

# CHƯƠNG 1

## VÀI NÉT VỀ QUÁ TRÌNH PHÁT TRIỂN VÀ MỤC TIÊU CỦA CÔNG NGHỆ PHẦN MỀM

### 1.1. Quá trình hình thành và phát triển của kỹ thuật lập trình

Khoảng trước những năm 1950, tin học đang ở trong thời kỳ sơ khai. Ban đầu, người ta sử dụng từng lệnh riêng cho máy hoạt động, tiến đến việc xây dựng một hệ thống các lệnh tuân theo trình tự nhất định để giải quyết những bài toán hay một vấn đề nào đó - người ta gọi đó là các chương trình. Thời kỳ đầu, người ta xây dựng các chương trình này bằng ngôn ngữ cấp thấp: MinCK 22, MinCK 23, Algol, Fortran, ...

Những chương trình này không thể sửa ngay trực tiếp trên máy tính được mà phải mã hoá thành dạng nhị phân.

Tin học ngày càng phát triển, người ta luôn tìm cách cải tiến cả phần cứng lẫn phần mềm.

**Về phần cứng:** Kích thước phần cứng ngày càng giảm và dung lượng bộ nhớ ngày càng lớn. Tốc độ tăng, giá thành hạ.

**Về phần mềm:** Ngày được cải tiến phong phú hơn.

Cho đến những năm 1960, việc ứng dụng tin học vào thực tế ngày càng nhiều lên. Tuy vậy, để giải quyết những tính toán thực tế ngày càng sâu hơn thì chương trình đòi hỏi phải ngày một đồ sộ hơn. Chính vì thế, vào thời điểm này một loạt các chương trình khi đưa vào thực tế đều đã thất bại. Người ta tìm hiểu thấy có 3 nguyên nhân chính:

- ◆ *Chương trình là một khối lớn liên nhau lên khó theo dõi và chỉnh sửa.*
- ◆ *Các chương trình sử dụng quá nhiều lệnh GOTO.*
- ◆ *Các quy định về ngữ pháp lỏng lẻo, gây lên hiểu nhầm cho máy tính, ví dụ: tên của các biến trong ngôn ngữ Fortran cho phép có cả dấu cách; các biến không phải khai báo kiểu của chúng trước khi được sử dụng.*

Để khắc phục sự thiếu chặt chẽ của Fortran, người ta đưa ra ngôn ngữ Algol. Nhưng ngôn ngữ này lại có quy định quá rườm rà, rắc rối về cấu trúc ngữ pháp nên rất khó cài đặt hay cài đặt thiếu hiệu quả.

Đến thập kỷ 1970, người ta nghĩ đến việc làm một cuộc cách mạng về lập trình. Mục tiêu là xây dựng ngôn ngữ lập trình dễ tiếp cận, dễ cải tiến và phát triển; có thể khai thác được tối đa các khả năng của máy tính và không phụ thuộc vào bản thân máy tính; chương trình có thể phân chia thành nhiều khối lớn rồi ghép lại và quan trọng là

phải đảm bảo về mặt toán học. Và người ta đã đưa ra Ngôn ngữ lập trình C - là ngôn ngữ đáp ứng được điều kiện này, sau đó là Ngôn ngữ PASCAL, ...

Hiện nay, các công cụ lập trình đã được cải tiến hơn. Hỗ trợ nhiều cho người lập trình. Công cụ lập trình Basic của hãng Microsoft ra đời đánh dấu một bước mới trong việc lập trình, bộ phát triển phần mềm này được vận hành trên nền của hệ điều hành Windows. Có thể kể đến một số công cụ như: Các công cụ trong bộ Visual Studio ở các phiên bản 5.0, 6.0, 7.0 và .NET (Visual Basic, Visual C++, Visual FoxPro). Ngôn ngữ PHP, Delphi, Java,... Hệ quản trị cơ sở dữ liệu như: Microsoft Access, Visual FoxPro, MySQL, SQL Server và gần đây nhất là Oracle.

## 1.2. Khái niệm về công nghệ phần mềm.

*Công nghệ phần mềm* là một lĩnh vực nghiên cứu của tin học nhằm đưa ra các *nguyên lý, phương pháp, công cụ, phương tiện* giúp cho việc thiết kế và cài đặt một sản phẩm phần mềm đạt được các yêu cầu sau một cách tốt nhất:

- ✦ Phải có tính đúng đắn và khoa học.
- ✦ Dễ tiếp cận và cải tiến.
- ✦ Phổ dụng.
- ✦ Độc lập với các thiết bị.

## 1.3. Các giai đoạn để cho ra đời một sản phẩm phần mềm.

### \* *Tìm hiểu nhu cầu của khách hàng:*

Đây là giai đoạn đầu tiên và không thể thiếu được trong việc xây dựng phần mềm cho một hệ thống nào đó. Sản phẩm phần mềm mà nhóm phát triển tạo ra suy cho đến cùng thì phải đáp ứng được (không phải là toàn bộ) nhu cầu của khách hàng.

Nhu cầu của khách hàng có thể chia làm 3 cấp độ:

*Nhu cầu của người có quyền cao nhất* đối với hệ thống (giám đốc, chủ tịch,...): Đây là người đưa ra yêu cầu tổng quát nhất của hệ thống. Đó là kết quả mà hệ thống cần đạt được. Tuy nhiên, do ở mức quản lý cấp cao lên nhu cầu đưa ra mang tính khái quát, trừu tượng, không cụ thể. Điều này đòi hỏi nhà phát triển hệ thống cần phải tìm hiểu sâu hơn ở những người khác.

*Nhu cầu của người quản lý* (trưởng phòng, ...): Đây là người quản lý mức thấp hơn. Họ nắm bắt được yêu cầu tổng thể đồng thời họ cũng dễ tiếp cận với các công việc cụ thể hơn, quản lý việc thực hiện các quy trình nghiệp vụ trong hệ thống. Do vậy, yêu cầu họ đưa ra sẽ mang tính cụ thể hơn, phân cấp rõ ràng hơn.

*Nhu cầu của người dùng cấp thấp nhất* (nhân viên): Đây là người dùng cấp cuối cùng của hệ thống (end user). Yêu cầu họ đưa ra là rất cụ thể, chi tiết. Thể hiện rõ được công việc cần thực hiện. Tuy nhiên, yêu cầu mà họ đưa ra không mang tính hệ thống, khó phân loại. Do vậy đòi hỏi

nhà phát triển hệ thống phải biết thu thập rồi phân loại các yêu cầu để từ đó có thể hiểu được toàn bộ nhu cầu của tổ chức đó.

\* **Xác định rõ các chức năng hệ thống.** Chia ra từng khối lớn tương đối độc lập và giao cho từng nhóm người thực hiện. Mỗi nhóm người phải chịu trách nhiệm từ việc thiết kế - sản xuất - thử nghiệm theo một nguyên tắc nhất định và một ngôn ngữ cùng với cơ sở dữ liệu thống nhất. Sau đó ghép nối các khối thành khối lớn.

\* **Sửa chữa và thử nghiệm nếu thấy cần thiết.** Đây là giai đoạn mang tính nội bộ của nhóm phát triển phần mềm. Hệ thống có thể được chia thành nhiều phần nhỏ (module) rồi rạc nhau. Do vậy khi xây dựng xong chúng ta cần phải thử nghiệm cho từng module đó. Sau đó tiến hành tích hợp các module lại để tạo thành hệ thống hoàn chỉnh. Việc kiểm thử tích hợp phải được tiến hành. Các thay đổi có thể được thêm vào; các ý kiến đóng góp của khách hàng cũng được ghi nhận và đưa vào trong phần mềm tại giai đoạn cuối cùng này.

\* **Bàn giao sản phẩm cho khách hàng,** tìm hiểu ý kiến của khách hàng để quyết định nhân bản nếu nó tốt hoặc là để sửa đổi. Đào tạo người sử dụng.

Trong quá trình từ khi tìm hiểu nhu cầu của khách hàng cho đến khi hoàn thiện, trong thời kỳ trước kia, trung bình mỗi người trong một ngày chỉ làm được 5 hoặc 6 lệnh. Khi đó có thể nói “Lập trình phần mềm hết sức nặng nhọc”. Chính vì vậy người ta phải cố gắng sử dụng những chương trình con (modul) chương trình của những người đi trước tạo ra (thường để trong thư viện) và đồng thời người ta cũng tạo ra các modul thêm vào thư viện để người khai thác có thể dùng.

Theo quan điểm hiện nay, các công cụ lập trình đã hỗ trợ rất lớn cho lập trình viên. Lập trình không còn là một công việc nặng nhọc nữa. Trái lại, người lập trình lại là người có vai trò cuối cùng trong quá trình sản xuất phần mềm. Quan trọng nhất bây giờ hiện tại là nắm bắt và phân tích yêu cầu của khách hàng. Do vậy người phân tích và thiết kế hệ thống là người đóng vai trò quyết định đối với toàn bộ hệ thống.

## 1.4. Nội dung cơ bản của công nghệ phần mềm.

\* **Phải tìm hiểu và phân tích các yêu cầu của bài toán hoặc đề tài,** thu thập đầy đủ các thông tin và phân tích các thông tin theo mọi khía cạnh cả chiều rộng lẫn chiều sâu.

\* **Đặc tả (hay mô tả) chương trình:** Tại mỗi nút của chương trình, người ta chỉ quan tâm đến đầu vào và đầu ra, còn cấu trúc và nội dung của các thao tác trong chương trình thì người ta không quan tâm. Các đặc tả này có thể sử dụng những mô hình toán học để đặc tả một cách hình thức, hoặc dùng ngôn ngữ thông thường để đặc tả phi hình thức hoặc có thể kết hợp cả hai dạng để đặc tả hỗn hợp. Việc nghiên cứu về đặc tả sẽ được đề cập đến trong chương sau.

\* **Thiết kế chương trình** bằng phương pháp lập trình có cấu trúc và phải kiểm thử chương trình bằng cách cho nhiều bộ dữ liệu khác nhau để kiểm tra chương trình xem còn lỗi hay không. Đồng thời kiểm tra tính ổn định, độ phức tạp theo thời gian, chi phí của miền nhớ và khả năng tối đa của chương trình.

\* **Phải chứng minh được tính đúng đắn của chương trình** về mặt toán học và phù hợp đối với cơ sở đó.

\* **Phản biện tính đúng đắn của chương trình** (những người khác xét duyệt).

\* **Tiến hành cài đặt, sử dụng bảo trì**, đồng thời phải cung cấp cho người dùng những phần mềm hỗ trợ cho hệ thống chương trình đang được sử dụng.

