hành và bảo trì phần mềm.
 - Để tạo ra phần mềm với độ tin cậy cao cần phải tạo ra các tư liệu chất lượng cao: đặc tả yêu cầu, mô tả thiết kế từng loại, điều kiện kiểm thử, thủ tục vận hành, hướng dẫn thao tác.

- Những yếu tố khác:
 - Sản xuất phần mềm phụ thuộc rất nhiều vào con người (kỹ sư phần mềm). Khả năng hệ thống hoá trừu tượng, khả năng lập trình, kỹ năng công nghệ, kinh nghiệm làm việc, tâm bao quát... khác nhau ở mỗi người.
 - Phần mềm phụ thuộc nhiều vào ý tưởng và kỹ năng của con người/nhóm tác giả.

1.2 . Các đặc tính của phần mềm

Để hiểu được khái niệm phần mềm, cần xem xét các đặc trưng của phần mềm làm cho nó khác biệt với những thứ khác mà con người đã xây dựng. Khi phần cứng được xây dựng, tiến trình sáng tạo của con người (phân tích, thiết kế, xây dựng, kiểm thử) cuối cùng được dịch thành dạng vật lý. Phần mềm là hệ thống logic, không phải là hệ thống vật lý. Do đó, phần mềm có các đặc trưng khác với các đặc trưng của phần cứng:

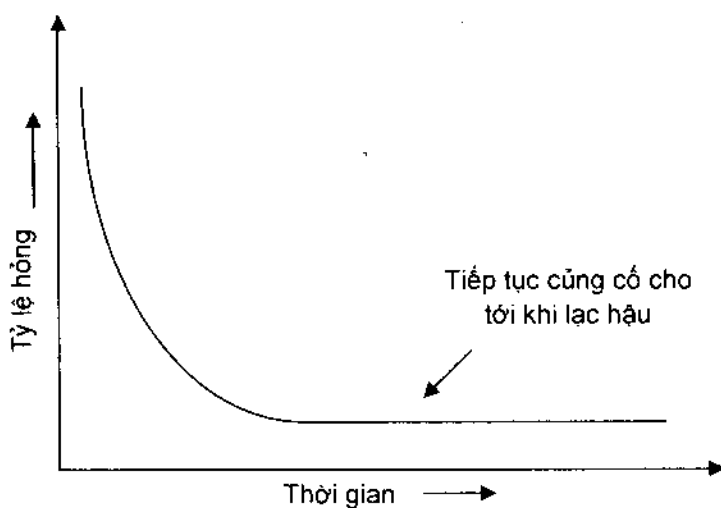
Phần mềm được phát triển hay được kỹ nghệ hoá, nó không được chế tạo theo nghĩa cổ điển.

Mặc dù có một số điểm tương đồng giữa phát triển phần mềm và chế tạo phần cứng nhưng hai hoạt động này về cơ bản là khác nhau. Trong cả hai hoạt động này, chất lượng cao sẽ đạt được nếu thiết kế tốt, nhưng giai đoạn chế tạo phần cứng có thể đưa ra các vấn đề chất lượng, điều mà không tồn tại hay dễ sửa đổi ở phần mềm. Cả hai hoạt động này đều phụ thuộc vào con người nhưng mối quan hệ giữa người được áp dụng và công việc là hoàn toàn khác nhau.

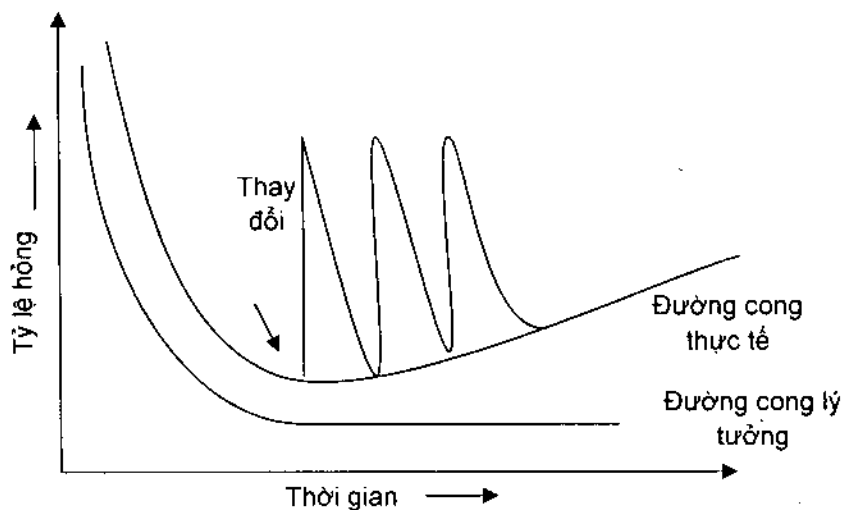
Phần mềm không hỏng đi

Phần mềm không bị tác động bởi môi trường – yếu tố vốn gây cho phần cứng bị mòn cũ đi. Do đó về lý thuyết, đường cong tỷ lệ hỏng hóc phần mềm có dạng như hình 1.2.

Những khiếm khuyết chưa được phát hiện sẽ làm cho chương trình có tỷ lệ hỏng hóc cao ngay từ ban đầu khi mới sử dụng. Tuy nhiên, những khiếm khuyết này được sửa đổi và đường cong trở nên phẳng như hình vẽ. Phần mềm không mòn cũ đi nhưng nó bị “suy thoái”. Điều này dường như mâu thuẫn nhưng có thể được giải thích rõ ràng nhất trong hình 1.3.



Hình 1.2. Đường cong hỏng hóc phần mềm (lý tưởng)



Hình 1.3. Đường cong hỏng hóc thực tế của phần mềm

Trong quá trình sử dụng, phần mềm sẽ có nhiều thay đổi. Khi thay đổi được thực hiện, có thể một số khiếm khuyết mới sẽ xuất hiện, làm cho đường cong tỷ lệ hỏng hóc trở thành có đầu nhọn như hình vẽ. Trước khi đường cong trở về với tỷ lệ hỏng hóc ổn định ban đầu thì một thay đổi khác lại được thực hiện và làm cho đường cong phát sinh đỉnh nhọn một lần nữa.

Dần dần, mức tỷ lệ hỏng hóc tối thiểu bắt đầu nâng lên – phần mềm bị thoái hoá do những thay đổi.

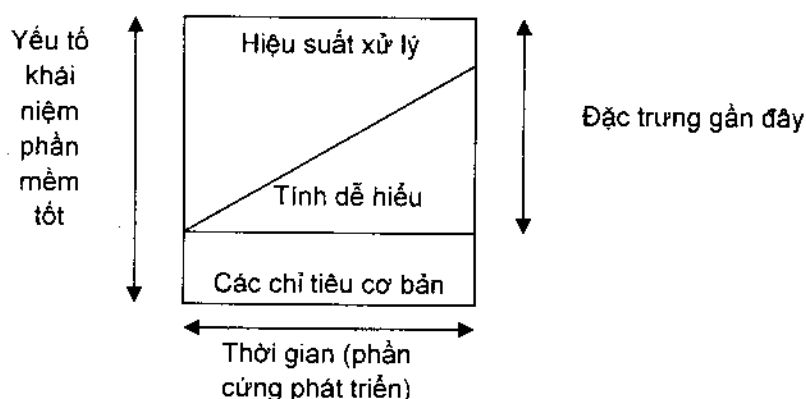
Một khía cạnh khác của sự mòn cũ sẽ minh hoạ cho sự khác biệt giữa phần cứng và phần mềm : Khi một yếu tố phần cứng bị mòn cũ đi, nó liền được thay thế. Nhưng không có phần thay thế cho phần mềm. Mọi hỏng hóc phần mềm đều chỉ ra lỗi thiết kế hay trong quá trình chuyển thiết kế thành mã máy thực hiện được. Do đó, việc bảo trì phần mềm có độ phức tạp phụ thêm đáng kể so với bảo trì phần cứng.

Phần lớn phần mềm đều được xây dựng theo các đơn đặt hàng, ít khi được lắp ráp từ các thành phần có sẵn.

Nói chung không có danh mục các thành phần phần mềm. Có thể đặt hàng phần mềm nhưng chỉ như một đơn vị hoàn chỉnh, không phải là những thành phần có thể lắp ráp lại thành chương trình mới. Mặc dù có nhiều bài viết đề cập tới vấn đề tái sử dụng phần mềm, nhưng chúng ta cũng mới chỉ thu được rất ít hiệu quả từ việc tái sử dụng này.

1.3. Thế nào là một phần mềm tốt ?

Một phần mềm tốt phải thoả mãn:



Hình 1.4. Thế nào là một phần mềm tốt?

Trong đó :

- Các chỉ tiêu cơ bản bao gồm:

- Phản ánh đúng yêu cầu người dùng.
- Chứa ít lỗi tiềm tàng.
- Giá thành không vượt quá ước lượng ban đầu.
- Dễ vận hành sử dụng.
- Tính an toàn, độ tin cậy cao.
- Hiệu suất xử lý cao:
 - Hiệu suất thời gian tốt: Độ phức tạp tính toán thấp, thời gian quay vòng ngắn, thời gian hồi đáp nhanh.
 - Sử dụng các tài nguyên như CPU, RAM, HDD, Internet một cách hữu hiệu.
- Tính dễ hiểu:
 - Kiến trúc và cấu trúc dễ hiểu.
 - Dễ kiểm tra, kiểm chứng và bảo trì.
 - Có tài liệu mô tả yêu cầu, điều kiện kiểm thử, vận hành, bảo trì với chất lượng cao.

Tính dễ hiểu ngày càng trở thành chỉ tiêu quan trọng với phần mềm.

Tuy nhiên để xây dựng phần mềm tốt, ta phải chú ý các điểm sau:

- Không có phương pháp mô tả rõ ràng định nghĩa yêu cầu của người dùng, khi bàn giao sản phẩm dễ phát sinh những trục trặc.
- Với những phần mềm có quy mô lớn, tư liệu đặc tả cố định thời gian dài, sẽ khó đáp ứng nhu cầu thay đổi của người dùng một cách kịp thời trong thời gian đó.
- Nếu không có phương pháp luận thì chất lượng phần mềm sẽ suy giảm.
- Nếu không có chuẩn về tư liệu quy trình sản xuất phần mềm thì những đặc tả không rõ ràng sẽ làm giảm chất lượng phần mềm.
- Nếu không kiểm thử tính đúng đắn của phần mềm ở từng giai đoạn mà chỉ kiểm tra giai đoạn cuối, khi đó nếu phát hiện ra lỗi thì bàn giao sản phẩm sẽ không đúng hạn.
- Coi trọng lập trình hơn thiết kế làm cho chất lượng phần mềm giảm.
- Coi thường tái sử dụng phần mềm làm cho năng suất lao động giảm.

- Phần lớn trong quy trình phát triển phần mềm do con người thực hiện cũng làm năng suất lao động giảm.
- Không chứng minh tính đúng đắn của phần mềm làm giảm độ tin cậy của phần mềm.
- Chuẩn một phần mềm tốt không thể đo được một cách định lượng, do vậy không thể đánh giá được một hệ thống đúng đắn hay không.
- Khi đầu tư nhân lực lớn vào bảo trì sẽ làm giảm hiệu suất lao động của nhân viên.
- Công việc bảo trì kéo dài làm giảm chất lượng của tư liệu và ảnh hưởng xấu đến các công việc khác.
- Quản lý dự án lỏng lẻo kéo theo quản lý lịch trình cũng không rõ ràng.
- Không có tiêu chuẩn để ước lượng nhân lực và dự toán sẽ làm kéo dài thời hạn vượt kinh phí của dự án.

1.4. Ứng dụng phần mềm

Phần mềm có thể được áp dụng trong bất kỳ tình huống nào bao gồm một tập các bước thủ tục (như một thuật toán) đã được xác định trước (các phần mềm ngoại lệ với quy tắc này là các phần mềm hệ chuyên gia và các phần mềm mạng nơron). Nội dung thông tin và tính tất định là các nhân tố quan trọng trong việc xác định bản chất ứng dụng của phần mềm. Nội dung thể hiện ý nghĩa và hình dạng của thông tin vào và ra. Tính tất định thông tin cho biết việc tiên đoán trước trật tự và thời gian của thông tin.

Các lĩnh vực phần mềm sau sẽ chỉ ra những phạm vi ứng dụng rộng rãi của phần mềm:

Phần mềm hệ thống: Phần mềm hệ thống là tập hợp các chương trình được viết để phục vụ cho chương trình khác. Phần mềm hệ thống (trình biên dịch, trình soạn thảo, các tiện ích quản lý tệp...) xử lý các cấu trúc thông tin phức tạp nhưng xác định. Các ứng dụng hệ thống khác (như thành phần hệ điều hành, bộ xử lý viễn thông) có dữ liệu thường không xác định. Lĩnh vực phần mềm hệ thống được đặc trưng chủ yếu bởi tương tác với hệ thống máy tính; sử dụng nhiều trong các hệ thống nhiều người dùng, thao

tác tương tranh đòi hỏi lập lịch, dùng chung tài nguyên và các quản lý tiến trình phức tạp; cấu trúc dữ liệu phức tạp và nhiều giao diện ngoài.

Phần mềm thời gian thực: Phần mềm điều phối, phân tích, kiểm soát các sự kiện của thế giới thực khi chúng xuất hiện gọi là phần mềm thời gian thực. Các yếu tố của phần mềm thời gian thực bao gồm một thành phần thu thập dữ liệu để thu thập và định dạng thông tin từ môi trường bên ngoài, một thành phần phân tích biến đổi thông tin theo yêu cầu của ứng dụng, một thành phần kiểm soát/đưa ra để đáp ứng với môi trường bên ngoài, một thành phần điều phối để điều hoà các thành phần khác sao cho có thể duy trì đáp ứng thời gian thực điển hình. Hệ thống thời gian thực phải đáp ứng những ràng buộc thời gian chặt chẽ.

Phần mềm nghiệp vụ: Xử lý thông tin nghiệp vụ là lĩnh vực ứng dụng phần mềm lớn nhất. Các hệ thống rời rạc (như tính lương, kế toán thu/chi, quản lý kho) đã phát triển thành các phần mềm hệ thống tin quản lý thâm nhập vào một hay nhiều cơ sở dữ liệu lớn chứa thông tin nghiệp vụ. Những ứng dụng trong lĩnh vực này cấu trúc lại dữ liệu hiện có theo cách thuận tiện cho các thao tác nghiệp vụ hay quản lý. Bên cạnh các ứng dụng xử lý dữ liệu quy ước, các ứng dụng phần mềm nghiệp vụ còn bao gồm cả các tính toán tương tác (như xử lý các giao dịch cho các điểm bán hàng).

Phần mềm khoa học và công nghệ: Phần mềm khoa học và công nghệ được đặc trưng bởi các thuật toán “máy nghiền số”. Các ứng dụng mở rộng từ thiên văn cho đến núi lửa, từ phân tích về ô tô cho tới sự biến động quỹ đạo tàu con thoi, từ sinh học phân tử đến chế tạo tự động. Tuy nhiên, những ứng dụng mới trong lĩnh vực khoa học và công nghệ đang “di chuyển nhanh” ra khỏi các thuật ngữ quy ước. Thiết kế có sự trợ giúp của máy tính (CAD), mô phỏng hệ thống và những ứng dụng tương tác khác đã bắt đầu kế tục các đặc trưng thời gian thực và thậm chí cả phần mềm hệ thống.

Phần mềm nhúng: Các sản phẩm thông minh đã trở nên thông dụng. Phần mềm nhúng nằm trong bộ nhớ chỉ đọc và được dùng để điều khiển các sản phẩm và các hệ thống cho người tiêu dùng và thị trường công nghiệp. Phần mềm nhúng có thể thực hiện các chức năng rất giới hạn như điều khiển bàn phím cho lò vi sóng hay đưa ra các khả năng điều khiển và vận hành như chức năng số hoá trong ô tô, kiểm tra xăng, hiển thị bảng đồng hồ, hệ thống phanh...

Phần mềm máy tính cá nhân: Xử lý văn bản, đồ hoạ máy tính, quản trị cơ sở dữ liệu, các ứng dụng tài chính cá nhân và nghiệp vụ, mạng bên ngoài hay thậm chí nhập cơ sở dữ liệu chỉ là một số lĩnh vực trong hàng trăm ứng dụng. Trong thực tế, phần mềm máy tính cá nhân biểu thị cho một số thiết kế giao diện người – máy được cải tiến nhiều nhất trong các phần mềm.

Phần mềm trí tuệ nhân tạo (AI): Phần mềm trí tuệ nhân tạo dùng các thuật toán phi số để giải quyết các vấn đề phức tạp mà tính toán hay phân tích trực tiếp không quản lý được. Hiện nay lĩnh vực trí tuệ nhân tạo hoạt động mạnh nhất là các hệ chuyên gia, còn gọi là các hệ cơ sở tri thức. Tuy nhiên, các lĩnh vực áp dụng khác của phần mềm trí tuệ nhân tạo còn là nhận dạng (hình ảnh và tiếng nói), chứng minh định lý và trò chơi. Trong những năm gần đây, một nhánh mới của phần mềm trí tuệ nhân tạo gọi là mạng nơron nhân tạo, đã phát triển. Mạng nơron mô tả cấu trúc việc xử lý trong não bộ và cuối cùng tạo ra lớp phần mềm mới có thể nhận dạng các mẫu phức tạp và học từ “kinh nghiệm quá khứ”.

Chương 2

CÔNG NGHỆ PHẦN MỀM

2.1. Định nghĩa công nghệ phần mềm

Có nhiều định nghĩa công nghệ phần mềm được đưa ra:

Bauer (1969): Công nghệ phần mềm là việc thiết lập và sử dụng các nguyên lý công nghệ đúng đắn để thu được phần mềm vừa kinh tế, tin cậy vừa làm việc hiệu quả trên các máy thực.

Parnas (1987): Công nghệ phần mềm là việc xây dựng phần mềm nhiều phiên bản bởi nhiều người.

Ghezzi (1991): Công nghệ phần mềm là một lĩnh vực của khoa học máy tính, liên quan đến xây dựng các hệ thống phần mềm vừa lớn vừa phức tạp bởi một hay một số nhóm kỹ sư.

IEEE (1993) : Công nghệ phần mềm là:

1. Việc áp dụng phương pháp tiếp cận có hệ thống, bài bản và được lượng hoá trong phát triển vận hành và bảo trì phần mềm.
2. Nghiên cứu các phương pháp tiếp cận nói trên.

Pressman (1995): Công nghệ phần mềm là sự phát triển của công nghệ phần cứng và hệ thống. Nó là một tập gồm ba yếu tố chủ chốt: Phương pháp, công cụ và thủ tục giúp người quản lý kiểm soát được tiến trình phát triển phần mềm và cung cấp cho người hành nghề một nền tảng để xây dựng phần mềm chất lượng cao có hiệu quả.

Các phương pháp công nghệ phần mềm đưa ra các “cách làm” về mặt kỹ thuật để xây dựng phần mềm. Các phương pháp này bao gồm một phạm vi các nhiệm vụ, bao gồm: Lập kế hoạch và ước lượng dự án, phân tích yêu cầu hệ thống và phần mềm, thiết kế cấu trúc dữ liệu, kiến trúc chương trình và thủ tục thuật toán, mã hoá, kiểm thử và bảo trì. Các phương pháp cho kỹ nghệ phần mềm thường đưa ra các ký pháp đồ hoạ hay hướng ngôn ngữ đặc biệt và đưa ra một tập tiêu chuẩn về chất lượng phần mềm.

Các công cụ công nghệ phần mềm cung cấp sự hỗ trợ tự động hay bán tự động cho các phương pháp. Ngày nay đã có các công cụ hỗ trợ cho từng phương pháp được nêu trên. Khi các công cụ được tích hợp đến mức thông tin do công cụ này tạo ra có thể được dùng cho các công cụ khác thì hệ thống hỗ trợ cho việc phát triển phần mềm được thiết lập và còn được gọi là công nghệ phần mềm có sự trợ giúp của máy tính CASE. CASE tổ hợp phần mềm, phần cứng và cơ sở dữ liệu kỹ nghệ phần mềm (cấu trúc dữ liệu chứa các thông tin quan trọng về việc phân tích, thiết kế, mã hoá và kiểm thử) để tạo ra môi trường kỹ nghệ phần mềm, điều này tương tự như thiết kế có máy tính hỗ trợ/kỹ nghệ có máy tính hỗ trợ (CAD/CASE) cho phần cứng.

Các thủ tục công nghệ phần mềm là chất keo dán các phương pháp và công cụ lại với nhau, làm cho chúng được sử dụng hợp lý và đúng hạn trong việc phát triển phần mềm máy tính. Thủ tục xác định trình tự các phương pháp sẽ được áp dụng, những sản phẩm cần bàn giao (tài liệu, báo cáo, mẫu...) cần cho việc kiểm soát để đảm bảo chất lượng và điều hoà thay đổi, xác định những cột mốc để cho người quản lý phần mềm nắm được tiến độ.

2.2. Vòng đời phần mềm

Vòng đời phần mềm là thời kỳ tính từ khi phần mềm được sinh ra cho đến khi chết đi (từ lúc hình thành đáp ứng yêu cầu, vận hành bảo dưỡng cho đến khi loại bỏ không được sử dụng). Quy trình phát triển phần mềm bao gồm 3 giai đoạn chính: Xác định, phát triển và bảo trì. Ba giai đoạn này xuất hiện trong tất cả các công việc phát triển phần mềm, không phụ thuộc vào miền ứng dụng, cỡ dự án hay độ phức tạp.

- **Giai đoạn xác định** : Trong giai đoạn xác định, người phát triển phần mềm tập trung vào xác định thông tin nào cần được xử lý, chức năng và hiệu năng nào cần có, giao diện nào cần được thiết lập, ràng buộc thiết kế nào hiện có và tiêu chuẩn hợp lệ nào cần có để xác định ra một hệ thống thành công. Yêu cầu chủ chốt của hệ thống và phần mềm được xác định. Mặc dù các phương pháp được áp dụng trong giai đoạn xác định thay đổi tùy theo khuôn cảnh công nghệ phần mềm, nhưng có ba bước luôn xuất hiện dưới dạng:

Phân tích hệ thống: Xác định ra vai trò của từng phần tử trong hệ thống dựa trên máy tính, chỉ ra vai trò của phần mềm.

Lập kế hoạch dự án phần mềm: Khi phạm vi của phần mềm được thiết lập, rủi ro được phân tích, tài nguyên được cấp phát, chi phí được ước lượng thì phải xác định nhiệm vụ công việc và lập lịch.

Phân tích yêu cầu: Phạm vi được xác định cho phần mềm sẽ đưa ra chiều hướng nhưng cần phải có thêm việc xác định chi tiết lĩnh vực thông tin và chức năng trước khi công việc có thể bắt đầu.

- **Giai đoạn phát triển :** Trong khi xác định, người phát triển cố gắng xác định cách cấu trúc dữ liệu và kiến trúc phần mềm cần được thiết kế, cách các chi tiết thủ tục được cài đặt, cách dịch thiết kế vào ngôn ngữ lập trình hay ngôn ngữ phi thủ tục và cách thực hiện kiểm thử. Phương pháp được áp dụng trong giai đoạn phát triển sẽ thay đổi nhưng có 3 bước đặc thù bao giờ cũng xuất hiện dưới dạng:

Thiết kế phần mềm: Thiết kế dịch các yêu cầu về phần mềm thành một tập hợp các biểu diễn (dựa trên đồ hoạ, bảng hay ngôn ngữ) mô tả cấu trúc dữ liệu, kiến trúc, thủ tục thuật toán và đặc trưng giao diện.

Mã hoá: Các biểu diễn thiết kế phải được dịch thành ngôn ngữ nhân tạo (ngôn ngữ có thể là ngôn ngữ lập trình quy ước hoặc ngôn ngữ phi thủ tục dùng trong ngôn ngữ thế hệ 4 – 4GT) mà kết quả sẽ tạo ra là các lệnh thực hiện được trên máy tính. Bước mã hoá thực hiện việc này.

Kiểm thử phần mềm: Mỗi khi phần mềm đã được cài đặt dưới dạng mã máy thực hiện được cần phải kiểm thử nó để phát hiện các khiếm khuyết khi vận hành, trong logic và trong cài đặt.

- **Giai đoạn bảo trì :** Tập trung vào những thay đổi gắn với việc sửa lỗi, thích ứng khi môi trường phần mềm tiến hoá và nâng cao do yêu cầu người dùng. Giai đoạn bảo trì áp dụng lại các bước của giai đoạn xác định và phát triển nhưng làm việc đó trong hoàn cảnh phần mềm hiện có. Có ba kiểu thay đổi gặp phải trong giai đoạn bảo trì:

Sửa đổi: Phần mềm luôn có lỗi tiềm tàng. Trong quá trình sử dụng, cho dù các hoạt động bảo đảm chất lượng phần mềm là tốt nhất, khách hàng vẫn có thể sẽ phát hiện ra khiếm khuyết trong phần mềm. Bảo trì sửa đổi thay đổi phần mềm để sửa các khiếm khuyết.

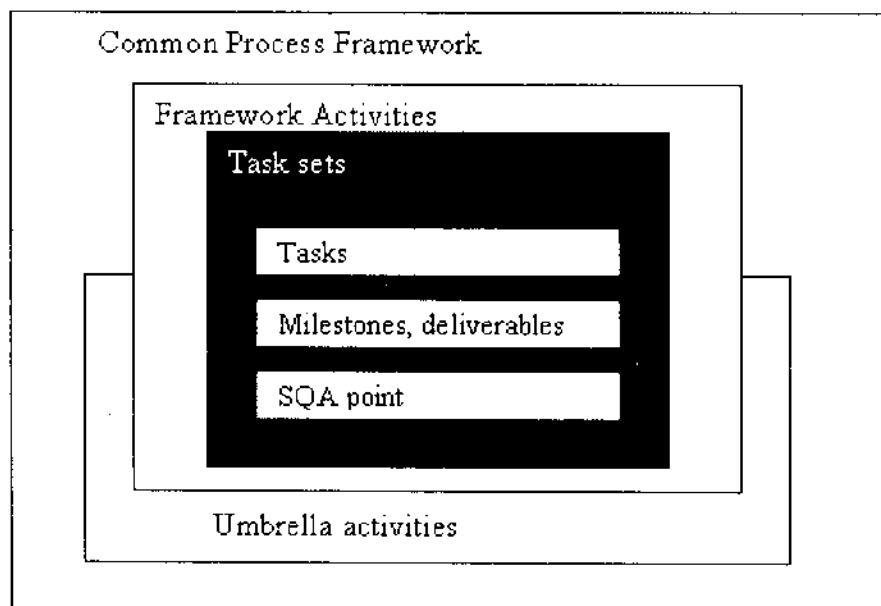
Thích nghi: Qua thời gian, môi trường ban đầu để phát triển phần mềm có thể thay đổi. Bảo trì để thích nghi thực hiện việc sửa đổi phần mềm để nó thích hợp với những thay đổi môi trường bên ngoài.

Nâng cao: Khi phần mềm được dùng, khách hàng/người dùng sẽ nhận ra những chức năng phụ có lợi. Bảo trì hoàn thiện mở rộng phần mềm ngoài các yêu cầu chức năng gốc của nó.

Bên cạnh những hoạt động cơ bản này, một số công ty còn xem xét tới kỹ nghệ ngược. Dùng một tập riêng các công cụ CASE, phần mềm cũ được kỹ nghệ hoá ngược lại để cho người dùng có thể hiểu và cải tiến sự làm việc bên trong của nó.

2.3. Quy trình phát triển phần mềm

Một quy trình phát triển phần mềm có thể được mô tả như hình 2.1.



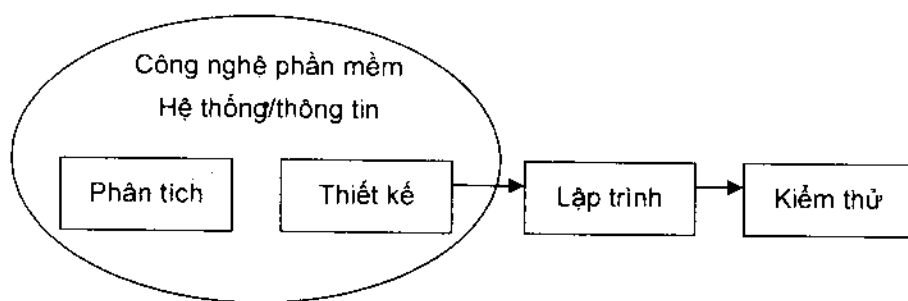
Hình 2.1. Quy trình phần mềm

Một khung quy trình chung (Common Process Framework) được thiết lập bằng cách xác định một số các hoạt động khung (Frame Activities) có khả năng áp dụng cho mọi dự án phần mềm không phân biệt kích cỡ, độ phức tạp. Tập nhiệm vụ công việc (task sets) – tập hợp các công việc công nghệ phần mềm, các cột mốc dự án, sản phẩm và các điểm bảo đảm chất

lượng – cho phép các hoạt động khung thích ứng với các đặc tính của dự án phần mềm và yêu cầu của đội dự án. Cuối cùng các hoạt động bao trùm (umbrella activities) được coi như các bảo đảm cho chất lượng phần mềm, quản lý cấu hình phần mềm, bảo trì – bao trùm mô hình quy trình. Các hoạt động bao trùm độc lập với bất cứ hoạt động khung nào và diễn ra trong suốt quy trình.

2.3.1. Mô hình tuyến tính

Diễn hình là mô hình vòng đời cổ điển bao gồm các hoạt động sau:



Hình 2.2. Mô hình tuyến tính

Công nghệ học/hệ thống/thông tin và mô hình hoá: Phần mềm là một phần trong hệ thống lớn, công việc bắt đầu bằng cách thiết lập các yêu cầu cho các thành phần hệ thống sau đó ánh xạ một số tập con các yêu cầu sang phần mềm trong quá trình tương tác giữa phần cứng, con người và cơ sở dữ liệu.

Phân tích yêu cầu phần mềm: Hiểu lĩnh vực thông tin, chức năng, hành vi tính năng và giao diện phần mềm sẽ phát triển. Yêu cầu cho cả hệ thống và phần mềm phải được tư liệu hoá và trao đổi với khách hàng.

Thiết kế: Là quá trình nhiều bước tập trung vào 4 thuộc tính chủ yếu của một chương trình: cấu trúc dữ liệu, kiến trúc phần mềm, biểu diễn giao diện, chi tiết thủ tục (thuật toán). Giống như các yêu cầu thiết kế cũng cần tư liệu hoá và là một phần quan trọng của cấu trúc phần mềm.

Tạo mã/Lập trình: Chuyển thiết kế thành một chương trình bằng một ngôn ngữ nào đó. Nếu thiết kế được chi tiết hoá thì lập trình thuần tuý là cơ học.

Hỗ trợ/bảo trì: Đáp ứng lại những thay đổi, nâng cấp phần mềm được phát triển theo sự thay đổi của nhu cầu, môi trường.

Đây là một mô hình lâu đời nhất và được ứng dụng rộng rãi nhất trong công nghệ phần mềm nhưng mô hình này cũng bộc lộ một số nhược điểm sau:

- Các dự án hiếm khi tuân theo dòng chảy tuần tự mà mô hình đề nghị. Bao giờ việc lặp lại cũng xuất hiện và nảy sinh các vấn đề khi áp dụng mô hình này.
- Khách hàng thường không trình bày được tường minh các yêu cầu của mình, mà mô hình lại đòi hỏi điều này nên sẽ không đáp ứng được với những bất trắc tự nhiên tồn tại vào lúc bắt đầu dự án.
- Khách hàng phải kiên nhẫn chờ đợi trong một thời gian nhất định mới có sản phẩm. Nếu phát hiện lỗi là một thảm họa.

2.3.2. Mô hình thuần thực khả năng

Mô hình này do Software Engineering Institute (SEI) phát triển bao gồm 5 mức sau:

Mức 1. Khởi đầu: Một số quy trình được xác định, sự thành công phụ thuộc vào các nỗ lực cá nhân.

Mức 2. Lập lại: Quy trình quản lý dự án cơ bản được thiết lập để theo vết của chi phí, lịch trình, chức năng. Công việc không phụ thuộc vào cá nhân, làm việc theo quy trình. Thành công dự án đáp ứng các ứng dụng cỡ nhỏ.

Mức 3. Xác định: Toàn bộ quy trình phần mềm cho cả các hoạt động quản lý và công nghệ được tư liệu hoá, chuẩn hoá.

Mức 4. Quản trị: Các độ đo chi tiết của quá trình phần mềm và chất lượng sản phẩm được thu thập. Toàn bộ quy trình và sản phẩm được điều khiển sử dụng các độ đo chi tiết này. Mức này bao gồm các đặc tính xác định ở mức 3.

Mức 5. Tối ưu: Tiếp tục cải tiến quy trình sản xuất bằng cách lấy thông tin phản hồi và đưa ra công nghệ mới.

SEI cũng đưa ra các lĩnh vực quy trình chủ yếu (key process areas –KPA) với từng mức trên. KPAs mô tả các chức năng công nghệ phần mềm (ví dụ các kế hoạch dự án phần mềm, quản lý các yêu cầu) đáp ứng cho từng mức cụ thể. Mỗi KPA được xác định dựa trên các đặc tính chủ yếu sau:

- *Mục đích*: Mục tiêu tổng thể mà mỗi KPA đạt được.
- *Cam kết*: Yêu cầu cần có để đạt được mục đích, cung cấp các bằng chứng về ý muốn đạt được mục đích.
- *Khả năng*: Những điều cho phép tổ chức đạt được cam kết.
- *Hoạt động*: Các nhiệm vụ cụ thể cần thực hiện để đạt được chức năng của KPA.
- *Phương thức điều khiển thực hiện*: Cách thức các hoạt động được điều khiển.
- *Phương thức kiểm tra thực hiện*: Cách thức kiểm tra các hoạt động cho KPA.

18 KPA được xác định trong các mô hình thuần thực được ánh xạ vào trong các mức độ thuần thực, bao gồm:

– *Mức 1*:

- + Quản lý cấu hình phần mềm;
- + Bảo đảm chất lượng phần mềm;
- + Quản lý các nhà thầu phụ;
- + Theo vết dự án phần mềm;
- + Lập kế hoạch quản lý dự án phần mềm;
- + Quản lý yêu cầu.

– *Mức 2*:

- + Rà soát lại nút mạng;
- + Điều phối trong nhóm;
- + Kỹ nghệ sản phẩm phần mềm;
- + Quản lý phần mềm tích hợp;
- + Chương trình đào tạo;

- + Xác định quy trình tổ chức;
- + Tập trung quy trình tổ chức;
- *Mức 3:*
 - + Quản lý chất lượng phần mềm;
 - + Quản lý quy trình số lượng;
- *Mức 4:*
 - + Quản lý thay đổi quy trình;
 - + Quản lý thay đổi công nghệ;
 - + Phòng ngừa khiếm khuyết.

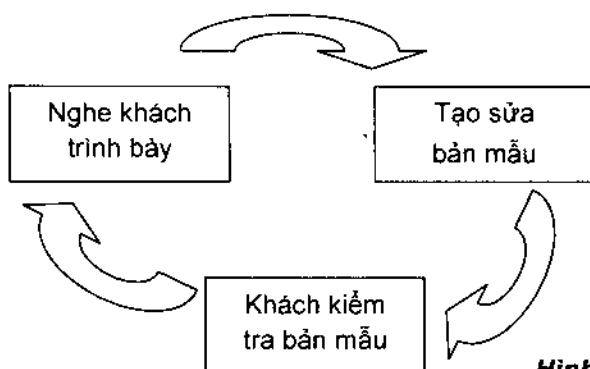
2.3.3. Mô hình chế thử

Trong nhiều trường hợp, khách hàng đã xác định một tập các mục tiêu tổng quát cho phần mềm, nhưng còn chưa xác định đầu vào, xử lý hay yêu cầu đầu ra hoặc trong các trường hợp khác, người phát triển chưa chắc chắn về tính hiệu quả của thuật toán, việc thích nghi hệ điều hành hay giao diện người – máy cần có. Đối với các trường hợp này, việc làm bản mẫu cho công nghệ phần mềm là cách tiếp cận tốt nhất.

Mô hình chế thử là một tiến trình giúp người phát triển phần mềm có khả năng tạo ra một mô hình cho phần mềm cần phải xây dựng. Mô hình này có thể là một trong 3 dạng:

- Bản mẫu trên giấy hay mô hình dựa trên PC mô tả giao diện người máy giúp cho người dùng hiểu cách các tương tác xuất hiện.
- Bản mẫu làm việc cài đặt một tập con các chức năng của phần mềm mong muốn.
- Một chương trình đã có thực hiện một phần hay tất cả các chức năng mong muốn nhưng cần cải tiến thêm các tính năng khác tùy theo nỗ lực phát triển mới.

Các sự kiện của việc làm bản mẫu được minh họa trong hình 2.3.



Hình 2.3. Mô hình chế thử

Giống như các cách tiếp cận khác, việc làm bản mẫu bắt đầu với việc thu thập yêu cầu phần mềm. Người phát triển và khách hàng gặp nhau và xác định các mục tiêu tổng thể của phần mềm, xác định các yêu cầu nào đã biết, và miền nào bắt buộc phải xác định thêm, sau đó đến việc “thiết kế nhanh”. Thiết kế nhanh tập trung vào việc biểu diễn các khía cạnh của phần mềm thấy được đối với người dùng (như cách đưa vào và định dạng đưa ra). Thiết kế nhanh dẫn tới việc xây dựng bản mẫu. Bản mẫu được khách hàng/người dùng đánh giá và được dùng để làm mịn dần các yêu cầu đối với phần mềm cần phát triển. Tiến trình được lặp đi lặp lại cho tới khi bản mẫu được vì chính thoả mãn nhu cầu của khách hàng, đồng thời cũng giúp cho người phát triển hiểu được kỹ hơn cần phải thực hiện nhu cầu như thế nào.

Một cách lý tưởng, bản mẫu phục vụ như một cơ chế để xác định các yêu cầu phần mềm. Nếu một bản mẫu làm việc được xây dựng thì người phát triển có thể dùng được các đoạn chương trình đã có hay áp dụng các công cụ (như bộ sinh báo cáo, bộ quản lý cửa sổ,...) để nhanh chóng sinh ra chương trình làm việc.

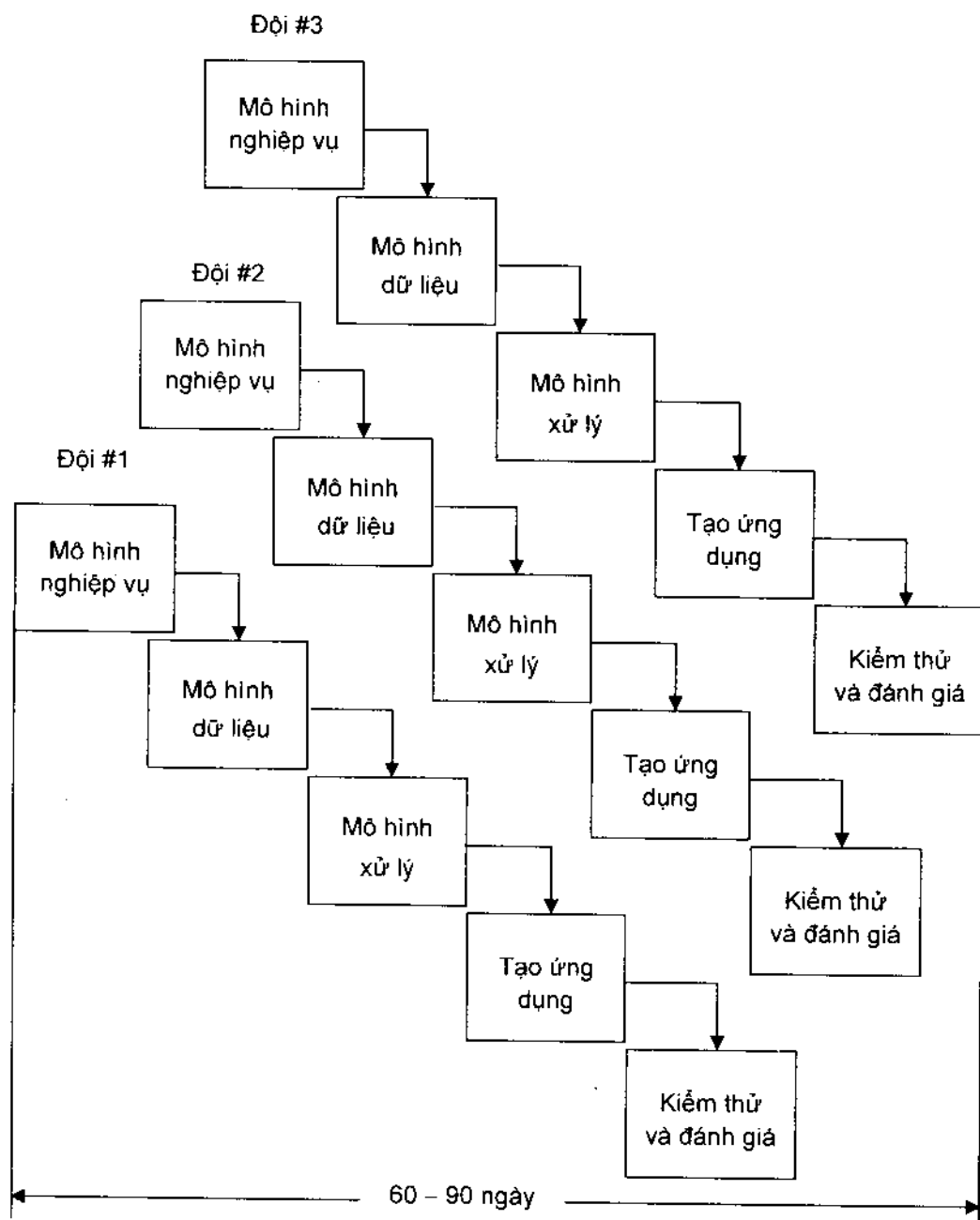
2.3.4. Mô hình phát triển ứng dụng nhanh (RAD)

Là quy trình phát triển phần mềm gia tăng từng bước với chu trình phát triển ngắn từ 60 đến 90 ngày. Mô hình này xây dựng dựa trên hướng thành phần với khả năng tái sử dụng cao. Mô hình gom một số nhóm, mỗi nhóm làm một RAD theo các pha sau:

Mô hình nghiệp vụ: Luồng thông tin được mô hình hoá để trả lời các câu hỏi sau đây:

- Thông tin nào điều khiển xử lý nghiệp vụ?
- Thông tin nào được sinh ra?

- Ai sinh ra nó?
- Thông tin đi đến đâu?
- Ai xử lý chúng?



Hình 2.4. Mô hình phát triển ứng dụng nhanh

Mô hình dữ liệu: Các đối tượng dữ liệu cần có để hỗ trợ nghiệp vụ, định nghĩa thuộc tính các đối tượng và xác lập mối quan hệ giữa các đối tượng.

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

Tạo ứng dụng: Dùng các kỹ thuật thể hệ thứ 4 để tạo phần mềm từ các thành phần đã có sẵn hoặc tạo ra các thành phần có thể tái sử dụng sau này. Dùng các công cụ tự động để xây dựng phần mềm.

Kiểm thử và đánh giá : Kiểm thử các thành phần mới và kiểm chứng lại mọi giao diện (các thành phần cũ được kiểm chứng và dùng lại).

Nếu một ứng dụng nghiệp vụ có thể module hoá sao cho các chức năng chính được thực hiện trong phạm vi ba tháng, khi đó mô hình RAD sẽ là sự lựa chọn phù hợp. Mỗi chức năng chính sẽ được thực hiện bởi một đội RAD riêng rẽ sau đó tích hợp tất cả chúng lại.

Tuy nhiên RAD cũng có các hạn chế sau:

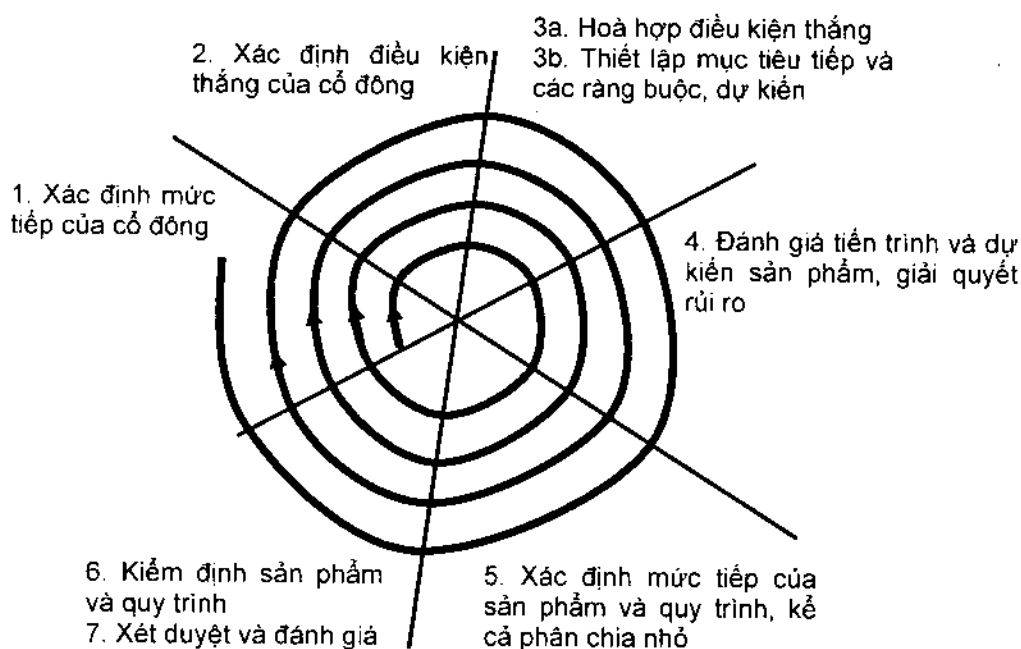
- Cần nguồn nhân lực dồi dào để tạo ra các nhóm chức năng chính.
- Yêu cầu hai bên giao kèo trong thời gian ngắn phải có phần mềm hoàn chỉnh, thiếu trách nhiệm một bên có thể làm dự án đổ vỡ.
- RAD không phải là tốt trong mọi ứng dụng, nhất là các ứng dụng không thể module hoá hoặc đòi hỏi tính năng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

2.3.5. Mô hình xoắn ốc

Mô hình xoắn ốc cho kỹ nghệ phần mềm đã được phát triển bao gồm các tính năng tốt nhất của cả mô hình tuyến tính lẫn mô hình chế thử trong khi vẫn bổ sung thêm các yếu tố mới – phân tích rủi ro – yếu tố bị thiếu trong các mô hình này. Mô hình này được biểu thị theo đường xoắn ốc, chia thành các lĩnh vực nhiệm vụ sau:

- *Giao tiếp khách hàng:* Thiết lập mối quan hệ hiệu quả giữa khách hàng và nhà phát triển.
- *Lập kế hoạch:* Xác định các tài nguyên, thời gian và các thông tin dự án liên quan khác.

- *Phân tích rủi ro*: Phân tích cả rủi ro kỹ thuật và rủi ro quản lý.
- *Kỹ nghệ*: Xây dựng một hay nhiều mô tả cho ứng dụng.
- *Xây dựng và xuất xưởng*: Xây dựng, kiểm thử, cài đặt và cung cấp các hỗ trợ người dùng (các tài liệu và đào tạo).
- *Khách hàng đánh giá*: Các phản hồi của khách hàng dựa trên đánh giá phần mềm tạo ra trong giai đoạn kỹ nghệ và thực hiện trong giai đoạn cài đặt.



Hình 2.5. Mô hình xoắn ốc

Với mỗi lần lặp xung quanh xoắn ốc (bắt đầu từ tâm và đi ra ngoài theo chiều kim đồng hồ) người ta lại xây dựng thêm các phiên bản được hoàn thiện dần của phần mềm. Trong mạch xoắn thứ nhất, các mục tiêu, phương án và ràng buộc được xác định, các rủi ro được định rõ và phân tích. Tại các vòng xung quanh xoắn ốc, cao điểm của phân tích rủi ro là quyết định tiến hành hay không tiến hành. Nếu rủi ro quá lớn có thể sẽ đình chỉ dự án. Tuy nhiên, trong hầu hết trường hợp, việc đi theo xoắn ốc vẫn tiếp tục, với mỗi con đường lại chuyển người phát triển đi xa hơn, hướng tới mô hình hoàn chỉnh hơn của hệ thống và sau cùng đưa đến chính bản thân hệ thống vận hành.

Mô hình xoắn ốc là cách tiếp cận thực tế nhất với việc phát triển các hệ thống và phần mềm có quy mô lớn. Người ta dùng cách tiếp cận tiến hoá đối với kỹ nghệ phần mềm để làm cho người phát triển, khách hàng hiểu được và phản ứng được rủi ro ở mỗi mức tiến hoá. Người ta dùng cách làm bản mẫu như một cơ chế làm giảm bớt rủi ro, nhưng điều quan trọng hơn để làm cho người phát triển áp dụng được cách tiếp cận làm bản mẫu tại mọi giai đoạn tiến hoá của sản phẩm. Nó duy trì cách tiếp cận từng bước một cách có hệ thống theo cách tiếp cận vòng đời cổ điển gợi ý nhưng tổ hợp cách tiếp cận này vào một khuôn khổ lặp lại, phản ánh được sát thực hơn thế giới thực. Mô hình xoắn ốc yêu cầu việc xem xét trực tiếp các rủi ro kỹ thuật tại mọi giai đoạn của dự án, và nếu áp dụng đúng thì nó sẽ làm giảm rủi ro trước khi chúng trở thành vấn đề thực sự. Tuy nhiên, giống như các mô hình khác, mô hình xoắn ốc vẫn có các hạn chế sau:

- Khó thuyết phục được khách hàng lớn rằng cách tiếp cận tiến hoá sẽ kiểm soát được. Nó đòi hỏi tri thức chuyên gia đánh giá rủi ro chính xác và dựa trên tri thức chuyên gia này mà đạt được thành công. Nếu một rủi ro chính không được phát hiện ra, vấn đề sẽ xuất hiện.
- Bản thân mô hình này còn tương đối mới và còn chưa được sử dụng rộng rãi như mô hình tuyến tính hay mô hình chế thử. Cần phải có thêm thời gian trước khi người ta có thể xác định được tính hiệu quả của mô hình này với sự chắc chắn an toàn.

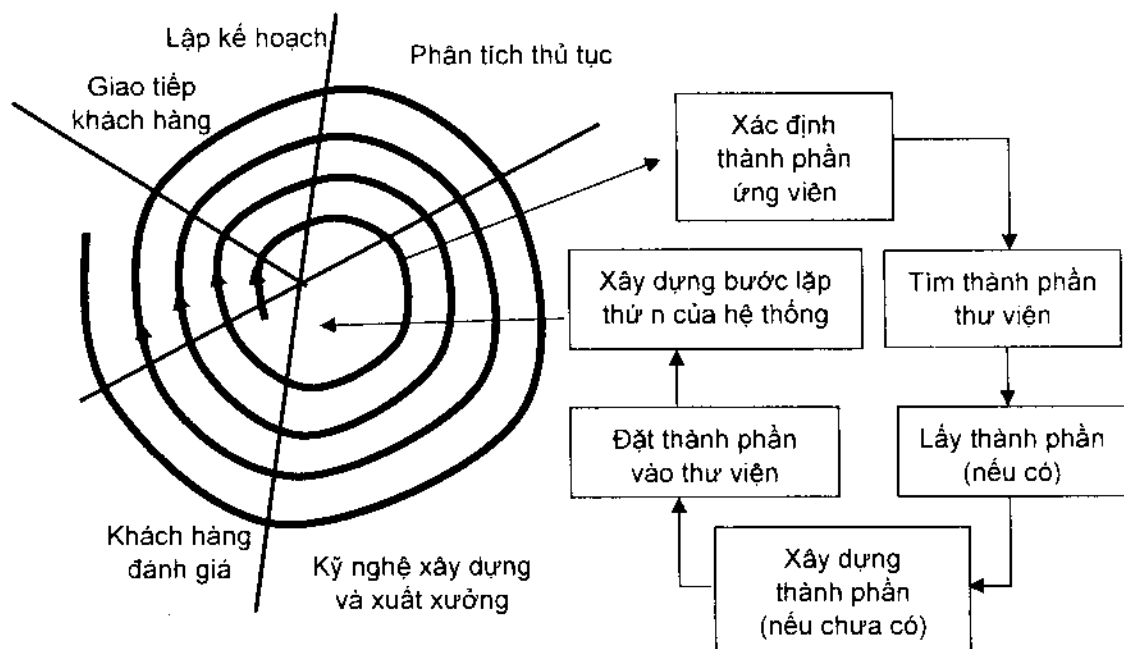
2.3.6. Mô hình theo thành phần

Mô hình này gắn với những công nghệ hướng đối tượng thông qua việc tạo các lớp có chứa dữ liệu và các giải thuật xử lý dữ liệu. Nếu thiết kế chính xác, các lớp hướng đối tượng sẽ được sử dụng lại trong các ứng dụng và các cấu trúc hệ thống máy tính khác nhau.

Mô hình này có nhiều điểm tương đương với mô hình xoắn ốc. Tuy nhiên, các mô hình lắp ráp thành phần tạo nên ứng dụng từ các thành phần phần mềm đã được đóng gói trước (đôi khi gọi là các lớp).

Các hoạt động kỹ nghệ bắt đầu bằng việc xác định các lớp. Công việc này được thực hiện bằng cách kiểm tra các dữ liệu được xử lý trong ứng dụng và các giải thuật được ứng dụng để hoàn thành các thao tác này. Các dữ liệu và thuật toán tương ứng được đóng gói trong các lớp.

Mô hình này tái sử dụng các thành phần từ các thư viện/kho các lớp và sẽ tiết kiệm được 70% thời gian, 84% giá thành, chỉ số hiệu suất so với chuẩn công nghiệp là 26,2/16,9.



Hình 2.6. Mô hình thành phần

2.3.7. Mô hình hình thức

Mô hình này còn được gọi là mô hình phòng sạch, tập hợp các công cụ đặc tả toán học phần mềm máy tính từ khâu định nghĩa phát triển đến kiểm soát.

Khi sử dụng mô hình này để phát triển, nó cung cấp các kỹ thuật để loại bỏ các vấn đề khó giải quyết khi sử dụng các mô hình khác. Sự mập mờ, không hoàn chỉnh, không nhất quán sẽ được tìm ra và giải quyết dễ dàng.

Khi sử dụng mô hình này trong thiết kế, nó cung cấp một nền tảng kiểm soát chương trình. Do đó, cho phép các kỹ sư phần mềm phát hiện và sửa các lỗi khó.

Mô hình này thường được áp dụng trong phát triển các phần mềm có độ an toàn rất cao trong y tế, hàng không... Tuy nhiên sử dụng mô hình này cũng vấp phải các hạn chế sau:

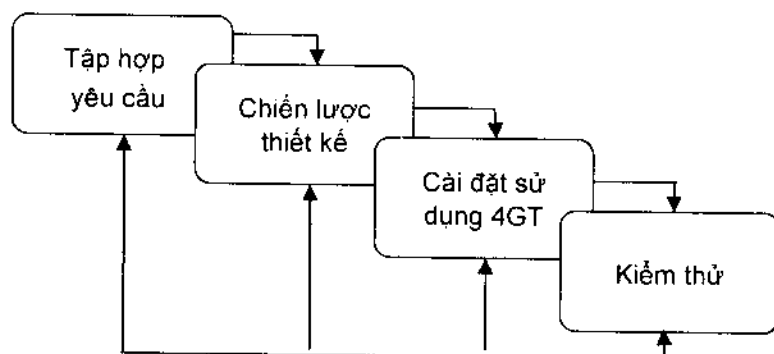
- Phát triển một mô hình hình thức tiêu tốn nhiều thời gian và kinh phí.
- Do chỉ có một số ít các nhà phát triển phần mềm có nền tảng cần thiết để áp dụng các phương pháp hình thức nên khi phát triển mô hình này yêu cầu phải đào tạo các nhà phát triển.
- Sử dụng mô hình này sẽ gặp phải khó khăn khi làm việc, giao tiếp với khách hàng không có nhiều kiến thức về kỹ thuật và chuyên môn.

2.3.8. Các công nghệ thế hệ thứ 4

Thuật ngữ công nghệ thứ 4 bao gồm một phạm vi rộng các công cụ công nghệ phần mềm có một điểm chung: mỗi công việc đều cho phép người phát triển phần mềm xác định một số đặc trưng phần mềm ở mức cao. Sau đó công cụ đó tự động sinh mã chương trình gốc theo nhu cầu của người phát triển. Các công nghệ thế hệ thứ 4 tập trung vào khả năng xác định phần mềm đối với một máy ở mức độ gần với ngôn ngữ tự nhiên hay dùng một ký pháp có ý nghĩa.

Hiện nay, một môi trường phát triển phần mềm hỗ trợ cho các công nghệ thứ 4 bao gồm một số hay tất cả các công cụ sau: ngôn ngữ phi thủ tục cho truy vấn cơ sở dữ liệu, bộ sinh báo cáo, xử lý dữ liệu, tương tác màn hình, tạo mã nguồn, khả năng đồ họa bậc cao, khả năng bảng tính, khả năng giao diện Web.

Mô hình 4GT được vẽ như trong hình 2.7.



Hình 2.7. Các công nghệ thế hệ thứ 4

Giống như các mô hình khác, 4GT bắt đầu từ bước thu thập yêu cầu. Một cách lý tưởng, khách hàng sẽ mô tả các yêu cầu và các yêu cầu đó sẽ được dịch trực tiếp thành bản mẫu vận hành được. Nhưng điều này thực tế không thực hiện được. Khách hàng nhiều khi không chắc chắn mình cần gì, mơ hồ trong việc xác định các sự kiện đã biết, không có khả năng hay không xác định thông tin cần cho công cụ 4GT thực hiện. Hơn nữa, các công cụ 4GT hiện nay cũng chưa đủ độ tinh vi thích ứng với ngôn ngữ tự nhiên thật sự.

Với các ứng dụng nhỏ, có thể chuyển trực tiếp từ bước thu thập yêu cầu sang cài đặt bằng cách dùng ngôn ngữ thế hệ thứ 4. Tuy nhiên, với những nỗ lực lớn, cần phải phát triển một chiến lược thiết kế cho hệ thống, ngay cả nếu có dùng 4GT. Việc dùng 4GT thiếu thiết kế với các dự án lớn sẽ gây khó khăn (chất lượng kém, khó bảo trì, người dùng khó chấp nhận).

Việc cài đặt dùng 4GT giúp cho người phát triển phần mềm biểu diễn được các kết quả mong muốn theo cách phát sinh tự động chương trình tính ra chúng.

Để biến đổi một cài đặt 4GT thành một sản phẩm, người phát triển phải trải qua việc kiểm thử, xây dựng các tài liệu có ý nghĩa và thực hiện các hoạt động chuyển tiếp cần có trong mô hình phần mềm khác. Bên cạnh đó, phần mềm được phát triển theo 4GT phải được xây dựng để việc bảo trì có thể được tiến hành nhanh chóng.

Có nhiều tranh cãi về mô hình này. Người ủng hộ cho là 4GT làm giảm đáng kể thời gian phát triển phần mềm và làm tăng hiệu suất của người xây dựng phần mềm. Người phản đối cho rằng các công cụ 4GT hiện tại nhiều khi phức tạp hơn các ngôn ngữ lập trình khác, chương trình gốc do các công cụ này tạo ra không hiệu quả và việc bảo trì các hệ thống phần mềm lớn dùng 4GT sẽ phát sinh ra các vấn đề mới.

Có thể tóm tắt cách tiếp cận 4GT như sau:

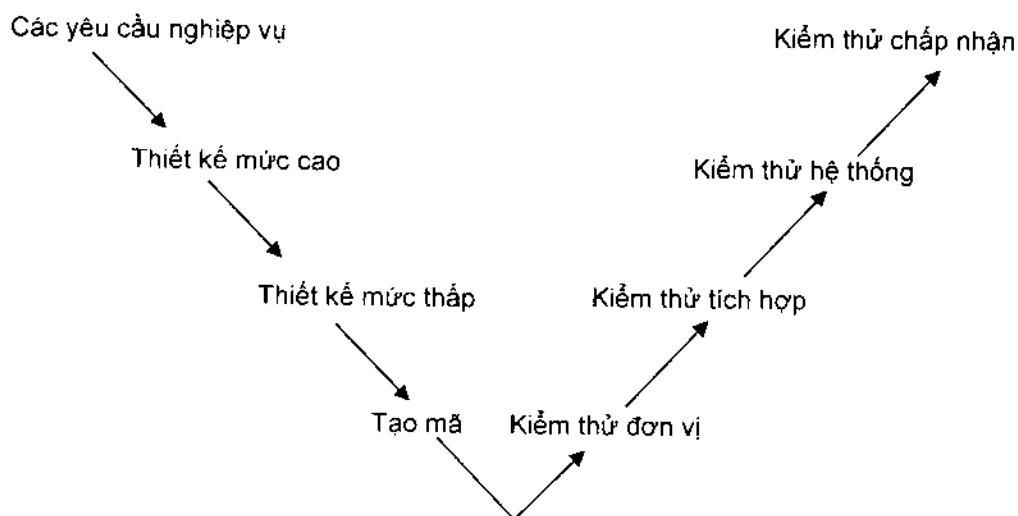
- Lĩnh vực ứng dụng hiện tại cho 4GT mới chỉ giới hạn trong các ứng dụng thông tin nghiệp vụ đặc biệt, việc phân tích thông tin và làm báo cáo là nhân tố chủ chốt cho các cơ sở dữ liệu lớn. Các công cụ CASE hỗ trợ việc dùng 4GT để tự động sinh ra khung chương trình cho các ứng dụng công nghệ và thời gian thực.

- Dữ liệu sơ bộ thu thập từ các công ty có dùng 4GT cho thấy thời gian cần thiết để tạo ra phần mềm được giảm đáng kể đối với các ứng dụng vừa và nhỏ và khối lượng thiết kế, phân tích cho các ứng dụng nhỏ cũng giảm bớt.
- Việc dùng 4GT cho các nỗ lực phát triển phần mềm lớn đòi hỏi tập trung phân tích, thiết kế và kiểm thử nhiều để tiết kiệm thời gian.

Tóm lại, các công nghệ thể hệ thứ tư đã trở thành một phần quan trọng của việc phát triển phần mềm trong lĩnh vực ứng dụng hệ thống tin và rất có thể sẽ được sử dụng rộng rãi trong các ứng dụng công nghệ và thời gian thực.

2.3.9. Mô hình chữ V

Mô hình chữ V được thực hiện từ trái sang phải, mô tả dãy các hoạt động phát triển và kiểm thử cơ bản. Mô hình này rất có giá trị vì nó nêu bật sự tồn tại của các mức kiểm thử và mô tả cách mỗi mức kiểm thử này liên quan đến một pha phát triển khác nhau.



Hình 2.8. Mô hình chữ V

Kiểm thử đơn vị dựa trên mã và được thực hiện đầu tiên bởi người phát triển để chứng minh các phần nhỏ nhất của mã thực thi thích hợp.

