ThS. TRẦN KHÁNH DUNG

GIÁO TRÌNH

NHẬP MÔN KỸ NGHỆ PHẦN MỀM

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

TRƯỜNG ĐẠI HỌC XÂY DỰNG
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN CÔNG NGHỆ PHẦN MỀM

ThS. Trần Khánh Dung

Giáo trình

NHẬP MÔN KỸ NGHỆ PHẦN MỀM

THƯ VIỆN
Trường Đại học
XÂY DỰNG
TKL 9034/12

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
HÀ NỘI - 2011

Chịu trách nhiệm xuất bản: **TS. Phạm Văn Diễn**
Biên tập: **Nguyễn Phương Liên**
Vẽ bìa: **Trịnh Thùy Dương**

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
70 - Trần Hưng Đạo - Hà Nội

In 600 bản khổ 16 x 24 tại xưởng in Văn hóa Dân tộc.
Số đăng ký kế hoạch xuất bản: 988-2010/CXB/12-124/KHKT ngày 1/10/2010.
Quyết định xuất bản số: 331/QĐXB - NXBKHKT ngày 31/12/2010.
In xong và nộp lưu chiểu quý I/2011.

MỞ ĐẦU

Sau 30 năm phát triển, kỹ nghệ phần mềm (SE - Software Engineering) đến nay đã được thừa nhận là bộ môn chính thống và vẫn còn là một lĩnh vực đang được tranh luận sôi nổi.

Trong toàn bộ ngành công nghiệp, "kỹ sư phần mềm" đã thay thế cho "người lập trình", và được xem như tên gọi của một công việc. Các phương pháp, thủ tục và công cụ kỹ nghệ phần mềm đã được chấp nhận và ứng dụng thành công trong rất nhiều lĩnh vực công nghiệp. Các nhà quản lý và chuyên gia công nghệ thông tin đều nhận ra nhu cầu về cách tiếp cận có nguyên tắc hơn tới việc phát triển phần mềm.

Bản chất thực của cách tiếp cận SE vẫn còn chưa được thống nhất, còn nhiều ý kiến trái ngược nhau. Phương pháp tiếp cận của người thực hành là theo sát các *hoạt động tổng quát* đã được thực hiện bất kể tới mô hình SE nào đã được chọn thay vì duy trì một quan điểm vòng đời chặt chẽ.

Công trình học phần mềm không phải là việc sản sinh ra phần mềm mà nó liên quan đến việc *sản sinh ra sản phẩm một cách hiệu quả.*

Với những nguồn lực không hạn chế thì đa số các vấn đề phần mềm là giải quyết được. Nhưng thử thách đối với kỹ sư phần mềm là tạo ra phần mềm chất lượng cao với hạn chế về nguồn lực và phải theo một lịch định trước.

Giáo trình ***Nhập môn kỹ nghệ phần mềm*** trang bị cho sinh viên ngành công nghệ thông tin những khái niệm cơ bản về phần mềm và cách chế tạo phần mềm, giúp sinh viên tiếp cận có nguyên tắc hơn tới việc phát triển phần mềm thông qua các phương pháp, công cụ và thủ tục của kỹ nghệ phần mềm và cuối cùng là "xây dựng" phần mềm một cách hiệu quả.

Xin chân thành cảm ơn những ý kiến đóng góp rất quý báu của các đồng nghiệp tại Bộ môn Công nghệ Phần mềm - Khoa Công nghệ Thông tin - Trường Đại học Xây Dựng trong quá trình biên soạn cuốn giáo trình này.

Tác giả

MỤC LỤC

Chương 1:

PHẦN MỀM VÀ KỸ NGHỆ PHẦN MỀM

1. PHẦN MỀM

Bắt đầu thập kỷ 80, "phần mềm" đã trở thành một chủ đề quan tâm của các tạp chí. Tiêu đề của các bài báo báo hiệu cho một cách hiểu mới về ầm quan trọng của phần mềm máy tính và những cơ hội hay thách thức mà nó đem đến.

Phần mềm giờ đã vượt trội hơn phần cứng, điều này được xem như điểm mấu chốt cho sự thành công của các hệ thống dựa trên máy tính. Những tính năng của mỗi phần mềm là cơ sở để đánh giá, phân biệt nó với các sản phẩm cạnh tranh có cùng chức năng tương tự. Cũng chính phần mềm tạo ra sự khác biệt của một công ty với các đối thủ cạnh tranh của nó.

Thách thức chủ yếu trước những năm 1990 là phải phát triển phần cứng máy tính nhằm giảm giá thành xử lý và lưu trữ dữ liệu. Trong suốt thập kỷ 1980, những tiến bộ trong vi điện tử đã làm phát sinh năng lực tính toán mạnh hơn và giá thành phần cứng giảm đi đáng kể. Ngày nay vấn đề đã khác đi. Mục tiêu chủ yếu trong những 1990 là phải cải thiện chất lượng và giảm giá thành của các giải pháp dựa trên máy tính chính là những giải pháp được cài đặt bằng phần mềm.

Khả năng xử lý và lưu trữ của phần cứng hiện đại thể hiện cho tiềm năng tính toán. Còn phần mềm là một cơ chế giúp chúng ta chế ngự và khai thác tiềm năng này.

1.1. Khái niệm phần mềm

Phần mềm là:

- Các lệnh (chương trình) khi được thực hiện trên máy vi tính thì đưa ra hoạt động và kết quả mong muốn.

- Các cấu trúc dữ liệu hoặc cơ sở dữ liệu làm cho chương trình thao tác thông tin thích hợp.
- Các tài liệu mô tả thao tác và cách dùng chương trình.

Vấn đề ở đây không phải là đưa ra các đinh nghĩa khác đầy đủ hơn mà chúng ta cần có một định nghĩa hình thức hơn.

1.2. Quá trình tiến hoá của phần mềm

1.2.1. Những năm đầu (1950-1960)

Trong những năm đầu của việc phát triển hệ thống máy tính, phần cứng được thay đổi liên tục, trong khi đó, việc lập trình được coi là một "nghệ thuật" theo bản năng và ít có phương pháp mang tính hệ thống, việc phát triển phần mềm không được quản lý.

Phần cứng vạn năng đã trở thành thông dụng. Phần mềm, mặt khác, lại được thiết kế theo đơn đặt hàng cho từng ứng dụng và được phân phối hạn chế. Môi trường phần mềm có tính cá nhân, cho nên việc thiết kế là một tiến trình không tường minh (được thực hiện trong đầu người lập trình và thường là không có tài liệu).

Trong những năm đầu, chúng ta đã học được nhiều về việc cài đặt các hệ thống dựa trên máy tính, nhưng gần như không học được gì mấy về kỹ nghệ hệ thống máy tính.

1.2.2. Thời đại thứ hai (1960 đến giữa những năm 1970)

Các hệ thống đa chương (multi-programming) và đa nhiệm (multi-tasking) đã đưa ra những khái niệm mới về tương tác người - máy, mở ra một thế giới mới cho các ứng dụng và các mức độ mới về độ tinh vi cho cả phần cứng và phần mềm.

Các hệ thống thời gian thực có thể thu thập, phân tích và biến đổi dữ liệu từ nhiều nguồn khác nhau, do đó kiểm soát được các tiến trình và sản xuất ra "output" trong phần nghìn giây thay vì nhiều phút.

Những tiến bộ trong lưu trữ trực tuyến dẫn tới thế hệ đầu tiên của các hệ quản trị cơ sở dữ liệu.

Phần mềm đã được phát triển để phân phối theo quy mô rộng trong một thị trường nhiều bên tham dự.

Khi số lượng các hệ thống dựa trên máy tính tăng lên thì *thư viện phần mềm* cũng bắt đầu mở rộng, hàng chục ngàn câu lệnh gốc chương trình

ược bổ sung hàng ngày. Một cuộc khủng hoảng phần mềm đã bắt đầu ló dạng ở chân trời": tất cả những câu lệnh gốc này, những chương rình này đều phải sửa lại khi người ta phát hiện ra lỗi, hoặc phải được ửa lại khi yêu cầu của người dùng thay đổi, hoặc phải thích nghi với hần cứng vừa mua (gọi chung là *bảo trì phần mềm*); tuy nhiên, bản hất cá nhân của nhiều phần mềm làm cho chúng thực tế không thể bảo rì được.

.2.3. Thời đại thứ ba (giữa những năm 1970 - 1990)

Hệ thống phân tán (là hệ thống nhiều máy tính, mỗi máy thực hiện một hức năng tương tranh và liên lạc với những máy khác) làm tăng dần độ hức tạp của hệ thống dựa trên máy tính.

Mạng toàn cục, mạng cục bộ, các liên lạc số giải thông cao, và nhu cầu hâm nhập dữ liệu "lập tức" đã đặt ra những yêu cầu rất lớn cho người ập trình.

Sự tiến bộ và sự phổ cập sử dụng các bộ vi xử lý, máy tính cá nhân và ác máy trạm để bàn khá mạnh.

Phần cứng giá rẻ nhanh chóng trở thành hàng hoá tiêu dùng. Chi phí ho phần mềm có khuynh hướng tăng lên so với chi phí mua phần ứng.

.2.4. Thời đại thứ tư (1990 đến nay)

Kỹ nghệ hướng sự vật đang nhanh chóng thay thế nhiều cách tiếp cận hát triển phần mềm truyền thống trong các lĩnh vực ứng dụng.

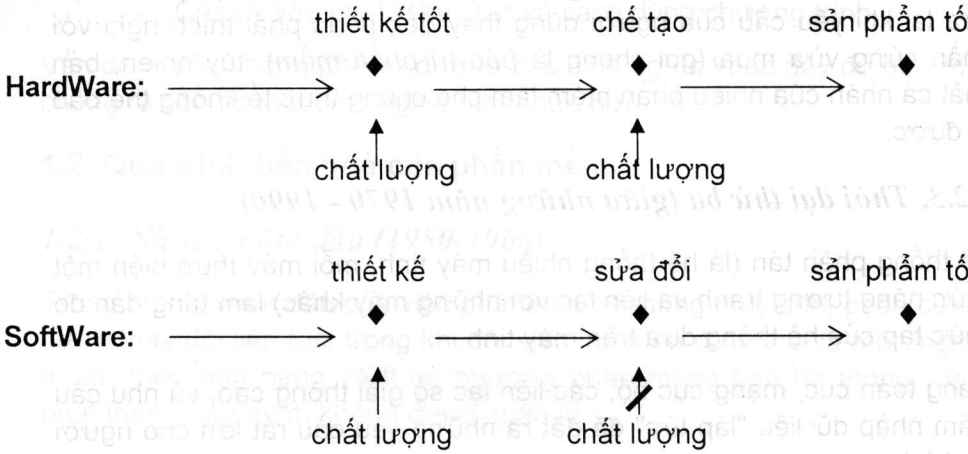
Việc áp dụng các kỹ thuật "thế hệ thứ tư" (được thảo luận ở phần sau) ang làm thay đổi cách thức phát triển phần mềm, trong đó, một phần ủa cộng đồng phần mềm xây dựng được phần mềm khác (phần mềm ạo ra phần mềm).

Hệ chuyên gia và phần mềm trí tuệ nhân tạo cuối cùng đã chuyển từ hòng thí nghiệm vào ứng dụng thực tế. Phần mềm mạng nơ ron nhân ạo đã mở ra những khả năng lý thú để nhận dạng và thực hiện những hả năng xử lý thông tin kiểu con người.

.3. Các đặc trưng của phần mềm

Phần mềm là một phần tử hệ thống logic chứ không phải là hệ thống vật ý. Do đó, phần mềm có đặc trưng khác biệt đáng kể với đặc trưng của hần cứng.

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



Hai quá trình này phụ thuộc vào con người, đều đòi hỏi việc xây dựng sản phẩm nhưng cách tiếp cận thì hoàn toàn khác nhau. Trong cả hai hoạt động, chất lượng cao được đạt tới thông qua thiết kế tốt, nhưng giai đoạn chế tạo phần cứng có thể đưa vào vấn đề chất lượng mà không tồn tại cho phần mềm.

Chi phí phần mềm tập trung vào kỹ nghệ, nghĩa là các dự án phần mềm dường như là các dự án chế tạo.

Vài thập kỷ qua, khái niệm "xưởng phần mềm" đã được đề cập nhiều, khái niệm này khuyến cáo về việc sử dụng các công cụ tự động hoá cho việc phát triển phần mềm.

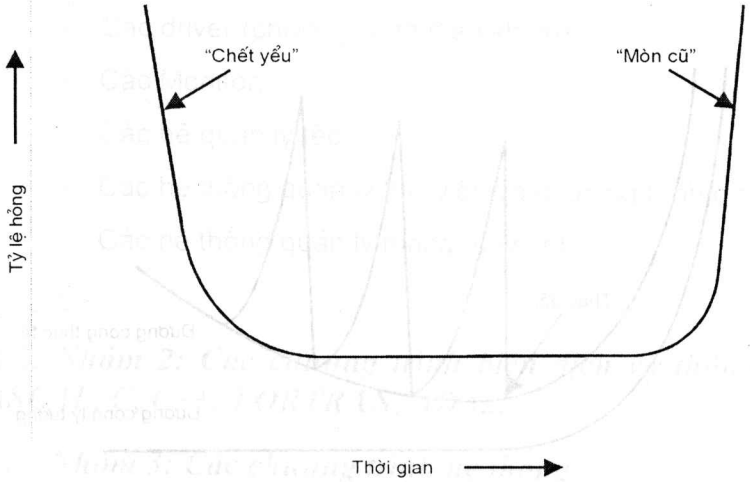
2. Phần mềm không hỏng đi

Đường cong mô tả tỉ lệ hỏng hóc cuả phần cứng theo thời gian trên được gọi là "đường cong bồn tắm". Các hỏng hóc này thường do thiết kế hoặc khiếm khuyết trong chế tạo, khi các khiếm khuyết này được sửa chữa thì tỉ lệ hỏng hóc giảm xuống tới một mức trạng thái không đổi trong một khoảng thời gian nào đó. Tuy nhiên khi vượt qua khoảng thời gian này thì tỉ lệ hỏng hóc lại tăng lên vì các yếu tố phần cứng phải chịu đựng (bụi bặm, rung động, nhiệt độ...) và bắt đầu mòn cũ đi.

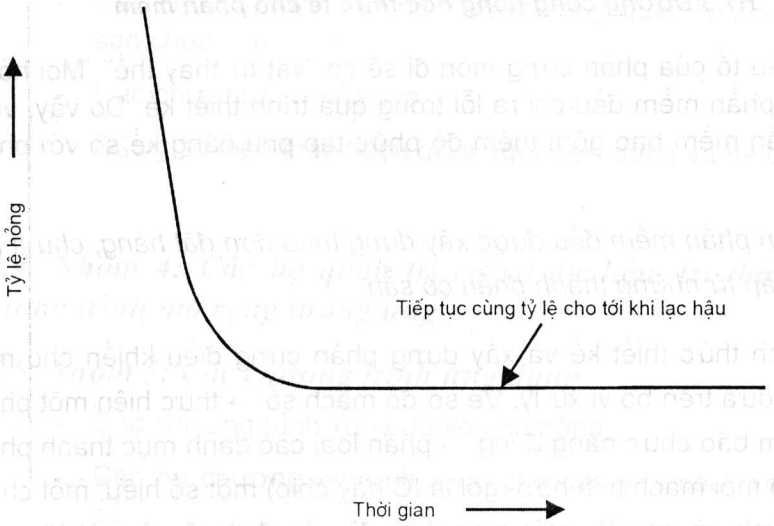
Phần mềm không cảm ứng với những ảnh hưởng môi trường vốn gây cho phần cứng bị mòn cũ. Do đó, về lý thuyết, đường cong tỉ lệ hỏng hóc cho phần mềm có dạng trong hình H1.2. Những khiếm khuyết chưa

được phát hiện sẽ gây ra tỷ lệ hỏng hóc cao từ khâu đầu của cuộc đời chương trình.

Khi các khiếm khuyết này được sửa chữa thì đường cong trở nên phẳng như được vẽ. Tuy nhiên, rõ ràng rằng phần mềm không mòn cũ đi nhưng nó lại bị lạc hậu.

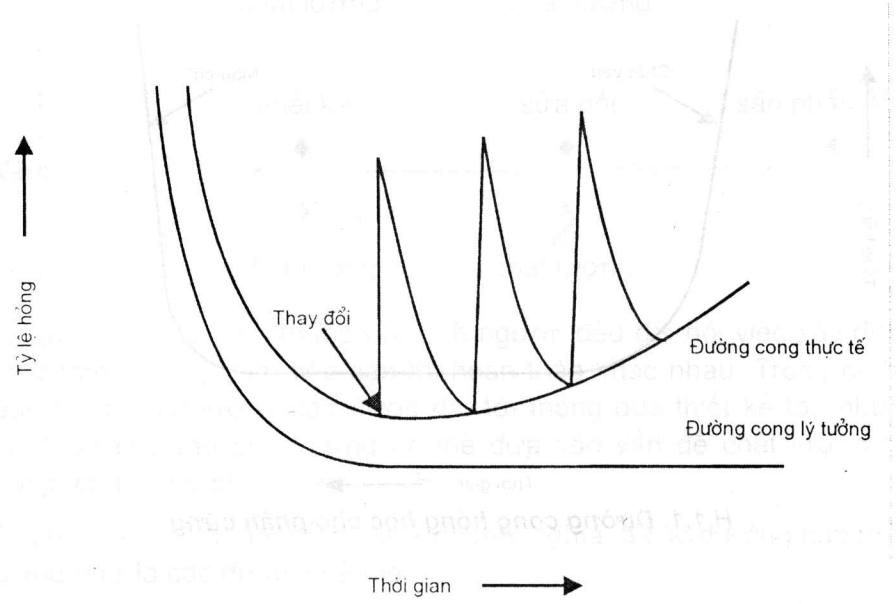


H 1.1. Đường cong hỏng hóc cho phần cứng



H 1.2. Đường cong hỏng hóc lý tưởng cho phần mềm

Thực tế, phần mềm sẽ trải qua sự thay đổi và bảo trì. Khi thay đổi được thực hiện có thể phát sinh một số khiếm khuyết mới, gây ra cho đường cong tỉ lệ hỏng hóc trở thành đầu nhọn như H1.3. Trước khi đường cong đó có thể trở về tỉ lệ hỏng hóc ổn định ban đầu thì một thay đổi khác lại được yêu cầu, lại khiến cho đường cong có thêm một đỉnh nhọn một lần nữa. Dần dần, mức tỷ lệ hỏng hóc tối thiểu bắt đầu nâng lên, phần mềm bị thoái hoá do sự thay đổi.



H1.3 Đường cong hỏng hóc thực tế cho phần mềm

Khi một yếu tố của phần cứng mòn đi sẽ có "vật tư thay thế". Mọi hỏng hóc trong phần mềm đều chỉ ra lỗi trong quá trình thiết kế. Do vậy, việc bảo trì phần mềm bao gồm thêm độ phức tạp phụ đáng kể so với phần cứng.

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

Ta xét cách thức thiết kế và xây dựng phần cứng điều khiển cho một sản phẩm dựa trên bộ vi xử lý: Vẽ sơ đồ mạch số → thực hiện một phân tích để đảm bảo chức năng đúng → phân loại các danh mục thành phần → gắn cho mỗi mạch tích hợp (gọi là IC hay chip) một số hiệu, một chức năng đã định và hợp lệ, một giao diện đã xác định rõ và một tập các hướng dẫn phân tích hợp chuẩn hoá.

Đối với phần mềm, nhìn chung các danh mục các thành phần phần mềm là không có sẵn. Có thể đặt hàng một đơn vị phần mềm hoàn chỉnh, chứ không phải là những thành phần có thể lắp ráp thành một chương trình mới. (Tuy nhiên điều này đang thay đổi nhanh chóng)

1.4. Phân loại phần mềm

1.4.1. Nhóm 1: Các hệ điều hành

- Các driver (chương trình điều khiển);
- Các Monitor;
- Các hệ quản lý tệp;
- Các hệ thống quản lý thư viện và chương trình dịch;
- Các hệ thống quản lý mạng máy tính;
- ...

1.4.2. Nhóm 2: Các chương trình biên dịch và thông dịch như PASCAL, C, C++, FORTRAN, ADA...

1.4.3. Nhóm 3: Các chương trình hệ thống

- Các chương trình soạn thảo văn bản;
- Các chương trình điều khiển các thiết bị ngoại vi;
- Các chương trình mở rộng chức năng quản lý tệp: sắp xếp, sao chép, cập nhật...;
- Các chương trình đồ hoạ;
- Các giao diện thân thiện giữa người sử dụng và hệ điều hành;
- ...

1.4.4. Nhóm 4: Các hệ quản trị cơ sở dữ liệu, tri thức và các chương trình mở rộng tương ứng

1.4.5. Nhóm 5: Các chương trình ứng dụng

- Các chương trình xử lý dữ liệu đa năng;
- Các bộ chương trình phục vụ cho các yêu cầu tính toán cơ sở;

- Các chương trình tối ưu hoá;
- Các hệ chuyên gia và các hệ tương tự;
- Các hệ mô phỏng;
- Các lớp ngôn ngữ dữ liệu và tri thức phục vụ cho việc khai thác các cơ sở dữ liệu và tri thức;
- Các hệ tự động hoá quản lý chương trình;
- Các hệ tự động hoá thiết kế;
- Các hệ thống dạy học;
- Các hệ thống tự học;
- Các hệ thống tự động phát sinh chương trình kiểm thử và sửa chương trình;
- Các hệ chương trình nhận dạng, phân tích và tổng hợp tiếng nói, hình ảnh, tín hiệu...;
- Các hệ chương trình điều khiển qui trình và các thiết bị công nghiệp;
- ...

1.4.6. Nhóm 6: Các chương trình tiện ích và trò chơi

- Các chương trình xử lý bảng tính điện tử;
- Các chương trình chuyển đổi (tiền dịch) ngôn ngữ, dịch chéo, khôi phục;
- Các chương trình chống và diệt virus máy tính;
- Các chương trình trò chơi giải trí;
- ...

1.5. Các thành phần của phần mềm

Phần mềm máy tính là thông tin tồn tại dưới hai dạng cơ bản: các thành phần máy không thực hiện được và các thành phần máy thực hiện được. Ở đây ta chỉ xét những thành phần phần mềm trực tiếp đưa tới các lệnh máy thực hiện được.

Thành phần phần mềm được thực hiện thông qua một loạt các hoạt động chuyển hoá yêu cầu của phần mềm thành mã máy thực hiện được. Một mô hình yêu cầu $\xrightarrow{\text{được dịch thành}}$ một thiết kế $\xrightarrow{\text{được dịch thành}}$ một

dạng ngôn ngữ xác định cấu trúc dữ liệu, thuộc tính thủ tục phần mềm và các yêu cầu có liên quan $\xrightarrow{\text{được dịch thành}}$ lệnh máy thực hiện được.

Tính *tái dụng* là một đặc trưng quan trọng của một phần mềm chất lượng cao, nghĩa là thành phần cần được thiết kế và cài đặt sao cho người ta có thể dùng lại chúng trong nhiều chương trình khác nhau (xây dựng những thư viện chương trình con khoa học). Ngày nay, cả thuật toán và cấu trúc dữ liệu đều được dùng lại. Chẳng hạn, các cửa sổ giao diện ngày nay sử dụng lại các thành phần có khả năng tạo cửa sổ đồ hoạ, đơn kéo xuống... Cấu trúc dữ liệu được đặt bên trong thư viện các thành phần dùng lại.

Các thành phần phần mềm được xây dựng bằng cách nào?

Dùng một ngôn ngữ với vốn từ vựng hạn chế, một văn phạm xác định rõ cùng các quy tắc thành lập chặt chẽ về cú pháp, ngữ nghĩa. Các thuộc tính này là ngôn ngữ cho việc dịch thành mã máy. Các dạng ngôn ngữ dang dùng hiện nay là:

Ngôn ngữ mức máy: là một biểu diễn ký hiệu cho tập lệnh của đơn vị xử lý trung tâm.

THƯ VIỆN TKL 9034/12

Nếu phần mềm được viết tốt, bảo trì được và có tư liệu tốt thì ngôn ngữ máy có thể làm cho việc sử dụng bộ nhớ và tối ưu tốc độ thực hiện chương trình rất hiệu quả.

Nếu phần mềm được thiết kế tồi, ít tài liệu thì việc sử dụng ngôn ngữ máy trở thành "ác mộng".

Ngôn ngữ cấp cao: cho phép người lập trình và chương trình được độc lập với máy. Khi các trình biên dịch phức tạp được dùng thì từ vựng, văn phạm, cú pháp và ngữ nghĩa của ngôn ngữ này phức tạp hơn nhiều so với ngôn ngữ máy.

Tuy đã có hàng trăm ngôn ngữ lập trình nhưng chỉ có một số trong số đó được dùng phổ biến, ví dụ: COBOL, FORTRAN, PASCAL, C, ADA, ngôn ngữ hướng đối tượng: C++, OBJECT PASCAL, EIFFEL, ngôn ngữ đặc thù: APL, LISP, OPS5, PROLOG, và các ngôn ngữ mô tả trong mạng nơ ron nhân tạo

Mã máy, hợp ngữ, các ngôn ngữ lập trình cấp cao thường được coi là "3 thế hệ đầu" của ngôn ngữ máy tính. Với những ngôn ngữ này bản thân người lập trình phải quan tâm đến cả việc đặc tả cấu trúc thông tin lẫn điều khiển chương trình. Do vậy, các ngôn ngữ trong ba thế hệ đầu này còn được gọi là ***ngôn ngữ thủ tục***.

Với ***ngôn ngữ phi thủ tục,*** thay vì phải yêu cầu người lập trình xác định chi tiết thủ tục thì các ngôn ngữ phi thủ tục đưa đến một chương trình bằng cách xác định kết quả mong muốn, thay vì xác định hành động cần để đạt được kết quả đó". Phần mềm hỗ trợ sẽ dịch đặc tả thành chương trình máy thực hiện được. Ngày nay các ngôn ngữ thế hệ thứ tư này đang được dùng trong các ứng dụng CSDL và các lĩnh vực xử lý dữ liệu nghiệp vụ khác.

1.6. Việc ứng dụng phần mềm

Phần mềm có thể được áp dụng khi đã có một tập các bước thủ tục (như một thuật toán) đã được xác định trước (trừ phần mềm chuyên gia và phần mềm mạng nơ ron). Nội dung thông tin và tính tất định là các nhân tố quan trọng trong việc xác định bản chất của ứng dụng phần mềm:

- Nội dung thông tin nói tới ý nghĩa và hình dạng của thông tin vào ra.
- Tính tất định thông tin nói tới việc tiên đoán trước trật tự và thời gian của thông tin.

Có bảy loại phần mềm ứng dụng:

1. *Phần mềm hệ thống:* là một tập hợp các chương trình được viết để phục vụ các chương trình khác . Ví dụ như: trình biên dịch, trình soạn thảo, tiện ích quản lý tệp... Phần mềm hệ thống xử lý cấu trúc thông tin phức tạp nhưng xác định. Nó được đặc trưng bởi tương tác chủ yếu tới phần cứng của máy tính, phục vụ nhiều người dùng, thao tác tương tranh, dùng chung tài nguyên, các quản lý tiến trình phức tạp, cấu trúc dữ liệu phức tạp và nhiều giao diện ngoài.

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

3. *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, quản lý...đã tiến hoá thành các hệ phần mềm quản lý thông tin (*Management Information System*). Những ứng dụng trong lĩnh vực này đã cấu trúc lại dữ liệu theo cách thuận tiện cho các thao tác nghiệp vụ. Ngoài ra phần mềm này còn bao gồm cả tính toán tương tác như xử lý giao tác cho các điểm bán hàng.

4. *Phần mềm khoa học và công nghệ:* phần mềm này được đặc trưng bởi các thuật toán "máy nghiền số". Có các ứng dụng phức tạp: thiên văn, núi lửa, biến động quỹ đạo tàu con thoi, sinh học phân tử... Thiết kế có máy tính trợ giúp (CAD - *Computer Aided Design*), 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.

5. *Phần mềm nhúng:* nằm sâu trong các bộ nhớ chỉ đọc và được dùng để điều khiển các sản phẩm và hệ thống người tiêu dùng và thị trường công nghiệp. Phần mềm nhúng thực hiện các chức năng giới hạn và huyền bí như điều khiển bàn phím cho lò vi sóng, hoặc đưa ra các khả năng điều khiển vận hành có ý nghĩa như chức năng số hoá trong ô tô, kiểm soát xăng...

6. *Phần mềm máy tính cá nhân:* thị trường này đã bùng nổ trong suốt một thập kỷ qua: xử lý văn bản, trang tính, đồ hoạ, quản trị cơ sở dữ liệu (CSDL), v.v. Phần mềm máy tính cá nhân biểu thị cho một số thiết kế giao diện người - máy được cải tiến nhiều nhất.

7. *Phần mềm trí tuệ nhân tạo (*AI - *Artificial intelligence):* phần mềm AI dùng các thuật toán phi số để giải quyết các vấn đề phức tạp mà tính toán trực tiếp không thể giải quyết nổi. Hiện nay, mạnh nhất là các *hệ chuyên gia* (hay còn gọi là *hệ cơ sở tri thức*). Các ứng dụng khác như: nhận dạng, chứng minh định lý, trò chơi...; Hiện nay con có *mạng nơ ron nhân tạo* đã phát triển, nó mô phỏng cấu trúc việc xử lý trong bộ óc để tạo ra một lớp phần mềm có thể nhận dạng các mẫu phức tạp.

1.7. Các thách thức đối với phần mềm máy tính

- Sự tinh vi của phần cứng luôn đi trước khả năng xây dựng phần mềm đạt tới tiềm năng của phần cứng.

- Khả năng xây dựng các chương trình mới không thể giữ cùng nhịp với các nhu cầu có chương trình mới.
- Khả năng bảo trì cho các chương trình bị đe doạ bởi những bản thiết kế nghèo nàn và tài nguyên không thích hợp.

Để đáp ứng lại các vấn đề trên, việc thực hành *kỹ nghệ phần mềm - chủ đề mà giáo trình này tập trung vào* - đang được chấp nhận trong ngành công nghiệp phần mềm.

2. KỸ NGHỆ PHẦM MỀM

2.1 Định nghĩa

Fritz Bauer đã đưa ra một định nghĩa ban đầu về *Kỹ nghệ phần mềm* như sau:

Kỹ nghệ phần mềm là việc thiết lập và sử dụng các nguyên lý công nghệ đúng đắn để thu được phần mềm một cách vừa kinh tế, vừa tin cậy, vừa làm việc hiệu quả trên các máy thực.

Đã có nhiều định nghĩa sâu sắc hơn được đưa ra, nhưng mọi định nghĩa đều nhấn mạnh vào yêu cầu về một kỷ luật công nghệ trong việc phát triển phần mềm.

Kỹ nghệ phần mềm là sự phát triển của kỹ nghệ phần cứng và hệ thống. Nó gồm một tập ba yếu tố chủ chốt:

- Phương pháp;
- Công cụ;
- Thủ tục.

Các yếu tố này giúp người quản lý kiểm soát được tiến trình phát triển phần mềm, đồng thời cung cấp cho người hành nghề một nền tảng để xây dựng được một phần mềm chất lượng cao theo một cách thức hiệu quả. Chúng ta cùng xem xét tóm tắt từng yếu tố đó.

1. ***Các phương pháp*** kỹ nghệ phần mềm đưa ra những "cách làm" về mặt kỹ thuật để xây dựng phần mềm.

Các phương pháp bao gồm những nhiệm vụ: 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 hóa, kiểm thử và bảo trì.

Các phương pháp kỹ nghệ phần mềm thường đưa ra các ký pháp đồ họa hay hướng ngôn ngữ đặc biệt và đưa ra một tập các tiêu chuẩn về chất lượng phần mềm.

2. ***Các công cụ*** kỹ nghệ phần mềm cung cấp sự hỗ trợ tự động hay bán tự động cho từng phương pháp nêu trên.

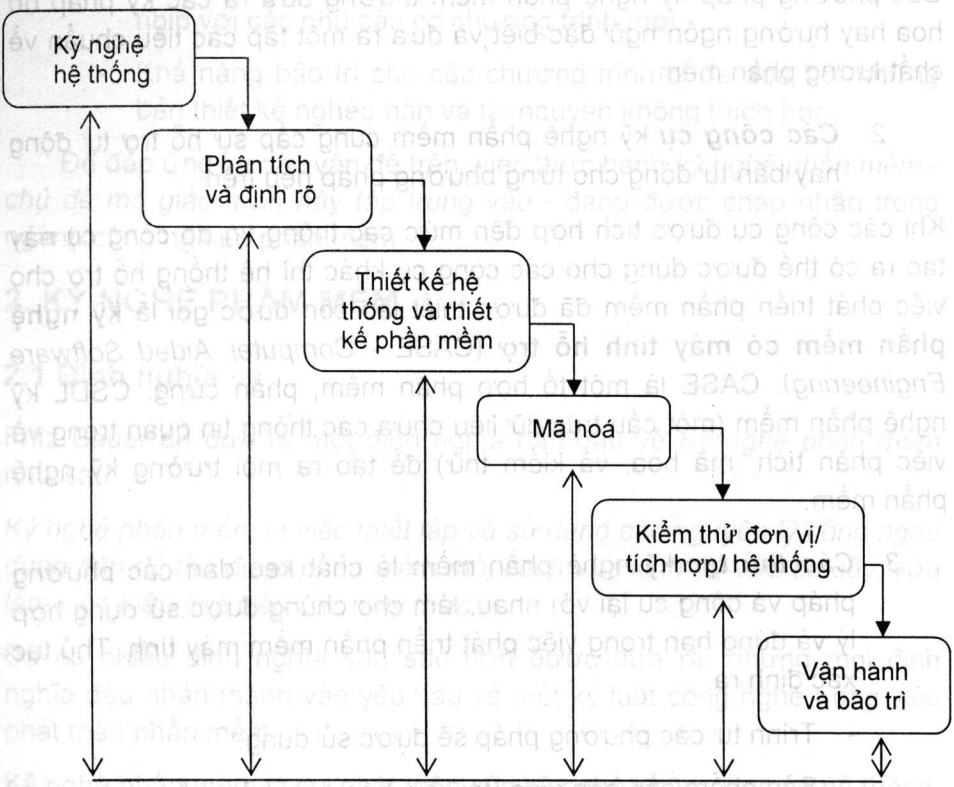
Khi các công cụ được tích hợp đến mức các thông tin do công cụ này tạo ra có thể được dùng cho các công cụ khác thì hệ thống hỗ trợ cho việc phát triển phần mềm đã được thiết lập còn được gọi là **kỹ nghệ phần mềm có máy tính hỗ trợ** (CASE - *Computer Aided Software Engineering)*. CASE là một tổ hợp phần mềm, phần cứng, CSDL kỹ nghệ phần mềm (một cấu trúc dữ liệu chứa các thông tin quan trọng về việc phân tích, mã hóa, và kiểm thử) để tạo ra môi trường kỹ nghệ phần mềm.