## 1.5. Một số mô hình cơ bản của công nghệ phần mềm

### 1.5.1. Khái niệm “Phần mềm”.

Hai mươi lăm năm trước đây (vào những năm 1975), ít hơn một phần trăm công luận có thể mô tả một cách thông minh “phần mềm máy tính” nghĩa là gì. Ngày nay, hầu hết các nhà chuyên môn và nhiều người trong đa số công luận cảm thấy rằng họ hiểu được phần mềm. Nhưng điều đó có thật không?

Mô tả về phần mềm trong sách giáo khoa có thể có dạng sau: “*Phần mềm là một tập hợp bao gồm:*

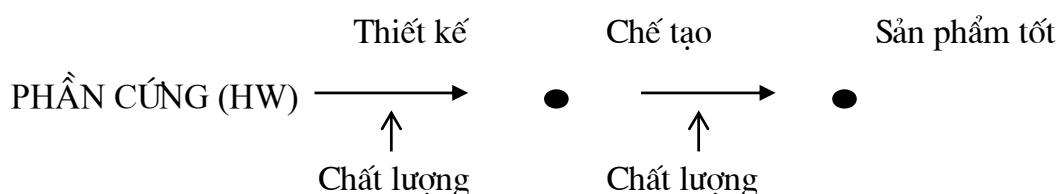
- 1 • *Các lệnh (chương trình máy tính) khi thực hiện thì đưa ra hoạt động và kết quả mong muốn.*
- 2 • *Các cấu trúc dữ liệu làm cho chương trình thao tác thông tin thích hợp.*
- 3 • *Các tài liệu mô tả thao tác và cách dùng chương trình.*

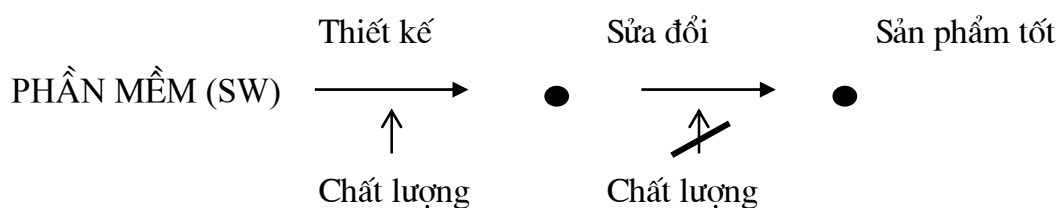
Mô tả như vậy thì không có vấn đề cần phải đưa ra các định nghĩa khác đầy đủ hơn. Nhưng ta cần một định nghĩa mang tính hình thức nhiều hơn.

#### **Các đặc trưng của phần mềm:**

Phần mềm là phần tử của hệ thống logic chưa không phải hệ thống vật lý. Do vậy, phần mềm có một số đặc trưng khác biệt đáng kể đối với đặc trưng của phần cứng.

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

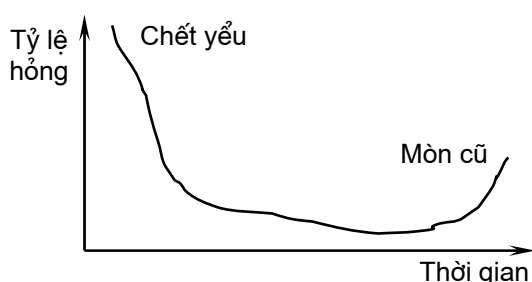




Mặc dầu có một số điểm tương đồng giữa phát triển phần mềm và chế tạo phần cứng, hai hoạt động này về cơ bản là khác nhau. Trong cả hai hoạt động này, chất lượng cao được đạt tới thông qua thiết kế tốt, nhưng giai đoạn chế tạo phần cứng có thể đưa vào vấn đề mà chất lượng không tồn tại (hay dễ được sửa đổi) cho phần mềm. Cả hai hoạt động này đều phụ thuộc vào con người, nhưng mối quan hệ giữa người được áp dụng và công việc được thực hiện hoàn toàn khác. Cả hai hoạt động này đòi hỏi việc xây dựng "sản phẩm", nhưng cách tiếp cận là hoàn toàn khác. Phần mềm được chế tạo ra là hoàn toàn mới, không có tiền lệ trước và nó cũng chỉ được tạo ra 1 lần duy nhất.

### **Đặc trưng 2: Phần mềm không “hỏng đi”.**

Phần mềm không cảm ứng với khiếm khuyết môi trường vốn gây cho phần cứng mòn cũ đi. Phần mềm nếu cứ với các bộ dữ liệu đầu vào hợp lý thì nó luôn cho kết quả có ý nghĩa giống nhau, không thay đổi theo thời gian, điều kiện khí hậu, ...

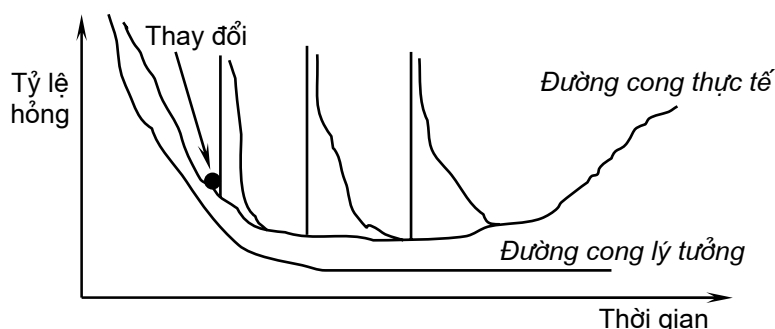


Đường cong hỏng hóc của phần cứng



Đường cong hỏng hóc của phần mềm (lý tưởng)

Thực tế, phần mềm sẽ trải qua sự thay đổi (bảo trì). Khi thay đổi được thực hiện, có thể một số khiếm khuyết sẽ được thêm vào, gây ra trong đường cong tỷ lệ hỏng có dấu hiệu như hình vẽ dưới đây. Trước khi đường cong đó có thể trở về tỷ lệ hỏng hóc ổn định ban đầu, thì một yêu cầu khác lại được đưa vào, lại gây ra đường cong phát sinh đỉnh nhọn một lần nữa. Dần dần, mức tỷ lệ hỏng tối thiểu tăng lên - phần mềm bị thoái hoá do sự thay đổi.



Hình 1: Đường cong hỏng hóc thực tế của phần mềm

**Nhận xét:** Phần cứng hồng có “vật tư thay thế”, nhưng không có phần mềm thay thế cho phần mềm. Mọi hồng học của phần mềm đều chỉ ra lỗi trong thiết kế hay trong tiến trình chuyển thiết kế thành mã hoá lệnh máy thực hiện được. Do đó, việc bảo trì phần mềm bao gồm việc phụ thêm đáng kể so với bảo trì phần cứng.

**Đặc trưng 3:** Phần lớn phần mềm được xây dựng theo đơn đặt hàng, chứ ít khi được lắp ráp từ các thành phần có sẵn.

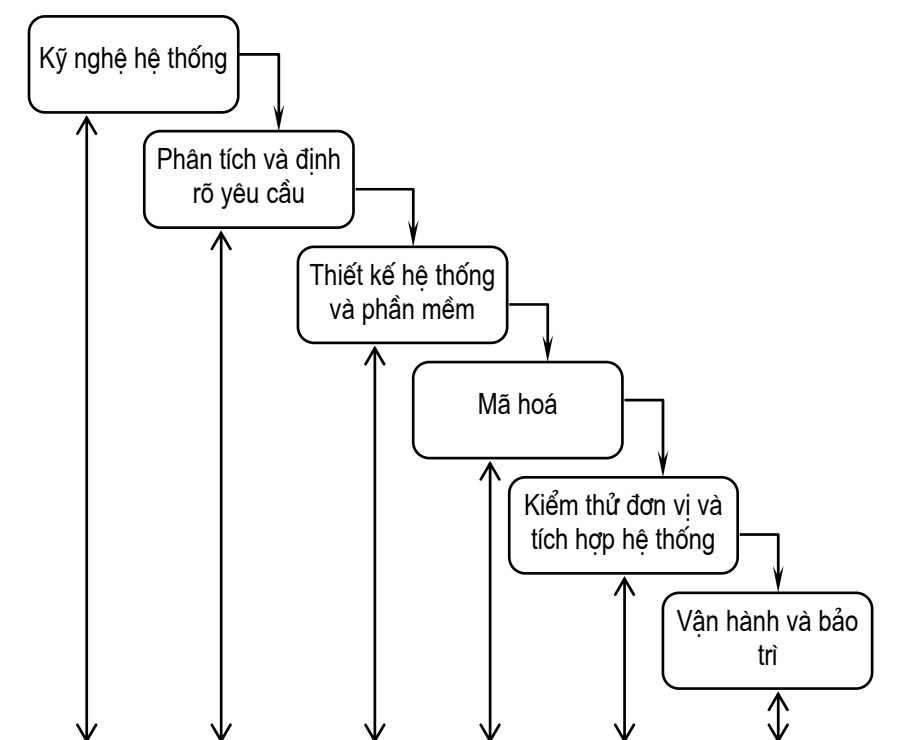
Cách thiết kế và xây dựng **phần cứng** điều khiển cho một sản phẩm dựa trên bộ vi xử lý: vẽ sơ đồ mạch số => thực hiện phân tích để đảm bảo chức năng đúng => phân loại các danh mục thành phần => gắn cho mỗi mạch tích hợp (thường gọi là IC hay chip) một số hiệu một chức năng đã định trước và hợp lệ;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

Đối với **phần mềm**: Khi xây dựng ta không có danh mục các thành phần. Phần mềm được đặt hàng với đơn vị hoàn chỉnh, không phải là những thành phần có thể lắp ráp lại thành chương trình mới.

Sau đây ta sẽ xem xét một số mô hình cơ bản hay được ứng dụng trong thực tế.

### 1.5.2. Mô hình "thác nước" (hay mô hình "vòng đời cổ điển").

Đôi khi còn được gọi là *mô hình tuần tự tuyến tính* hay *mô hình thác nước*, mô hình này gợi ý một cách tiếp cận tuần tự, có hệ thống tới việc phát triển phần mềm vốn bắt đầu từ mức hệ thống và tiến dần qua phân tích, thiết kế, mã hoá, kiểm thử và hỗ trợ. Dưới đây minh hoạ mô hình thác nước cho kĩ nghệ phần mềm. Được mô hình hoá theo chu kì kĩ nghệ qui ước, mô hình thác nước bao gồm các hoạt động sau:



**Hình 2: Mô hình thác nước**

**Kỹ nghệ và mô hình hoá hệ thống / thông tin.** Bởi vì phần mềm bao giờ cũng là một phần của một hệ thống (hay nghiệp vụ) lớn hơn nên công việc bắt đầu từ việc thiết lập yêu cầu cho mọi phần tử hệ thống và rồi cấp phát một tập con các yêu cầu đó cho phần mềm. Quan điểm hệ thống này là điều bản chất khi phần mềm phải tương tác với các thành phần khác như phần cứng, con người và CSDL. 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. Kỹ nghệ thông tin bao gồm việc thu thập yêu cầu tại mức nghiệp vụ chiến lược và tại mức lĩnh vực nghiệp vụ.