Kiểm thử tích hợp chứng minh hai hay nhiều đơn vị hoặc các công việc tích hợp phối hợp cùng nhau chính xác và có xu hướng tập trung vào các giao diện được mô tả trong các mức thiết kế cao.

Khi mọi đơn vị và các tích hợp của nó được kiểm thử, kiểm thử hệ thống được tiến hành để chứng tỏ rằng hệ thống làm việc end – to – end giống sản phẩm, cung cấp các chức năng nghiệp vụ được mô tả trong thiết kế mức cao. Khi tổ chức kỹ thuật hoàn thành công việc kiểm thử, tức là thực tế đã đạt được yêu cầu nghiệp vụ đặt ra.

Tuy nhiên có nhiều người nghi ngờ về tính phù hợp của mô hình này. Mô hình này dựa trên một tập các bước phân tích theo một trình tự cụ thể nhưng không phản ánh thực tế. Mô hình chữ V bắt đầu bằng các yêu cầu, thậm chí ngay cả khi nhiều dự án thiếu các yêu cầu chính xác. Mô hình chữ V nhắc nhở chúng ta kiểm thử mỗi khi thực hiện một pha phát triển nhưng không mô tả chúng ta cần làm gì.

Chương 3

QUẢN LÝ DỰ ÁN PHẦN MỀM

Quản lý dự án phần mềm là tăng đầu trong tiến trình công nghệ phần mềm. Ta gọi nó là tăng vì nó phủ lên toàn bộ tiến trình phát triển từ đầu đến cuối.

3.1. Các khái niệm quản lý dự án

3.1.1. Con người

Tiến trình phần mềm và mọi dự án phần mềm đều bao gồm các thành viên được chia thành 5 nhóm chính sau:

1. *Quản lý cấp cao*: Người xác định các vấn đề nghiệp vụ thường có ảnh hưởng đáng kể đến dự án.
2. *Quản lý dự án*: Người lập kế hoạch, thúc đẩy, tổ chức và điều khiển những người thực hiện làm các công việc phần mềm.
3. *Người thực hiện*: Người dùng các kỹ năng kỹ thuật cần thiết để làm ra một sản phẩm hay một ứng dụng.
4. *Khách hàng*: Người đặc tả các yêu cầu cho phần mềm.
5. *Người dùng cuối*: Người tương tác với phần mềm khi nó được phát hành.

Mọi dự án phần mềm đều được tạo ra bởi những người nói trên. Để hoạt động hiệu quả hơn, đội dự án nên được tổ chức theo cách tối đa hoá kỹ năng và khả năng của mỗi người. Đây là công việc của trưởng nhóm.

3.1.1.1. Trưởng nhóm

Quản lý dự án là một hoạt động nhấn mạnh đến con người, do đó, những người giỏi kỹ thuật nhiều khi lại là những trưởng nhóm tồi. Vậy khi

chọn một người lãnh đạo một dự án phần mềm chúng ta tìm người có phẩm chất gì? Jerry Weinberg đã đưa ra các gợi ý sau để trả lời câu hỏi này:

- *Khả năng thúc đẩy*: Khả năng khuyến khích các kỹ thuật viên làm việc với khả năng cao nhất.
- *Khả năng tổ chức*: Khả năng đưa ra một quy trình đã tồn tại hay tìm ra một quy trình mới có thể biến các khái niệm ban đầu thành sản phẩm cuối.
- *Cách tân*: Khả năng khuyến khích người khác tạo ra sáng tạo thậm chí khi họ làm việc trong các giới hạn đã được thiết lập cho một sản phẩm phần mềm hoặc một ứng dụng cụ thể.

Một người quản lý dự án thành công áp dụng các kiểu quản lý để giải quyết vấn đề sau: tập trung để hiểu các vấn đề cần phải giải quyết, quản lý các luồng ý kiến và cùng lúc đó cho mọi người trong đội dự án biết (bằng lời hoặc bằng các hành động) mà không thoả hiệp.

Một cách nhìn khác nhằm tạo nên một người quản lý dự án hiệu quả nhấn mạnh vào 4 đặc điểm chủ yếu sau:

1. *Giải quyết vấn đề*: Một người quản lý dự án hiệu quả có thể dự đoán các vấn đề kỹ thuật và tổ chức thích hợp. Nói một cách hệ thống anh ta có thể tổ chức một giải pháp hoặc thúc đẩy những người khác đưa ra các giải pháp, ứng dụng các bài học rút ra được trong các dự án trước cho các tình huống mới, và vẫn đủ mềm dẻo để thay đổi hướng nếu các nỗ lực ban đầu giải quyết vấn đề không có kết quả.

2. *Thống nhất quản lý*: Một quản lý dự án tốt phải tạo ra thay đổi của dự án. Anh ta phải đủ tự tin để thay đổi khi cần thiết và bảo đảm cho mọi người làm việc theo khả năng.

3. *Thành tích*: Để tối ưu hoá sản phẩm của một đội dự án, quản lý dự án phải thường khi bắt đầu và khi hoàn thành, và chứng minh qua các hành động của anh ta rằng các rủi ro được điều khiển sẽ không bị phạt.

4. *Ảnh hưởng và xây dựng đội ngũ*: Một quản lý dự án tốt phải có khả năng “đọc” con người, anh ta có thể hiểu các tín hiệu bằng lời hoặc không lời và tác động trở lại đến người đã phát ra tín hiệu.

3.1.1.2. Đội phân mềm

Có nhiều kiểu tổ chức con người để phát triển phần mềm. Cấu trúc tổ chức khó có thể thay đổi. Các lựa chọn sau đây có thể áp dụng cho các dự án gồm n người làm việc trong k năm:

1. n người được phân công làm m công việc chức năng khác nhau, ít liên hệ với nhau. Người quản lý dự án phân mềm có nhiệm vụ điều phối những người liên quan đến dự án.

2. n người riêng lẻ được phân công làm m công việc chức năng khác nhau ($m < n$), do đó đội được thành lập, và trưởng nhóm được chỉ định. Sự điều phối giữa các đội là trách nhiệm của quản lý dự án.

3. n người riêng lẻ được tổ chức trong t đội, mỗi đội được phân công một hay nhiều công việc chức năng, mỗi đội có một cấu trúc riêng biệt được xác định cho toàn bộ các đội trong dự án. Sự điều phối được thực hiện bởi cả quản lý dự án và đội.

Cấu trúc đội tốt phụ thuộc vào kiểu quản lý, số người trong đội, kỹ năng của họ. Mantel đã đưa ra 3 kiểu tổ chức sau đây:

- *Dân chủ tập quyền (DD)*: Đội công nghệ phần mềm không có lãnh đạo cố định. Nhà điều phối công việc được chỉ định trong một thời gian ngắn và được thay bằng người khác có thể điều phối công việc khác. Quyết định các vấn đề và cách tiếp cận được tạo ra bởi sự đồng thuận trong nhóm. Liên lạc giữa các thành viên trong đội theo chiều ngang.
- *Điều khiển tập quyền (CD)*: Đội công nghệ phần mềm chỉ định người lãnh đạo điều phối các công việc và nhà lãnh đạo thứ hai chịu trách nhiệm cho các công việc con. Giải quyết vấn đề vẫn là một nhóm các hoạt động, nhưng sự thực hiện được người lãnh đạo chia ra cho các nhóm con. Liên lạc giữa các nhóm và giữa các cá nhân theo chiều ngang. Cũng có thể liên lạc theo chiều dọc, dọc theo cấu trúc cây điều khiển.
- *Tập trung điều khiển (CC)*: Giải quyết các vấn đề mức cao và điều phối trong đội dự án được quản lý bởi trưởng nhóm. Liên lạc giữa các lãnh đạo và trưởng nhóm theo chiều dọc.

Khi lập kế hoạch về tổ chức đội công nghệ phần mềm, Mantel mô tả 7 nhân tố :

- Sự khác nhau của vấn đề phải giải quyết;
- Kích cỡ của chương trình kết quả theo số dòng mã lệnh hay số điểm chức năng;
- Thời gian đội dự án làm việc cùng nhau;
- Mức độ vấn đề có thể module hoá;
- Yêu cầu chất lượng và độ tin cậy của hệ thống được xây dựng;
- Sự khắt khe của ngày bàn giao sản phẩm;
- Mức độ liên lạc (cộng đồng) yêu cầu cho dự án.

Constantine đưa ra 4 mô thức tổ chức cho đội công nghệ phần mềm như sau:

1. *Mô thức đóng (close paradigm)* : Tổ chức một đội theo cây quyền lực truyền thống (tương tự như đội CC). Đội làm việc hiệu quả khi tạo ra các phần mềm ít giống với các sản phẩm trước đó, nhưng khi làm việc trong mô thức đóng, thường ít xuất hiện sáng kiến.

2. *Mô thức ngẫu nhiên (random paradigm)*: Cấu trúc đội lỏng lẻo và phụ thuộc vào các sáng tạo cá nhân của các thành viên trong đội.

3. *Mô thức mở (open paradigm)*: Cố gắng cấu trúc một đội đạt được một số các điều khiển liên kết với mô thức đóng nhưng sáng tạo hơn khi sử dụng mô thức ngẫu nhiên. Công việc được thực hiện với nhiều mối liên lạc và các quyết định dựa trên sự đồng thuận. Cấu trúc đội mô thức mở thích hợp để giải quyết các vấn đề phức tạp, nhưng hiệu quả thực hiện khác nhau với các đội phần mềm khác nhau.

4. *Mô thức đồng bộ (synchronous paradigm)* : Phụ thuộc vào cách chia vấn đề và tổ chức các thành viên làm việc trong các phần của vấn đề với các hoạt động liên lạc ít ỏi giữa họ.

3.1.1.3. Vấn đề điều phối và liên lạc

Có nhiều nguyên nhân làm cho dự án phần mềm gặp rắc rối : Quy mô của nỗ lực phát triển quá lớn, sự lãnh đạo phức tạp, có xung đột và khó khăn trong điều phối các thành viên. Để giải quyết các vấn đề một cách hiệu quả, mỗi đội công nghệ phần mềm phải thiết lập một phương thức điều

phối những người làm việc trong đội một cách có hiệu quả. Để thực hiện điều này, các kỹ thuật liên lạc hình thức và không hình thức giữa các thành viên trong đội và giữa các đội được thiết lập. Các kỹ thuật điều phối trong dự án được chia thành các loại sau:

- *Tiếp cận hình thức, không liên quan đến riêng ai:* Bao gồm các tài liệu công nghệ phần mềm và các sản phẩm (như mã nguồn), bản ghi nhớ kỹ thuật, cột mốc dự án lịch trình và các công cụ quản lý dự án, các yêu cầu thay đổi, các tài liệu liên quan, báo cáo theo vết lỗi, và các kho dữ liệu.
- *Thủ tục hình thức, không liên quan đến riêng ai:* Tập trung vào các hoạt động quản lý chất lượng ứng dụng. Chúng bao gồm thiết kế và kiểm tra mã chương trình.
- *Thủ tục giữa cá nhân với nhau, không liên quan đến riêng ai:* Bao gồm việc gặp gỡ nhóm để phổ biến thông tin, giải quyết vấn đề và sắp xếp các yêu cầu, phát triển đội ngũ.
- *Liên lạc điện tử:* Sử dụng e-mail, các bảng tin điện tử, Website, hệ thống hội nghị truyền hình.
- *Mạng giữa các cá nhân:* Thảo luận với những người bên ngoài dự án, cũng có thể có trợ giúp các thành viên trong đội dự án.

3.1.2. Vấn đề

Khi bắt đầu dự án, một nhà quản lý dự án phải đối mặt ngay với các vấn đề như yêu cầu phải đánh giá chất lượng và lập kế hoạch tổ chức nhưng các thông tin chủ yếu cần cung cấp lại chưa có. Một phân tích chi tiết của yêu cầu phần mềm có thể sẽ cung cấp các thông tin cần thiết để đánh giá nhưng những phân tích này thường mất đến vài tuần hoặc vài tháng để hoàn thành. Tồi tệ hơn, yêu cầu có thể thay đổi, thậm chí thay đổi thường xuyên trong suốt dự án. Do đó, phải nghiên cứu các vấn đề ngay khi bắt đầu dự án. Tối thiểu, phạm vi vấn đề phải được xác định. Phạm vi được xác định khi trả lời các câu hỏi sau:

Hoàn cảnh ra đời ? Mục đích thông tin ? Chức năng và hiệu năng ?

3.1.3. Quy trình

Các giai đoạn xác định quy trình phần mềm – xác định, phát triển và bảo trì được áp dụng cho mọi phần mềm. Quản lý dự án phải xác định mô hình nào được sử dụng cho dự án, sau đó xác định kế hoạch sơ bộ dựa trên các hành động trong quy trình. Khi kế hoạch sơ bộ được thiết lập, quá trình phân tích bắt đầu. Một kế hoạch hoàn chỉnh phản ánh các công việc yêu cầu trong các hoạt động khung được tạo ra.

3.2. Đo phần mềm

Người ta cần phải đo phần mềm bởi nhiều lý do như sau :

1. Để chỉ ra chất lượng phần mềm.
2. Để khẳng định hiệu suất của những người tạo ra sản phẩm.
3. Để khẳng định lợi ích (dưới dạng hiệu suất và chất lượng) mà phương pháp và công cụ kỹ nghệ phần mềm đem lại.
4. Để tạo ra một vạch ranh giới cho ước lượng.
5. Để giúp biện minh cho các yêu cầu về công cụ mới bổ sung.

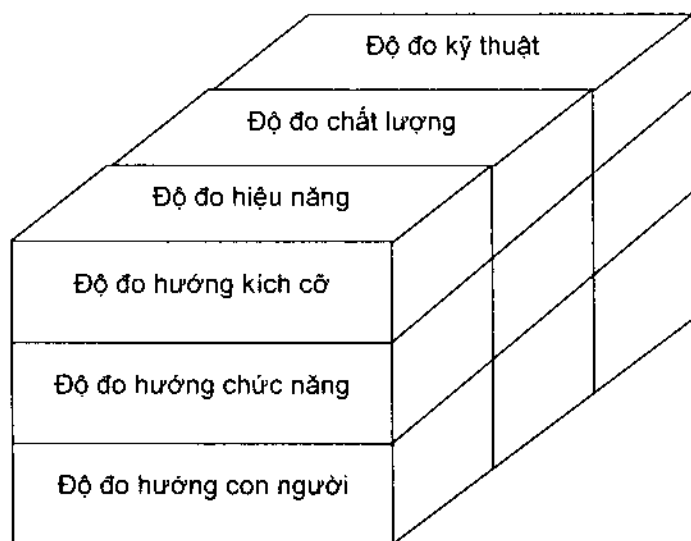
Ta có thể chia các cách đo theo phạm trù vật lý theo hai cách : Đo trực tiếp (chẳng hạn như chiều dài chiếc then), đo gián tiếp (như chất lượng chiếc then được tạo ra, được đo bằng cách đếm số các then bị loại bỏ). Độ đo phần mềm cũng có thể phân loại tương tự.

Cách đo trực tiếp cho tiến trình công nghệ phần mềm bao gồm việc đo về chi phí và công sức bỏ ra. Cách đo trực tiếp cho một sản phẩm bao gồm số dòng lệnh (LOC) được tạo ra, tốc độ thực hiện, kích cỡ bộ nhớ và những khiếm khuyết được báo cáo lại qua một thời kỳ nào đó. Việc đo gián tiếp sản phẩm bao gồm chức năng, chất lượng, độ phức tạp, tính hiệu quả, độ tin cậy, tính bảo trì và nhiều khả năng khác.

Chi phí và công sức cần cho việc xây dựng phần mềm, số dòng lệnh và những cách đo trực tiếp khác tương đối dễ xác định khi đã thiết lập trước được các quy ước đặc biệt cho việc đo. Tuy nhiên, chất lượng và chức năng phần mềm hay tính hiệu quả và tính dễ bảo trì của nó thì khó xác định hơn và chỉ có thể đo một cách gián tiếp.

Có thể phân loại được lĩnh vực các độ đo phần mềm khác như trong hình 3.1.

Độ đo hiệu năng tập trung vào đầu ra của tiến trình công nghệ phần mềm. *Độ đo chất lượng* đưa ra một chỉ dẫn về việc phần mềm tuân thủ chặt chẽ đến đâu đối với các yêu cầu tường minh và không tường minh của người dùng (tính thích hợp của phần mềm với việc sử dụng). *Độ đo kỹ thuật* tập trung vào đặc trưng của phần mềm như độ phức tạp logic, mức độ module hơn là chú ý vào tiến trình phần mềm phát triển.



Hình 3.1. Phân loại độ đo

Cũng có thể phát triển cách phân loại thứ hai. *Độ đo kích cỡ* được dùng để thu thập đầu ra và chất lượng công nghệ phần mềm. *Độ đo theo chức năng* đưa ra cách đo gián tiếp; *độ đo theo con người* thu thập thông tin, cách thức để phát triển phần mềm máy tính và cảm nhận của con người về tính hiệu quả của công cụ và phương pháp.

3.2.1. Độ đo theo kích cỡ

Độ đo phần mềm theo kích cỡ là cách đo trực tiếp cho phần mềm và tiến trình phân tích của nó. Nếu một tổ chức làm phần mềm duy trì các bản ghi đơn giản, thì có thể tạo ra một bảng các dữ liệu theo kích cỡ, như bảng 3.1.

Bảng 3.1. Độ đo theo kích cỡ

Dự án	Công sức	1000\$	KLOC	Trang tư liệu	Lỗi	Người
Aaa-01	24	168	12.1	365	29	3
Bbb-04	62	440	27.2	1224	86	5
Fff-05	43	314	20.2	1050	64	6
...	...	...		...		

KLOC (số dòng chương trình (đơn vị nghìn)).

Bảng 3.1 liệt kê theo từng dự án phát triển phần mềm đã được hoàn thành trong vài năm qua và dữ liệu theo kích cỡ tương ứng cho dự án đó.

Từ dữ liệu thô trong bảng này, ta có thể phát triển một tập các độ đo chất lượng và hiệu năng theo kích cỡ cho từng dự án. Có thể tính ra giá trị trung bình cho mọi dự án:

Hiệu năng=KLOC/người-tháng

Chất lượng=khiếm khuyết/KLOC

Bên cạnh đó có thể tính các độ đo đáng quan tâm khác:

Chi phí=\$/KLOC

Tư liệu=số trang tư liệu/KLOC

Các độ đo theo kích cỡ vẫn còn được bàn cãi rất nhiều và vẫn chưa được coi là cách tốt nhất đo chất lượng phần mềm. Phần lớn các tranh luận xoay quanh việc dùng các dòng chương trình làm thước đo chính. Những người ủng hộ LOC cho rằng LOC là độ đo phổ biến của các dự án phát triển phần mềm, dễ dàng được đếm, các mô hình ước lượng phần mềm hiện có đều dùng LOC hay KLOC làm đầu vào chính và cho rằng phần lớn các tài liệu và dữ liệu đều dựa trên LOC đã có sẵn. Những người phản đối thì nói rằng LOC phụ thuộc vào ngôn ngữ lập trình, có ảnh hưởng tới chương trình thiết kế nhưng rất ít và chúng không thể điều tiết dễ dàng các ngôn ngữ phi thủ tục và việc dùng chúng trong ước lượng đòi hỏi một mức độ chi tiết thường khó đạt tới.

3.2.2. Độ đo theo điểm chức năng

Độ đo phần mềm theo điểm chức năng là cách đo gián tiếp cho phần mềm và tiến triển phát triển nó. Thay vì đếm LOC, độ đo theo chức năng tập trung vào chức năng hay tiện ích của chương trình. Độ đo điểm chức

năng ban đầu được Albrecht đề nghị, người gợi ý ra một cách tiếp cận đo hiệu năng gọi là phương pháp điểm chức năng. Các điểm chức năng (FP) được suy ra bằng cách dùng một quan hệ kinh nghiệm dựa trên các cách đo đếm được trong lĩnh vực thông tin của phần mềm và các khẳng định về độ phức tạp phần mềm.

Điểm chức năng được tính bằng cách hoàn thiện bảng sau :

Bảng 3.2. Tính độ đo điểm chức năng

Tham biến độ đo	Số đếm	Nhân tố trọng số			
		Đơn giản	Trung bình	Phức tạp	
Số đầu vào của người dùng	? x	3	4	6	= ?
Số đầu ra của người dùng	? x	4	5	7	= ?
Số yêu cầu người dùng	? x	3	4	6	= ?
Số các tệp	? x	7	10	15	= ?
Số các giao diện ngoài	? x	5	7	10	= ?
Số đếm tổng cộng					

5 đặc trưng thông tin sẽ được xác định và số đếm được đưa ra tại các vị trí thích hợp. Các giá trị miền thông tin được xác định như sau:

Số đầu vào theo người dùng: Phải tính các đầu vào mà từng người dùng cung cấp dữ liệu theo ứng dụng phân biệt cho phần mềm. Cần phải phân biệt các đầu vào với những câu hỏi.

Số đầu ra theo người dùng: Cũng phải tính tới đầu ra mà từng người dùng cung cấp thông tin theo ứng dụng cho phần mềm. Trong hoàn cảnh này, đầu ra là các báo cáo màn hình, thông báo lỗi. Từng khoản mục dữ liệu riêng lẻ bên trong một báo cáo không được tính tách biệt.

Số yêu cầu người dùng dưới dạng các câu hỏi của người dùng: Câu hỏi được định nghĩa như một đầu vào trực tuyến sinh ra một đáp ứng ngay lập tức của phần mềm dưới dạng đầu ra trực tuyến. Mỗi câu hỏi phân biệt phải được tính tới.

Số các tệp : Các tệp dữ liệu được sử dụng.

Số các giao diện ngoài: Mọi giao diện máy móc được (như các tệp dữ liệu trên băng hay đĩa) được dùng để truyền thông tin sang hệ thống khác đều cần phải tính tới.

Khi dữ liệu trên đã được thu thập thì gán cho từng số đếm một giá trị. Các tổ chức sử dụng phương pháp điểm chức năng hiện vẫn đang phát triển các tiêu chí để xác định một mục tiêu là đơn giản, trung bình hay phức tạp. Tuy nhiên việc xác định độ phức tạp theo cách nào đó vẫn còn mang nhiều tính chủ quan.

Để tính các điểm chức năng người ta dùng mối quan hệ sau đây:

$$FP = \text{tổng số đếm} \times \{ 0,65 + 0,01 \times \text{SUM}(F_i) \}$$

Với tổng số đếm là tổng của tất cả các mục FP có được trong bảng. F_i (i = 1 tới 14) là giá trị điều chỉnh độ phức tạp dựa trên việc đáp ứng các câu hỏi trong bảng 3.3. Các giá trị hằng trong phương trình trên và nhân tố trọng số áp dụng cho các số đếm trong lĩnh vực thông tin được xác định theo kinh nghiệm.

Khi đã tính được các điểm chức năng thì có thể sử dụng chúng theo cách tương tự như LOC để đo hiệu năng, phẩm chất và các thuộc tính khác của phần mềm :

Hiệu năng = FP/người-tháng

Phẩm chất = khiếm khuyết/FP

Chi phí = \$/FP

Tư liệu = Số trang tư liệu/FP

Bảng 3.3. Tính điểm chức năng

Tỷ lệ nhân tố theo thang từ 0 đến 5					
0	1	2	3	4	5
Không ảnh hưởng	Ngẫu nhiên	Đơn giản	Trung bình	Có ý nghĩa	Bản chất

F_i :

1. Hệ thống có đòi hỏi sao lưu và khôi phục tin cậy hay không?
2. Có đòi hỏi trao đổi dữ liệu không?

3. Có chức năng xử lý phân bố không?
4. Có đòi hỏi cao về làm việc tốt không?
5. Hệ thống có chạy trong một môi trường hiện có nặng về vận hành tiện tích không?
6. Hệ thống có đòi hỏi việc vào dữ liệu trực tuyến không?
7. Việc vào dữ liệu trực tuyến có đòi hỏi phải xây dựng giao tác đưa vào thông qua nhiều màn hình hay thao tác không?
8. Tập chính có phải cập nhật trực tuyến hay không?
9. Đầu vào, đầu ra, tệp, câu hỏi có phức tạp không?
10. Xử lý bên trong có phức tạp không?
11. Mã chương trình có được thiết kế dễ dùng lại không?
12. Việc chuyển đổi và cài đặt có được đưa vào trong thiết kế hay không?
13. Hệ thống có được thiết kế cho nhiều cài đặt trong các tổ chức khác nhau không?
14. Liệu ứng dụng có được thiết kế để làm thuận tiện cho việc thay đổi và dễ dàng cho người dùng?

3.2.3. Độ đo điểm chức năng mở rộng

Độ đo điểm chức năng ban đầu được thiết kế để ứng dụng cho các ứng dụng hệ thống thông tin nghiệp vụ. Để thích hợp với các ứng dụng này, chiều dữ liệu (giá trị miền thông tin đã đề cập ở trên) được nhấn mạnh, bỏ qua chiều chức năng và hành vi (điều khiển). Vì lý do này, mà độ đo điểm chức năng không thích hợp với các hệ thống kỹ nghệ và nhúng (các hệ thống nhấn mạnh vào chức năng và điều khiển). Khái niệm độ đo điểm chức năng mở rộng được đưa ra để giải quyết tình trạng trên.

Một mở rộng điểm chức năng gọi là điểm tính năng (feature points) là một tập con của độ đo điểm chức năng được áp dụng cho các ứng dụng hệ thống và công nghệ phần mềm. Độ đo điểm tính năng thích hợp với các ứng dụng có các giải thuật phức tạp. Các ứng dụng thời gian thực, điều khiển tiến trình và các ứng dụng nhúng có các thuật toán phức tạp, do đó phải sử dụng điểm tính năng.

Để tính điểm tính năng, phải xác định các giá trị miền thông tin và đánh trọng số như đã mô tả ở trên. Bên cạnh đó, độ đo điểm tính năng còn tính đến đặc trưng phần mềm mới: thuật toán. Thuật toán được định nghĩa là : “một vấn đề tính toán có giới hạn được bao hàm trong một chương trình máy tính đặc biệt”. Việc lấy nghịch đảo ma trận, giải mã một xâu bit, hay xử lý một chương trình điều khiển máy tính tất cả đều là các thí dụ về thuật toán.

Một mở rộng điểm chức năng cho hệ thống thời gian thực và sản phẩm công nghệ đã được Boeing phát triển. Cách tiếp cận của Boeing tích hợp hướng dữ liệu của phần mềm với hướng chức năng và điều khiển để cung cấp một độ đo hướng chức năng gọi là điểm chức năng 3D.

Hướng dữ liệu được tính giống như phần trên. Các dữ liệu được giữ lại (cấu trúc dữ liệu bên trong chương trình, file) và các dữ liệu bên ngoài (đầu vào, đầu ra, các tham chiếu bên ngoài) được sử dụng cùng với các độ đo tính phức tạp nhận được từ việc đếm các hướng dữ liệu.

Hướng chức năng được tính bằng cách xem xét “số các phép toán bên trong được yêu cầu để biến đổi dữ liệu từ đầu vào thành đầu ra”. Để tính điểm chức năng 3D, ta coi sự biến đổi là một dãy các bước xử lý bị ràng buộc bởi các câu ngữ nghĩa (semantic statement). Nói chung, một sự biến đổi được hoàn thành với một thuật toán mà kết quả là chuyển dữ liệu đầu vào thành dữ liệu đầu ra. Các bước xử lý lấy dữ liệu từ file và đơn giản là đặt dữ liệu vào trong bộ nhớ chương trình không được coi là một sự biến đổi. Dữ liệu bản thân nó không được thay đổi theo bất kỳ cách cơ bản nào.

Mức độ phức tạp xác định cho mỗi sự biến đổi là một hàm của số bước xử lý và số các câu ngữ nghĩa điều khiển các bước xử lý. Bảng 3.4 cung cấp một hướng dẫn cho sự thiết lập tính phức tạp cho hướng chức năng.

Bảng 3.4. Tính độ đo điểm tính năng

Tham biến độ đo	Số đếm	Trọng số	Độ đo điểm chức năng
Số đầu vào theo người dùng	?	x 4	=
Số đầu ra theo người dùng	?	x 5	=
Số yêu cầu người dùng	?	x 4	=
Số các tệp	?	x 7	=
Số các giao diện ngoài	?	x 7	=
Thuật toán	?	x 3	=
Số đếm tổng cộng			

Hướng điều khiển được tính bằng cách đếm số lượng biến đổi giữa các trạng thái. Một trạng thái mô tả một ứng xử quan sát được từ bên ngoài và sự biến đổi xảy ra khi có một số sự kiện gây ra cho hệ thống hay phần mềm thay đổi các ứng xử của nó.

Để tính độ đo điểm chức năng 3D, ta sử dụng biểu thức sau:

$$\text{Index} = I + O + Q + F + E + T + R$$

Trong đó I, O, Q, F, E, T, R là các giá trị trọng số phức tạp cho các thành phần: đầu vào, đầu ra, câu hỏi, cấu trúc dữ liệu bên trong, file ngoài, sự biến đổi và chuyển. Mỗi giá trị trọng số phức tạp này lại được tính bằng công thức sau:

$$\text{Giá trị trọng số phức tạp} = N_{il} W_{il} + N_{ia} W_{ia} + N_{ih} W_{ih}$$

Với N_{il} , N_{ia} , N_{ih} là số đếm cho mỗi thành phần với mỗi mức độ của độ phức tạp (cao, thấp, trung bình) và W_{il} , W_{ia} và W_{ih} là các trọng số tương ứng. Toàn bộ cách tính điểm chức năng 3D được cho trong bảng 3.5.

Bảng 3.5. Bảng quyết định độ phức tạp của các biến đổi cho điểm chức năng 3D

Số câu ngữ nghĩa Số bước xử lý	1 – 5	6 – 10	11 +
1 – 10	Thấp	Thấp	Trung bình
11 – 20	Thấp	Trung bình	Cao
21 +	Trung bình	Cao	Cao

Cần chú ý rằng các điểm tính năng, các điểm chức năng và các điểm chức năng 3D biểu thị cho cùng một khái niệm – “chức năng” hay “tiện ích” do phần mềm cung cấp. Trong thực tế, các cách đo này đều cho giá trị FP đối với các ứng dụng tính toán công nghệ quy ước hay hệ thống tin. Đối với các hệ thống thời gian thực phức tạp hơn, số đếm điểm tính năng thường nằm trong khoảng từ 20 ÷ 35%, cao hơn số đếm được xác định bằng cách chỉ dùng điểm chức năng.

Độ đo điểm chức năng (hay độ đo điểm tính năng) như LOC vẫn còn đang được bàn cãi nhiều. Những người ủng hộ nói rằng FP phụ thuộc vào ngôn ngữ lập trình, lý tưởng cho các ứng dụng có dùng các ngôn ngữ quy

ước hay phi thủ tục; họ cũng nói rằng nó dựa trên dữ liệu nên có thể sử dụng được ngay từ rất sớm trong tiến trình của một dự án. Những người phản đối lại cho rằng phương pháp này tính toán dựa nhiều vào chủ quan, không mang tính khách quan, không dựa vào dữ liệu, thông tin khó thâm nhập và FP không có ý nghĩa vật lý trực tiếp, nó chỉ là con số.

3.3. Lập lịch dự án phần mềm

Lập lịch các dự án phát triển phần mềm có thể xem xét theo hai bối cảnh khá khác nhau. Quan điểm thứ nhất là ngày cuối cùng cần phải bàn giao hệ thống trên máy tính đã được ấn định và không thể thay đổi. Về tổ chức, phần mềm bị ràng buộc phải phân bổ nỗ lực trong khuôn khổ thời gian quy định. Quan điểm thứ hai của việc lập lịch phần mềm giả sử rằng giới hạn thời gian đã được thảo luận, nhưng ngày cuối đó do tổ chức làm phần mềm ấn định. Nỗ lực sẽ được phân bổ để việc sử dụng các nguồn tài nguyên được tốt nhất và ngày cuối đó sẽ được xác định sau khi phân tích kỹ lưỡng phần mềm.

Độ chính xác trong việc lập lịch đôi khi quan trọng hơn độ chính xác về chi phí. Trong một môi trường hướng sản phẩm, giá phụ trội có thể được làm lại giá hay khấu hao trên số bán lớn. Tuy nhiên, việc lập lịch thiếu sót có thể làm giảm tác động thị trường, làm cho khách hàng không hài lòng và làm tăng chi phí nội bộ do nảy sinh các vấn đề phụ khi tích hợp hệ thống.

3.3.1. Mối quan hệ con người – công việc

Trong một dự án phát triển phần mềm nhỏ, một người có thể phân tích các yêu cầu, thực hiện thiết kế, sinh mã và tiến hành kiểm thử. Khi dự án lớn, sẽ phải có nhiều người cùng tham dự.

Có một quan điểm mà vẫn được nhiều nhà quản lý tin dùng: “nếu chúng ta tụt lại so với lịch biểu thì ta có thể thêm người lập trình và đuổi kịp về sau trong dự án.” Tuy nhiên, việc thêm người về sau trong dự án thường làm ảnh hưởng phá vỡ dự án, gây cho lịch biểu càng bị trượt thêm nữa. Nguyên nhân là do người được bổ sung thêm phải học về hệ thống và người dạy họ lại là những người đang làm công việc đó. Trong khi dạy thì họ không làm thêm công việc và dự án bị tụt lại phía sau.

Bên cạnh vấn đề thời gian cần để học hệ thống, số người tăng lên thì số đường liên lạc cũng tăng lên và độ phức tạp trong toàn bộ hệ thống cũng tăng lên. Mặc dù liên lạc là cơ sở cho việc phân tích phần mềm hiệu quả nhưng các đường liên lạc mới đều yêu cầu thêm nỗ lực và do đó cần thêm thời gian.

Lấy một ví dụ, ta xét 4 kỹ sư phần mềm, mỗi người có khả năng tạo ra 5000 LOC/năm khi làm việc trên một dự án riêng lẻ. Khi 4 kỹ sư này tham gia vào một nhóm dự án thì có 6 đường liên lạc tiềm năng. Mỗi đường liên lạc đòi hỏi thời gian mà đáng ra có thể được dành cho việc phát triển phần mềm. Chúng ta sẽ giả sử hiệu năng nhóm (khi được đo theo LOC) sẽ phải thu lại 250LOC/năm cho mỗi đường liên lạc, do tổng phí liên kết với việc liên lạc. Do đó hiệu năng nhóm là $20000 - (250 \times 6) = 18500$ LOC/năm.

$$\frac{20000 - 18500}{20000} 100\% = 7,5\%$$

nhỏ hơn điều ta mong đợi.

Dự án một năm mà nhóm trên đang tiến hành bị trượt ra ngoài lịch biểu và với hai tháng còn lại, hai người nữa được bổ sung vào nhóm. Số các đường liên lạc tăng lên 15. Dẫu vào hiệu năng của nhân viên mới tương đương với $840 \times 2 = 1680$ LOC cho 2 tháng còn lại trước khi giao hàng. Hiệu năng nhóm bây giờ là $20000 + 1680 - (250 \times 15) = 14570$ LOC/năm.

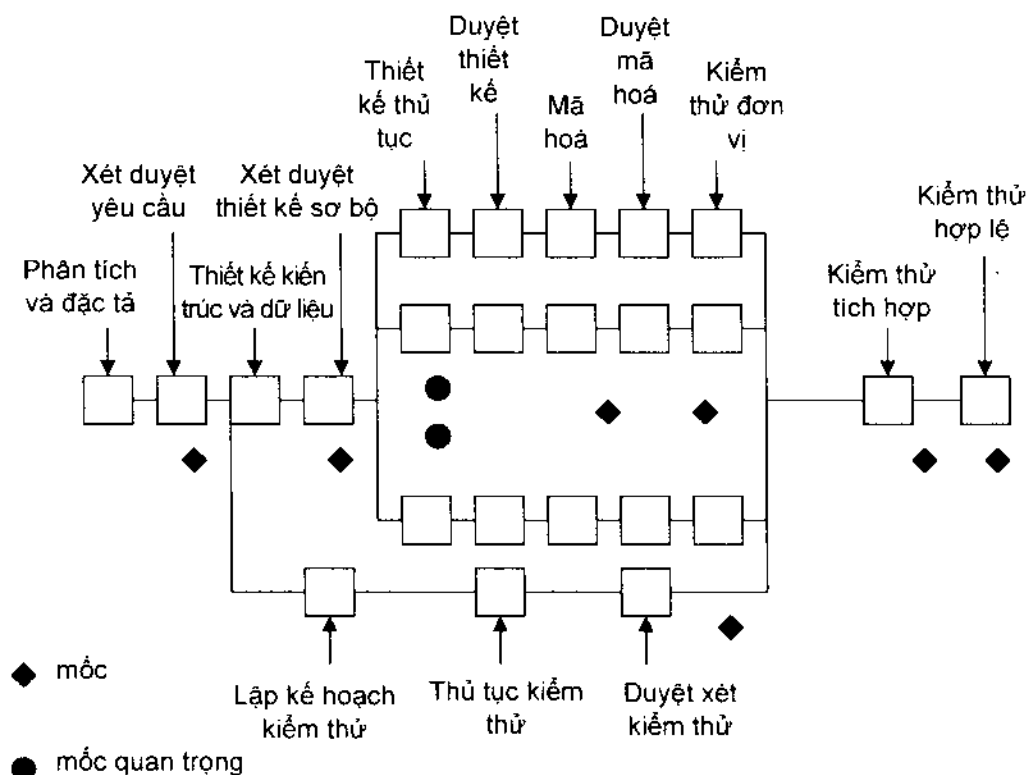
Ví dụ trên minh họa cho một luận điểm là mối quan hệ giữa số người làm việc cho dự án phần mềm và hiệu năng tổng thể là không tuyến tính.

Dựa trên mối quan hệ người – công việc như vậy thì nhóm có phải là phản hiệu năng không? Câu trả lời là không nếu liên lạc phục vụ cho việc cải thiện chất lượng và tính bảo trì cho phần mềm. Trong thực tế, các cuộc họp tổng kết kỹ thuật do các nhóm phát triển phần mềm tiến hành có thể giúp cho việc phân tích và thiết kế phần mềm tốt hơn, và quan trọng hơn, nó có thể làm giảm số lỗi chưa được phát hiện vào lúc kiểm thử (rút gọn được công sức kiểm thử). Do vậy, tính hiệu năng và chất lượng khi được đo theo thời gian hoàn thành dự án và thoả mãn khách hàng thực tế có thể được cải tiến.

3.3.2. Xác định nhiệm vụ và cơ chế song song

Khi có nhiều người cùng tham gia trong một dự án công nghệ phần mềm thì rất có thể các hoạt động phát triển sẽ được thực hiện song song.

Hình 3.2 chỉ ra một sơ đồ mạng nhiệm vụ cho một dự án công nghệ phần mềm nhiều người diễn hình. Mạng này biểu thị cho tất cả các nhiệm vụ dự án, trình tự và sự phụ thuộc của chúng.



Hình 3.2. Mạng nhiệm vụ và cơ chế song song

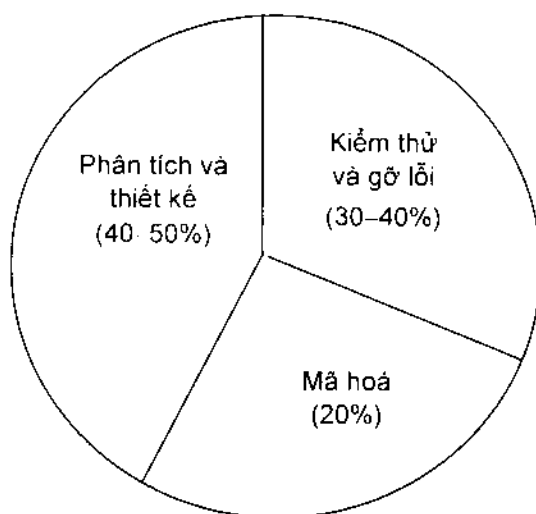
Phân tích, đặc tả và tổng quan về các yêu cầu kết quả là những nhiệm vụ đầu tiên cần phải thực hiện và đặt nền tảng cho các nhiệm vụ song song theo sau. Khi các yêu cầu đã được xác định và xem xét thì hoạt động thiết kế (thiết kế kiến trúc và dữ liệu) và việc vạch kế hoạch kiểm thử có thể bắt đầu. Bản chất module của phần mềm thiết kế tốt tự bản thân nó đã giúp cho việc theo dõi sự phát triển song song thiết kế thủ tục, mã hoá, và kiểm thử đơn vị, như được minh họa trong hình trên.

Khi các thành phần của phần mềm được hoàn tất, nhiệm vụ kiểm thử tích hợp bắt đầu. Cuối cùng là nhiệm vụ kiểm thử hợp lệ phần mềm, sau khi thực hiện nhiệm vụ này, phần mềm sẵn sàng bàn giao cho khách hàng.

Trên hình 3.2 cho thấy, điều quan trọng cần lưu ý là các mốc được đặt ở những khoảng đều đặn trong toàn bộ tiến trình công nghệ cung cấp cho nhà quản lý một chỉ báo đều đặn về tiến độ. Các mốc được đặt mỗi khi tài liệu được tạo ra xem như một phần của nhiệm vụ công nghệ phần mềm đã được xem xét thành công.

Bản chất tương tranh của các hoạt động công nghệ phần mềm dẫn tới một số yêu cầu quan trọng khi lập lịch. Các nhiệm vụ song song thường xuất hiện không đồng bộ nên người lập kế hoạch phải xác định sự phụ thuộc giữa các nhiệm vụ để đảm bảo tiến độ liên tục. Bên cạnh đó, người quản lý dự án cũng cần biết về những nhiệm vụ nằm trên đường găng, tức là những nhiệm vụ phải được hoàn thành theo lịch.

3.3.3. Phân bố nguồn lực



Hình 3.3. Phân bố nỗ lực

Hình 3.3 mô tả một sự phân bố về công sức cần thực hiện cho các giai đoạn xác định và phát triển. Phân bố này còn được gọi là quy tắc 40 – 20 – 40, nhấn mạnh vào các nhiệm vụ phân tích, thiết kế đầu trước và kiểm thử sau.

Phân bố công sức được thể hiện trong hình 3.3 chỉ nên sử dụng như bản hướng dẫn. Tùy theo đặc trưng của từng dự án sẽ quyết định cách phân phối công sức. Công sức dành cho việc lập kế hoạch dự án hiếm khi được tính lớn hơn 2 ÷ 3% công sức dự án, trừ khi kế hoạch liên quan đến một tổ

chức với chi phí lớn và rủi ro cao. Việc phân tích yêu cầu có thể chiếm $10 \div 25\%$ công sức dự án. Công sức dành cho phân tích và làm bản mẫu cần phải tăng lên tỷ lệ trực tiếp với kích cỡ và độ phức tạp dự án. $20 \div 25\%$ công sức dự án thông thường được dành cho thiết kế phần mềm. Thời gian dành cho tổng quan thiết kế và việc lập về sau cũng phải được xem xét.

Do công sức được dành cho thiết kế phần mềm nên việc mã hoá đi theo tương đối ít khó khăn, có thể chiếm $15 \div 20\%$ trên tổng số công sức. Kiểm thử và gỡ lỗi sau đó có thể chiếm $30 \div 40\%$ công sức phát triển phần mềm. Sự căng của phần mềm (khả năng hệ thống có thể làm việc nhiều giờ liên tục) thường quyết định khối lượng kiểm thử cần tới. Nếu phần mềm bị lỗi do con người thì có thể xem xét tăng số phần trăm nỗ lực cao hơn dành cho việc này.

3.3.4. Phương pháp lập lịch

Lập lịch cho dự án phần mềm không khác nhiều với việc lập lịch cho bất kỳ nỗ lực phát triển đa nhiệm nào. Do đó, các công cụ và kỹ thuật lập lịch dự án tổng quát có thể được áp dụng cho phần mềm nhưng có một số sửa đổi.

Kỹ thuật xem xét và đánh giá chương trình (PERT) và phương pháp đường gãy là hai phương pháp lập lịch dự án được áp dụng cho việc phát triển phần mềm. Cả hai kỹ thuật này đều xây dựng việc mô tả dự án theo mạng nhiệm vụ, tức là, việc biểu diễn các nhiệm vụ theo hình hay bảng được xác định bằng cách xây dựng một danh sách tất cả các nhiệm vụ, đôi khi còn gọi là bảng công việc (WBS) liên kết với một dự án xác định và một danh sách có trật tự (đôi khi còn được gọi là danh sách hạn chế) chỉ ra các nhiệm vụ phải thực hiện theo thứ tự nào.

Cả PERT và sơ đồ đường gãy (Critical Path method – CPM) đều cung cấp các công cụ định lượng cho phép người lập kế hoạch phần mềm:

- Xác định đường gãy – đây chuyển các nhiệm vụ xác định thời hạn dự án.
- Thiết lập ước lượng thời gian thích hợp cho từng nhiệm vụ bằng cách áp dụng mô hình thống kê.
- Tính toán thời gian biên, nhằm tạo ra “cửa sổ thời gian” cho một nhiệm vụ đặc biệt.

Việc tính thời gian rất có ích trong việc lập lịch dự án. Sai lệch thiết kế của một chức năng có thể làm chậm việc xây dựng các chức năng khác. Rigg mô tả thời gian biên quan trọng trong mạng PERT hay CPM:

1. Thời gian sớm nhất mà một nhiệm vụ có thể bắt đầu.
2. Thời gian muộn nhất cho việc khởi đầu nhiệm vụ trước khi hoàn thành dự án tối thiểu bị trễ.
3. Sự kết thúc sớm nhất – tổng của việc bắt đầu sớm và thời hạn nhiệm vụ.
4. Sự kết thúc muộn nhất – thời gian bắt đầu muộn nhất cộng với thời gian nhiệm vụ.
5. Tổng độ nổi – thời gian vượt quá mức hoặc chậm trễ được phép theo các nhiệm vụ lập lịch để đường găng trên mạng được duy trì theo lịch.

Việc tính toán thời gian biên dẫn tới việc xác định đường găng và cung cấp cho người quản lý một phương pháp định lượng để ước tính tiến độ cũng như nhiệm vụ cần hoàn thành.

3.4. Rủi ro

Phân tích rủi ro thực tế gồm 4 hoạt động phân biệt : xác định rủi ro, dự phòng rủi ro, định giá rủi ro và quản lý rủi ro.

3.4.1 Xác định rủi ro

Có thể phân loại rủi ro theo nhiều cách. Có thể xác định ở mức vĩ mô gồm các rủi ro dự án, rủi ro kỹ thuật, rủi ro nghiệp vụ. Rủi ro dự án xác định các vấn đề yêu cầu, khách hàng, tài nguyên, nhân sự, ngân sách tiềm năng và ảnh hưởng của chúng lên dự án phần mềm. Độ phức tạp, kích cỡ và cấu trúc dự án cũng được xác định như các nhân tố rủi ro. Rủi ro kỹ thuật xác định các vấn đề tiềm năng về thiết kế, cài đặt, giao diện, kiểm thử và bảo trì. Bên cạnh đó, độ mơ hồ, độ bất trắc kỹ thuật, sự lạc hậu kỹ thuật cũng là các nhân tố rủi ro. Rủi ro kỹ thuật xuất hiện trong trường hợp gặp phải những vấn đề khó giải quyết hơn ta tưởng. Rủi ro nghiệp vụ xảy ra ngay cả khi dự án phần mềm rất tốt. Có 5 loại rủi ro nghiệp vụ là:

1. Xây dựng một sản phẩm tuyệt vời nhưng không có người sử dụng (rủi ro tiếp thị).

2. Xác định một sản phẩm không thích hợp với chiến lược sản phẩm tổng thể công ty.

3. Xác định một sản phẩm mà người bán hàng không hiểu làm sao bán được.

4. Mất sự hỗ trợ của cấp quản lý cao do thay đổi mối quan tâm hay do thay đổi con người (rủi ro quản lý).

5. Mất cam kết về tài chính hay nhân sự (rủi ro ngân sách).

Xác định rủi ro, liệt kê những rủi ro dự án riêng trong toàn bộ các rủi ro đã được nêu ở trên. Một trong những phương pháp tốt nhất để nhận biết từng rủi ro là dùng một tập các câu hỏi giúp cho người lập kế hoạch dự án hiểu được rủi ro trong dự án. Bohm gợi ý dùng từ “ khoản mục rủi ro” chỉ một tập các câu hỏi liên quan tới từng nhân tố rủi ro. Người lập kế hoạch có thể nhận ra các rủi ro về nhân viên bằng cách trả lời các câu hỏi sau:

- Có người giỏi nhất không?
- Mọi người có tổ hợp đúng tài năng không?
- Có đủ người sẵn có không?
- Các nhân viên có cam kết theo đuổi toàn bộ dự án không?
- Một số nhân viên dự án chỉ dành một phần thời gian cho dự án?
- Các nhân viên có mong đợi đúng việc hiện được giao không?
- Các nhân viên có được huấn luyện đầy đủ không?
- Có hay luân chuyển nhân viên làm gián đoạn công việc không?

Sự chắc chắn tương đối của các câu trả lời cho những câu hỏi này cho phép người lập kế hoạch ước lượng được ảnh hưởng của rủi ro.

3.4.2. Dự phòng rủi ro

Dự phòng rủi ro còn được gọi là ước lượng rủi ro, cố gắng xác định tỷ lệ rủi ro theo hai cách – có thể xảy ra tức là rủi ro có thực và hậu quả của vấn đề liên quan tới rủi ro đó, nếu nó xuất hiện. Người lập kế hoạch dự án cùng với các nhà quản lý và các nhân viên kỹ thuật thực hiện 4 hoạt động dự phòng rủi ro:

1. Lập thang phản ánh khả năng cảm nhận được rủi ro.
2. Phác họa những hậu quả của rủi ro.
3. Ước lượng ảnh hưởng của rủi ro lên dự án và sản phẩm.
4. Chú ý đến độ chính xác của dự phòng rủi ro sao cho không có sự hiểu lầm.

Thang được xác định hoặc theo thuật ngữ có – không, định tính hay định lượng. Các câu hỏi trong bảng khoản mục rủi ro có thể được trả lời có hay không. Nhưng điều này không thực tế lắm. Cách tiếp cận tốt hơn là trả lời theo tỷ lệ xác suất định lượng có các giá trị sau: gần như hoàn toàn không có, không thể có, vừa phải, có thể, rất có thể. Theo một cách khác, người lập kế hoạch có thể ước lượng xác suất toán học mà một rủi ro có thể xảy ra bằng cách dùng phân tích thống kê về các độ đo thu được từ các dự án quá khứ, từ trực giác hay các thông tin khác.

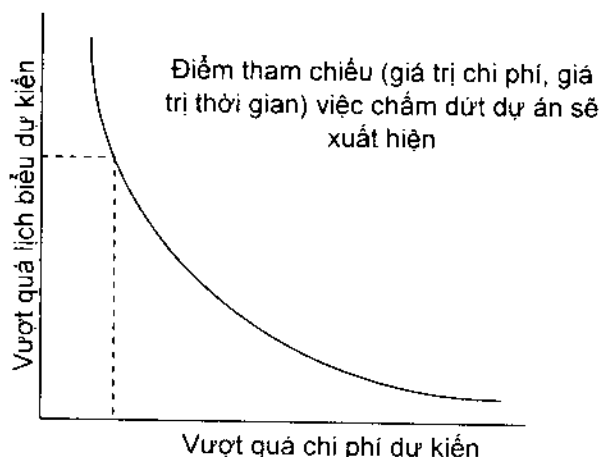
Cuối cùng rủi ro được đánh trọng số bởi ảnh hưởng được cảm nhận (theo dự án) và sau đó được xếp thứ tự ưu tiên. Ba nhân tố chi phối ảnh hưởng: bản chất của rủi ro, phạm vi của nó và thời gian. Chẳng hạn, một giao diện ngoài kém xác định đối với phần cứng (một rủi ro kỹ thuật) sẽ ngăn cản thiết kế sớm và việc kiểm thử rất có thể sẽ dẫn tới vấn đề tích hợp hệ thống về sau trong dự án. Phạm vi của một rủi ro tổ hợp cả sự ngặt nghèo (vấn đề nghiêm trọng đến đâu) với phân bố tổng thể của nó (dự án sẽ bị ảnh hưởng đến đâu hay bao nhiêu khách hàng sẽ phải chịu thiệt hại). Cuối cùng, thời gian của một rủi ro đề cập đến khi nào thì có tác động của rủi ro và nó kéo dài bao lâu.

3.4.3. Định giá rủi ro

Tại điểm này trong tiến trình phân tích rủi ro, ta đã thiết lập được một tập các bộ ba dạng: (r_i, l_i, x_i) , trong đó r_i là rủi ro, l_i là sự xuất hiện (xác suất) của rủi ro, x_i là tác động của rủi ro. Trong khi định giá rủi ro, chúng ta phải xem xét thêm cả độ chính xác của ước lượng đã được thực hiện trong khi dự phòng rủi ro, nỗ lực để sắp thứ tự ưu tiên các rủi ro đã được phát hiện, và bắt đầu nghĩ về cách kiểm soát và/hoặc ngăn cản sự có thể xuất hiện.

Để việc định giá có ích, phải đưa ra một mức tham khảo rủi ro. Đối với hầu hết các dự án phần mềm, lịch biểu và hiệu suất biểu thị cho 3 mức tham

khảo rủi ro. Tức là, có một mức cho việc chi phí quá mức, trượt lịch biểu hay suy giảm chất lượng sẽ làm cho dự án kết thúc. Nếu một tổ hợp các rủi ro vượt quá một hay nhiều mức tham khảo này thì công việc sẽ bị dừng lại. Trong phân tích rủi ro phần mềm, một mức tham khảo rủi ro chỉ có một điểm được gọi là điểm tham khảo hay điểm vỡ tại đó quyết định tiếp tục xử lý dự án hay kết thúc nó (vấn đề quá lớn) đều được chấp nhận ngang nhau.



Hình 3.4. Mức tham khảo rủi ro

Hình 3.4 biểu thị cho tình huống này bằng đồ thị. Nếu một tổ hợp các rủi ro dẫn đến quá mức chi phí và lịch biểu thì sẽ có một mức, được biểu thị bằng đường cong trong hình, khi vượt quá mức đó sẽ làm cho dự án phải kết thúc. Tại điểm tham khảo, các quyết định xử lý hay kết thúc đều có trọng số như nhau.

Trong thực tế, mức tham khảo hiếm khi được biểu thị như một đường mịn trên đồ thị. Trong phần lớn các trường hợp nó là một miền trong đó có những vùng bất trắc, tại đó nỗ lực để dự đoán một quyết định quản lý dựa trên việc tổ hợp các giá trị tham khảo thường là không thể làm được.

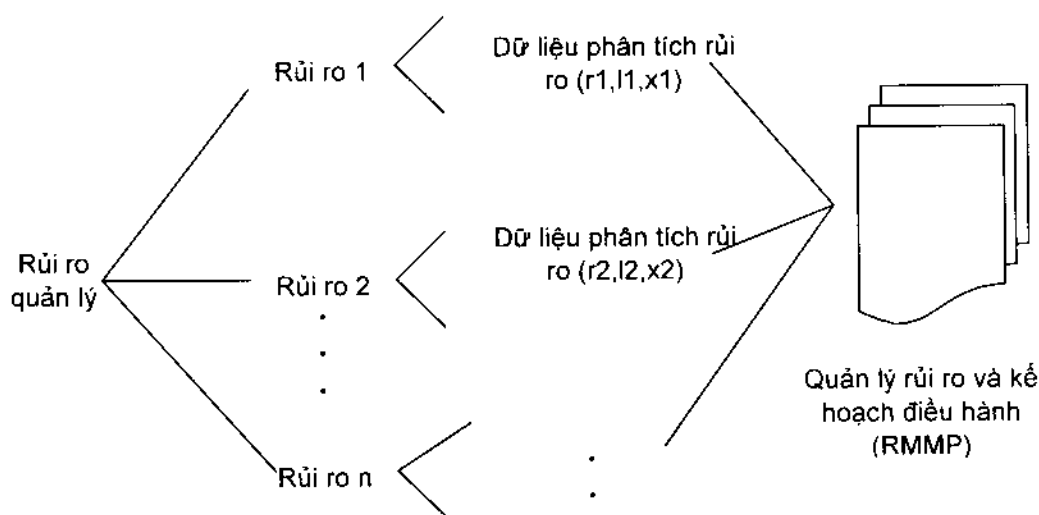
Do đó, trong khi định giá rủi ro, ta thực hiện các bước sau:

1. Xác định các mức tham khảo rủi ro cho dự án.
2. Cố gắng xây dựng mối quan hệ từng $[r_i, l_i, x_i]$ và từng mức tham khảo.
3. Dự đoán tập các điểm tham khảo.

4. Cố gắng dự đoán việc các tổ hợp thành của các rủi ro sẽ ảnh hưởng thế nào tới mức tham khảo.

3.4.4. Quản lý và điều phối rủi ro

Hoạt động quản lý rủi ro và điều phối rủi ro được minh họa như hình 3.5.



Hình 3.5. Quản lý và điều phối rủi ro

Bộ ba (mô tả rủi ro, sự có thể xảy ra và tác động) liên kết với từng rủi ro được coi như cơ sở để từ đó phát triển các bước quản lý rủi ro (cũng được gọi là các bước khắc phục rủi ro). Chẳng hạn, giả sử rằng việc quay vòng nhân viên nhiều lần có thể coi như một rủi ro dự án r_i . Dựa trên lịch sử quá khứ và trực giác quản lý, khả năng có thể xảy ra, l_i của việc quay vòng nhiều lần được ước lượng là 0,7 và tác động x_i được dự phòng để tăng thời hạn dự án lên 15% và chi phí tổng thể lên 12%. Với những dữ liệu này, người ta đề nghị các bước quản lý rủi ro sau:

- Hợp với nhóm nhân viên hiện tại để xác định các nguyên nhân luân chuyển (như điều kiện làm việc kém, lương thấp, thị trường công việc cạnh tranh).
- Tiến hành các hoạt động để giảm nhẹ các nguyên nhân rủi ro dưới quyền kiểm soát của chúng ta trước khi dự án bắt đầu.

- Khi dự án bắt đầu, phải giả sử việc luân chuyển sẽ xuất hiện và đưa ra các phương án giải quyết dự phòng để đảm bảo sự liên tục khi có người ra đi.
- Tổ chức nhóm dự án sao cho thông tin về từng hoạt động được phổ biến rộng rãi.
- Xác định các chuẩn tài liệu và thiết lập cơ chế để đảm bảo rằng tài liệu được xây dựng đúng hạn.
- Tiến hành duyệt xét ngang cấp cho toàn bộ công việc.

Điều quan trọng cần lưu ý là những bước quản lý rủi ro này phải chịu thêm chi phí dự án phụ. Chẳng hạn, thời gian dành để “dự phòng” cho các nhà kỹ thuật chủ chốt làm tốn thêm chi phí. Do đó phần quản lý rủi ro phải đánh giá khi nào lợi ích được tích lũy vì chi phí cho các bước quản lý dự án nhiều khi còn vượt trội hơn chi phí liên kết thực hiện chúng. Về thực chất, người lập kế hoạch dự án thực hiện việc phân tích chi phí – lợi ích cổ điển. Nếu các bước khắc phục rủi ro cho việc luân chuyển cán bộ nhiều sẽ làm tăng chi phí và thời hạn dự án lên khoảng 15% và chi phí sao lục trội lên thì cấp quản lý có thể quyết định không thực hiện bước quản lý rủi ro này. Nếu các bước khắc phục rủi ro được dự phòng làm tăng chi phí lên 5% và thời hạn kéo dài thêm 3% thì cấp quản lý rất có thể đưa chúng vào kế hoạch.

Với một dự án lớn, có thể xác định 30 hay 40 rủi ro. Nếu xác định được cho mỗi rủi ro từ 3 ÷ 7 bước quản lý rủi ro thì bản thân việc quản lý rủi ro lại trở thành một dự án. Bởi lý do này chúng ta sử dụng quy tắc Pareto 80/20 cho rủi ro phần mềm. Kinh nghiệm chỉ ra rằng 80% của toàn bộ rủi ro dự án (tức là 80% tiềm năng lỗi dự án) có thể được tính chỉ bởi 20% của các rủi ro được xác định. Công việc được thực hiện trong các bước phân tích rủi ro ban đầu sẽ giúp cho người lập kế hoạch xác định những rủi ro nào rơi vào trong 20% đó. Với lý do này, một số rủi ro được xác định, được định giá và dự phòng không thể trở thành kế hoạch quản lý rủi ro, chúng không chứa 20% cần thiết.

Các bước quản lý rủi ro được tổ chức thành kế hoạch quản lý và điều phối rủi ro (RMMP). RMMP làm tư liệu cho các công việc thực hiện, xem như một phần của phân tích rủi ro và được người quản lý dự án dùng như một phần trong kế hoạch dự án.

Dại cương về RMMP được trình bày như sau:

I. Giới thiệu

1. Phạm vi và mục tiêu của tài liệu.
2. Tổng quan.
 - a. Mục tiêu.
 - b. Ưu tiên khắc phục rủi ro.
3. Tổ chức.
 - a. Quản lý.
 - b. Trách nhiệm.
 - c. Mô tả công việc.
4. Mô tả chương trình khắc phục.
 - a. Lịch biểu.
 - b. Các mốc và cuộc họp chính.
 - c. Ngân sách.

II. Phân tích rủi ro.

1. Xác định.
 - a. Tổng quan về rủi ro.
 - b. Phân loại rủi ro.
2. Ước lượng rủi ro.
 - a. Ước lượng xác suất rủi ro.
 - b. Ước lượng hậu quả rủi ro.
 - c. Ước lượng tiêu chí.
 - d. Ước lượng nguồn gây ra rủi ro.
3. Đánh giá.
 - a. Phương pháp đánh giá được dùng.
 - b. Giả thiết và giới hạn của phương pháp đánh giá.
 - c. Đánh giá tham khảo rủi ro
 - d. Đánh giá kết quả

III. Quản lý rủi ro

1. Chọn khuyến cáo
2. Cách khắc phục rủi ro
3. Khuyến cáo khắc phục rủi ro
4. Thủ tục điều phối rủi ro

IV. Phụ lục

1. Ước lượng tình huống rủi ro
2. Kế hoạch giảm rủi ro

Chương 4

YÊU CẦU NGƯỜI DÙNG

4.1. Tại sao phải xác định yêu cầu ?

Yêu cầu phần mềm là tất cả các yêu cầu về phần mềm do khách hàng – người sử dụng đưa ra gồm:

- Các chức năng của phần mềm.
- Hiệu năng của phần mềm.
- Các yêu cầu về thiết kế và giao diện.
- Các yêu cầu đặc biệt khác.

Thông thường các yêu cầu phần mềm được phân thành 4 loại sau:

- Yêu cầu về phần mềm.
- Yêu cầu về phần cứng.
- Yêu cầu về dữ liệu.
- Yêu cầu về con người.

Mục đích của yêu cầu phần mềm là xác định được phần mềm đáp ứng các yêu cầu và mong muốn của khách hàng, người sử dụng.

Ta cần phải xác định yêu cầu phần mềm vì khách hàng chỉ có ý tưởng mơ hồ về phần mềm cần xây dựng để phục vụ công việc của họ. Chúng ta phải tiếp thu các ý tưởng và kiên trì theo đuổi để đi từ các ý tưởng mơ hồ đó cho đến khi tạo ra được phần mềm có đầy đủ các tính năng cần thiết. Trong quá trình thiết kế, khách hàng rất hay thay đổi các đòi hỏi của mình vì vậy chúng ta cần tiếp tục nắm bắt các thay đổi đó và sửa đổi các mô hình cho hợp lý.