3. ***Các thủ tục*** kỹ 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 ra:
   - Trình tự các phương pháp sẽ được sử dụng;
   - Sản phẩm cần bàn giao (tài liệu, báo cáo, bản mẫu...);
   - Cột mốc để người làm phần mềm nắm được tiến độ.

Kỹ nghệ phần mềm bao gồm tập hợp các bước bao hàm phương pháp, công cụ, thủ tục đã được xác định ở trên. Các bước này thường được gọi là các *khuôn cảnh kỹ nghệ phần mềm.* Khuôn cảnh kỹ nghệ phần mềm được lựa chọn dựa trên bản chất của dự án, ứng dụng, phương pháp, công cụ cần dùng, cách kiểm soát và cách bàn giao. Do vậy, sau đây chúng ta sẽ xét tới bốn cách tiếp cận cơ bản trong tiến trình phát triển phần mềm.

2.2. Cách tiếp cận 1: mô hình vòng đời cổ điển

Hình 1.4 minh họa cho *mô hình vòng đời cổ điển (hay khuôn cảnh vòng đời cổ điển)* đối với kỹ nghệ phần mềm. Có tên gọi là "mô hình thác nước", khuôn cảnh vòng đời yêu cầu một cách tiếp cận *tuần tự* đối với việc phát triển phần mềm. Nó bắt đầu ở mức hệ thống và tiến dần xuống phân tích, thiết kế, mã hóa, kiểm thử và bảo trì. Như vậy khuôn cảnh vòng đời bao gồm các hoạt động trong mô hình thác nước sau:



H1.4 Vòng đời cổ điển

1. *Kỹ nghệ và phân tích hệ thống:* vì phần mềm bao giờ cũng là một phần của hệ thống lớn hơn nhiều nên công việc bắt đầu từ việc thiết lập yêu cầu cho mọi phần tử hệ thống rồi cấp phát một tập con các yêu cầu đó cho phần mềm. Kỹ nghệ và phân tích hệ thống bao gồm việc thu thập yêu cầu ở mức hệ thống với một lượng nhỏ thiết kế và phân tích mức đỉnh.

2. *Phân tích yêu cầu phần mềm:* tiến trình thu thập yêu cầu được tập trung và làm mạnh đặc biệt vào phần mềm. Các kỹ sư phần mềm cần phải hiểu về lĩnh vực thông tin đối với phần mềm, các chức năng cần có, hiệu năng và giao diện. Cần lập tư liệu về các yêu cầu cho cả hệ thống và phần mềm, và được khách hàng duyệt lại.

3. *Thiết kế:* thiết kế phần mềm là một tiến trình nhiều bước tập trung vào bốn thuộc tính phân biệt của chương trình:
   - Cấu trúc dữ liệu;

- Kiến trúc phần mềm;
- Chi tiết thủ tục;
- Đặc trưng giao diện.

Tiến trình thiết kế chuyển hóa các yêu cầu thành một biểu diễn của phần mềm có thể khẳng định về chất lượng trước khi giai đoạn mã hóa bắt đầu. Việc thiết kế phải được lập tư liệu và trở thành một phần của cấu hình phần mềm.

4. *Mã hóa:* dịch thiết kế thành dạng mã máy đọc được.

5. *Kiểm thử:* việc kiểm thử bắt đầu sau khi đã sinh ra mã, tập trung vào phần logic bên trong chương trình, đảm bảo rằng tất cả các câu lệnh đều được kiểm thử. Về phần chức năng bên ngoài cần đảm bảo việc tiến hành kiểm thử phát hiện ra các lỗi và đảm bảo những cái vào xác định sẽ tạo ra kết quả thực tế thống nhất với kết quả muốn có.

6. *Bảo trì:* phần mềm chắc chắn sẽ có những thay đổi sau khi được bàn giao cho khách hàng (ngoại lệ là những phần mềm nhúng). Nguyên nhân có thể là lỗi do phần mềm, phải thích ứng với môi trường bên ngoài (hệ điều hành mới, thiết bị ngoại vi mới...), hoặc do khách hàng yêu cầu nâng cao chức năng hay hiệu năng... Việc bảo trì phần mềm phải áp dụng lại các bước vòng đời nói trên.

Vòng đời cổ điển là khuôn cảnh ra đời sớm nhất và được sử dụng rộng rãi nhất cho kỹ nghệ phần mềm. Tuy nhiên có một số vấn đề hay gặp phải khi sử dụng khuôn cảnh vòng đời cổ điển:

- Các dự án hiếm khi tuân theo dòng chảy tuần tự mà mô hình đề ra. Việc lặp lại bao giờ cũng xuất hiện và gây ra vấn đề;
- Khách hàng khó phát biểu hết mọi yêu cầu một cách tường minh. Vòng đời cổ điển đòi hỏi điều này và thường khó thích hợp với sự bất trắc tự nhiên tồn tại vào lúc đầu của nhiều dự án;
- Khách hàng phải kiên nhẫn. Bản làm việc của chương trình chỉ có được vào lúc cuối của thời gian dự án, khi chương trình làm việc mới phát hiện ra một sai lầm không đáng có thì đó có thể sẽ là một thảm họa.

Tuy nhiên, khuôn cảnh vòng đời cổ điển có một vị trí quan trọng và xác định trong công việc về kỹ nghệ phần mềm. Nó đưa ra một kịch bản trong đó có thể bố trí các phương pháp cho phân tích, thiết kế, mã hóa, kiểm thử và bảo trì. Các giai đoạn trong khuôn cảnh này rất giống với các bước tổng quát áp dụng được trong mọi khuôn cảnh kỹ nghệ phần mềm, nó vẫn còn là một mô hình thủ tục được sử dụng rộng rãi nhất cho kỹ nghệ phần mềm. Tuy còn điểm yếu nhưng nó còn tốt hơn một cách đáng kể nếu so với cách tiếp cận ngẫu nhiên.

2.3. Cách tiếp cận 2: mô hình làm bản mẫu

Thông thường khách hàng đã xác định được mục tiêu tổng quát của phần mềm, nhưng chưa xác định được cái vào, xử lý hay yêu cầu cái ra; người phân tích chưa hiểu rõ nhu cầu của khách hàng. Ngoài ra, người phát triển có thể không chắc về tính hiệu quả của một thuật toán hay giải pháp, việc thích nghi hệ điều hành hay dạng giao diện người máy (HCI – *Human Computer Interface*) cần có. Trong những trường hợp này và nhiều trường hợp khác cách tiếp cận làm bản mẫu cho kỹ nghệ phần mềm là tốt nhất.

Làm bản mẫu là một tiến trình làm cho người phát triển có khả năng tạo ra một mô hình cho phần mềm cần phải xây dựng. Mô hình có thể ở trong ba dạng:

1. Bản mẫu trên giấy hay mô hình dựa trên máy PC mô tả giao diện người máy dưới dạng làm cho người dùng hiểu được cách các tương tác xuất hiện;

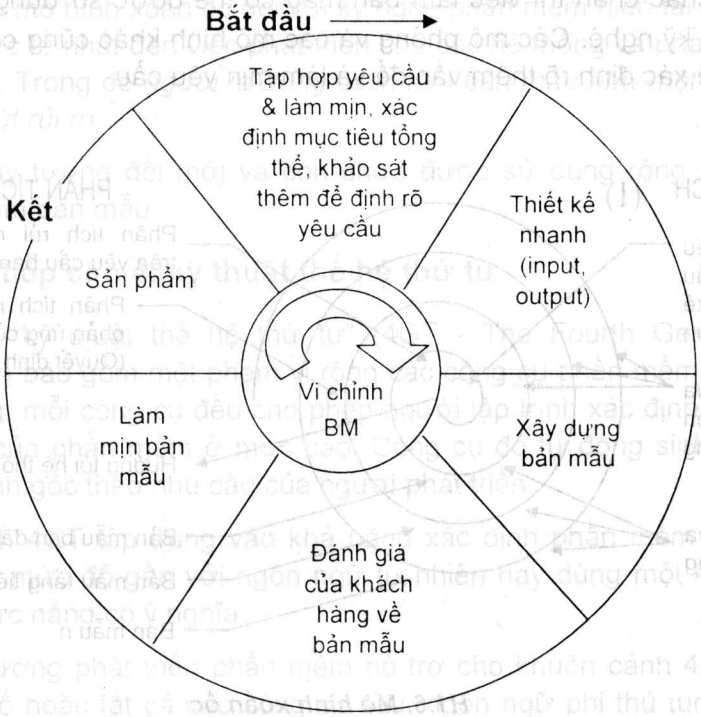
2. Bản mẫu làm việc cài đặt một tập con các chức năng của phần mềm mong muốn;

3. Một chương trình đã có thực hiện một phần hay tất cả các chức năng mong muốn nhưng cần phải cải tiến thêm các tính năng khác tùy theo khả năng phát triển.

Dãy các sự kiện của khuôn cảnh làm bản mẫu được minh họa trong hình 1.5.

Giống như mọi cách tiếp cận cho việc phát triển phần mềm, việc làm bản mẫu bắt đầu với việc thu thập yêu cầu. Người phát triển và khách hàng gặp nhau, xác định các mục tiêu tổng thể cho phần mềm, xác định các yêu cầu nào đã biết, miền nào bắt buộc phải xác định thêm. Rồi đến việc "thiết kế nhanh" để xây dựng một bản mẫu. Bản mẫu được người dùng đánh giá và được dùng để làm mịn các yêu cầu đối với phần mềm

cần phát triển. Tiến trình lặp đi lặp lại xảy ra để cho bản mẫu được "vi chỉnh" thỏa mãn nhu cầu của khách hàng, đồng thời giúp người phát triển hiểu kỹ hơn cần phải thực hiện nhu cầu nào. Giống như vòng đời cổ điển, việc làm bản mẫu tựa như một khuôn cảnh cho kỹ nghệ phần mềm có thể trở thành có vấn đề. Brook đã chỉ ra: "Trong hầu hết các dự án, hệ thống đầu tiên hiếm khi sử dụng được. Nó có thể là quá chậm, quá lớn, quá cồng kềnh trong sử dụng hay tất cả nhược điểm này. Không có cách nào là bắt đầu lại, đau đớn nhưng khôn ngoan hơn là xây dựng một phiên bản được thiết kế lại trong đó những vấn đề này đã được giải quyết..."



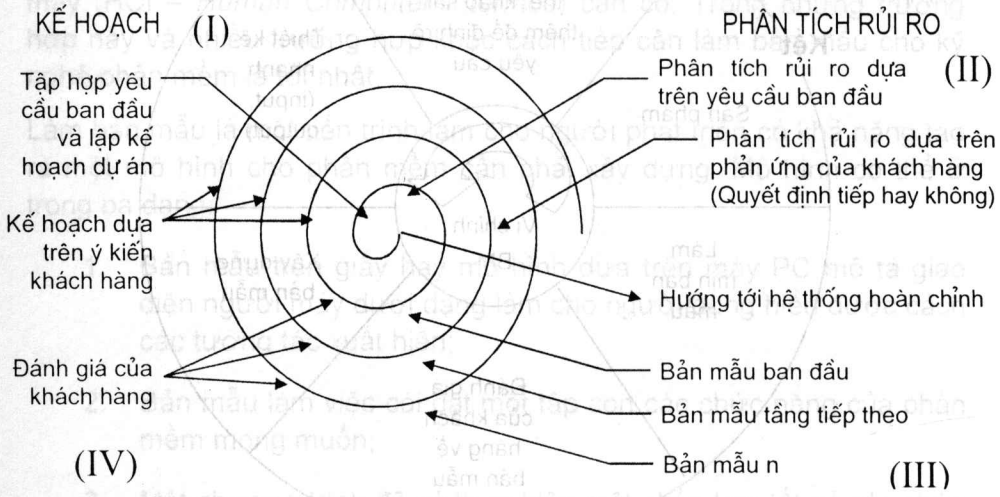
H1.5 Làm bản mẫu

2.4. Cách tiếp cận 3: mô hình xoắn ốc

Mô hình xoắn ốc bao gồm các tính năng tốt nhất của cả vòng đời cổ điển lẫn làm bản mẫu, trong khi còn bổ sung yếu tố mới là phân tích rủi ro – cái còn thiếu trong các mô hình này. Mô hình này được biểu diễn theo đường xoắn ốc, xác định ra bốn hoạt động chính:

1. *Lập kế hoạch:* xác định mục tiêu, giải pháp và ràng buộc;
2. *Phân tích rủi ro:* phân tích các phương án và xác định, giải quyết rủi ro;
3. *Kỹ nghệ:* phát triển sản phẩm "mức tiếp";
4. *Đánh giá của khách hàng:* khẳng định kết quả của kỹ nghệ.

Mỗi lần lặp xung quanh xoắn ốc (từ tâm đi ra ngoài) 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 pháp, ràng buộc, các rủi ro được định rõ và phân tích. Nếu phân tích rủi ro chỉ ra rằng các yêu cầu không chắc chắn thì việc làm bản mẫu có thể được sử dụng trong góc phần tư kỹ nghệ. Các mô phỏng và các mô hình khác cũng có thể được dùng để xác định rõ thêm vấn đề và làm mịn yêu cầu.



H1.6. Mô hình xoắn ốc

Cách tiếp cận thực tế nhất cho việc phát triển phần mềm của các hệ thống có quy mô lớn

Mỗi lần lặp xung quanh xoắn ốc (từ tâm đi ra ngoài) 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 pháp, ràng buộc, các rủi ro được định rõ và phân tích. Nếu phân tích rủi ro chỉ ra rằng, không chắc chắn trong các yêu cầu thì việc làm bản mẫu có thể được sử dụng trong góc phần tư kỹ nghệ. Các mô phỏng và các mô hình khác cũng có thể được dùng để xác định rõ thêm vấn đề và làm mịn yêu cầu.

Khách hàng đánh giá công việc kỹ nghệ và đưa ra gợi ý về những thay đổi (góc phần tư đánh giá của khách hàng), giai đoạn tiếp của việc lập kế hoạch và phân tích rủi ro sẽ được tiến hành. Tại mỗi vòng xung quanh xoắn ốc, cao điểm của việc phân tích rủi ro là "tiến hành hay không tiến hành", nếu rủi ro quá lớn có thể đình chỉ dự án.

Mọi mạch đi xung quanh xoắn ốc đều đòi hỏi **kỹ nghệ** (góc phần tư phía dưới bên phải) có thể được thực hiện bằng cách tiếp cận vòng đời và làm bản mẫu. Tất nhiên số các hoạt động phát triển xuất hiện trong góc phần tư phía dưới bên phải tăng lên khi các hoạt động chuyển xa hơn ra khỏi trung tâm xoắn ốc.

Khuôn cảnh mô hình xoắn ốc đối với kỹ nghệ phần mềm hiện tại là cách tiếp cận thực tế nhất đến việc phát triển cho các hệ thống và phần mềm quy mô lớn. Trong đó *người ta dùng cách làm bản mẫu như một cơ chế làm giảm bớt rủi ro.*

Mô hình này tương đối mới và còn chưa được sử dụng rộng rãi như vòng đời, làm bản mẫu.

2.5. Cách tiếp cận 4: kỹ thuật thế hệ thứ tư

Thuật ngữ "kỹ thuật thế hệ thứ tư" (4GT - The Fourth Generation Technology) bao gồm một phạm vi rộng các công cụ phần mềm có một điểm chung: mỗi công cụ đều cho phép người lập trình xác định một số đặc trưng của phần mềm ở mức cao. Công cụ đó tự động sinh ra mã chương trình gốc theo nhu cầu của người phát triển.

Khuôn cảnh 4GT 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 đem lại chức năng có ý nghĩa.

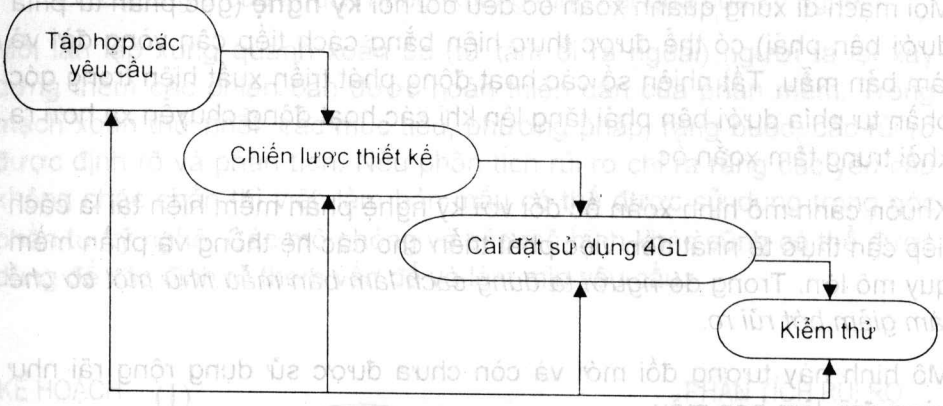
Một môi trường phát triển phần mềm hỗ trợ cho khuôn cảnh 4GT bao gồm một số hoặc tất cả các công cụ sau: ngôn ngữ phi thủ tục để hỏi CSDL, bộ sinh báo cáo, bộ thao tác dữ liệu, bộ tương tác và xác định màn hình, bộ sinh chương trình, khả năng đồ họa mức cao, khả năng làm trang tính.

Khuôn cảnh 4GT cho kỹ nghệ phần mềm được thể hiện trên H1.7.

Việc dùng khuôn cảnh 4GT còn có nhiều tranh cãi:

- Người ủng hộ cho rằng 4GT làm giảm đáng kể thời gian phát triển phần mềm, tăng hiệu suất của người lập trình.
- Người phản đối cho rằng các công cụ 4GT hiện tại không phải

tất cả đều dễ dùng hơn các ngôn ngữ lập trình, các chương trình gốc do các công cụ này tạo ra là "không hiệu quả", và rằng việc bảo trì các hệ thống phần mềm lớn hơn được phát triển bằng cách dùng 4 GT sẽ sinh ra nhiều vấn đề mới.

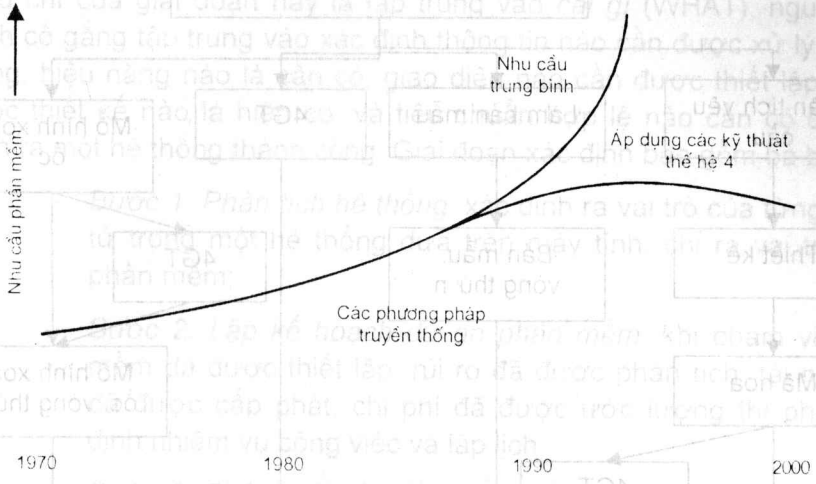


H1.7. Các kỹ thuật thế hệ thứ tư

Ta tóm tắt trạng thái hiện tại của cách tiếp cận 4 GT như sau:

- Đối với CSDL lớn, 4 GT chỉ mới giới hạn vào các ứng dụng hệ thông tin nghiệp vụ, đặc biệt là vào việc phân tích thông tin và làm báo cáo (là nhân tố chủ chốt cho các CSDL lớn)
- Đối với các ứng dụng vừa và nhỏ, thời gian thu thập dữ liệu sơ bộ cần để tạo phần mềm được giảm đáng kể. Khối lượng thiết kế cho các ứng dụng nhỏ cũng được rút bớt.
- Để phát triển phần mềm lớn, đòi hỏi tập trung nhiều vào phân tích, thiết kế và kiểm thử để đạt tới việc tiết kiệm thời gian là chủ yếu.

Tóm lại, các kỹ thuật thế hệ thứ tư đã trở thành một phần quan trọng của việc phát triển phần mềm trong lĩnh vực áp dụng công nghệ thông tin. Hình dưới đây minh họa nhu cầu phần mềm sẽ tiếp tục tăng trưởng, nhưng phần mềm được áp dụng các phương pháp và khuôn cảnh quy ước thì sẽ đóng góp ngày càng ít vào thị trường, và kỹ thuật thế hệ thứ tư sẽ lấp lỗ hổng này.



H1.8. Bản chất thay đổi của sự phát triển phần mềm

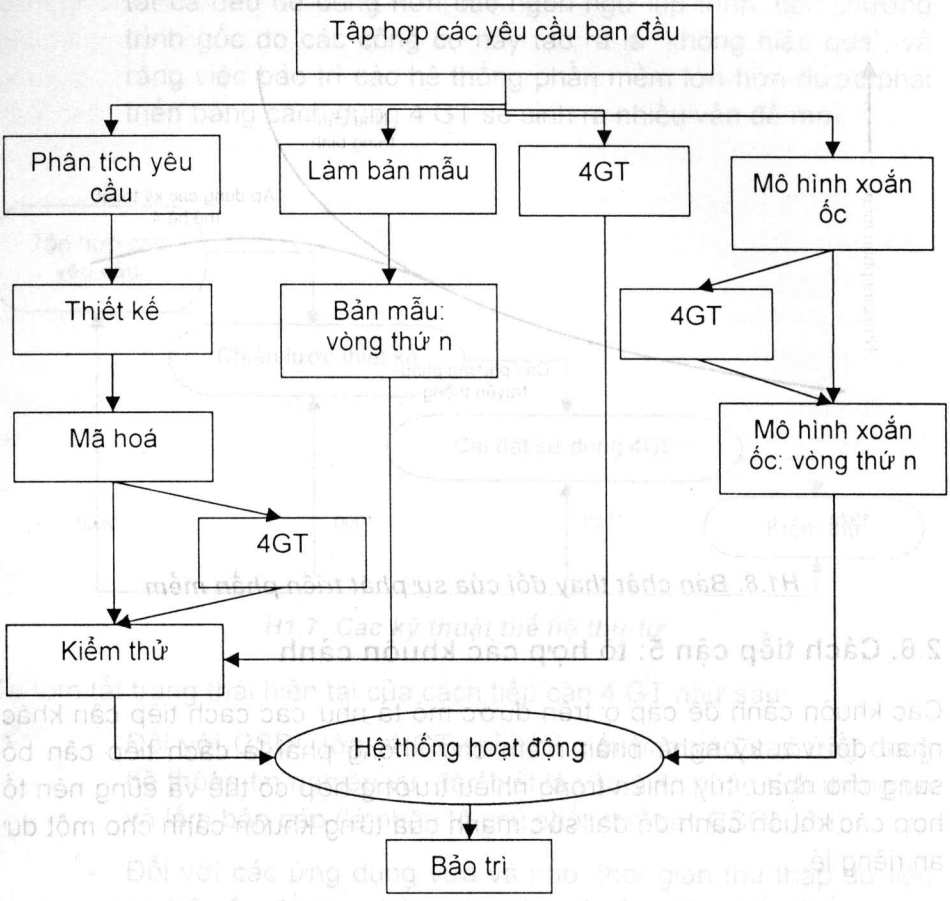
2.6. Cách tiếp cận 5: tổ hợp các khuôn cảnh

Các khuôn cảnh đề cập ở trên được mô tả như các cách tiếp cận khác nhau đối với kỹ nghệ phần mềm chứ không phải là cách tiếp cận bổ sung cho nhau, tuy nhiên trong nhiều trường hợp có thể và cũng nên tổ hợp các khuôn cảnh để đạt sức mạnh của từng khuôn cảnh cho một dự án riêng lẻ.

Hình 1.9 minh họa cách tổ hợp các khuôn cảnh kỹ nghệ phần mềm, trong mọi trường hợp, công việc bắt đầu với mọi khuôn cảnh là việc xác định mục tiêu, phương án và các ràng buộc (hay thu thập yêu cầu sơ bộ). Từ điểm này, bất kỳ một con đường nào trên hình 1.9 đều có thể được chọn.

Chẳng hạn, nếu có thể xác định được hoàn toàn hệ thống ngay từ đầu, có thể đi theo bước vòng đời cổ điển, nếu các yêu cầu còn chưa được chắc chắn thì có thể sử dụng bản mẫu như một bản hướng dẫn, người phát triển có thể trở lại các bước của vòng đời cổ điển (thiết kế, mã hóa, kiểm thử). Bản mẫu có thể tiến hóa thành hệ thống sản xuất, với việc quay trở về khuôn cảnh vòng đời để kiểm thử.

Các kỹ thuật thứ tư có thể dùng để cài đặt bản mẫu hay cài đặt hệ thống sản xuất trong bước mã hóa của vòng đời. 4GT có thể được dùng kèm với mô hình xoắn ốc cho các bước làm bản mẫu hay mã hóa.



H1.9. Tổ hợp các khuôn cảnh

Không nên cứng nhắc trong việc chọn khuôn cảnh cho kỹ nghệ phần mềm. Dựa vào bản chất của ứng dụng mà ấn định ra cách tiếp cận cần được chọn bằng cách tổ hợp các cách tiếp cận thì ích lợi một tổng thể sẽ còn lớn hơn là tổng thể của từng phần.

3. CÁC GIAI ĐOẠN TRONG TIẾN TRÌNH KỸ NGHỆ PHẦN MỀM

Tiến trình kỹ nghệ phần mềm chứa ba giai đoạn chính bất kể với khuôn cảnh kỹ nghệ phần mềm nào được lựa chọn, bao gồm: giai đoạn xác định, phát triển, và bảo trì.

3.1. Giai đoạn xác định

Tiêu chí của giai đoạn này là tập trung vào *cái gì* (WHAT), người lập trình cố gắng tập trung vào xác định thông tin nào cần được xử lý, chức năng, hiệu năng nào là cần có, giao diện nào cần được thiết lập, ràng buộc thiết kế nào là hiện có, và tiêu chuẩn hợp lệ nào cần có để xác định ra một hệ thống thành công. Giai đoạn xác định bao gồm ba bước:

- *Bước 1: Phân tích hệ thống:* xác định ra vai trò của từng phần tử trong một hệ thống dựa trên máy tính, chỉ ra vai trò của phần mềm;
- *Bước 2: Lập kế hoạch dự án phần mềm:* khi phạm vi phần mềm đã được thiết lập, rủi ro đã được phân tích, tài nguyên đã được cấp phát, chi phí đã được ước lượng thì phải xác định nhiệm vụ công việc và lập lịch;
- *Bước 3: Phân tích yêu cầu:* phạm vi xác định cho phần mềm sẽ đưa ra chiều hướng nhưng cần có thêm việc xác định chi tiết lĩnh vực thông tin và chức năng trước khi công việc bắt đầu.

3.2. Giai đoạn phát triển

Tiêu chí của giai đoạn này là tập trung vào *thế nào* (HOW), người lập trình cần xác định các cấu trúc dữ liệu, kiến trúc phần mềm cần được thiết kế, cách chi tiết thủ tục được cài đặt, cách chuyển thiết kế thành ngôn ngữ lập trình, và cách thực hiện kiểm thử. Giai đoạn phát triển bao gồm ba bước:

- *Bước 1: thiết kế phần mềm:* dịch các yêu cầu về phần mềm thành một tập các biểu diễn (đồ họa, bảng, ngôn ngữ), mô tả cấu trúc dữ liệu, kiến trúc, thủ tục, thuật toán và đặc trưng giao diện.
- *Bước 2: mã hóa:* dịch các biểu diễn thiết kế thành một hoặc nhiều ngôn ngữ nhân tạo (có thể là ngôn ngữ lập trình hay ngôn ngữ phi thủ tục trong khuôn cảnh 4GT), sẽ tạo ra kết quả là các lệnh thực hiện được trên máy tính.
- *Bước 3: kiểm thử phần mềm:* sau khi phần mềm đã được cài đặt dưới dạng máy thực hiện được, cần kiểm thử để phát hiện ra các lỗi, khiếm khuyết khi vận hành ở trong logic hoặc trong cài đặt.

3.3. Giai đoạn bảo trì

Trong giai đoạn này, người lập trình tập trung vào những *thay đổi* (CHANGE) gắn với việc sửa lỗi, thích ứng khi môi trường phần mềm nâng cao, gây ra bởi sự thay đổi yêu cầu của người dùng. Giai đoạn bảo trì gồm có ba kiểu hay gặp:

- *Sửa đổi:* làm thay đổi phần mềm để sửa chữa các khiếm khuyết;
- *Thích nghi:* thực hiện việc sửa đổi phần mềm để thích hợp với những thay đổi môi trường bên ngoài;
- *Nâng cao:* hoàn thiện, mở rộng phần mềm ra ngoài các yêu cầu chức năng gốc.

Các giai đoạn được mô tả theo cách nhìn tổng quát đang được bổ sung thêm một số "hoạt động che chắn": tiến hành xem xét để đảm bảo chất lượng được duy trì sau mỗi bước được hoàn tất. Phát triển và kiểm soát tài liệu để đảm bảo thông tin về hệ thống và phần mềm là đầy đủ, có sẵn cho cho việc sử dụng về sau.

4. TÓM TẮT

Phần mềm đã trở thành phần tử chủ chốt trong tiến hóa của các hệ thống và sản phẩm dựa trên máy tính. Hơn 40 năm qua, bản thân phần mềm đã tiến hóa từ một công cụ phân tích thông tin và giải quyết vấn đề trở thành một ngành công nghiệp.

Kỹ nghệ phần mềm là một bộ môn tích hợp cả các phương pháp, công cụ và thủ tục để phát triển phần mềm máy tính. Có thể đề ra một số khuôn cảnh khác nhau cho kỹ nghệ phần mềm, mỗi khuôn cảnh đều có những mặt mạnh và điểm yếu, nhưng nói chung tất cả đều có một dãy các giai đoạn tổng quát.

5. CỦNG CỐ

1. Môn học kỹ nghệ phần mềm (SE) phục vụ cho ai là chính, tại sao lại cần nó?
2. Phương pháp tiếp cận của người thực hành là gì?
3. Các chủ đề cần quan tâm trong SE?
4. Tại sao nói phần mềm là nhân tố để đánh giá sự khác biệt?
5. Việc chấp nhận thực hành SE là do nguyên nhân nào?

6. Các thành phần phần mềm được xây dựng bằng cách nào?
7. Bản chất ứng dụng của phần mềm được xác định bởi nhân tố nào?
8. Phần mềm ứng dụng được phân loại theo những chủ đề nào?
9. Phần mềm và kỹ nghệ phần mềm được anh (chị) hiểu như thế nào?
10. So sánh các cách tiếp cận cơ bản trong tiến trình phát triển phần mềm?
11. Ý nghĩa của việc tổ hợp các khuôn cảnh?
12. Nêu và phân tích các bước tổng quát trong tiến trình SE?

Chương 2:

PHÂN TÍCH YÊU CẦU VÀ ĐẶC TẢ PHẦN MỀM

Việc hiểu biết đầy đủ về các yêu cầu phần mềm là điểm mấu chốt cho sự thành công của nỗ lực phát triển phần mềm. Dù việc thiết kế và lập trình có tốt đến đâu thì một chương trình được phân tích và đặc tả nghèo nàn sẽ làm cho người dùng thất vọng và đem lại nuối tiếc cho người phát triển. Nhiệm vụ của phân tích yêu cầu là một tiến trình khám phá, làm mịn, mô hình hóa và đặc tả. Cả người phát triển và khách hàng đều đóng vai trò quan trọng trong việc phân tích và đặc tả yêu cầu.

1. NGƯỜI PHÂN TÍCH

Người ta đặt khá nhiều tên gọi cho nhà phân tích: người phân tích hệ thống, kỹ sư hệ thống, người thiết kế hệ thống...

Các yêu cầu đối với người phân tích:

- Khả năng hiểu thấu các khái niệm trừu tượng, có khả năng tổ chức lại thành các phân tích logic và tổng hợp các "giải pháp" dựa trên từng dải phân chia;
- Khả năng rút ra các sự kiện thích đáng từ các nguồn xung khắc và lẫn lộn;
- Khả năng hiểu được môi trường người dùng/khách hàng;
- Khả năng áp dụng các phần tử hệ thống phần cứng và/hoặc phần mềm vào môi trường người sử dụng/khách hàng;
- Khả năng trao đổi tốt thông qua dạng viết và nói;
- Khả năng "thấy rừng qua cây".

Trong giai đoạn phân tích, có hai đòi hỏi cơ bản để đạt được phần mềm tốt:

- Các yêu cầu phần mềm phải bộc lộ theo phương thức "trên - xuống". Các chức năng, giao diện và thông tin chủ yếu phải được hiểu hoàn toàn rõ trước khi xác định chi tiết các tầng lớp kế tiếp;
- Người phân tích cũng phải hiểu từng khuôn cảnh phần mềm và đánh giá được các bước kỹ nghệ phần mềm tổng quát, áp dụng được bất kể khuôn cảnh nào cần dùng. Nhiều yêu cầu phần mềm không tường minh (như thiết kế cho bảo trì) được tổ hợp vào Bản đặc tả yêu cầu chỉ nêu người phân tích hiểu được kĩ nghệ phần mềm.

2. NHIỆM VỤ PHÂN TÍCH YÊU CẦU

Việc phân tích yêu cầu phần mềm có thể chia thành NĂM bước:

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

Bắt đầu bằng việc 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 phần mềm được dùng để sinh ra ước lượng kế hoạch.

Sau đó, tiến hành trao đổi, giúp người phân tích có thể đảm bảo nhận thức đúng vấn đề. Người phân tích phải lập được mối tiếp xúc với cấp quản lý và nhân viên kỹ thuật của tổ chức phía khách hàng và tổ chức phát triển phần mềm. Người quản lý dự án có thể phục vụ như người điều phối để tạo điều kiện thuận lợi cho việc thiết lập đường trao đổi. Mục tiêu của người phân tích là nhận thức được các phần tử vấn đề cơ bản như khách hàng đã cảm nhận.

Việc đánh giá vấn đề và tổng hợp giải pháp là lĩnh vực chính tiếp theo. Cần thực hiện các bước sau:

- 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;
- Để lộ ra những ràng buộc thiết kế.

Mỗi một trong những nhiệm vụ này đều phục vụ cho việc mô tả vấn đề sao cho có thể tổng hợp được một cách tiếp cận hay giải phóng tổng thể.

Ví dụ, người ta cần tới một hệ thống quản lý kho cho một nhà cung cấp các phụ tùng ô tô. Người phân tích thấy rằng hệ thống thủ công hiện tại còn tồn tại một số vấn đề như sau:

- Không có khả năng nhận biết được trạng thái của các phụ tùng một cách nhanh chóng (thiếu thông tin);
- Phải mất đến 2 hay 3 ngày mới nhập được thẻ kho (vấn đề thời gian);
- Có nhiều mức đặt hàng lại cùng với một nhà cung cấp bởi vì không có cách nào liên kết nhà cung cấp với kho.