**Phân tích yêu cầu phần mềm.** Tiến trình thu thập yêu cầu được tăng cường và hội tụ đặc biệt vào phần mềm. Để hiểu được bản chất của các chương trình phải xây dựng, kỹ sư phần mềm ("nhà phân tích") phải hiểu về lĩnh vực thông tin (được mô tả trong phần sau) đối với phần mềm cũng như chức năng cần có, hành vi, hiệu năng và giao diện. Các yêu cầu cho cả hệ thống và phần mềm cần phải được lập tư liệu và xét duyệt cùng với khách hàng.

**Thiết kế.** Thiết kế phần mềm thực tế 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, biểu diễn giao diện và chi tiết thủ tục (thuật toán). Tiến trình thiết kế dịch các yêu cầu thành một biểu diễn của phần mềm có thể được định giá về chất lượng trước khi giai đoạn mã hoá bắt đầu. Giống như các yêu cầu, việc thiết kế phải được lập tư liệu và trở thành một phần của cấu hình phần mềm.

**Sinh mã.** Thiết kế phải được dịch thành dạng máy đọc được. Bước mã hoá thực hiện nhiệm vụ này. Nếu thiết kế được thực hiện theo một cách chi tiết thì việc sinh mã có thể được thực hiện một cách máy móc.

**Kiểm thử.** Một khi mã đã được sinh ra thì việc kiểm thử chương trình bắt đầu. Tiến trình kiểm thử hội tụ vào nội bộ logic của phần mềm, đảm bảo rằng tất cả các câu lệnh đều được kiểm thử, và vào bên ngoài chức năng; tức là tiến hành các kiểm thử để làm lộ ra các lỗi và đảm bảo những cái vào đã định sẽ tạo ra kết quả thống nhất với kết quả muốn có.

**Vận hành và bảo trì.** Phần mềm chắc chắn sẽ phải trải qua những thay đổi sau khi nó được bàn giao cho khách hàng (một ngoại lệ có thể là những phần mềm nhúng). Thay đổi sẽ xuất hiện bởi vì gặp phải lỗi, bởi vì phần mềm phải thích ứng với những thay đổi trong môi trường bên ngoài (chẳng hạn như sự thay đổi do hệ điều hành mới hay thiết bị ngoại vi mới), hay bởi vì 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 cho chương trình hiện tại chứ không phải chương trình mới.

Mô hình tuần tự tuyến tính là mô hình cũ nhất và được sử dụng rộng rãi nhất cho kỹ nghệ phần mềm. Tuy nhiên, những chỉ trích về mô hình này đã làm cho những người

ủng hộ nó tích cực phải đặt vấn đề về tính hiệu quả của nó. Một số các vấn đề thỉnh thoảng gặp phải khi dùng mô hình tuần tự tuyến tính này là:

1. Các dự án thực hiếm khi tuân theo dòng chảy tuần tự mà mô hình đề nghị. Mặc dầu mô hình tuyến tính có thể cho phép lặp, nhưng điều đó chỉ làm gián tiếp. Kết quả là những thay đổi có thể gây ra lẫn lộn khi tổ dự án tiến hành.
2. Khách hàng thường khó phát biểu mọi yêu cầu một cách tường minh. Mô hình tuần tự tuyến tính đòi hỏi điều này và thường khó thích hợp với sự bất trắc tự nhiên tồn tại vào lúc đầu của nhiều dự án.
3. Khách hàng phải kiên nhẫn. Bản làm việc được của chương trình chỉ có được vào lúc cuối của thời gian dự án. Một sai lầm ngớ ngẩn, nếu đến khi có chương trình làm việc mới phát hiện ra, có thể sẽ là một thảm họa.

Trong một phân tích thú vị về các dự án hiện tại, Brada thấy rằng bản chất tuyến tính của vòng đời cổ điển dẫn tới "các trạng thái nghẽn" mà trong đó một số thành viên tổ dự án phải đợi cho các thành viên khác của tổ hoàn thành các nhiệm vụ phụ thuộc. Trong thực tế, thời gian mất cho việc chờ đợi có thể vượt quá thời gian dành cho công việc sản xuất. Trạng thái nghẽn có khuynh hướng phổ biến vào lúc đầu và cuối của tiến trình tuần tự tuyến tính.

Từng vấn đề trên đều là thực. Tuy nhiên, mô hình 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 tiêu bản trong đó có thể bố trí các phương pháp cho phân tích, thiết kế, mã hoá, kiểm thử và bảo trì. Bên cạnh đó, vòng đời cổ điển vẫn còn là một mô hình thủ tục được dùng rộng rãi cho kỹ nghệ phần mềm. Trong khi nó quả thực còn điểm yếu, nó vẫn tốt hơn đáng kể nếu so với cách tiếp cận ngẫu nhiên tới việc phát triển phần mềm.

### 1.5.3. Mô hình "xoắn ốc" (hay mô hình thăm dò).

*Mô hình xoắn ốc*, ban đầu do Boehm đề xuất, là mô hình tiến trình phần mềm tiến hoá vốn cặp đôi bản chất lặp của làm bản mẫu với các khía cạnh hệ thống và có kiểm soát của mô hình trình tự tuyến tính. Nó cung cấp tiềm năng cho việc phát triển nhanh các phiên bản tăng dần của phần mềm. Dùng mô hình xoắn ốc này, phần mềm được phát triển thành từng chuỗi các lần đưa ra tăng dần. Trong những lần lặp đầu, việc đưa ra tăng dần có thể là mô hình trên giấy hay bản mẫu. Trong các lần lặp sau, các phiên bản đầy đủ tăng dần của hệ thống được kỹ nghệ hoá sẽ được tạo ra.

Mô hình xoắn ốc được chia thành một số khuôn khổ hoạt động, cũng còn được gọi là *vùng nhiệm vụ*. Về cơ bản, có từ ba tới sáu vùng. Hình sau mô tả cho mô hình xoắn ốc có chứa sáu vùng:

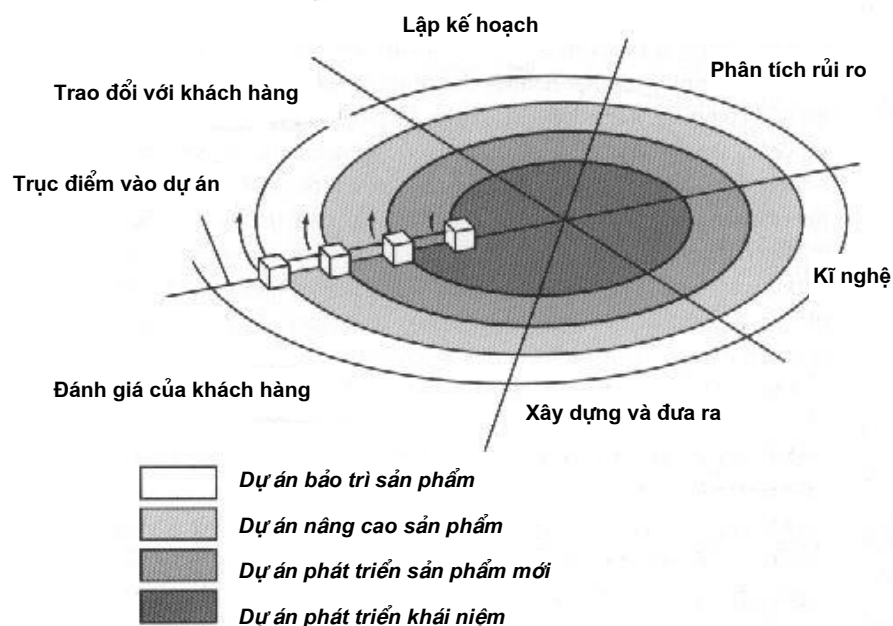
1. **Trao đổi với khách hàng** - nhiệm vụ đòi hỏi thiết lập việc trao đổi có hiệu quả giữa người phát triển và khách hàng.



2. **Lập kế hoạch** - nhiệm vụ đòi hỏi định nghĩa các tài nguyên, hạn thời gian và các thông tin liên quan tới dự án.
3. **Phân tích rủi ro** - nhiệm vụ đòi hỏi định giá cả những rủi ro kỹ thuật và quản lý
4. **Kỹ nghệ** - nhiệm vụ đòi hỏi xây dựng một hay nhiều biểu diễn cho ứng dụng
5. **Xây dựng và đưa ra** - nhiệm vụ đòi hỏi xây dựng, kiểm thử, thiết đặt và cung cấp sự hỗ trợ cho người dùng (như tài liệu và huấn luyện)
6. **Đánh giá của khách hàng** - nhiệm vụ đòi hỏi thu được phản hồi của khách hàng dựa trên đánh giá về biểu diễn phần mềm được tạo ra trong giai đoạn kỹ nghệ và được cài đặt trong giai đoạn cài đặt.

Mỗi một trong các vùng đều được đặt vào một tập các nhiệm vụ, được gọi là *tập nhiệm vụ*, vốn được thích ứng với các đặc trưng của dự án được tiến hành. Với các dự án nhỏ, số các nhiệm vụ công việc và tính hình thức của chúng là thấp. Với các dự án lớn, nhiều căng thẳng hơn, thì mỗi vùng nhiệm vụ lại chứa nhiều nhiệm vụ công việc vốn được xác định để đạt tới mức độ hình thức cao hơn. Trong mọi trường hợp, hoạt động hỗ trợ (như quản lý cấu hình phần mềm và đảm bảo chất lượng phần mềm) - được nêu trong phần sau - sẽ được áp dụng.

Khi tiến trình tiến hoá này bắt đầu, tổ kỹ nghệ phần mềm đi vòng xoắn ốc theo chiều ngược kim đồng hồ, bắt đầu từ trung tâm. Mạch đầu tiên quanh xoắn ốc có thể làm phát sinh việc phát triển đặc tả sản phẩm; các bước tiếp theo quanh xoắn ốc có thể được dùng để phát triển bản mẫu và thế rồi các phiên bản phức tạp dần thêm. Mỗi bước qua vùng lập kế hoạch lại làm nảy sinh việc điều chỉnh kế hoạch dự án. Chi phí và lịch biểu được điều chỉnh dựa trên phản hồi được suy từ đánh giá của khách hàng. Bên cạnh đó, người quản lý dự án điều chỉnh số việc lập đã lập kế hoạch cần để hoàn chỉnh phần mềm.