4.2. Nội dung xác định yêu cầu phần mềm

- Phát hiện các yêu cầu phần mềm.

- Phân tích các yêu cầu và thương lượng với khách hàng.
- Đặc tả các yêu cầu phần mềm.
- Mô hình hoá hệ thống.
- Kiểm tra tính hợp lý các yêu cầu phần mềm.
- Quản lý các yêu cầu phần mềm.

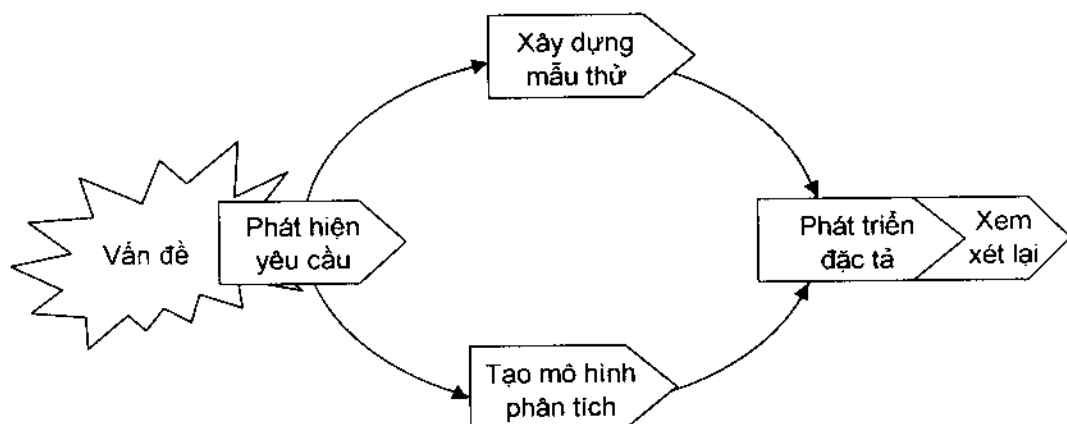
4.2.1. Phát hiện các yêu cầu phần mềm

Ta phải xác định các vấn đề sau:

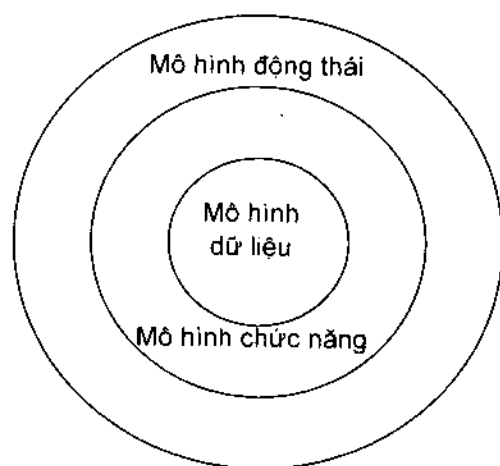
- Phạm vi của phần mềm.
- Hiểu rõ phần mềm.
- Các thay đổi của hệ thống.

Phương pháp xác định yêu cầu phần mềm:

- Xác định các phương pháp sử dụng phát hiện các yêu cầu phần mềm như phỏng vấn, làm việc nhóm, các buổi họp, gặp gỡ đối tác.
- Tìm kiếm nhân sự (chuyên gia, người sử dụng) có hiểu biết sâu sắc nhất, chi tiết nhất về hệ thống giúp chúng ta xác định yêu cầu phần mềm.
- Xác định môi trường kỹ thuật.
- Xác định các ràng buộc lĩnh vực.
- Thu hút sự tham gia của nhiều chuyên gia, khách hàng để có được các quan điểm xem xét phần mềm khác nhau từ phía khách hàng.



Hình 4.1. Quy trình xác định yêu cầu phần mềm



Hình 4.2. Mô hình phân tích

Sản phẩm của phát hiện yêu cầu phần mềm:

- Bảng kê các đòi hỏi và chức năng khả thi của phần mềm.
- Bảng kê phạm vi ứng dụng của phần mềm.
- Mô tả môi trường kỹ thuật của phần mềm.
- Bảng kê các tập hợp các kịch bản sử dụng phần mềm.
- Các nguyên mẫu xây dựng, phân tích hay sử dụng trong phần mềm (nếu có).
- Danh sách nhân sự tham gia vào quá trình phát triển các yêu cầu phần mềm kể cả các nhân sự từ phía công ty khách hàng.

4.2.2. Phân tích các yêu cầu phần mềm và thương lượng với khách hàng

Gồm các công việc sau:

- Phân loại các yêu cầu phần mềm và sắp xếp chúng theo các nhóm liên quan.
- Khảo sát tỷ lệ từng yêu cầu phần mềm trong mối quan hệ của nó với các yêu cầu phần mềm khác.
- Thẩm định từng yêu cầu phần mềm theo các tính chất: phù hợp, đầy đủ rõ ràng, không trùng lặp.

- Phân cấp các yêu cầu phần mềm dựa trên nhu cầu và đòi hỏi của khách hàng/người sử dụng.
- Thẩm định từng yêu cầu phần mềm để xác định chúng có khả năng thực hiện được trong môi trường kỹ thuật hay không, có khả năng kiểm định các yêu cầu phần mềm hay không.
- Thẩm định rủi ro có thể xảy ra với từng yêu cầu phần mềm.
- Đánh giá thô (tương đối) về giá thành và thời gian thực hiện của từng yêu cầu phần mềm trong giá thành sản phẩm phần mềm và thời gian thực hiện phần mềm.
- Giải quyết tất cả các bất đồng về yêu cầu phần mềm với khách hàng/người sử dụng trên cơ sở thảo luận và thương lượng các yêu cầu đặt ra.

4.2.3. Đặc tả yêu cầu phần mềm

Đây là công việc xây dựng các tài liệu trong đó có thể sử dụng các công cụ như: mô hình hoá, mô hình toán học hình thức, tập hợp các kịch bản sử dụng, các nguyên mẫu.

Chất lượng của hồ sơ đặc tả được đánh giá qua các tiêu thức:

- Tính rõ ràng, chính xác.
- Tính phù hợp.
- Tính đầy đủ, hoàn thiện.

Các thành phần của hồ sơ đặc tả:

- Đặc tả phi hình thức (informal specifications) được viết bằng ngôn ngữ tự nhiên.
- Đặc tả hình thức (formal specifications) được viết bằng tập các ký pháp có các quy định về cú pháp và ngữ nghĩa rất chặt chẽ.
- Đặc tả vận hành chức năng (operational specifications) mô tả các hoạt động của hệ thống phần mềm sẽ xây dựng.
- Đặc tả mô tả (descriptive specifications) đặc tả các đặc tính đặc trưng của phần mềm.
- Đặc tả chức năng (functional specifications) : thông thường khi đặc tả chức năng của phần mềm người ta sử dụng các công cụ sau:

- Biểu đồ luồng dữ liệu.
- Máy trạng thái hữu hạn.
- Mạng Petri.

Ta hãy xét lần lượt từng loại công cụ:

Biểu đồ luồng dữ liệu (DFD)

Diễn tả tập hợp các chức năng xử lý của hệ thống cùng với mối liên hệ của chúng trước và sau chức năng xử lý và chuyển giao dữ liệu cho nhau. Biểu đồ luồng dữ liệu là một mô tả động cung cấp một quan sát tổng thể về hệ thống.

Hạn chế của DFD:

- Ý nghĩa các ký pháp sử dụng được xác định bởi các định danh lựa chọn của khách hàng.
- Trong DFD không xác định rõ các hướng thực hiện.
- DFD không xác định rõ sự đồng bộ giữa các chức năng/module.

Máy trạng thái hữu hạn

Máy trạng thái hữu hạn chứa:

- Tập các trạng thái Q.
- Tập các đầu vào I.
- Các chức năng chuyển tiếp.

$$\delta : Q \times I \rightarrow Q$$

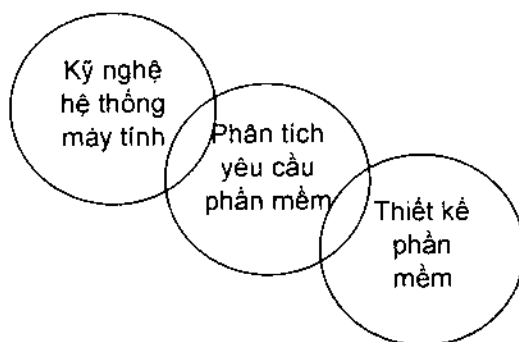
Các yêu cầu của đặc tả tốt:

- Dễ hiểu với người dùng, ít rắc rối.
- Có ít quy ước khi mô tả.
- Theo phong cách từ trên xuống (top – down).
- Dễ triển khai cho các pha sau của vòng đời: thiết kế hệ thống và thiết kế chương trình, giao diện dễ làm, đảm bảo tính nhất quán.

Chương 5

PHÂN TÍCH YÊU CẦU

5.1. Nhiệm vụ của phân tích yêu cầu



Hình 5.1. Phân tích yêu cầu

Phân tích yêu cầu là nhiệm vụ công nghệ phần mềm để bắc nhịp cầu qua lỗ hổng giữa việc cấp phát phần mềm mức hệ thống với thiết kế phần mềm. Phân tích phần mềm giúp cho người phân tích hệ thống xác định được chức năng và hiệu suất của phần mềm, chỉ ra giao diện của phần mềm với các phần tử hệ thống khác và thiết lập những ràng buộc thiết kế mà phần mềm phải đáp ứng. Việc phân tích yêu cầu phần mềm cho phép người kỹ sư thiết kế (người phân tích) tinh chế lại việc cấp phát phần mềm và xây dựng mô hình tiến trình, dữ liệu, và các lĩnh vực hành vi sẽ được phần mềm xử lý. Việc phân tích cung cấp cho người thiết kế phần mềm một cách biểu diễn thông tin và chức năng có thể dịch thành thiết kế dữ liệu, kiến trúc và thủ tục. Cuối cùng, đặc tả yêu cầu cung cấp cho người phân tích và khách hàng các phương tiện để xác định về chất lượng khi phần mềm đã được xây dựng.

Việc phân tích các yêu cầu phần mềm có thể chia thành 5 lĩnh vực:

1. Nhận thức vấn đề.
2. Đánh giá và tổng hợp.
3. Mô hình hoá.
4. Đặc tả.
5. Xét duyệt.

Ban đầu, người phân tích nghiên cứu bản *Đặc tả hệ thống* (nếu có) và bản *Kế hoạch dự án phần mềm*. Điều quan trọng là hiểu phần mềm trong hoàn cảnh hệ thống và xem xét phạm vi ứng dụng phần mềm để thiết lập các ước lượng kế hoạch. Tiếp đó, cần phải thiết lập trao đổi để người phân tích có thể nhận thức đúng được vấn đề. Người phân tích phải lập được mối quan hệ với cấp quản lý và nhân viên kỹ thuật của tổ chức người dùng/khách hàng và tổ chức phát triển phần mềm. Người quản lý dự án được coi nhà người điều phối, tạo điều kiện thuận lợi cho việc thiết lập đường trao đổi. Mục tiêu của người phân tích là nhận thức được các vấn đề cơ bản như người dùng/khách hàng đã cảm nhận.

Việc đánh giá vấn đề và tổng hợp giải pháp là lĩnh vực chính tiếp theo của nỗ lực đối với người phân tích. Người phân tích phải đánh giá luồng và nội dung thông tin, xác định và soạn thảo các chức năng phần mềm, hiểu hành vi phần mềm theo hoàn cảnh của các sự kiện ảnh hưởng tới hệ thống, thiết lập các đặc trưng giao diện, xác định ra những ràng buộc thiết kế. Các nhiệm vụ này đều phục vụ cho việc mô tả vấn đề sao cho có thể tổng hợp vấn đề theo cách tiếp cận hay tổng thể.

Khi đã đánh giá được vấn đề hiện tại và thông tin mong muốn (đầu vào và đầu ra), người phân tích bắt đầu tổng hợp thành một hay nhiều giải pháp. Một hệ thống dựa trên thiết bị cuối trực tuyến sẽ giải quyết một tập vấn đề nhưng cần phải xem liệu nó có thích hợp với phạm vi đã được xác định trong bản *kế hoạch phần mềm* hay không đồng thời phải có một hệ cơ sở dữ liệu để mô phỏng cho khách hàng. Tiến trình ước lượng và tổng hợp được tiếp tục tiến hành cho đến khi cả người phân tích và khách hàng đều cảm thấy rằng phần mềm có thể thích hợp cho các bước phát triển kế tiếp.

Thông qua việc ước lượng và tổng hợp giải pháp, người phân tích tập trung chủ yếu vào “cái gì” chứ không phải là “thế nào”. Hệ thống tạo ra và sử dụng dữ liệu nào, hệ thống phải thực hiện chức năng nào, giao diện nào cần xác định và hệ thống nằm trong các ràng buộc nào?

Trong hoạt động đánh giá và tổng hợp giải pháp, người phân tích tạo ra một mô hình hệ thống để hiểu rõ hơn luồng dữ liệu và điều khiển, xử lý chức năng, thao tác hành vi và nội dung thông tin. Mô hình này là nền tảng cho việc thiết kế phần mềm và là cơ sở cho việc tạo ra đặc tả về phần mềm.

Cần lưu ý là không có được đặc tả chi tiết ở giai đoạn này. Khách hàng không chắc chắn mình muốn gì và người phân tích cũng không chắc cách tiếp cận cụ thể có thể thực hiện đúng chức năng và hiệu suất mong muốn hay không. Bởi những lý do này và nhiều lý do khác nữa, cần tiến hành một cách tiếp cận phân tích yêu cầu đó là làm bản mẫu.

Khi đã mô tả được thông tin, chức năng, hiệu suất, hành vi và giao diện cơ sở, cần phải xác định các tiêu chuẩn hợp lệ để biểu diễn cách hiểu về việc cài đặt các tiêu chuẩn hợp lệ, từ đó biểu diễn cách hiểu về việc cài đặt phần mềm thành công. Những tiêu chuẩn này là cơ sở cho các hoạt động kiểm thử xuất hiện sau này trong tiến trình công nghệ phần mềm. Một đặc tả yêu cầu hình thức được viết ra để xác định các đặc trưng và thuộc tính phần mềm. Bên cạnh đó, cũng có thể soạn nhập “*Tài liệu người dùng sơ lược*” cho trường hợp bản mẫu còn chưa được xây dựng xong.

Các tài liệu phân tích yêu cầu (bản đặc tả và tài liệu người dùng) được dùng làm cơ sở cho việc xét duyệt của cả khách hàng và người phát triển. Cuộc họp xét duyệt yêu cầu bao giờ cũng sẽ nảy sinh việc sửa đổi chức năng, hiệu suất, biểu diễn thông tin, ràng buộc hay tiêu chuẩn hợp lệ.

5.2. Phân tích có cấu trúc

Phân tích có cấu trúc giống như các phương pháp yêu cầu phần mềm, là một hoạt động xây dựng mô hình. Bằng cách dùng một ký pháp duy nhất cho phương pháp phân tích có cấu trúc, chúng ta tạo ra các mô hình mô tả các luồng và nội dung thông tin (dữ liệu và điều kiện), phân hoạch hệ thống theo chức năng, hành vi và mô tả bản chất của nội dung phải xây dựng.

5.2.1. Cơ chế của phân tích có cấu trúc

5.2.1.1. Tạo biểu đồ luồng dữ liệu

Biểu đồ luồng dữ liệu (DFD) giúp kỹ sư phần mềm phát triển được các mô hình về thông tin và chức năng đồng thời. Khi DFD được làm mịn tới các mức chi tiết hơn thì người phân tích thực hiện một sự phân rã chức năng không tường minh của hệ thống. Đồng thời việc làm mịn DFD phát sinh ra một sự làm mịn tương ứng dữ liệu khi nó chuyển qua các tiến trình nằm trong ứng dụng.

Một vài hướng dẫn đơn giản sau có thể trợ giúp cho việc suy ra biểu đồ luồng dữ liệu:

1. Biểu đồ luồng dữ liệu mức 0 nên mô tả chỉ là một vòng tròn phần mềm/hệ thống.
2. Đầu vào và đầu ra chính cần được chú thích cẩn thận.
3. Việc làm mịn dần nên bắt đầu từ việc cô lập các tiến trình, khoản mục dữ liệu và kho ứng cử viên được biểu diễn ở mức tiếp.
4. Tất cả các mũi tên và hình tròn nên được gắn nhãn bằng những tên có ý nghĩa.
5. Phải duy trì được tính liên tục luồng thông tin từ mức này sang mức kia.
6. Mỗi lúc chỉ làm mịn một hình tròn.

Có một khuynh hướng tự nhiên hay làm phức tạp hoá biểu đồ luồng dữ liệu. Điều này xuất hiện khi người phân tích chỉ ra nhiều chi tiết quá sớm hay biểu diễn các khía cạnh thủ tục của phần mềm thay vì luồng thông tin.

Trong DFD mức 0, các tác nhân ngoài chủ yếu tạo ra thông tin cho hệ thống và tiêu thụ các thông tin do hệ thống sinh ra. Các mũi tên có nhãn biểu thị cho khoản mục dữ liệu hợp thành, tức là các dữ liệu thực là một tập hợp của nhiều khoản mục dữ liệu phụ.

DFD mức 0 được mở rộng thành DFD mức 1. Một cách tiếp cận đơn giản và hiệu quả là thực hiện “phân tích văn phạm”. Tức là, chúng ta cô lập tất cả các danh từ và cụm danh từ, động từ và cụm động từ trong lời thuật trên. Tất cả các danh từ đều hoặc là các tác nhân ngoài, các khoản mục dữ liệu, các điều khiển hay các kho dữ liệu. Do đó, bằng cách thực hiện phân tích văn phạm lời thuật về xử lý cho một hình tròn ở bất kỳ mức DFD nào

chúng ta có thể tạo ra rất nhiều thông tin có ích về cách tiến hành làm mịn cho mức tiếp theo.

Các tiến trình được biểu diễn ở DFD mức 1 có thể được làm mịn thêm ở các mức thấp hơn. Việc làm mịn cho các DFD tiếp tục cho đến khi hình tròn thực hiện một chức năng đơn giản, tức là, cho tới khi tiến trình được biểu diễn bởi một vòng tròn thực hiện một chức năng dễ dàng, được cài đặt như một thành phần của chương trình.

Trong phân tích yêu cầu, người kỹ sư phần mềm có thể phát hiện một số khía cạnh nào đó của hệ thống sẽ thay đổi hay sẽ được nâng cao trong tương lai hoặc chưa được xác định rõ. Một cách khác, người phân tích có thể làm việc trên phần mềm hiện có để tìm ra sự thay đổi. Trong cả hai trường hợp, biểu đồ luồng dữ liệu cho phép dễ dàng cô lập lĩnh vực thay đổi. Bằng việc hiểu rõ luồng thông tin đi qua biên giới lĩnh vực thay đổi, người ta có thể có chuẩn bị tốt cho việc thay đổi trong tương lai, hay có thể tiến hành thay đổi hiện tại mà không làm đảo lộn các phần tử khác của hệ thống.

5.2.1.2. Tạo ra mô hình luồng điều khiển

Đối với nhiều kiểu ứng dụng xử lý dữ liệu, biểu đồ luồng dữ liệu là tất cả những điều cần thiết để có được cái nhìn ý nghĩa đối với các yêu cầu phần mềm. Tuy nhiên, như đã lưu ý, tồn tại một lớp ứng dụng rộng rãi hơn, được “điều khiển” bởi các sự kiện thay vì dữ liệu, tạo ra thông tin điều khiển chứ không chỉ là báo cáo hay hiển thị, xử lý thông tin với mối quan tâm chủ yếu về thời gian và hiệu năng. Những ứng dụng như vậy đòi hỏi mô hình luồng điều khiển CFD bên cạnh biểu đồ luồng dữ liệu.

Để xét duyệt cách tiếp cận CFD, biểu đồ luồng dữ liệu được loại bỏ mọi mũi tên luồng dữ liệu. Các sự kiện và các khoản mục điều khiển được bổ sung vào biểu đồ và một cửa sổ được vẽ trong đặc tả điều khiển.

Để chọn các sự kiện ta có hướng dẫn sau:

- Liệt kê tất cả các cảm biến mà phần mềm đọc.
- Liệt kê tất cả các điều kiện ngắt.
- Liệt kê tất cả các “chuyển mạch” mà thao tác viên dùng.
- Liệt kê tất cả các điều kiện dữ liệu.

- Gọi lại việc phân tích danh từ – động từ áp dụng cho lời thuật xử lý, xét lại tất cả các “khoản mục điều khiển” như đầu vào/đầu ra.
- Mô tả hành vi của hệ thống bằng cách xác định trạng thái của nó, xác định cách đạt đến từng trạng thái và định nghĩa phép chuyển giữa các trạng thái.
- Tập trung vào những bỏ sót có thể – một lỗi rất thông thường trong xác định các điều khiển như hỏi “Liệu còn cách nào khác để tới hay ra khỏi trạng thái này không?”.

5.2.1.3. Đặc tả điều khiển (Control specification)

Đặc tả điều khiển (CSPEC) biểu diễn cho hành vi của hệ thống theo hai cách khác nhau. CSPEC chứa một biểu đồ chuyển trạng thái (STD) là đặc tả tuần tự cho hành vi. Nó cũng có thể chứa một bảng kích hoạt chương trình (PAT) – một đặc tả tổ hợp cho hành vi.

Bằng cách nghiên cứu STD, người kỹ sư phần mềm có thể xác định hành vi của hệ thống và điều quan trọng hơn là xác định được liệu có lỗi hồng nào trong hành vi được đặc tả không.

Một kiểu biểu diễn hành vi khác là bảng kích hoạt tiến trình PAT. PAT biểu thị cho thông tin chứa trong hoàn cảnh của tiến trình, không phải là trạng thái. Tức là bảng chỉ ra tiến trình nào (hình tròn nào) trong mô hình luồng sẽ được gọi đến khi sự kiện xuất hiện. PAT được dùng như bảng hướng dẫn cho người thiết kế, người phải xây dựng cách điều hành kiểm soát các tiến trình được biểu diễn ở mức này.

CSPEC mô tả hành vi của hệ thống, nhưng nó không cung cấp bất kỳ thông tin nào về cách làm việc của các tiến trình được xem như kết quả của hành vi này.

5.2.1.4. Đặc tả tiến trình (Process specification)

Đặc tả tiến trình (PSPEC) được dùng để mô tả cho các tiến trình mô hình luồng xuất hiện ở mức cuối của việc làm mịn. Nội dung của đặc tả tiến trình bao gồm đoạn văn tường thuật, ngôn ngữ thiết kế chương trình (PDL) mô tả thuật toán xử lý, phương trình toán học, bảng, biểu đồ hay lược đồ. Bằng cách đưa PSPEC đi kèm với từng hình tròn trong mô hình luồng, người kỹ sư phần mềm tạo ra một “mini đặc tả” được dùng như bước đầu

tiên trong việc tạo ra bản đặc tả các yêu cầu phần mềm và dùng như một hướng dẫn cho việc thiết kế thành phần chương trình sẽ cài đặt cho tiến trình.

5.3. Phân tích hướng đối tượng

Khi xây dựng một sản phẩm hay một hệ thống mới, chúng ta mô tả nó như thế nào để thích hợp với công nghệ phần mềm hướng đối tượng? Các đối tượng phù hợp là gì? Chúng liên hệ với nhau như thế nào? Các đối tượng ứng xử như thế nào trong ngữ cảnh của hệ thống? Chúng ta mô tả hoặc mô hình các vấn đề như thế nào để tạo ra được một thiết kế có hiệu quả?

Mỗi câu hỏi trên đều có thể trả lời trong ngữ cảnh phân tích hướng đối tượng (OOA). Để xây dựng một mô hình phân tích, 5 nguyên tắc sau được áp dụng:

1. Miền thông tin được mô hình.
2. Chức năng module được mô tả.
3. Mô hình hành vi được mô tả.
4. Các mô hình được chia nhỏ để chi tiết hơn.
5. Các mô hình ban đầu mô tả bản chất của vấn đề, các mô hình về sau cung cấp chi tiết thực hiện.

Mục đích của OOA là định nghĩa các lớp (và các mối quan hệ, các hành vi liên kết với chúng) phù hợp với vấn đề cần giải quyết. Để thực hiện điều này, phải thực hiện các công việc sau:

1. Các yêu cầu cơ bản của người sử dụng phải được liên lạc giữa khách hàng và kỹ sư phần mềm.
2. Các lớp phải được xác định (các phương thức và các thuộc tính phải được định nghĩa).
3. Một cây lớp phải được mô tả.
4. Mối quan hệ đối tượng tới đối tượng phải được mô tả.
5. Hành vi các đối tượng phải được mô hình hoá.
6. Các công việc từ 1 đến 5 được lặp đi lặp lại cho đến khi mô hình được hoàn tất.

Mục đích của phân tích hướng đối tượng là phát triển một tập các mô hình mô tả hệ thống phần mềm máy tính như nó hoạt động để thoả mãn các

yêu cầu người dùng đã được xác định. OOA giống như các phương pháp phân tích thông thường khác xây dựng một mô hình phân tích nhiều phần để đạt mục tiêu trên.

5.3.1. Cách tiếp cận thông thường và cách tiếp cận hướng đối tượng

Phân tích có cấu trúc đã đưa ra một khung nhìn đầu vào – xử lý – đầu ra của yêu cầu. Dữ liệu được xem xét riêng rẽ từ các xử lý chuyển dữ liệu. Hành vi của hệ thống, mặc dù rất quan trọng chỉ chiếm vị trí thứ yếu trong phân tích có cấu trúc.

Fichman và Kemerer đã gợi ý 11 “hướng mô hình” được sử dụng để so sánh phương pháp thiết kế thông thường và phương pháp thiết kế hướng đối tượng:

1. Xác định/phân loại các thực thể.
2. Mô tả chung và toàn bộ mối quan hệ thực thể.
3. Các mối quan hệ thực thể khác.
4. Mô tả thuộc tính các thực thể.
5. Phân chia mô hình phạm vi rộng.
6. Trạng thái và chuyển giữa các trạng thái.
7. Mô tả chi tiết các hàm.
8. Phân tích trên xuống.
9. Dây xử lý end – to – end.
10. Xác định các dịch vụ thực hiện riêng.
11. Giao tiếp thực thể (bằng thông điệp hay bằng sự kiện).

5.3.2. Các phương pháp phân tích hướng đối tượng

5.3.2.1. Phương pháp Booch

Phương pháp Booch bao gồm hai quy trình phát triển vi mô và vĩ mô. Mức vi mô xác định tập các nhiệm vụ phân tích được áp dụng lại cho các bước trong quy trình vĩ mô. Do đó, cách tiếp cận tiến hoá được duy trì. Một phác thảo của quy trình phát triển vi mô do Booch đề nghị như sau:

- Xác định các lớp và đối tượng.

- Đưa ra các lớp ứng viên.
- Quản lý phân tích hành vi.
- Xác định kịch bản thích hợp.
- Xác định các thuộc tính và phương thức cho mỗi lớp.
- Xác định ngữ nghĩa của các lớp và đối tượng
 - Chọn kịch bản và phân tích.
 - Phân công trách nhiệm cho các hành vi kế thừa của lớp và đối tượng.
 - Chọn đối tượng và liệt kê vai trò và nhiệm vụ của nó.
 - Xác định phương thức thoả mãn nhiệm vụ.
 - Tìm kiếm sự cộng tác giữa các đối tượng.
- Xác định mối quan hệ giữa các lớp và đối tượng.
 - Xác định tính phụ thuộc tồn tại giữa các lớp.
 - Mô tả vai trò của mỗi đối tượng cụ thể.
 - Kiểm tra bằng các kịch bản.
- Quản lý các hoạt động điều chỉnh thích hợp.
 - Tạo ra biểu đồ thích hợp cho công việc được quản lý bên trên.
 - Xác định phân cấp lớp khi thích hợp.
 - Thực hiện nhóm dựa trên sự tương đồng lớp.
- Thực thi các lớp và đối tượng (trong ngữ cảnh OOA, nó mang ý nghĩa hoàn thành mô hình phân tích).

5.3.2.2. Phương pháp Coad và Yourdon

Một phác thảo của quy trình phát triển do Coad và Yourdon đề nghị như sau:

- Xác định đối tượng sử dụng tiêu chuẩn “tìm kiếm cái gì”.
- Xác định cấu trúc tổng quát – đặc tả.
- Xác định cấu trúc toàn thể – bộ phận.
- Xác định chủ đề (mô tả các thành phần hệ thống con).

- Xác định các thuộc tính.
- Xác định các dịch vụ.

5.3.2.3. Phương pháp Jacobson

Phương pháp này nhấn mạnh vào các trường hợp sử dụng – mô tả cách người dùng tương tác với sản phẩm hay hệ thống. Một phác thảo của quy trình phát triển do Jacobson đề nghị như sau:

- Xác định người sử dụng hệ thống và toàn bộ nhiệm vụ của họ.
- Xây dựng mô hình yêu cầu.
 - Xác định các tác nhân và trách nhiệm.
 - Xác định trường hợp sử dụng cho mỗi tác nhân.
 - Chuẩn bị khung nhìn ban đầu của đối tượng hệ thống và các quan hệ.
 - Xem xét lại mô hình sử dụng trường hợp sử dụng như một kịch bản để kiểm tra.
- Xây dựng mô hình phân tích
 - Xác định đối tượng giao diện sử dụng thông tin tác nhân – tương tác.
 - Tạo khung nhìn cấu trúc của đối tượng giao diện.
 - Mô tả hành vi đối tượng.
 - Cô lập hệ thống con và mô hình cho chúng.
 - Xem lại mô hình sử dụng trường hợp sử dụng như một kịch bản để kiểm tra.

5.3.2.4. Phương pháp Rumbaugh

Kỹ thuật mô hình đối tượng (OMT) cho phân tích, thiết kế hệ thống và thiết kế mức đối tượng. Các hoạt động phân tích tạo ra 3 mô hình: mô hình đối tượng (mô tả các đối tượng, các lớp, cây và mối quan hệ), mô hình động (mô tả các hành vi của đối tượng và hệ thống) và mô hình chức năng (mô tả giống như DFD mức cao mô tả các luồng thông tin chuyển qua trong hệ thống). Một phác thảo của quy trình phát triển do Rumbaugh đề nghị như sau:

- Phát triển một bản kê phạm vi vấn đề.
- Xây dựng một mô hình đối tượng.
 - Xác định các lớp thích hợp cho vấn đề.
 - Định nghĩa các thuộc tính và liên kết.
 - Định nghĩa liên kết đối tượng.
 - Tổ chức lớp đối tượng sử dụng thừa kế.
- Phát triển mô hình động.
 - Chuẩn bị kịch bản.
 - Xác định sự kiện và phát triển vết các sự kiện cho mỗi kịch bản.
 - Xây dựng biểu đồ luồng sự kiện.
 - Phát triển biểu đồ trạng thái.
 - Xem lại hành vi cho tính nhất quán và sự hoàn chỉnh.
- Xây dựng một mô hình chức năng cho hệ thống.
 - Xác định đầu vào và đầu ra.
 - Sử dụng biểu đồ luồng dữ liệu để mô tả sự chuyển luồng.
 - Phát triển PSPEC cho mỗi chức năng.
 - Mô tả ràng buộc và các tiêu chuẩn tối ưu.

5.3.2.5. Phương pháp Wirfs – Brock

Phương pháp này không tạo ra sự khác biệt rõ ràng giữa các công việc phân tích và thiết kế. Thay vào đó, một quá trình tiếp diễn bắt đầu khi xác định đặc tả khách hàng và kết thúc khi thiết kế được đề xuất. Một phác thảo của quy trình phát triển do Wirfs – Brock đề nghị như sau:

- Ước lượng đặc tả khách hàng.
- Sử dụng phân tích ngữ pháp để rút ra các lớp ứng viên từ đặc tả.
- Nhóm các lớp để xác định các lớp Cha.
- Định nghĩa trách nhiệm cho mỗi lớp.
- Phân công trách nhiệm cho mỗi lớp.
- Xác định sự cộng tác giữa các lớp dựa trên trách nhiệm.

- Xây dựng mô tả cây của lớp để chỉ ra mối quan hệ thừa kế.
- Xây dựng một biểu đồ công tác cho hệ thống.

Mặc dù các bước tiến hành khác nhau với mỗi phương pháp OOA nhưng toàn bộ quy trình OOA có sự tương tự. Để thực hiện phân tích hướng đối tượng, một kỹ sư hệ thống phải thực hiện các bước sau:

- Thu nhận các yêu cầu của khách hàng cho hệ thống hướng đối tượng (Object Oriented – OO).
 - Xác định các kịch bản hay các trường hợp sử dụng.
 - Xây dựng mô hình yêu cầu.
- Chọn các lớp, các đối tượng sử dụng các yêu cầu cơ bản như là hướng dẫn.
- Xác định các thuộc tính và các phương thức cho mỗi đối tượng hệ thống.
- Xác định cấu trúc và cây cho mỗi lớp tổ chức.
- Xây dựng mô hình quan hệ đối tượng.
- Xây dựng mô hình hành vi đối tượng.
- Xem xét lại các phân tích OO.

5.3.3. Các thành phần chung cho mô hình phân tích OOA

Monarchi và Puhf xác định một tập các thành phần mô tả chung xuất hiện trong OOA. Các thành phần tĩnh là các cấu trúc tự nhiên và xác định các đặc tính được nắm giữ trong suốt vòng đời của ứng dụng. Các thành phần động tập trung vào điều khiển và có ý nghĩa với xử lý sự kiện và thời gian. Các thành phần sau được xác định:

- *Khung nhìn tĩnh của lớp ngữ nghĩa*: Yêu cầu được đánh giá và lớp được rút ra như một phần của mô hình phân tích. Các lớp này tồn tại trong suốt vòng đời của ứng dụng và được thừa kế dựa trên ngữ nghĩa của các yêu cầu khách hàng.
- *Khung nhìn tĩnh của các thuộc tính*: Các lớp đều phải được mô tả rõ ràng. Các thuộc tính liên kết với lớp cung cấp một mô tả của lớp, nó cũng chỉ ra các phương thức thích hợp với lớp.

- *Khung nhìn tĩnh của các mối quan hệ:* Các đối tượng được nối với nhau theo nhiều cách. Mô hình phân tích phải mô tả các quan hệ từ đó các phương thức được xác định.
- *Khung nhìn các hành vi:* Mỗi quan hệ xác định một tập các hành vi thích hợp với các kịch bản sử dụng của hệ thống. Các hành vi được thực hiện bằng cách xác định một dãy các phương thức.
- *Khung nhìn động các liên lạc:* Các đối tượng liên lạc với nhau dựa trên một loạt các sự kiện gây ra sự chuyển trạng thái của hệ thống.
- *Khung nhìn động điều khiển và thời gian:* Tính tự nhiên và thời gian của sự kiện gây ra sự chuyển trạng thái phải được mô tả.

5.3.4. Quy trình OOA

Quy trình bắt đầu với việc hiểu cách thức hệ thống mà con người sử dụng. Nếu hệ thống tương tác với con người bằng máy tính thì hệ thống sẽ bao gồm : điều khiển quá trình (hoặc bằng chương trình khác), phối hợp hệ thống và các ứng dụng điều khiển. Khi các kịch bản sử dụng được xác định, ta bắt đầu mô hình hoá hệ thống.

5.3.4.1. Trường hợp sử dụng

Dựa trên các yêu cầu người dùng, người kỹ sư phần mềm tạo ra một tập các kịch bản xác định một luồng sử dụng cho hệ thống. Kịch bản này thường được gọi là trường hợp sử dụng.

Để tạo ra các trường hợp sử dụng, người phân tích đầu tiên xác định các kiểu đối tượng (hay thiết bị) khác nhau sẽ sử dụng hệ thống hay sản phẩm. Các tác nhân mô tả vai trò mà con người hay thiết bị đảm nhận trong hệ thống.

Cần phải chú ý các tác nhân và người sử dụng không giống nhau. Người sử dụng có thể đóng các vai trò khác nhau khi sử dụng hệ thống, còn các tác nhân mô tả các lớp hay các thực thể ngoài đảm nhận một vai trò.

Khi đã xác định xong các tác nhân, ta phát triển các trường hợp sử dụng. Trường hợp sử dụng mô tả các tác nhân tương tác với hệ thống, Jacobson đề nghị trường hợp sử dụng nên trả lời các câu hỏi sau:

- Các tác nhân sẽ thực hiện các công việc hay các chức năng nào?

- Các tác nhân tạo ra, thay đổi các thông tin hệ thống nào?
- Các tác nhân có thông báo cho hệ thống các thay đổi của môi trường bên ngoài hay không?
- Tác nhân muốn thông tin gì từ hệ thống?
- Tác nhân có muốn được thông báo về những thay đổi không mong đợi không?

Mỗi trường hợp sử dụng cung cấp một kịch bản rõ ràng về tương tác giữa tác nhân và hệ thống cần được phát triển theo cách tương tự. Cần xem xét cẩn thận các trường hợp sử dụng, nếu có một thành phần mơ hồ, phải xem lại trường hợp sử dụng để phát hiện vấn đề.

5.3.4.2. Mô hình class – responsibility – collaborator (CRC)

Mô hình CRC (mô hình cộng tác) cung cấp một phương tiện đơn giản để xác định và tổ chức các lớp thích hợp với yêu cầu hệ thống hay sản phẩm. Responsibility là các thuộc tính và phương thức thích hợp với lớp. Cộng tác là các lớp được yêu cầu cung cấp một lớp với thông tin cần thiết để hoàn thành responsibility.

a) Lớp

Các đối tượng biểu diễn bản thân nó thông qua rất nhiều hình thức khác nhau: Thực thể ngoài, sự vật, biến cố, sự kiện, vai trò, các đơn vị tổ chức, vị trí hoặc cấu trúc. Một kỹ thuật để xác định chúng trong ngữ cảnh phần mềm là thực hiện phân tích ngữ pháp dựa trên các xử lý hệ thống. Mọi danh từ đều có thể là các đối tượng tiềm năng. Nếu một đối tượng thỏa mãn 6 đặc tính lựa chọn sau thì nó được xem xét nằm trong mô hình CRC: thông tin được giữ, dịch vụ cần thiết, đa thuộc tính, thuộc tính chung, phương thức chung và các yêu cầu bản chất.

Firesmith đã mở rộng sự phân loại các kiểu lớp bằng cách gợi ý ra các loại sau:

- *Lớp thiết bị*: Mô hình các thực thể ngoài như cảm biến, motor, bàn phím.
- *Lớp thuộc tính*: Mô tả một số thuộc tính quan trọng của môi trường.
- *Lớp tương tác*: Mô hình các tương tác xảy ra giữa các đối tượng.

Thêm vào đó, lớp và đối tượng có thể phân loại bằng tập các đặc tính sau:

- *Tính hiện thực* : Lớp mô tả một vật cụ thể (như bàn phím hay cảm biến) hay lớp mô tả một thông tin trừu tượng (như kết quả dự đoán).
- *Tính bao gồm*: Lớp là nguyên tố (nó không bao gồm các lớp khác) hay lớp là liên kết (nó bao gồm ít nhất một lớp khác).
- *Tính tuần tự*: Lớp là đồng thời hay tuần tự (được điều khiển bởi các tài nguyên ở bên ngoài).
- *Sự tồn tại*: Lớp là nhất thời (nó được tạo ra và loại bỏ trong khi thực hiện chương trình). lớp là tạm thời (nó được tạo ra trong khi thực hiện chương trình và được loại bỏ khi chương trình chấm dứt) hay lớp vĩnh viễn (được ghi trong cơ sở dữ liệu).
- *Tính toàn vẹn*: Lớp không được bảo vệ hay lớp được bảo vệ.

Khi sử dụng các loại lớp trên, “card chỉ mục” được tạo ra như một phần của mô hình CRC và được mở rộng để chứa các loại lớp và đặc tính của nó như bảng 5.1.

Bảng 5.1. Card chỉ mục mô hình CRC

Tên lớp:	
Kiểu lớp: (ví dụ thiết bị, thuộc tính, vai trò, sự kiện)	
Responsibilities (Trách nhiệm)	Collaborator (Cộng tác)
....	
.....	

b) Responsibility

Responsibility bao gồm thuộc tính và phương thức. Các thuộc tính mô tả các đặc tính của lớp, là các thông tin mà lớp phải có để hoàn thành mục tiêu của phần mềm được khách hàng mô tả. Thuộc tính được rút ra từ các bảng kê về phạm vi hoặc được nhận ra từ việc hiểu lớp. Phương thức được rút ra từ việc thực hiện các phân tích ngữ pháp dựa trên các xử lý của hệ thống. Động từ là các phương thức tiềm năng.

Wirfs - Brock đã đề ra 5 nguyên tắc để phân phối responsibility cho các lớp:

1. *Sự thông minh của hệ thống được phân phối công bằng.* Mỗi ứng dụng phân tích một mức độ thông minh nhất định, hệ thống biết gì và cần phải làm gì. Sự thông minh có thể phân chia trên các lớp theo nhiều cách. Lớp “dump” (có một số responsibility) được mô hình để phục vụ cho lớp “smart” (có nhiều responsibility). Cách tiếp cận này có các nhược điểm sau:

- Tập trung mọi sự thông minh trong một số lớp, rất khó thay đổi.
- Có xu hướng yêu cầu nhiều lớp và do đó cần nhiều nỗ lực để phát triển.

Do đó, sự thông minh của hệ thống phải được phân phối công bằng giữa các lớp trong ứng dụng. Khi mỗi lớp biết và thực hiện công việc, tính cố kết của hệ thống được cải thiện.

2. *Một responsibility phải bắt đầu từ tổng quan :* Các responsibilities chung (cả phương thức và thuộc tính) phải nằm bên trong cấu trúc cây (vì nó là chung, sẽ được áp dụng cho nhiều lớp con). Tính đa hình cũng được sử dụng để xác định các phương thức được áp dụng chung cho các lớp con nhưng được thực hiện khác nhau với mỗi lớp con.

3. *Thông tin và hành vi liên quan đến responsibility phải được đặt trong cùng một lớp :* Hướng tiếp cận này được gọi là đóng gói. Dữ liệu và các xử lý thao tác dữ liệu phải được đóng gói trong một đơn vị cố kết.

4. *Một thông tin cụ thể phải nằm trong một lớp đơn, không phân bố trên nhiều lớp :* Một lớp đơn chứa một responsibility để lưu trữ và thao tác một kiểu thông tin cụ thể. Không nên chia sẻ responsibility giữa nhiều lớp. Nếu thông tin bị phân tán, sẽ rất khó kiểm thử và bảo trì phần mềm.

5. *Chia sẻ các responsibility giữa các lớp liên quan khi thích hợp.*

c) Cộng tác

Lớp hoàn thành responsibility của nó theo 2 cách:

1. Một lớp có thể sử dụng các phương thức của lớp để thao tác trên các thuộc tính của nó, do đó hoàn thành responsibility cụ thể.

2. Một lớp có thể cộng tác với các lớp khác.

Sự cộng tác xác định mối liên hệ giữa các lớp. Khi một tập hợp các lớp cộng tác để đạt được yêu cầu, chúng được tổ chức trong hệ thống con.

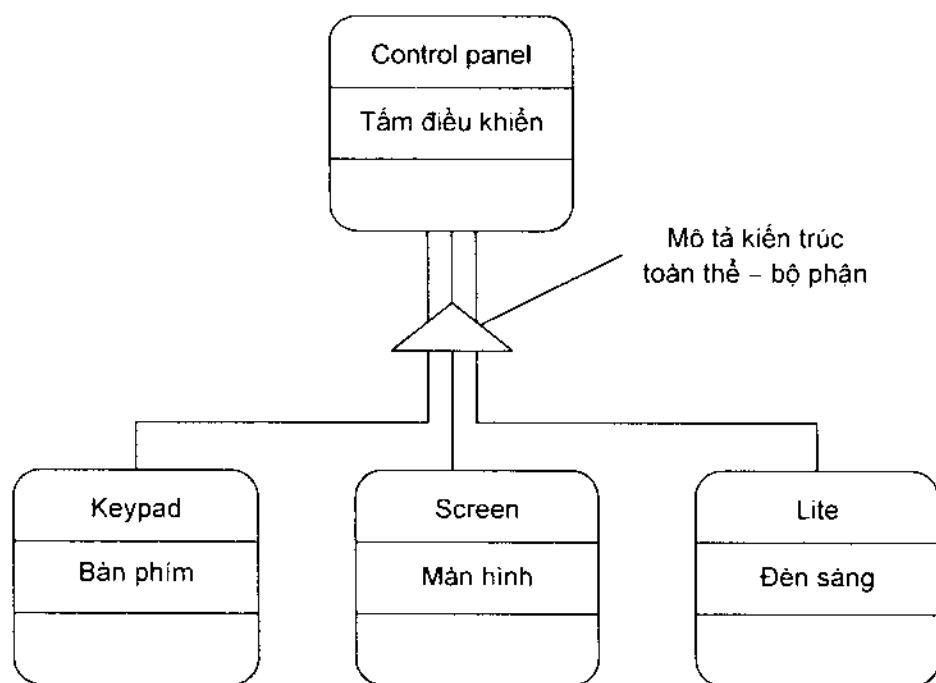
Sự cộng tác được xác định bằng cách quyết định khi nào lớp có thể tự hoàn thành responsibility, nếu không nó cần tương tác với các lớp khác.

Trong mọi trường hợp, tên của lớp cộng tác được ghi trong card chỉ mục mô hình CRC bên cạnh responsibility sinh ra sự cộng tác. Do đó, card chỉ mục chứa một danh sách các responsibility và các cộng tác tương ứng cho phép hoàn thành responsibility.

5.3.4.3. Xác định cấu trúc và cây

Khi lớp và đối tượng được xác định sử dụng mô hình CRC, người phân tích bắt đầu tập trung vào cấu trúc của mô hình lớp và cây tổng hợp. Coad và Yourdon gợi ý rằng cấu trúc tổng quát – đặc tả (generalization – specialization structure) có thể dùng để xác định các lớp.

Trong một số trường hợp, một đối tượng được mô tả trong một mô hình ban đầu bao gồm những thành phần mà bản thân nó được định nghĩa như một đối tượng. Các đối tượng liên kết được mô tả như cấu trúc tổng thể – bộ phận và được xác định sử dụng các ký pháp như trong hình 5.2.



Hình 5.2. Ký pháp mối quan hệ toàn thể – bộ phận

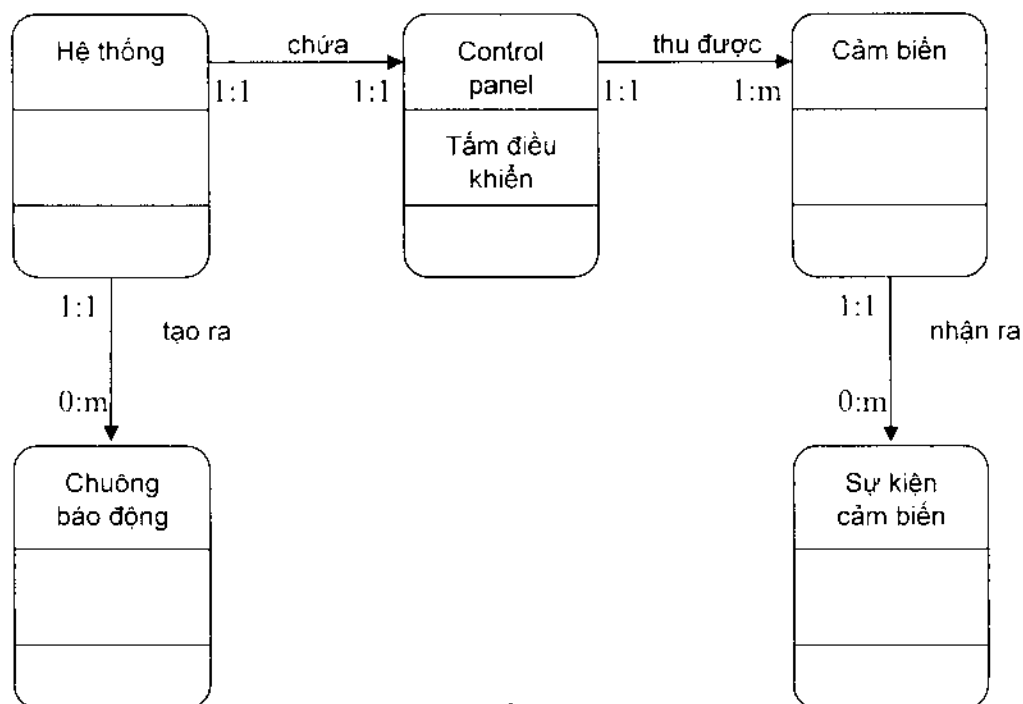
5.3.4.4. Xác định chủ đề và các lớp con

Một mô hình phân tích hệ thống phức tạp thường có hàng trăm lớp và hàng chục các cấu trúc. Do đó, cần phải mô tả ngắn gọn cho các cấu trúc.

Khi một tập con của các lớp cộng tác với nhau để hoàn thành các nhiệm vụ cố kết, chúng thường được gọi là chủ đề (subject) hoặc hệ thống con. Cả chủ đề và hệ thống con đều cung cấp một tham chiếu hay một con trỏ tới một chi tiết trong mô hình phân tích. Nhìn từ bên ngoài một chủ đề hay một hệ thống con được coi như một hộp đen chứa tập các nhiệm vụ và có cộng tác của riêng nó. Một hệ thống con thực hiện một hay nhiều hợp đồng với cộng tác bên ngoài. Một hợp đồng là một danh sách cụ thể các yêu cầu mà cộng tác có thể tạo của hệ thống.

Hệ thống con được mô tả trong ngữ cảnh mô hình hoá CRC bằng cách tạo ra card chỉ mục hệ thống con. Card chỉ mục hệ thống con chỉ ra tên của hệ thống con, các hợp đồng mà hệ thống con phải thực hiện và các lớp mà hệ thống con hỗ trợ hợp đồng.

5.3.5. Mô hình đối tượng – quan hệ



Hình 5.3. Mô hình đối tượng – quan hệ

Cách tiếp cận mô hình hoá CRC thiết lập các thành phần đầu tiên của quan hệ lớp và đối tượng. Bước đầu tiên trong xác định quan hệ là hiểu nhiệm vụ của mỗi lớp. Card chỉ mục mô hình CRC chứa một danh sách các nhiệm vụ. Bước tiếp theo là xác định các lớp cộng tác giúp cho việc đạt được các nhiệm vụ. Nó thiết lập "liên kết" giữa các lớp.

Một mối quan hệ tồn tại giữa hai đối tượng bất kỳ được liên kết. Do đó, cộng tác luôn được liên hệ theo cách nào đó. Kiểu quan hệ thông thường nhất là quan hệ nhị phân giữa hai lớp. Khi xem xét trong ngữ cảnh hệ thống OO, một quan hệ nhị phân xác định lớp nào đóng vai trò khách, lớp nào đóng vai trò chủ.

Rambaugh đã gợi ý các mối quan hệ được thừa kế bằng cách nghiên cứu các động từ và cụm động từ trong các bảng kê phạm vi hay trong các trường hợp sử dụng cho hệ thống. Sử dụng các phân tích ngữ pháp, nhà phân tích cô lập các động từ và chỉ ra các vị trí vật lý (như next to, part of, contained in), liên hệ (transmits to, acquires from), sự sở hữu (incorporated by, is composed of) và sự thoả mãn các điều kiện (manages, coordinates, controls). Chúng cung cấp một mô tả của quan hệ.

Mô hình đối tượng – quan hệ được thực hiện qua 3 bước:

1. Sử dụng card chỉ mục CRC, một mạng các đối tượng cộng tác được mô tả. Trước tiên các đối tượng được vẽ mà không có các đường có nhãn chỉ ra mối quan hệ tồn tại giữa các đối tượng.

2. Xem lại card chỉ mục CRC, nhiệm vụ và cộng tác được ước lượng và mỗi liên kết không có nhãn được đặt tên. Để tránh mơ hồ, một mũi tên chỉ ra hướng của quan hệ.

3. Khi mỗi quan hệ có tên được thiết lập, nó được ước lượng để quyết định kiểu liên kết: 0:1, 1:1, 0:n, 1:n.

Các bước trên được tiếp tục đến khi mô hình đối tượng – quan hệ được hoàn thành.

5.3.6. Mô hình đối tượng – hành vi

Mô hình CRC và mô hình đối tượng – quan hệ mô tả các thành phần tĩnh của một hình phân tích OO. Để tạo ra các hành vi động của hệ thống

OO, ta phải mô tả ứng xử của hệ thống như một hàm của các sự kiện và thời gian cụ thể.

Mô hình đối tượng -- hành vi chỉ ra cách thức một hệ thống OO đáp ứng các sự kiện bên ngoài. Để tạo ra mô hình này cần thực hiện các bước sau:

1. Định giá các trường hợp sử dụng để hiểu đầy đủ về dãy các tương tác trong hệ thống.
2. Xác định các sự kiện dẫn đến dãy tương tác và hiểu cách thức các sự kiện liên quan đến các đối tượng.
3. Tạo ra event trace cho mỗi trường hợp sử dụng.
4. Xây dựng một biểu đồ chuyển trạng thái cho hệ thống.
5. Xem lại mô hình đối tượng -- hành vi để kiểm tra tính chính xác và nhất quán.

5.3.6.1. Mô tả trạng thái

Trong ngữ cảnh hệ thống OO, có hai loại đặc tính khác nhau của trạng thái cần được xem xét:

- Trạng thái của mỗi lớp khi hệ thống thực hiện chức năng của nó.
- Trạng thái của hệ thống được quan sát từ bên ngoài khi hệ thống thực hiện chức năng của nó.

Trạng thái của đối tượng có hai đặc tính : Chủ động và bị động. Trạng thái bị động thường là trạng thái hiện tại của toàn bộ các thuộc tính đối tượng. Trạng thái chủ động xác định trạng thái hiện tại của một đối tượng khi nó nằm trong chuyển trạng thái tiếp diễn hoặc xử lý.

Khi một sự kiện được xác định cho một trường hợp sử dụng, người phân tích tạo ra một mô tả cách thức sự kiện gây ra luồng từ đối tượng này đến đối tượng khác. Ta gọi event trace là một mô tả phiên bản tóm tắt của trường hợp sử dụng. Nó mô tả các đối tượng và các sự kiện trọng yếu gây ra các hành vi đến các luồng từ đối tượng này sang đối tượng khác.

Khi một event trace hoàn chỉnh được phân tích, các sự kiện gây ra chuyển đổi giữa các đối tượng hệ thống được đối chiếu trong các sự kiện vào và sự kiện ra (từ đối tượng). Điều này có thể mô tả bằng cách sử dụng các biểu đồ luồng sự kiện.

Chương 6

THIẾT KẾ

6.1. Định nghĩa thiết kế

Thiết kế là bước đầu tiên trong giai đoạn phát triển bất cứ một sản phẩm hay hệ thống công nghệ nào. Nó được định nghĩa là “tiến trình áp dụng nhiều kỹ thuật và nguyên lý với mục đích xác định một thiết bị, một tiến trình hay một hệ thống đủ chi tiết để cho phép thực hiện nó về mặt vật lý”.

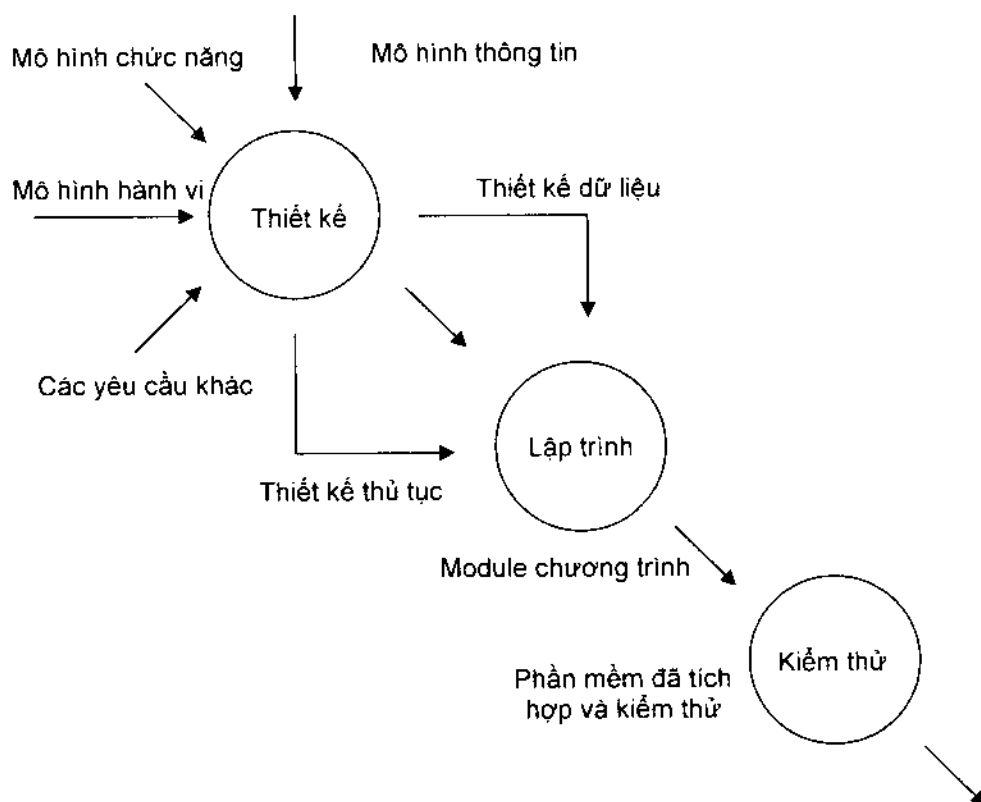
Mục tiêu của thiết kế là tạo ra một mô hình hay biểu diễn một thực thể sau này sẽ được áp dụng. Tiến trình phát triển mô hình này dựa trên kinh nghiệm trong việc xây dựng các thực thể tương tự, một tập các nguyên lý và/hoặc các trục cảm hướng dẫn cách tiến triển mô hình này, một tập các tiêu chuẩn để đánh giá chất lượng và một tiến trình lặp lại để sau đó dẫn tới biểu diễn thiết kế chung cuộc.

Thiết kế phần mềm là trung tâm kỹ thuật của tiến trình công nghệ phần mềm và được áp dụng cho khuôn cảnh phát triển phần mềm bất kỳ. Khi các yêu cầu phần mềm được phân tích và đặc tả thì thiết kế phần mềm là một trong 3 hoạt động – thiết kế, lập trình và kiểm thử – những hoạt động cần để xây dựng và kiểm chứng phần mềm.

Luồng thông tin trong giai đoạn kỹ thuật được minh họa trên hình 6.1.

Các yêu cầu phần mềm được biểu thị bởi mô hình thông tin, chức năng và hành vi, là đầu vào cho bước thiết kế. Sử dụng một trong các phương pháp thiết kế, tạo thiết kế dữ liệu, thiết kế kiến trúc, thiết kế thủ tục. Thiết kế dữ liệu chuyển mô hình lĩnh vực thông tin đã được tạo ra trong phân tích thành các cấu trúc dữ liệu cần thiết cho việc cài đặt phần mềm sau này. Thiết kế kiến trúc định nghĩa một mối quan hệ giữa các thành phần cấu trúc

chính của chương trình. Thiết kế thủ tục biến đổi các thành phần cấu trúc thành mô tả thủ tục phần mềm. Chương trình gốc sẽ được sinh ra, sau đó việc kiểm thử được thực hiện để tích hợp và làm hợp lệ phần mềm.



Hình 6.1. Luồng thông tin trong thiết kế

Thiết kế phần mềm là một tiến trình qua đó các yêu cầu được dịch thành một biểu diễn phần mềm. Ban đầu biểu diễn mô tả quan điểm của toàn bộ phần mềm. Việc làm mịn tiếp sau dẫn tới một biểu diễn thiết kế rất gần với chương trình gốc.

Theo quan điểm quản lý dự án, thiết kế phần mềm được tiến hành theo hai bước. Thiết kế sơ bộ quan tâm tới việc dịch các yêu cầu thành kiến trúc dữ liệu và phần mềm. Thiết kế chi tiết tập trung vào làm mịn biểu diễn kiến trúc để dẫn tới cấu trúc dữ liệu chi tiết và biểu diễn thuật toán cho phần mềm.

Trong phạm vi thiết kế sơ bộ và chi tiết, xuất hiện các hoạt động khác nhau. Bên cạnh việc thiết kế dữ liệu, kiến trúc và thủ tục, nhiều ứng dụng hiện đại còn có hoạt động thiết kế giao diện phân biệt. Thiết kế giao diện lập ra cách bố trí và cơ chế tương tác cho tương tác người – máy.

6.2. Thiết kế có cấu trúc

6.2.1. Các xem xét về tiến trình thiết kế

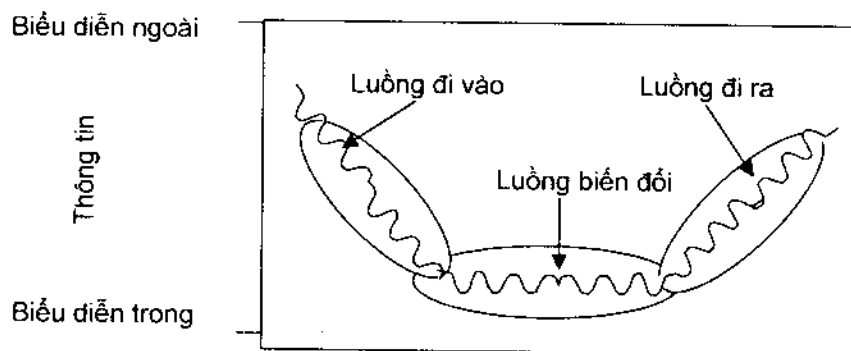
Thiết kế có cấu trúc cho phép biến đổi thuận tiện từ các biểu diễn thông tin như biểu đồ luồng dữ liệu có trong bản *Đặc tả yêu cầu phần mềm* sang mô tả về cấu trúc chương trình. Việc biến đổi từ luồng thông tin sang cấu trúc thông tin được thực hiện như một phần của tiến trình gồm 5 bước:

- Thiết lập kiểu luồng thông tin.
- Chỉ ra biên giới luồng.
- Ánh xạ DFD vào cấu trúc chương trình.
- Xác định các cấp bậc điều khiển theo cách tạo nhân tố.
- Cấu trúc kết quả được làm mịn bằng cách dùng các phương tiện thiết kế và trực cảm.

Kiểu luồng thông tin là bộ dẫn lối cho cách tiếp cận ánh xạ ở bước 3. Trong các nội dung sau đây chúng ta sẽ xem xét hai kiểu luồng.

6.2.1.1. Luồng biến đổi

Những dữ liệu bên ngoài đều được chuyển thành dạng bên trong để xử lý.