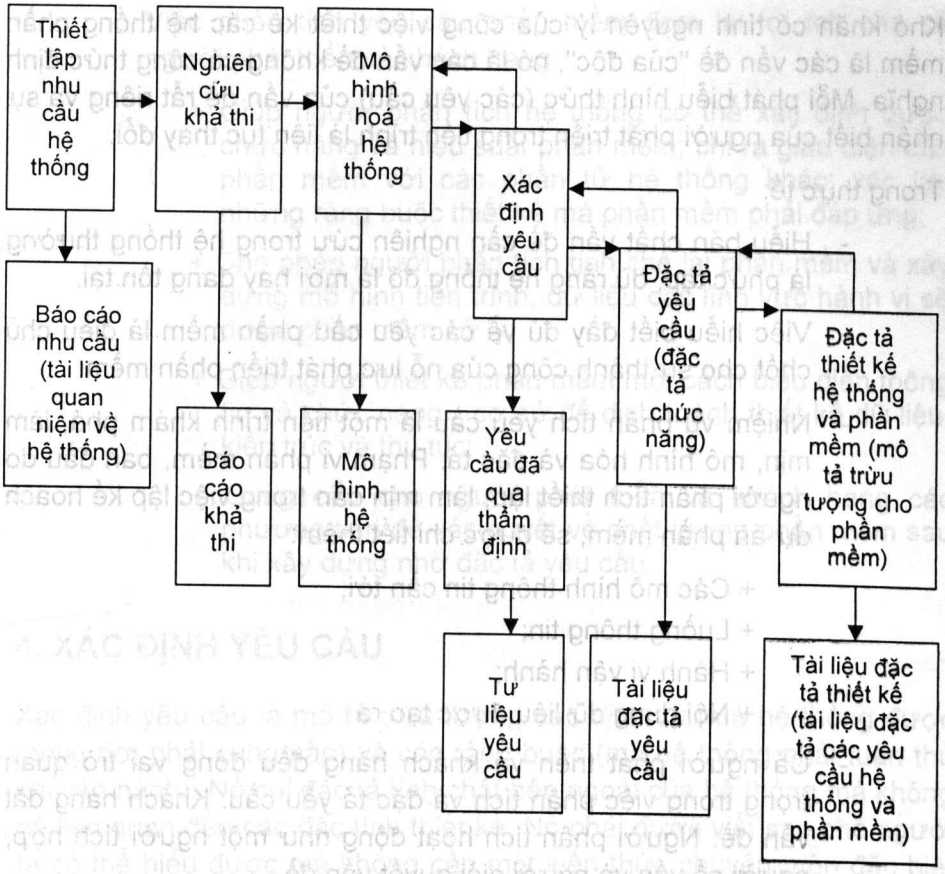
Một khi các vấn đề này đã được xác định thì người phân tích sẽ xác định hệ thống mới cần phải tạo ra thông tin nào đó và dữ liệu nào cần được cung cấp cho hệ thống. Chẳng hạn, khách hàng muốn có báo cáo hàng ngày chỉ rõ phụ tùng nào đã được đưa ra khỏi kho và bao nhiêu phụ tùng còn lại. Khách hàng chỉ ra rằng người coi kho sẽ vào sổ số hiệu của từng phụ tùng khi chúng ra khỏi khu vực kho.

Một khi đánh giá được các vấn đề hiện tại và thông tin mong muốn (cái vào và cái ra), người phân tích bắt đầu tổng hợp một hay nhiều giải pháp. Tiến trình ước lượng và tổng hợp vẫn tiếp tục cho tới khi cả người phân tích lẫn khách hàng để tin rằng phần mềm có thể được xác định thích hợp cho những bước phát triển kế tiếp.

Thông qua ước lượng và tổng hợp giải pháp, người phân tích tạo ra các mô hình hệ thống nhằm 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 được lấy làm nền tảng cho thiết kế phần mềm và là cơ sở cho việc tạo ra một đặc tả phần mềm.

3. VIỆC HÌNH THÀNH CÁC YÊU CẦU

Phân tích và định rõ yêu cầu là bước kỹ thuật đầu tiên trong tiến trình kỹ nghệ phần mềm. Hoạt động phân tích và định rõ yêu cầu hướng tới đặc tả yêu cầu phần mềm được thể hiện trong các khuôn cảnh như hình 2.1.



H2.1 Phân tích và định rõ yêu cầu

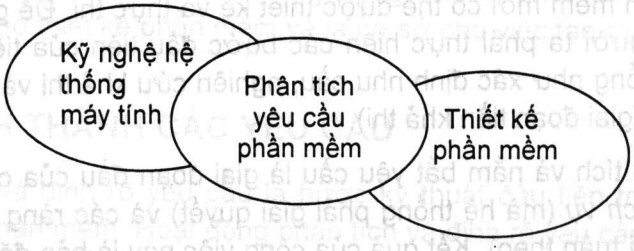
Nhu cầu của người dùng và yêu cầu của người dùng không phải như nhau: một tổ chức có thể quyết định rằng họ cần một phần mềm trợ giúp công việc kế toán. Nhưng việc trình bày một nhu cầu đơn giản như thế để cho một kỹ sư phần mềm chấp nhận được và sử dụng tốt là điều không dễ dàng. Các thông tin của vấn đề cần giải quyết phải được thu thập, phân tích và phải được xác định một cách rõ ràng. Khi đó thì giải pháp phần mềm mới có thể được thiết kế và thực thi. Để giải quyết vấn đề này người ta phải thực hiện các bước đầu tiên của tiến trình phân tích hệ thống như xác định nhu cầu, nghiên cứu khả thi và mô hình hóa hệ thống (giai đoạn tiền khả thi).

Việc phân tích và nắm bắt yêu cầu là giai đoạn đầu của quá trình thiết lập các *dịch vụ* (mà hệ thống phải giải quyết) và các ràng buộc (mà hệ thống phải tuân theo). Kết quả của công việc này là bản đặc tả yêu cầu. Đó thường là tư liệu chính thức đầu tiên được tạo ra trong quy trình xây dựng phần mềm.

Khó khăn có tính nguyên lý của công việc thiết kế các hệ thống phần mềm là các vấn đề "của độc", nó là các vấn đề không có công thức định nghĩa. Mỗi phát biểu hình thức (các yêu cầu) của vấn đề rất riêng và sự nhận biết của người phát triển trong tiến trình là liên tục thay đổi.

Trong thực tế:

- Hiểu bản chất vấn đề cần nghiên cứu trong hệ thống thường là phức tạp, dù rằng hệ thống đó là mới hay đang tồn tại.
- Việc hiểu biết đầy đủ về các yêu cầu phần mềm là điều chủ chốt cho sự thành công của nỗ lực phát triển phần mềm.
- Nhiệm vụ phân tích yêu cầu là một tiến trình khám phá, làm mịn, mô hình hóa và đặc tả. Phạm vi phần mềm, ban đầu do người phân tích thiết lập, làm mịn dần trong việc lập kế hoạch dự án phần mềm, sẽ được chi tiết thêm:
  + Các mô hình thông tin cần tới;
  + Luồng thông tin;
  + Hành vi vận hành;
  + Nội dung dữ liệu được tạo ra.
- Cả người phát triển và khách hàng đều đóng vai trò quan trọng trong việc phân tích và đặc tả yêu cầu. Khách hàng đặt vấn đề. Người phân tích hoạt động như một người tích hợp, người cố vấn và người giải quyết vấn đề.
- Việc phân tích và đặc tả yêu cầu là một nhiệm vụ không đơn giản. Nội dung trao đổi giữa hai bên là rất lớn. Việc hiểu lầm, hiểu sai, hiểu mơ hồ rất dễ phạm phải.
- Phân tích yêu cầu nhiệm vụ kỹ nghệ phần mềm để bắc nhịp cầu nối giữa kỹ nghệ hệ thống máy tính với thiết kế phần mềm:



H2.2 Nhịp cầu qua lỗ hổng

- Việc phân tích yêu cầu phần mềm đem lại lợi ích cho cả người phát triển và khách hàng.
  + Giúp người phân tích hệ thống có thể xác định được chức năng và hiệu suất phần mềm; chỉ ra giao diện của phần mềm với các phần tử hệ thống khác; xác lập những ràng buộc thiết kế mà phần mềm phải đáp ứng;
  + Cho phép người phân tích tinh chế lại phần mềm và xây dựng mô hình tiến trình, dữ liệu các lĩnh vực hành vi sẽ được phần mềm xử lý;
  + Giúp người thiết kế phần mềm một cách biểu diễn thông tin và chức năng - cơ sở để dịch thành thiết kế dữ liệu, kiến trúc và thủ tục;
  + Cung cấp cho người phát triển và khách hàng các phương tiện để xác quyết về chất lượng phần mềm sau khi xây dựng nhờ đặc tả yêu cầu.

4. XÁC ĐỊNH YÊU CẦU

Xác định yêu cầu là mô tả trừu tượng các *dịch vụ* (mà hệ thống được mong đợi phải cung cấp) và các ràng buộc (mà hệ thống phải tuân thủ khi vận hành). Nó chỉ đặc tả tính chất bên ngoài của hệ thống mà không hề liên quan đến các đặc tính thiết kế. Nó phải được viết sao cho người ta có thể hiểu được mà không cần một kiến thức chuyên môn đặc biệt nào.

Các yêu cầu được chia làm hai loại:

1. Các yêu cầu hệ thống chức năng: các dịch vụ mà hệ thống phải cung cấp.
2. Các yêu cầu phi chức năng: các ràng buộc mà hệ thống phải tuân theo.

Về nguyên tắc, các yêu cầu của hệ thống phải là vừa đầy đủ, vừa không gây ra mâu thuẫn. Thực tế đối với các hệ lớn và phức tạp thì thực khó đạt được tính đầy đủ và tính không gây mâu thuẫn cho phiên bản đầu của tư liệu yêu cầu phần mềm. Vấn đề là khi duyệt lại hoặc trong các pha sau này của vòng đời phần mềm người ta phát hiện ra các sự không thỏa mãn đó thì tư liệu yêu cầu phải được chỉnh lý lại.

Xác định yêu cầu thường được viết bằng ngôn ngữ tự nhiên cộng thêm việc dùng các bảng, các biểu đồ để cho người dùng dễ hiểu (xem như

người dùng không biết các khái niệm chuyên môn). Ngôn ngữ được dùng trong việc xác định yêu cầu thường là không chính xác và mơ hồ. Sự lầm lẫn giữa các biểu thị khái niệm và các biểu thị chi tiết đòi hỏi việc mô tả thông tin cần được biểu diễn ở nhiều mức chi tiết khác nhau.

Chú ý:

- Vì việc xác định yêu cầu khó hoàn thiện trước khi bắt đầu phát triển hệ thống nên việc áp dụng mô hình bản mẫu sẽ thích hợp hơn là mô hình thác nước.
- Cần phân biệt giữa mục tiêu của hệ thống và yêu cầu của hệ thống. Yêu cầu là một cái gì đó có thể kiểm tra được. Mục tiêu lại là một đặc trưng khái quát hơn mà hệ thống phải thể hiện. Chẳng hạn mục tiêu của hệ thống là thân thiện với người sử dụng. Nhưng sự thân thiện với người sử dụng lại không phải là một thuộc tính khách quan.

4.1. Yêu cầu đối với tư liệu các yêu cầu phần mềm

Heninger đòi hỏi 6 yêu cầu cho tư liệu các yêu cầu phần mềm:

1. Chỉ đặc tả tính chất bên ngoài của hệ thống;
2. Đặc tả các ràng buộc về sự thực hiện;
3. Phải là dễ thay đổi;
4. Phải được dùng làm công cụ tham khảo cho người bảo trì hệ thống;
5. Phải báo cáo dự tính trước về vòng đời của hệ thống;
6. Phải đặc trưng hóa các đáp ứng chấp nhận được cho các sự kiện bất ngờ.

4.2. Yêu cầu đối với cấu trúc của một tư liệu

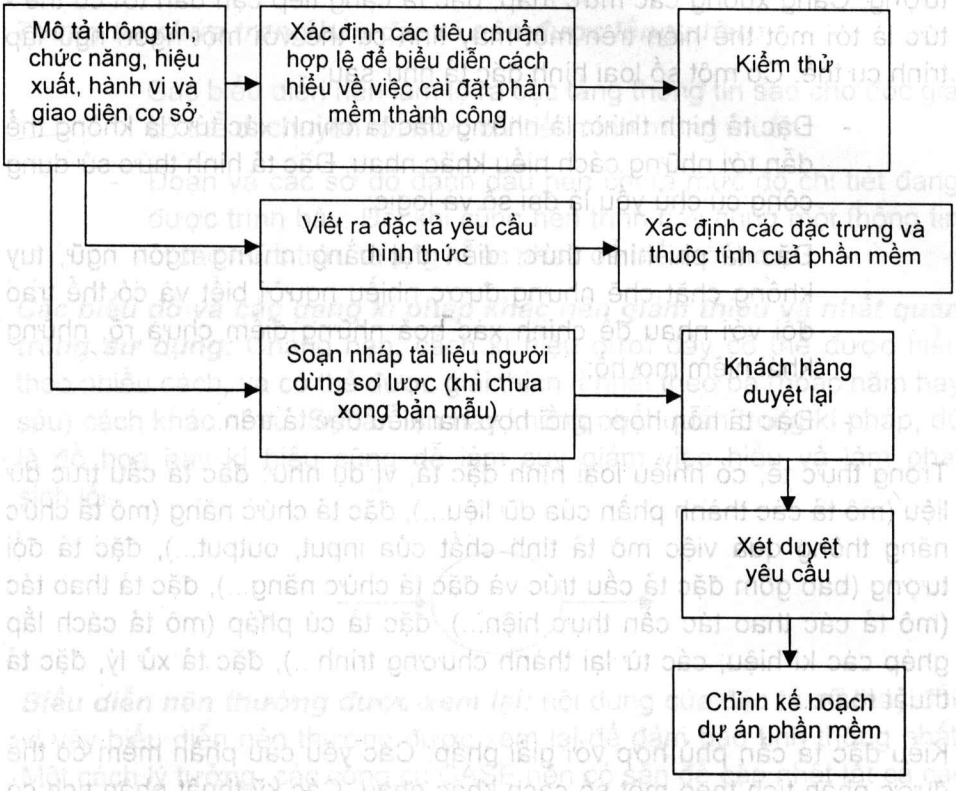
Cấu trúc của một tư liệu yêu cầu được gợi ý theo kết cấu sau:

1. Phần dẫn nhập;
2. Phần mô hình hệ thống;
3. Phần tiến triển của hệ thống;
4. Phần các yêu cầu chức năng;
5. Phần từ điển thuật ngữ.

Lưu ý:

1. Không thể có được đặc tả chi tiết trong giai đoạn này. Khách hàng có thể không chắc chắn là họ cần gì. Người phát triển cũng không chắc chắn rằng một cách tiếp cận cụ thể có thực hiện đúng chức năng và hiệu suất mong muốn không → tìm cách tiếp cận khác - làm bản mẫu;
2. Nhiệm vụ liên quan tới phân tích và đặc tả hệ thống được mô tả sao cho khách hàng có thể duyệt và chấp thuận. Lý tưởng nhất là khách hàng xây dựng bản đặc tả yêu cầu phần mềm, thực tế không có kết hợp giữa người phát triển và khách hàng.

Việc chỉnh kế hoạch trong thực tế thường được gợi ý qua sơ đồ sau:



H2.3 Vi chỉnh kế hoạch dự án phần mềm

5. ĐẶC TẢ PHẦN MỀM

5.1. Cách đặc tả và biểu diễn

5.1.1. Đặc tả

Đặc tả một vấn đề là mô tả (một cách rất riêng nhờ các kỹ thuật thể hiện) các đặc trưng của vấn đề đó. Vấn đề có thể là đối tượng, khái niệm hoặc một thủ tục nào đó...

Yêu cầu đầu tiên của đặc tả là tính chính xác.

Các đặc tả thường mang tính trừu tượng. Càng ở mức cao (những mức đầu tiên của quá trình làm mịn hoặc chính xác hóa) đặc tả càng trừu tượng. Càng xuống các mức thấp, đặc tả càng tiếp cận dần tới cụ thể - tức là tới một thể hiện trên một máy tính cụ thể với một ngôn ngữ lập trình cụ thể. Có một số loại hình đặc tả như sau:

- Đặc tả hình thức: là những đặc tả chính xác tức là không thể dẫn tới những cách hiểu khác nhau. Đặc tả hình thức sử dụng công cụ chủ yếu là đại số và logic;
- Đặc tả phi hình thức: diễn đạt bằng những ngôn ngữ, tuy không chặt chẽ nhưng được nhiều người biết và có thể trao đổi với nhau để chính xác hoá những điểm chưa rõ, những khái niệm mơ hồ;
- Đặc tả hỗn hợp: phối hợp hai kiểu đặc tả trên.

Trong thực tế, có nhiều loại hình đặc tả, ví dụ như: đặc tả cấu trúc dữ liệu (mô tả các thành phần của dữ liệu...), đặc tả chức năng (mô tả chức năng thông qua việc mô tả tính chất của input, output...), đặc tả đối tượng (bao gồm đặc tả cấu trúc và đặc tả chức năng...), đặc tả thao tác (mô tả các thao tác cần thực hiện...), đặc tả cú pháp (mô tả cách lắp ghép các kí hiệu, các từ lại thành chương trình...), đặc tả xử lý, đặc tả thuật toán...

Kiểu đặc tả cần phù hợp với giải pháp. Các yêu cầu phần mềm có thể được phân tích theo một số cách khác nhau. Các kỹ thuật phân tích có thể dẫn tới những đặc tả trên giấy hay trên máy tính (được xây dựng nhờ dùng CASE) có chứa các mô tả ngôn ngữ đồ họa và tự nhiên cho yêu cầu phần mềm. Việc làm bản mẫu đã giúp đặc tả thực hiện được, tức là bản mẫu thể hiện một biểu diễn của các yêu cầu phần mềm. Các ngôn ngữ đặc tả hình thức dẫn tới biểu diễn hình thức.

5.1.2. Biểu diễn

Các yêu cầu phần mềm có thể được biểu diễn theo nhiều cách. Các biểu diễn tốt nên tuân theo hướng dẫn sau:

Định dạng và nội dung biểu diễn theo hướng liên quan tới vấn đề:

- Theo một dàn bài chung cho nội dung của bản đặc tả các yêu cầu phần mềm.;
- Dạng biểu diễn có trong bản đặc tả có thể thay đổi theo lĩnh vực ứng dụng (chẳng hạn, đặc tả cho hệ thống tự động hóa chế tạo sẽ dùng cách kí hiệu khác, biểu đồ và ngôn ngữ khác với đặc tả cho trình biên dịch ngôn ngữ lập trình).

Thông tin chứa trong bản đặc tả nên được lồng nhau:

- Các biểu diễn nên làm lộ ra các tầng thông tin sao cho độc giả có thể di chuyển tới mức chi tiết mình mong muốn.
- Đoạn và các sơ đồ đánh dấu nên chỉ ra mức độ chi tiết đang được trình bày. Đôi khi cũng nên trình bày cùng một thông tin ở các mức trừu tượng khác nhau để hiểu tốt hơn.

Các biểu đồ và các dạng kí pháp khác nên giảm thiểu và nhất quán trong sử dụng: Chẳng hạn, cách kí hiệu dưới đây có thể được hiểu theo nhiều cách, và có thể được giải thích ít nhất theo ba (hoặc năm hay sáu) cách khác nhau. Sự lẫn lộn hay không nhất quán trong kí pháp, dù là đồ họa hay kí hiệu cũng dễ làm suy giảm việc hiểu và làm phát sinh lỗi.

Biểu diễn nên thường được xem lại: nội dung của đặc tả sẽ thay đổi, vì vậy biểu diễn nên thường được xem lại để đảm bảo tính thống nhất. Một cách lý tưởng, các công cụ CASE nên có sẵn để cập nhật tất cả các biểu diễn bị ảnh hưởng bởi từng thay đổi.

Nên sử dụng các kí hiệu, sơ đồ quen thuộc nhưng có chọn lọc: người ta đã tiến hành nhiều cuộc điều tra về nhân tố con người liên quan đến đặc tả. Dường như ít có hoài nghi rằng cách kí hiệu và thu xếp có ảnh hưởng tới việc hiểu. Tuy nhiên các kỹ sư thích các dạng ký hiệu, các sơ đồ riêng biệt. Sự quen thuộc thường thuận cho mọi người, nhưng các

nhân tố chọn lọc như cách bố trí không gian, các mẫu hình dễ nhận thức và mức hợp lý của hình thức sẽ giúp cho việc đặc tả có lợi về sau.

5.2. Các nguyên lý đặc tả

Đặc tả có thể được xem như một tiến trình biểu diễn. Mục đích cuối cùng của đặc tả các yêu cầu được biểu thị sao cho dẫn tới việc cài đặt phần mềm thành công. Balzer và Goldman đề nghị tám nguyên lý đặc tả tốt:

***Nguyên lý 1*:** phân biệt chức năng với cài đặt.

Trước hết, theo định nghĩa, đặt tả là một mô tả về điều mong muốn, chứ không phải là cách thực hiện nó (cài đặt). Đặc tả có thể chấp nhận hai dạng hoàn toàn khác nhau. Dạng thứ nhất là dạng của các hàm toán học: với một tập cái vào đã cho, tạo ra một tập cái ra đặc biệt. Dạng tổng quát của những đặc tả như thế là tìm ra một hoặc tất cả những kết quả ứng với P (cái vào), với P biểu thị một tân từ bất kỳ. Trong những đặc tả như thế, kết quả cần thu được phải hoàn toàn được diễn đạt *cái gì* không phải là *thế nào*.

***Nguyên lý 2*:** Cần tới ngôn ngữ đặc tả hệ thống hướng tiến trình.

Xét tình huống trong đó môi trường là động và sự thay đổi của nó ảnh hưởng tới hành vi của thực thể nào đó tương tác với môi trường đó. Hành vi của nó không thể biểu diễn được ở dạng hàm toán học của cái vào. Thay vì thế, cần phải sử dụng cách biểu diễn khác: cách mô tả hướng tiến trình, trong đó đặc tả cái gì đạt được bằng cách xác định một mô hình hành vi mong muốn của hệ thống dưới dạng các đáp ứng chức năng đối với các kích thước khác nhau từ môi trường.

Những đặt tả hướng tiến trình như vậy:

1. Trình bày một mô hình về hành vi hệ thống;
2. Thông thường không thuộc ngôn ngữ đặc tả hình thức;
3. Lột tả được bản chất của nhiều tình huống phức tạp cần phải đặc tả;
4. Trong những tình huống cần tự động hóa, cả tiến trình lẫn môi trường tồn tại của nó đều phải được mô tả một cách hình thức. Muốn vậy, toàn bộ hệ thống các bộ phận tương tác phải được đặc tả chứ không chỉ đặc tả một thành phần.

Nguyên lý 3: Đặc tả phải bao gồm hệ thống trong đó phần mềm là một thành phần.

Một hệ thống bao gồm các thành phần tương tác nhau. Chỉ bên trong hoàn cảnh của toàn bộ hệ thống và tương tác giữa các thành phần của nó thì hành vi của một thành phần riêng mới có thể được xác định. Nói chung, một hệ thống được mô hình hóa như một tập hợp các mối quan hệ giữa các thành phần thụ động. Những sự vật này có liên quan lẫn nhau và qua thời gian dẫn đến mối quan hệ giữa các sự vật thay đổi. Mối quan hệ động này đưa ra sự kích thích cho các sự vật tích cực, còn gọi là các tác nhân, đáp ứng. Sự đáp ứng có thể gây ra những thay đổi thêm nữa, và do đó, tạo ra thêm kích thích để cho các tác nhân có thể đáp ứng lại.

Nguyên lý 4: Đặc tả bao gồm cả môi trường mà hệ thống vận hành.

Môi trường trong đó hệ thống vận hành và các tương tác phải được xác định.

Bản thân môi trường cũng là một hệ thống bao gồm các sự vật tương tác, cả tích cực lẫn thụ động, mà trong hệ thống chúng chỉ là một tác nhân. Các tác nhân khác, theo định nghĩa là không thay đổi bởi vì chúng là một phần của môi trường, giới hạn phạm vi của việc thiết kế và cài đặt về sau. Trong thực tế, sự khác nhau duy nhất giữa hệ thống và môi trường của nó là ở chỗ nỗ lực thiết kế và cài đặt về sau sẽ vận hành chỉ trong đặc tả cho hệ thống. Đặc tả môi trường làm cho "giao diện" của hệ thống được xác định theo cùng cách như bản thân hệ thống chứ không đưa vào cách hình thức hóa khác.

Đặc tả hệ thống chính là bức tranh của tập hợp các tác nhân xoắn xuýt nhau cao độ, phản ứng lại những kích thích trong môi trường (thay đổi các sự vật). Chỉ có thông qua những hành động điều phối của tác nhân mà hệ thống mới đạt được các mục tiêu của nó. Thiết kế tuân theo đặc tả và quan tâm đến việc phân rã một đặc tả thành các mẩu gần tách biệt để chuẩn bị cho cài đặt. Tuy nhiên đặc tả phải vẽ lại chính xác bức chân dung của hệ thống và môi trường của nó như cộng đồng người dùng cảm nhận tới mức chi tiết, phục vụ cho các giai đoạn thiết kế và cài đặt. Vì mức độ chi tiết cần thiết này là khó thấy trước, nếu không nói là không thể, nên đặc tả, thiết kế và cài đặt phải được thừa nhận như một hoạt động tương tác. Do đó, điều mấu chốt là công nghệ cần bao quát thật nhiều các hoạt động này khi bản đặc tả được soạn thảo và thay đổi (trong cả hai giai đoạn phát triển khởi đầu và bảo trì về sau)

Nguyên lý 5: Đặc tả hệ thống phải là một mô hình nhận thức.

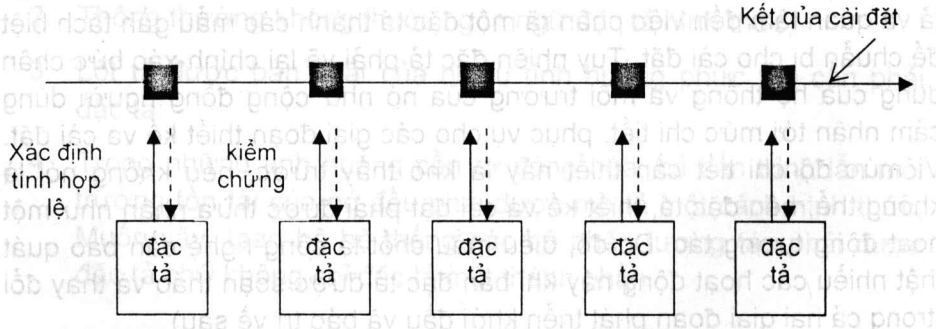
- Đặc tả hệ thống phải là một mô hình nhận thức chứ không phải là một mô hình thiết kế hay cài đặt;
- Mô tả một hệ thống sao cho đạt được sự cảm nhận của cộng đồng người sử dụng. Các sự vật mà nó thao tác phải tương ứng với các sự vật của lĩnh vực đó; các tác nhân phải được mô hình hóa cho các cá nhân, tổ chức và trang thiết bị trong lĩnh vực đó; còn các hành động họ thực hiện thì phải được mô hình hóa cho những hoạt động thực tế xuất hiện trong lĩnh vực;
- Phải có khả năng tổ hợp vào trong đặc tả những quy tắc hay luật bao trùm các sự vật thuộc lĩnh vực

Một trong những luật này bài trừ những trạng thái nào đó của hệ thống (như "hai sự vật không thể đồng thời cùng một chỗ và vào cùng một lúc") và do đó giới hạn hành vi của các tác nhân hay chỉ ra nhu cầu bổ trợ để ngăn cản những trạng thái khỏi nảy sinh.

Nguyên lý 6: Đặc tả phải thể hiện tính vận hành

Đặc tả phải đầy đủ và mang tính hình thức để có thể được dùng trong việc xác định rằng liệu một cài đặt được đề nghị có thỏa mãn đặc tả cho những trường hợp kiểm thử tùy ý không. Tức là, với kết quả của việc cài đặt trên một tập dữ liệu được chọn một cách tùy ý, phải có thể dùng đặc tả để xác định tính hợp lệ cho những kết quả đó. Điều này kéo theo rằng đặc tả, mặc dầu không phải là một đặc tả hoàn toàn về cách thức, vẫn có thể hành động như một bộ sinh các hành vi. Do đó theo nghĩa mở rộng, đặc tả này phải thể hiện tính vận hành...

Quá trình cài đặt xem hình 2.4.



H 2.4 Quá trình cài đặt

Nguyên lý 7: Đặc tả hệ thống chấp nhận dung sai về tính không đầy đủ và vì vậy có tính nâng cao.

Không đặc tả nào có thể là đầy đủ hoàn toàn. Môi trường mà hệ thống tồn tại trong đó quá phức tạp. Một đặc tả bao giờ cũng là một mô hình - một sự trừu tượng hóa - của một tình huống thực (hay được mường tượng) nào đó. Do đó nó sẽ không đầy đủ. Hơn thế nữa, nó tồn tại ở nhiều mức chi tiết. Các công cụ phân tích được sử dụng để giúp đặc tả và kiểm thử đặc tả phải có khả năng xử lý với tính không đầy đủ.

Nguyên lý 8: Đặc tả phải được cục bộ hóa và được ghép lỏng lẻo.

Các nguyên lý trước xử lý đặc tả như một thực thể tĩnh. Nguyên lý này nảy sinh từ tính động của đặc tả. Cần phải thừa nhận rằng mặc dầu mục tiêu chính của một đặc tả là dùng làm cơ sở cho thiết kế và cài đặt một hệ thống nào đó, nhưng nó không phải là một sự vật tĩnh dựng sẵn mà là một vật động đang trải qua thay đổi đáng kể.

Việc thay đổi (động) của đặc tả xuất hiện trong ba hoạt động chính:

- Phát biểu: khi một đặc tả ban đầu đang được tạo ra;
- Phát triển: khi đặc tả được soạn thảo trong quá trình thiết kế;
- Lặp: để phản ánh môi trường đã thay đổi và/hoặc các yêu cầu chức năng phụ.

Với nhiều thay đổi xuất hiện đối với đặc tả, điều mấu chốt là nội dung và cấu trúc của đặc tả được chọn để làm phù hợp với thay đổi này. Yêu cầu chính cho sự phù hợp đó là ở chỗ:

- Thông tin bên trong đặc tả phải được cục bộ hóa sao cho chỉ một phần nhỏ (một cách lý tưởng) cần phải sửa đổi khi thông tin thay đổi;
- Đặc tả cần được cấu trúc (ghép) một cách lỏng lẻo cho từng phần có thể được thêm vào hay loại bỏ một cách dễ dàng, và cấu trúc được điều chỉnh một cách tự động.

5.3. Các mức trừu tượng của đặc tả

Các đặc tả được thể hiện ở vài mức trừu tượng khác nhau cùng với mối tương quan giữa các mức ấy. Mỗi mức hướng đến các đối tượng đọc khác nhau mà họ có quyền quyết định về việc mua sắm và thực hiện.

Các mức đó là:

1. Định ra yêu cầu:

- Thể hiện bằng ngôn ngữ tự nhiên về các dịch vụ mà hệ thống sẽ phải cung cấp;
- Phải được viết sao cho dễ hiểu đối với khách hàng và người quản lý hợp đồng, người sẽ mua sắm sẽ sử dụng.

2. Đặc tả yêu cầu:

- Tài liệu nêu ra các dịch vụ một cách chi tiết hơn. Tài liệu này (thường được gọi là đặc tả chức năng);
- Đòi hỏi chính xác tới mức nó có thể làm cơ sở cho hợp đồng giữa người mua sắm hệ thống và người phát triển phần mềm;
- Được viết dễ hiểu đối với các nhân viên kỹ thuật ở cả nơi mua lẫn nơi phát triển;
- Kỹ thuật đặc tả hình thức hẳn là thích hợp cho các đặc tả kiểu như vậy, nhưng nó cũng tùy thuộc ở trình độ kiến thức cơ bản của người mua sắm hệ thống.

3. Đặc tả phần mềm/ đặc tả thiết kế (đây là một mô tả trừu tượng cho phần mềm):

- Dùng làm cơ sở cho việc thiết kế và thực thi;
- Thể hiện một quan hệ rõ ràng giữa tư liệu này và đặc tả yêu cầu;
- Đối tượng đọc ở đây chủ yếu là các kĩ sư phần mềm chứ không phải là người sử dụng hoặc người quản lý;
- Sử dụng kỹ thuật đặc tả hình thức là thích hợp cho tư liệu này.

5.4. Đặc tả yêu cầu

Việc định yêu cầu là việc đặc tả hướng khách hàng và được viết bằng ngôn ngữ của khách hàng. Khi đó có thể dùng các khái niệm không chính xác, ngôn ngữ tự nhiên và các biểu đồ. Đặc tả yêu cầu (được gọi là đặc tả chức năng) là cơ sở cho hợp đồng làm hệ thống nên không được mơ hồ, để không gây ra sự hiểu lầm của khách hàng hoặc của người ký hợp đồng.

Rất nhiều vấn đề của kỹ nghệ phần mềm là khó khăn khi đặc tả yêu cầu. Với một yêu cầu mơ hồ thì người phát triển sẽ thực hiện nó sao cho chi phí rẻ nhất. Trong khi đó khách hàng lại không muốn như vậy. Sau các cuộc tranh luận kéo dài thì các yêu cầu mới sẽ được thiết lập và có các đổi thay đối với hệ thống. Dĩ nhiên việc đó sẽ làm trì hoãn ngày phân phối hệ thống và làm tăng chi phí.

Chi phí do các sai sót trong việc phát biểu các yêu cầu có thể là rất cao, đặc biệt là nếu các sai sót này còn vẫn chưa được phát hiện khi hệ thống được thực hiện. Bochm báo cáo rằng trong một vài hệ thống lớn thì có đến 95% các mã đã phải viết lại để thỏa mãn cá yêu cầu của người dùng đã thay đổi và 12% các lỗi được phát hiện trong quá trình ba năm đầu sử dụng là các lỗi do đặc tả yêu cầu không đúng đắn.

Chi phí do thay đổi một yêu cầu là đắt hơn nhiều so với chi phí sửa chữa thiết kế hoặc do lỗi mã.

5.4.1. Những hạn chế của việc đặc tả bằng ngôn ngữ tự nhiên

1. Người đọc và người viết đặc tả bằng ngôn ngữ tự nhiên buộc phải hiểu mỗi từ theo cùng một khái niệm. Điều đó là không thể được do sự mơ hồ và bản tính cố hữu của ngôn ngữ tự nhiên và cũng vì không có một từ điển thuật ngữ máy tính chuẩn mực;
2. Một đặc tả bằng ngôn ngữ tự nhiên là quá mềm dẻo: nó cho phép các yêu cầu liên quan được biểu thị trong các cách hoàn toàn khác nhau;
3. Các yêu cầu là không phân hoạch được một cách rõ ràng: hiệu quả của việc đổi thay chỉ có thể được xác định bởi toàn bộ các yêu cầu chứ không phải là một nhóm các yêu cầu liên quan.