Không giống như mô hình tiến trình cổ điển vốn kết thúc khi phần mềm được chuyển giao, mô hình xoắn ốc có thể được thích ứng để áp dụng trong toàn bộ cuộc đời của phần mềm máy tính. Một cái nhìn khác có thể được xem xét bằng việc *kiểm tra trực điểm vào dự án*, như được vẽ trong hình trên. Mỗi hình hộp được đặt theo trục có thể được dùng để biểu diễn cho điểm bắt đầu cho các kiểu dự án khác nhau. "Dự án phát triển khái niệm" bắt đầu tại cốt lõi của xoắn ốc và sẽ tiếp tục (nhiều lần lặp xuất hiện theo con đường xoắn ốc mà vốn gắn với vùng tô đậm trung tâm) cho tới khi việc phát triển khái niệm là đầy đủ. Nếu khái niệm này được phát triển thành một sản phẩm thực tại, thì tiến trình tiến qua hình hộp tiếp (điểm vào dự án phát triển sản phẩm mới) và một "dự án phát triển mới" được khởi đầu. Sản phẩm mới sẽ tiến hoá qua một số lần lặp quanh xoắn ốc, đi theo con đường vốn gắn vùng có tô màu sáng hơn vùng lõi. Về bản chất, xoắn ốc, khi được đặc trưng theo cách này, vẫn còn làm việc cho tới khi phần mềm được cho nghỉ. Có những lúc tiến trình này "ngủ", nhưng bất kì khi nào một thay đổi được khởi đầu, thì tiến trình này lại bắt đầu tại điểm vào thích hợp (tức là nâng cao sản phẩm).

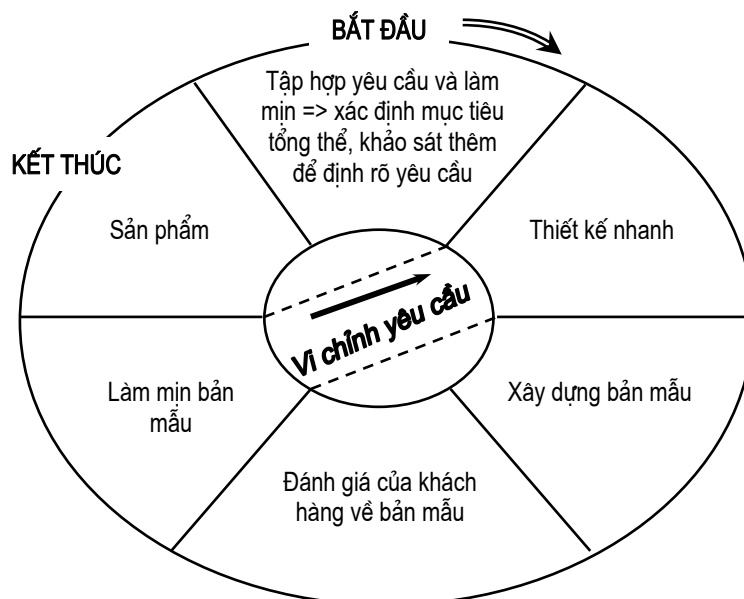
Mô hình xoắn ốc là cách tiếp cận thực tế cho việc phát triển cho các hệ thống và phần mềm qui mô lớn. Bởi vì phần mềm tiến hoá khi tiến trình tiến hoá, nên người phát triển và khách hàng hiểu rõ hơn và phản ứng với rủi ro tại từng mức tiến hoá. Mô hình xoắn ốc dùng cách làm bản mẫu như một cơ chế làm giảm bớt rủi ro, nhưng điều quan trọng hơn, làm cho người phát triển có khả năng áp dụng được cách tiếp cận làm bản mẫu tại mọi giai đoạn trong tiến hoá của sản phẩm. Nó duy trì cách tiếp cận từng bước một cách có hệ thống do cách tiếp cận vòng đời cổ điển gợi ý, nhưng tổ hợp cách tiếp cận này vào một khuôn khổ lặp lại, vốn phản ánh được sát thực hơn thế giới thực. Mô hình xoắn ốc đòi hỏi việc xem xét trực tiếp các rủi ro kĩ thuật tại mọi giai đoạn của dự án, và nếu được áp dụng đúng thì nó có thể làm giảm rủi ro trước khi chúng trở thành vấn đề thực sự.

Nhưng giống như các mô hình khác, mô hình xoắn ốc không phải là một liều thuốc bách bệnh. Có thể khó thuyết phục những khách hàng (đặc biệt trong tình huống có hợp đồng) rằng cách tiếp cận tiến hoá là kiểm soát được. Nó đòi hỏi tri thức chuyên gia đánh giá rủi ro chính xác và dựa trên tri thức chuyên gia này mà đạt được thành công. Nếu một rủi ro chính không được phát hiện và quản lí thì không nghi ngờ gì nữa vấn đề sẽ xuất hiện. Cuối cùng, chính bản thân mô hình này cũng còn chưa được sử dụng rộng rãi như mô hình trình tự tuyến tính hoặc làm bản mẫu. Cần phải có thêm một số năm nữa trước khi tính hiệu quả của mô hình quan trọng này có thể được xác định với sự chắc chắn hoàn toàn.

#### **1.5.4. Mô hình tạo bản mẫu.**

Thông thường khách hàng đã xác định một tập các mục tiêu tổng quát cho phần mềm, nhưng còn chưa định danh các yêu cầu cái vào chi tiết, hay xử lí cái ra. Trong các trường hợp khác, người phát triển có thể không chắc về tính hiệu quả của thuật

toán, việc thích nghi hệ điều hành hay dạng giao diện người máy cần có. Trong những trường hợp này và nhiều trường hợp khác *mô hình làm bản mẫu* có thể đưa ra cách tiếp cận tốt nhất.



Mô hình làm bản mẫu (hình dưới) bắt đầu với việc *thu thập yêu cầu*. Người phát triển và khách hàng gặp nhau và *xác định các mục tiêu tổng thể* cho phần mềm, xác định các yêu cầu nào đã biết, và miền nào bắt buộc phải xác định thêm. Rồi đến việc "*thiết kế nhanh*". Thiết kế nhanh tập trung vào việc biểu diễn các khía cạnh của phần mềm thấy được đối với người dùng (như cách đưa vào và định dạng đưa ra). Thiết kế nhanh dẫn tới việc xây dựng một bản mẫu. Bản mẫu được khách hàng / 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 trong khi đồng thời lại làm cho người phát triển hiểu được kỹ hơn cần phải thực hiện nhu cầu nào.

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

Nhưng chúng ta nghĩ về bản mẫu thế nào khi nó được dùng cho mục đích được nêu trên? Brook đã nêu ra câu trả lời:

Trong hầu hết các dự án, hệ thống đầu tiên hiếm khi sử dụng được. Nó có thể là quá chậm, quá lớn, công kênh trong sử dụng hay tất cả những nhược điểm này. Không có cách nào khác là bắt đầu lại, đau đớn nhưng tinh khôn hơn, và xây dựng một phiên bản được thiết kế lại trong đó những vấn đề này đã được giải quyết... Khi một khái niệm hệ thống mới hay một kỹ nghệ mới được dùng, người ta phải xây dựng một hệ thống để rồi vứt đi, cho dù việc lập kế hoạch được thực hiện chu đáo nhất thì nó cũng không thể bao quát hết để chạy đúng được ngay lần đầu. Do đó câu hỏi quản lí không phải là liệu chúng ta có nên xây dựng một hệ thống thử

nghiệm và rồi vứt nó đi hay không. Bạn sẽ làm như vậy. Câu hỏi duy nhất là liệu nên lập kế hoạch trước để xây dựng một cái vứt đi hay để hứa hẹn bàn giao cái vứt đi đó cho khách hàng...

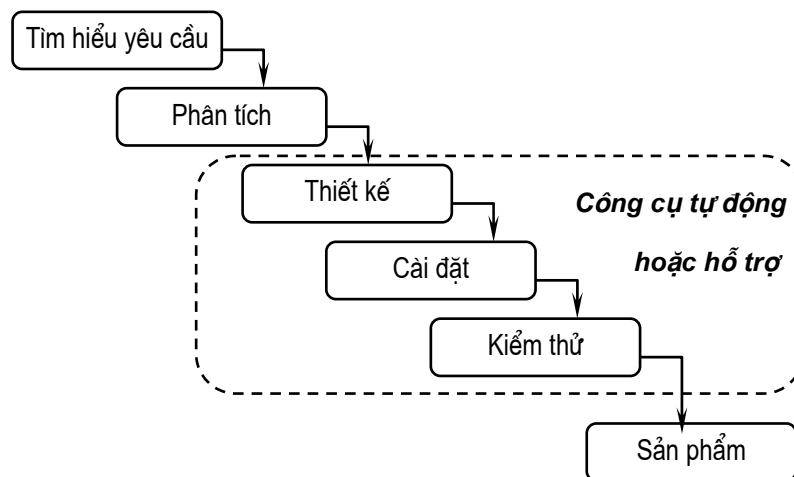
Bản mẫu có thể phục vụ như "hệ đầu tiên" - cái mà Brook lưu ý chúng ta nên vứt đi. Nhưng điều này có thể là một cách nhìn lí tưởng hoá. Giống như mô hình tuyến tính tuần tự (thác nước), việc làm bản mẫu tựa như một mô hình cho kĩ nghệ phần mềm có thể trở thành có vấn đề bởi những lí do sau:

1. Khách hàng thấy được cái dường như là phiên bản làm việc của phần mềm mà không biết rằng bản mẫu được gắn lại "bằng kẹo cao su và dây gói hàng", không biết rằng trong khi xô đẩy để cho nó làm việc thì chẳng ai xem xét tới chất lượng phần mềm tổng thể hay tính bảo trì thời gian dài. Khi được thông báo rằng sản phẩm phải được xây dựng lại để cho có thể đạt tới mức độ chất lượng cao, thì khách hàng kêu trời và đòi hỏi rằng "phải ít sửa chữa" để làm bản mẫu thành sản phẩm làm việc. Rất thường là việc quản lí phát triển phần mềm bị buông lỏng.
2. Người phát triển thường hay thoả hiệp cài đặt để có được bản mẫu làm việc nhanh chóng. Hệ điều hành hay ngôn ngữ lập trình không thích hợp có thể được dùng đơn giản bởi vì nó có sẵn và đã biết; một thuật toán không hiệu quả có thể được cài đặt đơn giản để chứng tỏ khả năng. Sau một thời gian, người phát triển mới có thể trở nên quen thuộc với những chọn lựa này và quên mất mọi lí do tại sao chúng lại không thích hợp. Việc chọn lựa không được theo lí tưởng bây giờ lại trở thành một phần tích hợp của hệ thống.