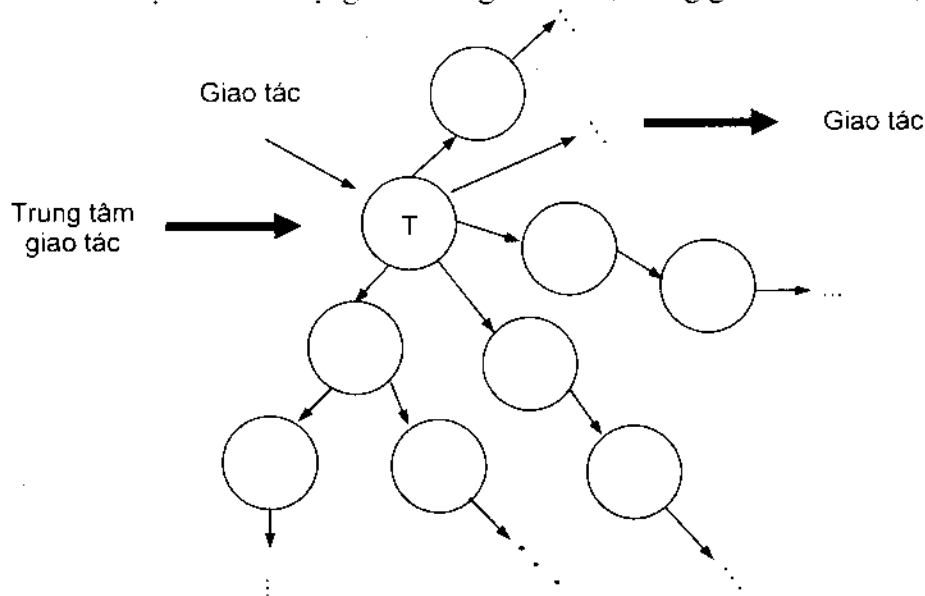


Hình 6.2. Luồng thông tin

Thông tin đi vào hệ thống theo con đường biến đổi dữ liệu ngoài thành dạng bên trong và được coi như luồng đi vào. Trong hạt nhân của phần mềm, việc chuyển đổi xuất hiện. Dữ liệu tới truyền qua một trung tâm biến đổi và bắt đầu chuyển dọc theo những con đường dẫn tới đầu ra của phần mềm. Dữ liệu chuyển dọc theo những con đường này gọi là luồng đi ra. Toàn bộ luồng dữ liệu xuất hiện một cách tuần tự đi theo một hay một vài đường trực tiếp. Khi một đoạn của biểu đồ luồng dữ liệu có những đặc trưng này thì đó là luồng biến đổi.

6.2.1.2. Luồng giao tác

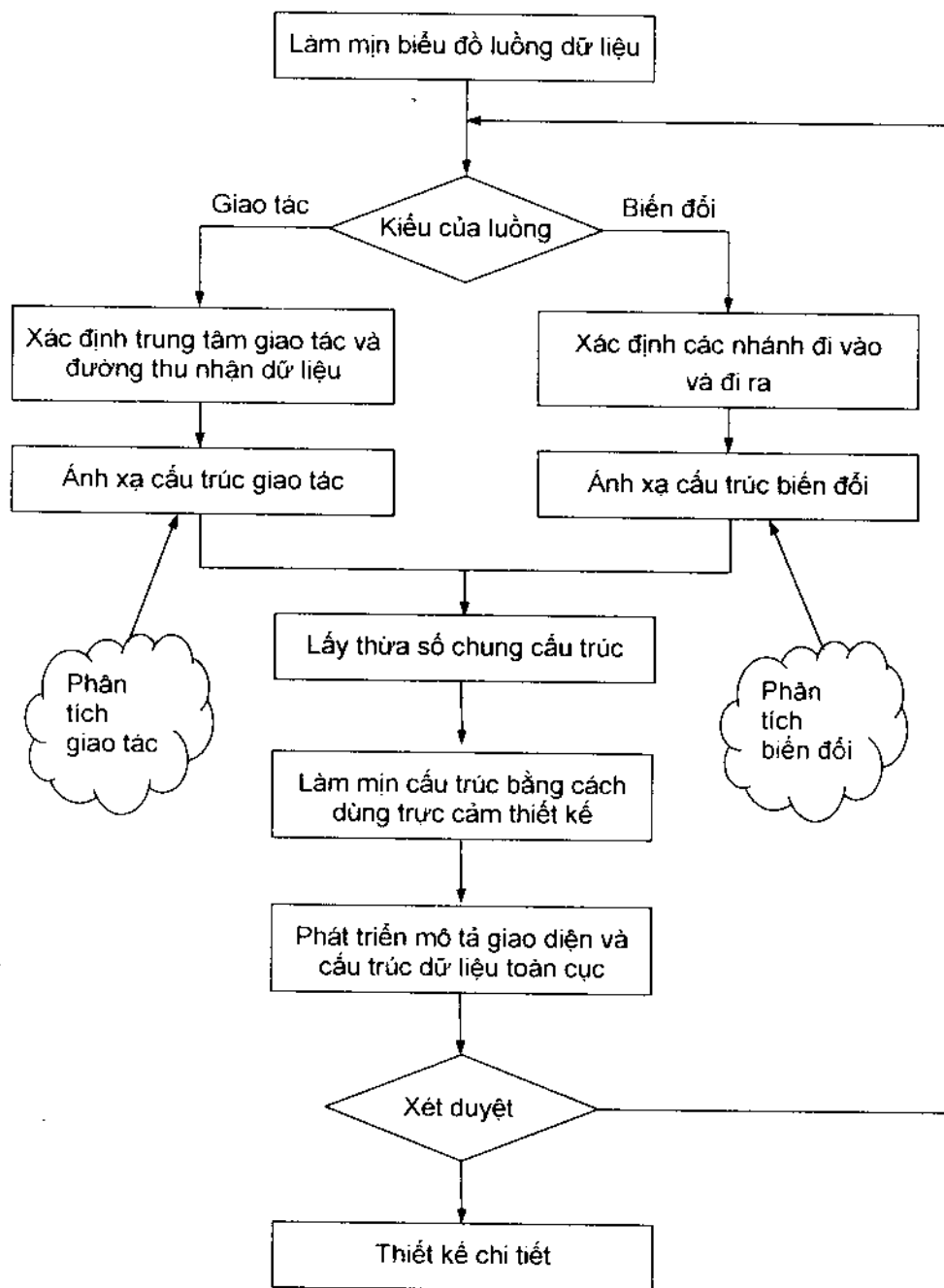
Luồng thông tin được đặc trưng bởi một khoản mục dữ liệu riêng biệt, gọi là giao tác, gây ra luồng dữ liệu khác đi theo một trong nhiều đường dẫn. Khi một DFD có dạng như trong hình 6.3, luồng giao tác xuất hiện.



Hình 6.3. Luồng giao tác

Luồng giao tác được đặc trưng bởi dữ liệu chuyển theo đường đi vào (còn gọi là đường nhận) đổi thông tin thế giới ngoài thành một giao tác. Giao tác được tính toán và dựa trên giá trị của nó, luồng thông tin mà từ đó nhiều đường hành động đi ra được gọi là trung tâm giao tác.

Cách tiếp cận tổng thể tới thiết kế có cấu trúc được minh họa trong hình 6.4.



Hình 6.4.Thiết kế hướng luồng dữ liệu

6.2.2. Phân tích biến đổi

Phân tích biến đổi là một tập các bước thiết kế cho phép một DFD với các đặc trưng luồng biến đổi được ánh xạ vào một tiêu bản có sẵn trong chương trình.

Các bước thiết kế bao gồm:

Bước 1: Xét duyệt lại mô hình hệ thống nền tảng. Mô hình hệ thống nền tảng bao gồm DFD mức 0 và thông tin hỗ trợ. Trong thực tế, bước thiết kế bắt đầu với một đánh giá cả bản đặc tả hệ thống và đặc tả yêu cầu phần mềm. Cả hai tài liệu này đều mô tả luồng và cấu trúc thông tin tại giao diện phần mềm.

Bước 2: Xét duyệt và làm mịn các biểu đồ luồng dữ liệu cho phần mềm. Thông tin thu được từ các mô hình phân tích có trong bản đặc tả yêu cầu phần mềm sẽ được làm mịn để tạo chi tiết hơn.

Bước 3: Xác định xem liệu DFD có các đặc trưng biến đổi hay giao tác. Nói chung, luồng thông tin bên trong một hệ thống bao giờ cũng có thể biểu diễn như một phép biến đổi. Trong bước này, người thiết kế nên chọn ra một luồng toàn cục dựa trên bản chất thông dụng của DFD đó. Bên cạnh đó, các miền cục bộ của luồng biến đổi hay giao tác sẽ được cô lập ra. Các luồng con này được dùng để làm mịn cho cấu trúc chương trình suy ra từ đặc trưng toàn cục đã mô tả ở trên.

Bước 4: Cô lập trung tâm biến đổi bằng cách xác định các biên giới đường vào và đường ra.

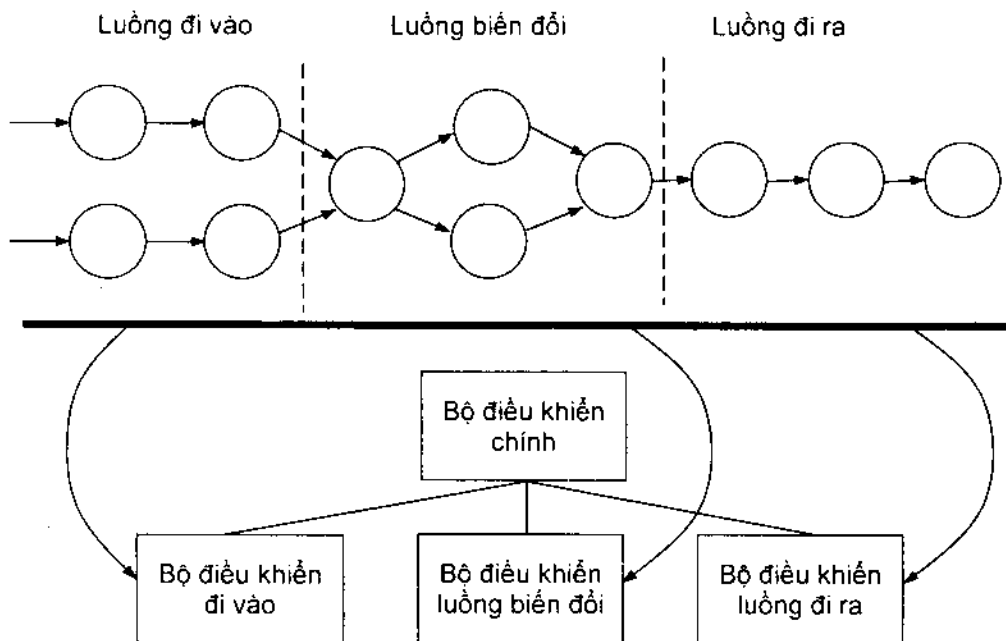
Bước 5: Thực hiện “lấy thừa số chung bậc nhất”. Cấu trúc chương trình biểu diễn một phân bố điều khiển từ trên xuống. Việc lấy thừa số chung này sinh trong một cấu trúc chương trình, trong đó các module mức đỉnh thực hiện việc ra quyết định còn các module mức thấp thì thực hiện hầu hết các công tác vào, tính toán, và đưa ra. Các module mức trung gian thực hiện một số điều khiển và tiến hành một khối lượng công việc vừa phải.

Khi gặp phải luồng biến đổi, một DFD được ánh xạ vào một cấu trúc xác định việc điều khiển xử lý thông tin đi vào, biến đổi và đi ra. Việc lấy thừa số chung bậc một được minh họa như trong hình 6.5.

Bộ điều khiển chính nằm tại đỉnh của cấu trúc chương trình và phục vụ việc điều phối các chức năng điều khiển thuộc cấp sau:

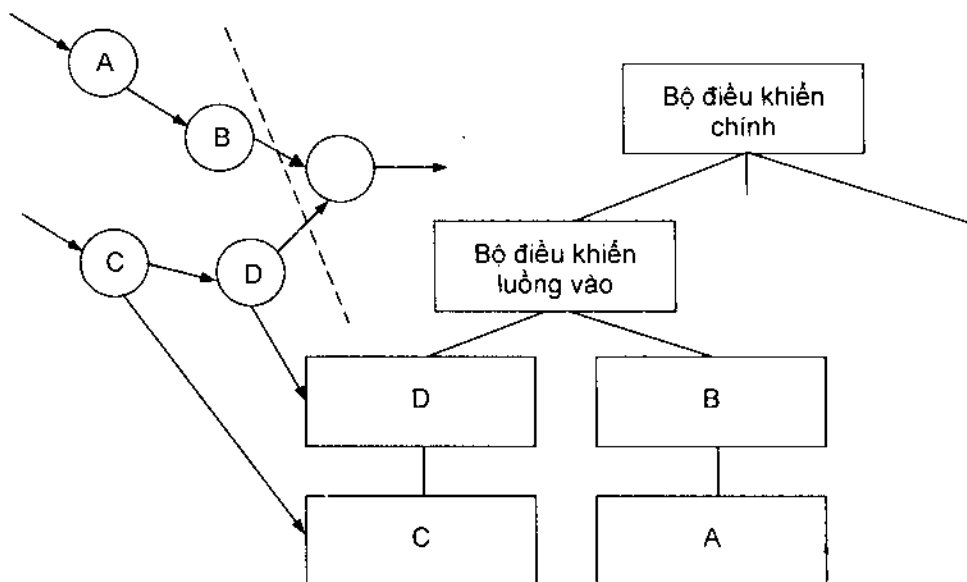
- Bộ điều khiển xử lý thông tin đi vào, phối hợp việc nhận các dữ liệu tới.
- Bộ điều khiển luồng biến đổi, giám sát mọi thao tác trên dữ liệu dưới dạng trung gian (như một module gọi đến nhiều thủ tục biến đổi).
- Bộ điều khiển xử lý thông tin đi ra, phối hợp việc tạo thông tin đi ra.

Cấu trúc 3 mũi tên này được biểu diễn như trong hình 6.5, luồng phức tạp trong hệ thống lớn không chế hai hay nhiều module điều khiển cho từng chức năng điều khiển tổng quát được mô tả ở trên. Số các module tại mức thứ nhất nên bị giới hạn vào số tối thiểu có thể thực hiện các chức năng điều khiển và vẫn duy trì các đặc trưng tương tác và có kết.



Hình 6.5. Lấy thừa số chung bậc nhất

Bước 6: Thực hiện “lấy thừa số chung bậc hai”. Lấy thừa số chung bậc hai được thực hiện bằng cách ánh xạ các phép biến đổi riêng của DFD vào các module thích hợp bên trong cấu trúc chương trình. Bắt đầu tại biên giới trung tâm biến đổi rồi chuyển ra ngoài theo đường đi vào sau đó theo đường đi ra, các phép biến đổi được ánh xạ vào các mức của các cấp thuộc cấu trúc chương trình.



Hình 6.6. Lấy thừa số bậc 2

Bước 7: Làm mịn cấu trúc chương trình “lát cắt thứ nhất” bằng cách dùng trực cảm thiết kế để cải thiện chất lượng phần mềm. Cấu trúc chương trình lát cắt đầu tiên được làm mịn bằng cách áp dụng các khái niệm độc lập module. Các module được hợp triển để rút ra thừa số nhảy cảm, cố kết, cố kết tốt, móc nối tối thiểu, và điều quan trọng nhất, một cấu trúc sẽ được cài đặt đơn giản, được kiểm thử không lẫn lộn và bảo trì không tốn kém.

Việc làm mịn cần được xem xét theo cách thực tế và thông thường. Ví dụ, có khi không cần bộ điều khiển luồng dữ liệu đi vào, hoặc khi một số xử lý đầu vào lại cần tới trong một module vốn là thuộc cấp của bộ điều khiển biến đổi, có khi bắt buộc phải nối với dữ liệu toàn cục hay khi các đặc trưng cấu trúc tối ưu không đạt được.

Mục tiêu của 7 bước nói trên để phát triển một biểu diễn toàn cục cho phần mềm. Tức là một khi cấu trúc được xác định thì chúng ta có thể đánh giá và làm mịn kiến trúc phần mềm bằng cách coi nó như một tổng thể. Những sửa đổi được tiến hành vào lúc này đòi hỏi khối lượng công việc không nhiều nhưng sẽ tác động sâu sắc đến chất lượng và việc bảo trì phần mềm.

6.2.3. Phân tích giao tác

Các bước thiết kế cho phân tích giao tác tương tự nhau và trong một số trường hợp đồng nhất với phân tích biến đổi. Một khác biệt chính nằm trong ánh xạ DFD vào cấu trúc chương trình.

Bước 1: Xét duyệt mô hình hệ thống nền tảng.

Bước 2: Xét duyệt và làm mịn biểu đồ luồng dữ liệu cho phần mềm.

Bước 3: Xác định liệu xem DFD có các đặc trưng luồng biến đổi hay giao tác không.

Bước 4: Xác định trung tâm giao tác và các đặc trưng luồng theo từng hành động.

Bước 5: Ánh xạ DFD vào cấu trúc chương trình tuân theo xử lý giao tác. Luồng giao tác được ánh xạ vào cấu trúc chương trình có một nhánh đi vào gọi là nhánh gửi. Cấu trúc nhánh đi vào được phát triển theo cùng cách như phân tích biến đổi. Bắt đầu từ trung tâm giao tác, các hình tròn trên đường đi vào (đường nhận) được ánh xạ vào các module. Cấu trúc của nhánh gửi có chứa một module gửi kiểm soát toàn bộ bộ điều khiển đường hành động thuộc cấp. Mỗi đường hành động của DFD đều được ánh xạ vào một cấu trúc tương ứng với các đặc trưng luồng riêng.

Bước 6: Lấy thừa số và làm mịn cấu trúc giao tác và cấu trúc từng đường hành động.

Bước 7: Làm mịn cấu trúc chương trình “lắt cắt thứ nhất” bằng việc dùng trực cảm thiết kế để có chất lượng phần mềm được nâng cao. Bước này tương tự như trong phân tích biến đổi.

6.3. Thiết kế hướng đối tượng

6.3.1. Thiết kế cho hệ thống hướng đối tượng

Với các hệ thống hướng đối tượng chúng ta có thể xác định ra 4 tầng thiết kế như sau:

Tầng hệ thống con: Chứa mô tả của mỗi hệ thống con cho phép hệ thống đạt được các yêu cầu của khách hàng và thực hiện các hạ tầng kỹ thuật hỗ trợ yêu cầu khách hàng.

Tầng lớp và đối tượng: Chứa phân lớp cho phép hệ thống được tạo ra bằng cách sử dụng sự tổng quát hoá và tính tăng dần đặc tả đích. Tầng này chứa các thiết kế mô tả các đối tượng.

Tầng thông điệp: Chứa các chi tiết cho phép các đối tượng giao tiếp. Tầng này thiết lập các giao diện bên trong và bên ngoài cho hệ thống.

Tầng trách nhiệm – responsibility: Chứa các cấu trúc dữ liệu và các thuật toán thiết kế cho mọi thuộc tính và phương thức của lớp.

6.3.2. Cách tiếp cận thông thường và hướng đối tượng

Fichman và Kemerer đề nghị 10 thành phần mô hình hoá thiết kế sau được dùng để so sánh cách tiếp cận thiết kế thông thường và cách tiếp cận hướng đối tượng:

1. Mô tả cây các module.
2. Đặc tả định nghĩa dữ liệu.
3. Đặc tả logic thủ tục.
4. Chỉ dẫn về trình tự xử lý end – to – end.
5. Mô tả các trạng thái của đối tượng và các giao dịch.
6. Xác định các lớp và cây.
7. Xác định các phương thức cho lớp.
8. Mô tả chi tiết các phương thức.
9. Mô tả mối liên hệ thông điệp.
10. Xác định các dịch vụ riêng biệt.

6.3.3. Vấn đề thiết kế

Bertrand Meyer đã đưa ra 5 tiêu chuẩn cho phương thức thiết kế để đạt được tính module và quan hệ tới thiết kế hướng đối tượng:

- *Tính phân tích được:* Khả năng trong đó phương pháp thiết kế giúp người thiết kế phân tích một vấn đề lớn thành các vấn đề nhỏ để giải quyết.
- *Tính tương thích :* Mức độ để một phương pháp thiết kế bảo đảm rằng các thành phần chương trình (module) khi được thiết kế và xây dựng có thể được sử dụng lại để tạo ra hệ thống khác.

- *Tính hiệu dụng* : Dễ dàng hiệu các thành phần chương trình mà không cần tham chiếu đến các thông tin hoặc các module khác.
- *Tính tiếp tục*: Khả năng tạo ra thay đổi nhỏ trong chương trình và có những thay đổi biểu thị bản thân nó tương ứng sự thay đổi trong một hoặc vài module.
- *Tính bảo vệ*: Một đặc tính kiến trúc làm giảm sự phổ biến của ảnh hưởng bên nếu một lỗi xảy ra trong module.

6.3.4. Các phương pháp thiết kế

Mỗi phương pháp phân tích đều tương ứng với một phương pháp thiết kế bao gồm một tập các mô tả thiết kế và các kỹ pháp cho phép các kỹ sư phần mềm tạo ra các mô hình thiết kế nhất quán.

6.3.4.1. Phương pháp Booch

Phương pháp Booch bao gồm cả hai quy trình phát triển vi mô và vĩ mô. Mức vi mô xác định tập các nhiệm vụ thiết kế được áp dụng lại cho các bước trong quy trình vĩ mô. Do đó, cách tiếp cận tiến hoá được duy trì. Một phác thảo của quy trình phát triển vi mô do Booch đề nghị như sau:

- Lập kế hoạch kiến trúc
 - Nhóm các đối tượng tương tự nhau vào các phần kiến trúc riêng rẽ.
 - Phân lớp đối tượng theo mức trừu tượng.
 - Xác định các kịch bản thích hợp.
 - Tạo ra các nguyên mẫu thiết kế.
 - Hợp lệ các nguyên mẫu thiết kế bằng cách áp dụng cho các kịch bản.
- Chiến lược thiết kế
 - Xác định chính sách phụ thuộc lĩnh vực (ví dụ như các luật quản lý việc sử dụng các phương thức và các thuộc tính).
 - Xác định chính sách mô tả lĩnh vực cho quản lý bộ nhớ, quản lý lỗi và các chức năng hạ tầng khác.
 - Phát triển kịch bản mô tả ngữ nghĩa của chính sách.
 - Tạo ra các nguyên mẫu cho mỗi chính sách.

- Phối hợp và tinh chế các nguyên mẫu.
- Xem xét lại các chính sách để đảm bảo rằng nó bao phủ hết các khung nhìn kiến trúc.
- **Lập kế hoạch phát hành**
 - Tổ chức các kịch bản được phát triển trong phân tích theo độ ưu tiên.
 - Cấp phát các kiến trúc khả thi cho các kịch bản.
 - Thiết kế và xây dựng mỗi kiến trúc khả thi tăng dần.
 - Điều chỉnh mục tiêu và lịch trình của các khả thi theo yêu cầu.

6.3.4.2. Phương pháp Coad và Yourdon

Phương pháp này được phát triển bằng cách nghiên cứu cách thức thiết kế hướng đối tượng hiệu quả. Cách tiếp cận thiết kế này không chỉ cho ứng dụng mà cho cả các hạ tầng của ứng dụng. Một phác thảo của quy trình phát triển do Coad và Yourdon đề nghị như sau:

- **Thành phần miền vấn đề.**
 - Nhóm các lớp mô tả miền.
 - Thiết kế một cấu trúc cây lớp phù hợp cho lớp ứng dụng.
 - Đơn giản thừa kế.
 - Tinh chế thiết kế để cải thiện hiệu năng.
 - Phát triển giao diện với các thành phần quản lý dữ liệu.
 - Xem xét lại thiết kế.
- **Thành phần tương tác với con người.**
 - Xác định các tác nhân con người.
 - Xác định các kịch bản công việc.
 - Thiết kế cây lệnh người dùng.
 - Tinh chế dãy các giao diện người dùng.
 - Tích hợp các lớp GUI khi thích hợp.
- **Thành phần quản lý công việc.**
 - Xác định kiểu công việc.
 - Thiết lập mức ưu tiên.

- Xác định công việc để phục vụ như người điều phối các công việc khác.
- Thiết kế đối tượng phù hợp cho mỗi công việc.
- Thành phần quản lý dữ liệu.
 - Thiết kế cấu trúc dữ liệu và layout.
 - Thiết kế các dịch vụ yêu cầu để quản lý cấu trúc dữ liệu.
 - Xác định các công cụ có thể trợ giúp trong thực hiện quản lý dữ liệu.
 - Xác định các lớp thích hợp và cây lớp.

6.3.4.3. Phương pháp Jacobson

Phương pháp này nhấn mạnh vào tính theo vết mô hình phân tích OOSE (object – oriented software engineering). Một phác thảo của quy trình phát triển do Jacobson đề nghị như sau:

- Xem xét sự thích nghi để các mô hình phân tích lý tưởng phù hợp với môi trường thế giới thực.
- Tạo ra các khối như các đối tượng thiết kế cơ bản.
 - Xác định một khối để thực hiện liên quan đến đối tượng phân tích.
 - Xác định khối giao diện, khối thực thể và khối điều khiển.
 - Mô tả cách thức các khối liên lạc khi thực hiện.
 - Xác định các kích thích được chuyển giữa các khối và thứ tự liên lạc.
- Tạo ra một biểu đồ tương tác cho biết cách các kích thích được chuyển giữa các khối.
- Tổ chức các khối trong các hệ thống con.
- Xem xét lại công việc thiết kế.

6.3.4.4. Phương pháp Rumbaugh

Kỹ thuật mô hình đối tượng (OMT) bao gồm các hành động thiết kế khuyến khích thiết kế quản lý hai mức trừu tượng khác nhau. Thiết kế hệ thống tập trung vào layout của các thành phần cần thiết để xây dựng sản phẩm hay hệ thống hoàn chỉnh. Thiết kế đối tượng nhấn mạnh vào layout chi tiết cho từng đối tượng. Một phác thảo của quy trình phát triển do Rumbaugh đề nghị như sau:

- Thực hiện thiết kế hệ thống.
 - Chia các mô hình phân tích thành các hệ thống con.
 - Xác định sự đồng bộ của vấn đề.
 - Phân công các xử lý và các công việc cho các hệ thống con.
 - Xác định các tài nguyên toàn cục và các kỹ thuật điều khiển yêu cầu để truy cập chúng.
 - Thiết kế kỹ thuật điều khiển thích hợp cho hệ thống.
 - Xem xét các thoả hiệp.
- Thiết kế đối tượng quản lý.
 - Chọn các phương thức từ mô hình phân tích.
 - Xác định các thuật toán cho mỗi phương thức.
 - Chọn cấu trúc dữ liệu phù hợp với thuật toán.
 - Xác định các lớp bên trong.
 - Sửa lại tổ chức lớp để tối ưu hoá truy cập dữ liệu và hoàn thiện khả năng tính toán.
 - Thiết kế các thuộc tính lớp.
- Thực hiện các kỹ thuật điều khiển đã được xác định trong thiết kế hệ thống.
- Điều chỉnh các cấu trúc lớp để tăng sức mạnh thừa kế.
- Thiết kế thông điệp để thực hiện các mối quan hệ đối tượng.
- Đóng gói các lớp và các liên kết trong một module.

6.3.4.5. Phương pháp Wirfs – Brock

Một phác thảo của quy trình phát triển vi mô do Wirfs – Brock đề nghị như sau:

- Xây dựng giao thức cho mỗi lớp.
 - Tính chế hợp đồng (contract) giữa các đối tượng vào các giao thức đã tính chế.
 - Thiết kế từng phương thức (tính trách nhiệm).
 - Thiết kế từng giao thức (thiết kế giao diện).

- Tạo ra đặc tả thiết kế cho mỗi lớp.
 - Mô tả chi tiết từng hợp đồng.
 - Xác định các trách nhiệm riêng.
 - Mô tả các thuật toán cho mỗi phương thức.
 - Chú ý các xem xét đặc biệt và các ràng buộc.
- Tạo ra đặc tả thiết kế cho mỗi hệ thống con.
 - Xác định các lớp bao.
 - Mô tả chi tiết các hợp đồng đề với nó hệ thống con là server.
 - Chú ý các xem xét đặc biệt và các ràng buộc.

Mặc dù các bước trong mỗi phương pháp thiết kế khác nhau nhưng toàn bộ quy trình thiết kế hướng đối tượng là nhất quán. Để thực hiện thiết kế hướng đối tượng, một kỹ sư phần mềm cần phải thực hiện các bước sau:

- Mô tả mỗi hệ thống con theo cách nó thực hiện.
 - Phân công các xử lý và công việc cho hệ thống con.
 - Chọn chiến lược thiết kế để thực hiện quản lý dữ liệu, hỗ trợ giao diện và quản lý công việc.
 - Thiết kế kỹ thuật điều khiển thích hợp cho hệ thống.
 - Xem xét lại các thoả hiệp.
- Thiết kế đối tượng.
 - Thiết kế mỗi phương thức tại mức thủ tục.
 - Xác định bất kỳ lớp bên trong.
 - Thiết kế cấu trúc dữ liệu bên trong cho các thuộc tính lớp.
- Thiết kế thông điệp.
 - Sử dụng sự cộng tác giữa các lớp và các mối quan hệ đối tượng, thiết kế mô hình thông điệp.
 - Xem xét lại mô hình thiết kế và nhắc lại khi được yêu cầu.

6.3.5. Các thành phần chung của mô hình thiết kế hướng đối tượng

Khi mô hình phân tích hoàn chỉnh được phát triển, kỹ sư hệ thống tập trung vào thiết kế hệ thống. Điều này được thực hiện bằng cách mô tả các đặc tính của các lớp con được yêu cầu để thực hiện cả yêu cầu của khách hàng và hỗ trợ môi trường cần thiết để bàn giao cho khách hàng.

Khi hệ thống con được xác định, chúng cần được phối hợp trong toàn bộ ngữ cảnh của yêu cầu khách hàng. Hệ thống con nào phù hợp cho yêu cầu khách hàng nào? Hệ thống con nào có đối tượng được xác định trong OOA? Những hệ thống con nào được thực hiện đồng thời và các thành phần hệ thống nào điều phối và điều khiển chúng? Các tài nguyên toàn cục được hệ thống con quản lý như thế nào?

Khi thiết kế hệ thống con, kỹ sư phần mềm cần phải xác định 4 thành phần thiết kế quan trọng như:

- *Miền vấn đề*: Hệ thống con chịu trách nhiệm thực hiện trực tiếp các yêu cầu khách hàng.
- *Tương tác con người*: Hệ thống con thực hiện giao diện người dùng (bao gồm các hệ thống GUI sử dụng lại được).
- *Quản lý công việc*: Hệ thống con chịu trách nhiệm điều khiển và điều phối các nhiệm vụ đồng thời được đóng gói trong hệ thống con hoặc giữa các hệ thống con.
- *Quản lý dữ liệu*: Hệ thống con chịu trách nhiệm lưu trữ đối tượng.

6.3.6. Quy trình thiết kế hệ thống

Rambaugh đã đưa ra các bước thiết kế như sau:

- Phân chia mô hình phân tích vào các hệ thống con.
- Xác định sự đồng bộ của vấn đề.
- Phân công các công việc và cách xử lý cho các hệ thống con.
- Xác định các tài nguyên toàn cục và các kỹ thuật điều khiển yêu cầu để truy cập chúng.
- Thiết kế kỹ thuật điều khiển thích hợp cho hệ thống.
- Xem xét các thỏa hiệp.

6.3.6.1. Phân chia mô hình phân tích vào các hệ thống con

Trong thiết kế, phân chia mô hình phân tích vào các hệ thống con nhằm xác định tập các cổ kết của lớp, quan hệ và các ứng xử, các thành phần thiết kế được đóng gói trong các hệ thống con.

Khi hệ thống con được xác định, nó phải thỏa mãn các tiêu chuẩn thiết kế sau:

- Hệ thống con phải có giao diện tốt thông qua toàn bộ các trao đổi với phần còn lại của hệ thống.
- Trường hợp ngoại lệ, một số ít các lớp liên lạc, các lớp trong hệ thống con cộng tác với nhau trong hệ thống con.
- Số lượng các hệ thống con nên nhỏ.
- Mỗi hệ thống con có thể phân chia nhỏ hơn để giảm độ phức tạp.

6.3.6.2. Sự đồng thời và phân phối hệ thống con

Các khía cạnh động của mô hình ứng xử đối tượng cung cấp các biểu thị sự đồng thời giữa các đối tượng (hoặc các hệ thống con). Nếu các đối tượng (hay hệ thống con) không hoạt động trong cùng một thời điểm, không cần xử lý đồng thời. Điều đó có nghĩa là các đối tượng (hay các hệ thống con) có thể được thực hiện trên cùng một phần cứng. Khi hệ thống con đồng thời có hai lựa chọn cấp phát sau:

- Phân phối cho mỗi hệ thống con một bộ xử lý riêng.
- Phân phối các hệ thống con cùng một bộ xử lý và cung cấp các hỗ trợ đồng thời qua các đặc trưng hệ điều hành.

Các công việc trong hệ thống hướng đối tượng được thiết kế bằng cách cô lập các luồng điều khiển. Để quyết định lựa chọn nào thích hợp, người thiết kế phải xem xét các yêu cầu hiệu năng, giá thành và tổng phí xác định bằng liên lạc giữa các xử lý.

6.3.6.3. Thành phần quản lý công việc

Coad và Yourdon đã đưa ra chiến lược cho thiết kế đối tượng quản lý các công việc đồng thời như sau:

- Đặc tính các công việc được quyết định.
- Công việc điều phối và các liên kết đối tượng được xác định.

- Điều phối và các công việc khác được tích hợp.

Đặc tính của công việc được xác định bằng cách hiểu cách thức các công việc khởi đầu. Thêm vào cách thức các công việc được khởi đầu, quyền ưu tiên và các tiêu chuẩn của công việc phải được xác định. Công việc ưu tiên cao phải ngay lập tức truy cập tài nguyên hệ thống. Công việc tiêu chuẩn cao phải tiếp tục vận hành, thậm chí ngay khi tài nguyên sẵn có bị giảm xuống hay hệ thống hoạt động trong trạng thái suy biến.

Khi các đặc tính của công việc được xác định, các thuộc tính đối tượng và các phương thức yêu cầu để đạt được sự điều phối và liên lạc với các công việc khác cũng được xác định. Các mẫu công việc cơ bản thường theo kiểu sau:

- *Tên công việc*: Tên của đối tượng.
- *Mô tả*: Mô tả thành văn mục đích của đối tượng.
- *Ưu tiên*: Ưu tiên công việc (cao, thấp, trung bình).
- *Dịch vụ*: Danh sách các hành động đối tượng chịu trách nhiệm.
- *Điều phối bởi*: Cách thức các ứng xử của đối tượng được gọi.
- *Liên kết bằng*: Giá trị dữ liệu vào và ra thích hợp cho công việc.

6.3.6.4. Thành phần quản lý dữ liệu

Quản lý dữ liệu bao gồm hai lĩnh vực xem xét sau:

- Quản lý dữ liệu là tiêu chuẩn cho bản thân ứng dụng.
- Tạo ra hạ tầng cho lưu trữ và lấy các đối tượng.

Trong ngữ cảnh hệ thống, hệ thống quản lý dữ liệu thường được dùng như một kho dữ liệu cho toàn bộ các hệ thống con.

Thiết kế thành phần quản lý dữ liệu bao gồm thiết kế các thuộc tính và các phương thức được yêu cầu cho đối tượng quản lý. Các thuộc tính thích hợp được mở rộng cho mọi đối tượng trong miền vấn đề và cung cấp thông tin để trả lời câu hỏi: Nó lưu trữ bản thân như thế nào?

6.3.6.5. Thành phần quản lý tài nguyên

Có nhiều loại tài nguyên khác nhau sẵn có trong hệ thống hoặc sản phẩm OO và nhiều thể nghiệm, hệ thống con cạnh tranh cho một tài nguyên trong một thời điểm.

Tài nguyên hệ thống toàn cục có thể là các thực thể ngoài (như các ổ đĩa, bộ xử lý, các kênh liên lạc) hoặc trừu tượng (như cơ sở dữ liệu, đối tượng). Rambaugh gợi ý rằng, mỗi tài nguyên đều phải được một “đối tượng cảnh giới – guardian object” sở hữu. Guardian object giữ các tài nguyên, điều khiển truy cập và làm giảm các yêu cầu xung đột cho nó.

6.3.6.6. Thành phần tương tác người – máy

Mặc dù các thành phần tương tác người – máy được thực hiện trong ngữ cảnh của miền vấn đề, nhưng bản thân tương tác mô tả hệ thống con theo tiêu chuẩn cho phần lớn các ứng dụng hiện đại.

Khi một tác nhân và kịch bản sử dụng được xác định, một cây lệnh được xác định. Cây lệnh xác định các loại menu hệ thống (menu bar hay tool palette) và các chức năng con sẵn có trong ngữ cảnh của các loại menu hệ thống chính (menu window). Cây lệnh được tinh chế đến khi mọi trường hợp sử dụng có thể thực hiện bằng cách duyệt cây của chức năng.

6.3.7. Quy trình thiết kế đối tượng

6.3.7.1. Xác định đối tượng

Một mô tả của đối tượng có thể là một trong hai dạng sau:

- *Mô tả giao thức:* Thiết lập giao diện của một đối tượng bằng cách xác định mỗi thông điệp mà đối tượng có thể nhận và các phương thức liên quan đối tượng thực hiện khi nó nhận được thông điệp.
- *Mô tả thực hiện:* Chỉ ra chi tiết thực hiện cho mỗi phương thức bằng cách chuyển thông điệp giữa các đối tượng. Chi tiết thực hiện bao gồm thông tin về các phần riêng của đối tượng, đó là các chi tiết bên trong về cấu trúc dữ liệu, mô tả các thuộc tính của đối tượng và các thủ tục chi tiết mô tả phương thức.

Giao thức mô tả là một tập các thông điệp và các chú thích tương ứng với mỗi thông điệp.

Với các hệ thống lớn có nhiều thông điệp, ta thường tạo ra các danh mục thông điệp.

Một mô tả thực hiện của một đối tượng cung cấp các chi tiết bên trong được yêu cầu để thực hiện nhưng không cần thiết cho viện dẫn. Người thiết kế đối tượng phải cung cấp mô tả thực hiện và do đó phải tạo ra chi tiết bên

trong của đối tượng. Tuy nhiên, một nhà thiết kế khác sử dụng đối tượng hoặc thể nghiệm của đối tượng chỉ yêu cầu mô tả giao thức nhưng không yêu cầu mô tả thực hiện.

Một mô tả thực hiện bao gồm các thông tin sau:

1. Đặc tả tên của đối tượng và tham chiếu tới lớp.
2. Đặc tả cấu trúc dữ liệu riêng với việc xác định các mục dữ liệu và các kiểu.
3. Mô tả có tính thủ tục mỗi phương thức.

6.3.7.2. Thuật toán thiết kế và cấu trúc dữ liệu

Có nhiều mô tả trong mô hình phân tích và thiết kế hệ thống cung cấp đặc tả cho toàn bộ các phương thức hay thuộc tính. Thuật toán và cấu trúc dữ liệu được thiết kế sử dụng cách tiếp cận có khác biệt đôi chút với cách tiếp cận trong công nghệ phần mềm thông thường.

Một thuật toán được tạo ra để thực hiện một đặc tả cho mỗi phương thức. Trong nhiều trường hợp, thuật toán chỉ là tính toán đơn giản hoặc dãy các thủ tục được thực hiện như một module phần mềm tự chứa (self – contained). Tuy nhiên, nếu đặc tả phức tạp, cần phải mô hình hoá phương thức. Các kỹ thuật thiết kế thủ tục thông thường được sử dụng để hoàn thành công việc này.

Cấu trúc dữ liệu được thiết kế đồng thời với thuật toán. Do các phương thức thao tác các thuộc tính của lớp, thiết kế cấu trúc dữ liệu phản ánh tốt nhất thuộc tính sinh ra trong thiết kế thuật toán của các phương thức tương ứng.

Mặc dù có nhiều kiểu phương thức khác nhau nhưng chúng thường được chia thành 3 loại chính như sau:

1. Phương thức thao tác dữ liệu (thêm, xoá, định dạng lại, chọn).
2. Phương thức thực hiện tính toán.
3. Phương thức điều khiển đối tượng cho sự kiện điều khiển.

Để tối ưu hoá thiết kế hướng đối tượng, Rumbaugh đã đưa ra các gợi ý sau:

1. Xem lại mô hình đối tượng – quan hệ để đảm bảo rằng thiết kế được thực hiện sử dụng tài nguyên hiệu quả và nói lòng rằng buộc thực hiện, thêm vào các dư thừa nếu thấy cần thiết.

2. Xem lại thuộc tính cấu trúc dữ liệu và các thuật toán tương ứng để tăng cường tính hiệu quả của xử lý.

3. Tạo ra thuộc tính mới để giữ các thông tin thừa kế, do đó tránh được việc tính toán lại.

6.3.7.3. Thành phần chương trình và giao diện

Một khía cạnh quan trọng của chất lượng thiết kế phần mềm là molarity – các đặc tả của các thành phần chương trình được kết hợp để tạo ra chương trình hoàn chỉnh. Cách tiếp cận hướng đối tượng xác định đối tượng như một thành phần chương trình, bản thân nó liên kết với các thành phần khác (dữ liệu riêng, phương thức). Nhưng các đối tượng và phương thức là chưa đủ. Trong thiết kế, chúng ta phải xác định giao diện tồn tại giữa các đối tượng và toàn bộ cấu trúc của đối tượng.

6.3.8. Mẫu thiết kế

6.3.8.1. Mô tả mẫu thiết kế

Mọi mẫu thiết kế có thể mô tả bằng cách đặc tả 4 loại thông tin sau:

- Tên mẫu thiết kế.
- Vấn đề mà mẫu thường được áp dụng.
- Đặc tính của mẫu thiết kế.
- Thứ tự ứng dụng mẫu thiết kế.

Tên mẫu thiết kế là trừu tượng, chứa đựng các thông tin quan trọng về tính ứng dụng và mục đích của nó. Mô tả vấn đề, xác định môi trường và điều kiện phải tồn tại để mẫu thiết kế có thể áp dụng. Đặc tính của mẫu chỉ ra các thuộc tính của thiết kế cần được điều chỉnh để mẫu thích hợp với nhiều vấn đề. Các thuộc tính mô tả đặc tính của thiết kế được kiểm tra (như cơ sở dữ liệu). Cuối cùng, thứ tự liên kết sử dụng mẫu thiết kế cung cấp sự phân nhánh của các quyết định thiết kế.

6.3.8.2. Sử dụng mẫu thiết kế

Trong hệ thống hướng đối tượng, mẫu thiết kế có thể được áp dụng trong 2 kỹ thuật khác nhau: thừa kế và hợp thành. Sử dụng thừa kế, một

mẫu thiết kế có thể trở thành một mẫu cho lớp con mới. Các thuộc tính và phương thức tồn tại trong mẫu trở thành một phần của lớp con.

Hợp thành là khái niệm liên quan đến các đối tượng kết hợp và được hình thành do đối tượng có chức năng phức tạp. Đối tượng phức tạp có thể được lắp ráp bằng cách chọn một tập các mẫu thiết kế và tạo ra đối tượng (hoặc hệ thống con) thích hợp. Mỗi mẫu thiết kế được đối xử như một hộp đen, và liên lạc giữa các mẫu chỉ có thể xảy ra bằng các giao diện được thiết kế tốt.

6.4. Thiết kế thời gian thực

Thiết kế các hệ thống tính toán thời gian thực là một nhiệm vụ phức tạp và có tính thách thức nhất mà kỹ sư phần mềm cần tiến hành. Bởi chính bản chất của nó, phần mềm cho hệ thống thời gian thực yêu cầu các kỹ thuật phân tích, thiết kế và kiểm thử mà chưa được biết đến trong các lĩnh vực khác.

Phần mềm thời gian thực liên hệ chặt chẽ với thế giới bên ngoài, đáp ứng vấn đề (thế giới thực) trong khuôn khổ thời gian do lĩnh vực vấn đề chi phối. Vì phần mềm thời gian thực phải vận hành dưới những ràng buộc hiệu năng chặt chẽ nên việc thiết kế phần mềm thường bị chi phối bởi phần cứng cũng như kiến trúc phần mềm, các đặc trưng hệ điều hành cũng như các yêu cầu áp dụng, tính thất thường của ngôn ngữ lập trình cũng như các vấn đề thiết kế.

6.4.1. Các xem xét hệ thống

Giống như bất cứ hệ thống máy tính nào, hệ thống thời gian thực phải tích hợp với các yếu tố phần cứng, phần mềm, con người và cơ sở dữ liệu để thu được một tập các yêu cầu chức năng và hoàn thiện. Kỹ sư hệ thống phải cấp phát chức năng và sự hoàn thiện trong các phần tử hệ thống. Vấn đề đối với hệ thống thời gian thực là phải cấp phát đúng. Sự hoàn thiện thời gian thực cũng quan trọng như chức năng, nhưng các quyết định cấp phát thường khó đảm bảo. Liệu một thuật toán xử lý có đáp ứng được các ràng buộc thời gian khắt khe, hay chúng ta phải xây dựng phần cứng đặc biệt để làm việc đó. Liệu hệ điều hành bán sẵn có đáp ứng việc điều giải ngắt có

hiệu quả, làm đa nhiệm và truyền thông, hay chúng ta phải xây dựng một hệ điều hành theo nhu cầu? Liệu phần cứng đặc biệt đi kèm với phần mềm được đề nghị có đáp ứng được tiêu chuẩn hiệu năng hay không? Những câu hỏi này và còn nhiều câu hỏi khác phải được các kỹ sư hệ thống trả lời.

6.4.2. Hệ thống thời gian thực

Hệ thống thời gian thực sinh ra một hành động nào đó để đáp ứng các sự kiện bên ngoài. Để hoàn thành chức năng này, chúng thực hiện việc thu thập và kiểm soát dữ liệu tốc độ cao trong những ràng buộc thời gian và độ tin cậy nghiêm ngặt. Vì những ràng buộc này là nghiêm ngặt nên các hệ thống thời gian thực thường chuyên dụng cho một ứng dụng.

Trong nhiều năm, khách hàng chính của các hệ thống thời gian thực là giới quân sự. Tuy nhiên ngày nay giá phần cứng giảm đáng kể đã làm cho các hệ thống này được ứng dụng ngày càng nhiều hơn, đa dạng hơn trong nhiều lĩnh vực như trong điều khiển tiến trình, tự động hoá công nghiệp, nghiên cứu y học và khoa học, đồ hoạ máy tính, truyền thông mạng cục bộ và mạng diện rộng, hệ thống hàng không, kiểm thử có máy tính trợ giúp và rất nhiều thiết bị công nghiệp.

6.4.2.1. Vấn đề tích hợp và hiệu năng

Tóm lại, hệ thống thời gian thực đặt ra cho kỹ sư hệ thống những quyết định khó khăn đối với phần mềm và phần cứng. Một khi phần tử phần mềm được cấp phát, các yêu cầu phần mềm chi tiết đã được thiết lập thì phải thiết kế phần mềm nền tảng. Trong số nhiều mối quan tâm thiết kế thời gian thực có sự điều phối giữa các nhiệm vụ thời gian thực, xử lý ngắt hệ thống, xử lý vào/ra để đảm bảo rằng không dữ liệu nào bị mất, xác định các ràng buộc thời gian bên trong và bên ngoài hệ thống, đảm bảo độ chính xác của cơ sở dữ liệu.

Mỗi mối quan tâm thiết kế thời gian thực cho phần mềm phải được áp dụng cho ngữ cảnh, hiệu năng hệ thống. Trong phần lớn các trường hợp, hiệu năng của hệ thống thời gian thực được đo bằng một hay nhiều đặc trưng có liên quan tới thời gian, nhưng cách đo khác như độ dung sai cũng có thể được sử dụng.

Một số hệ thống thời gian thực được thiết kế cho những ứng dụng trong đó chỉ có thời gian đáp ứng hay tỷ lệ truyền dữ liệu là cấp bách.

Những ứng dụng thời gian thực khác đòi hỏi việc tối ưu hoá cả hai tham biến bởi điều kiện tải cao điểm. Các hệ thống thời gian thực phải xử lý mức tải cao điểm trong khi thực hiện một số nhiệm vụ đồng thời.

Vì hiệu năng của hệ thống thời gian thực được xác định chủ yếu thông qua thời gian đáp ứng của hệ thống và tỷ lệ truyền dữ liệu nên điều quan trọng là phải hiểu hai tham biến này. Thời gian đáp ứng hệ thống là thời gian trong đó hệ thống phát hiện một sự kiện bên trong hay bên ngoài và đáp ứng bằng hành động. Thông thường, việc phát hiện hành động và phát sinh đáp ứng là đơn giản. Đó là việc xử lý thông tin về sự kiện để xác định đáp ứng thích hợp có thể bao gồm những thuật toán phức tạp, tốn thời gian.

Trong số những tham biến mấu chốt ảnh hưởng tới thời gian đáp ứng có chuyển ngữ cảnh và trễ ngắt. Chuyển ngữ cảnh bao gồm thời gian và tổng phí để chuyển giữa các nhiệm vụ, còn trễ ngắt là thời gian trễ trước khi việc chuyển thực tế xảy ra. Các tham biến khác ảnh hưởng tới thời gian đáp ứng là tốc độ tính toán và việc thâm nhập bộ nhớ ngoài.

Tỷ lệ truyền dữ liệu cho biết dữ liệu được truyền vào hoặc ra hệ thống nhanh hay chậm, một cách tuần tự hay song song cũng như dạng tương tự hay số. Các nhà cung cấp phần cứng thường đưa ra các giá trị thời gian và khả năng thực hiện các đặc trưng hiệu năng. Tuy nhiên, các đặc tả phần cứng về hiệu năng thường được đo một cách cô lập và ít có giá trị trong việc xác định hiệu năng của hệ thống thời gian thực tổng thể. Do đó, hiệu năng của thiết bị vào/ra, độ trễ bus, kích cỡ bộ đệm, hiệu năng đĩa và các nhân tố khác mặc dù quan trọng cũng chỉ là một phần trong thiết kế hệ thống thời gian thực.

Hệ thống thời gian thực thường đòi hỏi xử lý luồng dữ liệu vào liên tục. Thiết kế phải đảm bảo dữ liệu không bị bỏ qua. Bên cạnh đó, hệ thống thời gian thực phải đáp ứng cho các sự kiện không đồng bộ. Do đó, dây và khối lượng dữ liệu tới không thể nào dễ dàng dự đoán trước được.

Các ứng dụng phần mềm đều phải tin cậy, các hệ thống thời gian thực cũng phải có yêu cầu đặc biệt về độ tin cậy, về việc chạy lại và phục hồi lỗi. Thế giới thời gian thực được điều phối và kiểm soát nên việc mất điều khiển hay mất kiểm soát, hoặc cả hai là không được phép trong nhiều hoàn cảnh (ví dụ như hệ thống kiểm soát không lưu). Do vậy, các hệ thống thời gian thực phải chứa cơ chế cho chạy lại, khôi phục lỗi và phải có độ dư thừa để đảm bảo sao lưu.

6.4.2.2. Xử lý ngắt

Một đặc trưng hay dùng để phân biệt các hệ thống thời gian thực với các kiểu hệ thống khác là việc xử lý ngắt. Hệ thống thời gian thực phải đáp ứng với các kích thích bên ngoài – các ngắt – trong một khuôn khổ thời gian do thể giới bên ngoài ấn định. Thường có nhiều kích thích (ngắt) hiện hữu nên phải thiết lập thứ tự ngắt ưu tiên. Nói cách khác, nhiệm vụ quan trọng nhất phải được thực hiện ngay dù có bị ràng buộc về thời gian hay các sự kiện.

Xử lý ngắt bắt buộc không chỉ lưu trữ thông tin để máy tính có thể chạy lại đúng nhiệm vụ đã bị ngắt mà còn phải tránh việc chết tắc và các chu trình vô hạn. Cách tiếp cận tổng thể xử lý ngắt được minh họa như trong hình 6.7.

Luồng xử lý thông thường bị “ngắt” bởi một sự kiện do phần cứng bộ xử lý phát hiện ra. Một sự kiện là sự xuất hiện bất kỳ đòi hỏi phải phục vụ ngay lập tức, được sinh ra hoặc bởi phần cứng hoặc bởi phần mềm. Trạng thái của chương trình bị ngắt (tất cả nội dung các thanh ghi, các khối điều khiển...) được cất giữ và điều khiển được truyền cho chương trình làm dịch vụ ngắt, tức là nhảy tới một phần mềm thích hợp để xử lý ngắt. Khi hoàn thành xong dịch vụ ngắt thì trạng thái của máy được khôi phục lại và luồng thông thường được tiếp tục.

Luồng “xử lý” thông thường

Ngắt được
thông báo

Xử lý ngắt

- Cất giữ trạng thái chương trình bị ngắt
- Xác định bản chất ngắt
- Thực hiện dịch vụ ngắt
- Khôi phục trạng thái của chương trình bị ngắt
- Trở về chương trình bị ngắt

Hình 6.7. Xử lý ngắt

Trong nhiều tình huống, bản thân dịch vụ ngắt cho một sự kiện có thể bị ngắt bởi một sự kiện khác có mức ưu tiên cao hơn. Mức ưu tiên ngắt có

thể được thiết lập. Nếu một tiến trình có mức ưu tiên thấp ngẫu nhiên được phép ngắt một tiến trình có mức ưu tiên cao thì sẽ khó chạy lại các tiến trình theo đúng trật tự và có thể làm phát sinh chu trình vô hạn.

Để xử lý ngắt và vẫn đáp ứng các ràng buộc thời gian thực, nhiều hệ điều hành thời gian thực sẽ tiến hành tính toán động để xác định các mục tiêu hệ thống đã đạt được hay chưa. Những tính toán động này dựa trên tần số trung bình số lần xuất hiện của sự kiện, khối lượng thời gian cần phục vụ (nếu chúng được phục vụ), trình có thể ngắt chúng và tạm thời ngăn chúng phục vụ.

Nếu các tính toán động chỉ ra rằng không thể xử lý được các sự kiện xuất hiện trong hệ thống nhưng vẫn đáp ứng các ràng buộc thời gian thì hệ thống sẽ quyết định sơ đồ hành động. Một sơ đồ có thể bao gồm việc đặt bộ đệm để có thể xử lý nhanh chóng khi hệ thống sẵn sàng.

6.4.2.3. Cơ sở dữ liệu thời gian thực

Giống như nhiều hệ thống xử lý dữ liệu, hệ thống thời gian thực thường đi đôi với chức năng quản trị cơ sở dữ liệu. Tuy nhiên, cơ sở dữ liệu phân tán dường như là cách tiếp cận được ưa thích nhất trong hệ thống thời gian thực vì đa nhiệm thông dụng và dữ liệu thường được xử lý song song. Nếu cơ sở dữ liệu phân tán thì từng nhiệm vụ có thể thâm nhập vào dữ liệu của chúng nhanh hơn và tin cậy hơn, ít tắc nghẽn hơn cơ sở dữ liệu tập trung. Việc dùng cơ sở dữ liệu phân tán cho các ứng dụng thời gian thực phân chia lưu thông vào/ra và làm ngắn đi hàng nhiệm vụ đang chờ để thâm nhập vào cơ sở dữ liệu. Một sai sót của một cơ sở dữ liệu sẽ hiếm khi gây ra hỏng hóc toàn bộ hệ thống nếu sự dư thừa được xây dựng sẵn.

Tính hiệu quả hiệu năng đạt được thông qua việc dùng cơ sở dữ liệu phân tán phải tương ứng với vấn đề tiềm năng liên quan đến phân hoạch và tái tạo. Mặc dù việc dư thừa dữ liệu làm tăng thời gian đáp ứng do cung cấp nhiều nguồn thông tin, nhưng các yêu cầu tái tạo cho các tệp phân tán cũng tạo ra các vấn đề tổng chi phí và các nguồn vì tất cả các bản sao tệp đều phải được cập nhật. Bên cạnh đó, việc dùng các cơ sở dữ liệu phân tán sẽ đưa đến vấn đề điều khiển tương tranh. Điều khiển tương tranh bao gồm việc đồng bộ hoá các cơ sở dữ liệu sao cho mọi bản sao đều có thông tin thống nhất, đúng đắn, tự do cho việc truy nhập.

Cách tiếp cận quy ước dựa trên yếu tố đã được biết như khoá (locking) và dấu thời gian (time stamps). Theo những khoảng đều đặn, các nhiệm vụ sau đây được khởi đầu:

- Cơ sở dữ liệu được hãm lại để việc điều khiển tương tranh được đảm bảo không vào, ra nào được thực hiện.
- Việc cập nhật xuất hiện như yêu cầu.
- Cơ sở dữ liệu được mở lại.
- Các tệp được làm hợp lệ để đảm bảo rằng tất cả mọi cập nhật đều đã được thực hiện đúng.
- Việc cập nhật phải được hoàn tất.

Mọi việc chính đều được điều phối bởi đồng hồ thời gian. Vấn đề trễ bao gồm trong những thủ tục này, cũng như vấn đề tránh cập nhật không nhất quán và bế tắc là giảm bớt việc dùng rộng rãi cơ sở dữ liệu phân tán.

Tuy nhiên, một số kỹ thuật đã được phát triển để tăng tốc việc cập nhật và để giải quyết vấn đề tương tranh. Một trong những kỹ thuật này đó là loại trừ nơi ghi, duy trì tính nhất quán của các tệp tái tạo bằng cách cho phép chỉ riêng một nhiệm vụ ghi là được cập nhật tệp. Do đó, nó khử bớt tổng phí cao của thủ tục khoá hay dấu thời gian.

6.4.2.4. Hệ điều hành thời gian thực

Việc chọn hệ điều hành thời gian thực (real – time operating system – RTOS) cho một ứng dụng không phải là một công việc dễ dàng. Có thể một số cách phân loại hệ điều hành nhưng phần lớn không thích hợp, chúng có các ưu điểm và nhược điểm rõ ràng, có sự lẫn lộn nhau đáng kể về khả năng, hệ thống đích và các đặc trưng khác.

Một số hệ điều hành thời gian thực áp dụng cho một phạm vi rộng các cấu hình hệ thống, trong khi các hệ khác lại ăn khớp với một bộ vi xử lý đặc biệt, bất kể môi trường điện tử bao quanh. RTOS thực hiện khả năng của chúng qua một tổ hợp các đặc trưng phần mềm và gia tăng sự đa dạng của các khả năng vi mã được cài đặt trong phần cứng.

Ngày nay, người ta hay dùng hai lớp rộng các hệ điều hành cho công việc thời gian thực:

- RTOS chuyên dụng được thiết kế chuyên cho các ứng dụng thời gian thực.
- Hệ điều hành vạn năng được nâng cao để cung cấp khả năng thời gian thực.

Việc dùng hệ chấp hành thời gian thực (real – time executive) làm cho hiệu năng thời gian thực thành khả thi đối với hệ điều hành vạn năng. Hành xử giống như phần mềm ứng dụng, hệ chấp hành thực hiện một số chức năng của hệ điều hành, đặc biệt là các chức năng ảnh hưởng tới hiệu năng thời gian thực nhanh hơn và hiệu quả hơn các hệ điều hành vạn năng.

Tất cả các hệ điều hành phải có một cơ chế lập lịch ưu tiên, nhưng RTOS cung cấp một cơ chế ưu tiên cho phép ngắt có mức ưu tiên cao hơn được thực hiện trước so với các ngắt có độ ưu tiên thấp hơn. Hơn nữa, vì việc ngắt xảy ra để đáp ứng cho những sự kiện không đồng bộ, không tuần hoàn cho nên chúng được thực hiện mà không mất thời gian trao đổi chương trình từ bộ nhớ đĩa. Kết quả là, để đảm bảo thời gian đáp ứng theo yêu cầu, hệ điều hành thời gian thực phải có một cơ chế khoá bộ nhớ – tức là khoá ít nhất một số chương trình trong bộ nhớ chính sao cho tránh được tổng phí trao đổi.

Để xác định hệ điều hành thời gian thực phù hợp nhất với ứng dụng, cách đo phẩm chất RTOS được xác định và đánh giá. Thời gian chuyển ngữ cảnh và trễ ngắt xác định khả năng ngắt – khía cạnh quan trọng nhất của hệ điều hành thời gian thực. Thời gian chuyển ngữ cảnh là thời gian hệ điều hành cần để cất giữ trạng thái máy tính và nội dung của các thanh ghi sao cho nó có thể trở về trạng thái đang xử lý sau khi hoàn thành dịch vụ ngắt.

Trễ ngắt, thời gian trễ tối đa trước khi hệ thống chuyển sang một nhiệm vụ sẽ xuất hiện vì trong hệ điều hành thường có các đường xử lý gắng hay không đồng thời vào được mà phải hoàn tất trước khi một ngắt được xử lý.

Chiều dài của những con đường này (số các lệnh) trước khi hệ thống phục vụ một ngắt chỉ ra độ trễ thời gian xấu nhất. Trường hợp tồi nhất xuất hiện nếu một ngắt ưu tiên cao hơn được sinh ra ngay sau khi hệ thống đi vào một đường gắng giữa ngắt và dịch vụ ngắt. Nếu thời gian quá lâu thì hệ thống có thể bỏ một phần dữ liệu không khôi phục được. Điều quan trọng là người thiết kế biết về độ trễ thời gian để hệ thống có thể bù lại cho nó.

Nhiều hệ điều hành thực hiện đa nhiệm hay xử lý tương tranh một yêu cầu chính khác cho hệ thống thời gian thực. Nhưng để thao tác cho thời gian thực, tổng phí hệ thống bao gồm thời gian chuyển và không gian bộ nhớ cần dùng phải thấp.

6.4.2.5. Ngôn ngữ thời gian thực

Do các yêu cầu về hiệu năng và độ tin cậy đối với các hệ thống thời gian thực, nên việc lựa chọn ngôn ngữ lập trình là quan trọng. Nhiều ngôn ngữ lập trình vạn năng như C, FORTRAN, Modula-2 được dùng một cách có hiệu quả cho những ứng dụng thời gian thực. Tuy nhiên, một số lớp – cái gọi là các “ngôn ngữ thời gian thực” như Ada, Jovial và các ngôn ngữ khác lại thường được sử dụng trong các ứng dụng truyền thông và quân sự.

Việc tổ hợp các đặc trưng làm cho ngôn ngữ thời gian thực khác biệt với ngôn ngữ vạn năng. Những khác biệt này bao gồm khả năng đa nhiệm, các kết cấu trực tiếp cài đặt chức năng thời gian thực, và các tính năng ngôn ngữ hiện đại giúp đảm bảo tính đúng đắn của chương trình.

Một ngôn ngữ lập trình trực tiếp hỗ trợ cho đa nhiệm rất quan trọng vì hệ thống thời gian thực phải đáp ứng cho các sự kiện không đồng bộ xuất hiện đồng thời. Mặc dù nhiều RTOS cung cấp khả năng đa nhiệm, phần mềm thời gian thực nhúng vẫn thường tồn tại mà không có hệ điều hành. Thay vào đó, các ứng dụng nhúng được viết trong một ngôn ngữ cung cấp hỗ trợ thời gian chạy đủ để thực hiện chương trình thời gian thực. Hỗ trợ thời gian chạy đòi hỏi ít bộ nhớ hơn hệ điều hành và nó rất tương thích với ứng dụng, do vậy làm tăng hiệu năng.