Có người đề nghị rằng các ngôn ngữ đặc tả hình thức toán học nên được dùng để biểu thị các yêu cầu hệ thống. Nhưng đa số các khách hàng lại không hiểu đặc tả hình thức toán học. Hall (1990) đã đề xuất cách để giải quyết khó khăn đó là viết thêm các chú giải dài dòng ngay bên cạnh cho người dùng dễ hiểu. Khi nào người dùng trở nên quen thuộc hơn với những ưu điểm của đặc tả hình thức và các kỹ sư phần mềm không còn miễn cưỡng dùng các phương pháp hình thức thì sẽ có nhiều cách đặc tả yêu cầu dựa trên các mô hình hình thức của chức năng hệ thống.

5.4.2. Các yêu cầu phi chức năng

Một yêu cầu phi chức năng của hệ thống là một hạn chế hoặc ràng buộc về các dịch vụ của hệ thống. Các yêu cầu đó có thể được đưa ra:

- Vì nhu cầu của người dùng
- Vì hạn chế của kinh phí;
- Vì chính sách của tổ chức;
- Vì sự cần thiết tương tác giữa các phần cứng và phần mềm;
- Vì các nhân tố bên ngoài như các quy tắc an toàn, luật lệ bí mật riêng tư ...

Có ba kiểu yêu cầu phi chức năng chính:

1. Các yêu cầu sản phẩm: đây là các yêu cầu về hệ thống được phát triển, chẳng hạn yêu cầu về tốc độ, về bộ nhớ, về độ tin cậy, về tính di chuyển được và về tính dùng lại được;
2. Các yêu cầu về quá trình: đây là các yêu cầu về quá trình phát triển, chẳng hạn như các chuẩn cần phải theo, các yêu cầu về sự thức hiện như các ngôn ngữ lập trình, phương pháp thiết kế, yêu cầu về phân phát;
3. Các yêu cầu ngoại lai: tức là các yêu cầu không phải về sản phẩm cũng không phải về quá trình phát triển; chẳng hạn như về giao tiếp với các hệ thống khác, về pháp lý, về chi phí, về nhóm những người phát triển...

Tùy theo các tổ chức cụ thể, đặc tả yêu cầu có thể được thể hiện bằng các cách khác nhau kể từ mức phát biểu bằng ngôn ngữ tự nhiên về các dịch vụ mà hệ cần cung cấp đến mức đặc tả hệ thống một cách hình thức kiểu toán học. Ranh giới giữa các yêu cầu và đặc tả hình thức là một thứ rất tinh tế và các thuật ngữ này lại còn có thể được dùng theo nghĩa như nhau.

5.4.3. Khó khăn của việc xác định đặc tả yêu cầu

1. Các hệ phần mềm lớn thường được yêu cầu cải thiện dựa trên nguyên trạng: hoặc không có hệ thống nào hoặc có một hệ thống không đầy đủ. Mặc dầu có thể biết các khó khăn của hệ thống hiện có nhưng lại rất khó dự đoán được hiệu quả của hệ thống đã được cải thiện đối với tổ chức đó;
2. Các hệ thống lớn thường có một cộng đồng người sử dụng đa dạng, họ có những yêu cầu và ưu tiên khác nhau, thậm chí là

mâu thuẫn với nhau. Cuối cùng thì các yêu cầu hệ thống cũng không thể tránh được sự thỏa hiệp;

3. Những người bỏ tiền ra mua sắm hệ thống và những người sử dụng hệ thống hiếm khi lại là một. Những người mua sắm hệ thống đặt ra các yêu cầu theo những ràng buộc của tổ chức cơ quan và của ngân sách. Những cái đó lại thường mâu thuẫn với các yêu cầu thực sự của người dùng.

5.4.4. Thẩm định yêu cầu

Một khi đã thiết lập các yêu cầu thì phải thẩm định xem chúng có thỏa mãn các nhu cầu của người mua hệ thống không. Nếu việc thẩm định không phù hợp thì các sai lầm có thể lan truyền sang các giai đoạn thiết kế và thực hiện. Khi đó có thể cần đến sự nâng cấp hệ thống rất tốn kém để làm cho yêu cầu trở nên đúng đắn.

Có bốn vấn đề liên quan đến việc thẩm định yêu cầu:

1. Phải chỉ ra rằng các nhu cầu của người dùng là được thỏa mãn;
2. Các yêu cầu phải không gây ra mâu thuẫn nhau;
3. Các yêu cầu phải đầy đủ: chúng phải chứa mọi chức năng và mọi ràng buộc mà người dùng đã hướng đến;
4. Các yêu cầu phải là hiện thực.

5.5. Dàn bài đặc tả yêu cầu phần mềm

Bản đặc tả các yêu cầu phần mềm được tạo ra tại đỉnh cao của nhiệm vụ phân tích

Chức năng và hiệu năng được cấp phát cho phần mềm xem như một phần của kỹ nghệ hệ thống thường được làm mịn bằng cách thiết lập phần mô tả thông tin đầy đủ như mô tả chức năng chi tiết, chỉ báo về các yêu cầu hiệu năng và những ràng buộc thiết kế, các tiêu chuẩn hợp lệ thích hợp và những dữ liệu khác thường cần tới. Cơ quan tiêu chuẩn quốc gia, IEEE và Bộ Quốc phòng Mỹ đề nghị các định dạng cho các đặc tả yêu cầu phần mềm. Dàn bài đơn giản hóa được trình bày trong bảng sau:

Dàn bài đặc tả yêu cầu phần mềm

I. Giới thiệu	
A. Đại cương về hệ thống	*(mục tiêu, mục đích)*
B. Mô tả chung	*(cấu trúc)*
C. Các ràng buộc dự án phần mềm	*(phạm vi)*
II. Mô tả thông tin	*(chi tiết)*
A. Biểu diễn luồng thông tin	*(dựa vào cấu trúc thông tin)*
1. Luồng dữ liệu	
2. Luồng điều khiển	*(đặc biệt chú ý đến khi xét các hệ thời gian thực)*
B. Biểu diễn nội dung thông tin	
C. Mô tả giao diện hệ thống	*(cho từng chức năng)*
III. Mô tả chức năng	
A. Phân hoạch chức năng	*(cho từng chức năng)*
B. Mô tả chức năng	*(lời giải thích)*
1. Tường thuật về cách xử lý	
2. Hạn chế/giới hạn	*(về các mặt kinh tế, xã hội, kỹ thuật, chi phí, thời gian)*
3. Yêu cầu hiệu năng	*(theo từng modul)*
4. Ràng buộc thiết kế	*(có luận giải)*
5. Biểu đồ trợ giúp	*(cấu trúc tổng thể, tương quan với các phần tử hệ thống khác)*
C. Mô tả điều khiển	
1. Đặc tả điều khiển	*(có luận giải)*
2. Ràng buộc thiết kế	*(xem xét vận hành của phần mềm)*
IV. Mô tả hành vi	*(xem xét vận hành của phần mềm)*
A. Trạng thái hệ thống	*(mức toàn cảnh, mức đỉnh)*
B. Sự kiện và hành động	*(mức cụ thể, bộ phận)*
V. Tiêu chuẩn hợp lệ	
A. Giới hạn hiệu năng	
B. Lớp các kiểm thử	
C. Đáp ứng phần mềm trông đợi	*(tiêu chuẩn hướng tới (lý tưởng))*
D. Các xem xét đặc biệt	
VI. Sách tham khảo	*(tài liệu kỹ nghệ phần mềm khác, tham khảo kỹ thuật...)*
VII. Phụ lục	*(Danh sách các nhà cung cấp, các chuẩn...*

Trong nhiều trường hợp bản đặc tả yêu cầu phần mềm còn có thể có kèm theo một bản mẫu thực hiện được (mà trong một số trường hợp có thể thay thế được cho bản đặc tả), một bản mẫu trên giấy, hay tài liệu sơ bộ của người dùng. Bản tài liệu sơ bộ của người dùng trình bày phần mềm như hộp đen, tức là chủ yếu nhấn mạnh vào cái vào của người dùng và cái ra kết quả. Tài liệu này có thể dùng như một công cụ có giá trị để làm lộ ra vấn đề ở giao diện người - máy.

5.6. Xét duyệt đặc tả

Việc xét duyệt bản đặc tả các yêu cầu phần mềm (và hoặc bản mẫu) do cả người phát triển phần mềm và khách hàng cùng tiến hành. Bởi vì đặc tả tạo nên nền tảng cho giai đoạn phát triển nên cần phải cực kỳ cẩn thận trong khi tiến hành cuộc họp xét duyệt. Có hai mức xét duyệt.

5.6.1. Mức vĩ mô

Việc xét duyệt trước hết được tiến hành ở mức vĩ mô. Tại mức này, người xét duyệt cố gắng đảm bảo rằng bản đặc tả được đầy đủ, nhất quán và chính xác. Cần đề cập tới các câu hỏi sau:

- Các mục tiêu và mục đích được thiết lập cho phần mềm có nhất quán với mục tiêu và mục đích của hệ thống hay không?
- Những giao diện quan trọng với mọi phần tử hệ thống đã được mô tả chưa?
- Luồng và cấu trúc thông tin đã được mô tả thích hợp cho phạm vi vấn đề chưa?
- Các biểu đồ có rõ ràng không? Liệu mỗi biểu đồ có thể đứng riêng mà không cần lời giải thích không?
- Các chức năng chính có còn bên trong phạm vi và đã được mô tả thích hợp chưa?
- Liệu hành vi của phần mềm có nhất quán với thông tin nó phải xử lý và chức năng nó phải thực hiện hay không?
- Các ràng buộc thiết kế có thực tế không?
- Rủi ro công nghệ phát triển là gì?
- Các yêu cầu phần mềm khác đã được xem xét đến chưa?
- Các tiêu chuẩn hợp lệ đã được phát biểu chi tiết chưa?
- Liệu có sự không nhất quán, bỏ sót hay dư thừa nào không?

- Việc tiếp xúc với khách hàng có đầy đủ không?
- Người dùng đã xét duyệt bản Tài liệu sơ bộ của người dùng hay bản mẫu chưa?
- Các ước lượng về Kế hoạch dự án phần mềm bị ảnh hưởng thế nào?
- ...

5.6.2. Mức chi tiết

Để đưa ra câu trả lời cho nhiều câu hỏi trên, việc xét duyệt có thể tập trung vào mức chi tiết. Tại đây, mối quan tâm tập trung vào từ ngữ của bản đặc tả.

Việc xét duyệt chi tiết bản đặc tả có những gợi ý sau:

- Phải quan sát các từ nối có sức thuyết phục (như "chắc chắn", "do đó", "rõ ràng", "hiển nhiên", "từ đó suy ra rằng") và hỏi "tại sao chúng lại có ở đó?"
- Theo dõi những thuật ngữ mung lung (như "một số", "đôi khi", "thường", "thông thường", "bình thường", "phần lớn", "đa số"); để yêu cầu làm sáng tỏ.
- Khi có nêu danh sách, nhưng không đầy đủ thì phải đảm bảo mọi khoản mục đều được hiểu rõ. Chú ý vào các từ như "vân vân", "cứ như thế", "cứ tiếp tục như thế", "sao cho" ...
- Phải chắc chắn phát biểu phạm vi không chứa những giả thiết không được nói rõ (như "mã hợp lệ trong khoảng 10 tới 100". Đó là số nguyên, số thực hay số hệ 16?).
- Phải nhận biết về các động từ mơ hồ như "xử lý", "loại bỏ". Có thể có nhiều cách hiểu về nó.
- Phải nhận biết các đại từ "vu vơ" (như "modul vào/ra liên lạc với modul kiểm tra tính hợp lệ dữ liệu và đặt cờ báo kiểm soát của nó").
- Tìm các câu có chứa sự chắc chắn (như "bao giờ", "mọi", "tất cả", "không một", "không bao giờ") rồi yêu cầu bằng chứng.
- Khi một thuật ngữ được định nghĩa tường minh tại một chỗ thì hãy thử thay thế định nghĩa này vào chỗ xuất hiện khác của nó.

- Khi một cấu trúc được mô tả theo lời thì hãy vẽ ra bức tranh để giúp hiểu được nó.
- Khi một tính toán được xác định thì hãy thử với ít nhất 2 ví dụ.
- ...

Một khi việc xét duyệt đã hoàn tất thì bản đặc tả các yêu cầu phần mềm sẽ được cả khách hàng lẫn người phát triển "ký tắt". Bản đặc tả trở thành "hợp đồng" cho việc phát triển phần mềm. Những thay đổi trong yêu cầu được nêu ra sau khi bản đặc tả đã hoàn thành sẽ không bị loại bỏ, nhưng khách hàng phải lưu ý rằng từng thay đổi sau khi ký hợp đồng đều là một mở rộng của phạm vi phần mềm và do đó có thể làm tăng thêm chi phí và/hoặc kéo dài thời gian thực hiện.

Bản đặc tả rất khó "kiểm thử" theo một cách có nghĩa, do đó sự không nhất quán hay thiếu sót có thể bị bỏ qua không chú ý tới. Trong khi xét duyệt, người ta có thể khuyến cáo những thay đổi cho bản đặc tả. Có thể sẽ khó khăn để định lượng tác động toàn cục của thay đổi, tức là làm sao việc thay đổi trong một chức năng lại ảnh hưởng tới các yêu cầu cho chức năng khác? Người ta đã phát triển các công cụ đặc tả tự động hóa để giúp giải quyết vấn đề này.

6. KỸ NGHỆ HỆ THỐNG VÀ TẠO NGUYÊN MẪU

6.1. Kỹ nghệ hệ thống

6.1.1. Các hoạt động cơ bản trong tiến trình phân tích hệ thống

Bước 1: Xác định nhu cầu

Bước đầu tiên của tiến trình phân tích hệ thống bao gồm việc xác định nhu cầu. Để bắt đầu, người phân tích giúp cho khách hàng trong việc xác định các mục tiêu của hệ thống (sản phẩm):

- Thông tin nào cần phải tạo ra?
- Thông tin nào cần được cung cấp?
- Cần những chức năng và hiệu suất nào?

Người phân tích phải phân biệt được giữa "nhu cầu" của khácH hàng (những tính năng chủ chốt cho sự thành công của hệ thống) và "điều mong muốn" của khácH hàng (những tính năng tốt nên có nhưng không bản chất).

Một khi các mục tiêu tổng thể đã được xác định thì nhà phân tích chuyển sang việc đánh giá các thông tin phụ:

- Liệu có công nghệ để xây dựng hệ thống không?
- Cần có những tài nguyên chế tạo và phát triển đặc biệt nào?
- Cần phải đặt giới hạn nào về chi phí và lịch biểu?

Nếu hệ thống mới thực tế là một sản phẩm để bán cho nhiều khách hàng thì nên có những câu hỏi sau:

- Đâu là thị trường tiềm năng cho sản phẩm này?
- Sản phẩm này so với các sản phẩm cạnh tranh khác như thế nào?
- Sản phẩm này sẽ giữ vị trí nào trong tuyến sản phẩm của cả công ty?

Thông tin thu được trong bước xác định nhu cầu được kết tinh trong *Tài liệu quan niệm về hệ thống*. Tài liệu quan niệm nguyên bản đôi khi được khách hàng chuẩn bị trước cuộc gặp gỡ với người phân tích. Sự trao đổi thường xuyên giữa khách hàng - người phân tích tạo ra những thay đổi cho tài liệu này.

Bước 2: Nghiên cứu khả thi

Mọi dự án đều khả thi với nguồn tài nguyên vô hạn và thời gian vô hạn. Nhưng việc xây dựng hệ thống lại phải làm với sự hạn hẹp về tài nguyên và khó (nếu không phải là bất khả thi) và bảo đảm đúng ngày bàn giao. Cho nên cần phải thận trọng trong đánh giá tính khả thi của dự án từ thời điểm sớm nhất có thể được.

Phân tích khả thi và rủi ro có liên quan với nhau theo nhiều cách. Nếu rủi ro của dự án là lớn, thì tính khả thi của việc chế tạo phần mềm chất lượng sẽ bị giảm đi.

Trong kỹ nghệ hệ thống, chúng ta tập trung vào bốn lĩnh vực chính:

1. Khả thi về kinh tế: đánh giá về chi phí phát triển cần phải cân xứng với lợi tức cuối cùng hay lợi ích mà hệ thống được xây dựng đem lại.
2. Khả thi về kỹ thuật: khảo cứu về chức năng, hiệu suất và ràng buộc có thể ảnh hưởng tới khả năng đạt tới một hệ thống chấp nhận được.

3. Khả thi về hợp pháp: quy định của pháp luật về sự xâm phạm, vi phạm hay khó khăn nào có thể gây ra từ việc xây dựng hệ thống.

4. Khả thi về phương án: đánh giá tính khả thi của phương án đã xác định để tiếp cận tới việc xây dựng hệ thống.

Khảo cứu khả thi không tiến hành cho các hệ thống, trong đó việc biện minh kinh tế là hiển nhiên, rủi ro kỹ thuật thấp, ít các vấn đề pháp lý và không có các phương pháp hợp lý nào khác. Tuy nhiên, nếu bất kỳ điều kiện nói trên không được đáp ứng thì phải tiến hành khảo cứu khả thi.

Luận chứng kinh tế nói chung là xem xét "nền tảng" cho hầu hết các hệ thống (các ngoại lệ là hệ thống quốc phòng, hệ thống luật, các ứng dụng công nghệ cao như chương trình không gian vũ trụ).

Luận chứng kinh tế bao gồm:

- Các mối quan tâm, kể cả phân tích chi phí - lợi ích;
- Chiến lược lợi tức dài hạn của công ty;
- Ảnh hưởng tới các trung tâm hay sản phẩm lợi nhuận khác;
- Chi phí cho tài nguyên cần cho việc xây dựng và phát triển thị trường tiềm năng.

Khả thi kỹ thuật thường là lĩnh vực khó thâm nhập nhất tại giai đoạn này trong tiến trình phát triển hệ thống. Điều thực chất là tiến trình phân tích và xác định nhu cầu cần được tiến hành song song với việc xác định chúng.

Các xem xét thường được gắn với tính khả thi kỹ thuật bao gồm:

- Rủi ro xây dựng: liệu các phần tử hệ thống có thể được thiết kế sao cho chức năng và hiệu suất cần thiết làm lộ rõ những ràng buộc trong khi phân tích không?
- Có sẵn tài nguyên: có sẵn các nhân viên cho việc xây dựng phần tử hệ thống đang xét không? Các tài nguyên cần thiết khác (phần cứng và phần mềm) có sẵn cho việc xây dựng hệ thống không?
- Công nghệ: công nghệ liên quan đã đạt tới trạng thái sẵn sàng hỗ trợ cho hệ thống chưa?

Người phát triển các hệ thống về bản chất đều lạc quan. Tuy nhiên, trong đánh giá tính khả thi kỹ thuật, việc đánh giá sai trong giai đoạn này cũng có thể là một thảm họa.

Tính khả thi pháp lý bao gồm một phạm vi rộng các mối quan tâm kể cả hợp đồng, nghĩa vụ pháp lý; việc đánh giá sai trong giai đoạn này cũng có thể là một thảm họa.

Mức độ các phương án được xem xét tới thường bị giới hạn bởi ràng buộc chi phí và thời gian.

Có thể soạn tư liệu về nghiên cứu khả thi thành một báo cáo riêng cho cấp quản lý trên và đính kèm như phụ lục cho đặc tả hệ thống. Mặc dầu định dạng của báo cáo khả thi có thể thay đổi nhưng bản đại cương dưới đây đã bao quát được hầu hết những điểm quan trọng:

Bản đại cương về nghiên cứu khả thi

I. Giới thiệu	*A. Phát biểu vấn đề* *B. Môi trường thực hiện* *C. Ràng buộc*
II. Tóm tắt quản lý và khuyến cáo	*A. Những tóm lược quan trọng* *B. Bình luận* *C. Khuyến cáo* *D. Tác động*
III. Các phương án	*A. Các cấu hình hệ thống theo phương án* *B. Các tiêu chuẩn được dùng trong chọn lựa cách tiếp cận cuối cùng*
IV. Mô tả hệ thống	*A. Phát biểu vắn tắt về phạm vi* *B. Tính khả thi của các phần tử trong hệ thống*
V. Phân tích chi phí - lợi ích	
VI. Đánh giá rủi rõ kỹ thuật	
VII. Các chi nhánh pháp lý	
VIII. Các chủ đề khác chuyên cho dự án	
IX. Kết luận	

Bản nghiên cứu khả thi trước tiên được cấp quản lý dự án xem xét (để đánh giá độ tin cậy nội dung) rồi đến cấp quản lý cao hơn (để đánh giá trạng thái dự án). Bản nghiên cứu phải đề xuất quyết định "tiến hành/không tiến hành". Cũng cần chú ý rằng các quyết định tiến hành

hay không tiến hành sẽ được đưa ra trong các bước lập kế hoạch, đặc tả và xây dựng cho cả công nghệ phần mềm và phần cứng.

Bước 3: Mô hình hóa hệ thống

Việc mô hình hóa và mô phỏng hệ thống được sử dụng để loại bỏ những điều bất ngờ khi xây dựng hệ thống. Những công cụ CASE được áp dụng trong các tiến trình kỹ nghệ hệ thống. Vai trò của công cụ mô hình hóa và mô phỏng được tóm tắt như sau:

- Cung cấp một phương án cho tiến trình thiết kế, cài đặt và kiểm thử thông qua việc tiếp cận các công cụ mô hình hoá (một công cụ mô hình hóa và mô phỏng);
- Cho phép xây dựng một mô hình đề cập tới tất cả các vấn đề luồng chức năng và dữ liệu thông thường, đồng thời còn bao quát cả các khía cạnh hành động, hành vi của hệ thống. Có thể kiểm thử mô hình này bằng các công cụ phục vụ giám định và gỡ lỗi cho đặc tả và tìm kiếm thông tin;
- Dùng để "kiểm thử sự hoạt động" của đặc tả hệ thống: bằng cách kiểm thử mô hình (đặc tả), người kỹ sư hệ thống có thể hình dung được cách thức mà hệ thống sẽ hành xử ra sao khi được cài đặt. Người ta có thể sử dụng câu hỏi cái gì xảy ra nếu đi theo các kịch bản; xác định, kiểm tra rằng những tình huống mong muốn nào đó có xuất hiện hay không, và các tình huống không mong muốn khác sẽ không xuất hiện hay lại xuất hiện.

6.1.2 Đặc tả hệ thống

Bản Đặc tả hệ thống:

- Là một tài liệu làm nền tảng cho kỹ nghệ phần cứng, kỹ nghệ phần mềm, kỹ nghệ cơ sở dữ liệu và kỹ nghệ con người.
- Mô tả về chức năng và hiệu suất của hệ thống và những ràng buộc hệ thống.
- Quy định cả giới hạn cho từng phần tử hệ thống. Chẳng hạn, chỉ dẫn về vai trò của phần mềm trong hệ thống và các hệ thống con người khác nhau được mô tả trong biểu đồ luồng kiến trúc
- Mô tả thông tin (dữ liệu và điều khiển) vào/ ra khỏi hệ thống.

Dàn bài về bản đặc tả hệ thống sẽ được trình bày sau đây. Tuy nhiên cần lưu ý rằng đây chỉ là một trong nhiều dàn bài có thể được dùng để định nghĩa một tài liệu mô tả hệ thống. Định dạng và nội dung thực tế có thể còn tùy vào các chuẩn kỹ nghệ hệ thống hay phần mềm. Nó được điều chỉnh theo yêu cầu và tính ưa chuộng của người dùng.

Dàn bài đặc tả hệ thống

I. Giới thiệu	*A. Phạm vi và mục tiêu của dự án* *B. Tổng quan* *1. Mục tiêu* *2. Ràng buộc*
II. Mô tả chức năng và dữ liệu	*A. Kiến trúc hệ thống* *1. Biểu đồ ngữ cảnh kiến trúc* *2. Mô tả về biểu đồ ngữ cảnh hệ thống*
III. Mô tả các hệ thống con	*A. Đặc tả biểu đồ kiến trúc cho hệ con **n*** *1. Biểu đồ luồng kiến trúc* *2. Chú giải modul hệ thống* *3. Vấn đề về hiệu suất* *4. Ràng buộc thiết kế* *5. Cách tạo các thành phần hệ thống* *B. Từ điển kiến trúc* *C. Biểu đồ và mô tả liên nối hệ thống*
IV. Các kết quả mô hình hóa và mô phỏng hệ thống	*A. Mô hình hệ thống được dùng cho mô phỏng* *B. Kết quả mô phỏng* *C. Vấn đề hiệu suất đặc biệt*
V. Vấn đề dự án	*A. Chi phí xây dựng dự phòng* *B. Lịch biểu dự phòng*
VI. Phụ lục	

Xét duyệt đặc tả hệ thống

Cuộc họp xét duyệt đặc tả hệ thống đánh giá tính đúng đắn của định nghĩa chứa trong bản đặc tả hệ thống. Cuộc họp được cả người phát triển và khách hàng đảm bảo rằng:

1. Phạm vi của dự án đã được vạch ra đúng;
2. Các chức năng, hiệu suất và giao diện đã được định nghĩa đúng;
3. Phân tích rủi ro môi trường và tính khả thi của dự án;
4. Người phát triển và khách hàng có cùng cảm nhận về mục tiêu hệ thống.

Cuộc họp đặc tả hệ thống được tiến hành trong giai đoạn

- Đưa ra những quan điểm quản lý áp dụng cho hệ thống;
- Tiến hành đánh giá kỹ thuật về các phần tử và chức năng hệ thống.

Về khía cạnh quản lý, đặc tả hệ thống cần phải thỏa mãn những câu hỏi sau:

- Nhu cầu kinh doanh của hãng đã được thiết lập chưa? Luận chứng hệ thống có nghĩa không?
- Có cần môi trường (hay thị trường) riêng cho hệ thống đã được mô tả hay không?
- Những phương án nào đã được xem xét?
- Rủi ro phát triển cho từng phần tử hệ thống là gì?
- Các tài nguyên đã có sẵn cho việc xây dựng hệ thống chưa?
- Các giới hạn chi phí và lịch biểu có ý nghĩa gì không?

Các câu hỏi trên nên được đặt ra và trả lời một cách đều đặn trong nhiệm vụ phân tích. Mỗi câu hỏi nên được xem xét lại tại giai đoạn đánh giá kỹ thuật.

Mức độ chi tiết trong giai đoạn đánh giá kỹ thuật (họp xét duyệt hệ thống) phụ thuộc vào mức độ chi tiết trong nhiệm vụ xác định ban đầu.

Việc xét duyệt nên bao gồm các vấn đề sau:

- Về chức năng của hệ thống: có phù hợp với những định giá về rủi ro phát triển, chi phí và lịch biểu hay không?
- Các chức năng được xác định đã đủ chi tiết chưa?
- Giao diện giữa các phần tử hệ thống và với môi trường đã được xác định đủ chi tiết chưa?
- Các vấn đề độ tin cậy, hiệu suất và bảo trì đã được đề cập tới

trong bản đặc tả chưa?

- Liệu bản đặc tả hệ thống có cung cấp đủ nền tảng cho các bước kỹ nghệ phần cứng và phần mềm tiếp sau không?

Kết luận:

Sau cuộc họp xét duyệt, tiến hành các tiến trình kỹ nghệ tương ứng với các phần tử hệ thống chủ chốt như phần mềm, phần cứng, con người và CSDL. Các phần tử phần cứng, con người và CSDL của hệ thống được đề cập tới như phần tương ứng của các tiến trình kỹ nghệ.

Tại bước phân tích hệ thống người phân tích xác định ra nhu cầu của khách hàng, xác định tính khả thi kinh tế - kỹ thuật, xác định chức năng và hiệu suất cho các phần tử hệ thống chủ chốt (phần mềm, phần cứng, con người và CSDL).

Bản đặc tả hệ thống (tài liệu nền tảng cho toàn bộ công việc kỹ nghệ tiếp sau đó) được coi là đỉnh cao của nhiệm vụ kỹ nghệ hệ thống

Phải đảm bảo việc trao đổi liên lạc đều đặn giữa khách hàng và nhà phân tích vì nếu trao đổi giữa nhà phân tích và khách hàng bị gián đoạn tại giai đoạn này thì sự thành công của toàn bộ dự án sẽ bị đe dọa

Khó khăn là ở chỗ phải lập các tư liệu hệ thống sao cho:

- Dễ hiểu cho người sử dụng;
- ạo ra bản đặc tả hệ thống (cùng với đặc tả yêu cầu, được dùng làm cơ sở cho một hợp đồng giữa người mua và người cung cấp phần mềm đó). Nói chung, người dùng ưa thích một mô tả hệ thống trừu tượng chứ không thích một đặc tả chi tiết.

6.2. Tạo nguyên mẫu (prototype)

Duyệt lại là một phần của quá trình thẩm định yêu cầu. Từ đặc tả yêu cầu ta có thể tưởng tượng hệ thống sẽ được sử dụng như thế nào. Một chức năng được mô tả trong một đặc tả là hữu ích và đã được xác định tốt nhưng thực tế việc sử dụng chức năng đó kết hợp với các chức năng khác lại để lộ ra rằng cái nhìn ban đầu của người sử dụng đã không đúng và không đầy đủ.

Một cách giải quyết khó khăn đó là phát triển một nguyên mẫu hệ thống cho phép người sử dụng thí nghiệm trên nguyên mẫu đó. Các yêu cầu của hệ thống được thẩm định và người dùng sẽ phát hiện ra các sai sót.

Sự khác biệt giữa nguyên mẫu phần mềm với nguyên mẫu phần cứng:

Nguyên mẫu phần mềm hoàn toàn khác với nguyên mẫu phần cứng. Khi phát triển các hệ thống phần cứng, thì thực tế người ta phát triển một nguyên mẫu hệ thống để thẩm định thiết kế hệ thống. Một nguyên mẫu hệ thống điện tử có thể được thực hiện và được thử nghiệm bằng cách dùng các thành phần chưa được ráp vào vỏ trước khi đầu tư vào các mạch tích hợp chuyên dụng đắt tiền để thực hiện một đời sản phẩm mới của hệ thống. Một nguyên mẫu phần mềm là không phải nhằm vào việc thẩm định thiết kế (thiết kế của nó thường là hoàn toàn khác với hệ thống đã phát triển cuối cùng), mà là để thẩm định yêu cầu của người sử dụng phần mềm.

6.2.1. Lợi ích của việc phát triển nguyên mẫu

Lợi ích của việc phát triển nguyên mẫu ngay từ sớm trong quá trình phát triển phần mềm là:

1. Sự hiểu lầm giữa những người phát triển phần mềm và những người sử dụng phần mềm có thể được nhận thấy rõ khi các chức năng của hệ thống được trình diễn;
2. Sự thiếu hụt các dịch vụ người dùng có thể được phát hiện;
3. Sự khó sử dụng hoặc sự nhầm lẫn các dịch vụ người dùng có thể được thấy rõ và được sửa sang lại;
4. Đội ngũ phát triển phần mềm có thể tìm ra được các yêu cầu không đầy đủ hoặc không kiên định khi họ phát triển nguyên mẫu;
5. Một hệ thống hoạt động được (mặc dầu là hạn chế) là bằng chứng thuyết minh cho tính khả thi và tính hữu ích của ứng dụng đó cho các nhà quản lý;
6. Nguyên mẫu đó được dùng làm cơ sở cho việc viết đặc tả cho sản phẩm.

Mặc dù mục tiêu chủ yếu của việc tạo nguyên mẫu là thẩm định các yêu cầu phần mềm nhưng cũng nên chỉ ra rằng một nguyên mẫu phần mềm cũng có các ứng dụng khác:

1. Dùng để huấn luyện người sử dụng ngay từ trước khi hệ thống được phân phối;

2. Dùng trong quá trình thử nghiệm hệ thống. Điều đó nghĩa là cùng các trường hợp thử như nhau vừa dùng cho thử nguyên mẫu vừa cho thử hệ thống. Kết quả khác nhau có nghĩa là có sai sót.

Tạo nguyên mẫu là một kỹ thuật giảm bớt rủi ro. Một rủi ro lớn trong việc phát triển phần mềm là các sai sót mà đến giai đoạn cuối mới phát hiện và chỉnh sửa là rất tốn kém. Kinh nghiệm cho thấy rằng việc tạo nguyên mẫu sẽ giảm bớt số các vấn đề của đặc tả yêu cầu và tổng chi phí của việc phát triển có thể là thấp hơn nếu ta phát triển nguyên mẫu.

6.2.2. Các giai đoạn trong việc phát triển nguyên mẫu

1. Thiết lập các mục tiêu của việc tạo nguyên mẫu, chọn input, output cho nguyên mẫu;
2. Chọn các chức năng cho việc tạo nguyên mẫu và quyết định những đặc tả phi chức năng nào cần có trong nguyên mẫu;
3. Phát triển nguyên mẫu;
4. Đánh giá hệ nguyên mẫu.

Cần phải làm rõ ràng các mục tiêu tạo nguyên mẫu trước khi bắt đầu công việc. Mục tiêu đó có thể là phát triển một hệ thống liên quan chủ yếu đến tạo nguyên mẫu giao diện người dùng; nó cũng có thể là việc phát triển một hệ thống thẩm định các yêu cầu hệ thống chức năng, nó cũng có thể là phát triển một hệ thống để trình diễn tính khả thi của ứng dụng cho các nhà quản lý xem xét... Cùng một nguyên mẫu không thể đạt được tất cả các mục tiêu. Nếu mục tiêu còn chưa rõ ràng thì người quản lý hoặc người sử dụng có thể hiểu nhầm chức năng của nguyên mẫu và không đạt được lợi ích thiết thực đầy đủ của việc tạo nguyên mẫu.