Mặc dầu vấn đề có thể xuất hiện, việc làm bản mẫu có thể là một mô hình hiệu quả cho kĩ nghệ phần mềm. Chìa khoá là định nghĩa ra các qui tắc của trò chơi từ ngay lúc bắt đầu; tức là khách hàng và người phát triển phải cùng đồng ý rằng bản mẫu được xây dựng để phục vụ làm cơ chế xác định yêu cầu. Thế rồi nó phải bị bỏ đi (ít nhất cũng một phần) và phần mềm thực tại được đưa vào kĩ nghệ với con mắt hướng về chất lượng và tính bảo trì được.

#### **1.5.5. Kĩ nghệ thế hệ thứ 4 (4GT).**

Thuật ngữ *kĩ thuật thế hệ thứ tư* (4GT) bao gồm một phạm vi rộng các công cụ phần mềm có một điểm chung: mỗi công cụ đều cho phép người kĩ sư phần mềm xác định đặc trưng nào đó của phần mềm ở mức cao. Rồi công cụ đó tự động sinh ra mã chương trình gốc dựa trên đặc tả của người phát triển. Người ta gần như không còn bàn cãi về việc phần mềm có thể được xác định đối với một máy cày ở mức cao thì chương trình có thể được xây dựng càng nhanh hơn. Mô hình 4GT cho kĩ nghệ phần mềm tập trung vào khả năng xác định phần mềm bằng việc dùng các khuôn mẫu ngôn ngữ đặc biệt hay kí pháp đồ hoạ vốn mô tả cho vấn đề cần được giải quyết dưới dạng khách hàng có thể hiểu được.



**Hình 3: Mô hình kỹ nghệ thứ 4 - 4GT**

Hiện tại, một môi trường phát triển phần mềm hỗ trợ cho mô hình 4GT bao gồm một số hay tất cả các công cụ sau:

- Ngôn ngữ phi thủ tục để hỏi đáp cơ sở dữ liệu.
- 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 và việc sinh tự động HTML.
- Các ngôn ngữ tương tự được dùng cho việc tạo ra trang Web thông qua việc dùng các công cụ phần mềm tiên tiến.

Ban đầu nhiều trong những công cụ đã được nhắc tới đó đã có sẵn chỉ cho những lĩnh vực ứng dụng rất đặc thù, nhưng ngày nay môi trường 4GT đã được mở rộng để đề cập tới hầu hết các loại ứng dụng phần mềm.

Giống như các mô hình khác, 4GT bắt đầu từ bước thu thập yêu cầu. Một cách lý tưởng, khách hàng sẽ mô tả các yêu cầu và các yêu cầu đó sẽ được dịch trực tiếp thành một bản mẫu vận hành được. Nhưng điều này không thực hiện được. Khách hàng có thể không chắc chắn mình cần gì, có thể có sự mơ hồ trong việc xác định các sự kiện đã biết, có thể không có khả năng hay không sẵn lòng xác định thông tin theo cách thức mà công cụ 4GT có thể giải quyết được. Bởi lí do này, đối thoại khách hàng/người phát triển được mô tả cho các mô hình tiến trình khác vẫn còn là phần bản chất của cách tiếp cận 4GT.

Với những ứng dụng nhỏ, có thể chuyển trực tiếp từ bước thu thập yêu cầu sang cài đặt bằng cách dùng *ngôn ngữ sinh thể hệ thứ tư phi thủ tục* (4GL) hay một mô hình bao gồm một mạng các biểu tượng đồ hoạ. Tuy nhiên với nỗ lực lớn hơn, cần phải phát triển một chiến lược thiết kế cho hệ thống, ngay cả nếu có dùng 4GL. Việc dùng 4GT thiếu thiết kế (với các dự án lớn) sẽ gây ra cùng những khó khăn (chất lượng kém, khó bảo trì, người dùng khó chấp nhận) mà chúng ta đã gặp phải khi phát triển phần mềm bằng cách dùng các cách tiếp cận qui ước.

Việc cài đặt dùng 4GL làm cho người phát triển phần mềm biểu diễn được các kết quả mong muốn theo cách là phát sinh tự động chương trình tính ra chúng. Hiển nhiên, một cấu trúc dữ liệu với những thông tin có liên quan cần phải có sẵn và sẵn sàng cho 4GL truy nhập vào.

Để biến đổi một cài đặt 4GT thành một sản phẩm, người phát triển phải tiến hành việc kiểm thử toàn diện, xây dựng các tài liệu có ý nghĩa và thực hiện mọi hoạt động tích hợp giải pháp khác vốn cần tới trong các mô hình kỹ nghệ phần mềm khác. Bên cạnh đó, phần mềm được phát triển theo 4GT phải được xây dựng theo cách làm cho việc bảo trì có thể được tiến hành một cách chóng vánh.

Giống như mọi mô hình kỹ nghệ phần mềm, mô hình 4GT có ưu điểm và nhược điểm. Những người ủng hộ cho là làm giảm đáng kể thời gian phát triển phần mềm và làm tăng rất nhiều hiệu suất của người xây dựng phần mềm. Những người phản đối cho là các công cụ 4GT hiện tại không phải tất cả đều dễ dùng hơn các ngôn ngữ lập trình, rằng chương trình gốc do các công cụ này tạo ra là "không hiệu quả," và rằng tính bảo trì cho các hệ thống phần mềm lớn được phát triển bằng cách dùng 4GT vẫn còn là vấn đề mở.

Có đôi điều lợi ích trong các luận điểm của cả hai phía và có thể tóm tắt trạng thái hiện tại của cách tiếp cận 4GT như sau:

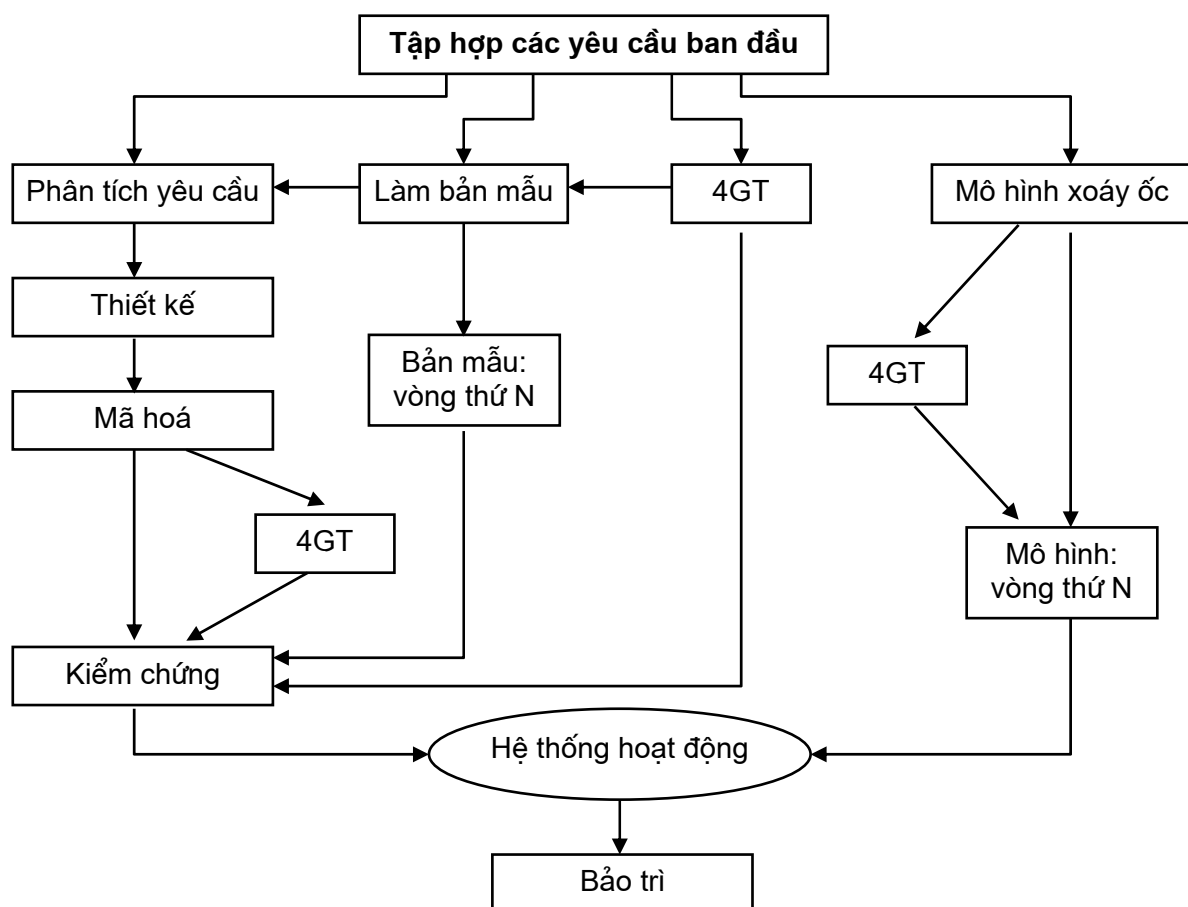
1. Việc dùng 4GT là cách tiếp cận có thể tồn tại được cho nhiều lĩnh vực ứng dụng khác nhau. Gắn với các công cụ kỹ nghệ phần mềm có máy tính hỗ trợ và bộ sinh mã, 4GT cung cấp một giải pháp tin cậy được cho nhiều vấn đề phần mềm.
2. Dữ liệu được thu thập từ các công ty có dùng 4GT chỉ ra rằng thời gian cần cho việc tạo ra phần mềm được giảm đáng kể đối với các ứng dụng vừa và nhỏ và rằng khối lượng thiết kế và phân tích cho các ứng dụng nhỏ cũng được rút bớt.
3. Tuy nhiên, việc dùng 4GT cho các nỗ lực phát triển phần mềm lớn đòi hỏi nhiều phân tích, thiết kế và kiểm thử (các hoạt động kỹ nghệ phần mềm) để đạt tới việc tiết kiệm thời gian vốn nảy sinh từ việc xoá bỏ mã hoá.

Tóm lại, các kỹ thuật thể hệ thứ tư đã trở thành một phần quan trọng của kỹ nghệ phần mềm. Khi đi đôi với cách tiếp cận dựa trên cấu phần (sẽ được trình bày ở mục

tiếp theo), mô hình 4GT có thể trở thành cách tiếp cận thống trị cho việc phát triển phần mềm.