Một hệ thống thời gian thực đã được thiết kế để thích nghi với nhiều nhiệm vụ thì cũng phải thích nghi với việc đồng bộ hoá giữa các nhiệm vụ. Một ngôn ngữ lập trình trực tiếp hỗ trợ cho các nguyên sơ đồng bộ hoá như SCHEDULE, SIGNAL, và WAIT làm đơn giản rất nhiều việc dịch từ thiết kế sang mã hoá. Chỉ lệnh SCHEDULE lập lịch cho một tiến trình dựa trên thời gian hay sự kiện, chỉ lệnh SIGNAL và WAIT thao tác một cờ đặc biệt, gọi là cờ báo hiệu semaphore, làm cho các nhiệm vụ tương tranh có thể đồng bộ hoá.

Cuối cùng, các tính năng tạo điều kiện cho lập trình tin cậy cần thiết vì các chương trình thời gian thực thường lớn và phức tạp. Các tính năng này

bao gồm cả lập trình module, bó buộc mạnh về kiểu dữ liệu và nhiều kết cấu định nghĩa dữ liệu, điều khiển.

6.4.2.6. Đồng bộ hoá và truyền thông nhiệm vụ

Hệ thống đa nhiệm phải chuyển giao một cơ chế cho nhiệm vụ truyền thông tin lẫn nhau cũng như đảm bảo sự đồng bộ của chúng. Với những chức năng này, hệ điều hành và ngôn ngữ với hỗ trợ thời gian chạy thông thường hay dùng cách đặt cờ hiệu báo sắp hàng, hộp thư hay hệ thống thông báo. Cờ báo hiệu cung cấp việc đồng bộ hoá và đặt tín hiệu nhưng không chứa thông tin. Thông báo cũng tương tự như cờ hiệu, ngoại trừ chúng mang thông tin có liên quan. Hộp thư không báo hiệu thông tin nhưng thay vì thế lại chứa thông tin.

Cờ báo hiệu sắp hàng là các nguyên sơ phần mềm giúp quản lý lưu thông. Chúng cung cấp một phương pháp chỉ đạo nhiều hàng đợi, chẳng hạn hàng đợi nhiệm vụ đang chờ tài nguyên, thâm nhập cơ sở dữ liệu và các thiết bị. Cờ tín hiệu điều phối (đồng bộ hoá) các nhiệm vụ đang đợi dù chúng đang đợi bất kỳ cái gì mà không để cho các nhiệm vụ hay tài nguyên can thiệp lẫn nhau.

Trong hệ thống thời gian thực, cờ báo hiệu thường được dùng để cài đặt và quản lý hộp thư. Hộp thư là những vị trí lưu trữ tạm thời (cũng được gọi là thùng hay các bộ đệm thông báo), các thông báo được gửi từ tiến trình này sang tiến trình kia. Tiến trình tạo ra một mẫu thông tin rồi gửi vào trong thùng thư và báo hiệu cho tiến trình tiêu thụ biết để lấy mẫu thông tin trong hộp thư đó sử dụng.

Một số cách tiếp cận tới các hệ thống thời gian thực hay hệ thống hỗ trợ khi chạy coi hộp thư là cách hiệu quả nhất để cài đặt liên lạc giữa các tiến trình. Một số hệ điều hành thời gian thực còn cung cấp chỗ để gửi và nhận con trỏ tới dữ liệu hộp thư. Điều này loại bỏ sự cần thiết phải truyền tất cả dữ liệu. Do vậy, tiết kiệm thời gian và chi phí.

Cách tiếp cận thứ ba tới truyền thông và đồng bộ hoá trong các tiến trình là hệ thống thông báo. Với hệ thống thông báo, một tiến trình gửi một thông báo cho một tiến trình khác. Tiến trình sau tự động được kích hoạt bằng hệ thống hỗ trợ khi chạy hay hệ điều hành để xử lý thông báo. Một hệ thống như vậy phải chịu tổng phí vì nó truyền thông tin thực tại, nhưng nó mềm dẻo hơn và dễ dùng.

6.4.3. Phân tích và mô phỏng hệ thống thời gian thực

Tập các thuộc tính động không thể tách khỏi các yêu cầu chức năng của hệ thống thời gian thực:

- Xử lý các ngắt và chuyển ngữ cảnh.
- Thời gian đáp ứng.
- Tỷ lệ truyền dữ liệu và hiệu suất.
- Cấp phát tài nguyên và xử lý ưu tiên.
- Đồng bộ hoá các nhiệm vụ và truyền thông giữa các nhiệm vụ.

Mỗi thuộc tính hiệu năng này đều xác định được nhưng khó kiểm chứng được xem liệu các phần tử hệ thống có đạt đến sự đáp ứng mong muốn hay không, tài nguyên hệ thống có đủ để thoả mãn các yêu cầu tính toán hay không, các thuật toán xử lý có đảm bảo đúng tốc độ hay không.

Qua phân tích các hệ thống thời gian thực bằng việc mô hình hoá và mô phỏng, kỹ sư hệ thống đánh giá được các vấn đề thời gian và kích cỡ. Mặc dù một số kỹ thuật đã được đề nghị nhưng cách tiếp cận phân tích và thiết kế thời gian thực vẫn còn trong giai đoạn non trẻ.

6.4.3.1. Phân tích hệ thống thời gian thực bằng công cụ toán học

Công cụ toán học giúp kỹ sư hệ thống mô hình hoá các phần tử hệ thống thời gian thực, định giá các vấn đề thời gian và kích cỡ đã được Thomas McCabe đề nghị. Dựa trên các kỹ thuật phân tích luồng dữ liệu, cách tiếp cận của McCabe giúp người phân tích có thể mô hình hoá cả các phần tử phần cứng và phần mềm của hệ thống thời gian thực, biểu diễn điều khiển theo xác suất; áp dụng cách phân tích mạng, lý thuyết xếp hàng, đồ thị và mô hình toán học Markov để xác định thời gian hệ thống và kích cỡ tài nguyên.

Kỹ thuật phân tích thời gian thực (DFD) của McCabe được dự đoán trên biểu đồ luồng dữ liệu của hệ thống thời gian thực. Tuy nhiên, thay vì dùng DFD theo quy ước thì McCabe cho rằng, các phép biến đổi của một DFD có thể biểu thị như các trạng thái của tiến trình của xích Markov (mô hình hàng đợi xác suất) và dữ liệu truyền qua chúng cho phép chuyển giữa các trạng thái tiến trình.

Kỹ thuật này giúp kỹ sư hệ thống tập dượt mô hình theo cách tương tác. Tuy nhiên, người ta mong muốn có một sự mô phỏng mạnh mẽ hơn.

Hiệu năng dưới điều kiện ngẫu nhiên trong cả hai tình huống điển hình và không điển hình cần được đánh giá lại. Với những tình huống cần tới việc mô phỏng mô hình thời gian thực mạnh mẽ hơn thì ngôn ngữ điều khiển mô phỏng (SCL) giúp kỹ sư điều khiển chung được các cách thực hiện, đồng thời khai thác sức mạnh của công cụ để chuyển giao chi tiết. Nhà phân tích có thể gán xác suất chuyển cho từng đường chảy dữ liệu.

Mỗi tiến trình trong mô hình dựa trên DFD có thể gán cho một “đơn giá” biểu thị thời gian thực hiện hay thực tế được ước lượng cần cho việc thực hiện chức năng của nó và một giá trị mô tả số các ngắt hệ thống tương ứng với tiến trình đó. Do vậy, mô hình này được phân tích bằng cách dùng một tập các công cụ toán học và tính:

1. Kỳ vọng việc dùng đến tiến trình.
2. Thời gian dành cho hệ thống khi tiến trình bắt đầu tại một điểm xác định.
3. Toàn bộ thời gian dành cho hệ thống.

Hiển nhiên, độ chính xác của cách tiếp cận phân tích của McCabe cũng chỉ ở mức ước lượng về xác suất luồng, tỷ lệ tới và tỷ lệ đầy đủ. Tuy nhiên, nó cũng có có ý nghĩa vì đã đưa ra nhiều quan điểm phân tích về hệ thống thời gian thực.

6.4.3.2. Mô phỏng và các kỹ thuật mô hình hoá cho hệ thống thời gian thực

Phân tích toán học về hệ thống thời gian thực biểu thị cách tiếp cận dùng để hiểu hiệu năng được dự phòng. Tuy nhiên, phân tích phần mềm thời gian thực bằng cách mô phỏng hoá không chỉ giúp kỹ sư phần mềm phân tích hiệu năng của hệ thống mà còn có khả năng xây dựng bản mẫu, thực hiện nó và do đó hiểu được hành vi của hệ thống.

Việc hiểu biết hành vi của hệ thống trong môi trường qua thời gian hay được đề cập tới, nhất là trong các giai đoạn thiết kế, cài đặt và kiểm thử dự án, qua việc thử sai lầm. Cách tiếp cận Statemate (công cụ kỹ nghệ hệ thống cho phép mô phỏng và mô hình hoá) đưa ra giải pháp cho tiến trình tốn kém này. Nó cho phép xây dựng một mô hình hệ thống dễ hiểu nhưng đủ chính xác. Mô hình này đề cập tới các vấn đề luồng và chức năng thông thường,

nhưng cũng bao quát cả các khía cạnh động, hành vi của hệ thống. Mô hình này có thể được kiểm thử bởi các công cụ phân tích và tìm kiếm Statemante, cung cấp một cơ chế mở rộng để giám định, gỡ lỗi đặc tả và tìm kiếm thông tin. Bằng cách kiểm thử mô hình cài đặt, kỹ sư hệ thống có thể thấy cách thức hệ thống như được đặc tả (nếu được cài đặt).

Cách tiếp cận của Statemate dùng một ký pháp bao gồm tổ hợp 3 cách nhìn khác nhau về hệ thống: Sơ đồ hoạt động, sơ đồ module và sơ đồ trạng thái.

– Cách nhìn quan niệm

Các vấn đề chức năng được xử lý bằng cách dùng các hoạt động biểu thị cho khả năng xử lý của hệ thống. Các hoạt động có thể lồng nhau, tạo nên mọi cấp bậc cần thiết cho sự phân rã chức năng của hệ thống. Các khoản mục thông tin, như khoảng cách tới mục tiêu hay tên khách hàng, sẽ chảy giữa các hoạt động và có thể được giữ trong kho dữ liệu. Cách nhìn chức năng này về hệ thống được thu tóm trong các sơ đồ hoạt động, tương tự như biểu đồ luồng dữ liệu quy ước.

Các vấn đề hành vi động, thường gọi là các khía cạnh điều khiển được giải quyết bằng cách dùng sơ đồ trạng thái. Tại đây, trạng thái có thể lồng nhau và nối theo một số cách để biểu diễn hành vi tuần tự hay tương tranh. Việc chuyển giữa những trạng thái này có thể bố trí theo sự kiện, chúng được định tính theo các điều kiện.

Hai cách nhìn này về hệ thống được tích hợp lại theo cách sau : Được liên kết với từng mức của sơ đồ hoạt động, thông thường sẽ là một sơ đồ trạng thái, gọi là hoạt động điều khiển, mà vai trò của nó là điều khiển các hoạt động và luồng dữ liệu của mức đó. Một sơ đồ trạng thái có thể thực thi điều khiển trên các hoạt động. Chẳng hạn, nó chỉ thị cho các hoạt động bắt đầu hay dừng lại, tạm ngừng hay chạy tiếp công việc. Nó thay đổi các giá trị của biến và do vậy, ảnh hưởng tới việc xử lý do các hoạt động đó tiến hành. Nó cũng có thể gửi các tín hiệu tới các hoạt động khác làm thay đổi hành vi riêng của chúng. Bên cạnh việc sinh ra các hoạt động thì một sơ đồ trạng thái kiểm soát cũng nhận ra các hoạt động của các sơ đồ trạng thái khác tiến hành. Ví dụ, nếu một sơ đồ trạng thái bắt đầu một hoạt động hay tăng giá trị của một biến thì một sơ đồ trạng thái khác cảm thấy sự kiện đó và dùng nó, chẳng hạn, bố trí một việc chuyển.

Điều quan trọng cần hiểu rằng các sơ đồ hoạt động và sơ đồ trạng thái có liên hệ chặt chẽ với nhau, nhưng chúng không phải là các biểu diễn khác nhau của cùng một vấn đề. Sơ đồ hoạt động là không đầy đủ khi xét như mô hình hệ thống, vì chúng không đề cập đến hành vi. Sơ đồ trạng thái cũng không đầy đủ vì không có các hoạt động thì chúng chẳng có gì để điều khiển cả. Sơ đồ hoạt động chi tiết và sơ đồ trạng thái điều khiển đưa ra mô hình khái niệm. Sơ đồ hoạt động là xương sống của mô hình; việc phân rã các khả năng của hệ thống trong sơ đồ này sẽ thống trị trong đặc tả, trong khi sơ đồ trạng thái của nó lại là lực điều khiển đằng sau hành vi của hệ thống.

– Cách nhìn vật lý

Một đặc tả dùng sơ đồ hoạt động và sơ đồ trạng thái dưới dạng mô hình quan niệm là một nền tảng rất tốt, nhưng nó lại không phải là hệ thống thực vì bỏ mất phương tiện để mô tả hệ thống từ khía cạnh vật lý (cài đặt) và một phương tiện để chắc chắn hệ thống được cài đặt theo đúng đặc tả đó. Phần quan trọng là mô tả việc phân rã vật lý của hệ thống và mối quan hệ của nó với mô hình khái niệm.

Các khía cạnh vật lý được giải quyết trong Statemate bằng cách dùng ngôn ngữ sơ đồ module. Thuật ngữ vật lý hay module được dùng một cách tổng quát để chỉ các thành phần của hệ thống, dù đó là phần cứng hay phần mềm. Giống như các hoạt động trong sơ đồ, các module được bố trí theo thứ bậc để chỉ ra việc phân rã của hệ thống thành các thành phần của nó và các thành phần con. Các module được nối bởi các đường luồng chảy truyền thông tin giữa các module.

– Phân tích và mô phỏng

Khi đã xây dựng được mô hình khái niệm, bao gồm cả sơ đồ hoạt động và sơ đồ trạng thái điều khiển, ta có thể phân tích và kiểm thử kỹ. Mô hình này có thể mô tả toàn bộ hệ thống đến mức chi tiết thấp nhất, hay nó có thể chỉ là một đặc tả bộ phận.

Trước hết, chúng ta phải chắc chắn rằng mô hình này đúng đắn về cú pháp. Điều này làm nảy sinh nhiều phép kiểm thử tương đối trực tiếp; chẳng hạn, kiểm thử nhiều sơ đồ không đầy đủ (như thiếu nhãn hay tên,...); kiểm thử các định nghĩa của các phần tử không đồ họa như các sự kiện và điều kiện... Việc kiểm tra cú pháp cũng bao gồm nhiều phép kiểm thử tĩnh

vi như tính đúng đắn của đầu vào và đầu ra. Một ví dụ cho việc này là kiểm thử các phần tử được dùng trong sơ đồ trạng thái, nó không đơn giản như sự kiện bất nguồn mà kiểm tra trạng thái thay đổi như thế nào. Tất cả những điều này thường được nói tới như các phép kiểm thử tính nhất quán và đầy đủ và phần lớn chúng đều tương tự với việc kiểm tra được thực thi bởi trình biên dịch.

– *Kịch bản chạy*

Một mô hình đúng đắn về cú pháp mô tả chính xác một hệ thống nào đó. Tuy nhiên, nó có thể không phải là hệ thống ta đã định hướng từ trước. Trong thực tế, hệ thống đã mô tả có thể còn nhược điểm nghiêm trọng – việc đúng cú pháp không đảm bảo tính đúng đắn cho chức năng hành vi. Mục tiêu thực của việc phân tích mô hình là để tìm ra liệu nó có mô tả đúng hệ thống ta muốn hay không. Việc phân tích giúp chúng ta biết được nhiều hơn về mô hình đã được xây dựng, qua đó xem xét lại cách thức hệ thống sẽ hành xử để kiểm chứng rằng nó quả thực đáp ứng sự mong đợi. Điều này đòi hỏi một ngôn ngữ mô hình hoá với nhiều cú pháp hình thức hơn; đòi hỏi hệ thống được dùng tạo ra mô hình nhận dạng được cả ngữ nghĩa hình thức.

Nếu mô hình dựa trên ngữ nghĩa hình thức thì kỹ sư hệ thống có thể thực hiện mô hình đó. Người kỹ sư có thể tạo ra và cho chạy một kịch bản để qua đó quan sát hành vi của mô hình trước khi hệ thống thực tế được xây dựng. Chẳng hạn, để thực hiện một mô hình về máy trả tiền tự động ATM các bước sau sẽ được khởi tạo:

1. Mô hình quan niệm được tạo ra.
2. Kỹ sư đóng vai trò khách hàng và máy tính được coi như ngân hàng sinh ra một sự kiện ví dụ đưa vào một thẻ ngân hàng, nhấn các phím và thông tin số dư ngân hàng mới đưa ra.
3. Phản ứng của hệ thống đối với những sự kiện được điều phối.
4. Tính không nhất quán trong hành vi được ghi lại.
5. Mô hình quan niệm được sửa đổi để phản ánh hành vi đúng.
6. Lập đi lập lại điều này cho tới khi hệ thống tiến hoá tới chỗ mong muốn.

Kỹ sư hệ thống cho chạy kịch bản và xem đáp ứng của hệ thống về mặt đồ họa. Các phần tử “tích cực” của mô hình (như trạng thái hệ thống đang ở vào thời điểm đó và các hoạt động đang được kích hoạt) đều được chiếu sáng về đồ họa và việc thực hiện nảy sinh trong một cách biểu diễn hoạt hình của mô hình. Việc thực hiện kịch bản mô phỏng hệ thống chạy trong thời gian thực và lưu giữ dấu vết về thông tin phụ thuộc thời gian. Tại bất kỳ điểm nào trong khi thực hiện, người kỹ sư hệ thống cũng đều xem trạng thái của bất kỳ phần tử không đồ họa khác, như giá trị của biến hay một điều kiện.

– Mô phỏng lập trình

Kịch bản cho phép kỹ sư hệ thống tập mô phỏng theo mô hình bằng cách tương tác. Tuy nhiên, nhiều khi người ta mong muốn có một sự mô phỏng mạnh mẽ hơn. Hiệu năng dưới điều kiện ngẫu nhiên trong cả hai tình huống diễn hình cần phải được đánh giá lại. Với những tình huống cần tới việc mô phỏng mô hình thời gian thực thì ngôn ngữ điều khiển mô phỏng (SCL) giúp người kỹ sư giữ điều khiển chung về cách xử lý thực hiện, đồng thời khai thác được sức mạnh của công cụ để chuyển giao nhiều chi tiết.

Một trong những điều đơn giản nhất làm được với SCL là đọc danh sách các sự kiện từ một tệp theo lô. Điều này có nghĩa là có thể chuẩn bị trước các kịch bản dài hay một phần của chúng rồi cho chạy tự động. Kỹ sư quan sát những kịch bản này. Theo cách khác, người kỹ sư hệ thống lập trình bằng SCL để đặt các điểm đứt và kiểm soát các biến, trạng thái hay điều kiện nào đó.

Việc dùng kịch bản mô phỏng cũng cho phép thu thập được những thông kê có ý nghĩa về sự vận hành của hệ thống cần phải xây dựng.

– Dịch tự động thành mã

Khi mô hình hệ thống đã được xây dựng, nó có thể dịch toàn bộ thành mã chạy được bằng cách dùng chức năng tạo bản mẫu. Các sơ đồ hoạt động và sơ đồ trạng thái điều khiển của chúng được dịch theo ngôn ngữ lập trình cấp cao, như ADA hay C. Ngày nay, việc dùng chủ yếu mã kết quả để quan sát một hệ thống thực hiện trong những hoàn cảnh gần với thế giới thực nhất. Chẳng hạn, mã bản mẫu được thực hiện theo bộ mô phỏng thật đầy đủ cho môi trường đích hay trong bản thân môi trường cuối cùng. Mã được các công cụ CASE tạo ra nên được coi như “có tính bản mẫu”. Đó chưa

phải là mã sản xuất hay cuối cùng. Do vậy, nó có thể phản ánh hiệu năng thời gian thực chính xác của hệ thống dự định. Tuy nhiên, việc kiểm thử hiệu năng của hệ thống gắn với hoàn cảnh thực nhất bao giờ cũng có ích.

6.4.4. Phương pháp thiết kế

Việc thiết kế phần mềm thời gian thực phải bao gồm tất cả các khái niệm nền tảng liên quan đến phần mềm chất lượng cao. Bên cạnh đó, phần mềm thời gian thực còn phải đặt ra một tập các vấn đề cho người thiết kế:

- Biểu diễn các ngắt và việc chuyển hoàn cảnh.
- Tương tranh như được thể hiện trong đa nhiệm và đa xử lý.
- Truyền thông và đồng bộ giữa các nhiệm vụ.
- Độ biến thiên lớn trong tỷ lệ dữ liệu và truyền thông.
- Biểu diễn các ràng buộc thời gian.
- Xử lý không đồng bộ.
- Việc móc nối không tránh khỏi và cần thiết với hệ điều hành, phần cứng và các phần tử hệ thống ngoài khác.

Khi thiết kế phần mềm thời gian thực phải chú ý đến các nguyên tắc mô hình hoá sau đây:

- **Tính nguyên tố:** Cần phải xác định rõ ràng các hành động nguyên tố như là một phần của mô hình thiết kế thời gian thực. Một hành động nguyên tố hay một sự kiện là một chức năng ràng buộc tốt và có giới hạn có thể thực thi bằng một công việc đơn lẻ hoặc thực thi đồng thời bằng nhiều công việc. Một hành động nguyên tố chỉ được gọi khi các công việc này yêu cầu và kết quả của nó chỉ ảnh hưởng đến các phần này, không ảnh hưởng đến các phần khác của hệ thống.
- **Xen kê:** Mặc dù các quá trình có thể đồng thời, nhưng một số tính toán có thể thu được qua dãy các hành động tuần tự. Bắt đầu với một trạng thái khởi đầu, hành động đầu tiên được nhúng và được thực hiện. Kết quả là, trạng thái bị thay đổi và hành động thứ hai xảy ra. Do nhiều hành động có thể xảy ra để đưa đến một trạng thái, các kết quả khác nhau có thể thu được từ cùng một trạng thái khởi đầu.

- **Nguyên tắc hệ thống đóng:** Một mô hình thiết kế thời gian thực có thể chứa phần mềm và môi trường của phần mềm. Do đó, các hành động có thể chia nhỏ thành các phần để hệ thống tự bản thân nó đáp ứng, hoặc các phần được thực thi bởi môi trường.
- **Cấu trúc của trạng thái :** Hệ thống thời gian thực có thể mô hình như một tập các đối tượng, mỗi đối tượng có một trạng thái của riêng nó.

6.4.5. Phương pháp thiết kế hướng luồng dữ liệu

Phương pháp thiết kế hướng luồng dữ liệu được dùng rộng rãi nhất trong công nghiệp.

6.4.5.1. Các yêu cầu về phương pháp thiết kế hệ thống thời gian thực

Các kỹ thuật thiết kế hướng luồng dữ liệu đưa ra nền tảng đánh giá cho thiết kế thời gian thực. Các biểu đồ luồng dữ liệu mô tả luồng thông tin giữa các chức năng hệ thống và các kỹ thuật ánh xạ giúp cho người thiết kế có thể suy dẫn ra các cấu trúc chương trình từ các đặc trưng luồng dữ liệu.

Phương pháp thiết kế phần mềm thời gian thực DARTS xây dựng trên ký pháp và cách tiếp cận cho hệ thống thiết kế hướng luồng dữ liệu của phần mềm quy ước. Để hỗ trợ cho thiết kế thời gian thực, các phương pháp hướng luồng dữ liệu phải được mở rộng bằng cách cung cấp:

- Một cơ chế để biểu diễn việc truyền thông và đồng bộ hoá vấn đề.
- Một ký pháp để biểu diễn sự phụ thuộc trạng thái.
- Một cách tiếp cận nối các phương pháp luồng dữ liệu quy ước với thế giới thời gian thực.

Vì phần lớn các hệ thống thời gian thực sinh ra nhiều nhiệm vụ hoặc cùng dùng chung một bộ xử lý hay thực hiện đồng thời các bộ xử lý phân tán nên những cơ chế này phải có được khả năng đồng bộ hoá các nhiệm vụ và đưa ra việc truyền thông giữa các nhiệm vụ này. Việc đồng bộ hoá xảy ra thông qua việc loại trừ lẫn nhau hay kích thích chéo. Loại trừ lẫn nhau được áp dụng khi hai nhiệm vụ đồng thời thâm nhập vào một vùng dữ liệu chung. Cờ dữ liệu được dùng để loại trừ nhiệm vụ này khỏi việc thâm nhập dữ liệu trong khi nhiệm vụ khác đang dùng (đọc hoặc ghi) cùng dữ liệu đó.

Kích thích chéo được cài đặt khi nhiệm vụ này báo hiệu cho nhiệm vụ khác (đang đợi) rằng nó đã hoàn thành một hoạt động nào đó và có thể tiến hành nhiệm vụ được báo.

Việc truyền thông nhiệm vụ xuất hiện khi nhiệm vụ này phải truyền thông cho nhiệm vụ khác. Liên lạc thông báo là cách tiếp cận thông thường. Khi nhiệm vụ sản xuất gửi một thông báo cho nhiệm vụ tiêu thụ, sau đó đợi sự đáp ứng từ nhiệm vụ tiêu thụ, thì việc liên lạc được gọi là gắn chặt. Khi các nhiệm vụ sản xuất và tiêu thụ tiếp tục theo xử lý riêng của chúng và dùng hàng đợi thông báo để làm bộ đệm thông báo thì việc liên lạc được gọi là gắn lỏng.

Để cài đặt việc liên lạc thông báo, Gomma mô tả 3 cách tiếp cận khác nhau:

- Hệ điều hành thời gian thực có thể cung cấp các nguyên sơ liên lạc giữa các nhiệm vụ.
- Cơ chế liên lạc giữa các nhiệm vụ có thể được cài đặt bên trong ngữ cảnh ngôn ngữ lập trình (như Ada).
- Một bộ khiên giải (handler) liên lạc đặc biệt được tạo ra bằng cách dùng các nguyên sơ đồng bộ hoá do hệ điều hành cung cấp.

Cách tiếp cận DARTS cung cấp một kỹ pháp hỗ trợ cho cả liên lạc gắn chặt và gắn lỏng.

6.4.5.2. DARTS

Phương pháp thiết kế DARTS bắt đầu với việc áp dụng các nguyên tắc phân tích phần mềm nền tảng (như phân tích lĩnh vực thông tin, phân hoạch vấn đề) được áp dụng trong ngữ cảnh kỹ pháp luồng dữ liệu. Các biểu đồ luồng dữ liệu được tạo ra, từ điển dữ liệu tương ứng được định nghĩa và giao diện giữa các chức năng hệ thống chính được thiết lập.

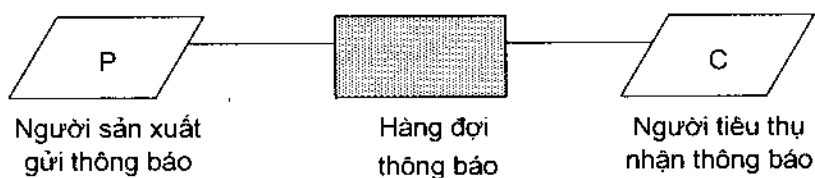
Khi mô hình luồng dữ liệu được tạo ra thì hệ thống phải được xem xét theo một quan điểm khác. DFD không mô tả các nhiệm vụ không đồng bộ và tương tranh, cái vẫn cài đặt cho luồng dữ liệu. Do đó, chúng ta cần một cách tiếp cận xác định ra các nhiệm vụ hệ thống thời gian thực trong hoàn cảnh của một chức năng (phép biến đổi) hệ thống được rút ra từ DFD. Các phép biến đổi được gộp thành nhóm các nhiệm vụ thời gian thực. Luồng dữ liệu giữa các nhiệm vụ mới được xác định sẽ định ra các yêu cầu liên lạc

giữa các nhiệm vụ. Gomma đưa ra tiêu chuẩn sau đây để xác định xem liệu các phép biến đổi DFD có nên được xác định như các nhiệm vụ tách biệt hay gộp nhóm với các phép biến đổi khác thành một nhiệm vụ:

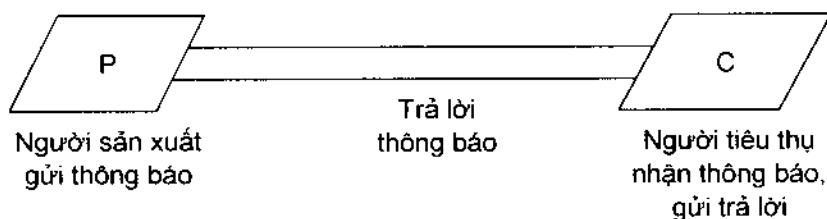
- *Phụ thuộc vào vào/ra*: Phụ thuộc vào đầu vào hay đầu ra, một phép biến đổi thường bị ràng buộc phải chạy với tốc độ của các thiết bị vào/ra chỉ phối khi nó thực hiện tương tác. Trong trường hợp này, phép biến đổi phải là một nhiệm vụ tách biệt.
- *Chức năng căng về thời gian*: Một chức năng căng về thời gian phải chạy với mức ưu tiên cao hơn và do đó phải là một nhiệm vụ tách biệt.
- *Yêu cầu tính toán*: Chức năng đòi hỏi tính toán nhiều (hay tập các chức năng) có thể chạy như một nhiệm vụ với số ưu tiên thấp – tiêu thụ chu kỳ CPU dự phòng.
- *Tính cố kết chức năng*: Những phép biến đổi thực hiện một tập các chức năng có liên quan chặt chẽ có thể gộp nhóm với nhau trong một nhiệm vụ. Vì lưu lượng dữ liệu giữa các chức năng này có thể cao nên việc để chúng là các nhiệm vụ tách biệt sẽ làm tăng tổng phí hệ thống, trong khi việc cài đặt cho từng chức năng như các module tách biệt bên trong cùng nhiệm vụ sẽ đảm bảo tính cố kết chức năng tại mức module lẫn mức nhiệm vụ.
- *Tính cố kết thời gian*: Một số phép biến đổi thực hiện các chức năng được tiến hành đồng thời. Những chức năng này có thể gộp nhóm thành một nhiệm vụ sao cho chúng được thực hiện mỗi lần nhiệm vụ nhận được kích thích.
- *Thực hiện theo định kỳ*: Một phép biến đổi cần được thực hiện định kỳ cũng có thể được cấu trúc như một nhiệm vụ tách biệt và được kích hoạt theo từng khoảng đều đặn.

Khi các nhiệm vụ được xác định thì DARST cung cấp một cơ chế để giải quyết liên lạc giữa các nhiệm vụ bằng cách xác định ra hai lớp module giao diện nhiệm vụ: module liên lạc nhiệm vụ (TCM) và module đồng hoá nhiệm vụ (TSM). Một TCM được sinh ra bởi nhiệm vụ liên lạc và dùng các nguyên sơ đồng bộ hoá của hệ điều hành để đảm bảo việc thâm nhập đúng đắn vào dữ liệu. TCM được chia thành hai phạm trù:

- *Module liên lạc thông báo (MCM)* hỗ trợ cho việc liên lạc thông báo và cài đặt các cơ chế thích hợp để quản lý hàng đợi thông báo (đối với liên lạc gắn lỏng) và các nguyên sơ đồng bộ hoá (đối với các liên lạc gắn chặt). Ký pháp DARST cho việc liên lạc giữa các nhiệm vụ được cài đặt bởi MCM được minh hoạ như trong hình 6.8a và 6.8b.

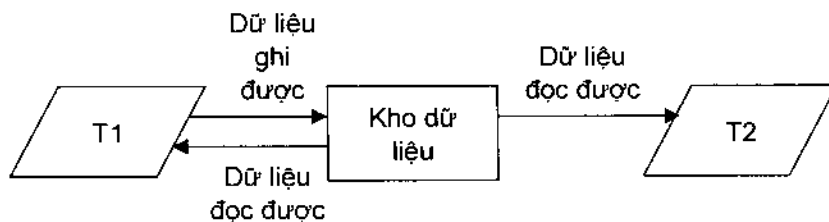


Hình 6.8a. Gắn lỏng (Module liên lạc thông báo)



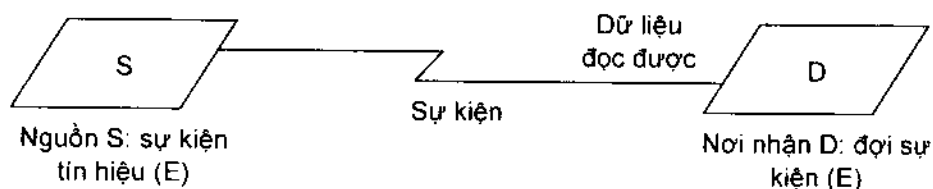
Hình 6.8b. Gắn chặt (Module liên lạc thông báo)

- Các module che dấu thông tin (IHM) cung cấp việc thâm nhập vào nhóm dữ liệu hay kho dữ liệu. IHM cài đặt cấu trúc dữ liệu cũng như phương pháp thâm nhập cho phép các nhiệm vụ khác thâm nhập vào cấu trúc dữ liệu. Hình 6.9 minh hoạ cho ký pháp DARTS đối với việc liên lạc được cài đặt bởi IHM.



Hình 6.9. Module che dấu thông tin

Khi điều khiển mà không phải dữ liệu được truyền giữa các nhiệm vụ thì module đồng bộ hoá nhiệm vụ bắt đầu làm việc. Một nhiệm vụ có thể báo cho một nhiệm vụ khác rằng một sự kiện đã xuất hiện hay một nhiệm vụ có thể đợi một tín hiệu như vậy. Ký pháp DARTS cho việc đồng bộ hoá nhiệm vụ được mô tả trong hình 6.10.



Hình 6.10. Đồng bộ hoá nhiệm vụ

Một TSM được coi như một module giám sát, quản lý điều khiển và điều phối các sự kiện xuất hiện.

Chương 7

KIỂM THỬ

7.1. Khái niệm kiểm thử

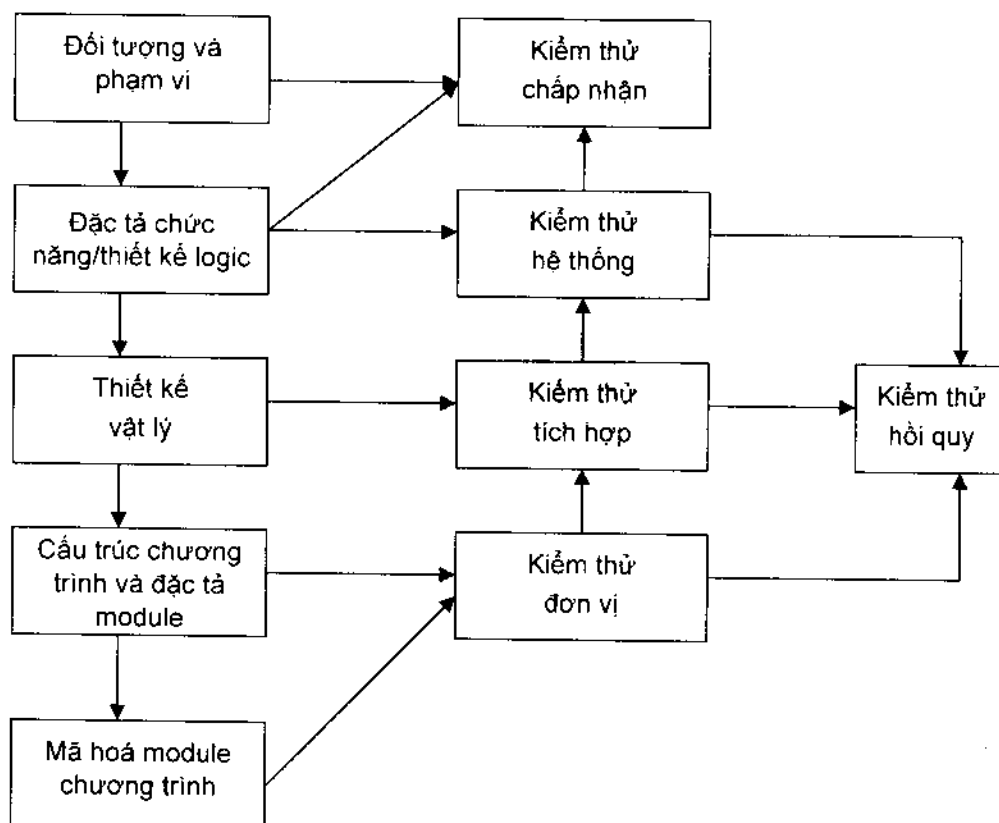
Việc phát triển hệ thống phần mềm liên quan tới hàng loạt các hoạt động sản xuất, nơi con người có thể mắc sai sót. Lỗi có thể xuất hiện ngay từ khởi đầu của tiến trình – nơi các mục tiêu bị xác định sai hay không hoàn chỉnh, cũng có thể xuất hiện trong giai đoạn thiết kế và phát triển về sau. Vì con người không có khả năng thực hiện và trao đổi một cách hoàn chỉnh nên việc phát triển phần mềm cần đi kèm với hoạt động đảm bảo chất lượng.

Kiểm thử phần mềm là phần tử mấu chốt của đảm bảo chất lượng phần mềm và biểu thị cho việc xét duyệt quan trọng về đặc tả, thiết kế và mã hoá. Kiểm thử thành công là kiểm thử phát hiện ra lỗi, kiểm thử không thành công là kiểm thử không tìm ra lỗi. Do đó, mục đích của chúng ta là thiết kế kiểm thử để phát hiện được một cách có hệ thống các lớp lỗi khác nhau với một thời gian và công sức tối thiểu. Tuy nhiên, khi kiểm thử cần phải chú ý đến 6 điều sau:

- Chất lượng phần mềm do khâu thiết kế quyết định là chủ yếu, mà không phải là khâu kiểm thử.
- Tính dễ kiểm thử phụ thuộc vào cấu trúc chương trình.
- Người kiểm thử và người phát triển nên khác nhau.
- Dữ liệu kiểm thử cho kết quả bình thường thì không có ý nghĩa nhiều, cần có dữ liệu kiểm thử phát hiện ra lỗi.
- Khi thiết kế trường hợp kiểm thử, không chỉ có dữ liệu thử nhập vào mà còn thiết kế cả dữ liệu kết quả sẽ có.

- Khi phát sinh thêm trường hợp thử thì nên thử lại những trường hợp thử trước đó để tránh ảnh hưởng lan truyền.

Tương ứng giữa vòng đời dự án và kiểm thử được minh họa trên hình 7.1.



Hình 7.1. Tương ứng giữa vòng đời dự án và kiểm thử

7.2. Các nguyên tắc kiểm thử

Trước khi bắt đầu bất cứ hoạt động kiểm thử nào, một kỹ sư phần mềm cũng phải biết các nguyên tắc cơ bản sau trong kiểm thử:

- Mọi hoạt động kiểm thử đều phải tuân theo yêu cầu của khách hàng. Do mục đích của kiểm thử là tìm ra lỗi nên phần lớn các lỗi (từ quan điểm của khách hàng) sẽ làm cho chương trình không đáp ứng được yêu cầu của khách hàng.

- Kiểm thử phải được lập kế hoạch trước khi tiến hành: Lập kế hoạch kiểm thử có thể bắt đầu ngay khi hoàn thành xong các mô hình yêu cầu. Xác định chi tiết các trường hợp thử có thể bắt đầu khi các mô hình thiết kế được hoàn thiện. Do đó, mọi kiểm thử được lập kế hoạch và thiết kế trước khi xây dựng mã chương trình.
- Kiểm thử ban đầu ít, sau đó quá trình lớn dần. Kiểm thử đầu tiên được lập kế hoạch và thực hiện thường tập trung vào các module chương trình riêng lẻ. Trong tiến trình kiểm thử, chú trọng vào việc tìm ra lỗi trong các module tích hợp và cuối cùng là của toàn bộ hệ thống.
- Không thể kiểm thử mọi khía cạnh: Có nhiều cách kiểm thử cho các chương trình, thậm chí cả với các chương trình có kích thước vừa phải. Vì lẽ đó, không thể thực hiện mọi cách trong kiểm thử.
- Đề kiểm thử đạt hiệu quả cao nhất, trong quá trình kiểm thử phải có sự tham gia của bên thứ 3. Kỹ sư phần mềm tạo ra hệ thống không phải là người tốt nhất để thực hiện mọi kiểm thử cho phần mềm.

7.3. Phương pháp thiết kế trường hợp kiểm thử

Thiết kế kiểm thử cho phần mềm và các sản phẩm công nghệ có tính thách đố giống như việc thiết kế sản phẩm ban đầu. Chúng ta cần thiết kế các trường hợp kiểm thử đạt hiệu quả cao nhất để tìm ra nhiều lỗi với thời gian và chi phí tối thiểu.

Trong suốt thập kỷ 1995 – 2005, có rất nhiều phương pháp thiết kế trường hợp kiểm thử phần mềm đã phát triển. Những phương pháp này cung cấp cho người phát triển một cách tiếp cận hệ thống tới việc kiểm thử. Quan trọng hơn, những phương pháp này đưa ra một cơ chế giúp đảm bảo tính đầy đủ của kiểm thử và đưa ra khả năng phát hiện lỗi cao nhất trong phần mềm.

Bất cứ sản phẩm công nghệ nào đều có thể kiểm thử theo 2 cách:

- Khi biết chức năng xác định của một sản phẩm đã được thiết kế thì có thể tiến hành kiểm thử xem từng chức năng có vận hành hoàn hảo không.

- Biết cách làm việc bên trong của một sản phẩm, tiến hành kiểm thử để đảm bảo rằng tất cả “các bánh răng ăn khớp vào nhau” tức là đảm bảo sự vận hành bên trong thực hiện tương ứng với đặc tả và tất cả các thành phần bên trong đã được thử một cách thích hợp.

Cách tiếp cận thứ nhất được gọi là kiểm thử hộp đen, cách tiếp cận thứ hai gọi là kiểm thử hộp trắng.

Khi xem xét một phần mềm máy tính thì kiểm thử hộp đen ám chỉ việc kiểm thử được tiến hành tại giao diện phần mềm. Mặc dù chúng được thiết kế để phát hiện ra lỗi, kiểm thử hộp đen được dùng để thể hiện rằng các chức năng phần mềm đều vận hành; rằng đầu vào được chấp nhận là đúng, đầu ra được tạo ra đúng; tính toàn vẹn của thông tin ngoài (như các tệp dữ liệu) được duy trì. Phép kiểm thử hộp đen xem xét một số khía cạnh của hệ thống mà ít để ý đến cấu trúc logic bên trong của phần mềm.

Việc thiết kế kiểm thử hộp trắng cho phần mềm được xác định dựa trên việc xem xét kỹ về chi tiết thủ tục. Các đường logic đi qua phần mềm được kiểm thử bằng cách đưa ra các trường hợp kiểm thử, vốn thực hiện trên một tập các điều kiện và/hoặc chu trình. Trạng thái của chương trình được xem xét tại nhiều điểm khác nhau để xác định liệu trạng thái được trông đợi hay khẳng định có tương ứng với trạng thái hiện tại hay không.

7.3.1. Kiểm thử hộp trắng

Kiểm thử hộp trắng là phương pháp thiết kế trường hợp kiểm thử có dùng cấu trúc điều khiển của thiết kế thủ tục để suy ra các trường hợp kiểm thử. Bằng việc dùng các phương pháp kiểm thử hộp trắng, kỹ sư phần mềm có thể suy dẫn ra các trường hợp kiểm thử mà:

- Đảm bảo rằng mọi con đường độc lập bên trong một module đều được thực hiện ít nhất một lần.
- Thực hiện tất cả các quyết định logic trên nhánh đúng và sai.
- Thực hiện tất cả các chu trình tại biên giới của chúng và bên trong giới hạn chức năng.
- Thực hiện các cấu trúc thủ tục dữ liệu bên trong để đảm bảo tính hợp lệ của chúng.

Một câu hỏi hợp lý được đặt ra là: Tại sao phải tốn thời gian và năng lực để kiểm thử những chi tiết vụn vặt khi mà ta có thể mở rộng hơn nỗ lực để chứng minh rằng các yêu cầu của chương trình đã được đáp ứng. Nói cách khác, tại sao ta không dành năng lực vào phép kiểm thử hộp đen. Câu trả lời nằm ở bản chất phần mềm:

- Các lỗi logic và các giả thiết không đúng tỷ lệ nghịch với xác suất một đường chương trình sẽ thực hiện. Lỗi có khuynh hướng nảy sinh khi chúng ta thiết kế và thực hiện các hàm, điều kiện hay điều khiển nằm ngoài luồng mạch chính. Việc xử lý hàng ngày có khuynh hướng được xem xét kỹ lưỡng trong khi việc xử lý trường hợp đặc biệt có khuynh hướng rơi vào kẽ hở.
- Chúng ta thường cho rằng, một đường logic thì không thể được thực hiện nhưng trong thực tế nó có thể được thực hiện đều đặn. Luồng logic của một chương trình đôi khi khác thường, nghĩa là những giả thiết của chúng ta về luồng điều khiển và dữ liệu có thể dẫn đến các lỗi thiết kế và chỉ phát hiện được khi kiểm thử đường được tiến hành.
- Lỗi gõ vào là ngẫu nhiên: Khi một chương trình được dịch sang chương trình gốc trong ngôn ngữ lập trình thì có thể một số lỗi gõ máy xuất hiện. Nhiều lỗi sẽ được cơ chế kiểm tra cú pháp phát hiện, nhưng những lỗi khác sẽ không bị phát hiện cho đến khi việc kiểm thử bắt đầu. Rất có thể là một lỗi gõ sai sẽ tồn tại trên một đường logic chưa rõ như trên một đường đi của luồng mạch.

Mỗi lý do đều đưa ra luận cứ để tiến hành các kiểm thử hộp trắng. Việc kiểm thử hộp đen, dù tiến hành kỹ lưỡng đến đâu vẫn có thể bỏ lỡ các lỗi trên.

7.3.2. Kiểm thử hộp đen

Các phương pháp kiểm thử hộp đen tập trung vào các yêu cầu chức năng của phần mềm. Tức là, qua kiểm thử kỹ sư phần mềm suy ra được tập các điều kiện vào sẽ thao diễn qua tất cả các yêu cầu chức năng đối với một chương trình. Việc kiểm thử hộp đen không phải là phương án kỹ thuật của kiểm thử hộp trắng. Thay vì thế, nó là cách tiếp cận bổ sung có thể phát hiện ra lớp lỗi khác hơn là kỹ thuật hộp trắng.

Việc kiểm thử hộp đen có thể tìm ra lỗi trong các phạm vi sau đây:

- Các chức năng không đúng hay bị bỏ sót.
- Lỗi giao diện.
- Lỗi trong cấu trúc dữ liệu hay thâm nhập cơ sở dữ liệu ngoài.
- Lỗi hiệu năng.
- Lỗi khởi đầu và kết thúc.

Không giống như kiểm thử hộp trắng được thực hiện sớm trong tiến trình kiểm thử, kiểm thử hộp đen có khuynh hướng được áp dụng trong các giai đoạn sau của kiểm thử. Vì kiểm thử hộp đen chủ ý không xem xét đến cấu trúc điều khiển nên sự chú ý được dồn vào miền thông tin. Các phép kiểm thử được thiết kế để trả lời các câu hỏi sau:

- Tính hợp lệ chức năng được kiểm thử khi nào?
- Lớp đầu vào nào sẽ tạo ra những trường hợp kiểm thử tốt?
- Hệ thống có nhạy cảm với những giá trị nào đó không?
- Biên giới của các lớp dữ liệu được cô lập như thế nào?
- Hệ thống có thể dung sai cho tỷ lệ dữ liệu và khối lượng dữ liệu nào?
- Việc tổ hợp dữ liệu đặc biệt có tác động gì đến sự vận hành của hệ thống?

Bằng cách áp dụng các kỹ thuật hộp đen ta suy ra một tập các trường hợp kiểm thử thoả mãn các tiêu chuẩn sau:

- Các trường hợp kiểm thử làm rút gọn, theo số đếm lớn hơn 1, số các trường hợp kiểm thử phụ phải thiết kế để đạt tới việc kiểm thử hợp lý.
- Các trường hợp kiểm thử cho ta biết điều gì đó về sự hiện diện hay vắng mặt của các lớp lỗi hơn là một lỗi liên kết chỉ với một tập xác định hiện có.

7.3.2.1. Phân hoạch tương đương

Phân hoạch tương đương là một phương pháp kiểm thử hộp đen phân chia miền đầu vào của một chương trình thành các lớp dữ liệu và từ đó có thể dẫn ra các trường hợp kiểm thử. Một trường hợp kiểm thử lý tưởng đơn sẽ làm lộ ra một lớp lỗi như xử lý sai các dữ liệu ký tự mà nếu không có nó

thì có thể phải thực hiện nhiều trường hợp kiểm thử trước khi phát hiện lỗi chung. Phân hoạch tương đương cố gắng xác định ra một trường hợp kiểm thử làm lộ ra một lớp lỗi, do đó làm giảm tổng số các trường hợp kiểm thử cần phải xây dựng.

Thiết kế trường hợp kiểm thử cho phân hoạch tương đương dựa trên đánh giá về các lớp tương đương với một điều kiện vào. Lớp tương đương biểu thị một tập hợp các trạng thái hợp lệ hay không hợp lệ với điều kiện vào. Một cách điển hình, một điều kiện vào hoặc là một giá trị số đặc biệt, một miền giá trị, một tập các giá trị có liên quan hay một điều kiện bun (logic boole). Các lớp giá trị tương đương có thể thực hiện theo nghĩa sau:

- Nếu một điều kiện vào xác định một miền, thì một lớp hợp lệ và hai lớp không hợp lệ được xác định.
- Nếu một điều kiện vào đòi hỏi một giá trị xác định, thì một lớp hợp lệ và hai lớp không hợp lệ được xác định.
- Nếu một điều kiện vào xác định một biên của một tập, thì một lớp hợp lệ và một lớp không hợp lệ được xác định.
- Nếu một điều kiện vào là bun, thì một lớp hợp lệ và một lớp không hợp lệ được xác định.

7.3.2.2. Phân tích giá trị biên

Vì các lý do còn chưa rõ ràng, một số lớn các lỗi có khuynh hướng xuất hiện rời rạc tại biên các miền vào hơn là tại trung tâm. Chính vì lý do đó mà phân tích giá trị biên (BVA) đã được phát triển như một kỹ thuật kiểm thử. Việc phân tích giá trị biên dẫn đến việc lựa chọn các trường hợp kiểm thử thực hiện các giá trị cận.

Phân tích giá trị biên là kỹ thuật thiết kế trường hợp kiểm thử, bổ sung cho phân hoạch tương đương. Thay vì chọn bất kỳ phần tử nào của một lớp tương đương, BVA chọn các trường hợp kiểm thử tại “cạnh” của lớp. Thay vì chỉ tập trung vào các điều kiện vào, BVA suy dẫn các trường hợp kiểm thử từ các miền đầu ra.

Hướng dẫn cho BVA là tương tự về nhiều khía cạnh với hướng dẫn nêu ra cho phân hoạch tương đương:

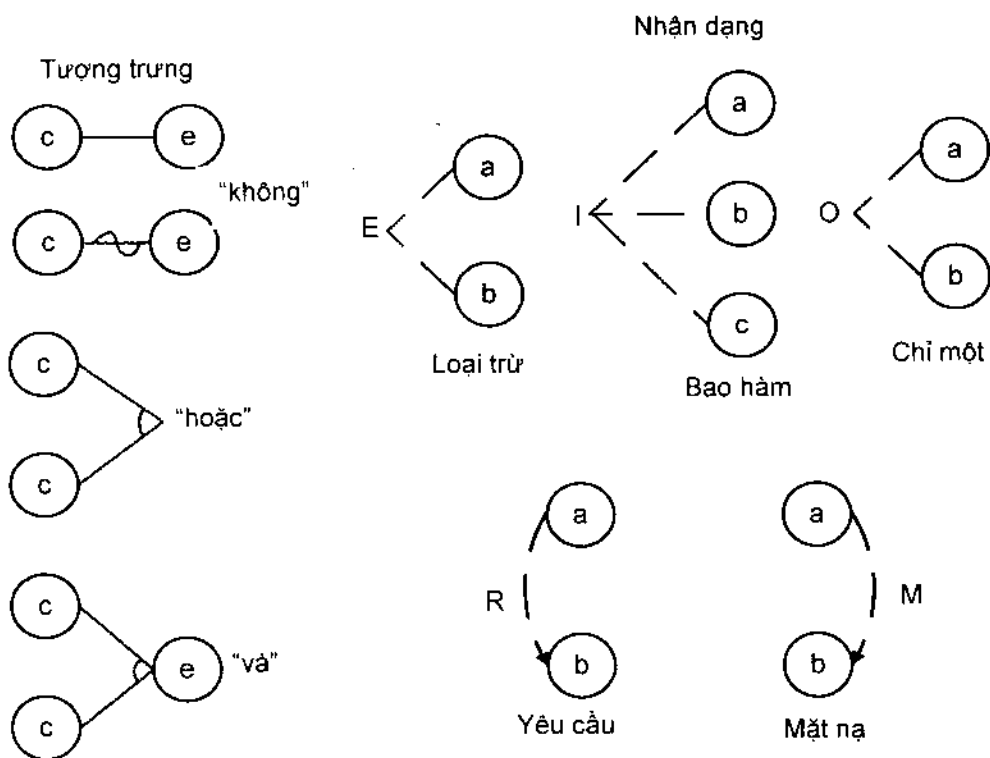
- Nếu một điều kiện vào xác định một miền được giới hạn bởi các giá trị a và b thì các trường hợp kiểm thử nên được thiết kế vào các giá trị a và b, ngay ở trên a và b, ngay ở dưới a và b tương ứng.
- Nếu một điều kiện vào xác định ra một số các giá trị thì các trường hợp kiểm thử nên được xây dựng để thử cho các giá trị cực tiểu và cực đại. Các giá trị ngay ở trên và ngay ở dưới cực đại và cực tiểu cũng được kiểm thử.
- Áp dụng các hướng dẫn trên cho các điều kiện ra. Các trường hợp kiểm thử nên được thiết kế để đưa ra báo cáo làm phát sinh số các ô được phép trong bảng tối đa và tối thiểu.
- Nếu các cấu trúc dữ liệu chương trình bên trong đã mô tả trước các biên thì chắc chắn phải thiết kế một trường hợp kiểm thử để thực hiện cấu trúc dữ liệu đó tại các biên.

Phần lớn các kỹ sư phần mềm thực hiện BVA một cách trực giác ở mức độ nào đó. Bằng việc áp dụng những hướng dẫn đã nêu ở trên, việc kiểm thử sẽ đầy đủ hơn, do đó có nhiều khả năng để phát hiện lỗi hơn.

7.3.2.3. Kỹ thuật đồ thị nhân quả

Đồ thị nhân quả là một kỹ thuật thiết kế trường hợp kiểm thử cung cấp cách biểu diễn chính xác các điều kiện logic và hành động tương ứng. Kỹ thuật này tuân theo bốn bước:

1. Nguyên nhân (điều kiện vào) và hậu quả (hành động) được liệt kê cho một module và một tên được gán cho mỗi chúng. Xây dựng đồ thị nhân quả.
2. Đồ thị được chuyển hoá thành bảng quyết định.
3. Các quy tắc của bảng quyết định được chuyển thành trường hợp kiểm thử.
4. Một bản đơn giản tượng trưng cho đồ thị nhân quả được vẽ như trong hình 7.2. Cột về trái của hình 7.2 minh họa mối quan hệ logic giữa nguyên nhân e_i và hậu quả e_j , ký pháp vẽ đường đứt đoạn ở cột phải chỉ ra các mối quan hệ ràng buộc tiềm năng có thể áp dụng cho hoặc nguyên nhân hoặc kết quả.



Hình 7.2. Tạo đồ thị nhân quả

7.3.2.4. Kiểm thử so sánh

Trong một số tình huống như điều khiển máy bay, điều khiển nhà máy năng lượng hạt nhân yêu cầu độ tin cậy tuyệt đối của phần mềm. Trong những ứng dụng như vậy, phần cứng và phần mềm dư thừa thường được dùng để tối thiểu hoá khả năng lỗi. Khi phần mềm dư thừa được phát triển, nhóm công nghệ phần mềm tách biệt sẽ phát triển những bản độc lập của ứng dụng bằng cách dùng cùng một đặc tả. Trong những tình huống như thế, mỗi bản có thể được kiểm thử với cùng dữ liệu thử để đảm bảo rằng tất cả đều đưa ra đầu ra giống nhau. Vậy tất cả các bản đều được thực hiện song song với việc so sánh thời gian thực về kết quả để đảm bảo tính nhất quán.

Dùng những bài học đã biết từ hệ thống dư thừa, các nhà nghiên cứu đã gợi ý rằng, các bản độc lập của phần mềm phải được phát triển cho các ứng dụng chủ chốt, thậm chí khi chỉ có một bản được dùng trong hệ thống

máy tính. Những bản độc lập này tạo nên cơ sở cho kỹ thuật kiểm thử hộp đen gọi là kiểm thử so sánh hay kiểm thử sau lưng.

Khi có nhiều cài đặt của cùng một đặc tả được tạo ra, các trường hợp kiểm thử được thiết kế để dùng cho các kỹ thuật hộp đen khác (như phân hoạch tương đương) được nêu ra như đầu vào cho từng bản phần mềm. Nếu đầu ra của mỗi bản là giống nhau thì người ta giả thiết rằng tất cả các cài đặt đều đúng. Nếu các đầu ra là khác nhau thì từng ứng dụng sẽ được nghiên cứu lại để xác định liệu một khiếm khuyết trong một hay nhiều bản có phải là nguyên nhân sinh ra sự khác biệt hay không. Trong phần lớn các trường hợp, việc so sánh các đầu ra được thực hiện bằng công cụ tự động.

Việc kiểm thử hộp đen không đơn giản. Nếu đặc tả mà từ đó các bản đã xây dựng bị lỗi thì mọi bản đều phản ánh lỗi đó. Hơn nữa, nếu từng bản độc lập lại tạo ra kết quả giống nhau nhưng không đúng thì việc kiểm thử điều kiện sẽ không phát hiện ra lỗi.

7.4. Các chiến lược kiểm thử

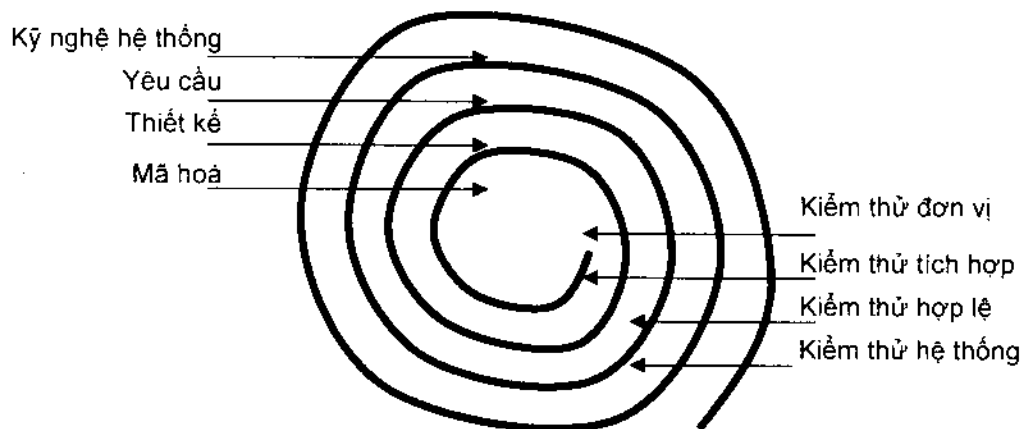
Một chiến lược kiểm thử phần mềm tích hợp các kỹ thuật thiết kế trường hợp kiểm thử bao gồm nhiều bước, được hoạch định chu đáo giúp xây dựng phần mềm thành công. Điều quan trọng là chiến lược kiểm thử phần mềm đưa ra lộ trình cho người phát triển phần mềm, cho tổ chức đảm bảo chất lượng và cho khách hàng một bản lộ trình mô tả các bước cần tiến hành và xác định nỗ lực, thời gian, tài nguyên sẽ cần tới. Do đó, bất kỳ chiến lược kiểm thử nào cũng phải tổ hợp kế hoạch kiểm thử, thiết kế trường hợp kiểm thử, thực hiện kiểm thử và thu thập các đánh giá kết quả.

Một chiến lược kiểm thử phần mềm nên đủ mềm dẻo để thúc đẩy tính sáng tạo và điều chỉnh theo yêu cầu. Đây là điều cần thiết để kiểm thử thích hợp với tất cả các hệ thống phần mềm lớn. Đồng thời, chiến lược này cũng phải đủ chặt chẽ để thúc đẩy việc lập kế hoạch hợp lý và theo dõi việc quản lý khi dự án phát triển.

Người ta đã đưa ra một số chiến lược kiểm thử phần mềm, cung cấp cho người phát triển phần mềm một tiêu bản để kiểm thử và tất cả đều có các đặc trưng sau:

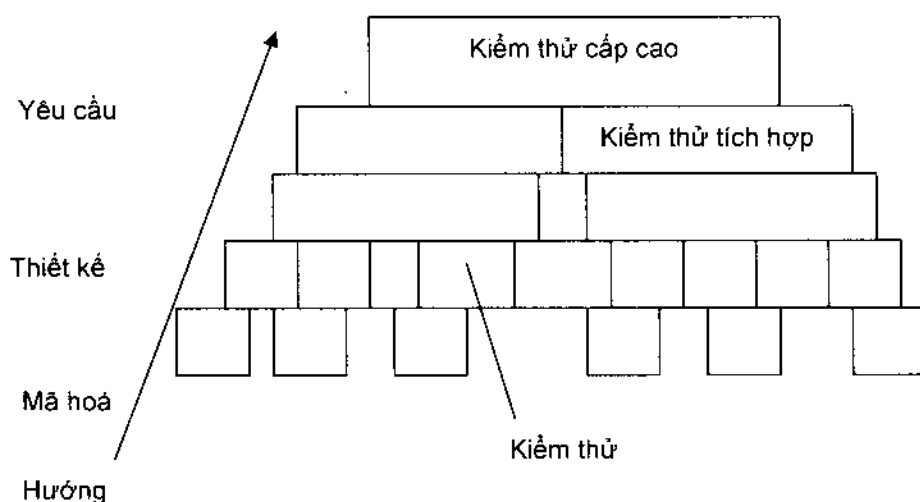
- Việc kiểm thử bắt đầu tại mức module và làm việc “hướng ra ngoài” tới việc tích hợp cho toàn bộ hệ thống dựa trên máy tính.
- Các kỹ thuật kiểm thử khác nhau thích hợp tại các điểm khác nhau theo thời gian.
- Việc kiểm thử được tiến hành bởi người phát triển phần mềm và (với các dự án lớn) một nhóm kiểm thử độc lập.
- Việc kiểm thử và gỡ lỗi là những hoạt động khác nhau, nhưng gỡ lỗi phải phù hợp với bất kỳ chiến lược kiểm thử nào.