Giai đoạn tiếp theo trong quá trình này là quyết định xem cái gì được đưa vào và cái gì được lấy ra từ hệ nguyên mẫu đó. Nguyên mẫu của phần mềm là đắt nếu nó được thực hiện bằng cách dùng cùng các công cụ và các chuẩn y như cho chính hệ thống cuối cùng

Bởi vậy, có thể quyết định tạo nguyên mẫu cho tất cả các chức năng của hệ thống nhưng ở mức thu gọn. Cũng có thể chỉ một số chức năng hệ thống được tạo mẫu. Bình thường trong thực tế người ta hay bỏ bớt các yêu cầu phi chức năng chẳng hạn như yêu cầu về tốc độ, về không gian nhớ. Việc xử lý sai sót và việc quản trị có thể được bỏ qua và có thể là thừa khi mục tiêu của việc tạo mẫu là thiết lập một giao diện

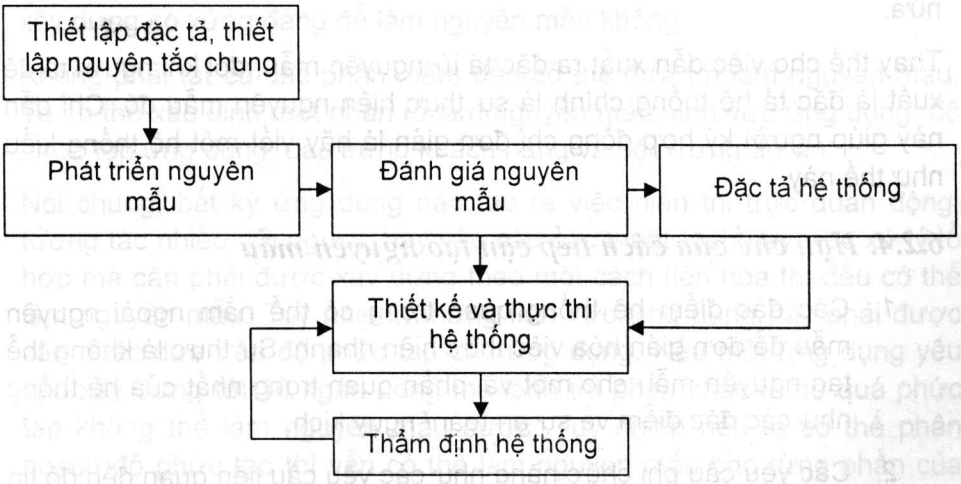
người máy. Các chuẩn của độ tin cậy và của chất lượng chương trình cũng có thể chưa tính đến.

Giai đoạn cuối cùng của quá trình này là đánh giá nguyên mẫu. Có người cho rằng đây là giai đoạn quan trọng nhất của việc tạo nguyên mẫu. Phải dự tính cho việc huấn luyện người sử dụng trong giai đoạn này và các mục tiêu của việc tạo mẫu hẳn là nên dẫn ra một kế hoạch đánh giá. Phải có đủ thời gian cho người sử dụng làm quen với hệ thống và thiết lập mẫu sử dụng chuẩn tắc và bởi vậy phát hiện được các sai sót yêu cầu.

Việc tạo nguyên mẫu là kỹ thuật chính trong mô hình quá trình xoắn ốc hỗ trợ định giá rủi ro.

6.2.3. Tạo nguyên mẫu trong tiến trình phần mềm

Vấn đề cơ bản là khó đánh giá phần mềm mới được xây dựng có ảnh hưởng thế nào tới công việc của người dùng. Đối với các hệ thống mới, đặc biệt là đối với các hệ thống lớn và phức tạp thì có lẽ là không thể đánh giá được trước khi hệ thống được xây dựng xong và đưa vào ứng dụng.



H2.5. Việc tạo nguyên mẫu trong quá trình phần mềm

Một cách để vượt qua khó khăn là sử dụng cách tiếp cận thăm dò để phát triển hệ thống. Điều này có nghĩa là trình bày cho người dùng một hệ thống chưa đầy đủ và rồi cải biên, tăng cường hệ thống đó khi mà các yêu cầu thực của người dùng trở nên trong suốt. Sau khi đánh giá nguyên mẫu đó sẽ bị loại và một hệ chất lượng tốt hơn sẽ được xây dựng.

Sự khác biệt của hai cách tiếp cận:

1. Lập trình thăm dò: bắt đầu với một hiểu biết mơ hồ về yêu cầu hệ thống và hệ thống sẽ được tăng cường một khi các yêu cầu mới được phát hiện. Bản đặc tả hệ thống không được xây dựng và sự thật là hệ thống được phát triển theo cách tiếp cận này thì không thể đặc tả được (ví dụ: một vài hệ trí tuệ nhân tạo là không thể đặc tả được).
2. Cách tiếp cận tạo nguyên mẫu:
   - Nhằm phát hiện ra đặc tả hệ thống và kết quả của giai đoạn phát triển nguyên mẫu là bản đặc tả đó;
   - Mở rộng quá trình phân tích yêu cầu nhằm rút bớt tổng chi phí cho toàn bộ vòng đời phần mềm;
   - Làm sáng tỏ các yêu cầu;
   - Cung cấp các thông tin phụ cho người quản lý để họ đánh giá các rủi ro của dự án.

Sau khi đánh giá, nguyên mẫu đó không dùng để phát triển hệ thống nữa.

Thay thế cho việc dẫn xuất ra đặc tả từ nguyên mẫu, đôi khi người ta đề xuất là đặc tả hệ thống chính là sự thực hiện nguyên mẫu đó. Chỉ dẫn này giúp người ký hợp đồng chỉ đơn giản là hãy viết một hệ thống kiểu như thế này.

6.2.4. Hạn chế của cách tiếp cận tạo nguyên mẫu

1. Các đặc điểm hệ thống quan trọng có thể nằm ngoài nguyên mẫu để đơn giản hóa việc thực hiện nhanh. Sự thực là không thể tạo nguyên mẫu cho một vài phần quan trọng nhất của hệ thống như các đặc điểm về sự an toàn/ nguy kịch.
2. Các yêu cầu phi chức năng như các yêu cầu liên quan đến độ tin cậy, tính mâu thuẫn và độ an toàn là không thể biểu thị đầy đủ trong khi thực hiện nguyên mẫu.
3. Người dùng có thể không dùng nguyên mẫu y như cách dùng một hệ đang hoạt động.

Có một vài lý do dẫn tới việc tái tạo nguyên mẫu khi một hệ thống lớn và tuổi thọ cao được phát triển.

1. Các đặc trưng hệ thống quan trọng chẳng hạn như sự thực thi, an ninh, tính không mâu thuẫn, độ tin cậy có thể được lờ đi khi tạo nguyên mẫu để đẩy nhanh việc thực thi nguyên mẫu. Bản chất của nguyên mẫu có thể lại là những đặc trưng đó không thể thêm vào nguyên mẫu.
2. Trong quá trình phát triển nguyên mẫu thì nguyên mẫu sẽ bị thay đổi để phản ánh các nhu cầu của người dùng và dường như các thay đổi đó sẽ được tiến hành theo một cách không thể không chế được.
3. Các thay đổi trong quá trình phát triển nguyên mẫu sẽ có thể làm giảm sút cấu trúc hệ thống đến mức mà do yêu cầu bảo trì nên các thay đổi tiếp theo càng ngày càng trở nên khó khăn hơn.

Phát triển nguyên mẫu là dễ quản lý hơn lập trình thăm do vì các chuẩn quá trình phần mềm là được tuân theo. Các kế hoạch và các tư liệu có thể viết thêm cho mỗi phần thêm vào.

6.2.5 Các bước tiến hành làm nguyên mẫu phần mềm

Bước 1: Đánh giá yêu cầu phần mềm và xác định liệu phần mềm cần xây dựng có xứng đáng để làm nguyên mẫu không.

Không phải tất cả các phần mềm đều có thể đưa tới làm nguyên mẫu. Ta có thể xác định một nhân tố làm nguyên mẫu: lĩnh vực ứng dụng, độ phức tạp ứng dụng, đặc trưng khách hàng và đặc trưng dự án.

Nói chung, bất kỳ ứng dụng nào tạo ra việc hiển thị trực quan động, tương tác nhiều với người, hay yêu cầu các thuật toán hay việc xử lý tổ hợp mà cần phải được xây dựng theo một cách tiến hóa thì đều có thể làm nguyên mẫu. Tuy nhiên, những lĩnh vực ứng dụng này phải được cân nhắc dựa trên độ phức tạp của ứng dụng. Nếu một ứng dụng yêu cầu xây dựng tới 10 ngàn dòng mã lệnh thì phần chắc là nó quá phức tạp không thể làm nguyên mẫu được. Tuy nhiên nếu ta có thể phân hoạch độ phức tạp thì vẫn có thể làm nguyên mẫu cho từng phần của phần mềm.

Để đảm bảo tính tương tác giữa khách hàng với nguyên mẫu cần:

1. Khách hàng phải cam kết dùng tài nguyên để đánh giá và làm mịn nguyên mẫu

2. Khách hàng phải có khả năng đưa ra những quyết định về yêu cầu một cách kịp thời.

Bản chất của dự án quyết định tính hiệu quả của làm nguyên mẫu.

Bước 2: Với một dự án chấp thuận được, người phân tích sẽ tiến hành xây dựng một biểu diễn vắn tắt các yêu cầu.

Trước khi có thể bắt đầu xây dựng một nguyên mẫu, người phân tích phải biểu diễn miền thông tin và các lĩnh vực hành vi và chức năng của vấn đề rồi xây dựng một cách tiếp cận hợp lý tới việc phân hoạch. Có thể ứng dụng các nguyên lý phân tích nền tảng (trên xuống) và các phương pháp phân tích yêu cầu.

Bước 3: Sau khi đã duyệt mô hình và yêu cầu, tạo ra một đặc tả thiết kế vắn tắt cho nguyên mẫu

Việc thiết kế phải thực hiện trước khi bắt đầu làm nguyên mẫu. Tuy nhiên thiết kế chỉ tập trung chủ yếu vào các vấn đề thiết kế dữ liệu và kiến trúc mức đỉnh chứ không phải tập trung vào thiết kế thủ tục chi tiết.

Bước 4: Phần mềm nguyên mẫu được tạo ra, kiểm thử và làm mịn

Một cách lý tưởng, các khối xây dựng phần mềm hiện có được dùng để tạo ra nguyên mẫu một cách nhanh chóng.

Bước 5: Một khi đã kiểm thử xong nguyên mẫu thì có thể trình bày nó cho khách hàng

Khách hàng sẽ kiểm thử ứng dụng và gợi ý những thay đổi. Bước này là cốt lõi của cách tiếp cận làm nguyên mẫu. Chính ở bước này khách hàng có thể xem xét cách biểu diễn được cài đặt cho yêu cầu phần mềm, gợi ý những thay đổi để phần mềm đáp ứng tốt hơn với các nhu cầu thực tế.

Bước 6: Lặp lại các bước 4 và 5 cho tới khi tất cả các yêu cầu đã được hình thức hóa hay cho tới khi nguyên mẫu đã tiến hóa thành một hệ thống sản phẩm cuối cùng.

Khuôn mẫu làm nguyên mẫu có thể được tiến hành với một trong hai mục tiêu:

1. Mục tiêu của việc làm nguyên mẫu là thiết lập một tập hợp các yêu cầu hình thức có thể được dịch thành phần mềm (sản xuất bằng các phương pháp và kỹ thuật của kỹ nghệ phần mềm);

2. Mục tiêu của việc làm nguyên mẫu là cung cấp động lực liên tục thúc đẩy phát triển tiến hóa phần mềm.

6.2.6. Các phương pháp và công cụ làm nguyên mẫu

Để việc làm nguyên mẫu phần mềm được hiệu quả, phải xây dựng nhanh chóng nguyên mẫu sao cho khách hàng có thể định giá được kết quả và khuyến cáo những thay đổi.

Để xây dựng nguyên mẫu nhanh, hiện có ba lớp phương pháp và công cụ sẵn có:

1. Các kỹ thuật thế hệ 4;
2. Thành phần phần mềm dùng lại;
3. Đặc tả hình thức và môi trường làm nguyên mẫu.

Kỹ thuật thế hệ 4

Các kỹ thuật thế hệ 4 (4GT) bao gồm một phạm vi rộng các ngôn ngữ hỏi CSDL và làm báo cáo, các bộ sinh chương trình và ứng dụng các ngôn ngữ phi thủ tục cấp rất cao khác. Bởi vì 4GT sinh được mã thực hiện rất nhanh nên chúng là lý tưởng cho việc làm nguyên mẫu nhanh. Mặc dầu lĩnh vực ứng dụng 4GT hiện tại vẫn còn bị giới hạn vào các hệ thông tin kinh doanh, những công cụ cho các ứng dụng kỹ nghệ đang bắt đầu chiếm ưu thế.

Thành phần phần mềm dùng lại

Một cách tiếp cận khác với việc làm nguyên mẫu nhanh là lắp ráp, thay vì xây dựng nguyên mẫu bằng cách dùng một tập hợp các thành phần phần mềm đã có sẵn. Một thành phần phần mềm có thể là một cấu trúc dữ liệu (hay CSDL) hay một thành phần kiến trúc phần mềm (như chương trình) hay một thành phần thủ tục (như một modul). Trong mỗi trường hợp thành phần phần mềm phải được thiết kế theo cách thức làm cho nó có thể dùng lại được mà không cần biết chi tiết về cách làm việc bên trong.

Việc làm nguyên mẫu với việc dùng lại các thành phần chương trình chỉ hoạt động được nếu hệ thống thư viện đã được phát triển sao cho các thành phần thực tồn tại có thể được đưa vào danh mục rồi tìm kiếm.

Cũng nên lưu ý rằng một sản phẩm phần mềm hiện có, có thể được dùng như nguyên mẫu cho một sản phẩm "mới", hay "được cải tiến".

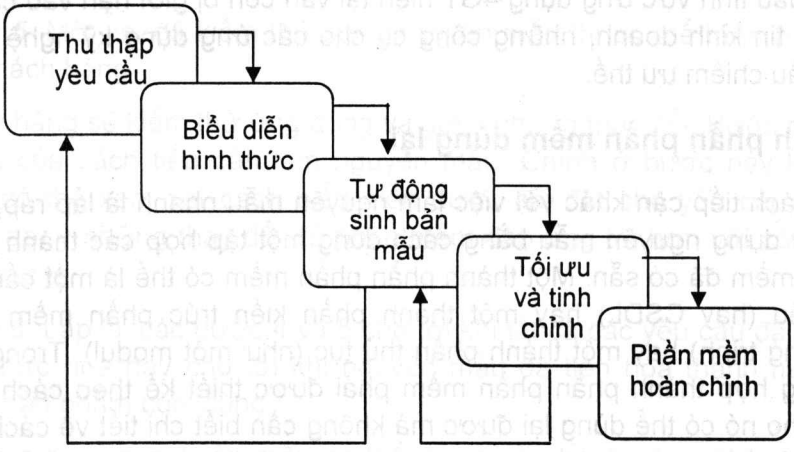
Mặt khác đây cũng là một hình thức của việc dùng lại cho việc làm nguyên mẫu phần mềm.

Đặc tả hình thức và môi trường làm nguyên mẫu

Một số ngôn ngữ và công cụ đặc tả hình thức cũng đã được xây dựng xem như một sự thay thế cho các kỹ thuật đặc tả theo ngôn ngữ tự nhiên. Các ngôn ngữ này đang trong tiến trình phát triển các môi trường tương tác, và"

1. Làm cho người phân tích tạo ra cách tương tác với một đặc tả về hệ thống hay phần mềm dựa trên ngôn ngữ;
2. Có thể gọi tự động các công cụ dịch đặc tả dựa trên ngôn ngữ này thành mã thực hiện.

Làm cho khách hàng có thể dùng nguyên mẫu chương trình theo mã thực hiện được để làm mịn các yêu cầu hình thức. Các ngôn ngữ đặc tả như RAISE, PSL, RSL, IORL, GYPSY, OBJ và nhiều ngôn ngữ khác đang đi kèm với các môi trường tương tác. Mặc dầu vẫn còn đang trong giai đoạn sơ khai của phát triển ứng dụng, những môi trường như vậy phần nào đáp ứng kỳ vọng việc làm nguyên mẫu nâng cao và tính hiệu quả của việc phát triển phần mềm.



H2.6. Một khuôn cảnh kỹ nghệ phần mềm tự động hoá

7. TÓM TẮT

Phân tích và định rõ yêu cầu là kỹ thuật đầu tiên trong tiến trình kỹ nghệ phần mềm. Chính tại điểm này mà phát triển chung về phạm vi phần

mềm được làm mịn thành một bản đặc tả cụ thể để trở thành nền tảng cho mọi hoạt động kỹ nghệ phần mềm theo sau.

Việc phân tích phải tập trung vào các miền thông tin, chức năng và hành vi của vấn đề. Để hiểu rõ hơn cần cái gì, cần tạo ra mô hình, phân hoạch vấn đề và tạo ra những biểu diễn mô tả cho bản chất của yêu cầu rồi sau đó xây dựng các chi tiết cài đặt.

Đặc tả yêu cầu phần mềm được xây dựng như kết quả của việc phân tích. Việc xét duyệt là bản chất để đảm bảo rằng người phát triển và khách hàng có cùng nhận thức về hệ thống.

Trong nhiều trường hợp, không thể nào đặc tả được đầy đủ một vấn đề tại giai đoạn đầu. Việc làm nguyên mẫu thường giúp chỉ ra cách tiếp cận khác để từ đó có thể làm mịn theo yêu cầu. Để tiến hành việc làm nguyên mẫu một cách đúng đắn, có thể cần tới các công cụ và kỹ thuật đặc biệt.

8. CỦNG CỐ

1. Mô tả bằng sơ đồ việc hình thành các yêu cầu trong tiến trình phân tích và định rõ yêu cầu?
2. Phân tích các nguyên lý đặc tả?
3. Khái niệm về đặc tả, biểu diễn và mức trừu tượng của đặc tả?
4. Tại sao chúng ta phải coi việc xác định nhu cầu và phân tích khả thi là những bước đầu tiên của tiến trình phân tích hệ thống?
5. Đặc tả yêu cầu bao gồm những hoạt động nào, hãy mô tả tóm tắt những hoạt động đó?
6. Tại sao nói đặc tả yêu cầu là cơ sở cho hợp đồng làm hệ thống?
7. Đặc tả hệ thống được phân biệt với đặc tả yêu cầu ở điểm nào?
8. Dàn bài đặc tả hệ thống tối thiểu cần bao quát những nội dung nào?
9. Nội dung và ý nghĩa của việc xét duyệt đặc tả hệ thống?
10. Mục tiêu và lợi ích chủ yếu của việc tạo nguyên mẫu?
11. Trình bày sơ lược nội dung của bốn giai đoạn phát triển nguyên mẫu?
12. Sơ lược về các bước/ phương pháp và công cụ làm nguyên mẫu?

Chương 3:

THIẾT KẾ PHẦN MỀM

Thiết kế là bước đầu tiên trong giai đoạn **phát triển** đối với bất kỳ sản phẩm hay hệ thống công nghệ nào. Thiết kế được định nghĩa là tiến trình áp dụng nhiều kỹ thuật và nhiều nguyên lý với mục đích xác định ra 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 thiết kế là để tạo ra một **mô hình** hay **biểu diễn** một thực thể mà sau này sẽ được xây dựng. Thiết kế phải được thực hành và học bằng kinh nghiệm, còn bằng khảo sát các hệ thống đang tồn tại và học bằng sách vở là không đủ. Thiết kế là một quá trình sáng tạo, đòi hỏi kinh nghiệm và sự tinh nhanh của người thiết kế .

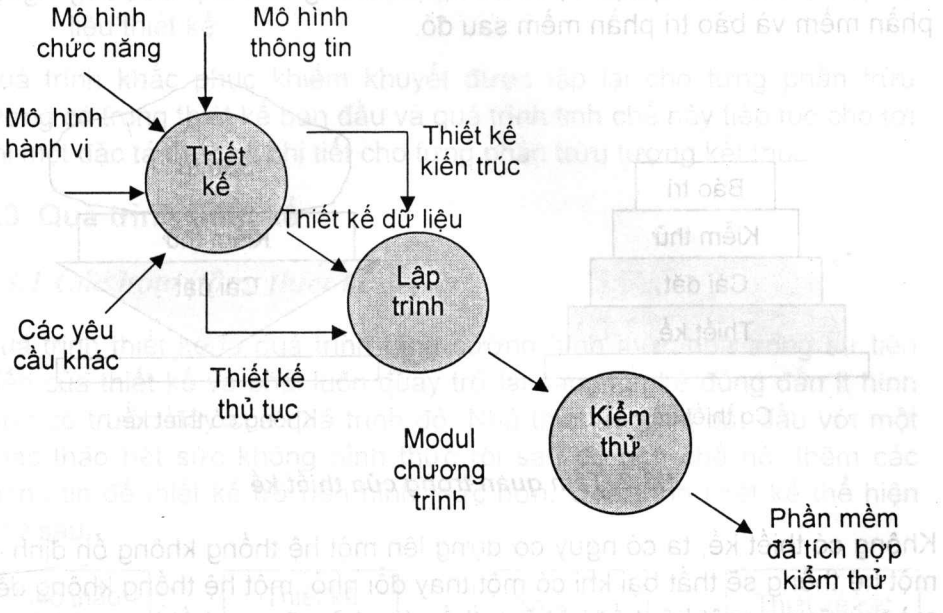
Thiết kế phần mềm là một tiến trình chuyển hóa các yêu cầu thành một biểu diễn phần mềm. Việc làm mịn tiếp sau dẫn tới một biểu diễn thiết kế rất gần với chương trình gốc.

1. THIẾT KẾ PHẦN MỀM

1.1. Thiết kế phần mềm trong kỹ nghệ phần mềm

Thiết kế phần mềm nằm ở trung tâm kỹ thuật của tiến trình kỹ nghệ phần mềm và được áp dụng bất kể tới mọi khuôn cảnh phát triển được sử dụng. Một khi các yêu cầu phần mềm đã được phân tích và đặc tả thì thiết kế phần mềm sẽ là một trong ba hoạt động kỹ thuật: ***thiết kế - lập trình - kiểm thử*** (là những hoạt động cần để xây dựng và kiểm chứng phần mềm). Từng hoạt động này biến đổi thông tin sao cho có thể tạo ra phần mềm máy tính hợp lệ.

Luồng thông tin trong giai đoạn kỹ thuật này của tiến trình kỹ nghệ phần mềm được minh họa trong hình sau:



H3.1. Thiết kế phần mềm và kỹ nghệ phần mềm

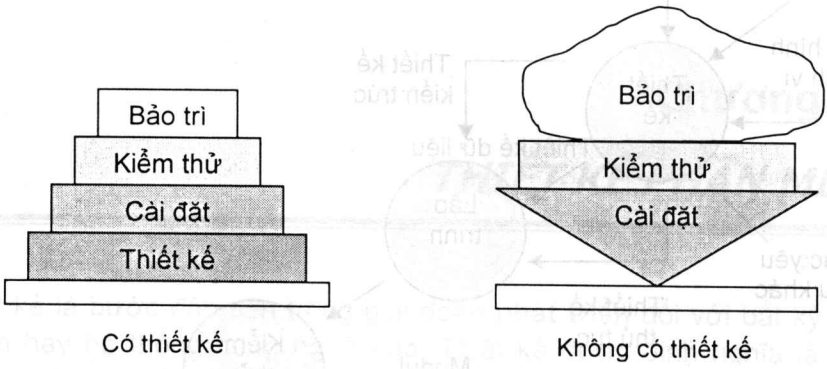
Các yêu cầu phần mềm được biểu thị bằng các mô hình thông tin, chức năng và hành vi, là cái vào cho bước thiết kế. Bằng việc sử dụng một trong số các phương pháp thiết kế, bước thiết kế tạo ra thiết kế dữ liệu, thiết kế kiến trúc và thiết kế thủ tục

- *Thiết kế dữ liệu:* chuyển mô hình lĩnh vực thông tin đã được tạo ra trong bước phân tích thành các cấu trúc dữ liệu hay cơ sở dữ liệu, sẽ cần cho việc cài đặt phần mềm.
- *Thiết kế kiến trúc*: định nghĩa ra 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 được sinh ra rồi việc kiểm thử được thực hiện để tích hợp và làm hợp lệ.

Tầm quan trọng của thiết kế phần mềm có thể phát biểu bằng một từ: *chất lượng*. Thiết kế là nơi chất lượng được nuôi dưỡng trong việc phát triển phần mềm. Thiết kế cung cấp cách biểu diễn phần mềm có thể được xác nhận về chất lượng, là cách duy nhất mà chúng ta có thể chuyển hóa một cách chính xác yêu cầu thành sản phẩm hay hệ thống phần mềm cuối cùng.

Thiết kế phần mềm phục vụ như một nền tảng cho mọi bước kỹ nghệ phần mềm và bảo trì phần mềm sau đó.



H3.2. Tầm quan trọng của thiết kế

Không có thiết kế, ta có nguy cơ dựng lên một hệ thống không ổn định - một hệ thống sẽ thất bại khi có một thay đổi nhỏ, một hệ thống không dễ mà thử được, một hệ thống không thể xác nhận được chất lượng chừng nào chưa đến cuối quá trình kiểm thử, khi thời gian còn rất ngắn và nhiều tiền đã phải chi ra.

Thiết kế tốt là chìa khóa cho công trình hữu hiệu, không thể hình thức hóa quá trình thiết kế trong bất cứ một công trình nào (RAISE chỉ là phương pháp nghiêm ngặt để viết ra thiết kế, kiểm tra thiết kế chứ không phải là phương pháp hình thức để phát triển thiết kế).

1.2. Các giai đoạn trong thiết kế phần mềm

Thiết kế phần mềm trải qua một số giai đoạn sau:

1. Nghiên cứu và hiểu ra vấn đề. Không hiểu rõ vấn đề thì không thể có được thiết kế phần mềm hữu hiệu
2. Xác định các đặc điểm thô của ít nhất một trong vài giải pháp có thể. Chọn giải pháp phụ thuộc vào kinh nghiệm của người thiết kế, vào các cấu kiện dùng lại được và sự đơn giản của giải pháp dẫn xuất. Theo kinh nghiệm, nếu các nhân tố khác là tương tự thì nên chọn giải pháp đơn giản nhất.
3. Mô tả từng điều trừu tượng trong giải pháp. Trước khi tạo ra các tư liệu chính thức người thiết kế nên thấy rằng cần phải xây dựng một mô tả ban đầu sơ khai rồi chi tiết hóa nó. Các sai sót và khiếm khuyết trong thiết kế mức đỉnh được phát hiện (trong

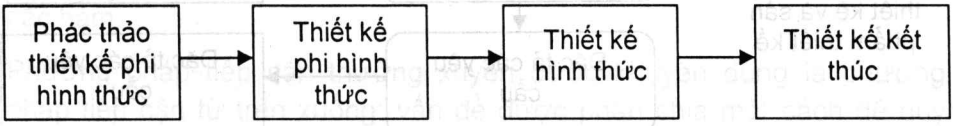
quá trình thiết kế mức đỉnh) phải được chỉnh đốn trước khi lập tư liệu thiết kế.

Quá trình khắc phục khiếm khuyết được lặp lại cho từng phần trừu tượng có trong thiết kế ban đầu và quá trình tinh chế này tiếp tục cho tới khi một đặc tả thiết kế chi tiết cho từng phần trừu tượng kết thúc.

1.3. Quá trình thiết kế

1.3.1 Các hoạt động thiết kế

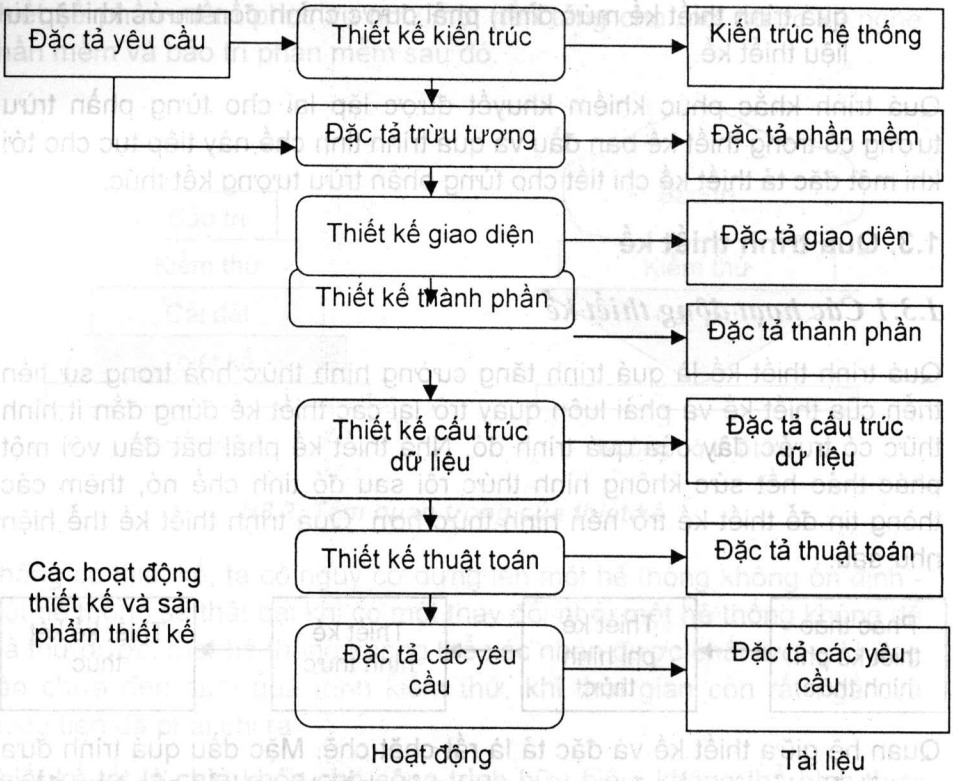
Quá trình thiết kế là quá trình tăng cường hình thức hoá trong sự tiến triển của thiết kế và phải luôn quay trở lại các thiết kế đúng đắn ít hình thức có trước đây của quá trình đó. Nhà thiết kế phải bắt đầu với một phác thảo hết sức không hình thức rồi sau đó tinh chế nó, thêm các thông tin để thiết kế trở nên hình thức hơn. Quá trình thiết kế thể hiện như sau:



Quan hệ giữa thiết kế và đặc tả là rất chặt chẽ. Mặc dầu quá trình đưa ra một đặc tả yêu cầu được xem như một phần tử cơ bản của hợp đồng là một hoạt động riêng biệt song việc hình thức hoá đặc tả yêu cầu hẳn là một phần của quá trình thiết kế. Thực tế, người làm thiết kế sẽ lặp đi lặp lại giữa đặc tả và thiết kế.

Quá trình thiết kế liên quan mật thiết đến việc mô tả hệ thống ở một số mức trừu tượng khác nhau. Khi một thiết kế được phân chia thành nhiều thành phần thì người ta thường phát hiện được các sai sót của các giai đoạn trước, do đó phải quay trở lại các giai đoạn trước để tinh chế lại. Thông thường là người ta bắt đầu giai đoạn này ngay trước khi giai đoạn trước kết thúc, đơn giản là để lui lại quá trình tinh chế.

Hình vẽ dưới đây nêu các hoạt động của quá trình thiết kế và các thành phẩm của nó. Các giai đoạn là khá tuỳ ý, nhưng nó làm cho quá trình thiết kế trở nên nhìn thấy được và do đó quản lý được.



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

Thực tế các hoạt động thiết kế diễn ra song song với các sản phẩm thiết kế khác nhau. Các sản phẩm này đã được triển khai ở các mức chi tiết khác nhau trong diễn biến của quá trình thiết kế,

Các hoạt động cốt yếu trong việc thiết kế một hệ thống phần mềm lớn là:

- *Thiết kế kiến trúc*: các hệ con tạo nên hệ tổng thể và các quan hệ của chúng là được minh định và ghi thành tài liệu;
- *Đặc tả trừu tượng*: đối với mỗi hệ con, một đặc tả trừu tượng các dịch vụ mà nó cung cấp và các ràng buộc tuân theo phải cung cấp;

- *Thiết kế giao diện*: giao diện của từng hệ con với các hệ con khác là được thiết kế và ghi thành tài liệu, đặc tả giao diện không được mơ hồ và cho phép sử dụng hệ con đó mà không cần biết về phép toán hệ con;
- *Thiết kế các thành phần*: các dịch vụ được cung cấp bởi một hệ con là được phân chia qua các thành phần hợp thành của hệ con đó;
- *Thiết kế cấu trúc dữ liệu*: các cấu trúc dữ liệu/CSDL được dùng trong việc thực hiện hệ thống là được thiết kế chi tiết và được đặc tả;
- *Thiết kế thuật toán*: các thuật toán được dùng để cung cấp cho các dịch vụ được thiết kế chi tiết và được đặc tả.

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

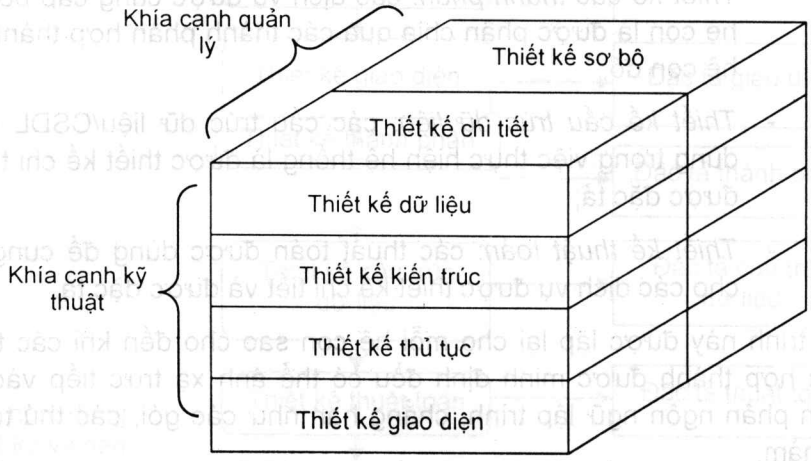
Phương pháp tiếp cận thường xuyên được khuyên dùng là phương pháp tiếp cận từ trên xuống: vấn đề được phân chia một cách đệ quy thành các vấn đề con cho tới khi các vấn đề dễ giải quyết được minh định. Trong quá trình này, người thiết kế sẽ nhận ra các thành phần có thể dùng lại được. Chú ý rằng người thiết kế không nhất thiết phải phân chia tất cả các thành phần trừu tượng khi mà bằng kinh nghiệm họ đã biết rằng thành phần nào là chắc chắn xây dựng được. Do đó họ có thể tập trung sức lực cho phần đáng xét nhất.

Chú ý rằng, khi mà phương pháp hướng đối tượng được chấp nhận thì phương pháp từ trên xuống ít có hiệu quả. Khi đó người thiết kế sử dụng các đối tượng sẵn có để làm khung cho việc thiết kế.

Theo quan điểm quản lý dự án, thiết kế phần mềm được tiến hành theo hai bước:

1. Thiết kế sơ bộ: quan tâm tới việc chuyển hóa các yêu cầu thành kiến trúc dữ liệu cho phần mềm;
2. Thiết kế chi tiết: tập trung vào việc làm mịn biểu diễn kiến trúc dữ liệu 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, có xuất hiện một số hoạt động thiết kế khác nhau. Bên cạnh việc thiết kế dữ liệu, kiến trúc và thủ tục, nhiều ứng dụng hiện đại có hoạt động thiết kế giao diện. Thiết kế giao diện lập ra cách bố trí và cơ chế tương tác người máy.



Mối quan hệ giữa các kỹ thuật và quản lý của thiết kế được minh hoạ trong hình vẽ trên.

1.3.2. Việc mô tả thiết kế

a. Thiết kế phần mềm tạo ra một mô hình của thế giới thực mô tả các thực thể và các mối quan hệ của chúng với nhau.

b. Thiết kế cần mô tả sao cho đạt mức:

- Làm cơ sở cho thực hiện chi tiết;
- Làm phương tiện liên lạc giữa các nhóm thiết kế các hệ con;
- Cung cấp đủ thông tin cho những người bảo trì hệ thống.

c. Người ta thường dùng các khái niệm đồ hoạ, các ngôn ngữ mô tả chương trình, văn bản không hình thức để tạo dựng các tài liệu thiết kế.