### 1.5.6. Tổ hợp các khuôn cảnh.

Tổ hợp các khuôn cảnh kỹ nghệ phần mềm được thảo luận trong các mục trước thường được mô tả như là các cách tiếp cận khác nhau tới kỹ nghệ phần mềm chứ không phải là các 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 được sức mạnh của từng khuôn cảnh cho một dự án riêng lẻ. Khuôn cảnh xoắn ốc thực hiện điều này một cách trực tiếp tổ hợp cả làm bản mẫu và các yếu tố của vòng đời cổ điển trong một cách tiếp cận tiến hoá tới kỹ nghệ phần mềm. Nhưng bất kỳ một trong các khuôn cảnh này cũng đều có thể làm nền tảng để tích hợp các khuôn cảnh khác.



Hình vẽ trên đây minh hoạ một cách tổ hợp các khuôn cảnh của kỹ nghệ phần mềm trong một nỗ lực phát triển phần mềm duy nhất. Trong mọi trường hợp, công việc bắt đầu với việc xác định mục tiêu, phương án ràng buộc - một bước đôi khi vẫn còn được gọi là *thu thập yêu cầu sơ bộ*. Từ điểm này, bất kỳ một con đường nào được vẽ trên hình dưới đây đều có thể được chọn. Chẳng hạn có thể đi theo con đường vòng đời cổ điển (đường bên trái), nếu có thể xác định được toàn bộ hệ thống ngay từ đầu. Nếu các yêu cầu còn chưa được chắc chắn thì có thể sử dụng bản mẫu để xác định yêu cầu một cách đầy đủ hơn. Bằng cách dùng bản mẫu như là một hướng dẫn, người phát triển có thể trở lại các bước của vòng đời cổ điển (thiết kế, mã hoá và kiểm thử). Theo một

cách khác, bản mẫu có thể tiến hoá thành hệ thống sản xuất, với việc quay trở về khuôn cảnh vòng đời để kiểm thử. Các kỹ thuật thế hệ thứ 4 có thể được dùng để cài đặt bản mẫu hay cài đặt hệ thống sản xuất trong bước mã hoá của vòng đời. 4GT có thể được dùng kèm với mô hình xoắn ốc cho các bước làm bản mẫu hay mã hoá.

Không cần phải võ đoán về việc chọn khuôn cảnh cho kỹ nghệ phần mềm. Bản chất của ứng dụng nên ấn định ra cách tiếp cận cần được chọn. Bằng cách tổ hợp các cách tiếp cận thì một tổng thể sẽ còn lớn hơn là tổng của từng thành phần.



## CHƯƠNG 2

### TIÊU CHUẨN CỦA MỘT SẢN PHẨM PHẦN MỀM

#### 2.1. Tiêu chuẩn về trình độ và cấu trúc của nhóm sản xuất phần mềm.

Trong nhóm những người phát triển phần mềm, cần có hiểu biết về các lĩnh vực sau:

1. PC: Tri thức về phần cứng.
2. HT: Khả năng tiếp cận hệ thống.
3. PM: Hiểu biết về công nghệ phần mềm.
4. TT: Tri thức về toán học và thuật toán.
5. LT: Khả năng lập trình.
6. MKT: Khả năng tiếp thị.

Các thành viên trong nhóm phát triển phần mềm cần có mức độ hiểu biết về các lĩnh vực như sau:

*Chủ nhiệm đề tài:* Là người có hiểu biết về khá về hệ thống và MKT, là người có khả năng tâm lý học cao nhất, có khả năng về đối nội và đối ngoại.

*Người phân tích và thiết kế hệ thống:* Phải khá về tất cả mọi mặt, còn phần cứng và phần mềm chỉ cần biết là được.

*Người đảm bảo phần cứng:* Giỏi về phần cứng và phần mềm.

*Người đảm bảo phần mềm:* Là cố vấn về phần mềm, góp ý và cung cấp các công cụ phần mềm hệ thống và tiện ích thích hợp giúp cho nhóm giảm được tối đa công sức và thời gian là những công việc trùng lặp.

*Người lập trình:* là người chuyên về lập trình, hiểu biết thuật toán và chuyển đổi theo cú pháp của các ngôn ngữ.

*Phụ trách MKT:* Giỏi giao dịch và biết về hệ thống.

**Bảng tóm tắt về tiêu chuẩn của nhóm thành viên sản xuất phần mềm**

Kiến thức Thành viên	PC	HT	TT	PM	LT	MKT
1. Chủ nhiệm đề tài	B	K	B	B	B	K / G
2. PT & TK hệ thống	B	K	K	K	K	B
3. Đảm bảo phần cứng	K / G	B	B	K	B	B
4. Đảm bảo phần mềm	B	K	K	K / G	K	B
5. Người lập trình	B	B / K	K	K / G	K / G	B
6. Phụ trách MKT	B	K	B	B	B	K / G

Ghi chú: Các ký hiệu G - Giỏi, K - Khá, B - Biết.

## 2.2. Các lỗi có thể mắc trong quá trình thiết kế và cài đặt các phần mềm.

**Lỗi thứ 1:** *Lỗi về ý đồ thiết kế sai.* Đây là lỗi nặng nhất. Hệ thống mà chúng ta xây dựng sẽ không thể đáp ứng được yêu cầu của khách hàng.

**Lỗi thứ 2:** *Lỗi phân tích các yêu cầu không đầy đủ hoặc lệch lạc.* Đây là lỗi cũng thường xảy ra. Thực tế cho thấy, những người làm chuyên môn thì không hiểu sâu về tin học nên không cung cấp được những thông tin cần thiết cho những người làm tin học. Ngược lại, những người làm tin học là không hiểu hết về chuyên môn nghiệp vụ của khách hàng. Do vậy mà việc thu thập thông tin sẽ không đầy đủ hoặc thiếu chính xác. Chính vì vậy mà dễ mắc lỗi. Lỗi này có thể được khắc phục tại các cuộc gặp gỡ giữa hai bên và giải đáp những điều còn mơ hồ.

**Lỗi thứ 3:** *Lỗi hiểu sai các chức năng.* Đây là lỗi thường hay mắc phải do trong hệ thống có thể có các chức năng hay lĩnh vực có tính chuyên môn cao. Các từ chuyên ngành. Dẫn đến khó hiểu đối với nhà phát triển phần mềm.

Ví dụ: Đối với phân số, khi cài đặt để đỡ rắc rối thì ta quan niệm

$Tử\_số \in \mathbb{Z}$  (số nguyên);  $Mẫu\_số \notin \mathbb{N}$  (số tự nhiên). Như vậy biểu thức 3/-4 sẽ được hiểu là thương của hai số nguyên. Khi cài đặt, đôi khi người ta không chú ý đến chuyện này, do vậy có thể mắc lỗi.

**Lỗi thứ 4:** *Lỗi bỏ sót các chức năng.* Lỗi này các nhà phát triển phần mềm cũng hay mắc phải, do điều kiện thời gian và chuyên môn có hạn, đôi khi các chức năng không thể được đưa ra một cách đầy đủ. Lỗi này có thể được hạn chế (không phải là khắc phục tất cả) qua thời gian làm việc nhiều hơn với khách hàng, do vậy mà ta có thể biết được nhiều thông tin hơn.

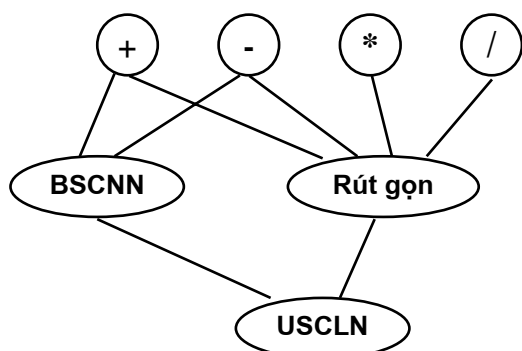
Ví dụ: Khi thực hiện các phép toán với  $Phân\_số$  ta quên rút gọn phân số; không khởi tạo; kiểm tra phép chia cho số 0, ...

Một khía cạnh khác nữa, đối với việc thiết kế hướng đối tượng (sẽ nghiên cứu sau), ta cần phải tuân theo nguyên lý về hướng đối tượng (chủ yếu là tính che dấu thông tin và kế thừa): ta phải biết cách để truy nhập đến từng thành phần của đối tượng.

**Lỗi thứ 5:** *Lỗi tại các đối tượng chịu tải.* Lỗi xảy ra tại các hàm hoặc các thủ tục cấp thấp xây dựng lên các thủ tục khác. Lỗi này cũng là một lỗi nặng, có thể kéo theo sai sót ở một loạt các hàm hoặc thủ tục khác.

Xét về nguyên lý và mức độ lỗi thì lỗi nặng nhất vẫn là ở ý đồ thiết kế sai hoặc là ở thủ tục chịu tải mức thấp nhất. Xét ví dụ sau: Giả sử ta cần thực hiện các phép toán trên một đối tượng  $Phân\_số$ . Khi đó cần thực hiện các phép toán + (cộng), - (trừ), \*

(nhân), / (chia). Để thực hiện phép +, - thì ta cần gọi thủ tục tìm BSCNN, thủ tục này lại phải gọi tới thủ tục tìm USCLN để lấy tích chia cho USCLN; các phép toán + - \* / phải rút gọn USCLN. Như vậy ở đây có các đối tượng chịu tải có mức độ khác nhau và thủ tục USCLN chịu tải nhiều nhất (thủ tục ở cấp thấp nhất).



Như vậy, nếu như trong quá trình thiết kế hay thực hiện cài đặt thủ tục USCLN mà có lỗi thì lỗi đó sẽ ảnh hưởng đến toàn bộ hệ thống của chúng ta.

**Lỗi thứ 6: Lỗi lây lan.** Đây là lỗi do virus có thể lây từ chương trình này sang chương trình khác. Ví dụ, nếu trong thư viện có một chương trình bị lỗi. Nếu ta gọi thủ tục này thì sẽ có lỗi.

**Lỗi thứ 7: Lỗi cú pháp.** Lỗi này sinh ra do việc viết sai các quy định về văn phạm. Những lỗi này thường được chương trình dịch thông báo ngay khi dịch theo nguyên lý an toàn: các lỗi nhỏ nhất cũng phải được xử lý ngay khi dịch đến đó.