Một chiến lược cho kiểm thử phần mềm có thể xem xét thông qua đường xoắn ốc như hình 7.3. Việc kiểm thử đơn vị bắt đầu từ tâm xoắn ốc và tập trung vào các đơn vị của phần mềm được cài đặt trong chương trình gốc. Việc kiểm thử tiến triển bằng cách đi ra theo đường xoắn ốc tới kiểm thử tích hợp tập trung vào thiết kế và xây dựng kiến trúc phần mềm. Đi ra thêm một vòng xoáy trên đường xoắn ốc chúng ta gặp kiểm thử hợp lệ, nơi các yêu cầu được thiết lập như một phần của việc phân tích yêu cầu phần mềm, được hợp lệ hoá theo phần mềm đã xây dựng. Cuối cùng, tới kiểm thử hệ thống, nơi phần mềm và các phần tử hệ thống được kiểm thử toàn bộ. Để kiểm thử phần mềm máy tính, theo đường xoáy chúng ta mở rộng dần phạm vi kiểm thử mỗi lần.



Hình 7.3. Chiến lược kiểm thử

Xem xét tiến trình này theo quan điểm thủ tục thì việc kiểm thử trong phạm vi công nghệ phần mềm thực tại là một chuỗi gồm ba bước được hiện tuần tự nhau. Các bước này được minh hoạ trên hình 7.4.



Hình 7.4. Các bước kiểm thử phần mềm

Ban đầu, việc kiểm thử tập trung vào từng module riêng biệt, đảm bảo rằng nó vận hành đúng như từng đơn vị. Do đó mới có tên kiểm thử đơn vị. Kiểm thử đơn vị dùng nhiều kỹ thuật kiểm thử hộp trắng, thử các đường đặc biệt trong cấu trúc của một module để đảm bảo bao quát đầy đủ và phát hiện ra tối đa lỗi. Tiếp đó, các module phải được lắp ghép hay tích hợp lại để tạo thành bộ phần mềm hoàn chỉnh. Việc kiểm thử tích hợp đề cập đến các vấn đề có liên quan tới kiểm chứng và xây dựng chương trình. Các kỹ thuật thiết kế kiểm thử hộp đen chiếm đại đa số trong việc tích hợp, mặc dù một số giới hạn các kiểm thử hộp trắng cũng có thể được dùng để đảm bảo bao quát đa số các đường điều khiển.

Sau khi phần mềm được tích hợp (được xây dựng), một tập các phép kiểm thử cấp cao sẽ được thực hiện. Các tiêu chuẩn hợp lệ (được thiết lập trong phân tích yêu cầu) cũng phải được kiểm thử. Việc kiểm thử hợp lệ đưa ra sự đảm bảo cuối cùng rằng phần mềm đáp ứng tất cả các yêu cầu về chức năng, hành vi và sự hoàn thiện. Các kỹ thuật kiểm thử hộp đen được dùng chủ yếu trong việc hợp lệ hoá này.

Bước kiểm tra cấp cao cuối cùng nằm ngoài phạm vi của công nghệ phần mềm, chúng thuộc phạm vi rộng hơn của kỹ nghệ hệ thống máy tính. Phần mềm, khi được hợp lệ hoá, phải được tổ hợp với các phần tử hệ thống khác (như phần cứng, con người, cơ sở dữ liệu). Kiểm thử hệ thống kiểm

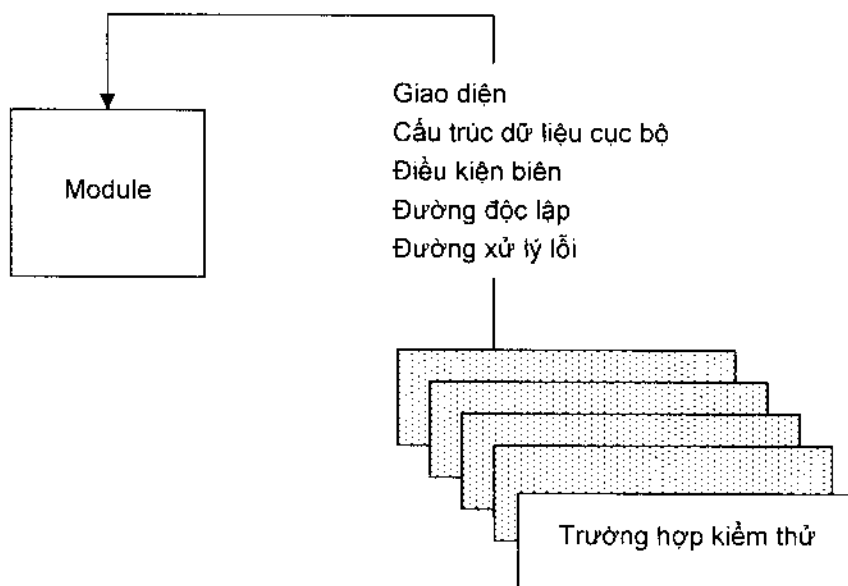
chúng lại xem tất cả các yếu tố có khớp đúng với nhau không và chức năng/độ hoàn thiện hệ thống toàn bộ đã đạt được hay chưa.

7.4.1. Kiểm thử đơn vị

Kiểm thử đơn vị tập trung nỗ lực kiểm chứng vào các đơn vị thiết kế nhỏ nhất của chương trình – module. Bằng việc dùng mô tả thiết kế làm hướng dẫn, các đường điều khiển quan trọng được kiểm thử để phát hiện lỗi trong biên giới của module. Độ phức tạp của việc kiểm thử và các lỗi được phát hiện xem như là kết quả bị giới hạn bởi phạm vi ràng buộc đã thiết lập cho việc thiết kế kiểm thử đơn vị. Việc kiểm thử đơn vị bao giờ cũng hướng theo hộp trắng và bước này được tiến hành song song cho nhiều module.

7.4.1.1. Các xem xét kiểm thử đơn vị

Phép kiểm thử xuất hiện như một phần kiểm thử đơn vị được minh họa trong hình 7.5.



Hình 7.5. Kiểm thử đơn vị

Module giao diện được kiểm thử để đảm bảo thông tin chạy đúng cho việc vào và ra khỏi chương trình đang kiểm thử. Cấu trúc dữ liệu cục bộ được xem xét để đảm bảo rằng dữ liệu được lưu giữ tạm thời vẫn duy trì

tính toàn vẹn của nó trong tất cả các bước khi thực hiện thuật toán. Các điều khiển biên được kiểm thử để đảm bảo rằng module vận hành đúng tại các biên được thiết lập có giới hạn hay hạn chế. Tất cả các đường độc lập (đường cơ sở) đi qua cấu trúc điều khiển đều được cho chạy qua để đảm bảo rằng tất cả các câu lệnh trong một module đều đã được thực hiện ít nhất một lần. Và cuối cùng, tất cả các đường giải quyết lỗi cũng được kiểm thử.

Việc kiểm thử luồng dữ liệu đi qua giao diện module là cần thiết trước khi bất kỳ kiểm thử nào khác được bắt đầu. Nếu dữ liệu không đi vào và đi ra đúng thì tất cả các kiểm thử khác đều phải xem xét lại. Trong một văn bản viết về kiểm thử phần mềm, Myer đề nghị một danh sách các phép kiểm thử đơn vị như sau:

- Số các tham biến vào có bằng số đối hay không?
- Các thuộc tính tham biến và đối có sánh đúng với nhau không?
- Hệ thống các đơn vị tham biến và đối có sánh đúng với nhau không?
- Số các đối được truyền tới module được gọi có bằng số các tham biến hay không?
- Thuộc tính của các đối được truyền cho module được gọi có bằng với thuộc tính của tham biến không?
- Hệ thống các đơn vị của đối được truyền tới module được gọi có bằng hệ thống đơn vị của tham biến hay không?
- Các thuộc tính số và trật tự các tham biến của chức năng có sẵn có đúng hay không?
- Có tham khảo nào tới các tham biến không được gắn với điểm vào hiện tại hay không?
- Có đối ứng với tham biến bị thay đổi không?
- Các định nghĩa biến toàn cục có nhất quán qua các module không?
- Các ràng buộc có được truyền như đối không?

Khi một module thực hiện vào/ra ngoài, cần tiến hành các kiểm thử giao diện phụ:

- Thuộc tính tệp có đúng không?
- Các câu lệnh OPEN/CLOSE có đúng không?

- Đặc tả định dạng có sánh với các câu lệnh vào/ra không?
- Kích cỡ bộ đệm có sánh đúng với kích cỡ bản ghi không?
- Tập có được mở trước khi dùng không?
- Điều kiện cuối tập có được giải quyết không?
- Lỗi vào/ra có được giải quyết không?
- Có lỗi văn bản nào trong thông tin vào không?

Cấu trúc dữ liệu cục bộ cho một module là nguồn gây lỗi thông thường. Các trường hợp kiểm thử nên được thiết kế để phát hiện lỗi trong các phạm trù sau:

- Dặt kiểu không đúng hay không nhất quán.
- Khởi đầu giá trị ngầm định sai.
- Tên biến không đúng (sai chính tả hay bị ngắt cụt).
- Kiểu dữ liệu không nhất quán.
- Tràn dưới, tràn trên và các địa chỉ không xác định.

Bên cạnh các cấu trúc dữ liệu cục bộ, tác động của dữ liệu toàn cục trên một module nên được xác định (nếu có thể) trong kiểm thử đơn vị.

Việc kiểm thử có lựa chọn các đường thực hiện là một nhiệm vụ chủ chốt trong kiểm thử đơn vị. Các trường hợp kiểm thử nên được thiết kế để phát hiện ra lỗi do tính toán sai, so sánh sai hay luồng điều khiển không đúng. Kiểm thử đường cơ sở và chu trình là những kỹ thuật hiệu quả để phát hiện phạm vi rộng các lỗi.

Trong số nhiều lỗi trong tính toán có:

- Hiểu nhầm hay số ưu tiên phép toán số học không đúng.
- Phép toán trộn lẫn nhiều mode.
- Khởi đầu không đúng.
- Độ chính xác không thích hợp.
- Biểu diễn ký hiệu không đúng cho biểu thức.

Việc so sánh và luồng dữ liệu đi đôi chặt chẽ với nhau, tức là thay đổi luồng dữ liệu thường xuất hiện sau một so sánh. Các trường hợp kiểm thử cần phát hiện ra các lỗi như:

- So sánh các kiểu dữ liệu khác nhau.
- Các toán tử logic hay số ưu tiên không đúng.
- Có sự bằng nhau trong khi lỗi độ chính xác trong tính toán bằng nhau không thể xảy ra.
- So sánh với biến không đúng.
- Việc kết thúc chu trình không đúng hay chu trình không tồn tại.
- Lỗi đi ra khi gặp phải việc lặp khác nhau.
- Các biến chu trình bị sửa không đúng.

Thiết kế tốt quy định rằng các điều kiện lỗi phải được thấy trước và đường giải quyết lỗi phải được tạo dựng để việc chọn đường hay có kết thúc việc xử lý rõ rệt mỗi khi lỗi xuất hiện.

Các lỗi tiềm tàng nên được kiểm thử khi giải quyết lỗi là:

- Mô tả lỗi là khó đọc.
- Lỗi được chú thích không tương ứng với lỗi gặp phải.
- Điều kiện lỗi gây ra sự can thiệp hệ thống trước khi giải quyết lỗi.
- Xử lý điều kiện biệt lệ là không đúng.
- Mô tả lỗi không đưa ra đầy đủ thông tin để trợ giúp về vị trí của nguyên nhân lỗi.

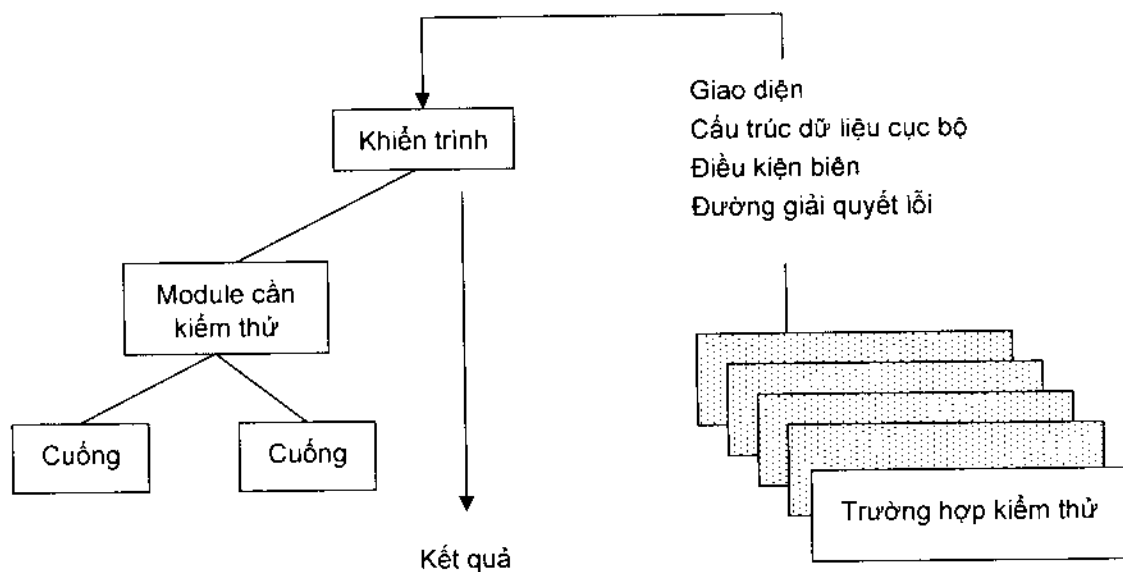
Kiểm thử biên là nhiệm vụ cuối và quan trọng nhất của bước kiểm thử đơn vị. Phần cứng thường bị hỏng ở biên, tức là lỗi thường xuất hiện khi phần tử thứ n của mảng n chiều được xử lý; khi lần lặp thứ i của chu trình với i bước được gọi; khi gặp phải các giá trị tối thiểu hay tối đa cho phép. Các trường hợp kiểm thử chạy qua cấu trúc dữ liệu, luồng điều khiển và giá trị dữ liệu ngay dưới, ở tại và ngay trên cực đại và cực tiểu thường phát hiện ra lỗi.

7.4.1.2. Thủ tục kiểm thử đơn vị

Kiểm thử đơn vị thông thường được xem xét như một bước chuyển sang bước mã hoá. Sau khi mức chương trình gốc đã được xây dựng, xét duyệt và kiểm chứng đúng về cú pháp thì việc thiết kế trường hợp kiểm thử đơn vị bắt đầu. Việc xét duyệt thông tin thiết kế đưa ra hướng dẫn thiết lập các trường hợp kiểm thử có thể phát hiện lỗi trong từng phạm trù đã được

thảo luận ở trên. Mỗi phép kiểm thử nên đi đôi với một tập các kết quả dự kiến.

Vì một module không phải của một chương trình nên phải xây dựng phần mềm khiển trình hay cuống cho từng phép kiểm thử đơn vị. Môi trường kiểm thử đơn vị được minh họa trong hình 7.6.



Hình 7.6. Môi trường kiểm thử đơn vị

Trong phần lớn các ứng dụng, một khiển trình là một “chương trình chính” chấp nhận dữ liệu trường hợp kiểm thử, truyền dữ liệu đó cho module (đã kiểm thử) và in ra kết quả liên quan. Cuống phục vụ thay thế các module vốn là phụ thuộc (được gọi bởi) module đang được kiểm thử. Cuống hay “chương trình con câm” dùng giao diện của các module phụ thuộc có thể làm giảm tối thiểu các thao tác dữ liệu, in ra việc kiểm chứng khi vào và trở về.

Khiển trình và cuống biểu thị tổng phí, tức là cả hai đều là phần mềm cần phải được viết ra (thiết kế chính thức nói chung không được áp dụng) nhưng chúng không được bàn giao trong sản phẩm phần mềm cuối cùng. Nếu khiển trình và cuống được giữ đơn giản thì tổng phí thực tại tương đối thấp. Điều không may là nhiều module không thể được kiểm thử ở mức

đơn vị thích hợp với phần mềm tổng phí “đơn giản”. Trong những trường hợp như vậy, việc kiểm thử đầy đủ được hoàn lại cho tới bước kiểm thử tích hợp.

Việc kiểm thử đơn vị sẽ được đơn giản hoá khi một module với độ phức tạp cao đã được thiết kế sẵn. Khi mỗi module chỉ giải quyết cho một chức năng thì số các trường hợp kiểm thử sẽ được giảm đi và lỗi có thể dự kiến và phát hiện dễ dàng hơn.

7.4.2. Kiểm thử tích hợp

Kiểm thử tích hợp là một kỹ thuật hệ thống để xây dựng cấu trúc chương trình trong khi đồng thời tiến hành các kiểm thử để phát hiện lỗi liên quan đến việc giao tiếp. Mục đích là lấy các module đã kiểm thử đơn vị xong và xây dựng cấu trúc chương trình được quy định bởi thiết kế.

Thường có một khuynh hướng cố gắng tích hợp không tăng dần. Tất cả các module đều được tổ hợp trước. Toàn bộ chương trình được kiểm thử một cách tổng thể. Và thường sẽ là một kết quả lộn xộn, có một tập hợp các lỗi. Việc sửa đổi trở nên khó khăn vì việc cô lập nguyên nhân sẽ phức tạp do trải rộng trên toàn chương trình. Khi những lỗi này được sửa chữa thì những lỗi mới lại xuất hiện và tiến trình này cứ tiếp diễn trong chu trình vô hạn.

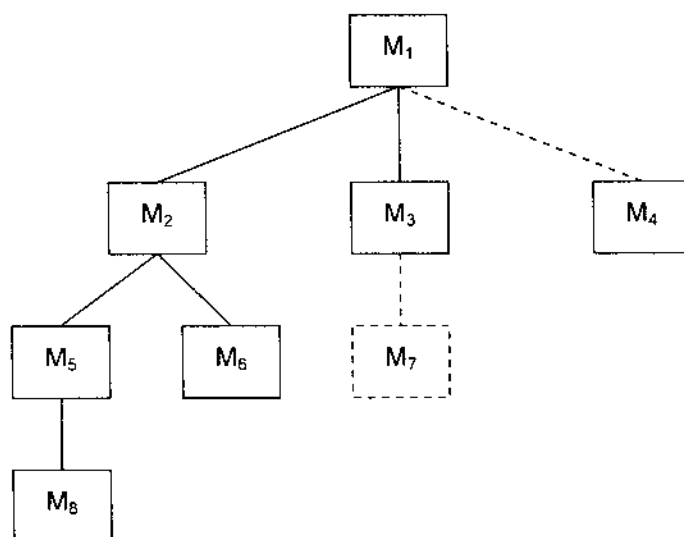
Tích hợp tăng dần là ngược lại cách tiếp cận trên. Chương trình được xây dựng và kiểm thử trong từng đoạn nhỏ, nơi lỗi dễ cô lập và sửa chữa hơn; giao diện được kiểm thử đầy đủ hơn và cách tiếp cận hệ thống được áp dụng.

7.4.2.1. Tích hợp trên xuống

Tích hợp trên xuống là cách tiếp cận tăng dần tới việc xây dựng cấu trúc chương trình. Các module được tích hợp bằng cách đi dần xuống qua các cấp bậc điều khiển, bắt đầu với module chính (chương trình chính). Các module phụ thuộc (và phụ thuộc cuối cùng) và module điều khiển sẽ được tổ hợp dần vào trong cấu trúc theo chiều sâu trước hay chiều rộng trước.

Trong hình 7.7, việc tích hợp chiều sâu trước sẽ tích hợp tất cả các module trên đường điều khiển chính của cấu trúc. Việc chọn một đường chính là tùy ý và phụ thuộc vào các đặc trưng của ứng dụng. Chẳng hạn, chọn đường bên tay trái, các module M_1 , M_2 , M_5 sẽ được tích hợp trước.

Tiếp đó M_8 (nếu cần cho chức năng riêng M_5) sẽ được tích hợp vào, sau đó xây dựng tiếp các đường điều khiển ở giữa và bên phải. Việc tích hợp theo chiều rộng tổ hợp tất cả các module trực tiếp phụ thuộc vào từng mức, đi xuyên qua cấu trúc ngang. Từ hình vẽ, các module M_2, M_3, M_4 (thay thế cho cuống S_4) được tích hợp trước nhất. Các mức điều khiển tiếp $M_5, M_6...$ theo sau.



Hình 7.7. Tích hợp trên xuống

Tiến trình tích hợp được thực hiện qua 5 bước:

1. Module điều khiển chính được dùng như một khiên trình kiểm thử và các cuống được thế vào cho tất cả các module phụ thuộc trực tiếp vào module điều khiển chính.

2. Tùy theo cách tiếp cận tích hợp được chọn lựa (theo chiều sâu hay chiều rộng trước) các cuống phụ thuộc được thay thế từng cái một mỗi lần bằng các module thực tại.

3. Việc kiểm thử được tiến hành khi từng module được tích hợp vào.

4. Khi hoàn thành từng tập các phép kiểm thử, cuống khác sẽ được thay thế bằng các module thực.

5. Kiểm thử hồi quy (tức là tiến hành tất cả hay một số các phép kiểm thử trước) được tiến hành để đảm bảo rằng những lỗi mới không bị đưa thêm vào.

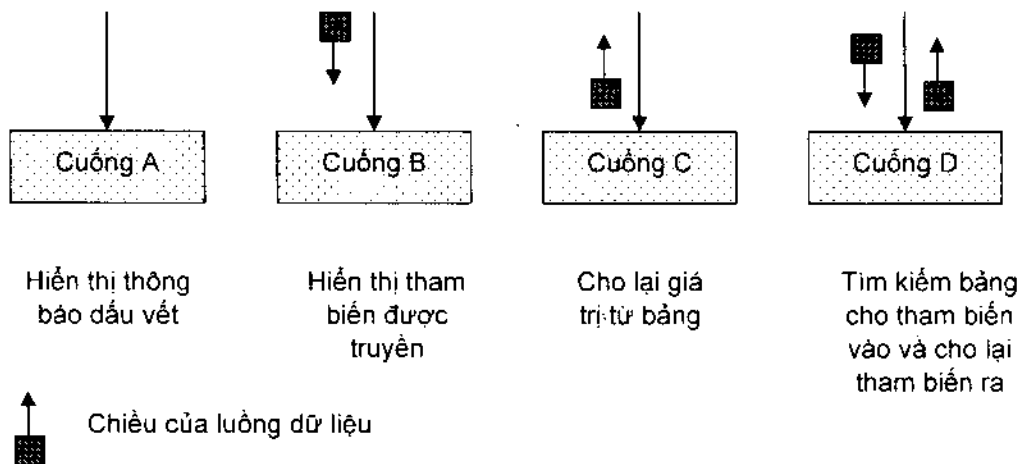
Tiến trình này tiếp tục từ bước 2 tới khi toàn bộ cấu trúc chương trình đã được xây dựng sau. Hình 7.7 minh họa cho tiến trình này. Giả sử ta dùng cách tiếp cận độ sâu trước và một cấu trúc hoàn chỉnh bộ phận, cuống S_7 chuẩn bị được thay thế bởi module M_7 . Bản thân có thể có các cuống sẽ bị thay thế bằng các module tương ứng. Điều quan trọng cần chú ý là phải tiến hành các phép kiểm thử cho mỗi lần thay thế để kiểm thử giao diện.

Chiến lược tích hợp trên xuống kiểm chứng việc điều khiển chính hay các điểm quyết định ngay từ đầu trong tiến trình kiểm thử. Trong một cấu trúc chương trình bố trí khéo, việc quyết định thường xuất hiện tại đỉnh các mức trên trong cấp bậc và do đó sẽ gặp trước. Nếu vấn đề điều khiển chính tồn tại thì việc nhận ra chúng là điều quan trọng nhất. Nếu việc tích hợp theo độ sâu được lựa chọn thì chức năng đầy đủ của phần mềm có thể được cài đặt và chứng minh.

Chiến lược trên xuống có vẻ không phức tạp nhưng trong thực tế, các vấn đề logic có thể nảy sinh. Thông thường, những vấn đề này xuất hiện khi việc xử lý tại mức thấp trong cấp bậc điều khiển đòi hỏi việc kiểm thử thích hợp ở mức trên. Cuống thay thế cho các module cấp thấp vào lúc bắt đầu kiểm thử trên xuống, do đó không có dữ liệu nào có nghĩa có thể dẫn ngược trong cấu trúc chương trình. Người kiểm thử đứng trước các lựa chọn:

1. Dè dặt nhiều việc kiểm thử tới khi cuống thực hiện những chức năng giới hạn mô phỏng cho module thực tại.
2. Xây dựng các cuống thực hiện những chức năng giới hạn mô phỏng cho module thực tại.
3. Tích hợp phần mềm từ đáy cấp bậc lên.

Cách tiếp cận thứ nhất (dè dặt kiểm thử cho tới khi cuống được thay thế bởi module thực tại) làm mất điều khiển tương ứng giữa kiểm thử đặc biệt và việc tổ hợp các module đặc biệt. Điều này có thể dẫn đến những khó khăn trong việc xác định nguyên nhân lỗi và có khuynh hướng vi phạm bản chất ràng buộc cao độ của cách tiếp cận từ trên xuống. Cách tiếp cận thứ hai thì được nhưng có thể dẫn đến tổng phí khá lớn, vì cuống ngày càng phức tạp hơn. Cách tiếp cận thứ ba gọi là kiểm thử dưới lên được trình bày trong mục sau.



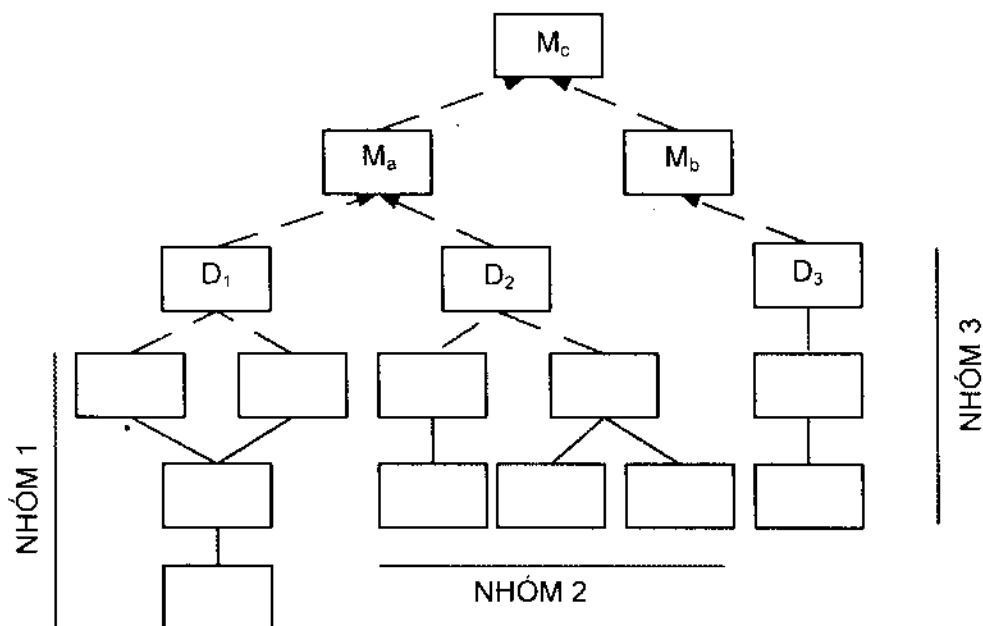
Hình 7.8. Cuồng

7.4.2.2. Tích hợp dưới lên

Kiểm thử tích hợp dưới lên bắt đầu xây dựng và kiểm thử với các module nguyên tử (tức là các module ở mức thấp nhất trong chương trình). Vì các module này được tích hợp từ dưới lên, việc xử lý yêu cầu đối với các module phụ thuộc của một mức nào đó bao giờ cũng có sẵn và nhu cầu về công cụ bị dẹp bỏ.

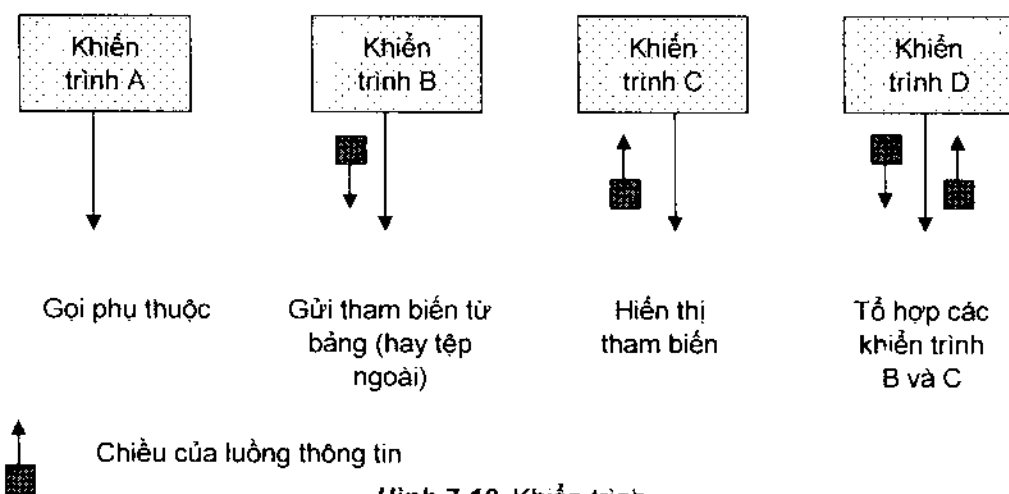
Chiến lược tích hợp từ dưới lên có thể được thực hiện qua những bước sau:

- Các module mức thấp được tổ hợp vào các chùm (đôi khi còn được gọi là kiểu kiến trúc) thực hiện cho một chức năng con phần mềm đặc biệt.
- Khiến trình (một chương trình điều khiển cho kiểm thử) được viết ra để phối hợp việc vào và ra trường hợp kiểm thử.
- Kiểm thử chùm.
- Loại bỏ khiến trình và chùm tổ hợp chuyển lên trong cấu trúc chương trình.
- Việc tích hợp đi theo mô hình được minh họa trong hình 7.9.



Hình 7.9. Tích hợp dưới lên

Các module được tổ hợp tạo nên các chùm 1, 2, 3. Từng chùm đều được kiểm thử bằng cách dùng một khiển trình. Các module trong các chùm 1, 2 phụ thuộc vào M_a . Các khiển trình D_1 và D_2 được loại bỏ và chùm được giao tiếp trực tiếp với M_a . Tương tự, khiển trình D_3 cho chùm 3 bị loại bỏ trước khi tích hợp với module M_b . Cả M_a và M_b cuối cùng sẽ được tích hợp với module M_c , và cứ tiếp tục như vậy. Các loại khiển trình khác được minh họa trong hình 7.10.



Hình 7.10. Khiển trình

Khi tích hợp từ dưới lên, nhu cầu về khiến trình kiểm thử tách biệt ít dần. Trong thực tế, nếu hai mức đỉnh của cấu trúc chương trình được tích hợp theo kiểu trên xuống thì số các khiến trình có thể được giảm bớt khá nhiều và việc tích hợp các chùm đơn giản hơn.

Kết luận

Đã có nhiều thảo luận về ưu và nhược điểm của việc kiểm thử tích hợp trên xuống và dưới lên. Nói chung, ưu điểm của chiến lược này có khuynh hướng trở thành nhược điểm của chiến lược kia. Nhược điểm chính của cách tiếp cận từ trên xuống là cần tới cuống và có những khó khăn kiểm thử kèm theo liên kết với chúng. Tuy nhiên bù lại nó lại có ưu điểm là việc kiểm thử các chức năng điều khiển chính sớm. Nhược điểm chính của việc tích hợp dưới lên là ở chỗ “chương trình như một thực thể chưa tồn tại chừng nào module cuối cùng chưa được thêm vào”. Nhược điểm này được bù đắp bằng thiết kế trường hợp kiểm thử sớm hơn và không cần cuống.

Sự chọn lựa một chiến lược tích hợp phụ thuộc vào các đặc trưng phần mềm và đôi khi cả lịch biểu dự án. Nói chung, một cách tiếp cận tổ hợp (đôi khi còn gọi là kiểm thử sandwich) dùng chiến lược trên xuống cho các mức trên của cấu trúc chương trình, đi đôi với chiến lược dưới lên cho các mức phụ thuộc sẽ là sự thoả hiệp tốt nhất.

Khi việc kiểm thử tích hợp được tiến hành, người kiểm thử phải xác định module găng. Một module găng có một hay nhiều các đặc trưng sau:

- Dề cập tới nhiều yếu tố phần mềm.
- Có mức điều khiển cao (nằm ở vị trí tương đối cao trong cấu trúc chương trình).
- Phức tạp hay dễ sinh lỗi (độ phức tạp xoay vòng có thể được dùng làm chỉ báo).
- Có yêu cầu độ hoàn thiện xác định.

Bên cạnh đó, kiểm thử hồi quy nên tập trung vào chức năng module găng.

7.4.3. Kiểm thử hợp lệ

Vào cao điểm của việc kiểm thử tích hợp, phần mềm được lắp ráp hoàn thành một hệ trình, lỗi giao tiếp đã được phát hiện và sửa chữa, loạt kiểm thử phần mềm cuối cùng – kiểm thử hợp lệ được bắt đầu. Việc hợp lệ có thể xác định theo nhiều cách, nhưng định nghĩa đơn giản nhất của việc hợp lệ hoá thành công là khi các chức năng phần mềm theo cách nào đó chính là thứ khách hàng trông đợi.

7.4.3.1. Tiêu chuẩn kiểm thử hợp lệ

Hợp lệ hoá phần mềm đạt được thông qua một loạt các kiểm thử hợp lệ tuân thủ với các yêu cầu. Bản kế hoạch kiểm thử nêu đại cương các lớp kiểm thử cần tiến hành, còn thủ tục kiểm thử thì xác định các trường hợp kiểm thử đặc biệt được dùng để biểu thị tính tuân thủ yêu cầu. Cả hai bản kế hoạch và thủ tục đều được thiết kế đảm bảo rằng tất cả các yêu cầu chức năng đều được thoả mãn, tất cả các yêu cầu hiệu năng đều đạt được, tài liệu đúng và được viết theo công nghệ cho con người, cùng các yêu cầu khác đã được đáp ứng (như tính khả chuyển, tương hợp, khắc phục lỗi, bảo trì).

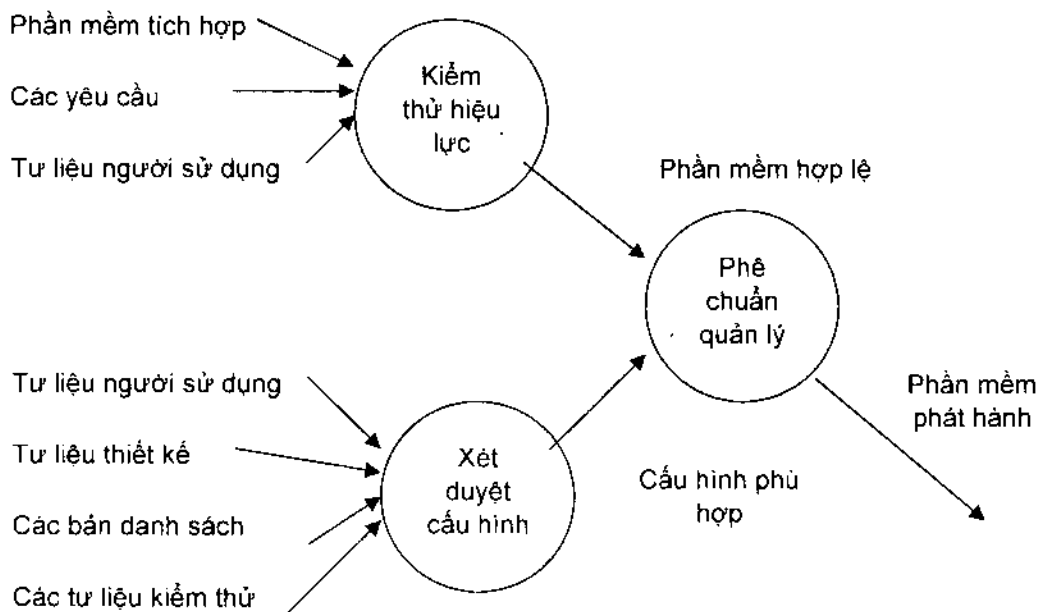
Sau khi tiến hành mỗi trường hợp kiểm thử, một trong hai điều kiện có thể tồn tại:

1. Các đặc trưng chức năng và hiệu năng tuân thủ với đặc tả đã được chấp nhận.
2. Một độ lệch so với đặc tả được phát hiện ra và một danh sách các khiếm khuyết được tạo ra.

Độ lệch hay lỗi được phát hiện ra tại giai đoạn này trong một dự án hiếm khi được sửa đổi trước khi hoàn thành theo lịch. Thường cần phải thương lượng với khách hàng để thiết lập một phương pháp giải quyết các khiếm khuyết.

7.4.3.2. Xem xét cấu hình

Một phần tử quan trọng của tiến trình hợp lệ hoá là xem xét cấu hình. Xét duyệt nhằm để đảm bảo rằng tất cả các phần tử của cấu hình phần mềm đã được phát triển đúng đắn, được phân loại và có chi tiết cần thiết để hỗ trợ cho giai đoạn bảo trì trong vòng đời phần mềm.



Hình 7.11. Xét duyệt cấu hình

7.4.3.3. Kiểm thử alpha và beta

Với người phát triển phần mềm thực tế không thể nào thấy trước được cách khách hàng sẽ dùng một chương trình như thế nào. Các hướng dẫn có thể bị hiểu sai, việc tổ hợp dữ liệu xa lạ có thể được sử dụng; đầu ra dường như rõ ràng với người kiểm thử lại không dễ đọc cho người dùng tại hiện trường.

Khi phần mềm làm theo yêu cầu của khách hàng thì một loạt các kiểm thử chấp nhận sẽ được tiến hành có khả năng hợp lệ hoá các yêu cầu. Việc này được tiến hành bởi người dùng cuối chứ không phải người phát triển hệ thống, việc kiểm thử chấp nhận có thể biến thiên từ khiên trình kiểm thử không hình thức cho tới hàng loạt kiểm thử được vạch kế hoạch trước và có hệ thống. Trong thực tế, việc kiểm thử chấp nhận có thể được tiến hành trong thời kỳ vài tuần hay vài tháng, do đó việc phát hiện các lỗi tích lũy có thể làm suy thoái hệ thống qua thời gian.

Nếu phần mềm được phát triển như một sản phẩm cho nhiều người dùng thì việc thực hiện các kiểm thử chấp nhận chính thức cho từng khách hàng là điều không thực tế. Phần lớn những người xây dựng sản phẩm phần

mềm đều dùng một tiến trình gọi là kiểm thử alpha và beta để phát hiện ra lỗi mà người dùng cuối mới có thể tìm ra.

Kiểm thử alpha được khách hàng tiến hành tại cơ quan của người phát triển. Tuy có sự có mặt của người phát triển nhưng phần mềm được tiến hành kiểm thử tự nhiên, ghi lại lỗi và các vấn đề sử dụng. Kiểm thử alpha được tiến hành trong môi trường có kiểm soát.

Kiểm thử beta được tiến hành tại một hay nhiều cơ quan khách hàng bởi khách hàng hay người dùng cuối của phần mềm. Không giống như kiểm thử alpha, người phát triển không có mặt. Do đó, kiểm thử beta là việc áp dụng “sống” cho phần mềm trong một môi trường người phát triển không có mặt. Khách hàng ghi lại tất cả các vấn đề (thực hay tưởng tượng) gặp phải trong khi kiểm tra beta và báo cáo lại các vấn đề đó cho người phát triển trong các khoảng thời gian đều đặn. Vấn đề được báo cáo trong khi kiểm thử beta sẽ được người phát triển tiến hành sửa đổi và chuẩn bị đưa ra toàn bộ sản phẩm cho khách hàng.

7.4.4. Kiểm thử hệ thống

Phần mềm chỉ là một yếu tố của hệ thống thông tin trên máy tính. Phần mềm được tổ hợp với các yếu tố khác của hệ thống (như phần cứng mới, thông tin) và một loạt các kiểm thử hợp lệ hoá và tích hợp hệ thống sẽ được tiến hành. Những kiểm thử này vượt ra ngoài phạm vi của tiến trình công nghệ phần mềm và được tiến hành không chỉ bởi người phát triển phần mềm. Tuy nhiên, các bước được thực hiện trong thiết kế và kiểm thử phần mềm có thể làm tăng xác suất tích hợp phần mềm thành công trong hệ thống lớn.

Một vấn đề thường xảy ra khi kiểm thử hệ thống là đổ lỗi cho nhau. Điều này xuất hiện khi một khiếm khuyết bị phát hiện ra và người phát triển hệ thống trách cứ người khác về vấn đề này. Đối với kỹ sư phần mềm nên xem xét trước:

- Các vấn đề giao diện tiềm năng.
- Những đường giải quyết lỗi thiết kế.
- Tiến hành các phương pháp mô phỏng dữ liệu tồi hay những lỗi tiềm năng khác tại giao diện phần mềm.

- Ghi lại kết quả của kiểm thử dùng làm bằng chứng nếu xảy ra việc đổ lỗi cho nhau.
- Tham dự vào việc lập kế hoạch và thiết kế các kiểm thử hệ thống để đảm bảo rằng phần mềm đã được kiểm thử tích hợp.

Việc kiểm thử hệ thống thực tế là một loạt các bước kiểm thử khác nhau có mục đích chính là thử đầy đủ hệ thống dựa trên máy tính. Mặc dù, mỗi kiểm thử có một mục tiêu khác nhau nhưng tất cả công việc cùng có mục đích kiểm chứng lại rằng mọi phần tử hệ thống đều đã được tích hợp đúng đắn và thực hiện chức năng được cấp phát.

7.4.4.1. Kiểm thử phục hồi

Nhiều hệ thống trên máy tính phải phục hồi các lỗi và tiếp tục xử lý trong một thời gian xác định trước. Trong một số trường hợp, một hệ thống phải dung sai, tức là lỗi xử lý phải không làm cho toàn bộ hệ thống dừng lại. Trong các trường hợp khác, sai hỏng hệ thống phải được sửa chữa trong một thời kỳ xác định, nếu không sẽ xảy ra thiệt hại kinh tế nghiêm trọng.

Kiểm thử phục hồi là kiểm thử hệ thống bắt buộc phần mềm phải hồng theo nhiều cách và kiểm chứng rằng việc phục hồi được thực hiện đúng. Nếu việc phục hồi là tự động (được thực hiện bởi bản thân hệ thống) thì việc khởi đầu lại, cơ chế điểm kiểm tra, phục hồi dữ liệu và cho chạy lại sẽ được đánh giá là thực hiện đúng. Nếu việc phục hồi đòi hỏi sự can thiệp của con người thì thời gian trung bình để sửa chữa sẽ được ước lượng để xác định xem liệu nó có trong những giới hạn chấp nhận được hay không.

7.4.4.2. Kiểm thử an toàn

Bất kỳ hệ thống nào trên máy tính quản lý những thông tin nhạy cảm hay gây ra những hành động có thể gây hại cho các cá nhân thì đều là mục tiêu xâm nhập không đúng hay bất hợp pháp. Việc thâm nhập này trải trên một phạm vi rộng, các tên tin tặc cố gắng lọt vào hệ thống để giải tri, các nhân viên bất bình định thâm nhập vào hệ thống để trả thù, những cá nhân thâm nhập để trục lợi cá nhân.

Kiểm thử an toàn sẽ kiểm chứng rằng các bảo vệ được thiết lập trong hệ thống thực tế sẽ bảo vệ cho hệ thống khỏi sự xâm nhập có hại.

Trong khi kiểm thử an toàn, người kiểm thử đóng vai trò của cá nhân muốn thâm nhập vào hệ thống. Mọi thứ đều có thể xảy ra. Người kiểm thử

có ý định lấy mật hiệu thông qua các phương tiện thư ký bên ngoài, có thể công kích vào hệ thống bằng các phần mềm chuyên dụng nhằm phá vỡ bất kỳ sự phòng vệ nào đã được xây dựng, có thể tràn ngập hệ thống, do đó hệ thống từ chối phục vụ người khác; có thể gây ra lỗi hệ thống có chủ ý, thâm nhập khi đang phục hồi dữ liệu, duyệt qua các dữ liệu, tìm ra cách mở khoá để vào hệ thống.

Với đủ thời gian và tài nguyên, việc kiểm thử an toàn tốt nhất cuối cùng sẽ là thâm nhập được vào hệ thống, vai trò của người thiết kế hệ thống sẽ làm cho việc thâm nhập này tốn kém hơn những giá trị thông tin thu được.

7.4.4.3. Kiểm thử gay cấn

Trong các bước kiểm thử phần mềm trước đây, các kỹ thuật kiểm thử hộp trắng và kiểm thử hộp đen cho kết quả đánh giá thấu đáo về các chức năng và hiệu năng chương trình thông thường. Kiểm thử gay cấn được thiết kế để làm cho chương trình phải đương đầu với những tình huống bất thường. Về bản chất, người kiểm thử thực hiện việc kiểm thử bất thường hỏi: Ta có thể quay ngược chương trình này trước khi nó bị hỏng được không?

Kiểm thử gay cấn cho hệ thống thử chạy theo cách thức yêu cầu những tài nguyên với số lượng, tần số hay khối lượng bất thường. Chẳng hạn:

- Các kiểm thử đặc biệt có thể được thiết kế để sinh ra 10 ngắt trong một giây, khi tỷ lệ trung bình chỉ là một hoặc hai.
- Tỷ lệ dữ liệu vào có thể tăng lên theo một cấp độ lớn nào đó để xác định cách các chức năng vào sẽ đáp ứng.
- Các trường hợp kiểm thử đòi hỏi bộ nhớ tối đa hay các tài nguyên khác có thể thực hiện được.
- Các trường hợp kiểm thử có thể gây vỡ hệ điều hành thực được thiết kế ra.
- Các trường hợp kiểm thử có thể tiêu tốn dữ liệu thường trú trên đĩa cũng cần phải được thiết kế ra.

Về bản chất, người kiểm thử cố gắng phá vỡ chương trình.

Một biến thể nhạy cảm của kiểm thử gay cấn gọi là kiểm thử nhạy cảm. Trong một số tình huống, một phạm vi dữ liệu rất nhỏ được chứa bên

trong các cận dữ liệu hợp lệ của một chương trình có thể gây ra việc xử lý cục đoan và thậm chí lỗi hay suy giảm hiệu năng. Kiểm thử nhạy cảm cố gắng phát hiện ra các tổ hợp dữ liệu bên trong các lớp đầu vào hợp lệ gây ra việc xử lý ổn định hay không đúng.

7.4.4.4. Kiểm thử hiệu năng

Đối với các hệ thống thời gian thực và nhúng, phần mềm cung cấp các chức năng yêu cầu nhưng không tuân theo các yêu cầu hiệu năng thì không được chấp nhận. Kiểm thử hiệu năng được thiết kế để kiểm thử hiệu năng khi chạy phần mềm trong hệ thống đã tích hợp. Kiểm thử hiệu năng xuất hiện trong toàn bộ các bước của tiến trình kiểm thử. Ngay cả ở kiểm thử đơn vị, hiệu năng của một module riêng cũng có thể được thẩm định khi kiểm thử hộp trắng được tiến hành. Tuy nhiên, phải đến khi tất cả các phần tử hệ thống đều đã được tích hợp hết thì hiệu năng đúng của hệ thống mới chắc chắn có được.

Kiểm thử hiệu năng đôi khi đi kèm với kiểm thử gay cần và thường đòi hỏi cả các thiết bị phần cứng và phần mềm. Tức là, cần đo việc sử dụng tài nguyên (như chu trình bộ nhớ) một cách chính xác. Các thiết bị ngoài có thể điều phối các khoảng thực hiện, sự kiện (ghi lại như các ngắt) khi chúng xuất hiện và ghi lại đều đặn các trạng thái trên. Bằng việc cung cấp thiết bị cho các hệ thống, người kiểm thử có thể phát hiện ra các tình huống dẫn đến việc suy giảm và khả năng sai hỏng của hệ thống.

Chương 8

BẢO TRÌ

8.1. Định nghĩa bảo trì

Không thể tạo ra được một hệ thống hoàn hảo mà hoàn toàn không cần thay đổi. Trong suốt thời gian tồn tại của một hệ thống, các yêu cầu ban đầu của nó sẽ thay đổi do những thay đổi trong nhu cầu của khách hàng và người sử dụng. Môi trường hệ thống cũng thay đổi khi một phần cứng mới được giới thiệu. Lỗi không được phát hiện trong quá trình kiểm thử cũng nguy hiểm và cần phải sửa chữa.

Quá trình thay đổi hệ thống sau khi chuyển giao cho người sử dụng và vẫn đang được sử dụng gọi là bảo trì phần mềm. Sự thay đổi có thể bao gồm một số thay đổi như sửa các lỗi lập trình, sửa các lỗi thiết kế, sửa chữa các lỗi đặc tả hoặc thay đổi để thích ứng với các yêu cầu mới. Do đó, bảo trì trong ngữ cảnh này gần như có nghĩa là tiến hoá.

Người ta chia ra thành các loại bảo trì sau mặc dù sự khác biệt của chúng là không rõ ràng:

- Bảo trì để tu sửa.
- Bảo trì để thích hợp.
- Bảo trì để cải tiến.
- Bảo trì để phòng ngừa.

8.1.1. Bảo trì để tu sửa

Hoạt động bảo trì này để khắc phục những khiếm khuyết trong phần mềm. Hoạt động này được tiến hành vì việc kiểm thử phần mềm chắc chắn

không thể phát hiện ra tất cả các lỗi tiềm tàng trong một hệ thống phần mềm nhất là các hệ thống phần mềm lớn. Khi sử dụng bất cứ chương trình lớn nào, lỗi sẽ xuất hiện và được báo yê cho người phát triển.

Bảo trì này xuất hiện do một số nguyên nhân chủ yếu sau:

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

Khi áp dụng hoạt động bảo trì này cần lưu ý đến:

- Kỹ nghệ ngược: dò lại thiết kế để tu sửa.
- Mức trừu tượng.
- Tính tương tác.
- Tính đầy đủ.

8.1.2. Bảo trì để thích hợp

Hoạt động bảo trì thích hợp chỉnh sửa phần mềm theo những thay đổi của môi trường bên ngoài nhằm duy trì và quản lý phần mềm theo vòng đời của nó. Các thế hệ phần cứng mới, các hệ điều hành mới hay các phiên bản mới của hệ điều hành cũ liên tục xuất hiện, thiết bị ngoại vi và các phần tử hệ thống khác thường xuyên được nâng cấp và thay đổi sẽ yêu cầu phần mềm phải thay đổi để khớp với môi trường thay đổi.

8.1.3. Bảo trì để cải tiến

Khi phần mềm được đưa vào sử dụng, ta sẽ nhận được từ người sử dụng những khuyến cáo về khả năng mới, những sửa đổi về chức năng hiện tại và những yêu cầu nâng cao. Để thoả mãn những yêu cầu trong phạm trù này, người ta tiến hành bảo trì để cải tiến.

Các bước thực hiện bảo trì để cải tiến:

- Xây dựng lưu đồ phần mềm.
- Suy dẫn ra biểu thức Bun cho từng dãy xử lý.

- Biên dịch bảng chân lý (đúng/sai).
- Tái cấu trúc phần mềm.

8.1.4. Bảo trì để phòng ngừa

Đây là hoạt động bảo trì tu chỉnh chương trình có tính đến tương lai phần mềm đó sẽ mở rộng và thay đổi như thế nào. Thực ra, trong khi thiết kế phần mềm, người thiết kế luôn tính đến khả năng mở rộng của nó nên thực tế ít khi thực hiện bảo trì phòng ngừa nếu như phần mềm được thiết kế tốt.

Bảo trì này được thực hiện trên các thiết kế không tường minh. Sử dụng các công cụ CASE cho kỹ nghệ ngược và tái kỹ nghệ sẽ tự động hoá một phần công việc.

8.2. Các đặc trưng của bảo trì

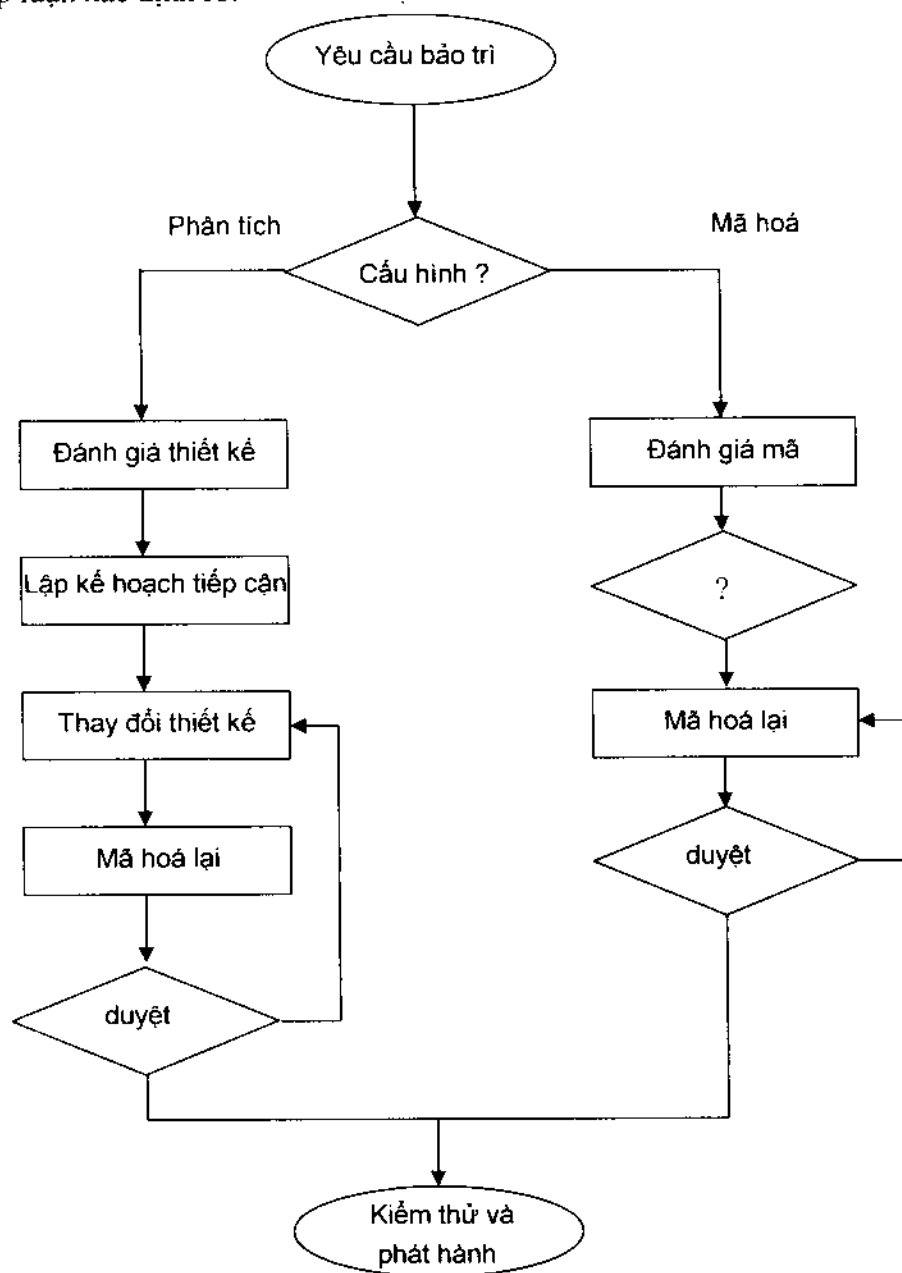
Để hiểu các đặc trưng của bảo trì phần mềm, ta xem xét dựa trên ba quan điểm sau:

- Các hoạt động đòi hỏi hoàn thành giai đoạn bảo trì và tác động của cách tiếp cận công nghệ phần mềm (hay thiếu cách tiếp cận) lên tính hiệu quả của các hoạt động như vậy.
- Chi phí phần mềm giai đoạn bảo trì.
- Các vấn đề thường gặp khi tiến hành bảo trì phần mềm.

8.2.1. Bảo trì có cấu trúc so với bảo trì phi cấu trúc

Luồng các sự kiện có thể xuất hiện như là kết quả của yêu cầu bảo trì được minh hoạ trên hình 8.1. Nếu phần tử sẵn có của cấu hình phần mềm là chương trình gốc, thì hoạt động bảo trì bắt đầu với một đánh giá cẩn thận về mã, tuy nhiên công việc này thường khá phức tạp do tài liệu nghèo nàn. Những đặc trưng tinh vi như cấu trúc chương trình, cấu trúc dữ liệu toàn cục, giao diện hệ thống và các ràng buộc hiệu năng hoặc thiết kế thường không chắc chắn, dễ bị hiểu sai. Việc phân nhánh các thay đổi thường được tiến hành sau cùng cho chương trình nên rất khó thẩm định. Các phép kiểm thử hồi quy (lập lại phép thử quá khứ để đảm bảo rằng những sửa đổi không đưa ra lỗi vào phần mềm vận hành trước đây) không thể nào tiến hành được vì không có bản ghi lại việc kiểm thử. Việc bảo trì phi cấu trúc

trên đây sẽ gây ra sự lãng phí công sức và nhầm lẫn cho con người thường đi kèm với những phần mềm chưa được phát triển có dùng phương pháp luận xác định rõ.



Hình 8.1. Bảo trì có cấu trúc so với bảo trì phi cấu trúc

Nếu tồn tại một cấu hình phần mềm đầy đủ, bảo trì sẽ bắt đầu với việc đánh giá tài liệu thiết kế. Cấu trúc quan trọng, hiệu năng và các đặc trưng giao diện của phần mềm được xác định. Việc sửa đổi hay sửa chữa theo yêu cầu sẽ được thẩm định và lên kế hoạch. Bản thiết kế sẽ được sửa đổi và xét duyệt lại. Chương trình gốc mới được xây dựng, các phép thử hồi quy được tiến hành bằng cách dùng các thông tin có trong bản *Đặc tả kiểm thử* và phần mềm xuất xưởng là kết quả của việc áp dụng các phương pháp luận về công nghệ phần mềm. Tuy nhiên sự tồn tại của cấu hình phần mềm không đảm bảo việc bảo trì không có vấn đề, nhưng khối lượng công việc cũng đã được rút gọn đáng kể và chất lượng tổng thể của việc thay đổi hay sửa đổi được nâng cao.

8.2.2. Chi phí bảo trì

Chi phí cho việc bảo trì phần mềm đã tăng dần trong 20 năm qua. Chi phí về tài chính luôn là mối quan tâm của chúng ta. Tuy nhiên, có những chi phí ít thấy nhiều khi lại là mối quan tâm cần được ưu tiên. Một chi phí vô hình của việc bảo trì phần mềm là cơ hội phát triển bị trì hoãn hay bị mất tài nguyên sẵn có phải chuyển cho các nhiệm vụ bảo trì. Các chi phí vô hình khác bao gồm:

- Sự không thoả mãn của khách hàng khi các yêu cầu sửa chữa hay thay đổi hợp lý lại không được đề cập đến một cách kịp thời.
- Sự suy giảm chất lượng phần mềm tổng thể do những thay đổi phát sinh thêm lỗi trong phần mềm sau khi đã bảo trì.
- Biến động đột ngột xảy ra trong nỗ lực phát triển khi nhân viên bị “kéo” sang làm việc bảo trì.

Chi phí cuối cùng cho việc bảo trì phần mềm làm giảm hiệu suất bảo trì (được đo theo số dòng lệnh (LOC) trên người – tháng hay điểm chức năng (FP) trên người – tháng), điều thường gặp phải khi bắt đầu bảo trì chương trình cũ.

Nỗ lực dành cho việc bảo trì có thể bị phân chia vào các hoạt động sản xuất (như phân tích và đánh giá, sửa đổi thiết kế, mã hoá) và các hoạt động như hiệu mã chương trình làm gì, thử diễn giải cấu trúc dữ liệu, các đặc trưng giao diện, các biên hiệu năng.

Biểu thức sau đây đưa ra mô hình về nỗ lực bảo trì:

$$M = p + K_e^{(c \cdot d)}$$

Với : p – nỗ lực hiệu suất

M – tổng nỗ lực dành cho bảo trì.

K_e – hằng số kinh nghiệm.

c – đo độ phức tạp.

d – việc đo mức độ quen thuộc với phần mềm.

Mô hình được mô tả trên cho thấy nỗ lực (và chi phí) có thể tăng lên theo hàm mũ nếu sử dụng cách tiếp cận phát triển phần mềm nghèo nàn (tức là thiếu kỹ nghệ phần mềm) và người hay nhóm người dùng cách tiếp cận này còn chưa sẵn có để thực hiện việc bảo trì.

Lưu ý:

Chi phí cho việc thêm vào một chức năng mới của hệ thống sau khi nó được triển khai thường lớn hơn chi phí xây dựng các chức năng tương tự khi phần mềm được phát triển ban đầu bởi vì:

- Các nhân viên bảo trì thường không có kinh nghiệm và không quen thuộc với miền ứng dụng. Bảo trì là một công việc nhằm chán đối với các kỹ sư phần mềm. Đây thường được xem là công việc đòi hỏi ít kỹ năng hơn phát triển hệ thống nên nó thường được giao cho các nhân viên cấp thấp.
- Chương trình được bảo trì đã được phát triển nhiều năm trước đây nên không có các kỹ thuật kỹ nghệ phần mềm hiện đại. Chúng cần được xây dựng và tối ưu hoá để nâng cao tính hiệu quả.
- Thay đổi chương trình có thể làm phát sinh ra các lỗi mới. Các lỗi này lại yêu cầu các thay đổi.
- Khi hệ thống thay đổi, cấu trúc của nó có thể bị suy biến. Điều đó làm cho hệ thống khó hiểu hơn và các thay đổi sẽ khó khăn hơn khi tính cố kết của hệ thống bị giảm.
- Mỗi liên quan giữa chương trình và các tài liệu liên quan giảm đi. Do đó, các tài liệu này không còn đủ tin cậy để trợ giúp việc hiểu chương trình.

8.2.3. Các vấn đề

Phần lớn các vấn đề liên quan đến bảo trì phần mềm đều có thể quy về những khiếm khuyết trong việc vạch kế hoạch và xây dựng phần mềm.