1.4. PHƯƠNG PHÁP THIẾT KẾ

1.4.1. Phương pháp thiết kế

Trong nhiều tổ chức, việc thiết kế phần mềm vẫn còn là quá trình tự học, với tập hợp các yêu cầu (thường được mô tả bằng ngôn ngữ tự

nhiên) người ta xây dựng một bản thiết kế không hình thức. Sau đó bắt tay ngay vào việc mã hoá và bản thiết kế được cải biên trong khi hệ thống được thực hiện. Khi giai đoạn thực hiện kết thúc thì so với đặc tả ban đầu, sản phẩm đã thay đổi hoàn toàn.

Một cách tiếp cận có phương pháp hơn là: "phương pháp cấu trúc". Đó là các phương pháp làm mịn kiến trúc phần mềm theo kiến trúc từ trên xuống. Các khía cạnh thủ tục của định nghĩa thiết kế đã tiến hoá thành một triết lý là lập trình có cấu trúc.

Phương pháp cấu trúc được dùng rộng rãi trong những năm đầu của thập kỷ 80. Nó đã được dùng thành công trong nhiều dự án lớn, làm giảm giá thành đáng kể, sử dụng được các khái niệm chuẩn và đảm bảo rằng các thiết kế tuân theo một chuẩn. Các công cụ CASE đã được dùng để trợ giúp cho phương pháp này.

Trong những năm sau này, phương pháp hướng đối tượng được đề cập tới, được phát triển và được dùng phổ biến hiện nay.

Các phương pháp thiết kế thường trợ giúp một vài cách nhìn nhận hệ thống như sau:

- Nhìn nhận cấu trúc: cho cái nhìn cấu trúc thông qua Biểu đồ cấu trúc;
- Nhìn nhận quan hệ thực thể: mô tả các cấu trúc dữ liệu logic được dùng, nói đến đặc tả dữ liệu, mô hình quan hệ thực thể;
- Nhìn nhận dòng dữ liệu: thông qua Biểu đồ dòng dữ liệu.

Người ta còn dùng lược đồ chuyển trạng thái để bổ sung cho các phương pháp trên

Để đảm bảo chất lượng cao cho một biểu diễn thiết kế, cần có các tiêu chuẩn cho thiết kế tốt. Song về mặt phương pháp, chúng ta đưa ra các hướng dẫn sau:

1. Thiết kế nêu ra cách tổ chức theo cấp bậc để dùng cách kiểm soát thông minh;
2. Thiết kế nên theo các modul, tức là phần mềm nên được phân hoạch một cách logic thành các thành phần thực hiện những chức năng xác định;
3. Thiết kế nên chứa các biểu diễn phân biệt và tách biệt giữa dữ liệu và thủ tục;

4. Thiết kế nên dẫn tới (như chương trình con hay thủ tục) nêu ra các đặc trưng chức năng đặc biệt ;

5. Thiết kế nên dẫn tới giao diện giúp rút gọn độ phức tạp của việc nối ghép giữa các modul và với môi trường bên ngoài;

6. Thiết kế nên được hướng theo phương pháp lặp lại, được điều khiển bởi các dữ liệu có trong bản phân tích các yêu cầu phần mềm.

Các đặc trưng trên của một thiết kế tốt có được khi thực hiện đúng tiến trình thiết kế kỹ nghệ phần mềm thông qua việc áp dụng các nguyên lý thiết kế cơ bản, phương pháp luận hệ thống và việc xét duyệt thấu đáo.

Như vậy mỗi phương pháp thiết kế phần mềm đều đưa vào những cách biểu diễn trực cảm, ký pháp thống nhất cũng như một cách nhìn đặc trưng

Tuy vậy mỗi phương pháp đều có các đặc trưng sau:

1. Một cơ chế để chuyển hóa từ biểu diễn miền thông tin thành biểu diễn thiết kế;

2. Một bộ ký pháp để biểu diễn các thành phần chức năng và giao diện của chúng;

3. Cách trực cảm để làm mịn phân hoạch;

4. Các hướng dẫn về định giá chất lượng.

Bất kể phương pháp luận thiết kế nào được dùng, công trình sư phần mềm phải áp dụng một tập các khái niệm nền tảng cho thiết kế dữ liệu, kiến trúc và thủ tục bao gồm: **trừu tượng, làm mịn, modul, kiến trúc phần mềm, cấp bậc điều khiển, cấu trúc dữ liệu, thủ tục phần mềm, che dấu thông tin.**

Tập hợp các khái niệm thiết kế nền tảng đã được áp dụng hơn ba thập kỷ. Mỗi khái niệm đều cung cấp cho người thiết kế phần mềm một nền tảng để từ đó người ta có thể áp dụng nhiều phương pháp thiết kế phức tạp.

M.A.JACKSON nói: "điểm bắt đầu của một kỹ sư phần mềm khôn ngoan là thừa nhận sự khác biệt giữa việc bắt tay vào làm chương trình hay trước khi bắt tay vào làm phải hiểu vấn đề một cách đúng đắn". Các khái niệm thiết kế phần mềm nền tảng cung cấp một khuôn khổ cần thiết để hiểu vấn đề một cách đúng đắn.

1.4.2. Các khái niệm nền tảng cho thiết kế

1.4.2.1. Trừu tượng

Có nhiều mức trừu tượng

- Mức cao nhất: một giải pháp được phát biểu theo *thuật ngữ đại thể* bằng cách dùng ngôn ngữ của môi trường vấn đề;
- Mức vừa: lấy khuynh hướng thủ tục nhiều hơn. *Thuật ngữ hướng vấn đề* thường đi đôi với thuật ngữ hướng cài đặt trong mô tả giải pháp;
- Mức thấp: giải pháp được phát biểu theo *thuật ngữ chi tiết* để có thể cài đặt trực tiếp

Mỗi bước trong tiến trình kỹ nghệ phần mềm đều là sự làm mịn cho một mức trừu tượng của phần mềm. Trong kỹ nghệ hệ thống, phần mềm được dùng như một phần tử của hệ thống dựa trên máy tính. Trong phân tích các yêu cầu phần mềm, giải pháp phần mềm được phát biểu dưới dạng "đó là cái quan trọng trong môi trường vấn đề". Khi chúng ta chuyển từ thiết kế sơ bộ sang thiết kế chi tiết thì mức độ trừu tượng được rút lại. Cuối cùng, ta đi tới mức trừu tượng thấp nhất khi sinh ra chương trình gốc.

Có nhiều dạng trừu tượng:

Khi chúng ta chuyển sang mức trừu tượng khác nhau, chúng ta làm việc để tạo ra các trừu tượng thủ tục và dữ liệu. *Trừu tượng thủ tục* là một dãy các lệnh có tên, có một chức năng xác định và có giới hạn. Một ví dụ về thủ tục trừu tượng là từ "đi vào" cửa. "Đi vào" kéo theo một dãy dài các bước thủ tục (như bước tới cửa, lại gần và nắm lấy quả đấm, xoay quả đấm cửa và kéo cửa ra, bước vào...). *Trừu tượng dữ liệu* là một tập hợp các dữ liệu có tên mô tả cho sự vật dữ liệu. Ví dụ về dữ liệu trừu tượng là "séc thanh toán". Đối tượng dữ liệu này thực chất là một tập hợp nhiều mẩu thông tin khác nhau (như tên người thanh toán, số tiền thanh toán, tiền thuế...). Vậy chúng ta có thể tham khảo mọi dữ liệu bằng cách nói tên của trừu tượng dữ liệu.

- **Trừu tượng thủ tục**

Tại mức trừu tượng này đã có việc biểu diễn thủ tục sơ bộ. Thuật ngữ này đã hướng phần mềm (như việc dùng các cấu trúc *do while*) và việc dính líu tới modul bắt đầu nổi lên bề mặt.

Khái niệm về làm mịn dữ liệu từng bước và modul gắn liền với việc trừu tượng. Khi thiết kế phần mềm tiến hóa, từng mức modul trong cấu trúc chương trình sẽ biểu diễn cho việc làm mịn dần trong mức trừu tượng của phần mềm.

- **Trừu tượng dữ liệu**

Giống như trừu tượng thủ tục, trừu tượng dữ liệu làm cho người thiết kế có thể biểu diễn một sự vật dữ liệu ở các mức chi tiết khác nhau, nhưng điều quan trọng hơn, là xác định một sự vật dữ liệu trong hoàn cảnh các thao tác (thủ tục) có thể được áp dụng vào nó.

Một số các ngôn ngữ lập trình (như ADA, MODULA, CLU) đưa ra cơ chế để tạo kiểu dữ liệu trừu tượng. Chẳng hạn, *package* của ADA là một cơ chế ngôn ngữ lập trình đưa ra sự hỗ trợ cho cả trừu tượng dữ liệu và thủ tục. Kiểu trừu tượng dữ liệu nguyên gốc được dùng như một tiêu bản hay cấu trúc dữ liệu sinh ra để từ đó có thể làm *thể nghiệm* cho các cấu trúc dữ liệu khác.

- **Trừu tượng điều khiển**

Trừu tượng điều khiển là dạng thứ ba của trừu tượng hóa được dùng trong thiết kế phần mềm. Giống như trừu tượng dữ liệu và thủ tục, trừu tượng điều khiển áp dụng cho cơ chế điều khiển chương trình mà không xác định các chi tiết bên trong. Một ví dụ về điều khiển là cơ chế đồng bộ hóa được dùng để điều hòa các hoạt động trong hệ điều hành.

1.4.2.2. Làm mịn

Làm mịn từng bước là một *chiến lược thiết kế trên - xuống* ban đầu do NIKLAUS WIRTH đề nghị. Kiến trúc của một chương trình được phát triển bằng các mức làm mịn liên tiếp các chi tiết thủ tục. Một cấp bậc được xây dựng nên bằng cách phân tách một phát biểu vĩ mô về chức năng (trừu tượng thủ tục) theo kiểu từng bước cho tới khi đạt đến phát biểu bằng ngôn ngữ lập trình.

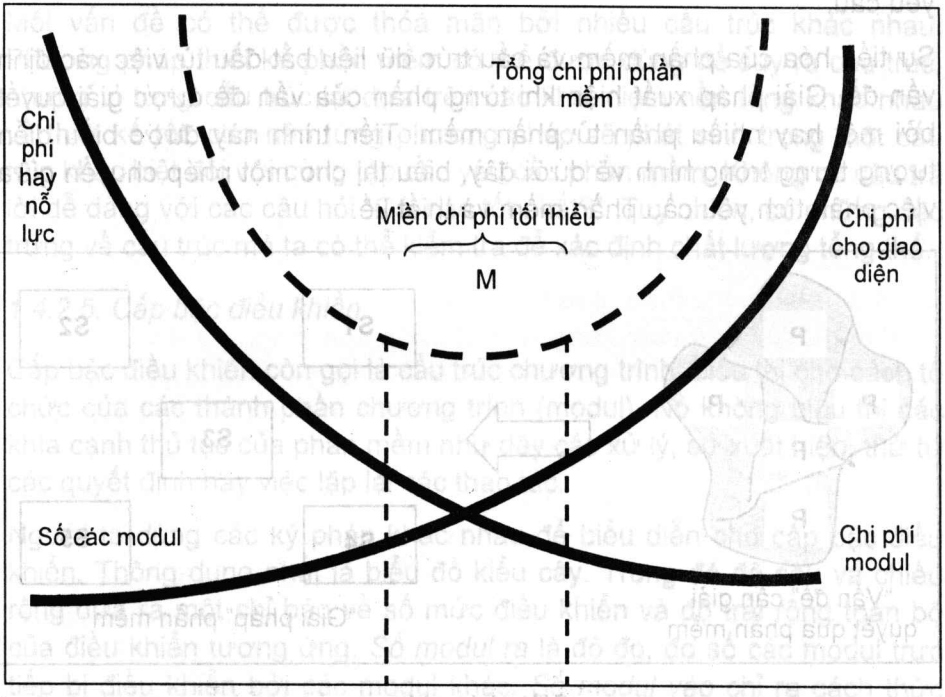
Làm mịn trước tiên là một tiến trình khởi thảo. Bắt đầu với một phát biểu về chức năng (hay mô tả thông tin) được xác định như mức trừu tượng cao. Tức là, phát biểu mô tả chức năng hay thông tin về mặt quan niệm, nhưng không đưa ra thông tin về cách làm việc nội bộ của chức năng hay cấu trúc nội bộ của thông tin đó. Việc làm mịn buộc người thiết kế phải khởi thảo phát biểu nguyên gốc, sau đó đưa ra ngày càng nhiều chi tiết khi việc làm mịn khởi thảo kế tiếp xuất hiện.

1.4.2.3. Tính modul

Phần mềm được chia thành các thành phần có tên riêng biệt và định địa chỉ được, gọi là các *modul*, sau đó chúng được tích hợp lại để thỏa mãn yêu cầu.

Người ta nói rằng, tính modul là thuộc tính riêng của phần mềm cho phép một chương trình nên quản lý được theo cách thông minh. Người đọc không thể nào hiểu thấu phần mềm nguyên khối (như một chương trình lớn chỉ gồm một modul). Điều này dẫn đến kết luận chia để trị sẽ dễ giải quyết một vấn đề phức tạp hơn khi chia nó thành những phần quản lý được.

Nỗ lực (chi phí) để phát triển một modul phần mềm riêng lẻ không giảm đi khi tổng số các modul tăng lên. Với cùng tập hợp các yêu cầu, nhiều modul hơn tức là kích cỡ từng modul sẽ nhỏ hơn. Tuy nhiên khi số các modul tăng lên thì nỗ lực liên kết với việc làm giao diện cho các modul cũng tăng lên. Hình vẽ mô tả đặc trưng này dẫn đến đường cong tổng phí:



Có M modul sẽ gây ra chi phí phát triển tối thiểu, nhưng không có độ phức tạp cần thiết để dự kiến M với sự đảm bảo.

Đường cong được vẽ đưa ra thông tin có ích khi phải xét tính tới modul. Chúng ta nên modul hóa nhưng cũng phải chú ý duy trì trong vùng lân cận của M. Modul hóa còn chưa đủ hay qua mức đều nên tránh.

Việc tìm vùng lân cận của M, chia modul phần mềm đến đâu là câu hỏi cần trả lời vì kích cỡ của modul bị khống chế bởi chức năng của nó và ứng dụng.

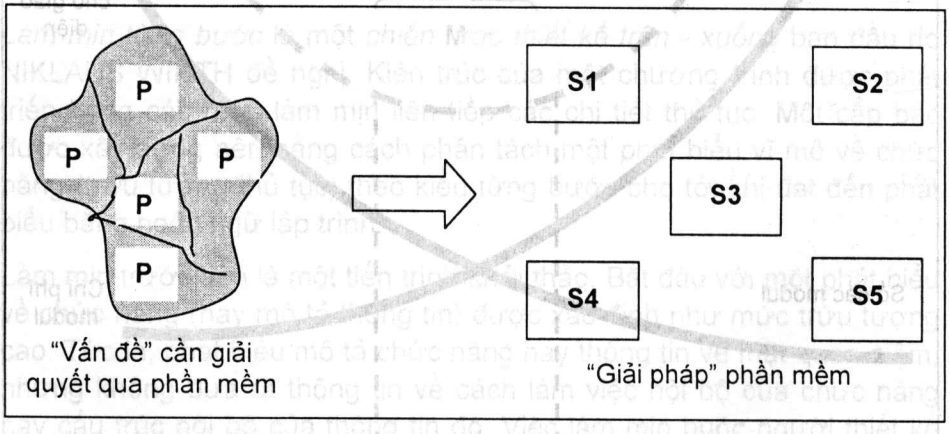
1.4.2.4. Kiến trúc phần mềm

Kiến trúc phần mềm gồm hai đặc trưng quan trọng của chương trình máy tính:

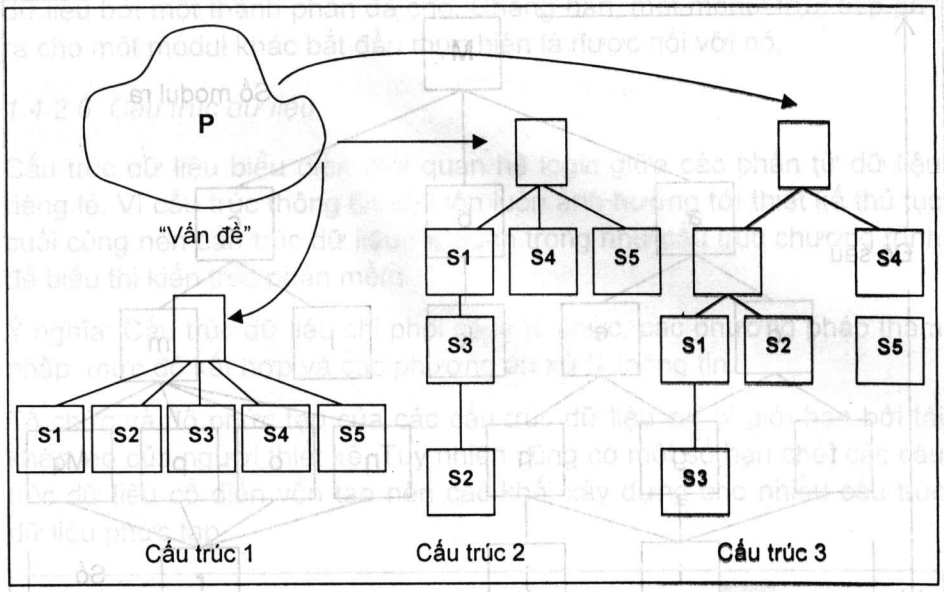
1. Cấu trúc cấp bậc điều khiển các thành phần thủ tục (modul, hoặc cấu trúc chương trình);
2. Cấu trúc dữ liệu.

Kiến trúc phần mềm được suy dẫn ra qua tiến trình phân hoạch và đặt mối quan hệ giữa các phần tử của giải pháp phần mềm với các bộ phận của thế giới thực, được xác định không tường minh trong bản phân tích yêu cầu.

Sự tiến hóa của phần mềm và cấu trúc dữ liệu bắt đầu từ việc xác định vấn đề. Giải pháp xuất hiện khi từng phần của vấn đề được giải quyết bởi một hay nhiều phần tử phần mềm. Tiến trình này được biểu diễn tượng trưng trong hình vẽ dưới đây, biểu thị cho một phép chuyển giữa việc phân tích yêu cầu phần mềm và thiết kế.



Giải pháp rời rạc



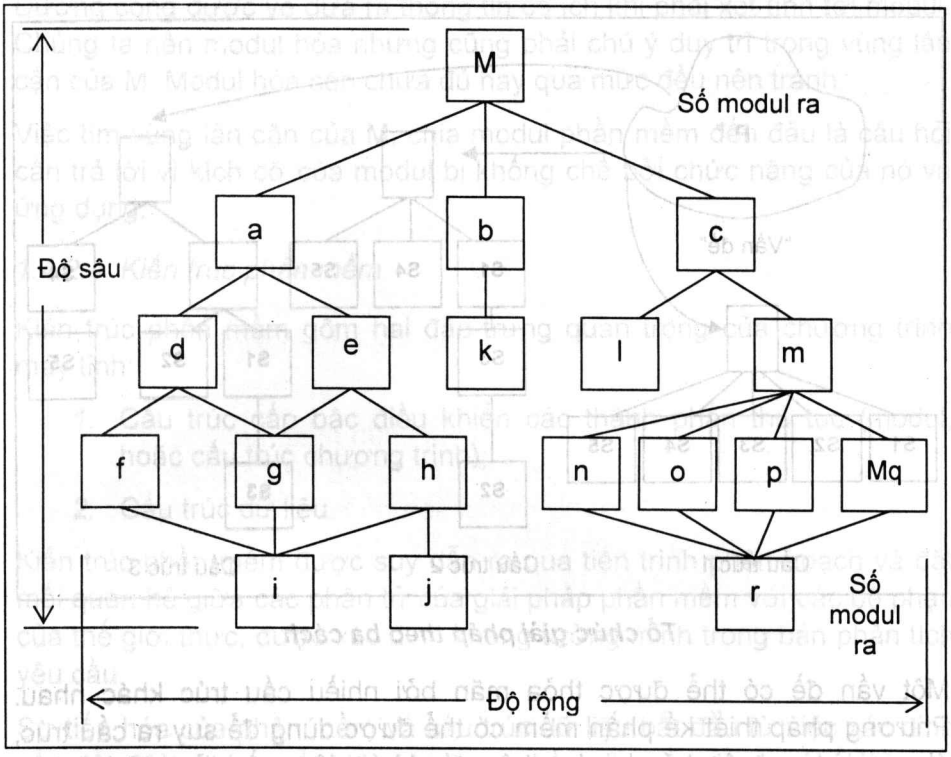
Tổ chức giải pháp theo ba cách

Một vấn đề có thể được thỏa mãn bởi nhiều cấu trúc khác nhau. Phương pháp thiết kế phần mềm có thể được dùng để suy ra cấu trúc, nhưng vì từng cấu trúc lại dựa trên các khái niệm nền tảng khác nhau về thiết kế tốt, cho nên từng phương pháp sẽ phát sinh trong một cấu trúc khác biệt đối với cùng tập các yêu cầu phần mềm. Không có câu trả lời dễ dàng với các câu hỏi "Cái nào tốt nhất?". Tuy nhiên, có những đặc trưng về cấu trúc mà ta có thể kiểm tra để xác định chất lượng tổng thể.

1.4.2.5. Cấp bậc điều khiển

Cấp bậc điều khiển còn gọi là cấu trúc chương trình, biểu thị cho cách tổ chức của các thành phần chương trình (modul). Nó không biểu thị các khía cạnh thủ tục của phần mềm như dãy các xử lý, sự xuất hiện, thứ tự các quyết định hay việc lặp lại các thao tác.

Người ta dùng các ký pháp khác nhau để biểu diễn cho cấp bậc điều khiển. Thông dụng nhất là biểu đồ kiểu cây. Trong đó độ sâu và chiều rộng đưa ra một chỉ báo về số mức điều khiển và độ trải rộng toàn bộ của điều khiển tương ứng. *Số modul ra* là độ đo, đo số các modul trực tiếp bị điều khiển bởi các modul khác. *Số modul vào* chỉ ra cách thức modul trực tiếp điều khiển một modul đã cho.



Mối quan hệ điều khiển giữa các modul được diễn tả theo cách sau: một modul điều khiển một modul khác thì được gọi là thượng cấp của nó, và ngược lại modul bị một modul khác điều khiển thì được gọi là thuộc cấp của modul điều khiển. Chẳng hạn: modul M là thượng cấp của các modul a, b và c; modul d, e là thuộc cấp của modul a; modul h là thuộc cấp của modul e và cuối cùng là thuộc cấp của modul M. Mối quan hệ theo chiều rộng (tức là mối quan hệ giữa các modul d và e), mặc dầu có thể diễn tả trong thực tế nhưng không nhất thiết phải được xác định bằng thuật ngữ tường minh (xem hình trên).

Cấp bậc điều khiển cũng biểu diễn cho hai đặc trưng khác nhau của cấu trúc phần mềm: tính *thấy được* và tính *nối được*. Tính thấy được chỉ ra tập hợp các thành phần chương trình có thể được gọi hay được dùng như dữ liệu bởi một thành phần đã cho, ngay cả khi điều này được thực hiện gián tiếp. Chẳng hạn, một modul trong hệ thống hướng đối tượng có thể thâm nhập vào một mảng rộng các sự vật dữ liệu mà nó đã kế thừa, nhưng chỉ dùng một số nhỏ các các sự vật dữ liệu đó. Tính nối được chỉ tập các thành phần trực tiếp được gọi hay được sử dụng như

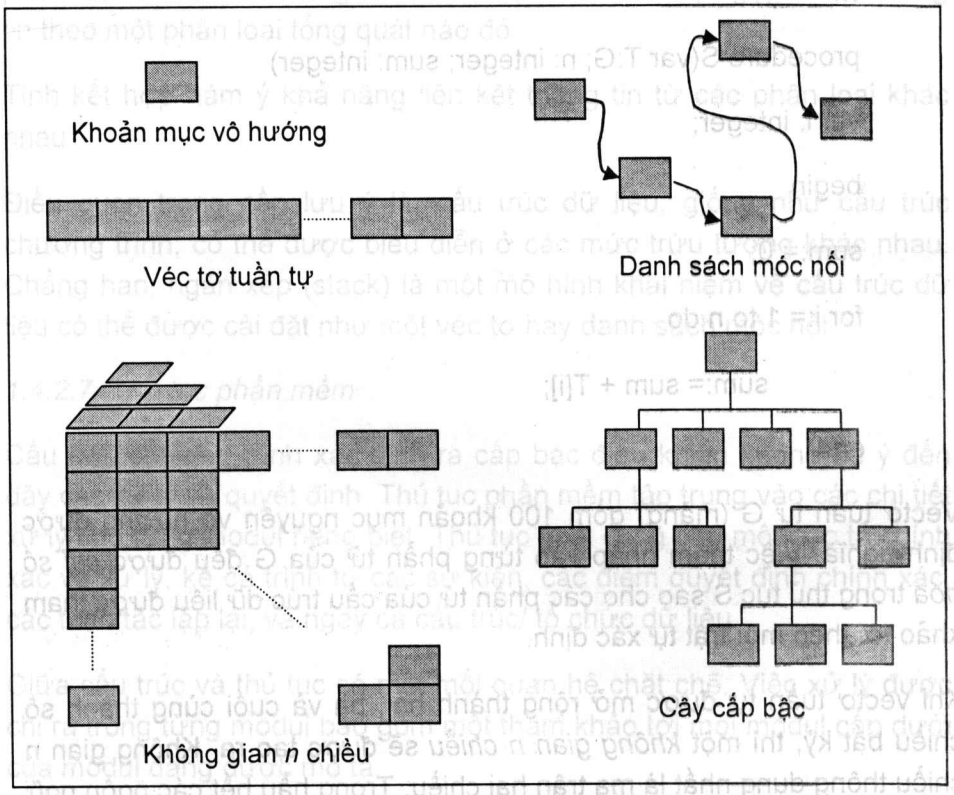
dữ liệu bởi một thành phần đã cho. Chẳng hạn, một modul trực tiếp gây ra cho một modul khác bắt đầu thực hiện là được nối với nó.

1.4.2.6. Cấu trúc dữ liệu

Cấu trúc dữ liệu biểu diễn mối quan hệ logic giữa các phần tử dữ liệu riêng lẻ. Vì cấu trúc thông tin sẽ luôn luôn ảnh hưởng tới thiết kế thủ tục cuối cùng nên cấu trúc dữ liệu rất quan trọng như cấu trúc chương trình để biểu thị kiến trúc phần mềm.

Ý nghĩa: Cấu trúc dữ liệu chi phối cách tổ chức, các phương pháp thâm nhập, mức độ kết hợp và các phương án xử lý thông tin.

Tổ chức và độ phức tạp của các cấu trúc dữ liệu chỉ bị giới hạn bởi tài khéo léo của người thiết kế. Tuy nhiên cũng có một số hạn chế, các cấu trúc dữ liệu cổ điển vốn tạo nên các khối xây dựng cho nhiều cấu trúc dữ liệu phức tạp.



Khoản mục vô hướng: là cấu trúc dữ liệu đơn giản nhất trong các cấu trúc dữ liệu. Như tên của nó hàm ý, khoản mục vô hướng biểu thị cho một phần tử thông tin đơn giản có thể được đánh địa chỉ bằng một tên

gọi; tức là việc thâm nhập có thể được đạt tới bằng cách xác định chỉ một địa chỉ trong bộ nhớ. Kích cỡ và định dạng của một khoản mục vô hướng có thể thay đổi bên trong các giới hạn do ngôn ngữ lập trình khống chế. Chẳng hạn, một khoản mục vô hướng có thể là thực thể logic dài 1 bit, một số nguyên hay số phảy động dài từ 8 bit đến 64 bit, hay một xâu ký tự một trăm hay một nghìn byte.

Khi các khoản mục vô hướng được tổ chức như một danh sách hay nhóm liên tục thì một *vectơ tuần tự* sẽ được hình thành. Vectơ là phần chung nhất của tất cả các cấu trúc dữ liệu và mở cánh cửa tới việc làm chỉ số thay đổi cho thông tin. Để minh họa cho chúng ta xem xét một thí dụ PASCAL đơn giản:

```
Type G = array [1..100] of integer;

...

procedure S(var T:G; n: integer; sum: integer)

var i: integer;

begin

sum:= 0;

for i:= 1 to n do

        sum:= sum + T[i];

end;
```

Vectơ tuần tự G (mảng) gồm 100 khoản mục nguyên vô hướng được định nghĩa. Việc thâm nhập vào từng phần tử của G đều được *chỉ số hóa* trong thủ tục S sao cho các phần tử của cấu trúc dữ liệu được tham khảo tới theo một trật tự xác định.

Khi vectơ tuần tự được mở rộng thành hai, ba và cuối cùng thành số chiều bất kỳ, thì một *không gian n chiều* sẽ được tạo ra. Không gian n chiều thông dụng nhất là ma trận hai chiều. Trong hầu hết các ngôn ngữ lập trình, một không gian n chiều được gọi là một mảng.

Khoản mục véc tơ và không gian có thể được tổ chức theo nhiều định dạng. Danh sách móc nối là một cấu trúc dữ liệu tổ chức các khoản mục vô hướng, véc tơ hay không gian liên tục theo một cách (còn gọi là nút, đỉnh) làm cho chúng được xử lý như một danh sách. Mỗi đỉnh chứa cách tổ chức dữ liệu thích hợp (như véc tơ) và một hay nhiều con trỏ chỉ ra địa chỉ trong bộ nhớ của đỉnh tiếp trong danh sách. Có thể bổ sung thêm các đỉnh tại bất kỳ điểm nào trong danh sách bằng các định nghĩa lại các con trỏ để thích hợp với lối vào danh sách mới.

Các cấu trúc dữ liệu khác tổ hợp hay được xây dựng bằng cách dùng các cấu trúc dữ liệu nền tảng được mô tả ở trên. Chẳng hạn, cấu trúc dữ liệu cấp bậc được điều khiển bằng cách sử dụng danh sách đa móc nối có chứa các khoản mục vô hướng, véc tơ và có thể cả không gian n chiều. Cấu trúc cấp bậc thường hay gặp trong các ứng dụng có đòi hỏi phân loại và liên kết thông tin. Phân loại bao hàm việc gộp nhóm thông tin theo một phân loại tổng quát nào đó.

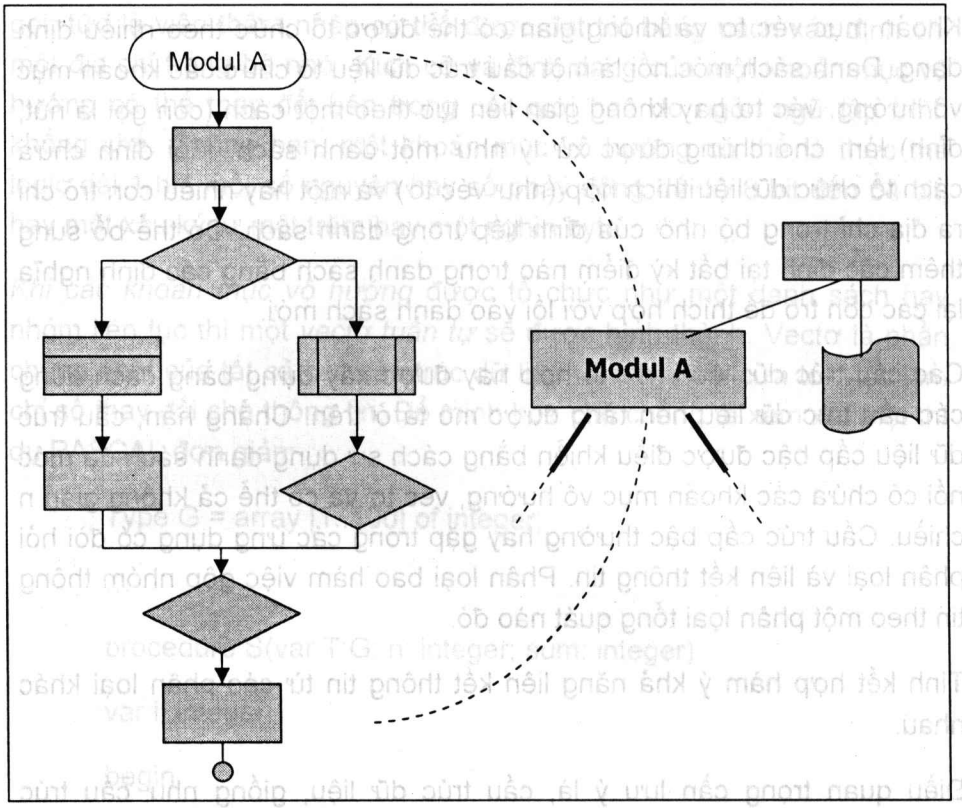
Tính kết hợp hàm ý khả năng liên kết thông tin từ các phân loại khác nhau.

Điều quan trọng cần lưu ý là, cấu trúc dữ liệu, giống như cấu trúc chương trình, có thể được biểu diễn ở các mức trừu tượng khác nhau. Chẳng hạn, ngăn xếp (stack) là một mô hình khái niệm về cấu trúc dữ liệu có thể được cài đặt như một véc tơ hay danh sách móc nối.

1.4.2.7. Thủ tục phần mềm

Cấu trúc chương trình xác định ra cấp bậc điều khiển không để ý đến dãy các xử lý và quyết định. Thủ tục phần mềm tập trung vào các chi tiết xử lý cho từng modul riêng biệt. Thủ tục phải cung cấp một đặc tả chính xác về xử lý, kể cả trình tự các sự kiện, các điểm quyết định chính xác, các thao tác lặp lại, và ngay cả cấu trúc/ tổ chức dữ liệu.

Giữa cấu trúc và thủ tục có một mối quan hệ chặt chẽ. Việc xử lý được chỉ ra trong từng modul bao gồm một tham khảo tới mọi modul cấp dưới của modul đang được mô tả.