**Lỗi thứ 8: Lỗi do hiệu ứng phụ.** Lỗi xảy ra do việc sử dụng hàm, thủ tục hay chương trình con, có các phép tính biến đổi chương trình con nằm ngoài ý muốn của người lập trình. Xét ví dụ sau:

```

Var x, y, z : real;
Function FF : real;
Begin
    x := 10 + 2*x;
    FF := x + y/5;
End;
Begin
    Write(' x = '); readln(x);
    Write(' y = '); readln(y);
    z := FF;
    Writeln(' 10 + 2*', x, '+', y, '/5 = ', z);
End.
    
```

Chương trình sai là do x là biến toàn cục nên nó tác động trên toàn bộ chương trình. Có nhiều cách sửa bằng cách sửa chương trình bằng cách thêm biến phụ hoặc biến địa phương. Hay bằng cách chuyển đổi lại cách in ra (vì kết quả vẫn đúng). Để tránh hiệu ứng phụ ta cần phải tuân theo:

- ✦ Tất cả các biến được khai báo ở trong chương trình con đều là biến địa phương.
- ✦ Tất cả các tham biến hình thức được truyền theo tham trị trong chương trình con dù có trùng tên với biến toàn cục cũng không làm thay đổi giá trị của biến toàn cục.
- ✦ Đối với các phép tính làm thay đổi giá trị của biến thì phải dùng biến phụ.

## 2.3. Các tiêu chuẩn của một sản phẩm phần mềm.

### 2.3.1. Sản phẩm phần mềm là gì?

Sản phẩm phần mềm là một hoặc một nhóm các chương trình được xây dựng để giải quyết một vấn đề nào đó. Ví dụ: chương trình quản lý hoạt động của máy móc và các chương trình ứng dụng.

### 2.3.2. Nhóm các sản phẩm hiện có.

Hiện nay người ta phân chia thành 7 nhóm phần mềm chính.

#### *Nhóm 1: Phần mềm hệ thống.*

Là một tập hợp các chương trình được viết để phục vụ cho các chương trình khác. Chương trình này xử lý các thông tin phức tạp nhưng xác định cấp thấp, tạo môi trường hoạt động (trình biên dịch, trình soạn thảo, quản lý tệp tin, ...).

Các chương trình này đặc trưng bởi tương tác chủ yếu với phần cứng máy tính, phục vụ nhiều người dùng, có cấu trúc dữ liệu phức tạp và nhiều giao diện ngoài.

#### *Nhóm 2: Phần mềm thời gian thực.*

Là phần mềm điều phối hoặc phân tích hay 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 bên 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 các đáp ứng cho 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 được những ràng buộc thời gian chặt chẽ.

#### *Nhóm 3: Phần mềm nghiệp vụ.*

Ngày nay, xử lý thông tin nghiệp vụ là lĩnh vực ứng dụng phần mềm lớn nhất. Phần mềm loại này phục vụ cho các hệ thống rời rạc: *hệ thống tin quản lý*. Các ứng dụng phần mềm nghiệp vụ còn bao gồm cả tính toán tương tác (như xử lý các giao tác cho các điểm bán hàng) ngoài ứng dụng xử lý dữ liệu.

**Nhóm 4: Phần mềm khoa học công nghệ.**

Phần mềm này được đặc trưng bởi các thuật toán. Phần mềm tạo ra một ứng dụng mới, thiết kế có máy tính trợ giúp (computer aided of design - CAD), có chú ý đến các đặc trưng thời gian thực và phần mềm hệ thống.

**Nhóm 5: Phần mềm nhúng.**

Nằm trong bộ nhớ chỉ đọc và được dùng để điều khiển các sản phẩm và hệ thống cho người dùng và thị trường công nghiệp. Có thể thực hiện các chức năng đơn giản nhưng mang tính chuyên biệt (huyền bí), ví dụ: điều khiển chức năng cho lò vi sóng; hay có thể đưa ra các khả năng điều khiển và vận hành (chức năng số hoá ở ô-tô, kiểm soát xăng, biểu thị bảng đồng hồ, các hệ thống phanh...).

**Nhóm 6: Phần mềm máy tính cá nhân.**

Loại phần mềm này bùng nổ trong hơn thập kỷ vừa qua (như xử lý văn bản, trang tính, đồ hoạ, quản trị cơ sở dữ liệu). Hiện nay được tiếp tục phát triển biểu thị giao diện người máy, tạo ra sự thân thiện, dễ sử dụng cho người dùng.

**Nhóm 7: Phần mềm trí tuệ nhân tạo.**

Dùng các thuật toán phi số để giải quyết các vấn đề phức tạp mà tính toán hay phân tích trực tiếp đều không thể quản lý nổi. Phần mềm này hoạt động mạnh ở hệ chuyên gia (hệ cơ sở tri thức); trong lĩnh vực nhận dạng và xử lý hình ảnh và âm thanh; chứng minh các định lý và chơi trò chơi. Hiện nay phát triển mạnh mạng nơ-ron nhân tạo: mô phỏng cấu trúc việc xử lý trong bộ não của con người.

**2.3.3. Các tiêu chuẩn của một sản phẩm phần mềm hiện có.**

Người ta xác định một số tiêu chuẩn để đánh giá một sản phẩm phần mềm.

**Tiêu chuẩn 1: Tính đúng đắn.**

Các sản phẩm phần mềm phải thực hiện được chính xác các mục tiêu được đặt ra ở giai đoạn thiết kế, không bị treo máy hoặc ra kết quả sai đối với bộ dữ liệu nằm trong phạm vi yêu cầu. Để đạt được yêu cầu này, các sản phẩm phần mềm trước hết phải có thuật toán đúng và chương trình tình phải tương ứng với thuật toán.

**Tiêu chuẩn 2: Tính khoa học.**

*Tính khoa học về cấu trúc:* Các sản phẩm phần mềm được chia thành các đơn vị nhỏ cân đối và có quan hệ hữu cơ không trùng lặp và có thể tổ hợp từng nhóm để tạo ra các chức năng mới. Thuật toán và chức năng được xây dựng một cách có cấu trúc.

*Tính khoa học về nội dung:* Thuật toán được xây dựng dựa trên những thành tựu mới của toán học và tin học. Các chương trình phải được xây dựng trên các ngôn ngữ lập trình mới và phổ dụng.

*Tính khoa học về hình thức thao tác:* Mỗi lệnh của chương trình cần phải được tối ưu. Muốn vậy, các lệnh phải được xây dựng một cách hợp lý, logic và phù hợp với tư duy tự nhiên của người sử dụng. Các lỗi phải được thông báo một cách rõ ràng (lỗi số bao nhiêu, vị trí lỗi, nội dung lỗi, cách khắc phục).

***Tiêu chuẩn 3: Tính hữu hiệu.*** Thể hiện ở các mặt sau:

*Hữu hiệu về kinh tế:* Có giá trị kinh tế hoặc có ý nghĩa giá trị thu được khi áp dụng sản phẩm đó.

*Hữu hiệu về tốc độ xử lý:* Có số lượng lớn các đối tượng được xử lý trong một đơn vị thời gian. Lượng tối đa của sản phẩm quản lý được (ví dụ: trong Excel quản lý được 65536 bản ghi, FoxPro quản lý được 255 trường).

*Hữu hiệu về dung lượng bộ nhớ.* Tốn càng ít càng tốt.

***Tiêu chuẩn 4: Tính sáng tạo.***

Sản phẩm phải mới mẻ và độc đáo. Nếu phát triển trên cái cũ thì phải tiếp theo được những cái hay của nó đồng thời phải cung cấp được các chức năng mới tốt hơn so với cái đã có.

***Tiêu chuẩn 5: Tính an toàn.***

Sản phẩm phần mềm phải có cơ chế bảo mật chống xâm phạm, sao chép trộm và làm biến dạng chương trình. Có cơ chế bảo vệ đối tượng mà nó phát sinh và quản lý, có cơ chế hồi phục khi có sự cố.

***Tiêu chuẩn 6: Tính đầy đủ và toàn vẹn.***

Sản phẩm thực hiện được đầy đủ yêu cầu của khách hàng. Các chức năng phải có tính đối xứng, nghĩa là: có tạo lập thì có xoá bỏ, có mở thì có đóng, có tiếp theo thì cũng cho phép trở về, ...

***Tiêu chuẩn 7: Tính độc lập với các thiết bị.***

Sản phẩm có thể sử dụng trên nhiều loại máy khác nhau và sử dụng nhiều các thiết bị đi kèm khác nhau. Độc lập cả với cấu trúc của đối tượng mà nó phát sinh ra.

***Tiêu chuẩn 8: Tính phổ dụng.***

Có thể sử dụng được rộng rãi trong nhiều lĩnh vực và ở nhiều chế độ làm việc.

**Tiêu chuẩn 9:** *Tính dễ học và dễ sử dụng, cải tiến.*

Sản phẩm hợp với yêu cầu người dùng về ngôn ngữ, hệ thống các chức năng (menu), các thông báo, cú pháp đơn giản, rõ ràng, dễ nhớ, dễ thao tác, dễ tăng cường các chức năng, dễ mở rộng và cải tiến.

## 2.4. Hồ sơ của một sản phẩm phần mềm.

Khi bàn giao sản phẩm cho khách hàng ta cần chú ý đến các thành phần sau:

- ✦ Phải có chương trình nguồn, chương trình đích để trên đĩa.
- ✦ Đính kèm các phần mềm tiện ích có liên quan.
- ✦ Các bản in chương trình nguồn, trên đó có lời giải thích rõ ràng để tiện cho việc chứng minh tính đúng đắn của chương trình.
- ✦ Bản mô tả các thuật toán của chương trình.
- ✦ Bảng hướng dẫn sử dụng chi tiết, các lỗi có thể có và cách xử lý lỗi.
- ✦ Các thông số đặc trưng của chương trình, sản phẩm gồm: Tên chủ nhiệm đề tài, chức vụ, nơi công tác, địa chỉ, điện thoại, ... Các thông tin về sản phẩm: tên đầy đủ, tên vắn tắt, số hiệu phiên bản, ngày tháng thiết kế và cài đặt, các chức năng của hệ thống, chế độ làm việc (hộp hội thoại, menu, ...).
- ✦ Cấu hình tối thiểu của hệ thống, các thiết bị đi kèm. Cấu hình tối đa sử dụng hết công suất của sản phẩm.
- ✦ Giới thiệu về ngôn ngữ lập trình được sử dụng để tạo ra sản phẩm.
- ✦ Cơ chế bảo mật và một số phần mềm tương thích.
- ✦ Cung cấp thêm một số tư liệu khác: phần mềm quốc phòng, một số các kết quả đã sử dụng ở một số nơi, thời gian sử dụng là bao nhiêu, yêu cầu về bản quyền, có thể biết khoá bảo mật hay không? ...