Trong số các vấn đề cổ điển có liên quan đến việc bảo trì, có các vấn đề sau:

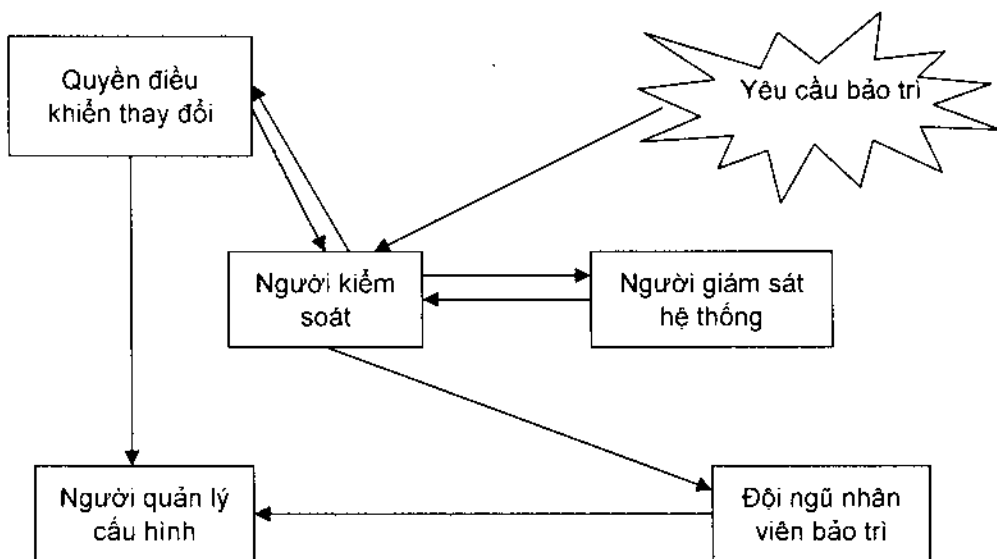
- Thường khó hay không thể thay đổi được sự phát triển của phần mềm qua nhiều phiên bản hay lần phát hành. Những thay đổi không được chứng minh bằng tư liệu một cách đầy đủ.
- Khó hay không thể nào theo dõi được tiến trình tạo nên phần mềm.
- Đặc biệt khó để biết chương trình là của ai đó. Độ khó tăng lên khi số phần tử trong cấu hình phần mềm giảm đi. Nếu chỉ có mã chương trình mà không có tư liệu thì chắc chắn sẽ xảy ra nhiều vấn đề nghiêm trọng.
- Thường không có ai đó bên cạnh để giải thích. Tính cơ động giữa các nhân viên phần mềm khá cao. Chúng ta không thể dựa vào giải thích cá nhân của người phát triển phần mềm khi bảo trì.
- Tài liệu không có hoặc có nhưng tồi. Thừa nhận rằng, phần mềm phải được tư liệu hoá, nhưng việc tư liệu hoá phải làm sao để có thể hiểu được và nhất quán với chương trình gốc.
- Phần lớn các phần mềm đều không được thiết kế có tính đến sự thay đổi. Khi phương pháp thiết kế chưa phù hợp với sự thay đổi thông qua sự phụ thuộc vào hàm hay lớp sự vật thì việc thay đổi phần mềm sẽ gặp khó khăn và phát sinh lỗi.
- Việc bảo trì phần mềm còn chưa được coi là công việc hấp dẫn.

Tất cả những vấn đề được mô tả ở trên một phần đều xuất phát từ lý do các chương trình đang tồn tại hiện nay chưa chú trọng tới công nghệ phần mềm. Công nghệ phần mềm cung cấp các giải pháp cụ thể cho từng vấn đề có liên quan đến việc bảo trì.

8.3. Nhiệm vụ bảo trì

Nhiệm vụ bảo trì phần mềm được tiến hành trước khi một yêu cầu về bảo trì được nêu ra. Ban đầu, phải thành lập tổ chức bảo trì chính thức hay ngầm định, phải mô tả các thủ tục báo cáo, đánh giá và xác định trình tự chuẩn hoá cho các sự kiện đối với từng yêu cầu bảo trì. Bên cạnh đó, cũng cần thiết lập thủ tục lưu giữ các hoạt động bảo trì và việc xác định ra các tiêu chuẩn xét duyệt và đánh giá.

8.3.1. Tổ chức bảo trì



Hình 8.2. Tổ chức

Các yêu cầu bảo trì được chuyển cho người kiểm soát bảo trì, là người chuyển tiếp từng yêu cầu đánh giá cho người giám sát hệ thống. Người giám sát hệ thống là thành viên của bộ phận kỹ thuật, những người được trao trách nhiệm này phải am hiểu từng tập con nhỏ nhất của chương trình sản phẩm. Khi việc đánh giá được tiến hành thì người có thẩm quyền điều khiển thay đổi (đôi khi còn được gọi là ban điều khiển thay đổi) phải xác định hoạt động cần tiến hành.

Cách tổ chức trên sẽ làm giảm bớt được sự xáo trộn và cải tiến luồng hoạt động bảo trì. Tất cả các yêu cầu bảo trì đều tập trung về một cá nhân hay một nhóm, sự sắp xếp các yêu cầu sẽ tránh được sự xáo trộn vì một cá nhân bao giờ cũng có một sự am hiểu nào đó với chương trình sản phẩm, nên yêu cầu thay đổi (bảo trì) sẽ được thẩm định nhanh chóng hơn. Trong hoạt động bảo trì, vì chấp nhận các điều kiện thay đổi đặc biệt nên có thể tránh được việc thay đổi có lợi cho một người yêu cầu này mà lại có tác dụng tiêu cực với người khác.

Mỗi một tiêu đề công việc trên đều dùng để thiết lập một lĩnh vực trách nhiệm cho bảo trì. Người kiểm soát quyền điều khiển thay đổi có thể chỉ là một người hay một nhóm các nhà quản lý và nhân viên kỹ thuật cấp cao

(với hệ thống lớn). Người giám sát hệ thống có thể có nghĩa vụ khác, nhưng cũng “tiếp xúc” với bộ phận mềm đặc biệt.

Khi các trách nhiệm được trao để bắt đầu hoạt động bảo trì thì sự lẫn lộn được giảm đi nhiều. Nhưng điều quan trọng hơn, việc xác định trách nhiệm từ sớm có thể giảm bớt cảm giác khó chịu thường xảy ra khi một người bị “kéo ra” khỏi nỗ lực phát triển khi tiến hành bảo trì.

8.3.2. Báo cáo

Tất cả các yêu cầu về bảo trì phần mềm được trình bày theo cách thức chuẩn hoá. Người phát triển phần mềm tạo ra một mẫu yêu cầu bảo trì (MRF), đôi khi còn được gọi là báo cáo vấn đề phần mềm, do người muốn có hoạt động bảo trì điền vào. Nếu gặp phải lỗi, cần phải đưa ra bản mô tả đầy đủ về hoàn cảnh dẫn tới lỗi (dữ liệu vào, bản in và các tài liệu hỗ trợ khác). Với các yêu cầu bảo trì thích nghi hay hoàn thiện, một đặc tả thay đổi ngắn gọn (đặc tả yêu cầu tóm tắt) cần được đưa ra.

MRF là tài liệu được tạo ra bên ngoài, vốn được dùng làm cơ sở cho việc lập kế hoạch nhiệm vụ bảo trì. Về nội bộ, tổ chức phần mềm xây dựng một báo cáo thay đổi phần mềm (SCR) đưa ra:

1. Độ lớn của nỗ lực cần cho việc thoả mãn MRF.
2. Bản chất của những thay đổi cần có.
3. Số ưu tiên của yêu cầu.
4. Dữ liệu sau sự kiện về việc thay đổi.

SCR được đệ trình để thay đổi quyền kiểm soát trước khi việc lập kế hoạch bổ sung được tiến hành.

8.3.3. Luồng sự kiện

Yêu cầu đầu tiên là xác định kiểu bảo trì cần được tiến hành. Trong nhiều trường hợp, người dùng có thể yêu cầu bảo trì về lỗi phần mềm (bảo trì sửa chữa) trong khi người phát triển có thể coi yêu cầu đó là thích nghi hay nâng cao. Nếu tồn tại những ý kiến khác nhau thì phải thương lượng phương pháp.

Một yêu cầu cho bảo trì sửa lỗi (đường lỗi) bắt đầu với một đánh giá về mức nghiêm trọng của lỗi. Nếu lỗi nghiêm trọng xuất hiện (như hệ thống

tới hạn không thể vận hành) thì nhân sự sẽ được phân bổ theo chi thị của người giám sát hệ thống và việc phân tích vấn đề được tiến hành ngay lập tức. Với những lỗi ít nghiêm trọng hơn, yêu cầu về bảo trì sửa chữa được ước lượng và phân loại rồi lập lịch đi kèm với các nhiệm vụ khác cần tới tài nguyên phát triển phần mềm.

Trong một số trường hợp, một lỗi có thể nghiêm trọng đến mức việc kiểm soát thông thường về bảo trì bị tạm thời ngưng lại. Chương trình phải được sửa đổi ngay lập tức. Cách “chữa cháy” này cho bảo trì sửa chữa chỉ được dành riêng cho các tình huống “khủng hoảng” và chỉ là một vài phần trăm rất nhỏ trong tất cả các hoạt động bảo trì. Cũng nên lưu ý rằng việc chữa cháy trì hoãn nhưng không được bỏ qua việc kiểm soát và ước lượng. Sau khi giải quyết xong khủng hoảng thì những công việc này phải được tiến hành để đảm bảo rằng những lỗi hiện tại sẽ không gây ra các vấn đề nghiêm trọng khác.

Các yêu cầu về bảo trì thích nghi và hoàn thiện được tiến hành theo một con đường khác. Bảo trì thích nghi được đánh giá và phân loại (sắp xếp ưu tiên) trước khi đặt vào hàng đợi bảo trì. Bảo trì nâng cao cũng trải qua khâu đánh giá này. Tuy nhiên, không phải tất cả các yêu cầu nâng cao đều được tiến hành. Chiến lược kinh doanh, tài nguyên có sẵn, khuynh hướng sản phẩm phần mềm hiện tại và tương lai và nhiều vấn đề khác có thể làm cho yêu cầu nâng cao bị bác bỏ. Bảo trì nâng cao cũng cần được đặt vào hàng đợi bảo trì. Mức độ ưu tiên cho từng yêu cầu được thiết lập và công việc cần thiết sẽ được lên lịch thực hiện. Nếu có một số yêu cầu ưu tiên cao thì công việc có thể bắt đầu ngay lập tức.

Bất kể kiểu bảo trì nào cũng phải tiến hành theo kỹ thuật. Những nhiệm vụ bảo trì này bao gồm sửa đổi thiết kế phần mềm, xét duyệt, sửa chương trình, kiểm thử đơn vị và tích hợp (kể cả kiểm thử hồi quy dùng các trường hợp kiểm thử trước đây), kiểm thử hợp lệ và xét duyệt. Trong thực tế, việc bảo trì phần mềm chính là kỹ nghệ phần mềm được áp dụng đệ quy. Tùy từng kiểu bảo trì sẽ có sự khác biệt, nhưng cách tiếp cận toàn bộ vẫn không thay đổi. Công việc cuối cùng trong luồng bảo trì là xét duyệt làm hợp lệ tất cả các phần tử của cấu hình phần mềm và đảm bảo rằng MRF trong thực tế đã được hoàn thành.

Sau khi nhiệm vụ bảo trì được hoàn tất, cần tiến hành xét duyệt tình huống. Nói chung, việc xét duyệt này nhằm trả lời các câu hỏi sau:

- Với các tình huống được cho, những khía cạnh nào của thiết kế, mã hoá hay kiểm thử có thể được thực hiện khác đi?
- Tài nguyên bảo trì nào đáng phải có mà lại không có?
- Khó khăn chính (phụ) cho nỗ lực này là gì?
- Liệu việc bảo trì phòng ngừa theo yêu cầu có được báo cáo lại hay không?

Việc xét duyệt tình huống này có thể ảnh hưởng tới việc tiến hành các nỗ lực bảo trì tương lai và đưa ra phản hồi, là điểm quan trọng trong quản lý của tổ chức phần mềm.

8.3.4. Lưu trữ bản ghi

Vấn đề đầu tiên gặp phải khi lưu giữ bản ghi bảo trì là phải hiểu dữ liệu nào đáng ghi lại. Swanson đã nêu ra một danh sách có cân nhắc sau:

1. Xác định chương trình.
2. Số các câu lệnh chương trình gốc.
3. Số các lệnh mã máy.
4. Ngôn ngữ lập trình được dùng.
5. Ngày thiết lập chương trình.
6. Số chương trình chạy từ lần thiết lập đó.
7. Số lần hỏng hóc xử lý liên quan đến mục 6.
8. Mức độ và việc định danh sự thay đổi chương trình.
9. Số câu lệnh gốc được thêm vào do thay đổi chương trình.
10. Số câu lệnh gốc bị xoá.
11. Số người – giờ dành cho việc thay đổi.
12. Ngày thay đổi chương trình.
13. Định danh kỹ sư phần mềm.
14. Định danh MRF.
15. Kiểu bảo trì.

16. Ngày bắt đầu và kết thúc bảo trì.
17. Số tích lũy về người – giờ dành cho bảo trì.
18. Ích lợi rõ ràng liên kết với việc bảo trì được thực hiện.

Các dữ liệu trên được thu thập cho từng nỗ lực bảo trì. Theo Swanson, đó là nền tảng cho cơ sở dữ liệu bảo trì.

Bảo trì phần mềm còn tập trung vào các đặc trưng phần mềm ảnh hưởng tới tần số bảo trì và các mô hình kinh nghiệm để dự đoán khối lượng công việc bảo trì dựa trên các đặc trưng chương trình khác. Bằng cách dùng mô hình COCOMO làm cơ sở, mô hình sau đây đã gợi ý như một bộ dự báo về số người – tháng (E.maint) cần cho việc bảo trì phần mềm hằng năm:

$$E.maint = ACT \times 2,4 \times KLOC^{1,05}$$

Với ACT là lưu lượng thay đổi hằng năm được định nghĩa là:

$$ACT = KLOC \text{ cho hệ thống đang được bảo hành/CI}$$

Với CI là số lệnh chương trình gốc bị thay đổi hay thêm vào trong một năm bảo trì.

Tuy các kết quả nghiên cứu được áp dụng một cách thận trọng, nhưng các mô hình định lượng đưa ra một phần tử của việc kiểm soát quản lý hay bị bỏ qua khi bảo trì phần mềm.

8.3.5. Ước lượng

Ước lượng về các hoạt động bảo trì thường phức tạp do thiếu dữ liệu. Nếu việc lưu giữ bản ghi được thực hiện từ đầu thì có thể xây dựng được một số cách đo hiệu năng bảo trì. Swanson đưa ra một danh sách tóm tắt các độ đo hiệu năng:

1. Số trung bình những sai hỏng xử lý khi chạy chương trình.
2. Toàn bộ số người – giờ dành cho từng phạm trù bảo trì.
3. Số trung bình những thay đổi chương trình được thực hiện theo chương trình, theo ngôn ngữ, theo kiểu bảo hành.
4. Số trung bình người – giờ trên câu lệnh chương trình gốc cần thêm vào hay xóa đi do việc bảo trì.
5. Số người – giờ trung bình dành ra cho ngôn ngữ.

6. Thời gian quay vòng trung bình cho một MRF.

7. Số phần trăm các yêu cầu bảo trì phần mềm.

7 cách đo trên có thể cung cấp giá trị định lượng để đưa ra những quyết định về kỹ thuật phát triển, chọn lựa ngôn ngữ, dự phòng nỗ lực bảo trì, cấp phát tài nguyên và nhiều vấn đề khác. Rõ ràng, những dữ liệu như vậy có thể được áp dụng để ước lượng nhiệm vụ bảo trì.

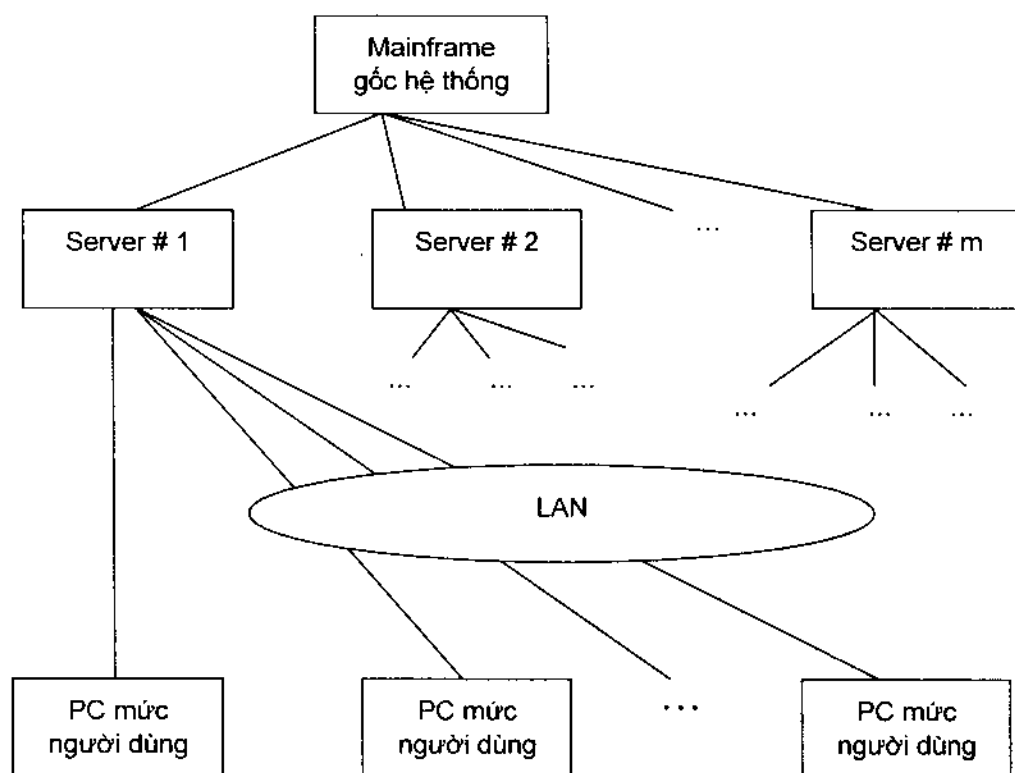
Chương 9

CÁC CHỦ ĐỀ NÂNG CAO TRONG CÔNG NGHỆ PHẦN MỀM

9.1. Công nghệ phần mềm client/server

9.1.1. Cấu trúc hệ thống client/server

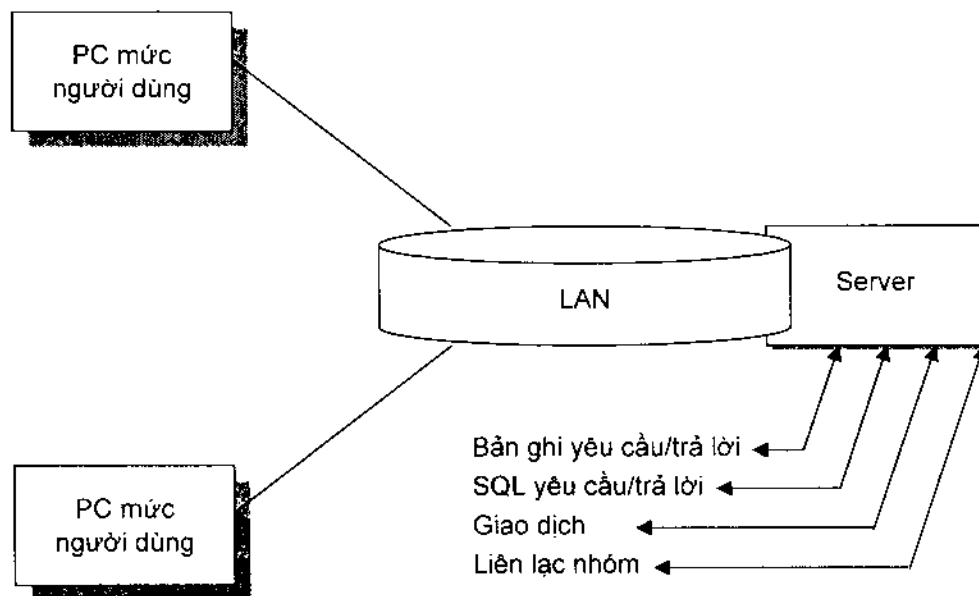
Phần cứng, phần mềm, cơ sở dữ liệu và công nghệ truyền thông mạng là các thành phần của cấu trúc máy tính phân tán. Nói chung, một hệ thống máy tính phân tán được minh họa như trong hình 9.1.



Hình 9.1. Cấu trúc hệ thống máy tính phân tán

Hệ thống gốc, thường là mainframe, được dùng như kho chứa dữ liệu. Hệ thống gốc được nối với server (thường là các workstation mạnh hay các PC) có hai vai trò : Server hoạt động để cập nhật và yêu cầu dữ liệu được lưu trong mainframe. Chúng cũng duy trì các hệ thống cục bộ và đóng vai trò trọng yếu trong các mạng PC mức người dùng qua các mạng LAN.

Trong cấu trúc client/server (C/S), máy tính nằm ở trên các máy tính khác được gọi là server và các máy tính ở các mức thấp hơn gọi là client. Client yêu cầu các dịch vụ, và server sẽ cung cấp chúng. Tuy nhiên, có thể có một số điểm khác như mô tả trong hình 9.2.



Hình 9.2. Các lựa chọn kiến trúc client/server

- *File servers* : Client yêu cầu các bản ghi cụ thể từ một file. Server chuyển các bản ghi này tới client qua mạng.
- *Database servers*: Client gửi các yêu cầu SQL (Structure Query Language – ngôn ngữ hỏi đáp cấu trúc) đến server. Chúng được chuyển như các thông điệp trên mạng. Server xử lý các yêu cầu SQL và tìm các thông tin yêu cầu, gửi kết quả lại cho client.
- *Transaction servers*: Client gửi yêu cầu gọi các thủ tục từ xa tại phía server. Các thủ tục từ xa có thể là một tập các lệnh SQL. Một giao

dịch xảy ra khi các thủ tục từ xa được thực hiện và chuyển kết quả về client.

- *Groupware servers*: Khi server cung cấp một tập các ứng dụng có khả năng liên lạc giữa các client (và giữa những người sử dụng chúng) sử dụng văn bản, hình ảnh, các bảng tin, video..., kiến trúc groupware tồn tại.

9.1.1.1. Các thành phần phần mềm của hệ thống C/S

Thay vì nhìn phần mềm như một ứng dụng nguyên khối được thực hiện trên một máy, phần mềm liên kết với kiến trúc C/S có nhiều thành phần khác biệt được đặt ở client, hoặc server, hoặc phân tán giữa các máy. Cụ thể như sau :

- *Các thành phần trình diễn/ tương tác với người dùng*: thực hiện tất cả các chức năng thường được liên kết với giao diện người dùng đồ hoạ GUI.
- *Các thành phần ứng dụng*: thực hiện các yêu cầu được xác định bởi các ứng dụng trong ngữ cảnh của miền trong đó ứng dụng hoạt động. Ví dụ, một ứng dụng nghiệp vụ tạo ra nhiều loại báo cáo dựa trên các dữ liệu số đầu vào, tính toán, thông tin cơ sở dữ liệu... Một ứng dụng groupware cung cấp khả năng liên lạc bằng tin hoặc email. Trong cả hai trường hợp, phần mềm ứng dụng có thể được chia nhỏ để một số phần nằm ở phía client, một số phần nằm ở phía server.
- *Quản lý cơ sở dữ liệu*: thực hiện các thao tác quản lý dữ liệu yêu cầu bởi một ứng dụng. Thao tác này có thể chỉ đơn giản là chuyển một bản ghi hoặc phức tạp như thực thi một lệnh SQL phức tạp.

Ngoài các thành phần này, các khối phần mềm khác thường được gọi là middleware cũng tồn tại trong tất cả các hệ thống C/S. Middleware bao gồm các thành phần phần mềm tồn tại ở cả client và server, các thành phần của hệ điều hành mạng cũng như các phần mềm ứng dụng chuyên biệt chứa hỗ trợ các ứng dụng về cơ sở dữ liệu, các chuẩn môi giới yêu cầu đối tượng, công nghệ groupware, quản lý truyền thông và các đặc tính khác làm cho kết nối client/server dễ dàng hơn.

9.1.1.2. Sự phân bố của các thành phần phần mềm

Khi các yêu cầu cơ bản cho một ứng dụng client/server được đưa ra, kỹ sư phần mềm phải quyết định cách phân bố các thành phần phần mềm đã đề cập ở trên giữa client và server. Khi phần lớn các chức năng liên kết với ba thành phần này được đặt trên phía server, thiết kế fat server sẽ được tạo ra. Ngược lại, khi client thực hiện phần lớn các trình diễn/tương tác người dùng, ứng dụng và thành phần dữ liệu, thiết kế fat client được tạo ra.

Fat client thường được dùng khi cấu trúc file server và database server đã được thực thi. Trong trường hợp này, server cung cấp các hỗ trợ quản lý dữ liệu, nhưng tất cả các ứng dụng và các phần mềm GUI nằm ở phía client. Fat server thường được thiết kế khi các giao dịch và các phần mềm groupware được thực hiện. Server cung cấp hỗ trợ ứng dụng được yêu cầu để đáp ứng các giao dịch và truyền thông từ phía client. Phần mềm client tập trung vào GUI và quản lý truyền thông.

Fat server và fat client thường được dùng để minh họa các cách tiếp cận chung cho việc phân phối các thành phần client/server. Tuy nhiên có một cách tiếp cận chung hơn về cách phân phối thành phần phần mềm xác định 5 cấu hình khác nhau:

- *Trình diễn phân tán*: Trong cách tiếp cận client/server ban đầu, cơ sở dữ liệu logic và ứng dụng logic để chuẩn bị cho các thông tin màn hình sử dụng các phần mềm như CICS. Các phần mềm dựa trên PC đặc biệt được dùng để chuyển các thông tin màn hình bằng các ký tự được truyền từ server thành các trình diễn GUI cho PC.
- *Trình diễn từ xa*: Trong mở rộng của cách tiếp cận trình diễn phân tán, các cơ sở dữ liệu cơ sở, các ứng dụng logic trên server và các dữ liệu gửi bởi server được client sử dụng để chuẩn bị trình diễn người dùng.
- *Logic phân tán*: Client được phân công thực hiện tất cả các nhiệm vụ trình diễn người dùng và các quy trình liên kết với các dữ liệu vào như các trường dữ liệu hợp lệ, server được phân công các nhiệm vụ quản lý cơ sở dữ liệu và xử lý các truy vấn từ client, cập nhật file server và các ứng dụng khác.

- *Quản lý dữ liệu từ xa:* Ứng dụng trên server tạo ra nguồn dữ liệu bằng cách định dạng dữ liệu đã được trích từ một nơi khác. Ứng dụng đặt ở client được dùng để khai thác dữ liệu mới đã định dạng bởi server. Hệ trợ giúp quyết định là một hệ kiểu này.
- *Cơ sở dữ liệu phân tán:* Dữ liệu bao gồm cơ sở dữ liệu trên nhiều server và client. Do đó, client phải hỗ trợ các thành phần phần mềm quản lý dữ liệu cũng như các ứng dụng và các thành phần GUI.

9.1.1.3. Các nguyên tắc cho các thành phần ứng dụng phân tán

Các nguyên tắc sau được áp dụng trong phân phối các thành phần phân tán giữa client và server:

- Các thành phần trình diễn/tương tác thường được đặt trên client. Sự sẵn có của môi trường windows và sức mạnh của máy tính yêu cầu giao diện người dùng đồ họa làm cho cách tiếp cận này hiệu quả hơn.
- Nếu cơ sở dữ liệu được chia sẻ bởi nhiều người sử dụng mạng LAN, cơ sở dữ liệu thường được đặt ở server. Hệ thống quản lý cơ sở dữ liệu và khả năng truy cập dữ liệu thường được đặt ở server cùng với cơ sở dữ liệu vật lý.
- Các dữ liệu tĩnh được dùng để tham chiếu nên được đặt ở client.

9.1.1.4. Liên kết các thành phần phần mềm C/S

Có nhiều kỹ thuật được dùng để liên kết các thành phần của cấu trúc client/server. Các cấu trúc này được kết hợp chặt chẽ trong mạng, trong cấu trúc hệ điều hành và chuyển đến người dùng ở phía client. Các kỹ thuật phổ biến nhất bao gồm:

- *Đường ống:* Được sử dụng rộng rãi trong các hệ thống UNIX, đường ống cho phép thông điệp giữa các máy khác nhau chạy trên các hệ điều hành khác nhau.
- *Gọi thủ tục từ xa:* Cho phép một tiến trình gọi một sự thực thi của một tiến trình khác hoặc một module khác nằm trên máy khác.
- *Tương tác client/server SQL:* Sử dụng để chuyển các yêu cầu SQL và các dữ liệu từ một thành phần (thường là client) tới một thành phần khác (thường là hệ quản trị cơ sở dữ liệu (DBMS) hay server). Kỹ thuật này chỉ áp dụng trong các ứng dụng hệ quản trị cơ sở dữ liệu từ xa (DBMS).

9.1.2. Công nghệ phần mềm cho các hệ thống C/S

9.1.2.1. Vấn đề mô hình hoá yêu cầu

Các hoạt động yêu cầu cho các hệ thống C/S chỉ khác biệt nhỏ so với các phương pháp mô hình hoá phân tích áp dụng cho các cấu trúc máy tính thông dụng. Do đó, có thể áp dụng các nguyên tắc phân tích đã trình bày ở trên cho các hệ C/S. Bởi vì mô hình hoá phân tích tránh đặc tả chi tiết, nó chỉ là một sự chuyển đổi được tạo ra cho thiết kế liên quan đến phân phối các thành phần phần mềm giữa client và server. Tuy nhiên, cách tiếp cận tiên hoá cho công nghệ phần mềm được áp dụng cho các hệ thống C/S, thực hiện quyết định trên toàn bộ hướng tiếp cận C/S (như fat server, fat client) có thể được tạo ra trong phân tích và thiết kế.

9.1.2.2. Thiết kế cho hệ thống C/S

Khi phần mềm được phát triển để sử dụng cho các cấu trúc máy tính cụ thể, hướng thiết kế phải xem xét đến các môi trường xây dựng cụ thể. Thiết kế phải “tùy biến hoá” để thích hợp với cấu trúc phần cứng.

Khi phần mềm được thiết kế để sử dụng kiến trúc client/server, hướng thiết kế phải “tùy biến hoá” để thích hợp với :

- Thiết kế dữ liệu: để sử dụng hiệu quả khả năng của hệ quản trị cơ sở dữ liệu từ xa (RDBMS), hoặc hệ quản trị cơ sở dữ liệu hướng đối tượng (OODBMS), thiết kế dữ liệu trở nên quan trọng hơn bất kỳ ứng dụng đã có nào khác.
- Khi các mô thức hướng sự kiện được chọn, mô hình động thái và các hoạt động phân tích phải được điều khiển và hướng điều khiển trong các mô hình động thái phải được dịch sang mô hình thiết kế.
- Các thành phần giao diện người dùng/trình diễn của hệ thống C/S thực hiện tất cả các chức năng liên kết với giao diện người dùng đồ hoạ (GUI). Do đó, thiết kế giao diện trở nên quan trọng.
- Các khung nhìn hướng đối tượng của thiết kế thường được chọn, thay vì cấu trúc tuần tự trong các ngôn ngữ thủ tục, cấu trúc đối tượng được cung cấp với các liên kết giữa các sự kiện khởi tạo tại GUI và các sự kiện xử lý chức năng trong các phần mềm dựa trên client.

a) Cách tiếp cận thiết kế thông thường (DFD)

Trong hệ thống client/server, biểu đồ DFD được dùng để thiết lập phạm vi của hệ thống, xác định các chức năng mức cao và cho phép phân tích các chức năng cấp cao. Trong khởi đầu của cách tiếp cận DFD truyền thống, phân tích dừng lại ở mức các quy trình nghiệp vụ cơ bản hơn là tiếp tục tới mức các quy trình nguyên tố.

Trong ngữ cảnh C/S, một quy trình nghiệp vụ cơ bản (EBP) có thể được xác định như một tập các nhiệm vụ được thực hiện mà không bị bắt cứ người dùng ở phía client nào ngắt.

Sơ đồ thực thể liên kết ERD được dùng để phân tích các kho dữ liệu của DFD, để thiết lập một khung nhìn cơ sở dữ liệu mức cao sử dụng RDBMS. ERD có vai trò cung cấp cấu trúc để xác định các đối tượng nghiệp vụ mức cao.

Thay vì được sử dụng như một công cụ phân tích chức năng, lược đồ cấu trúc được sử dụng như một biểu đồ lắp ráp để chỉ ra các thành phần trong giải pháp cho quy trình nghiệp vụ cơ bản. Các thành phần này, chứa các đối tượng giao diện, đối tượng ứng dụng, đối tượng cơ sở dữ liệu thiết lập cách dữ liệu được xử lý.

b) Thiết kế cơ sở dữ liệu

Thiết kế cơ sở dữ liệu được sử dụng để xác định và mô tả cấu trúc của các đối tượng nghiệp vụ được dùng trong hệ thống client/server. Phân tích yêu cầu xác định các đối tượng nghiệp vụ được thực hiện bằng cách sử dụng các phương pháp công nghệ thông tin đã trình bày trong các phần trên. Các ký pháp mô hình phân tích như ERD được dùng để xác định đối tượng nghiệp vụ, nhưng các kho dữ liệu nên được thiết lập để chứa đựng các thông tin thêm vào mà không được tư liệu hoá đầy đủ khi sử dụng các ký pháp như ERD.

Trong các kho, một đối tượng nghiệp vụ được xác định. Mỗi đối tượng (thực thể) xác định trong ERD được mở rộng trong suốt thiết kế thành cấu trúc dữ liệu (như file và các trường liên quan); tất cả các file, mối quan hệ giữa các mục dữ liệu trong một bản ghi của file, các luật hợp lệ cho các mối

quan hệ và các quy tắc nghiệp vụ mô tả khung nhìn ngoài của xử lý xảy ra với dữ liệu.

Thông tin thiết kế phải được phát triển trong khi thiết kế cơ sở dữ liệu bằng cách sử dụng các cơ sở dữ liệu quan hệ, thông tin đó được lưu trong các kho dữ liệu. Các bảng riêng được dùng để xác định các thông tin sau cho cơ sở dữ liệu client/server:

- *Thực thể*: Được xác định trong các ERD cho hệ thống mới.
- *File*: Thực hiện các thực thể xác định trong ERD.
- *Mối quan hệ file – to – file*: Thiết lập cách bố trí file bằng cách xác định các trường trong file.
- *Trường*: Xác định các trường trong thiết kế.
- *Mối quan hệ file – to – field*: Xác định các file liên hệ có thể được nối để tạo thành các khung nhìn logic hoặc các truy vấn.
- *Hợp lệ mối quan hệ*: Xác định các kiểu file – to – file và file – to – field sử dụng cho hợp lệ.
- *Kiểu trường*: Cho phép thừa kế đặc tính của trường từ các trường superclass (ngày tháng, văn bản, giá trị, giá cả).
- *Kiểu dữ liệu*: Đặc tính của dữ liệu trong trường.
- *Kiểu file*: Sử dụng để xác định vị trí của file.
- *Chức năng trường*: khoá, khoá ngoại, thuộc tính, trường ảo, trường thừa kế.
- *Giá trị cho phép*: Xác định các giá trị cho phép cho trường.
- *Luật nghiệp vụ*: Luật để soạn thảo, tính toán các file thừa kế.

c) Tổng quan về cách tiếp cận thiết kế

Porter đưa ra một tập các bước thiết kế quy trình nghiệp vụ cơ bản liên kết các thành phần của thiết kế thông thường với các thành phần thiết kế hướng đối tượng. Nó cũng giả định rằng, các mô hình yêu cầu xác định các đối tượng nghiệp vụ đã được phát triển và cải tiến từ trước để bắt đầu thiết kế các quy trình nghiệp vụ cơ bản. Các bước sau dùng để thừa kế thiết kế:

1. Với các quy trình nghiệp vụ cơ bản, xác định các file được tạo ra, cập nhật, tham chiếu hoặc xoá.

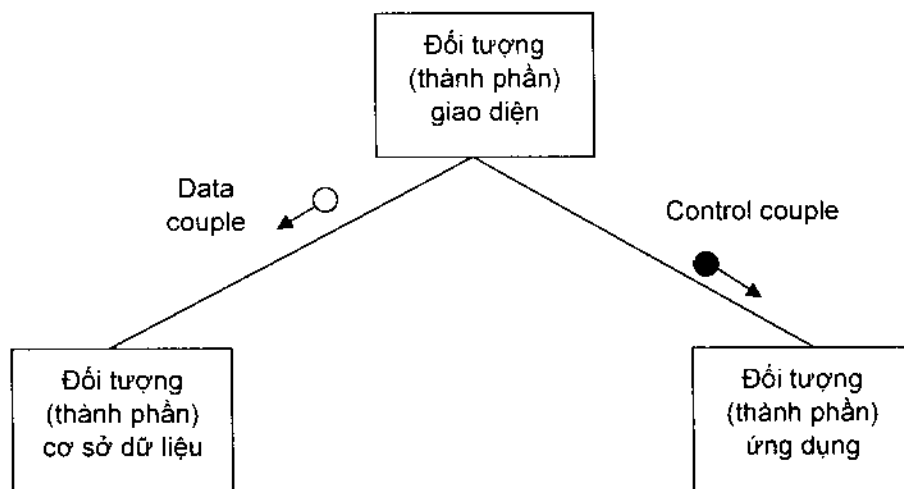
2. Sử dụng các file được xác định trong bước 1 làm cơ sở để xác định các thành phần hoặc các đối tượng.

3. Với mỗi thành phần, lấy các luật nghiệp vụ hoặc các thông tin đối tượng nghiệp vụ khác được thiết lập trong các file thích hợp.

4. Quyết định luật nào thích hợp với quy trình, và phân tích các luật xuống mức mô hình phương thức.

5. Với các yêu cầu, xác định bất kỳ thành phần nào cần thiết để thực hiện phương thức.

Porter cũng đưa ra ký pháp lược đồ cấu trúc như hình 9.3 để miêu tả cấu trúc thành phần của các quy trình nghiệp vụ cơ bản.



Hình 9.3. Ký pháp lược đồ cấu trúc các thành phần C/S

Trong hình 9.3 có 5 ký hiệu khác nhau được đưa ra:

- **Đối tượng giao diện:** Còn được gọi là thành phần tương tác/trình diễn người dùng. Nó thường được xây dựng trên một file đơn hoặc một file đơn và các file liên quan được liên kết qua truy vấn. Nó bao gồm phương thức để định dạng giao diện GUI và các ứng dụng phía client liên kết với các điều khiển trên giao diện. Nó cũng bao gồm các lệnh SQL nhúng, mô tả xử lý dữ liệu trên file cơ sở với giao diện được xây dựng. Nếu ứng dụng logic được liên kết với đối tượng giao diện được hoàn thành trên server, thông qua việc sử

dùng các công cụ middleware, ứng dụng logic thực hiện trên server có thể được xác định như một đối tượng ứng dụng riêng biệt.

- *Đối tượng dữ liệu*: Dùng để xử lý dữ liệu như tạo ra hay lựa chọn một bản ghi.
- *Đối tượng ứng dụng*: Cả đối tượng giao diện và đối tượng cơ sở dữ liệu đều sử dụng. Thành phần này được cả database trigger và thủ tục từ xa gọi. Nó cũng được dùng để xác định các logic nghiệp vụ thông thường liên kết với quy trình giao diện được chuyển đến server để thực hiện.
- *Data couple*: Khi một đối tượng gọi một đối tượng độc lập khác, một thông điệp được chuyển giữa hai đối tượng, ta sử dụng ký hiệu data couple.
- *Control couple*: Khi một đối tượng gọi một đối tượng độc lập khác và không có dữ liệu được chuyển giữa hai đối tượng, ta sử dụng ký hiệu control couple.

d) Quá trình thiết kế

Kho thiết kế được sử dụng để mô tả các đối tượng nghiệp vụ mô tả giao diện, ứng dụng và đối tượng dữ liệu. Các thực thể sau được xác định:

- *Phương thức*: Mô tả làm cách nào để một luật nghiệp vụ được thực hiện.
- *Quy trình cơ bản*: Xác định các quy trình nghiệp vụ cơ bản trong mô hình phân tích.
- *Liên kết quy trình/thành phần*: Bảng này xác định các thành phần tạo nên các giải pháp cho các quy trình nghiệp vụ cơ bản.
- *Các thành phần*: Mô tả các thành phần được chỉ ra trong lược đồ cấu trúc.
- *Luật nghiệp vụ/liên kết thành phần* : Xác định các thành phần quan trọng để thực hiện một luật nghiệp vụ được đưa ra.

9.1.3. Vấn đề kiểm thử

Hệ thống client/server phân tán thường đưa ra một tập các vấn đề cho những nhà kiểm thử phần mềm. Binder đã gợi ý cần chú ý các lĩnh vực sau:

- Xem xét client GUI.

- Xem xét môi trường đích và đa dạng nền.
- Xem xét cơ sở dữ liệu phân tán.
- Xem xét quy trình phân tán.
- Các quan hệ hiệu năng không tuyến tính.

9.1.3.1. Chiến lược kiểm thử C/S toàn bộ

Nói chung, kiểm thử phần mềm client/server diễn ra theo 3 mức:

1. Các ứng dụng client riêng lẻ được kiểm thử theo kiểu “disconnected” – các hoạt động của server và mạng không được xem xét.
2. Phần mềm client và các ứng dụng liên kết phía server được kiểm thử phối hợp, nhưng các hoạt động mạng không được kiểm thử rõ ràng.
3. Kiến trúc C/S hoàn chỉnh, bao gồm các hoạt động mạng và hiệu năng được kiểm thử.

Mặc dù có nhiều kiểu kiểm thử tương ứng với mỗi mức trên, nhưng nên sử dụng các cách tiếp cận kiểm thử sau cho các ứng dụng C/S:

- *Kiểm thử chức năng ứng dụng:* Chức năng của các ứng dụng client được kiểm thử sử dụng các phương pháp đã đề cập ở trên. Các ứng dụng được kiểm thử riêng biệt để cố gắng khắc phục các lỗi trong hoạt động.
- *Kiểm thử server:* Các chức năng quản lý dữ liệu và điều phối của server được kiểm thử. Hiệu năng của server (thời gian đáp ứng và thông lượng dữ liệu) cũng được xem xét.
- *Kiểm thử cơ sở dữ liệu:* Tính chính xác và toàn vẹn của dữ liệu được lưu trên server được kiểm thử. Giao dịch do các ứng dụng client gửi được kiểm thử để bảo đảm dữ liệu được lưu trữ và cập nhật chính xác.
- *Kiểm thử giao dịch:* Một loạt các kiểm thử được tạo ra để đảm bảo rằng mỗi lớp giao dịch được xử lý theo yêu cầu. Kiểm thử tập trung vào tính đúng đắn của xử lý và cả các vấn đề hiệu năng.
- *Kiểm thử truyền thông mạng:* Kiểm thử việc truyền thông giữa các node của mạng chính xác và gửi thông điệp, truyền thông không lỗi. Kiểm thử bảo mật mạng cũng được coi như một phần của kiểm thử này.

- *Kiểm thử cấu hình:* Kiểm thử hệ thống trong mọi môi trường phần cứng và phần mềm mà hệ thống hoạt động. Kiểm thử thích hợp bảo đảm giao diện nhất quán về chức năng giữa các nền phần cứng và phần mềm. Gartner Group đã đưa ra một mẫu kế hoạch kiểm thử như sau:
 - 1.0 – Kiểm thử Windows (GUI).
 - 1.1 – Xác định các kịch bản nghiệp vụ.
 - 1.2 – Tạo các trường hợp thử.
 - 1.3 – Kiểm tra.
 - 1.4 – Các công cụ kiểm thử.
 - 2.0 – Server.
 - 2.1 – Tạo các dữ liệu kiểm thử.
 - 2.2 – Kiểm thử khối lượng.
 - 2.3 – Kiểm tra.
 - 2.4 – Các công cụ kiểm thử.
 - 3.0 – Ghép nối.
 - 3.1 – Hiệu năng.
 - 3.2 – Kiểm thử khối lượng.
 - 3.3 – Kiểm tra.
 - 3.4 – Các công cụ kiểm thử.
 - 4.0 – Chất lượng kỹ thuật.
 - 4.1 – Xác định.
 - 4.2 – Xác định khiếm khuyết.
 - 4.3 – Độ đo.
 - 4.4 – Chất lượng mã.
 - 4.5 – Các công cụ kiểm thử.
 - 5.0 – Kiểm thử chức năng.
 - 5.1 – Xác định.

- 5.2 – Tập dữ liệu kiểm thử.
- 5.3 – Kiểm tra.
- 5.4 – Các công cụ kiểm thử.
- 6.0 – Kiểm thử hệ thống.
- 6.1 – Xác định.
- 6.2 – Kiểm thử tính sử dụng.
- 6.3 – Giám sát thoả mãn người dùng.
- 6.4 – Kiểm tra.
- 6.5 – Các công cụ kiểm thử.
- 7.0 – Quản lý kiểm thử.
- 7.1 – Đội kiểm thử.
- 7.2 – Lịch trình kiểm thử.
- 7.3 – Tài nguyên yêu cầu.
- 7.4 – Phân tích kiểm thử, báo cáo, và các kỹ thuật theo vết.

9.2. Công nghệ phần mềm có sự trợ giúp của máy tính (CASE)

9.2.1. CASE là gì?

Trước năm 1980, các kỹ sư phần mềm chưa có các công cụ trợ giúp tự động hoá quy trình xây dựng phần mềm. Từ đầu những năm 1980, một lĩnh vực về công cụ hỗ trợ phát triển phần mềm bắt đầu phát triển. Thuật ngữ CASE được hiểu là sự tự động hoá quy trình công nghệ phần mềm. Năm 1995, các nhà cung cấp CASE đưa ra hàng trăm sản phẩm khác nhau. Mặc dù chưa được như mong muốn nhưng chúng đã trở thành những công cụ quan trọng trong trợ giúp các kỹ sư phần mềm.

CASE Tools trợ giúp các nhà quản lý công nghệ phần mềm và những người thực hiện trong các hoạt động của quy trình phần mềm. CASE Tools tự động hoá các hoạt động, quản lý mọi công việc tạo ra sản phẩm trong

quy trình. CASE Tools hỗ trợ các kỹ sư trong phân tích, thiết kế, mã hoá và kiểm thử.

Vậy ai là người dùng CASE Tools? Đó là các nhà quản lý dự án và các kỹ sư phần mềm.

Một xưởng kỹ nghệ phần mềm được coi là tốt cho công việc của một người thợ nói chung và của kỹ sư phần mềm nói riêng phải có 3 đặc tính sau:

1. Có đầy đủ các công cụ hỗ trợ trong mọi công việc để làm ra sản phẩm.
2. Tổ chức, sắp xếp một cách hợp lý để khi cần là tìm được nhanh.
3. Người thợ khéo tay biết sử dụng Tools thành thạo và có hiệu quả. Các kỹ sư phần mềm cần có một xưởng kỹ nghệ phần mềm có nhiều tools và được tổ chức tốt.

9.2.2. Vai trò của CASE Tools

CASE Tools làm giảm một số lớn các phép thử mà một sản phẩm phải đáp ứng. Chúng cung cấp một cách nhìn mới về công nghệ phần mềm, hỗ trợ việc đưa ra các quyết định tốt hơn và đảm bảo chất lượng phần mềm cao hơn.

CASE Tools được dùng kết hợp với mô hình quy trình (process model) được chọn. Nếu có được tập tools đầy đủ, CASE sẽ được dùng cho mọi bước của quy trình công nghệ phần mềm.

Chúng ta dùng Tools để bổ sung việc định hình các công việc thuộc công nghệ phần mềm chứ không thay thế chúng.

CASE Tools chỉ thực sự mang lại lợi ích nếu trước khi sử dụng, khung công nghệ phần mềm phải được thiết lập, các khái niệm và các phương pháp phải được coi trọng.

9.2.3. Các khối cấu thành CASE

CASE có thể là một công cụ đơn giản và đơn lẻ, nó hỗ trợ một hoạt động công nghệ phần mềm đặc biệt hoặc một môi trường bao gồm các công cụ, cơ sở dữ liệu, con người, phần cứng, mạng, hệ điều hành, các chuẩn và vô số thành phần khác.

Các khối xây dựng cho CASE được minh hoạ như hình 9.4.



Hình 9.4. Các khối cấu thành CASE

Mỗi khối xây dựng tạo nên nền tảng cho khối tiếp theo, với các công cụ ở trên đỉnh. Cần lưu ý là nền tảng cho các môi trường CASE hiệu quả thì tương đối ít liên quan đến các công cụ kỹ nghệ ; những môi trường thành công cho kỹ nghệ phần mềm lại được xây dựng trên kiến trúc môi trường bao gồm phần cứng thích hợp và phần mềm hệ thống. Bên cạnh đó, kiến trúc môi trường phải xét cả mẫu công việc, con người trong tiến trình công nghệ phần mềm.

Trong những năm 1960, 1970, 1980 việc phát triển phần mềm được thực hiện trên các máy tính lớn. Thiết bị cuối được nối với máy tính trung tâm và mỗi người đều dùng chung tài nguyên của máy tính đó. Các công cụ phần mềm có sẵn đã được thiết kế để vận hành trong môi trường phân chia thời gian dựa trên thiết bị cuối.

Ngày nay, hướng phát triển phần mềm đã mở rộng hơn và hướng trọng tâm làm việc xem như nền kỹ nghệ phần mềm. Từng trạm làm việc riêng được nối mạng giúp kỹ sư phần mềm có thể trao đổi hiệu quả. Giờ đây, cơ sở dữ liệu dự án có sẵn qua bộ phục vụ tệp mạng, có thể thâm nhập tới mọi trạm làm việc. Hệ điều hành hỗ trợ cho phần cứng, mạng và các công cụ nối lại thành môi trường.

Kiến trúc môi trường cứng với nền phần cứng và các hỗ trợ hệ thống (phần mềm mạng, quản trị cơ sở dữ liệu, các dịch vụ quản lý đối tượng) đặt nền tảng cho CASE.

Nhưng bản thân môi trường CASE lại đòi hỏi các khối xây dựng khác. Các dịch vụ khả chuyển cung cấp cầu nối giữa các công cụ CASE và khung hợp nhất của chúng với kiến trúc môi trường.

Khung hợp nhất là tập các chương trình đặc biệt, có thể là các CASE Tools riêng lẻ nối kết với nhau để tạo ra một cơ sở dữ liệu đối tượng và để đưa ra một cách nhìn như với người sử dụng cuối.

Các dịch vụ khả chuyển cho phép CASE Tools và các khung hợp nhất của chúng dễ dàng chuyển làm việc với các nền phần cứng và các hệ điều hành khác nhau mà không cần sự bảo trì thích nghi nào.

Trong hình 9.4, các khối trong hình trên biểu thị một cơ sở toàn diện cho việc hợp nhất các CASE Tools. Tuy nhiên, phần lớn các công cụ CASE đang dùng hiện nay vẫn chưa được xây dựng với tất cả các khối xây dựng đã được mô tả ở trên. Trong thực tế, đại đa số các công cụ CASE đều là các “giải pháp điểm”. Tức là, một công cụ được dùng để trợ giúp cho một hoạt động kỹ nghệ phần mềm đặc biệt (như mô hình hoá phân tích), chưa trực tiếp truyền thông với các công cụ khác, chưa được gắn với cơ sở dữ liệu dự án và không phải là một phần của môi trường CASE hợp nhất (I-CASE).

9.2.4. Phân loại các công cụ CASE

CASE Tools có thể phân loại theo các tiêu chí sau:

- Theo chức năng, theo vai trò của chúng như một công cụ cho người quản lý hoặc người làm kỹ thuật.
- Theo sự sử dụng trong các bước khác nhau của quy trình phần mềm.
- Theo nguồn gốc hoặc giá thành.

Ngoài ra còn các yếu tố phân loại khác như :

- *Hỗ trợ quy trình*: Pha quy trình nào được hỗ trợ bởi công nghệ CASE Tools? Có thể phân loại thành tools thiết kế, tools lập trình, tools bảo trì...
- *Phạm vi hỗ trợ*: Phạm vi hoạt động được hỗ trợ bởi công nghệ. Hỗ trợ phạm vi hẹp là hỗ trợ cho các nhiệm vụ đặc biệt trong quy trình như tạo sơ đồ liên kết thực thể, dịch chương trình... Hỗ trợ phạm vi rộng – hỗ trợ cho các pha của quy trình như thiết kế, hỗ trợ bao trùm tất cả hoặc phân lớp các pha của quy trình phần mềm.

Phần sau sẽ trình bày phân loại theo chức năng.

9.2.5. Công cụ lập kế hoạch hệ thống

Bằng cách mô hình hoá các yêu cầu thông tin chiến lược của một tổ chức, các công cụ lập kế hoạch hệ thống kinh doanh đưa ra một siêu mô hình để từ đó dẫn ra hệ thống thông tin chuyên dụng. Thay vì tập trung vào các yêu cầu của một ứng dụng đặc biệt, thông tin được mô hình hoá khi nó chuyển qua giữa nhiều thực thể tổ chức khác nhau bên trong một công ty. Mục đích chính cho những công cụ trong phân loại này là để nâng cao hiểu biết về cách di chuyển thông tin giữa nhiều đơn vị tổ chức.

Điều quan trọng cần lưu ý là các công cụ lập kế hoạch kinh doanh không dành cho mọi tổ chức. Chúng đòi hỏi một cam kết chính về tài nguyên và của cấp quản lý để tạo ra một mô hình đầy đủ rồi hoạt động theo thông tin suy ra từ đó. Những công cụ đó đưa ra cách nhìn bản chất khi chiến lược hệ thống tin đã được xây dựng và khi hệ thống cũng như phương pháp hiện tại không đáp ứng cho nhu cầu của tổ chức.

9.2.6. Công cụ quản lý dự án

Ngày nay, phần lớn các công cụ quản lý dự án CASE đều tập trung vào một phần tử xác định của việc quản lý dự án, thay vì đưa ra sự hỗ trợ bao quát cho hoạt động quản lý. Bằng cách dùng một tập các công cụ CASE có lựa chọn, người quản lý dự án có thể đưa ra các ước lượng hữu ích về nỗ lực, chi phí và thời hạn của dự án phần mềm, phân chia công việc và lập kế hoạch lịch dự án làm việc, theo dõi dự án một cách liên tục. Bên cạnh đó, người quản lý có thể dùng các công cụ để thu thập độ đo cuối cùng, từ đó đưa ra chỉ dẫn về hiệu suất phát triển phần mềm và chất lượng sản phẩm. Với những nhà quản lý có trách nhiệm phát triển phần mềm theo hợp đồng thì các công cụ CASE đều có sẵn để theo dõi các yêu cầu từ các đòi hỏi, đề nghị (RFP) của khách hàng.

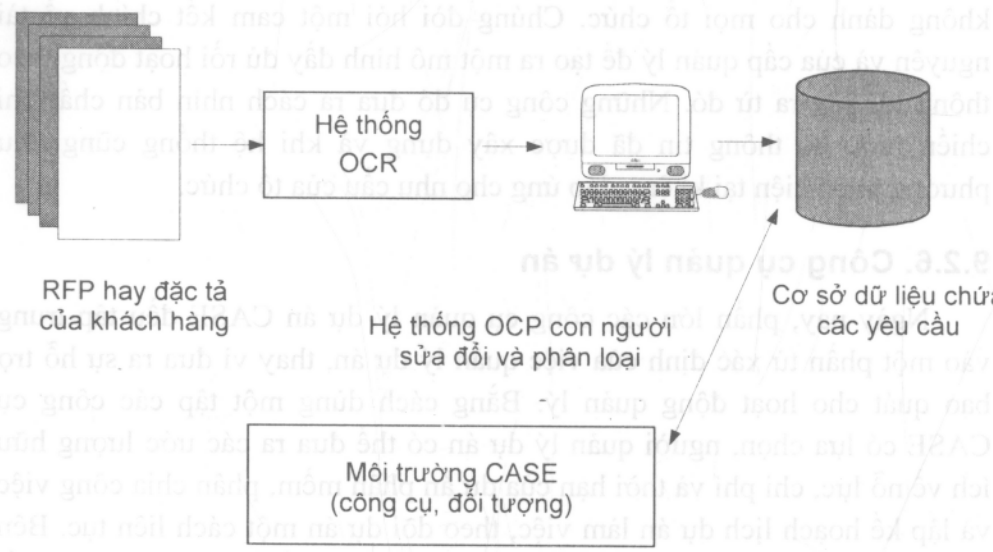
9.2.6.1. Các công cụ lập kế hoạch dự án

Công cụ trong phạm vi này hội tụ vào hai lĩnh vực chính: Ước lượng nỗ lực, chi phí dự án phần mềm và lập lịch dự án. Các công cụ ước lượng chi phí cho phép người quản lý dự án ước lượng kích cỡ của dự án bằng việc dùng phép đo gián tiếp (như số dòng lệnh và điểm chức năng) và mô tả các đặc trưng của dự án (như độ phức tạp của vấn đề, kinh nghiệm của đội

ngũ, độ chín muồi của tiến trình). Công cụ ước lượng sẽ tính ra nỗ lực được ước lượng, thời hạn dự án và số người cần thiết.

Các công cụ lập lịch dự án giúp xác định được tất cả các nhiệm vụ dự án (cấu trúc phân việc), tạo ra một mạng nhiệm vụ (thông thường dùng đầu vào đồ hoạ), biểu diễn sự phụ thuộc lẫn nhau giữa các nhiệm vụ và mô hình hoá khối lượng các khả năng song song cho dự án. Phần lớn các công cụ đều dùng phương pháp lập lịch đường găng để xác định tác động của việc trượt theo ngày bàn giao.

9.2.6.2. Các công cụ theo dõi yêu cầu



Hình 9.5. Công cụ theo dõi yêu cầu

Khi phát triển các hệ thống lớn, hệ thống cần bàn giao có thể không đáp ứng hoàn toàn các yêu cầu người dùng đã xác định. Trong một số trường hợp, việc không tuân thủ các yêu cầu này sinh từ những khó khăn kỹ thuật. Nhưng trong các tình huống khác, các yêu cầu bị bỏ qua vì chúng đơn thuần không được đề cập đến.

Mục đích các công cụ theo dõi yêu cầu là đưa ra một cách tiếp cận có hệ thống tới việc cô lập các yêu cầu, bắt đầu với yêu cầu cá nhân (Request from Person – RFP) hay đặc tả của khách hàng. Công cụ theo dõi yêu cầu diễn hình tổ hợp cả ước lượng văn bản tương tác con người với hệ thống quản trị cơ sở dữ liệu, nơi vốn lưu trữ và phân loại từng yêu cầu hệ thống đã được rút ra từ “phân tích câu” trong RFP hay đặc tả gốc. Việc phân tích

câu cho các yêu cầu có thể đơn giản như việc tìm mọi sự xuất hiện của động từ “sẽ” (chỉ ra một yêu cầu) và làm sáng tỏ câu có chứa “sẽ”. Sau đó người phân tích phân loại các yêu cầu hàm chứa trong câu và đưa vào trong cơ sở dữ liệu. Công việc phát triển tiếp theo có thể được tham khảo chéo sang cơ sở dữ liệu để việc tuân thủ các yêu cầu được đảm bảo.

9.2.6.3. Công cụ độ đo và quản lý

Độ đo phần mềm làm tăng khả năng của người quản lý trong kiểm soát, phối hợp tiến trình kỹ nghệ phần mềm và khả năng của người thực hành nhằm nâng cao chất lượng phần mềm được tạo ra. Các công cụ độ đo và cách đo hiện nay tập trung vào các đặc trưng tiến trình và sản phẩm. Công cụ quản lý độ đo riêng cho dự án (như LOC/người-tháng, khiếm khuyết trên điểm chức năng), đưa ra một chỉ dẫn về hiệu suất hay chất lượng. Các công cụ kỹ thuật xác định độ đo kỹ thuật (như độ phức tạp xoay vòng), nêu cách nhìn rõ hơn về chất lượng của thiết kế hay mã. Phần lớn công cụ độ đo tiên tiến hơn duy trì cả một cơ sở dữ liệu các cách đo “trung bình công nghiệp”. Dựa trên các đặc trưng dự án và sản phẩm do người dùng nêu ra, những công cụ đó “tính tỷ lệ” số cục bộ so với trung bình công nghiệp và gợi ý chiến lược để cải thiện.

Các công cụ quản lý (ngoại trừ các công cụ ước lượng và lập lịch dự án) đều hỗ trợ người quản lý hệ thống thông tin trong việc định thứ tự ưu tiên nhiều dự án đang cạnh tranh nhau về tài nguyên phát triển hữu hạn. Bằng cách dùng các yêu cầu và độ ưu tiên của khách hàng, các ràng buộc của tổ chức phát triển, các rủi ro kỹ thuật và kinh doanh, những công cụ đó dùng cách tiếp cận hệ chuyên gia để gợi ý đưa ra trình tự tiến hành một dự án.

9.2.7. Công cụ hỗ trợ

Phạm trù công cụ hỗ trợ bao quát các công cụ hệ thống và ứng dụng bổ sung cho tiến trình công nghệ phần mềm. Các công cụ trong phạm trù rộng lớn này bao gồm các hoạt động bảo trợ, áp dụng được trong toàn bộ tiến trình công nghệ phần mềm. Chúng bao gồm các công cụ làm tư liệu, công cụ phần mềm hệ thống và mạng, công cụ đảm bảo chất lượng, công cụ quản lý cấu hình phần mềm và quản lý cơ sở dữ liệu (tính cả các thành phần phạm trù công cụ khung).

9.2.7.1. Công cụ làm tài liệu

Các công cụ sản xuất tài liệu và xuất bản tại bàn hỗ trợ gần như mọi khía cạnh của công nghệ phần mềm và biểu thị một cơ hội “đòn bẩy” chủ chốt đối với tất cả những người phát triển phần mềm. Phần lớn các tổ chức phát triển phần mềm đều dành một khối lượng thời gian chủ yếu để phát triển tài liệu, và trong nhiều trường hợp bản thân tiến trình làm tài liệu lại hoàn toàn không hiệu quả. Bởi vậy, công cụ làm tài liệu đem lại một cơ hội quan trọng làm tăng hiệu suất.

Các công cụ làm tài liệu thường gắn với các công cụ CASE khác qua cầu dữ liệu do người sản xuất công cụ kỹ thuật cài đặt. Chẳng hạn, một số công cụ phân tích và thiết kế có quan hệ với một hay nhiều hệ thống, các mô hình và văn bản được tạo ra trong khi phân tích, thiết kế đó có thể truyền sang một công cụ làm tài liệu và sau đó được nhúng vào trong đặc tả được tạo ra từ công cụ làm tài liệu này.

9.2.7.2. Công cụ phần mềm hệ thống

CASE là công cụ trạm làm việc. Do đó, môi trường CASE phải phù hợp với phần mềm hệ thống mạng chất lượng cao, thư điện tử, bản tin và những khả năng truyền thông khác, mặc dù hệ điều hành cho hầu hết các trạm làm việc kỹ nghệ là UNIX, các dịch vụ khả chuyên do IPSE đưa ra có thể làm cho CASE chuyển sang các hệ điều hành khác mà ít có trục trặc lớn.

9.2.7.3. Công cụ đảm bảo chất lượng

Tuy đại đa số các công cụ CASE tập trung vào đảm bảo chất lượng nhưng thực tế mới chỉ là các công cụ độ đo mà chúng kiểm toán chương trình gốc để xác định việc tuân thủ các ngôn ngữ. Các công cụ khác trích ra các độ đo kỹ thuật trong nỗ lực dự phòng chất lượng của phần mềm đang được xây dựng.