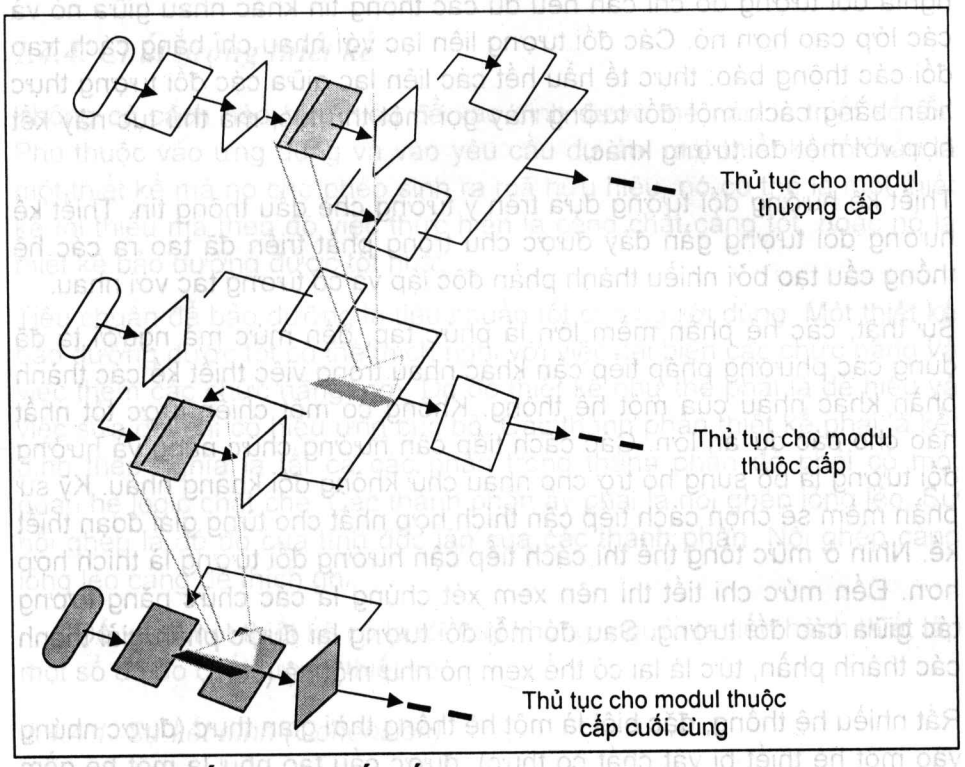
1.4.2.8. Che dấu thông tin

Khái niệm về modul đưa người thiết kế phần mềm tới một câu hỏi nền tảng: làm sao ta phân rã một giải pháp phần mềm ra để thu được tập các modul tốt nhất. Nguyên lý về che dấu thông tin gợi ý rằng các modul nên được đặc trưng bởi những quyết định thiết kế mà ẩn kín với mọi modul khác. Nói cách khác, modul nên được đặc tả và thiết kế sao cho thông tin được chứa trong một modul này không thể thâm nhập tới được từ các modul khác vốn không cần tới những thông tin đó.

Việc che dấu kéo theo rằng, người ta có thể đạt được tính modul bằng cách xác định một tập các modul độc lập mà trao đổi giữa modul nọ với modul kia chỉ là những thông tin cần thiết cho sự vận hành của phần mềm. Việc trừu tượng hóa giúp cho việc xác định các thực thể thủ tục có chứa phần mềm. Việc che dấu xác định và áp đặt các ràng buộc thâm nhập tới cả chi tiết thủ tục bên trong modul đó và bất kỳ cấu trúc dữ liệu cục bộ nào mà modul đó sử dụng.

Việc che dấu thông tin được coi là một tiêu chuẩn thiết kế đối với hệ thống modul, đưa ra những ích lợi lớn nhất khi cần có những thay đổi

trong việc kiểm thử và sau này trong bảo trì phần mềm. Vì phần lớn dữ liệu và thủ tục đều bị che dấu khỏi các bộ phận khác của phần mềm, nên những sai sót bị đưa vào trong khi thay đổi sẽ ít có khả năng lan truyền sang các vị trí khác bên trong phần mềm.



1.4.3 Các chiến lược thiết kế

Xét các chiến lược hay được nhắc đến: thiết kế chức năng, thiết kế hướng đối tượng, thiết kế hệ thống tương tranh.

1.4.3.1. Thiết kế chức năng

Hệ thống được thiết kế theo quan điểm chức năng, bắt đầu ở mức cao nhất, sau đó tinh chế dần dần để thành thiết kế chi tiết hơn. Trạng thái của hệ thống là tập trung và được chia sẻ cho các chức năng thao tác trên trạng thái đó.

1.4.3.2. Thiết kế hướng đối tượng

Hệ thống được nhìn nhận như một bộ các đối tượng (chứ không phải là một bộ chức năng). Hệ thống được phân tán, mỗi đối tượng có những thông tin trạng thái riêng của nó. Đối tượng là bộ các 'thuộc tính' xác

định trạng thái của đối tượng đó và các phép toán thực hiện trên các thuộc tính đó. Mỗi đối tượng là một khách thể của một lớp mà *lớp được xác định bởi các thuộc tính và các phương thức của nó*. Đối tượng có thể được thừa kế từ một vài lớp đối tượng lớp cao hơn, sao cho định nghĩa đối tượng đó chỉ cần nêu đủ các thông tin khác nhau giữa nó và các lớp cao hơn nó. Các đối tượng liên lạc với nhau chỉ bằng cách trao đổi các thông báo: thực tế hầu hết các liên lạc giữa các đối tượng thực hiện bằng cách một đối tượng này gọi một thủ tục, mà thủ tục này kết hợp với một đối tượng khác.

Thiết kế hướng đối tượng dựa trên ý tưởng che dấu thông tin. Thiết kế hướng đối tượng gần đây được chú trọng phát triển đã tạo ra các hệ thống cấu tạo bởi nhiều thành phần độc lập và có tương tác với nhau.

Sự thật, các hệ phần mềm lớn là phức tạp, đến mức mà người ta đã dùng các phương pháp tiếp cận khác nhau trong việc thiết kế các thành phần khác nhau của một hệ thống. Không có một chiến lược tốt nhất nào cho các dự án lớn. Các cách tiếp cận hướng chức năng và hướng đối tượng là bổ sung hỗ trợ cho nhau chứ không đối kháng nhau. Kỹ sư phần mềm sẽ chọn cách tiếp cận thích hợp nhất cho từng giai đoạn thiết kế. Nhìn ở mức tổng thể thì cách tiếp cận hướng đối tượng là thích hợp hơn. Đến mức chi tiết thì nên xem xét chúng là các chức năng tương tác giữa các đối tượng. Sau đó mỗi đối tượng lại được phân giải thành các thành phần, tức là lại có thể xem nó như một hệ (con).

Rất nhiều hệ thống, đặc biệt là một hệ thống thời gian thực (được nhúng vào một hệ thiết bị vật chất có thực), được cấu tạo như là một hệ gồm một bộ các quá trình song song có liên lạc với nhau. Các hệ này thường phải tuân theo các ràng buộc nghiêm ngặt về thời gian, mà các phần cứng thường hoạt động tương đối chậm, chỉ có cách tiếp cận nhiều bộ xử lý hoạt động song song mới có thể hoàn thành được yêu cầu về thời gian.

Các chương trình tuần tự là dễ thiết kế, thực hiện, kiểm tra và thử nghiệm hơn so với các hệ song song. Sự phụ thuộc thời gian giữa các quá trình là khó hình thức hóa, khó khống chế và thử nghiệm.

Do đó quá trình thiết kế nên được xem như một hoạt động gồm hai giai đoạn:

1. Minh định cấu trúc thiết kế logic, cụ thể là các thành phần của hệ thống và các mối quan hệ giữa chúng. Có thể dùng cách nhìn chức năng hoặc cách nhìn đối tượng.

2. Thực hiện cấu trúc đó trong dạng có thể thực hiện được. Giai đoạn này đôi khi được gọi là thiết kế chi tiết và đôi khi là lập trình. Sự quyết định về tính song song nên làm ở giai đoạn này chứ không phải là các giai đoạn sớm hơn trong quá trình thiết kế.

1.4.4. Chất lượng thiết kế

Không có cách nào hữu hiệu để xác định được thế nào là thiết kế tốt. Phụ thuộc vào ứng dụng và vào yêu cầu dự án, một thiết kế tốt hẳn là một thiết kế mà nó cho phép sinh ra mã hữu hiệu, nó có thể là một thiết kế tối thiểu mà theo đó việc thực hiện là càng chặt càng tốt, hoặc nó là thiết kế bảo dưỡng được tốt nhất.

Tiêu chuẩn để bảo dưỡng là tiêu chuẩn tốt cho người dùng. Một thiết kế bảo dưỡng được tốt có thể thích nghi với việc cải biên các chức năng và việc thêm các chức năng mới. Do đó thiết kế như thế phải là dễ hiểu và việc sửa đổi chỉ có hiệu ứng cục bộ. Các thành phần thiết kế phải là kết dính theo nghĩa là tất cả các phần trong thành phần đó phải có một quan hệ logic chặt chẽ. Các thành phần ấy phải là nối ghép lỏng lẻo. Sự nối ghép là độ đo của tính độc lập của các thành phần. Nối ghép càng lỏng lẻo càng dễ thích ghi.

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

1.4.4.1. Sự kết dính (Cohension)

Sự kết dính của một thành phần là độ đo về tính khớp lại với nhau giữa một thành phần hoặc việc thực hiện một chức năng logic hoặc việc thực hiện một thực thể logic. Tất cả các phần của thành phần đó đều tham gia vào việc thực hiện. Nếu một thành phần không tham gia trực tiếp vào chức năng logic đó thì mức độ kết dính của nó là thấp.

Constantin và Yourdon định ra bảy mức kết dính theo thứ tự tăng dần sau đây:

1. Kết dính gom góp: các phần của thành phần không liên quan với nhau, song bị bó vào một thành phần;
2. Hội hợp logic: các thành phần cùng thực hiện các chức năng tương tự chẳng hạn như vào, xử lý lỗi...;
3. Kết dính theo thời điểm: tất cả các thành phần cùng hoạt hoá một lúc, chẳng hạn như bắt đầu và kết thúc;

4. Kết dính thủ tục: các phần tử trong thành phần được ghép lại trong một dãy điều khiển;

5. Kết dính truyền thống: tất cả các phần tử của thành phần cùng thao tác trên một dữ liệu vào và đưa ra cùng một dữ liệu ra;

6. Kết dính tuần tự: trong một thành phần, ra của một thành phần là vào của phần tử khác;

7. Kết dính chức năng: mỗi phần tử của một thành phần đều là cần thiết để thi hành cùng một chức năng nào đó.

Một đối tượng kết dính là một đối tượng thể hiện một thực thể đơn, tất cả các phép toán trên thực thể đó đều nằm trong thực thể đó. Vậy có thể xác định một lớp kết dính nữa, đó là:

8. Kết dính đối tượng: mỗi phép toán cho một chức năng, chức năng này cho phép các thuộc tính của đối tượng đó được cải biên, thanh tra và sử dụng như là cơ sở cho sự cung cấp dịch vụ.

Các lớp kết dính này không được định nghĩa chặt chẽ và cũng không phải luôn luôn quyết định được.

1.4.4.2. Sự ghép nối (Coupling)

Ghép nối liên quan đến kết dính, nó chỉ ra độ nối ghép giữa các đơn vị của chương trình. Hệ thống có nối ghép cao sẽ có độ kết dính mạnh giữa các đơn vị, và làm cho các đơn vị phụ thuộc lẫn nhau. Hệ thống nối ghép lỏng lẻo làm cho các đơn vị là độc lập hoặc là tương đối độc lập với nhau.

Các modul là được ghép nối chặt chẽ nếu chúng dùng các biến chung và nếu chúng trao đổi các thông tin điều khiển (ghép nối chung nhau và ghép nối điều khiển). Ghép nối lỏng lẻo đạt được khi đảm bảo rằng các thông tin biểu diễn được giữ trong thành phần này là giao diện dữ liệu của nó với các đơn vị khác lại thông qua danh sách tham số của nó.

Có lẽ ưu việt của thiết kế hướng đối tượng là bản chất của đối tượng dẫn tới việc tạo ra các hệ ghép nối lỏng lẻo.

Việc thừa kế trong hệ thống hướng đối tượng lại dẫn tới một dạng khác của ghép nối.

1.4.4.3. Sự hiểu được (Understandability)

Sự hiểu được liên quan tới một số đặc trưng thành phần sau:

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

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

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

d. Độ phức tạp: độ phức tạp của các thuật toán được dùng để thực hiện thành phần đó như thế nào?

Từ phức tạp ở đây được dùng theo nghĩa không hình thức. Độ phức tạo cao ám chỉ nhiều quan hệ giữa các thành phần khác nhau của thành phần thiết kế đó và một cấu trúc logic phức tạp mà nó dính líu đến độ sâu lồng nhau của phát biểu if-then-else. Các thành phần phức tạp là khó hiểu, vì thế người thiết kế nên làm cho thiết kế thành phần càng đơn giản càng tốt.

Đa số công việc về đo chất lượng thiết kế được tập trung và cố gắng đo độ phức tạp của thành phần và từ đó thu được một vài độ đo về sự dễ hiểu của thành phần. Độ phức tạp phản ánh độ dễ hiểu, nhưng cũng có một số nhân tố khác ảnh hưởng đến độ dễ hiểu, chẳng hạn như tổ chức dữ liệu và kiểu cách mô tả thiết kế. Các số đo độ phức tạp có thể chỉ cung cấp một chỉ số cho độ dễ hiểu của một thành phần.

Sự thừa kế trong thiết kế hướng đối tượng phản ánh độ dễ hiểu. Nếu sự thừa kế được dùng để gắn các chi tiết thiết kế thì thiết kế sẽ dễ hiểu hơn. Mặt khác nếu sử dụng sự thừa kế đòi hỏi người thiết kế phải nhìn nhiều lớp đối tượng khác nhau trong trật tự thừa kế thì độ dễ hiểu của thiết kế là được rút gọn.

1.4.4.4. Sự thích nghi được (Adaptability)

Nếu một thiết kế nhằm bảo trì được thì nó phải sẵn sàng thích nghi được, nghĩa là các thành phần của chúng nên được ghép nối lỏng lẻo. Để thích nghi được, thiết kế đó phải được soạn thảo tư liệu tốt, dễ hiểu, kiên định với việc thực hiện, việc thực hiện cũng được viết ra một cách dễ đọc.

Một thiết kế dễ thích nghi nghĩa là có mức nhìn thấy được cao, có một quan hệ rõ ràng giữa các mức khác nhau của thiết kế. Khi đó người đọc

thiết kế có thể tìm được các biểu diễn liên quan tới lược đồ cấu trúc biểu diễn sự vận chuyển của biểu đồ dòng dữ liệu.

Cần dễ dàng kết hợp chặt chẽ các biến đổi về thiết kế trong toàn bộ tư liệu thiết kế, nếu không, các thay đổi thiết kế có thể sẽ không thể đưa vào trong các mô tả liên quan. Tư liệu thiết kế đó có thể trở nên không kiên định.

Để có độ thích nghi cao thì một thành phần phải là tự chứa. Một thành phần có thể ghép nối lỏng lẻo (theo nghĩa là chỉ hợp tác) với các thành phần khác thông qua việc truyền các hộp thông báo. Điều này không giống như là tự chứa vì thành phần đó có thể dựa trên các thành phần khác. Muốn là tự chứa một cách hoàn toàn thì một thành phần không nên dùng các thành phần khác được xác định ngoại lai. Tuy nhiên điều đó lại mâu thuẫn với quan điểm là các thành phần hiện có nên được dùng lại. Vậy là cần có sự cân bằng giữa tính ưu việt của sự dùng lại và sự mất mát tính thích nghi được của thành phần.

Một trong những ưu việt chính của kế thừa trong thiết kế hướng đối tượng là các thành phần này có thể sẵn sàng thích nghi được. Cơ cấu thích nghi được này không dựa trên việc cải biên thành phần đó mà trên việc tạo ra một thành phần mới có kế thừa các thuộc tính và các phép toán của thành phần gốc. Khi các thuộc tính và phép toán được cải biên, các thành phần dựa trên thành phần cơ bản đó là không bị ảnh hưởng gì. Chỉ riêng tính thích nghi này là lý do duy nhất để các ngôn ngữ hướng đối tượng hữu hiệu trong việc tạo mẫu nhanh.

2. THIẾT KẾ HƯỚNG ĐỐI TƯỢNG (OBJECT ORIENTED DESIGN)

2.1. Cách tiếp cận hướng đối tượng

Bản chất duy nhất của thiết kế hướng đối tượng là khả năng xây dựng ba khái niệm thiết kế phần mềm quan trọng: trừu tượng, che dấu thông tin và modul.

Che dấu thông tin là chiến lược thiết kế dấu đi càng nhiều thông tin trong các thành phần càng hay. Có nghĩa là, việc kết hợp điều khiển logic và cấu trúc dữ liệu được thiết kế trong được thực hiện trong thiết kế càng chậm càng tốt, liên lạc thông qua các thông tin trạng thái dùng chung (biến tổng thể) là ít nhất, nhờ vậy khả năng hiểu được nâng lên.

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

2.2. Đặc trưng của thiết kế hướng đối tượng

1. Không có vùng dữ liệu dùng chung. Các đối tượng liên lạc với nhau bằng cách trao đổi thông báo chứ không phải các biến dùng chung;
2. Các đối tượng là các thực thể độc lập, dễ thay đổi vì rằng tất cả các trạng thái các thông tin biểu diễn chỉ ảnh hưởng trong phạm vi chính đối tượng đó thôi. Các thay đổi về biểu diễn thông tin có thể được thực hiện không cần có sự tham khảo tới các đối tượng hệ thống nào khác;
3. Các đối tượng có thể phân tán và có thể hành động tuần tự hoặc song song.

2.3. Các ưu nhược điểm của thiết kế hướng đối tượng

Ưu điểm

1. Dễ bảo trì vì các đối tượng là độc lập. Các đối tượng có thể hiểu và cải biên như là một thực thể độc lập. Thay đổi trong khi thực hiện một đối tượng hoặc thêm các dịch vụ sẽ không làm thay đổi hay ảnh hưởng tới các đối tượng hệ thống khác.
2. Các đối tượng là các thành phần dùng lại được. Một thiết có thể dùng lại các đối tượng đã được thiết kế trong các bản thiết kế trước đó.
3. Có các lớp hệ thống thể hiện quan hệ rõ ràng giữa các thực thể có thực (như các thành phần phần cứng) với các đối tượng điều khiển nó trong hệ thống. Điều này đạt được tính dễ hiểu trong thiết kế.

Nhược điểm

Việc minh định các đối tượng trong hệ thống là khó khăn. Cách nhìn tự nhiên là cách nhìn hướng chức năng và việc thích nghi với cách nhìn hướng đối tượng đôi khi là khó khăn.

2.4. Phân biệt giữa thiết kế hướng đối tượng và lập trình hướng đối tượng

Ta dễ nhầm lẫn hai khái niệm này. Ngôn ngữ lập trình hướng đối tượng là một ngôn ngữ lập trình cho phép thực hiện trực tiếp các đối tượng và cung cấp các lớp đối tượng và sự thừa kế.

Thiết kế hướng đối tượng là một chiến lược thiết kế, không phụ thuộc vào một ngôn ngữ cụ thể nào. Các ngôn ngữ lập trình hướng đối tượng và các khả năng bao gói đối tượng làm cho thiết kế hướng đối tượng được thực hiện một cách đơn giản hơn. Tuy nhiên một thiết kế hướng đối tượng cũng có thể thực hiện trong một ngôn ngữ kiểu như PASCAL hay C (không có đặc điểm bao gói như vậy).

Việc chấp nhận thiết kế hướng đối tượng như là một chiến lược hữu hiệu dẫn đến sự phát triển phương pháp thiết kế hướng đối tượng. ADA không phải là ngôn ngữ hướng đối tượng vì nó không trợ giúp sự thừa kế của các lớp, nhưng lại có thể thực hiện các đối tượng trong ADA bằng cách sử dụng các gói hoặc các nhiệm vụ, do đó ADA được dùng để thiết kế hướng đối tượng.

Phương pháp thiết kế hướng đối tượng vẫn còn là tương đối chưa chín muồi và đang có những thay đổi nhanh chóng. Phương pháp hiện đang sử dụng rộng rãi nhất là phương pháp dựa trên sự phân rã chức năng.

3. THIẾT KẾ HƯỚNG CẤU TRÚC

3.1. Cách tiếp cận hướng cấu trúc

Thiết kế hướng cấu trúc (hay thiết kế hướng chức năng) là một cách tiếp cận thiết kế phần mềm trong đó bản thiết kế được phân giải thành một bộ các đơn thể tác động lẫn nhau, mà mỗi đơn thể có một chức năng được xác định rõ ràng. Các chức năng có các trạng thái cục bộ nhưng chúng chia sẻ với nhau trạng thái hệ thống - trạng thái tập trung, mọi chức năng đều có thể truy cập được.

Có người nghĩ rằng thiết kế hướng chức năng đã lỗi thời và nên được thay thế bởi cách tiếp cận hướng đối tượng. Thế nhưng, nhiều tổ chức đã phát triển các chuẩn và các phương pháp dựa trên độ phân giải chức năng. Nhiều phương pháp thiết kế kết hợp với công cụ CASE đều là hướng chức năng. Rất nhiều các hệ thống đã được phát triển bằng cách sử dụng phương pháp tiếp cận hướng chức năng. Các hệ thống

đó vẫn sẽ được bảo trì cho tới một tương lai xa, bởi vậy thiết kế hướng chức năng vẫn sẽ còn được tiếp tục sử dụng rộng rãi.

Trong thiết kế hướng chức năng, người ta dùng các biểu đồ dòng dữ liệu (mô tả việc xử lý dữ liệu logic), các lược đồ cấu trúc (cấu trúc của phần mềm) và các mô tả PDL (mô tả thiết kế chi tiết). Khái niệm dòng dữ liệu đang hướng tới việc sử dụng một hệ thống vẽ biểu đồ tự động và sử dụng một dạng lược đồ cấu trúc có kèm theo các thông tin điều khiển.

Chiến lược thiết kế hướng chức năng dựa trên việc phân giải hệ thống thành bộ các chức năng có tương tác với trạng thái hệ thống tập trung – dùng chung cho các chức năng đó. Các chức năng này có thể có các thông tin trạng thái cục bộ nhưng chỉ dùng cho quá trình thực hiện các chức năng đó mà thôi.

Thiết kế chức năng gắn với các chi tiết một thuật toán của mỗi chức năng nhưng thông tin trạng thái hệ thống là không được che dấu. Điều này có thể gây ra vấn đề vì rằng một chức năng có thể thay đổi trạng thái theo một cách mà các chức năng khác không thể ngờ tới. Việc thay đổi một chức năng và cách nó sử dụng trạng thái của hệ thống có thể gây ra các tương tác bất ngờ đối với các chức năng khác.

Cách tiếp cận chức năng để thiết kế là tốt nhất khi mà khối lượng thông tin trạng thái hệ thống được làm nhỏ nhất và thông tin dùng chung nhau là rõ ràng.

3.2. Biểu đồ luồng dữ liệu

Bước thứ nhất của thiết kế hướng chức năng là phát triển một biểu đồ dòng dữ liệu hệ thống.

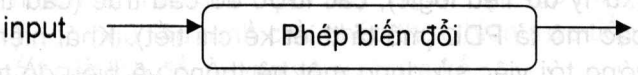
Biểu đồ dòng dữ liệu chỉ ra cách thức biến đổi dữ liệu *vào* thành dữ liệu *ra* thông qua một dãy các phép biến đổi.

Đó là một cách để mô tả hệ thống một cách dễ hiểu và không cần một sự huấn luyện đặc biệt nào. Biểu đồ này không nhất thiết bao gồm các thông tin điều khiển nhưng nên lập tư liệu các phép biến đổi dữ liệu.

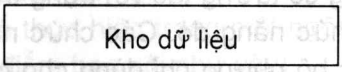
Biểu đồ dòng dữ liệu là một phần hợp nhất của một số các phương pháp thiết kế và các công cụ CASE, thường trợ giúp cho việc tạo ra biểu đồ dòng dữ liệu.

Các ký pháp trong biểu đồ dòng dữ liệu:

1. Hình chữ nhật góc tày: biểu diễn một phép biến đổi dòng dữ liệu *vào* thành dòng dữ liệu *ra,* tên của hình chữ nhật là tên của phép biến đổi đó.



2. Hình chữ nhật: biểu diễn một kho dữ liệu, tên của hình là tên của dữ liệu.



3. Hình tròn: biểu diễn giao tác người dùng với hệ thống (cung cấp thông tin vào hoặc hoặc thu nhận thông tin ra).

4. Các mũi tên: chỉ hướng dòng dữ liệu.
5. Các từ khoá **and** và **or.**
6. Khuyên tròn nối các dòng dữ liệu.

3.3. Lược đồ cấu trúc

Lược đồ cấu trúc chỉ ra *cấu trúc* các thành phần theo thứ bậc của hệ thống. Nó chỉ ra rằng các phần tử của một biểu đồ dòng dữ liệu có thể được thực hiện như thế nào với tư cách là một thứ bậc của đơn vị chương trình.

Lược đồ cấu trúc có thể được dùng như là một mô tả chương trình nhìn thấy được và các thông tin xác định các lựa chọn và các vòng lặp được dùng để trình bày một tổ chức tĩnh của thiết kế.

Mỗi thành phần chức năng được biểu diễn trên đồ thị có cấu trúc như là một hình chữ nhật. Thứ bậc này trình bày bằng cách nối các hình chữ nhật bởi các đường. Thông tin vào và thông tin ra cho một thành phần

được chỉ bởi việc dùng các mũi tên có gán tên, gồm mũi tên vào và mũi tên ra. Các kho dữ liệu được chỉ bởi các hình chữ nhật góc tầy và các thông tin vào từ người dùng được chỉ bởi các khuyên tròn.

Sau này để tránh nhầm với ký pháp đã dùng trong biểu đồ dòng dữ liệu, người ta dùng các khối trụ để biểu diễn kho dữ liệu và hình bình hành để biểu diễn thông tin vào.

3.4. Từ điển dữ liệu

Từ điển dữ liệu vừa có ích cho việc bảo trì hệ thống vừa có ích cho quá trình thiết kế. Với một lối vào đã được minh định trong biểu đồ phải có một từ điển dữ liệu cung cấp thông tin, thông tin về kiểu, chức năng của dữ liệu và một lý do cơ bản cho việc vào thông tin. Đôi khi người ta gọi cái này là một mô tả ngắn của chức năng thành phần.

Lối vào từ điển thành phần phải là một mô tả kiểu văn bản cho thành phần và phải được mô tả chi tiết.

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

4. GIAO DIỆN NGƯỜI SỬ DỤNG

4.1. Nhân tố con người và tương tác người máy

Thiết kế hệ thống máy tính bao gồm một loạt các hoạt động từ thiết kế phần cứng tới thiết kế giao diện người sử dụng. Trong đó, các kỹ sư điện tử thường chịu trách nhiệm về thiết kế phần cứng, còn các kỹ sư phần mềm phải chịu trách nhiệm thiết kế hệ thống và thiết kế giao diện người sử dụng. Các chuyên gia về nhân tố con người thường chỉ là cố vấn cho kỹ sư phần mềm chứ không trực tiếp thiết kế giao diện.

- Giao diện người dùng là cơ chế thiết lập giao tiếp giữa chương trình và người dùng. Nếu nhân tố con người được tính tới thì đối thoại sẽ trôi chảy và sẽ thiết lập được nhịp điệu giữa người dùng và chương trình. Nếu nhân tố người dùng bị

bỏ qua thì hệ thống gần như bao giờ cũng bị coi như "*không thân thiện*".

- Giao diện sử dụng của một hệ thống thường được chọn làm tiêu chuẩn so sánh để đánh giá hệ thống theo quan điểm người sử dụng.

Một giao diện khó sử dụng sẽ ít nhiều gây ra sai lầm của người sử dụng. Trong trường hợp xấu nhất nó có thể làm cho hệ thống bị huỷ hoại bất chấp chức năng của nó.

Một giao diện thiết kế kém có thể làm cho người sử dụng gây ra lỗi. Nếu thông tin được biểu diễn lẫn lộn có thể làm người dùng hiểu nhầm ý nghĩa các khoản mục thông tin gây ra một chuỗi các hành vi nguy hiểm.

- Giao diện người sử dụng phải tính đến nhu cầu, kinh nghiệm và khả năng của người sử dụng.

Trong những ngày đầu của tin học (trước khi có những thiết bị hiển thị đồ hoạ như chuột, máy tính tốc độ cao...) một kiểu tương tác người-máy thực tế duy nhất là giao diện chỉ lệnh và hỏi (thế hệ 1). Trao đổi thuần tuý văn bản và được dẫn thông qua chỉ lệnh và các đáp ứng với các câu hỏi do hệ thống sinh ra. Người sử dụng có thể trao đổi với hệ thống bằng cách xác định các chỉ lệnh như:

```
>run progr1.exe/debug='on'/out=p1/in=1/alloc=1000k
*  RUN ALLOCATION TO BE QUEUED? >> yes
*AUTOMATIC CHECKPOINTING INTERVALS?>>5
```

Như vậy mặc dầu chỉ lệnh và câu hỏi như thế rất chính xác nhưng cũng vẫn sinh lỗi và khó học, khó nhớ. Một cải tiến về giao diện chỉ lệnh và hỏi là *giao diện đơn* (menu) đơn giản (thế hệ 2). Tại đây, một danh sách các tùy chọn được nêu ra cho người dùng chọn thông qua một mã gõ vào nào đó.

Khi phần cứng trở nên tinh vi hơn và kỹ sư phần mềm học được nhiều hơn về nhân tố con người và tác động tới thiết kế giao diện thì giao diện trỏ và chọn hướng cửa sổ bắt đầu tiến hóa (đôi khi còn được gọi là giao diện cửa sổ) gồm các biểu tượng, menu và thiết bị trỏ chuột (thế hệ 3)

Lợi ích của giao diện "thế hệ ba":

1. Có thể hiển thị đồng thời nhiều kiểu thông tin khác nhau, cho phép người dùng chuyển hoàn cảnh mà không mất mối nối trực

quan với công việc khác. Cửa sổ cho phép người dùng thực hiện nhiều nhiệm vụ trao đổi và nhận biết mà không chán.

2. Nhiều nhiệm vụ tương tác khác có sẵn qua sơ đồ kéo xuống. Những đơn như vậy cho phép người dùng thực hiện các nhiệm vụ kiểm soát và đối thoại một cách dễ dàng.

3. Việc dùng biểu tượng đồ họa, đơn, kéo xuống, nút và kỹ thuật cuộn làm giảm khối lượng gõ. Điều này có thể làm tăng tính hiệu quả tương tác cho những người mà kỹ năng sử dụng bàn phím của họ còn hạn chế, làm cho máy tính trở thành thân thiện với những người sợ bàn phím.

Thế hệ giao diện người máy (HCI - *Human Computer Interface*) hiện tại nối tất cả các thuộc tính của giao diện thế hệ ba với siêu văn bản (hypertext) và chế độ đa nhiệm - khả năng thực hiện một số nhiệm vụ khác nhau đồng thời (theo quan điểm của người dùng). Các giao diện "thế hệ bốn" này hiện nay đã có nhiều máy trạm làm việc (work station) và PC (Personal Computer).

4.2. Thiết kế giao diện người - máy

Thiết kế giao diện người - máy là một phần của chủ đề lớn là thiết kế phần mềm mà chúng ta đã biết. Toàn bộ tiến trình thiết kế giao diện người dùng bao gồm:

- Bắt đầu với việc tạo ra các mô hình khác nhau về chức năng hệ thống (như được cảm nhận từ bên ngoài).
- Phác họa ra các nhiệm vụ hướng con người và máy tính cần để đạt tới chức năng hệ thống.
- Xem xét vấn đề thiết kế áp dụng cho mọi thiết kế giao diện.
- Sử dụng các công cụ làm bản mẫu.
- Cài đặt mô hình thiết kế và đánh giá kết quả và chất lượng.

4.2.1. Mô hình thiết kế giao diện

Có 4 mô hình khác nhau khi thiết kế giao diện người - máy:

- Mô hình thiết kế: kỹ sư phần mềm tạo ra.
- Mô hình người dùng: kỹ sư phần mềm/ kỹ sư con người.
- Mô hình của người dùng/cảm nhận hệ thống: người dùng cuối cùng xây dựng một hình ảnh tinh thần.

- Hình ảnh hệ thống: người cài đặt hệ thống tạo ra.

Các mô hình này khác nhau. Vai trò của thiết kế giao diện là điều hòa những sự khác biệt này và đưa ra một cách biểu diễn nhất quán cho giao diện, cụ thể:

- Mô hình thiết kế của toàn bộ hệ thống là tổ hợp các biểu diễn dữ liệu, kiến trúc và thủ tục của phần mềm.
- Mô hình người dùng: mô tả sơ lược hệ thống cho người dùng cuối. Để có hiệu quả, mọi thiết kế nên bắt đầu với một hiểu biết về người dùng đã dự kiến, kể cả thông tin về tuổi tác, giới tính, khả năng thể chất, nền tảng giáo dục, văn hóa, động cơ, mục đích và nhân cách (có thể phân thành người mới học, người ít dùng nhưng có hiểu biết, người dùng thường xuyên và có hiểu biết).
- Cảm nhận hệ thống là hình ảnh của hệ thống mà người dùng mang trong đầu.
- Hình ảnh hệ thống tổ hợp cách biểu lộ bên ngoài của hệ thống dựa trên máy tính (nhìn và cảm thấy giao diện) với mọi thông tin hỗ trợ (sách, tài liệu sử dụng, băng video) mô tả cho cú pháp và ngữ nghĩa của hệ thống. Khi hình ảnh hệ thống và cảm nhận hệ thống trùng nhau thì nói chung người dùng cảm thấy thoải mái với phần mềm và dùng nó một cách hiệu quả.

Về bản chất, các mô hình này làm cho người thiết kế giao diện thỏa mãn với vấn đề mấu chốt của nguyên lý quan trọng nhất trong thiết kế giao diện người dùng: biết người dùng, biết nhiệm vụ

4.2.2. Phân tích và mô hình hóa nhiệm vụ trong thiết kế giao diện

Việc khởi thảo từng bước (cũng còn được gọi là phân tách chức năng hay làm mịn dần từng bước) như cơ chế để làm mịn các nhiệm vụ xử lý cần cho phần mềm hoàn thành một chức năng mong muốn nào đó.

Phân tích nhiệm vụ cũng có cách tiếp cận tương tự nhưng được áp dụng cho các hoạt động con người. Trước hết phải xác định và phân loại các nhiệm vụ.

Một khi từng nhiệm vụ hay hành động đã được xác định thì việc thiết kế giao diện bắt đầu. Bước đầu tiên trong tiến trình thiết kế giao diện có thể được thực hiện bằng cách tiếp cận sau:

1. Thiết lập các mục tiêu và ý đồ cho nhiệm vụ.

2. Ánh xạ thành dãy các hành động xác định.
3. Xác định dãy hành động khi nó được thực hiện ở mức giao diện.
4. Chỉ ra trạng thái của hệ thống, tức là giao diện giống thế nào vào lúc hành động trong khi dãy đó được thực hiện.
5. Xác định các cơ chế điều khiển, như thiết bị và hành động sẵn có cho người dùng để thay đổi trạng thái hệ thống.
6. Chỉ ra cách thức cơ chế điều khiển này ảnh hưởng tới trạng thái hệ thống.
7. Chỉ ra cách thức người dùng diễn giải trạng thái của hệ thống từ thông tin được cung cấp qua giao diện.

4.2.3. Các vấn đề trong thiết kế giao diện

Bốn vấn đề thiết kế thông thường gần như bao giờ cũng nổi lên:

1. Thời gian hệ thống đáp ứng (độ dài, độ biến thiên).
2. Tiện nghi giúp đỡ người dùng (tích hợp, phụ thêm).
3. Giải quyết thông tin lỗi (thông báo, cảnh bảo).
4. Gắn nhãn chỉ lệnh (tiện nghi tạo macro).

4.2.3.1. Thời gian hệ thống đáp ứng

Thời gian hệ thống đáp ứng là vấn đề nan giải đối với nhiều hệ thống tương tác. Nói chung, thời gian hệ thống đáp ứng được đo từ điểm tại đó người dùng thực hiện hành động điều khiển nào đó (như gõ phím hay nháy chuột) cho tới khi phần mềm đáp ứng cái ra hay hành động mong muốn.

Thời gian hệ thống đáp ứng có hai đặc trưng quan trọng: độ dài và độ biến thiên. Nếu độ dài thời gian đáp ứng quá lâu thì người dùng không tránh khỏi chán nản và căng thẳng. Tuy nhiên thời gian đáp ứng quá ngắn cũng có thể bất lợi nếu người dùng bị giao diện giữ nhịp. Sự đáp ứng nhanh chóng có thể buộc người dùng phải vội vã do đó phạm sai lầm.

"Độ biến thiên" nói tới độ lệch khỏi thời gian đáp ứng trung bình và theo nhiều cách thức thì nó còn quan trọng hơn đặc trưng thời gian đáp ứng. Độ biến thiên thấp làm cho người dùng thiết lập được nhịp điệu, cho dù thời gian đáp ứng tương đối lâu. Chẳng hạn đáp ứng 1 giây đối với một lệnh thì được ưa chuộng hơn cho một đáp ứng biến thiên từ 0,1 đến

0,25 giây. Trong trường hợp sau, người dùng bao giờ cũng bị mất cân bằng, bao giờ cũng phải tự hỏi liệu có cái gì đó "khác" sẽ xuất hiện bên ngoài khung cảnh này hay không.

4.2.3.2. Tiện nghi giúp đỡ người dùng

Ta thường gặp phải hai kiểu tiện nghi trợ giúp khác nhau: tích hợp và phụ thêm. *Tiện nghi trợ giúp tích hợp* được thiết kế trong phần mềm ngay từ đầu. Nó thường cảm ngữ cảnh, làm cho người dùng lựa được từ các chủ đề có liên quan tới hành động hiện đang được thực hiện. Hiển nhiên, điều này rút gọn thời gian cần để người dùng thu được sự trợ giúp và tạo ra "sự thân thiện" của giao diện. *Tiện nghi trợ giúp phụ thêm* được thêm vào cho phần mềm sau khi hệ thống đã được xây dựng. Theo nhiều cách, nó thực sự là một tài liệu cho người dùng trực tuyến với một khả năng hỏi hạn chế. Người dùng có thể phải tìm kiếm trong một danh sách hàng trăm chủ đề để tìm hướng dẫn thích hợp, đôi khi lựa chọn sai và nhận được những thông tin không liên quan. Chắc Vì vậy tiện nghi trợ giúp tích hợp được ưa thích hơn cách tiếp cận phụ thêm.

Cần phải đề cập tới một số vấn đề khi xem xét tiện nghi trợ giúp:

- Liệu trợ giúp có sẵn có với tất cả các chức năng hệ thống và vào mọi lúc trong tương tác hệ thống không? Các tùy chọn bao gồm: trợ giúp chỉ cho một tập con của mọi chức năng và hành động; trợ giúp cho tất cả các chức năng.
- Người dùng sẽ yêu cầu trợ giúp như thế nào? Các tùy chọn bao gồm: menu trợ giúp; phím trợ giúp đặc biệt; chỉ lệnh HELP.
- Trợ giúp sẽ được trình bày như thế nào? Các tùy chọn bao gồm: cửa sổ riêng biệt; tham khảo tới tài liệu in; gợi ý một hay hai dòng được tạo ra trong một vị trí màn hình cố định.
- Người dùng sẽ trở về với tương tác thông thường như thế nào? Các tùy chọn bao gồm: nút trở về được hiển thị trên màn hình; phím chức năng hay dãy điều khiển.
- Thông tin trợ giúp sẽ được cấu trúc như thế nào? Các tùy chọn bao gồm:
  + Cấu trúc "phẳng" trong đó mọi thông tin đều được thâm nhập tới qua một từ khóa;

+ Cấp bậc phân tầng của thông tin cung cấp chi tiết ngày càng tăng khi người dùng tiến sâu vào cấu trúc;

+ Sử dụng siêu văn bản.

4.2.3.3. Giải quyết thông tin lỗi

Thông báo lỗi và cảnh báo là tin dữ đối với người dùng khi một cái gì đó chạy không đúng. Trường hợp thông báo lỗi hoặc lời cảnh báo truyền đạt thông tin lỗi chỉ làm tăng thêm sự chán nản của người dùng. Có vài người dùng máy tính gặp phải lỗi có dạng SERVER SYSTEM FAILURE - 14A.

Thì ở đâu đó, nhất định phải có lời giải thích cho lỗi 14A; nếu không thì tại sao người thiết kế phải thêm vào thông tin định dạng này? Như vậy, thông báo lỗi không đưa ra chỉ dẫn cụ thể về cái gì sai hay phải lấy thông tin phụ ở đâu. Thông báo lỗi được trình bày như ví dụ được nêu ở trên là rất mơ hồ và không cung cấp đủ thông tin để sửa lỗi.

Nói chung, mọi thông báo lỗi hay cảnh báo do hệ thống tương tác tạo ra nên có các đặc trưng sau:

- Thông báo nên mô tả vấn đề theo lối nói mà người dùng có thể hiểu được.
- Thông báo nên đưa ra những lời khuyên có tính xây dựng để khôi phục từ lỗi.
- Thông báo nên chỉ ra bất kỳ hậu quả lỗi tiêu cực nào (như tệp dữ liệu có thể hỏng) để cho người dùng có thể kiểm tra đảm bảo rằng chúng không xuất hiện (hay sửa ngay chúng nếu có xuất hiện)
- Thông báo nên đi kèm với tín hiệu nghe được hay thấy được. Tức là một tiếng bíp có thể sinh ra đi kèm với việc hiển thị thông báo, hay thông báo có thể nhấp nháy chốc lát hay được hiển thị theo màu dễ nhận ra như "màu lỗi".
- Thông báo nên có tính chất "phi đánh giá", tức là lời đưa ra đừng hàm ý trách móc người dùng.

4.2.3.4. Gắn nhãn chỉ lệnh

Ngày nay, việc dùng giao diện hướng cửa sổ, chỉ và chọn, đã làm giảm bớt việc dựa vào chỉ lệnh gõ vào, nhưng nhiều người dùng vẫn ưa thích cách tương tác theo chỉ lệnh. Trong nhiều tình huống, người dùng có thể được cung cấp một tùy chọn - các chức năng phần mềm có thể

được lựa ra từ một đơn tĩnh hay đơn kéo xuống thông qua một dãy chỉ lệnh bàn phím nào đó.

Một số vấn đề thiết kế nảy sinh khi chỉ lệnh được đưa ra như một mốt tương tác:

- Liệu mọi tùy chọn đơn có tương ứng với một chỉ lệnh không?
- Chỉ lệnh sẽ có dạng nào? Các tùy chọn bao gồm: dãy điều khiển (như ^P); phím chức năng; từ gõ vào.
- Việc học và nhớ chỉ lệnh sẽ khó đến mức nào? Có thể làm được gì nếu quên mất chỉ lệnh (xem thảo luận về trợ giúp được trình bày trong mục này)?
- Liệu chỉ lệnh có được người dùng làm cho phù hợp hay viết tắt không?

Trong số ứng dụng ngày càng tăng, người thiết kế giao diện đưa ra một tiện nghi tạo macro chỉ lệnh để cho phép người dùng ghi lại một dãy các chỉ lệnh hay dùng theo tên do người dùng xác định. Thay vì phải gõ từng lệnh một thì chỉ cần gõ macro chỉ lệnh và mọi chỉ lệnh trong đó sẽ được thực hiện tuần tự.

4.2.4. Công cụ cài đặt

Tiến trình thiết kế giao diện người dùng có tính lặp. Tức là một mô hình thiết kế được tạo ra, cài đặt như một bản mẫu, được người dùng xem xét (người thích hợp với mô hình người dùng đã mô tả dưới đây), và được thay đổi dựa trên lời góp ý của họ. Để thích nghi với cách tiếp cận thiết kế lặp này, một lớp rộng các công cụ thiết kế giao diện và làm bản mẫu đã được cải tiến. Được gọi là bộ công cụ giao diện người dùng hay hệ thống phát triển giao diện người dùng (UIDS - *User Interface Development System*), các công cụ này đưa ra các modul hay đối tượng làm thuận tiện cho việc tạo ra cửa sổ, đơn, tương tác thiết bị, thông báo lỗi, chỉ lệnh và nhiều phần tử khác của môi trường tương tác.

Sử dụng phần mềm đóng gói sẵn có thể được người thiết kế và người cài đặt hay giao diện người dùng sử dụng trực tiếp, một UIDS cung cấp các cơ chế có sẵn cho những vấn đề sau:

- Việc quản lý thiết bị đưa vào (như chuột hay bàn phím);
- Làm hợp lệ cái vào của người dùng;
- Giải quyết lỗi và hiển thị thông báo lỗi;

- Cung cấp phản hồi (như tự động hiển thị cái vào);
- Cung cấp trợ giúp và lời nhắc;
- Giải quyết cửa sổ và trường, cuộn bên trong cửa sổ;
- Thiết lập mối nối giữa phần mềm ứng dụng và giao diện;
- Cô lập ứng dụng với các chức năng quản lý giao diện;
- Cho phép người dùng làm phù hợp giao diện.

Các chức năng được mô tả trên đây có thể được cài đặt bằng cách dùng hoặc cách tiếp cận dựa trên ngôn ngữ hoặc đồ họa.

4.2.5. Tiến hóa thiết kế

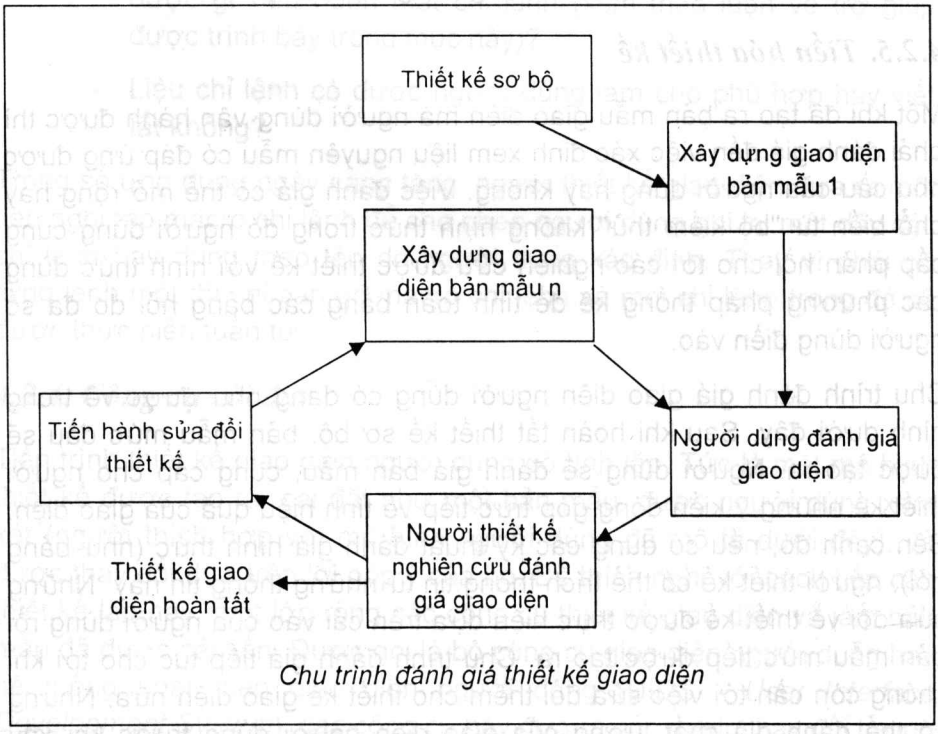
Một khi đã tạo ra bản mẫu giao diện mà người dùng vận hành được thì phải đánh giá đến việc xác định xem liệu nguyên mẫu có đáp ứng được nhu cầu của người dùng hay không. Việc đánh giá có thể mở rộng hay phổ biến từ "bộ kiểm thử" không hình thức trong đó người dùng cung cấp phản hồi cho tới các nghiên cứu được thiết kế với hình thức dùng các phương pháp thống kê để tính toán bằng các bảng hỏi do đa số người dùng điền vào.

Chu trình đánh giá giao diện người dùng có dạng như được vẽ trong hình dưới đây. Sau khi hoàn tất thiết kế sơ bộ, bản mẫu mức đầu sẽ được tạo ra. Người dùng sẽ đánh giá bản mẫu, cung cấp cho người thiết kế những ý kiến đóng góp trực tiếp về tính hiệu quả của giao diện. Bên cạnh đó, nếu có dùng các kỹ thuật đánh giá hình thức (như bảng hỏi), người thiết kế có thể trích thông tin từ những thông tin này. Những sửa đổi về thiết kế được thực hiện dựa trên cái vào của người dùng rồi bản mẫu mức tiếp được tạo ra. Chu trình đánh giá tiếp tục cho tới khi không còn cần tới việc sửa đổi thêm cho thiết kế giao diện nữa. Nhưng có thể đánh giá chất lượng của giao diện người dùng trước khi xây dựng bản mẫu không? Câu trả lời là nếu các vấn đề tiềm năng có thể được bộc lộ và sửa đổi sớm thì có thể rút gọn bớt một số chu trình và thời gian phát triển cho bản mẫu sẽ ngắn đi.

Nếu một mô hình thiết kế giao diện được tạo ra thì có thể áp dụng một số tiêu chuẩn đánh giá trong các cuộc họp xét duyệt thiết kế:

1. Độ dài và độ phức tạp của đặc tả viết về hệ thống và giao diện của nó cung cấp một chỉ dẫn về khối lượng học tập mà người dùng hệ thống cần học.

2. Số các chỉ lệnh được xác định và số trung bình các đối số trên chỉ lệnh đưa ra một chỉ dẫn về thời gian tương tác và hiệu quả tổng thể của hệ thống.
3. Số các hành động, chỉ lệnh và trạng thái hệ thống được mô hình thiết kế nêu ra cũng chỉ ra khối lượng cần nhớ của người dùng hệ thống
4. Phong cách tương tác, tiện nghi trợ giúp và giao thức xử lý lỗi đưa ra một chỉ dẫn chung về độ phức tạp của giao diện và mức độ người dùng sẽ chấp nhận được.



Chu trình đánh giá thiết kế giao diện

Một khi bản mẫu đầu tiên được xây dựng thì người thiết kế có thể thu thập rất nhiều dữ liệu định tính và định lượng sẽ trợ giúp cho việc đánh giá giao diện. Để thu thập dữ liệu định tính, có thể phân phát bảng hỏi cho người dùng bản mẫu.

Câu hỏi có thể là:

1. Trả lời đơn giản có/không.
2. Trả lời số.
3. Trả lời theo thang (chủ quan).

4. Trả lời theo phần trăm (chủ quan).

Thí dụ:

1. Các chỉ lệnh có dễ nhớ không? Có/không.

2. Bạn đã dùng bao nhiêu chỉ lệnh khác nhau?

3. Học cách vận hành hệ thống dễ đến mức nào? (thang 1 tới 5)

4. So sánh với các giao diện khác bạn đã dùng, giao diện này ở tỉ lệ nào? đỉnh 1%, đỉnh 10%, đỉnh 25%, đỉnh 50%, đáy 50%.

Nếu đòi hỏi dữ liệu định lượng thì có thể tiến hành một dạng phân tích nghiên cứu thời gian. Quan sát người dùng trong khi tương tác và những dữ liệu như số nhiệm vụ được hoàn thành đúng trong một khoảng thời gian chuẩn; tần số sử dụng chỉ lệnh; dãy các chỉ lệnh; thời gian dành cho "việc nhìn" vào màn hình; số lỗi; kiểu lỗi và thời gian khôi phục lỗi; thời gian dùng trợ giúp; và số lần tham khảo trợ giúp trong khoảng thời gian chuẩn được thu thập và dùng như hướng dẫn cho việc sửa đổi giao diện.

4.3. Hướng dẫn thiết kế giao diện

Ba phạm trù của hướng dẫn thiết kế giao diện người máy HCI là:

1. Tương tác chung
2. Hiển thị thông tin
3. Vào dữ liệu

4.3.1. Tương tác chung

Những hướng dẫn sau tập trung các yêu cầu tổng thể giao diện

Nhất quán: phải dùng định dạng nhất quán cho việc chọn đơn, vào chỉ lệnh, hiển thị dữ liệu và vô số các chức năng khác xuất hiện trong PC.

Cho thông tin phản hồi có nghĩa: cung cấp cho người sử dụng những thông tin phản hồi bằng hình ảnh và âm thanh nhằm thiết lập việc trao đổi thông tin hai chiều (giữa người sử dụng và giao diện).

Yêu cầu kiểm chứng mọi hành động phá hủy không tầm thường: nếu người dùng yêu cầu xóa một tệp, ghi đè lên thông tin bản chất hay yêu cầu kết thúc chương trình thì một thông báo "Bạn có chắc ...?" nên xuất hiện ra.

Cho phép dễ dàng lần ngược hành động: Các chức năng UNDO (hoàn tác) hay REVERSE (đảo ngược) đã giúp cho hàng nghìn người dùng khỏi mất hàng nghìn giờ làm việc. Khả năng lần ngược nên có sẵn trong mọi ứng dụng tương tác.

Giảm thiểu khối lượng thông tin phải nhớ giữa các hành động: không nên trông đợi người dùng cuối cùng nhớ được một danh sách các số hiệu hay tên gọi để cho người ấy có thể dùng lại chúng trong những chức năng kế tiếp sau. Cần phải tối thiểu tải trọng ghi nhớ.

Tìm kiếm tính hiệu quả trong đối thoại, vận động và ý nghĩ: nên tối thiểu việc dùng các phím, cần phải xem xét khoảng cách chuột phải đi qua giữa các điểm trong thiết kế bố trí màn hình, và đừng đẩy người dùng vào tình huống phải tự hỏi "Cái này nghĩa là gì nhỉ?"

Dung thứ cho sai lầm: hệ thống nên được thiết kế để có thể tự bảo vệ khỏi lỗi của người dùng để khỏi bị hỏng.

Phân loại các hoạt động theo chức năng và tổ chức màn hình hài hòa theo vùng. Một trong những cái lợi chính của đơn kéo xuống là khả năng tổ chức các lệnh theo kiểu. Về bản chất người thiết kế nên cố gắng đặt các chỉ lệnh và hành động "nhất quán".

Cung cấp tiện nghi trợ giúp cảm ngữ cảnh.

Dùng các động từ đơn giản hay cụm động từ ngắn để đặt tên chỉ lệnh: Tên chỉ lệnh dài dòng thì khó nhận dạng và nhớ. Nó cũng có thể chiếm không gian không cần thiết trong danh sách đơn.

4.3.2. Hiển thị thông tin

Thông tin được "hiển thị" theo nhiều cách khác nhau: với văn bản, tranh ảnh và âm thanh bằng cách sắp đặt, di chuyển và kích cỡ; dùng mầu sắc, độ phân giải; và thậm chí bằng cả việc bỏ lửng. Các hướng dẫn sau đây tập trung vào hiển thị thông tin:

Chỉ hiển thị thông tin có liên quan tới hiện tại để người dùng không phải khó nhọc lần qua dữ liệu, đơn, và hiệu ứng đồ họa để thu được thông tin liên quan tới một chức năng hệ thống riêng.

Đừng chôn vùi người dùng dưới dữ liệu - hãy dùng định dạng trình bày cho phép hấp thụ nhanh chóng thông tin: đồ họa hoặc sơ đồ nên thay thế cho các bảng lớn.

Dùng nhãn nhất quán, cách viết tắt chuẩn và mầu sắc dự kiến trước được. Ý nghĩa của hiển thị hiển nhiên không cần tới tham khảo thêm nguồn thông tin bên ngoài.

Cho phép người dùng duy trì ngữ cảnh trực quan: nếu việc hiển thị đồ họa máy tính được thay đổi tỷ lệ thì hình ảnh gốc nên được hiển thị thường xuyên (dưới dạng rút gọn tại góc màn hình) để cho người dùng hiểu được vị trí tương đối của phần hình ảnh hiện đang được xét.

Đưa ra những thông báo lỗi có nghĩa

Dùng chữ hoa hay chữ thường, cách tụt lề và gộp nhóm văn bản để trợ giúp cho việc hiểu. Nhiều thông tin được HCI truyền đạt là văn bản, ngay cả cách bố trí và hình dạng của văn bản cũng có tác động đáng kể đến sự thoải mái để người dùng hấp thu thông tin.

Sử dụng cửa sổ (nếu sẵn có) để đóng khung các kiểu thông tin khác nhau: cửa sổ cho phép người dùng "giữ" nhiều kiểu thông tin trong phạm vi truy nhập tới dễ dàng.

Dùng cách hiển thị tương tự để biểu diễn những thông tin dễ được hấp thu hơn với dạng biểu diễn này. Chẳng hạn, hiển thị áp suất của bể chứa trong xưởng lọc dầu sẽ ít tác dụng nếu dùng cách biểu diễn số. Tuy nhiên, nếu hiển thị dạng nhiệt kế dùng để chuyển động theo chiều đứng và sự thay đổi màu sắc để chỉ ra những điều kiện áp suất thay đổi thì điều này sẽ cung cấp cho người dùng cả thông tin tuyệt đối và tương đối.

Xem xét vùng hiển thị có sẵn trên màn hình và dùng nó một cách có hiệu quả. Khi dùng nhiều cửa sổ, ít nhất nên có sẵn không gian để chỉ ra một phần cho từng cửa sổ này. Bên cạnh đó, kích cỡ màn hình (vấn đề công nghệ hệ thống) nên được đưa lựa chọn để hòa hợp với kiểu ứng dụng cần được cài đặt.

4.3.3. Vào dữ liệu

Những hướng dẫn sau đây tập trung vào việc đưa vào dữ liệu:

Tối thiểu số hành động đưa vào mà người dùng cần thực hiện. Việc rút gọn số liệu gõ vào là điều yêu cầu trước hết. Điều này có thể được thực hiện bằng cách dùng chuột để chọn từ một tập đã xác định sẵn các cái vào; dùng "thanh trượt" để xác định cái vào trong một miền giá trị; dùng "macro" làm cho chỉ một phím được chuyển thành một tập dữ liệu với miền cái vào phức tạp hơn.

Duy trì sự nhất quán giữa hiển thị thông tin và cái vào dữ liệu: các kí tự hiển thị trực quan (như kích cỡ văn bản, mầu sắc, cách bố trí) nên được thực hiện đối với miền cái vào.

Cho phép người dùng làm phù hợp cái vào: điều này áo dụng cho đối với người dùng chuyên gia để có thể quyết định tạo ra các chỉ lệnh đã sửa đổi phù hợp với mình hay bỏ qua một số kiểu cảnh báo và kiểm chứng hành động và HCI nên cho phép điều này.

Tương tác nên mềm dẻo nhưng cũng nên hòa hợp với cách đưa vào ưa thích. Mô hình người dùng sẽ trợ giúp cho việc xác định cách đưa vào nào là ưa thích. Một thư ký có thể rất thích với cách đưa vào từ bàn phím, trong khi người quản lý lại có thể thấy thoải mái khi dùng thiết bị trỏ và nháy như chuột.

Khử kích hoạt các chỉ lệnh không thích hợp trong hoàn cảnh của hành động hiện tại. Điều này bảo vệ cho người dùng khỏi phải cố dùng một hành động nào đó có thể làm phát sinh lỗi.

Để cho người dùng kiểm soát luồng tương tác, người dùng nên có khả năng nhảy qua các hành động không cần thiết, thay đổi trật tự các hành động yêu cầu (khi có thể được trong hoàn cảnh của ứng dụng), và khôi phục được dữ liệu từ các điều kiện lỗi mà không phải ra khỏi chương trình.

Cung cấp trợ giúp cho mọi hành động đưa vào, hỗ trợ tối đa. Đừng yêu cầu người dùng phải gõ nhiều. VD: phải gõ .00 cho toàn bộ số tiền đưa vào, nên đưa ra các giá trị mặc định mọi lúc có thể và không bao giờ yêu cầu người dùng đưa vào những thông tin có thể tự động thu thập hay tính toán được bên trong chương trình.

4.4. Chuẩn giao diện

Phần lớn những người xây dựng ứng dụng phần mềm hiện đại đều cài đặt các giao diện trỏ và hướng cửa sổ. Việc tạo ra những giao diện như vậy không phải là dễ dàng. Việc chuẩn hoá giao diện sẽ làm lợi cho người phát triển và người dùng HCI.

Người phát triển có thể dùng lại các modul và đối tượng HCI đã có (hoặc đã đóng gói), do đó có thể tạo ra giao diện nhanh chóng hơn và có chất lượng cao hơn đáng kể.

Người dùng trở nên quen thuộc với cách bố trí và nhịp điệu của HCI và do đó có thể học bất kỳ ứng dụng mới nào có dùng chuẩn giao diện một cách nhanh chóng hơn nhiều. Sau một thời gian việc sử dụng giao diện

trở thành trực giác, do đó hiệu suất hơn nhiều đối với người dùng cuối cùng.

Mặc dầu một số chuẩn giao diện cạnh tranh đang được phát triển, chuẩn được dùng phổ biến nhất là X-WINDOWS (nhiều cửa sổ). Hệ thống X-WINDOW xác định một cú pháp và ngữ nghĩa cho thiết kế HCI và cung cấp các công cụ để tạo ra hiển thị, cửa sổ và đồ hoạ cũng như những giao thức để xử lý tài nguyên, tương tác thiết bị và giải quyết sự kiện. Một số biến thể và mở rộng của chuẩn hệ thống X-WINDOW đã được phát triển và dùng trên PC và máy trạm làm việc dưới UNIX và các hệ điều hành khác.

5. TÀI LIỆU THIẾT KẾ PHẦN MỀM

Dàn bài tài liệu thiết kế được nêu trong bảng dưới đây có thể được dùng như một mô hình cho đặc tả thiết kế. Mỗi mục đều đề cập tới các khía cạnh khác nhau của biểu diễn thiết kế.

Dàn bài tài liệu trình bày một mô tả thiết kế đầy đủ về phần mềm. Các mục của đặc tả thiết kế được hoàn chỉnh khi người thiết kế làm mịn việc trình bày của mình về phần mềm.

Mục I mô tả phạm vi toàn cục của thiết kế, nhiều thông tin được suy ra từ đặc tả hệ thống và các tài liệu khác trong pha xác định phần mềm. Các tham khảo riêng tới tài liệu hỗ trợ được thực hiện trong mục II. Mục III, mô tả thiết kế được hoàn tất như một phần của thiết kế sơ bộ. Chúng ta đã lưu ý rằng thiết kế là hướng thông tin (luồng dữ liệu và/hoặc cấu trúc dữ liệu) sẽ khống chế kiến trúc của phần mềm. Biểu đồ luồng dữ liệu hay các biểu diễn dữ liệu khác được phát triển, trong khi, các phân tích yêu cầu được làm mịn và được dùng để điều khiển cấu trúc phần mềm. Bởi vì luồng thông tin đã có sẵn nên mô tả giao diện có thể được phát triển cho các phần tử của phần mềm.

Các mục IV và V được phát triển khi thiết kế sơ bộ chuyển thành thiết kế chi tiết. Các modul hay các phần tử định địa chỉ tách biệt được của phần mềm (chương trình con, hàm thủ tục) khởi đầu được mô tả bằng lời thuật xử lý trong tiếng Anh. Lời thuật xử lý giải thích chức năng thủ tục của một modul. Sau này, một công cụ thiết kế thủ tục sẽ được dùng để dịch lời thuật thành một mô tả có cấu trúc. Mô tả cách tổ chức dữ liệu được nêu trong mục V. Các cấu trúc tệp duy trì trên bộ nhớ phụ được mô tả trong thiết kế sơ bộ; dữ liệu toàn cục được gán; tham khảo chéo gắn các moule riêng lẻ với dữ liệu toàn cục được thiết lập.

Dàn bài đặc tả thiết kế

I. Phạm vi

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

II. Tài liệu tham khảo

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

III. Mô tả thiết kế

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

IV. Modul (cho mỗi modul)

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

V. Cấu trúc tệp và dữ liệu toàn cục

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

VI. Tham khảo chéo yêu cầu

VII. Điều khoản kiểm thử

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

VIII. Đóng gói A. Các điều khoản đặc biệt B. Xem xét chuyển đổi
IX. Lưu ý đặc biệt
X. Phụ lục

Mục VI của đặc tả thiết kế chứa các tham khảo chéo về các yêu cầu. Mục đích của ma trận tham khảo chéo này là:

1. Thiết lập mọi yêu cầu được phần mềm thoả mãn.
2. Chỉ ra modul nào là chủ chốt cho việc cài đặt các yêu cầu xác định.

Mục yêu cầu	Modul A	Modul B	Modul C	...	Modul Z
Mục 3.1.1	v				v
Mục 3.1.2		v	v		
Mục 3.1.3		v			
...	...	...	...		...
Mục 3.1.n			v		v

Tham khảo chéo yêu cầu

Giai đoạn đầu tiên trong việc phát triển các kiểm thử được đưa vào trong mục VII của tài liệu thiết kế. Một khi cấu trúc và giao diện phần mềm đã được thiết lập thì chúng ta có thể phát triển các hướng dẫn để kiểm thử từng modul riêng, rồi kiểm thử tích hợp toàn bộ hệ thống.

Trong một số trường hợp, một đặc tả chi tiết về thủ tục kỹ nghệ phần mềm sẽ xuất hiện song song với thiết kế. Trong những trường hợp như vậy, mục này có thể xoá đi khỏi bản thiết kế.

Các ràng buộc thiết kế, như những giới hạn bộ nhớ vật lý, hay sự cần thiết một giao diện ngoài chuyên dụng có thể khống chế các yêu cầu riêng để lắp ráp hay đóng gói phần mềm. Những xem xét đặc biệt cần thiết cho sự chồng chất chương trình, quản lý bộ nhớ ảo, xử lý tốc độ

cao, hay các nhân tố khác, có thể gây ra sự thay đổi trong thiết kế suy ra từ luồng cấu trúc thông tin.

Các yêu cầu và xem xét cho việc đóng gói phần mềm được trình bày trong mục VIII, mô tả cho các tiếp cận sẽ được dùng để chuyển phần mềm cho khách hàng.

Mục IX và X của bản đặc tả thiết kế chứa các dữ liệu phụ trợ. Các mô tả thuật toán, thủ tục khác, dữ liệu bảng, các trích đoạn từ các văn bản khác và các thông tin liên quan được trình bày như những lưu ý đặc biệt hoặc như những phụ lục tách biệt. Chúng ta nên phát triển *Bản Hướng dẫn sơ bộ* hay *Tài liệu cài đặt* và đưa nó vào như phụ lục cho tài liệu thiết kế.

6. TÓM TẮT

Thiết kế là cái lõi của kỹ thuật kỹ nghệ phần mềm. Trong khi thiết kế người ta sẽ phát triển, xét duyệt và làm tư liệu cho việc làm mịn dần các chi tiết thủ tục, cấu trúc chương trình, cấu trúc dữ liệu. Việc thiết kế nảy sinh trong việc biểu diễn cho phần mềm và chất lượng phần mềm có thể được xác nhận.

Trong suốt ba thập kỷ qua, người ta đã đề nghị một số khái niệm thiết kế phần mềm nền tảng. Tính modul (trong cả chương trình và dữ liệu) và khái niệm trừu tượng làm cho người thiết kế có khả năng đơn giản hoá và dùng lại các thành phần phần mềm. Việc làm mịn đưa ra một cơ chế để biểu diễn các tầng kế tiếp của chi tiết chức năng. Cấu trúc chương trình và dữ liệu đóng góp một quan điểm tổng thể về kiến trúc phần mềm, trong khi thủ tục lại đưa ra những chi tiết cần thiết cho việc cài đặt thuật toán. Che dấu thông tin và độc lập chức năng là để đạt tới tính modue hiệu quả.

Thiết kế phần mềm có thể được xem xét hoặc theo cách nhìn kỹ thuật hoặc theo cách nhìn quản lý dự án. Theo quan điểm kỹ thuật, thiết kế bao gồm bốn hoạt động: thiết kế dữ liệu, thiết kế kiến trúc, thiết kế thủ tục và thiết kế giao diện. Theo quan điểm quản lý, thiết kế tiến hoá từ thiết kế sơ bộ sang thiết kế chi tiết.

Ký pháp thiết kế, đi kèm với các khái niệm lập trình có cấu trúc làm cho người thiết kế biểu diễn được chi tiết thủ tục theo cách thức làm thuận tiện cho việc dịch sang mã chương trình. Các ký pháp đồ hoạ, bảng và văn bản đều có sẵn.

Còn nhiều phương pháp thiết kế phần mềm quan trọng như thiết kế hướng luồng dữ liệu, hướng sự vật... Những phương pháp này được kết hợp với những nền tảng đã trình bày ở trên tạo nên một cách nhìn đầy đủ về thiết kế phần mềm.

7. CỦNG CỐ

1. Tầm quan trọng của thiết kế phần mềm? Các giai đoạn phải trải qua?
2. Chuẩn bị cho lập trình, những loại thiết kế nào cần tạo ra trong giai đoạn thiết kế? Vẽ sơ đồ hoạt động thiết kế và sản phẩm thiết kế? Mô tả các hoạt động cốt yếu trong đó.
3. Vẽ và phân tích sơ đồ mô tả mối quan hệ giữa các khía cạnh quản lý và kỹ thuật?
4. Tư tưởng của phương pháp cấu trúc?
5. Các hướng dẫn đảm bảo chất lượng thiết kế?
6. Các khái niệm nền tảng cho thiết kế (trừu tượng, làm mịn, modul, kiến trúc phần mềm, cấp bậc điều khiển, cấu trúc dữ liệu, thủ tục phần mềm, che dấu thông tin)?
7. Tóm lược hai chiến lược thiết kế chức năng và đối tượng?
8. Độ đo chất lượng thiết kế?
9. Bàn về thiết kế hướng đối tượng (cách tiếp cận, đặc trưng ưu nhược điểm)?
10. Khái niệm biểu đồ dòng dữ liệu, lược đồ cấu trúc, từ điển dữ liệu trong thiết kế hướng chức năng?
11. Các thế hệ giao diện người - máy? Tiến trình thiết kế giao diện?
12. Các Mô hình thiết kế giao diện?
13. Các vấn đề thiết kế giao diện thường nảy sinh?
14. Chu trình đánh giá thiết kế giao diện? Tiêu chuẩn đánh giá xét duyệt thiết kế?
15. Tóm lược các hướng dẫn thiết kế giao diện?