9.2.7.4. Cơ sở dữ liệu và công cụ SCM

Phần mềm quản lý cơ sở dữ liệu làm nền tảng cho việc thiết lập cơ sở dữ liệu (kho) CASE, được gọi là cơ sở dữ liệu dự án. Với việc nhấn mạnh vào các đối tượng cấu hình, các công cụ quản lý cơ sở dữ liệu cho CASE có thể tiến hoá từ các hệ quản trị cơ sở dữ liệu quan hệ (RDMS) thành các hệ quản trị cơ sở dữ liệu hướng đối tượng (OODMS).

Những người ủng hộ cho rằng, OODMS sẽ làm cho việc quản lý cấu hình dễ thực hiện hơn và biện minh rằng cấu trúc hướng đối tượng là cách tổ chức tự nhiên cho các khoản mục cấu hình phần mềm, thứ vẫn tổ hợp nhiều kiểu thông tin khác nhau. Những người ủng hộ RDMS lại cho rằng, RDMS có hiệu năng tốt hơn và nhiều kinh nghiệm công nghiệp hơn OODMS và biện minh rằng mô hình quan hệ thực hiện tốt nhất bằng cách dùng mô hình hướng đối tượng.

Các công cụ CASE có thể trợ giúp cho tất cả 5 nhiệm vụ SCM – xác định, kiểm soát bản, kiểm soát thay đổi, kiểm toán và kế toán trạng thái. Cơ sở dữ liệu CASE đưa ra một cơ chế để xác định từng khoản mục cấu hình và đặt nó quan hệ với các khoản mục khác; các công cụ truyền thông CASE có thể làm tăng chất lượng kế toán trạng thái. Việc quản lý cấu hình phần mềm nằm ở ngay lõi của mọi môi trường CASE. Bằng cách kiểm soát sự thay đổi cấu hình phần mềm, các công cụ SCM buộc con người phải có sự nhận thức với từng thay đổi, do đó làm giảm hiệu lực và làm tăng chất lượng.

Việc dùng cơ sở dữ liệu, các công cụ quản lý cấu hình và các công cụ duyệt chuyên dụng đưa ra bước đầu tiên hướng tới việc tạo ra một thư viện phần mềm khuyến khích tái sử dụng các thành tố phần mềm. Ngày nay, mặc dù việc tái sử dụng còn tương đối ít, CASE đưa ra hứa hẹn thực sự để đạt đến việc tái sử dụng rộng hơn cho các thành tố phần mềm máy tính.

9.2.8. Công cụ phân tích và thiết kế

Công cụ phân tích và thiết kế giúp cho kỹ sư phần mềm tạo ra mô hình hệ thống cần xây dựng. Mô hình này chứa luồng dữ liệu và điều khiển, nội dung dữ liệu, các biểu diễn tiến trình, đặc tả điều khiển và nhiều biểu diễn mô hình hoá khác. Các công cụ phân tích, thiết kế trợ giúp cho việc tạo ra mô hình và ước lượng chất lượng mô hình. Bằng cách kiểm tra tính nhất quán và hợp lệ trên mô hình, các công cụ phân tích, thiết kế cung cấp cho kỹ sư phần mềm một hiểu biết nào đó trong biểu diễn phân tích và giúp loại bỏ các lỗi trước khi chúng lan truyền vào thiết kế, hay tồi tệ hơn, vào bản thân việc cài đặt.

9.2.8.1. Các công cụ SA/SD

Phần lớn các công cụ phân tích và thiết kế đều cài đặt phương pháp phân tích có cấu trúc và thiết kế có cấu trúc (SA/SD). SA/SD là một kỹ thuật mô hình hoá giúp kỹ sư phần mềm tạo ra các mô hình phức tạp hơn về hệ thống, bắt đầu từ mức các yêu cầu và kết thúc với thiết kế kiến trúc. SA/SD tổ hợp một kỹ pháp riêng, trực cảm phân tích thiết kế, và một tiến trình biến đổi từ phân tích sang thiết kế biểu diễn khả thi của phần mềm.

9.2.8.2. Công cụ PRO/SIM

Các công cụ làm bản mẫu (PRO) và mô phỏng (SIM) cung cấp cho kỹ sư phần mềm khả năng tiên đoán hành vi của hệ thống thời gian thực trước lúc nó được xây dựng. Bên cạnh đó, nó còn giúp kỹ sư phần mềm phát triển được mô hình của hệ thống thời gian thực, cho phép khách hàng có được sự hiểu biết thấu đáo về chức năng, vận hành và đáp ứng trước khi cài đặt thực tế.

Phần lớn các công cụ PRO/SIM cung cấp cho kỹ sư phần mềm một phương tiện để tạo ra các mô hình hành vi và chức năng của hệ thống. Công cụ trong phạm trù này cung cấp một phương tiện để xác định các đặc trưng hiệu năng dự phòng của từng thành phần hệ thống (như tốc độ thực hiện của phần cứng hay chức năng phần mềm), xác định các đặc trưng dữ liệu vào và ra (như tỷ lệ dữ liệu vào hay các đặc trưng ngắt), và mô hình hoá tính nối được/giao diện giữa các phần tử hệ thống.

Nhiều công cụ PRO/SIM cung cấp khả năng sinh mã cho ngôn ngữ ADA và các ngôn ngữ lập trình khác. Bên cạnh đó, tất cả các công cụ trong phạm trù này đều dùng một ngôn ngữ đặc tả hình thức hay nửa hình thức nền tảng, mở ra cánh cửa tới nhiều bộ sinh mã nâng cao và bộ kiểm chứng hình thức cho đặc tả hệ thống.

9.2.8.3. Các công cụ thiết kế và phát triển giao diện

Các công cụ thiết kế và phát triển giao diện thực tế là một bộ công cụ các thành tố chương trình như nút, cấu trúc cửa sổ, biểu tượng, các thanh cuộn... Tuy nhiên, bộ công cụ này đang bị thay thế bởi công cụ làm bản mẫu giao diện cho phép tạo nhanh chóng trên màn hình những giao diện người dùng phức tạp tuân theo các chuẩn giao diện đã được chấp nhận cho phần mềm.

Hệ thống phát triển giao diện người dùng (UIDS) tổ hợp các công cụ CASE đơn lập cho tương tác người máy với một thư viện các thành tố chương trình giúp người phân tích xây dựng được giao diện người – máy một cách nhanh chóng. Một UIDS cung cấp các thành tố chương trình quản lý thiết bị vào, hợp lệ hoá đầu vào của người dùng, giải quyết các điều kiện lỗi, bỏ tiến trình và hoàn tác, cung cấp phản hồi ngược trực quan, lời nhắc, trợ giúp, cập nhật hiển thị, quản lý dữ liệu ứng dụng, giải quyết việc cuộn và soạn thảo, cách ly ứng dụng khỏi các chức năng quản lý màn hình và hỗ trợ các tính năng đáp ứng nhu cầu người dùng cuối.

9.2.8.4. Động cơ phân tích và thiết kế

Thế hệ các công cụ phân tích và thiết kế mới được gọi là động cơ phân tích và thiết kế, cấu trúc dựa trên quy tắc điều chỉnh công cụ theo bất kỳ phương pháp phân tích và thiết kế nào. Khi dùng các công cụ CASE tiên tiến này, phương pháp phân tích và thiết kế như SADT sẽ được hỗ trợ bởi việc xây dựng ký pháp đồ thị thích hợp, đưa vào các quy tắc quản lý ngữ nghĩa của ký hiệu và phát triển một giao diện hỗ trợ phân tích, thiết kế. Về bản chất, động cơ phân tích và thiết kế giúp kỹ sư phần mềm điều chỉnh được công cụ để đáp ứng các nhu cầu của một phương pháp đặc biệt.

9.2.9. Công cụ lập trình

Các công cụ lập trình bao gồm trình biên dịch, trình soạn thảo và trình gỡ lỗi đều có sẵn để hỗ trợ cho phần lớn các ngôn ngữ lập trình. Bên cạnh đó, môi trường lập trình hướng đối tượng (OO), ngôn ngữ thế hệ 4, bộ sinh ứng dụng và ngôn ngữ truy vấn cơ sở dữ liệu cũng nằm trong phạm trù này.

9.2.9.1. Công cụ mã hoá quy ước

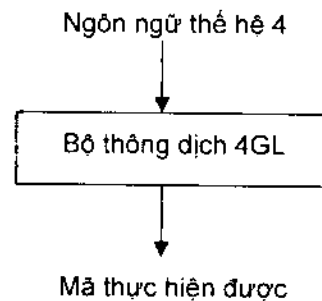
Trong gần 30 năm, công cụ duy nhất có sẵn cho người lập trình là các công cụ mã hoá quy ước và do đó, mọi vấn đề công nghệ phần mềm đều giống như vấn đề mã hoá. Ngày nay, các công cụ quy ước vẫn tiếp tục tồn tại trên tuyến trước của phân tích phần mềm, nhưng chúng được hỗ trợ bởi tất cả các công cụ CASE khác.

9.2.9.2. Công cụ mã hoá thế hệ thứ 4

Việc biểu diễn các ứng dụng phần mềm tại mức độ trừu tượng cao phát triển đã làm cho nhiều người chuyển sang các công cụ mã hoá thế hệ 4. Các

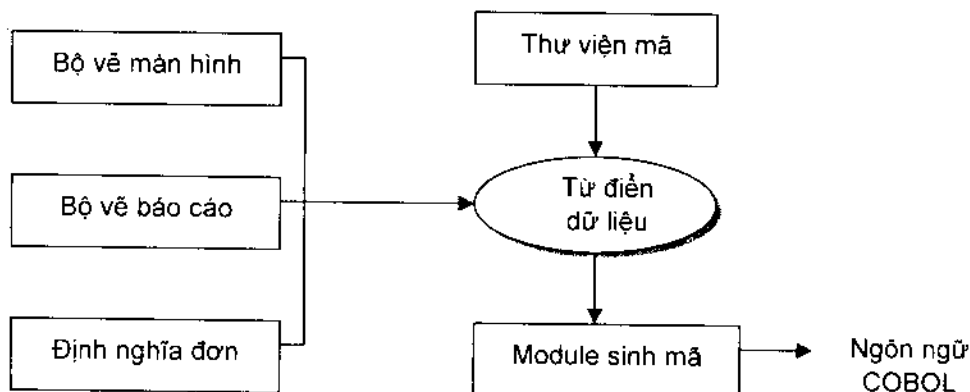
hệ thống truy vấn cơ sở dữ liệu, bộ sinh mã và ngôn ngữ thế hệ 4 đã làm thay đổi cách thức hệ thống được phân tích. Mục tiêu cuối cùng của CASE là sinh mã tự động, tức là biểu diễn hệ thống tại mức trừu tượng cao hơn các ngôn ngữ lập trình quy ước. Một cách lý tưởng, các công cụ sinh mã đó không chỉ dịch một mô tả hệ thống thành chương trình vận hành mà còn giúp kiểm chứng lại tính đúng đắn của đặc tả hệ thống sao cho kết quả đáp ứng với yêu cầu của người dùng.

Mặc dù ngôn ngữ thế hệ 4, bộ sinh mã và các bộ sinh ứng dụng (như hệ thống truy vấn cơ sở dữ liệu) đều giúp kỹ sư phần mềm xác định một cách hệ thống ở mức trừu tượng cao, nhưng mỗi công cụ này đều có sự khác nhau ở một số cách thức. Ngôn ngữ thế hệ thứ 4 là đầu vào trực tiếp của bộ thông dịch 4GL. Bộ thông dịch dịch 4GL thành mã thực hiện được.

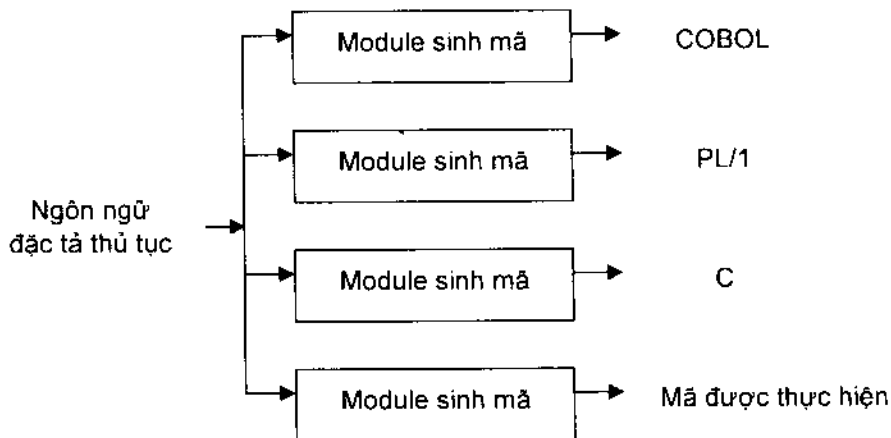


Hình 9.6a

Công cụ thế hệ thứ 4: 4GL



Hình 9.6b. Công cụ thế hệ thứ 4 : Bộ sinh mã



Hình 9.6c. Công cụ thể hệ thứ 4 : Bộ sinh ứng dụng

9.2.9.3. Các công cụ lập trình hướng đối tượng

Lập trình hướng đối tượng là một trong những công cụ “nóng nhất” của công nghệ phần mềm. Bởi lý do này, các nhà sản xuất CASE đang đẩy mạnh những công cụ mới để phát triển phần mềm hướng đối tượng ra thị trường.

Các môi trường hướng đối tượng được gắn chặt với một ngôn ngữ lập trình xác định. Một môi trường OO điển hình tổ hợp các tính năng giao diện thể hệ thứ ba (chuột, cửa sổ, trình đơn kéo xuống, đa nhiệm) với các chức năng chuyên dụng như trình duyệt – một chức năng giúp kỹ sư phần mềm xem xét được tất cả các đối tượng có chứa trong bất kỳ thư viện đối tượng nào để xác định xem liệu cái nào có thể tái sử dụng trong ứng dụng hiện hành.

9.2.10. Công cụ tích hợp và kiểm thử

Trong thư mục các công cụ kiểm thử phần mềm, kỹ nghệ chất lượng phần mềm xác định các phạm trù công cụ kiểm thử sau:

- *Thu nhận dữ liệu* : Các công cụ thu thập dữ liệu được dùng trong khi kiểm thử.
- *Cách đo tĩnh* : Các công cụ phân tích chương trình gốc mà không thực hiện trường hợp kiểm thử.
- *Cách đo động* : Các công cụ phân tích chương trình gốc trong khi thực hiện.

- *Mô phỏng* : Các công cụ mô phỏng chức năng của phần cứng hay phần mềm.
- *Quản lý kiểm thử* : Các công cụ trợ giúp cho việc lập kế hoạch, phát triển và kiểm soát kiểm thử.
- *Công cụ chéo chức năng* : Các công cụ xuyên qua biên giới của các phạm trù trên.

9.2.10.1. Công cụ phân tích tĩnh

Các công cụ phân tích tĩnh hỗ trợ cho kỹ sư phần mềm đưa ra các trường hợp kiểm thử. Có 3 kiểu công cụ thử tĩnh đang được dùng trong công nghiệp: Công cụ kiểm thử dựa trên mã, ngôn ngữ kiểm thử chuyên dụng và công cụ kiểm thử dựa trên yêu cầu.

Công cụ kỹ thuật dựa trên mã chấp nhận chương trình gốc làm đầu vào và thực hiện một số phân tích để sinh ra trường hợp kiểm thử. Bằng cách dùng một mô tả cho đầu vào chương trình và thiết kế thủ tục là hướng dẫn, công cụ kiểm thử tĩnh dẫn ra các trường hợp kiểm thử bằng cách dùng các tiêu chuẩn gom đường, kiểm thử điều kiện và luồng dữ liệu.

Các ngôn ngữ kiểm thử chuyên dụng cho phép kỹ sư phần mềm viết các đặc tả kiểm thử chi tiết để mô tả cho từng trường hợp kiểm thử và các vấn đề để thực hiện nó sau đó. Tuy nhiên, những công cụ như vậy không giúp người kiểm thử trong việc thiết kế các trường hợp kiểm thử.

Các công cụ kiểm thử dựa trên yêu cầu cô lập các yêu cầu người dùng và gợi ý trường hợp kiểm thử (hay lớp các kiểm thử) sẽ thực hiện qua các yêu cầu. Để làm việc cho đúng, các công cụ trong phạm trù con này phải thâm nhập vào đặc tả hình thức cho phần mềm.

Trong phần lớn các trường hợp, các công cụ kiểm thử tĩnh sẽ làm tư liệu và thư mục cho các phép kiểm thử (như kiểm thử để thực thi một kiểu đầu vào đặc biệt). Chúng sẽ tiến hành so sánh đầu ra kiểm thử để lưu ý những khác biệt giữa kết quả trông đợi và thực tế.

9.2.10.2. Công cụ phân tích động

Các công cụ phân tích động tương tác với một chương trình đang thực hiện, kiểm tra việc gom đường, các khẳng định kiểm thử về giá trị của biến xác định và các phối hợp khác cho luồng thực hiện của chương trình.

Các công cụ động có thể hoặc xâm nhập vào hoặc không xâm nhập vào chương trình. Công cụ xâm nhập làm thay đổi phần mềm được kiểm thử bằng việc chen thêm các đầu dò (các lệnh phụ) thực hiện những hoạt động đã nói trên. Công cụ kiểm thử không xâm nhập dùng một bộ xử lý phân cứng tách biệt chạy song song với bộ xử lý đang chứa chương trình được kiểm thử.

Phần lớn các công cụ trong phạm trù phân tích động đều tạo ra báo cáo chỉ ra số lần các khối câu lệnh đã thực hiện (phân tích gộp đường) và thời gian thực hiện trung bình cho các câu lệnh (phân tích hiệu năng).

Một kiểu công cụ kiểm thử động khác đôi khi được gọi là công cụ bắt giữ/chạy lại. Mỗi bắt giữ/chạy lại ghi lại tất cả các luồng thông tin tại điểm đặc biệt trong vòng thực hiện của chương trình. Thông thường, điểm bắt giữ xuất hiện ngay sau khi cung cấp đầu vào tương tác, tức là, điểm bắt giữ “ở ngay sau màn hình”. Về sau, khi công cụ này được chạy lại thì chương trình có thể được bắt đầu lại tại điểm bắt giữ và sẽ thực hiện như dữ liệu gốc mới được đưa vào chương trình. Công cụ bắt giữ/chạy lại cực kỳ có ích để tạo ra loạt kiểm thử hồi quy đối với các chương trình nặng về tương tác.

Một công cụ kiểm thử động có thể được dùng tiếp nối với một công cụ kiểm thử tĩnh. Bộ kiểm thử được dùng để suy ra các trường hợp kiểm thử mà sau đó được điều khiển bởi công cụ động.

9.2.10.3. Công cụ quản lý kiểm thử

Các công cụ quản lý kiểm thử được dùng rộng rãi để kiểm soát và điều phối việc kiểm thử phần mềm cho từng bước kiểm thử chính. Các công cụ trong phạm trù này quản lý và điều phối việc kiểm thử hồi quy, thực hiện những so sánh xác nhận sự khác biệt giữa đầu ra thực tại và mong đợi, tiến hành kiểm thử theo lô cho chương trình với các giao diện người – máy tương tác.

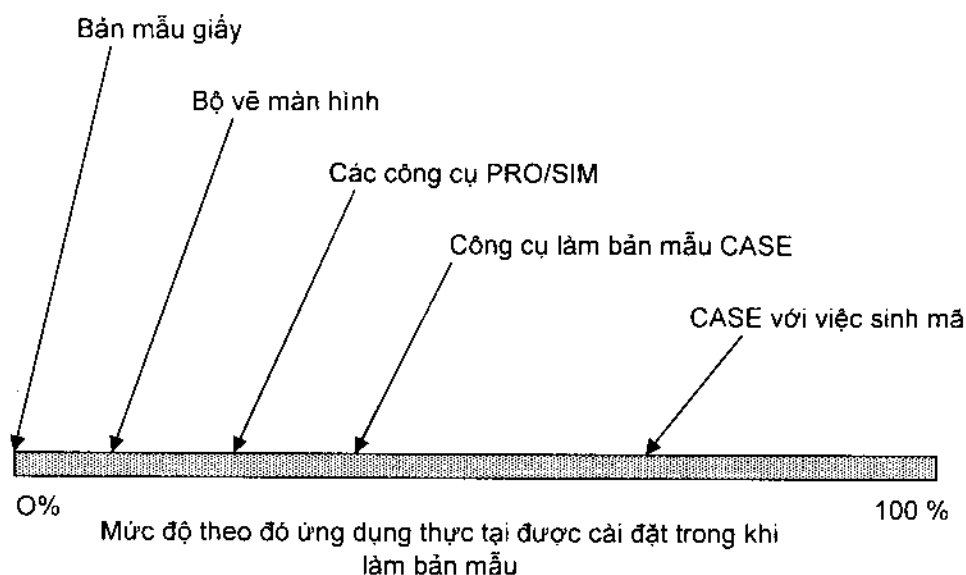
Bên cạnh các chức năng đã được lưu ý trên đây, nhiều công cụ quản lý kiểm thử cũng được dùng như khiên trình tổng quát. Một khiên trình kiểm thử đọc một hay nhiều trường hợp kiểm thử, định dạng cho dữ liệu kiểm thử để tuân theo nhu cầu phần mềm đang được kiểm thử, rồi cho phần mềm đó được kiểm thử. Các công cụ kiểm thử trong phạm trù con này được người kiểm thử điều chỉnh để đáp ứng các nhu cầu kiểm thử chuyên dụng.

Cuối cùng, người quản lý kiểm thử đôi khi làm việc gần với các công cụ theo dõi yêu cầu để đưa ra việc phân tích gộp các yêu cầu kiểm thử. Đọc tuần tự từng trường hợp kiểm thử, bộ phận phân tích gộp yêu cầu cố gắng xác định (dựa trên thông tin mô tả mục đích của trường hợp kiểm thử) xem những yêu cầu phần mềm nào được phép kiểm thử để cập tới. Một ma trận tham khảo chéo thường được dùng để chỉ ra phép kiểm thử nào đề cập đến yêu cầu nào.

9.2.11. Công cụ làm bản mẫu

Làm bản mẫu được sử dụng rộng rãi trong mô thức kỹ nghệ phần mềm và bất cứ công cụ nào hỗ trợ nó đều được gọi là công cụ làm bản mẫu. Do đó, nhiều công cụ CASE cũng có thể được đưa vào phạm trù này.

Tất cả các công cụ làm bản mẫu đều tập trung tại vị trí nào đó trong phổ cài đặt được minh họa trên hình 9.7.



Hình 9.7. Công cụ làm bản mẫu

Tại đầu thấp của phổ này, các công cụ tồn tại tạo ra một “bản mẫu trên giấy”. Một công cụ vẽ trên máy tính cá nhân hay trạm làm việc tạo ra các hình ảnh trên màn hình thực sự, các hình ảnh này được dùng để minh họa cho chức năng và hành vi của hệ thống cho khách hàng. Những hình ảnh này không thể thực hiện được. Bộ vẽ màn hình giúp kỹ sư phần mềm xác

định việc bố trí màn hình một cách nhanh chóng cho các ứng dụng tương tác. Trong một số trường hợp, bộ vẽ màn hình cũng sẽ sinh ra chương trình gốc để tạo nên màn hình. Các công cụ làm bản mẫu CASE phức tạp cho phép tạo ra thiết kế dữ liệu đi đôi với việc bố trí màn hình và báo cáo. Nhiều công cụ phân tích và thiết kế có những mở rộng đưa ra tùy chọn làm bản mẫu. Nhiều công cụ thế hệ thứ 4 cũng có tính năng làm bản mẫu.

Khi các công cụ làm bản mẫu tiến hoá, có thể một số sẽ trở thành chuyên lĩnh vực. Tức là, công cụ này sẽ được thiết kế để đề cập tới một lĩnh vực ứng dụng tương đối hẹp. Các công cụ làm bản mẫu cho viễn thông, những ứng dụng hàng không, tự động hoá nhà máy, và nhiều lĩnh vực khác sẽ dùng một cơ sở tri thức “hiều” được miền ứng dụng, thuận tiện cho việc tạo ra các hệ thống bản mẫu.

9.2.12. Công cụ bảo trì

Công cụ CASE bảo trì phần mềm đề cập tới một hoạt động hiện tiêu tốn khoảng 70% nỗ lực liên quan đến phần mềm. Phạm trù các công cụ bảo trì có thể chia nhỏ theo các chức năng sau:

- *Kỹ nghệ ngược đối với công cụ đặc tả* : Nhận chương trình gốc làm đầu vào và sinh ra các mô hình phân tích, thiết kế có cấu trúc đồ thị, các danh sách được dùng và những thông tin thiết kế khác.
- *Công cụ tái cấu trúc và phân tích mã* : Phân tích cú pháp chương trình, sinh ra đồ thị luồng điều khiển và sinh tự động một chương trình có cấu trúc.
- *Công cụ tái kỹ nghệ hệ thống trực tuyến* : Được dùng để thay đổi các hệ thống cơ sở dữ liệu trực tuyến.

Các công cụ trên bị giới hạn cho các ngôn ngữ lập trình đặc biệt và đòi hỏi một số mức độ tương tác với người kỹ sư phần mềm.

Các công cụ kỹ nghệ ngược và tái kỹ nghệ thế hệ tiếp sẽ sử dụng rất nhiều các kỹ thuật trí tuệ nhân tạo, bằng cách áp dụng cơ sở tri thức chuyên lĩnh vực ứng dụng (như tập các quy tắc phân tách sẽ áp dụng cho tất cả các chương trình trong miền ứng dụng đặc biệt như kiểm soát hay điện tử hàng không trên máy bay). Thành tố AI sẽ trợ giúp phân tách hệ thống và tái cấu trúc, nhưng sẽ đòi hỏi tương tác với kỹ sư phần mềm qua vòng tái kỹ nghệ.

9.2.12.1. Công cụ kỹ nghệ ngược

Công cụ kỹ nghệ ngược thực hiện việc phân tích sau khi phát triển một chương trình đã có. Giống như các công cụ kiểm thử, các công cụ kỹ nghệ ngược được phân thành loại tĩnh và động.

Công cụ kỹ nghệ tĩnh dùng chương trình gốc làm đầu vào, phân tích và đưa ra cấu trúc chương trình, cấu trúc điều khiển, luồng logic, cấu trúc dữ liệu và luồng dữ liệu. Các công cụ khác trong phạm trù này áp dụng một kỹ thuật gọi là cắt lát chương trình. Kỹ sư phần mềm xác định các kiểu cấu trúc chương trình (khai báo dữ liệu, chu trình, logic khác) cần quan tâm còn công cụ kỹ nghệ ngược thì loại bỏ các mã không liên quan, chỉ để lại các mã đáng quan tâm được biểu diễn. Công cụ phân tích sự phụ thuộc thực hiện phần lớn các chức năng đã được thảo luận, nhưng bên cạnh đó, các công cụ trong phạm trù con này xây dựng ra các bản đồ phụ thuộc biểu diễn theo đồ thị để chỉ ra mối nối giữa các cấu trúc dữ liệu, các thành tố chương trình, và các đặc trưng chương trình do người dùng xác định. Các công cụ kỹ nghệ ngược tĩnh được gọi là công cụ “trực quan mã”. Trong thực tế, bằng cách giúp kỹ sư phần mềm hình dung ra chương trình, những công cụ đó làm tăng chất lượng của những thay đổi được tạo ra và năng suất của người làm ra chúng.

Các công cụ kỹ nghệ ngược động điều phối phần mềm khi nó thực hiện và dùng thông tin thu được trong khi điều phối để xây dựng mô hình hành vi của chương trình. Mặc dù những chương trình như vậy còn tương đối hiếm, nhưng chúng cũng đưa ra thông tin quan trọng cho kỹ sư phần mềm, người phải duy trì phần mềm thời gian thực hay các hệ thống nhúng.

9.2.12.2. Công cụ tái kỹ nghệ

Các công cụ tái kỹ nghệ hiện có có thể chia thành 2 phạm trù con: công cụ tái cấu trúc mã và công cụ tái kỹ nghệ dữ liệu. Công cụ tái cấu trúc mã chấp nhận chương trình gốc phi cấu trúc làm đầu vào, thực hiện việc phân tích kỹ nghệ ngược và tái cấu trúc lại chương trình tuân thủ các khái niệm lập trình có cấu trúc hiện đại. Mặc dù những công cụ như vậy có ích nhưng chúng vẫn chỉ tập trung vào thiết kế thủ tục chương trình.

Các công cụ tái kỹ nghệ dữ liệu làm việc ở đầu kia của phổ thiết kế. Những công cụ này thẩm định lại các định nghĩa dữ liệu hay cơ sở dữ liệu

được mô tả trong ngôn ngữ lập trình hay ngôn ngữ mô tả cơ sở dữ liệu, sau đó chúng dịch các định nghĩa dữ liệu thành ký pháp đồ họa và được kỹ sư phần mềm phân tích. Bằng cách làm việc tương tác với các công cụ tái kỹ nghệ, người kỹ sư phần mềm có thể thay đổi cấu trúc logic của cơ sở dữ liệu, chuẩn hoá các tệp kết quả và tự động sinh ra một thiết kế vật lý cho cơ sở dữ liệu mới. Những công cụ này sử dụng hệ chuyên gia để nâng cao hiệu quả tối ưu phần mềm được tái kỹ nghệ.

9.2.13. Công cụ khung

Các công cụ khung là công cụ phần mềm quản lý dữ liệu, quản lý cấu hình và các công cụ CASE tích hợp các khả năng – là những thúc đẩy đầu tiên theo hướng giao thức (Interface Protocol Software Engineering – IPSE).

Công cụ trong phạm trù này biểu thị các thành tố chức năng hỗ trợ cho dữ liệu, giao diện và việc tích hợp công cụ. Phần lớn chúng cài đặt một cơ sở dữ liệu hướng đối tượng, với một tập công cụ nội bộ để thiết lập giao diện tron tru với công cụ của các nhà sản xuất khác. Phần lớn các công cụ khung đều cung cấp một khả năng quản lý cấu hình nào đó, làm cho người dùng công cụ này kiểm soát được những thay đổi với mọi khoản mục cấu hình được tạo ra bởi tất cả các công cụ CASE được tích hợp với công cụ khung đó.

9.2.14. CASE và trí tuệ nhân tạo AI

Một số công cụ CASE có hệ chuyên gia giới hạn, nhưng đại đa số các công cụ CASE hiện tại còn ít dùng kỹ thuật trí tuệ nhân tạo. Phần lớn các công cụ dùng hữu hạn AI đều sử dụng công nghệ này để kiểm tra tính đúng đắn về mặt đồ thị của các mô hình phân tích và thiết kế, áp dụng các quy tắc thiết kế kế thừa từ một phương pháp phân tích và kỹ thuật đặc biệt cho mô hình đã được kỹ sư phần mềm tạo ra. Tuy nhiên, hứa hẹn thực sự của CASE AI lại nằm ở chỗ khác.

Những nhà nghiên cứu ước lượng các môi trường lập trình có dùng tác nhân phân tích và thiết kế – các công cụ thông minh trợ giúp cho phân tích, thiết kế và kiểm thử các hệ thống dựa trên máy tính. Thay vì chỉ ước lượng mô hình mà người dùng tạo ra, một tác nhân sẽ hỗ trợ cho kỹ sư phần mềm

giải quyết vấn đề. Những tác nhân như vậy phải thuộc lĩnh vực chuyên môn của cơ sở tri thức này và có khả năng dùng cơ sở tri thức này để hướng dẫn kỹ sư phần mềm phân tích, thiết kế và kiểm thử.

9.3. UML

9.3.1. Khái niệm UML

UML (Unified Modeling Language) là ngôn ngữ mô hình hoá chuẩn được sử dụng để đặc tả, thiết kế và xây dựng các hệ thống phần mềm. Ngôn ngữ này được xây dựng và phát triển bởi tổ chức UML Partners Consortium do tập đoàn Rational Software Corporation đứng đầu dưới sự bảo trợ của Tổ chức quản lý đối tượng OMG (Object Management Group).

Vào cuối thập kỷ 80, đầu thập kỷ 90, sự phát triển mạnh mẽ của kỹ thuật hướng đối tượng đã kéo theo sự ra đời của hàng loạt các phương pháp và ngôn ngữ phân tích, thiết kế mới. Các phương pháp này tuy đều có chung một mục đích song lại sử dụng các thuật ngữ và ký hiệu khác nhau nên gây khó khăn khi so sánh các mô hình và sử dụng lại các thiết kế. Thêm vào đó, tất cả các phương pháp này đều chưa có tính hoàn chỉnh và không có phương pháp nào thật sự nổi bật. Đến giữa thập niên 90, bắt đầu xuất hiện xu hướng hợp nhất các phương pháp và ngôn ngữ để tận dụng các ưu điểm của nhau. Kết quả là dẫn đến sự ra đời của một vài phương pháp và ngôn ngữ có nhiều ưu điểm nổi trội.

UML mang tính hợp nhất. Sự kiện đánh dấu sự ra đời của UML là vào năm 1994 khi Booch và Rumbaugh – hai chuyên gia hàng đầu về phương pháp luận trong lĩnh vực hệ thống thông tin và kỹ thuật công nghệ – cùng tham gia vào việc hợp nhất hai phương pháp có tên là Booch và OMT (Object Modeling Technique). Một năm sau đó, Jacobson – một chuyên gia hàng đầu khác – đã kết hợp phương pháp OOSE (Object Oriented Software Engineering) với hai phương pháp kể trên và cho ra đời một phương pháp chung, lấy tên là UML. Như vậy, UML ra đời là kết quả của sự hợp nhất những phương pháp cũ (Booch, OMT, OOSE) với những kinh nghiệm và kiến thức thực tế.

UML mang tính mô hình hoá (Modeling) : UML giúp chúng ta hiểu được thế giới thực bằng cách mô hình hoá thế giới thực, qua đó nắm bắt được các đặc trưng, tính toán các thông số và dự đoán các kết quả sẽ đạt được.

UML là một ngôn ngữ. Chức năng của UML như là một phương tiện để diễn đạt và trao đổi tri thức. UML có bốn đặc điểm chủ yếu phân biệt với các ngôn ngữ mô hình hoá khác:

- Tính đa mục đích.
- Có thể ứng dụng rộng rãi.
- Được hỗ trợ bởi các công cụ.
- Chuẩn công nghiệp.

UML là một ngôn ngữ mô hình hoá chuẩn nhưng không phải là một quy trình phát triển phần mềm chuẩn. UML phải được áp dụng trong phạm vi một quy trình cụ thể. Trong thực tế, các quy trình này thường khác nhau ở các tổ chức phát triển phần mềm, ở các vấn đề cần giải quyết thuộc nhiều lĩnh vực khác nhau. Do đó, các nhà phát triển UML cố gắng định nghĩa một mức siêu mô hình để thống nhất các khái niệm về ngữ nghĩa và ký hiệu, có thể hỗ trợ nhiều ngôn ngữ lập trình và quy trình phần mềm khác nhau.

UML là tổng hợp các phương pháp của Booch, OMT, OOSE tạo thành một ngôn ngữ mô hình hoá chung và có thể sử dụng rộng rãi cho những người trước đây đã quen với phương pháp trên hay các phương pháp khác. Ngoài ra, UML còn mở rộng phạm vi mô hình hoá các phương pháp hiện có và có thể mô hình hoá đầy đủ các hệ thống đồng thời hay các hệ thống phân tán. UML là ngôn ngữ có thể được sử dụng cho nhiều mục đích khác nhau, cung cấp cơ chế tổ chức và phân loại tri thức theo ngữ cảnh của vấn đề cần giải quyết. Các tri thức này được nắm bắt đầy đủ bởi mô hình bao gồm nhiều thành phần và được thể hiện qua tập các biểu đồ khác nhau có liên hệ chặt chẽ với nhau. Các biểu đồ này mô tả nội dung giao tiếp giữa các thành viên trong quy trình phát triển phần mềm và được tích hợp với nhau để tạo nên tri thức mô tả hệ thống, những vấn đề cũng như cách thức thực hiện để giải quyết chúng.

9.3.2. Khung nhìn trong UML

Việc phân tích hệ thống thường được tiến hành dưới nhiều góc độ khác nhau như yêu cầu chức năng, yêu cầu phi chức năng, cách tổ chức hệ thống. Vì vậy, để mô hình hoá hệ thống một cách chi tiết, UML đưa ra định nghĩa cấu trúc khung nhìn. Mỗi khung nhìn là một cách nhìn hệ thống dưới một góc độ cụ thể, bao gồm nhiều biểu đồ cụ thể.

UML định nghĩa 5 loại khung nhìn:

- Khung nhìn trường hợp sử dụng (use – case view).
- Khung nhìn logic (logical view).
- Khung nhìn thành phần (component view).
- Khung nhìn đồng thời (concurrency view).
- Khung nhìn triển khai (deployment view).

9.3.2.1. Khung nhìn trường hợp sử dụng

Là cách nhìn hệ thống từ góc độ người sử dụng. Nó bao gồm các vấn đề và cách giải quyết vấn đề theo từng người sử dụng hay nhóm người dùng riêng lẻ. Khung nhìn này bao gồm biểu đồ các trường hợp sử dụng.

9.3.2.2. Khung nhìn logic

Liên quan đến các đặc tính tĩnh và động của hệ thống. Lớp, đối tượng và quan hệ mô tả các khía cạnh tĩnh của hệ thống. Trong khi đó, tương tác và cộng tác động của hệ thống được mô tả qua biểu đồ trạng thái, biểu đồ diễn tiến, biểu đồ cộng tác và biểu đồ hoạt động.

9.3.2.3. Khung nhìn thành phần

Được mô tả bằng các biểu đồ thành phần mô tả sự phụ thuộc lẫn nhau giữa các module phần mềm như tài nguyên, đối tượng, thư viện và các thành phần thực thi.

9.3.2.4. Khung nhìn đồng thời

Được mô tả bằng các biểu đồ động (biểu đồ trạng thái, biểu đồ cộng tác, biểu đồ hoạt động) và các biểu đồ thực hiện (biểu đồ thành phần và biểu đồ triển khai). Khung nhìn này mô tả các thuộc tính đồng thời của hệ thống và sự triển khai qua các xử lý. Khung nhìn này liên quan đến mô tả

của hệ thống về mặt luồng thực thi đồng thời, sự tương tác của chúng và đồng bộ.

9.3.2.5. Khung nhìn triển khai

Được mô tả qua các biểu đồ triển khai, mô tả cách thức các thành phần hệ thống như máy tính và các thiết bị khác liên kết với nhau một cách vật lý.

9.3.3. Các loại biểu đồ và ký hiệu

UML định nghĩa các loại biểu đồ sau:

- Biểu đồ trường hợp sử dụng (use case diagram).
- Biểu đồ lớp (class diagram).
- Biểu đồ đối tượng (object diagram).
- Biểu đồ diễn tiến (sequence diagram).
- Biểu đồ cộng tác (collaboration diagram).
- Biểu đồ trạng thái (state – chart diagram).
- Biểu đồ hoạt động (activity diagram).
- Biểu đồ thành phần (component diagram).
- Biểu đồ triển khai (deployment diagram).

9.3.3.1. Biểu đồ trường hợp sử dụng

Biểu đồ này mô hình hoá các chức năng của hệ thống nhìn từ góc độ người sử dụng. Nó chỉ mô tả khái quát các hành động của hệ thống cần có để đáp ứng nhu cầu của người sử dụng mà không phân tích sâu hơn. Nó được tạo nên từ ba yếu tố: Các tác nhân, các trường hợp sử dụng và các quan hệ.

Tác nhân được biểu diễn bằng một hình nhân nhỏ, diễn tả một vai trò của một người, vật hay một hệ thống nào đó có tương tác với hệ thống.



Hình 9.8. Tác nhân

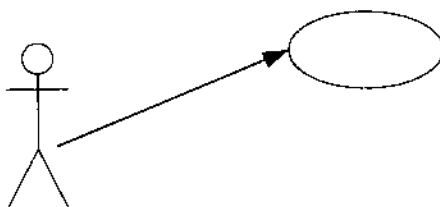
Trường hợp sử dụng được ký hiệu bằng một hình oval, biểu diễn của những kịch bản mà hệ thống có thể tương tác với một tác nhân.



Hình 9.9. Trường hợp sử dụng

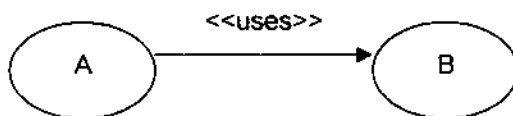
Quan hệ được biểu diễn bằng một mũi tên nối một tác nhân với một trường hợp sử dụng hoặc nối giữa trường hợp sử dụng này với một trường hợp sử dụng khác. Các quan hệ được sử dụng bao gồm quan hệ trao đổi, quan hệ sử dụng và quan hệ mở rộng.

Quan hệ trao đổi: Nối một tác nhân với một trường hợp sử dụng khởi phát bởi đầu mối đó.



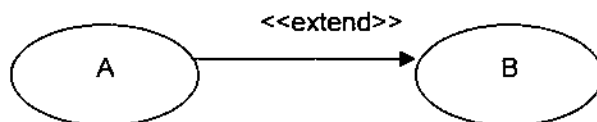
Hình 9.10. Quan hệ trao đổi

Quan hệ sử dụng: Nối hai hay nhiều trường hợp sử dụng, biểu diễn một trường hợp sử dụng này bao gồm hành vi của trường hợp sử dụng kia.



Hình 9.11. Quan hệ sử dụng

Quan hệ mở rộng: Nối hai trường hợp sử dụng, biểu diễn trường hợp sử dụng này là sự mở rộng hành vi của trường hợp sử dụng kia.



Hình 9.12. Quan hệ mở rộng

9.3.3.2. Biểu đồ lớp

Lớp là một tập hợp các đối tượng có cùng các thuộc tính, phương thức và ngữ nghĩa. Các thuộc tính mô tả khoảng giá trị mà một thể hiện của lớp có thể nắm giữ; thao tác là các dịch vụ mà một thể hiện của lớp được yêu cầu thực hiện, còn phương thức chính là sự thực thi của một thao tác nào đó. Lớp được ký hiệu như hình 9.13.

Biểu đồ lớp là biểu đồ dùng để mô hình hoá cấu trúc tĩnh của hệ thống. Nó mô tả các lớp, các giao tiếp, sự cộng tác và các mối quan hệ giữa các thành phần trong mô hình. Biểu đồ lớp gồm hai thành phần là lớp và quan hệ giữa các lớp.

Tên lớp
Thuộc tính 1
...
Thuộc tính n
Phương thức 1
...
Phương thức n

Hình 9.13. Biểu đồ lớp

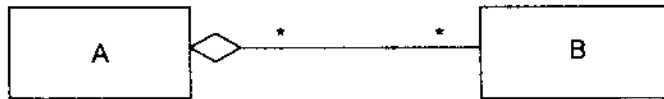
Giữa các lớp tồn tại 3 kiểu quan hệ: Quan hệ kết hợp, quan hệ kết tập và quan hệ tổng quát hoá.

- **Quan hệ kết hợp (association)** : Là quan hệ về mặt cấu trúc giữa hai hay giữa một và nhiều lớp đối tượng. Trên quan hệ kết hợp có thể có hai mũi tên theo hai hướng ngược nhau. Multiplicity (sự vô số) được mô tả trong quan hệ kết hợp, ví dụ (0..1, 1..n). Quan hệ kết hợp được ký hiệu như sau:



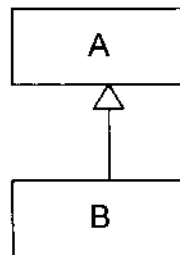
Hình 9.14. Quan hệ kết hợp

- **Quan hệ kết tập (aggregation)** : Là trường hợp riêng của quan hệ kết hợp, nó phản ánh quan hệ toàn thể – bộ phận. Quan hệ kết tập được biểu diễn như sau:



Hình 9.15. Quan hệ kết tập

- *Quan hệ tổng quát hoá (generalization)*: Là quan hệ giữa một lớp tổng quát hơn và một lớp đặc biệt hơn. Lớp đặc biệt hơn chứa đầy đủ các thuộc tính và phương thức của lớp tổng quát hơn. Ngoài ra, còn có các thuộc tính và phương thức riêng, quan hệ tổng quát còn gọi là quan hệ kế thừa.



Hình 9.16. Quan hệ tổng quát hoá

9.3.3.3. Biểu đồ đối tượng

Mô tả các đối tượng và các lớp. Các ký pháp sử dụng cho biểu đồ đối tượng được thừa kế từ biểu đồ lớp, các thành phần của nó được gạch chân.

Biểu đồ đối tượng dùng để mô tả ngữ cảnh – ví dụ trước hoặc sau tương tác. Tuy nhiên, nó cũng trợ giúp việc hiểu các cấu trúc dữ liệu phức tạp như các cấu trúc đệ quy.

9.3.3.4. Biểu đồ diễn tiến

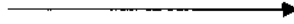
Mô tả tương tác giữa các đối tượng theo trình tự thời gian. Biểu đồ này được mô tả theo 2 trục : trục x là các đối tượng trong tương tác, trục y mô tả thời gian. Các thông điệp thể hiện sự liên lạc giữa các đối tượng gồm:

Thông điệp đơn:



Thông điệp này tích hợp với các hệ thống chỉ có một luồng điều khiển duy nhất, tại mỗi thời điểm chỉ có một đối tượng hoạt động. Một thông điệp đơn nếu được gửi từ một đối tượng hoạt động sang một đối tượng không hoạt động thì sẽ kích hoạt đối tượng này.

Thông điệp đồng bộ:



Thông điệp này sẽ khởi động thao tác ở đối tượng nhận chỉ khi đối tượng nhận tiếp nhận thông điệp. Khi một thông điệp được gửi đi thì đối tượng gửi sẽ bị phong tỏa cho đến khi đối tượng nhận tiếp nhận thông điệp.

Thông điệp không đồng bộ:



Khi một đối tượng gửi một thông điệp không đồng bộ sẽ không đợi thông điệp được trả lời mà sẽ tiếp tục quá trình thực hiện.

9.3.3.5. Biểu đồ cộng tác

Tương tự như biểu đồ diễn tiến. Mỗi biểu đồ tương ứng với một ngữ cảnh xảy ra sự tương tác giữa các đối tượng.

Có nhiều loại thông điệp liên kết như thông điệp biểu thị một lời gọi thủ tục đồng bộ, thông điệp biểu thị lời gọi thủ tục không đồng bộ... Để biểu diễn dãy các thông điệp trong sự tương tác tổng thể, các thông điệp được đánh số bắt đầu từ 1. Các thông điệp lồng nhau được ký hiệu 1.1, 1.2...

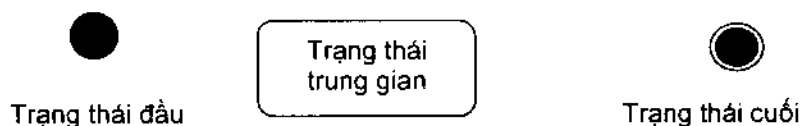
9.3.3.6. Biểu đồ trạng thái

Trong khi các biểu đồ diễn tiến và cộng tác chỉ mô tả tương tác giữa các đối tượng nhìn từ bên ngoài thì biểu đồ trạng thái mô tả các ứng xử, hành vi của các đối tượng khi tiếp nhận các thông điệp.

Có hai loại đối tượng là chủ động và bị động. Đối tượng bị động là loại đối tượng có hành vi không đổi. Khi đối tượng bị động tiếp nhận một thông điệp, nó sẽ có phản ứng không phụ thuộc vào thời gian và trạng thái của đối tượng. Đối tượng chủ động là các đối tượng có hành vi thay đổi. Khi tiếp nhận một thông điệp, các đối tượng này có phản ứng tùy thuộc vào trạng thái của chúng lúc đó. Khi một thông điệp được gửi tới đối tượng sẽ làm

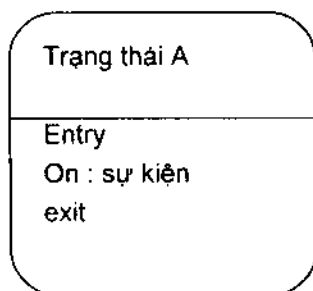
thay đổi trạng thái của nó và một số thao tác trên đối tượng sẽ được kích hoạt để đáp ứng lại thông điệp đó. Biểu đồ trạng thái sẽ mô tả hành vi ứng xử của các đối tượng chủ động, gồm các trạng thái, dịch chuyển và các sự kiện.

Trạng thái của đối tượng có đặc tính kéo dài theo thời gian và tính ổn định, nghĩa là tại một thời điểm, đối tượng phải nằm ở một trạng thái xác định, trạng thái đó tồn tại trong một khoảng thời gian và trong khoảng thời gian ấy hành vi của đối tượng là không đổi. Số các trạng thái của một đối tượng là hữu hạn. Có ba loại trạng thái là trạng thái đầu, trạng thái cuối và các trạng thái trung gian. Trong mỗi biểu đồ đều có một trạng thái đầu và có thể có hay không có một hay nhiều trạng thái kết thúc.



Hình 9.17. Ký hiệu các trạng thái

Tại mỗi trạng thái, đối tượng có thể thực hiện một trong các hoạt động tương ứng với các từ khoá *entry*, *exit*, *on*, *do*.



Hình 9.18. Biểu diễn các hoạt động bên trong trạng thái

Hoạt động được biểu diễn bằng từ khoá *entry* là hoạt động trạng thái. Ví dụ, trạng thái nhập mật khẩu, hoạt động vào trạng thái là hoạt động định dạng các ký tự nhập vào dưới dạng ẩn. Hoạt động biểu diễn bằng từ khoá *exit* là hoạt động ra khỏi trạng thái. Trong trạng thái ở ví dụ trên, trước khi ra khỏi trạng thái nhập mật khẩu, cần định dạng lại chuẩn thông thường cho việc nhập ký tự từ bàn phím (không hiển thị dưới dạng ẩn nữa). Các hoạt động biểu diễn bằng từ khoá "*on:sự kiện*" là các hoạt động của đối tượng

khi tiếp nhận một sự kiện nhưng không làm thay đổi trạng thái. Hoạt động biểu diễn bằng từ khoá “do” là các hoạt động khi đối tượng đang ở trạng thái đã cho. Các hoạt động này có thể bị ngắt khi có sự kiện làm dịch chuyển trạng thái, chúng có thể được kích hoạt ngay sau khi hoạt động vào trạng thái được hoàn thành. Nếu khi kết thúc, đối tượng ra khỏi trạng thái mà không có sự kiện bên ngoài thì bước dịch chuyển được gọi là bước dịch chuyển tự động.

Mỗi trạng thái có thể có nhiều trạng thái con và được gọi là trạng thái kép. Trong mỗi trạng thái kép có thể có trạng thái bắt đầu giả.

Các dịch chuyển biểu diễn sự thay đổi trạng thái, các dịch chuyển là tức thời vì đối tượng luôn phải ở trong một trạng thái xác định.

Các sự kiện xảy ra trong hệ thống và gây nên sự dịch chuyển trạng thái. Đối với đối tượng thì việc tiếp nhận một thông điệp được coi là một sự kiện. Nếu có nhiều thông điệp khác nhau nhưng có cùng hiệu quả về hành vi của đối tượng thì chỉ được coi là một sự kiện; ngược lại có những sự kiện được thực hiện bằng cách tiếp nhận một số thông điệp.

9.3.3.7. Biểu đồ hoạt động

Mô tả hành vi bên trong của một phương thức hay một trường hợp sử dụng. Biểu đồ hoạt động là một tập hợp các hoạt động liên kết với nhau bằng các dịch chuyển. Có thể xem biểu đồ hoạt động như một dạng của biểu đồ trạng thái nhưng các dịch chuyển hoàn toàn tự động.

Mỗi biểu đồ hoạt động có thể có nhiều mục đích khác nhau như mô tả các hoạt động của một thao tác, mô tả các hoạt động bên trong đối tượng.

9.3.3.8. Biểu đồ thành phần

Biểu đồ thành phần biểu diễn kiến trúc logic của hệ thống làm cơ sở cho việc thực hiện. Biểu đồ này bao gồm các module, các quá trình nhiệm vụ, chương trình chính, các chương trình con và các hệ con.

Mỗi module biểu diễn cho một loại phần tử vật lý có tham gia vào việc xây dựng hệ thống. Nó có thể là một tệp, một thư viện. Thông thường, mỗi lớp trong mô hình được thực hiện bằng hai module là đặc tả (chứa giao diện lớp) và thân (chứa phần cài đặt của lớp đó).

9.3.3.9. Biểu đồ triển khai

Biểu đồ này cho thấy các thiết bị khác nhau tham gia vào hệ thống và sự phân bố các chương trình được thực hiện trên các thiết bị đó.

Trong biểu đồ triển khai, các thiết bị và bộ xử lý được biểu diễn bằng một nút. Giữa các nút là các kết nối.

TÀI LIỆU THAM KHẢO

1. Roger S.Pressman. *Software Engineering A Practitioner's Approach*, 4th Ed, McGraw – Hill, 1997.
2. Roger S.Pressman (Ngô Trung Việt dịch) *Kỹ nghệ phần mềm tập 1, 2, 3*, NXB Giáo dục Hà Nội, 1997.
3. I.Smamerville. *Software Engineering*, 5th Ed Addison, Wesley, 1995.
4. Dr J. Murphy. *Introduction to OO Software Development Methods (Analysis Phases)* Module 1 – UML.
5. Bộ môn Công nghệ phần mềm trường Đại học Bách khoa Hà Nội. *Bài giảng môn học Công nghệ phần mềm*, 2005.

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU CHUNG VỀ PHẦN MỀM

1.1. Định nghĩa chung về phần mềm	3
1.2. Các đặc tính của phần mềm	5
1.3. Thế nào là một phần mềm tốt	7
1.4. Ứng dụng phần mềm	9

CHƯƠNG 2. CÔNG NGHỆ PHẦN MỀM

2.1. Định nghĩa công nghệ phần mềm	12
2.2. Vòng đời phần mềm	13
2.3. Quy trình phát triển phần mềm	15
2.3.1. Mô hình tuyến tính	16
2.3.2. Mô hình thuần thực khả năng	17
2.3.3. Mô hình chế thử	19
2.3.4. Mô hình phát triển ứng dụng nhanh	20
2.3.5. Mô hình xoắn ốc	22
2.3.6. Mô hình theo thành phần	24
2.3.7. Mô hình hình thức	25
2.3.8. Các công nghệ thế hệ thứ 4	26
2.3.9. Mô hình chữ V	28

CHƯƠNG 3. QUẢN LÝ DỰ ÁN PHẦN MỀM

3.1. Các khái niệm quản lý dự án	30
3.1.1. Con người	30
3.1.2. Vấn đề	34
3.1.3. Quy trình	35
3.2. Đo phần mềm	35
3.2.1. Độ đo theo kích cỡ	36
3.2.2. Độ đo theo điểm chức năng	37
3.2.3. Độ đo điểm chức năng mở rộng	40
3.3. Lập lịch dự án phần mềm	43
3.3.1. Mối quan hệ con người – công việc	43
3.3.2. Xác định nhiệm vụ và cơ chế song song	44
3.3.3. Phân bổ nguồn lực	46
3.3.4. Phương pháp lập lịch	47
3.4. Rủi ro	48

3.4.1	Xác định rủi ro.....	48
3.4.2	Dự phòng rủi ro.....	49
3.4.3	Định giá rủi ro	50
3.4.4	Quản lý và điều phối rủi ro	52
CHƯƠNG 4. YÊU CẦU NGƯỜI DÙNG		
4.1	Tại sao phải xác định yêu cầu.....	56
4.2	Nội dung xác định yêu cầu phần mềm.....	56
4.2.1	Phát hiện các yêu cầu phần mềm.....	57
4.2.2	Phân tích các yêu cầu phần mềm và thương lượng với khách hàng	58
4.2.3	Đặc tả yêu cầu phần mềm	59
CHƯƠNG 5. PHÂN TÍCH YÊU CẦU		
5.1	Nhiệm vụ của phân tích yêu cầu.....	61
5.2	Phân tích có cấu trúc.....	63
5.2.1	Cơ chế của phân tích có cấu trúc.....	64
5.3	Phân tích hướng đối tượng.....	67
5.3.1	Cách tiếp cận thông thường và cách tiếp cận hướng đối tượng.....	68
5.3.2	Các phương pháp phân tích hướng đối tượng.....	68
5.3.3	Các thành phần chung cho mô hình phân tích OOA.....	72
5.3.4	Quy trình OOA.....	73
5.3.5	Mô hình đối tượng – quan hệ	78
5.3.6	Mô hình đối tượng – hành vi.....	79
CHƯƠNG 6. THIẾT KẾ		
6.1	Định nghĩa thiết kế.....	81
6.2	Thiết kế có cấu trúc.....	83
6.2.1	Các xem xét về tiến trình thiết kế	83
6.2.2	Phân tích biến đổi.....	86
6.2.3	Phân tích giao tác.....	89
6.3	Thiết kế hướng đối tượng	89
6.3.1	Thiết kế cho hệ thống hướng đối tượng	89
6.3.2	Cách tiếp cận thông thường và hướng đối tượng.....	90
6.3.3	Vấn đề thiết kế.....	90
6.3.4	Các phương pháp thiết kế	91
6.3.5	Các thành phần chung của mô hình thiết kế hướng đối tượng.....	96
6.3.6	Quy trình thiết kế hệ thống.....	96
6.3.7	Quy trình thiết kế đối tượng	99
6.3.8	Mẫu thiết kế	101

6.4. Thiết kế thời gian thực	102
6.4.1. Các xem xét hệ thống	102
6.4.2. Hệ thống thời gian thực	103
6.4.3. Phân tích và mô phỏng hệ thống thời gian thực	111
6.4.4. Phương pháp thiết kế	117
6.4.5. Phương pháp thiết kế hướng luồng dữ liệu	118
CHƯƠNG 7. KIỂM THỬ	
7.1. Khái niệm kiểm thử	123
7.2. Các nguyên tắc kiểm thử	124
7.3. Phương pháp thiết kế trường hợp kiểm thử	125
7.3.1. Kiểm thử hộp trắng	126
7.3.2. Kiểm thử hộp đen	127
7.4. Các chiến lược kiểm thử	132
7.4.1. Kiểm thử đơn vị	135
7.4.2. Kiểm thử tích hợp	140
7.4.3. Kiểm thử hợp lệ	146
7.4.4. Kiểm thử hệ thống	148
CHƯƠNG 8. BẢO TRÌ	
8.1. Định nghĩa bảo trì	152
8.1.1. Bảo trì để tu sửa	152
8.1.2. Bảo trì để thích hợp	153
8.1.3. Bảo trì để cải tiến	153
8.1.4. Bảo trì để phòng ngừa	154
8.2. Các đặc trưng của bảo trì	154
8.2.1. Bảo trì có cấu trúc so với bảo trì phi cấu trúc	154
8.2.2. Chi phí bảo trì	156
8.2.3. Các vấn đề	157
8.3. Nhiệm vụ bảo trì	158
8.3.1. Tổ chức bảo trì	159
8.3.2. Báo cáo	160
8.3.3. Luồng sự kiện	160
8.3.4. Lưu trữ bản ghi	162
8.3.5. Ước lượng	163
CHƯƠNG 9. CÁC CHỦ ĐỀ NÂNG CAO TRONG CÔNG NGHỆ PHẦN MỀM	
9.1. Công nghệ phần mềm client/server	165
9.1.1. Cấu trúc hệ thống client/server	165
9.1.2. Công nghệ phần mềm cho các hệ thống C/S	170

9.1.3. Vấn đề kiểm thử	174
9.2. Công nghệ phần mềm có sự trợ giúp của máy tính (CASE)	177
9.2.1. CASE là gì?	177
9.2.2. Vai trò của CASE Tools.....	178
9.2.3. Các khối cấu thành CASE.....	178
9.2.4. Phân loại các công cụ CASE	180
9.2.5. Công cụ lập kế hoạch hệ thống	181
9.2.6. Công cụ quản lý dự án.....	181
9.2.7. Công cụ hỗ trợ.....	183
9.2.8. Công cụ phân tích và thiết kế	185
9.2.9. Công cụ lập trình	187
9.2.10. Công cụ tích hợp và kiểm thử	189
9.2.11. Công cụ làm bản mẫu	192
9.2.12. Công cụ bảo trì.....	193
9.2.13. Công cụ khung.....	195
9.2.14. CASE và trí tuệ nhân tạo AI.....	195
9.3. UML	196
9.3.1. Khái niệm UML	196
9.3.2. Khung nhìn trong UML	198
9.3.3. Các loại biểu đồ và ký hiệu	199

Chịu trách nhiệm xuất bản :

Chủ tịch HĐQT kiêm Tổng Giám đốc NGÔ TRẦN ÁI
Phó Tổng Giám đốc kiêm Tổng biên tập NGUYỄN QUÝ THAO

Tổ chức bản thảo và chịu trách nhiệm nội dung :

Chủ tịch HĐQT kiêm Giám đốc Công ty CP Sách ĐH – ĐN
TRẦN NHẬT TÂN

Biên tập nội dung và sửa bản in:

BÙI MINH HIỂN

Trình bày bìa :

ĐINH XUÂN DŨNG

Chế bản :

ĐAN NGỌC

NHẬP MÔN CÔNG NGHỆ PHẦN MỀM

Mã số : 7K754Y8 – DAI

In 1.500 cuốn (QĐ : 63), khổ 16 x 24. In tại Công ty CP In Phú Thọ.

Địa chỉ : Phường Gia Cẩm, TP. Việt Trì, Phú Thọ.

Số ĐKKH xuất bản : 183 – 2008/CXB/14 – 363/GD.

In xong và nộp lưu chiểu tháng 10 năm 2008.



CÔNG TY CỔ PHẦN SÁCH ĐẠI HỌC – DẠY NGHỀ

HEVOBCO

Địa chỉ : 25 Hàn Thuyên, Hà Nội

Website : www.hevobco.com.vn

TÌM ĐỌC SÁCH THAM KHẢO KỸ THUẬT CỦA NHÀ XUẤT BẢN GIÁO DỤC

- | | | |
|----|--|------------------------------|
| 1. | Thực hành sử dụng dreamweaver
cho thiết kế website | Thạc Bình Cường – Vũ Thị Hậu |
| 2. | Ứng dụng excel trong giải quyết
các bài toán kinh tế | Trịnh Hoài Sơn |
| 3. | Hướng dẫn học và thiết kế website
bằng Macromedia flash | Hoàng Văn Anh |
| 4. | Sử dụng Autocad lập bản vẽ kỹ thuật
Autocad 2008 | TS. Nguyễn Văn Hiến |
| 5. | Nhập môn công nghệ phần mềm | Thạc Bình Cường |

*Bạn đọc có thể mua tại các Công ty Sách - Thiết bị trường học ở các địa phương
hoặc các Cửa hàng sách của Nhà xuất bản Giáo dục :*

Tại Hà Nội : 25 Hàn Thuyên ; 187B Giảng Võ ; 232 Tây Sơn ; 23 Tràng Tiền ;

Tại Đà Nẵng : Số 15 Nguyễn Chí Thanh ; Số 62 Nguyễn Chí Thanh ;

Tại Thành phố Hồ Chí Minh : Cửa hàng 451B - 453, Hai Bà Trưng – Quận 3 ;

240 Trần Bình Trọng – Quận 5 ;

Tại Thành phố Cần Thơ : Số 5/5, đường 30/4 ;

Website : www.nxbgd.com.vn



8 934980 181927 2



Giá: 27.000 đ