## CHƯƠNG 3

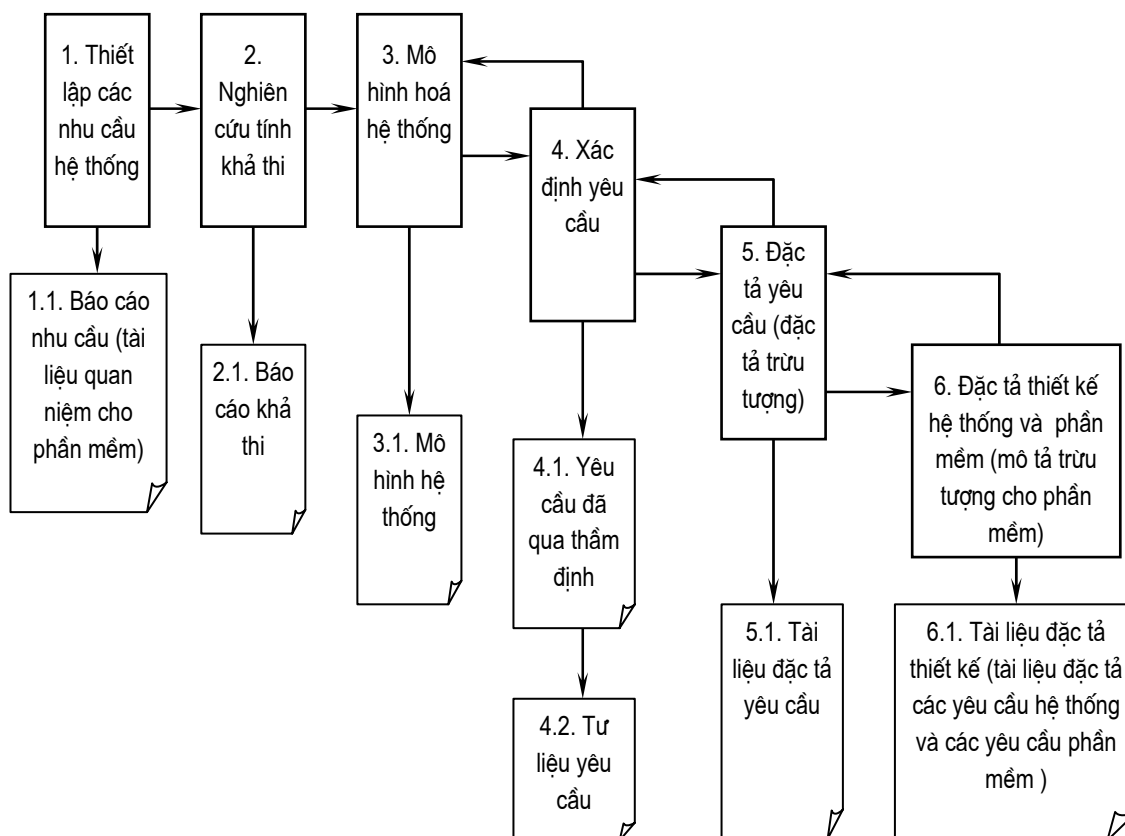
### ĐẶC TẢ

#### 3.1. Khái niệm về 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 đó có thể là đối tượng, khái niệm, một thủ tục nào đó, ...

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

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 sau:



Các đặc tả thường mang tính trừu tượng hoá cao. Do vậy người ta phân chia thành nhiều mức đặc tả. Càng ở mức cao (những mức đầu tiên của quá trình làm mịn hoặc chính xác hoá) đặc tả càng trừu tượng. Càng xuống các mức thấp hơn, đặc tả càng tiến dần tới cụ thể - tức là một thể hiện trên một máy tính cụ thể với một ngôn ngữ lập trình cụ thể - đây chính là quá trình làm mịn dần.

#### 3.2. Các loại hình đặc tả.

Có hai kiểu đặc tả đó là *đặc tả hình thức* và *đặc tả phi hình thức*.



**Đặc tả hình thức:** Là các 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.

Ví dụ: Đặc tả một ma trận:

- Cấp của ma trận  $n \times n$  ( $n$  là số tự nhiên lẻ).
- Phần tử cuối của hàng 1 bằng phần tử đầu của hàng cuối.
- Phần tử trung tâm bằng trung bình cộng của các phần tử ở 4 góc.

Hoặc có thể diễn đạt như sau:

- $A_{n \times n} = (a[i, j])_{n \times n}; n = 2k + 1, k \in \mathbb{Z}.$
- $a[1, n] = a[n, 1].$
- $a\left[\frac{n+1}{2}, \frac{n+1}{2}\right] = (a[1, 1] + a[1, n] + a[n, 1] + a[n, n]) / 4$

**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 các điểm chưa rõ ràng, những khái niệm còn mơ hồ.

Ví dụ: Có hai con hậu trên bàn cờ. Hai con hậu sẽ đụng độ nếu chúng nằm trên cùng hàng, cùng cột hoặc trên cùng một đường chéo song song với đường chéo chính hay đường chéo phụ.  $\Rightarrow$  Rõ ràng ở đây có một số khái niệm mơ hồ.

**Đặc tả hỗn hợp:** Phối hợp cả hai kiểu đặc tả trên.

**Trong thực tế, có nhiều loại hình đặc tả, ví dụ như:**

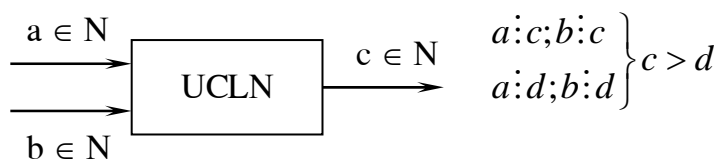
a. **Đặc tả cấu trúc dữ liệu:** Nêu các thành phần của dữ liệu

Ví dụ: Đặc tả một phân số:  $\text{Phân\_số} = \{ x/y, x \in \mathbb{Z}, y \in \mathbb{N} \}$

$\text{Số\_phức} = \{ a + b.i \mid a, b \in \mathbb{R} \}$

b. **Đặc tả chức năng:** Mô tả thông qua việc nêu lên các tính chất hay thuộc tính của tên vào và tên ra.

Ví dụ:

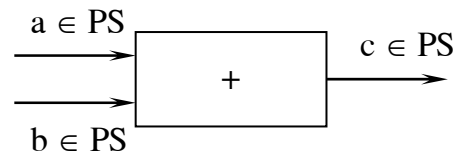


c. **Đặc tả đối tượng:** Bao gồm đặc tả cấu trúc dữ liệu và mô tả các chức năng.

Ví dụ: đặc tả đối tượng phân số.

$$PS = \{ x/y, x \in \mathbb{Z}, y \in \mathbb{N} \}$$

Phép cộng:  $+: PS \times PS \rightarrow PS$



d. *Đặc tả thao tác*: Nêu lên trình tự tiến hành công việc.

Ví dụ 1:  $x, y, z \in PS$ . Các bước cần thực hiện đối với phép cộng (+) 2 phân số.

```

z = x + y {   Quy_đồng_mẫu_số(x, y);
              z.tử_số = x.tử_số + y.tử_số;
              z.mẫu_số = x.mẫu_số;
            };
    
```

Ví dụ 2: Quy trình **Bán hàng**:

1. Khách hàng yêu cầu được mua hàng.
2. Hướng dẫn khách xem và lựa chọn hàng hoá.
3. Thoả thuận hình thức thanh toán: Tiền mặt, séc, chuyển khoản, ...
4. Ghi hoá đơn cho khách.
5. Nhận tiền và giao hàng hoá cho khách.

e. *Đặc tả cú pháp*: Thực chất là các định nghĩa có tính truy hồi từ tổng thể đến cơ sở. Mô tả cách lắp ghép các ký hiệu, các từ với nhau lại để tạo thành chương trình. Ví dụ: Trong ngôn ngữ lập trình PASCAL, tên (định danh - identify) được khái quát như sau: Là dãy các ký tự bắt đầu bằng chữ cái hoặc dấu gạch nối dưới, sau đó có thể là chữ số, chữ cái hoặc dấu gạch nối dưới.

$\langle \text{định danh} \rangle = \langle \text{chữ cái} \rangle \cup \langle \text{định danh} \rangle \cup \langle \text{ký tự} \rangle$

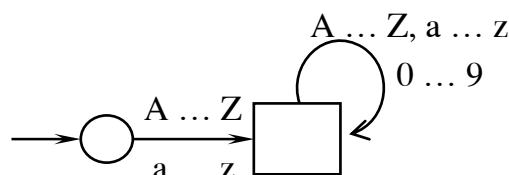
$\langle \text{ký tự} \rangle = \langle \text{chữ cái} \rangle \cup \langle \text{chữ số} \rangle$

$\langle \text{chữ cái} \rangle = \{ A, B, C, \dots, Z \} \cup \{ a, b, \dots, z \}$

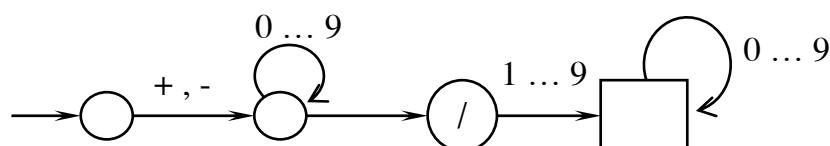
$\langle \text{chữ số} \rangle = \{ 0, 1, 2, \dots, 9 \}$

f. *Đặc tả qua sơ đồ:*

Ví dụ: Đặc tả định danh



Đặc tả phân số



g. *Đặc tả thuật toán:* Các bước thao tác để giải quyết bài toán.

Kiểu đặc tả phải phù hợp với giải pháp. Các yêu cầu của 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 CASE) có chứa các mô tả ngôn ngữ đồ họa và tự nhiên cho yêu cầu phần mềm. Việc làm bản mẫu giúp đặc tả có thể được triển khai, tức là bản mẫu sẽ thể hiện những công việc thực hiện các yêu cầu. Các ngôn ngữ đặc tả hình thức dẫn đến biểu diễn hình thức.

### 3.3. Các nguyên lý đặc tả.

Đặc tả có thể xem như một tiến trình biểu diễn. Mục đích cuối cùng của đặc tả là các yêu cầu được biểu thị sao cho dẫn tới việc cài đặt phần mềm thành công. Balzer và Goldman đề nghị 8 nguyên lý đặc tả tốt.

**Nguyên lý 1:** *Phân tách chức năng với cài đặt.*

Trước hết, theo định nghĩa, đặc 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 2 dạng hoàn toàn khác nhau. *Dạng thứ nhất* là dạng của các *hàm toán học*: Với một tập dữ liệu đầu vào đã cho, tạo ra một tập dữ liệu đầu ra đặc biệt. Dạng tổng quát của đặc tả như thế là tìm ra (một hoặc tất cả những) kết quả ứng với P (đầu vào), với P biểu thị một tân từ bất kỳ. Trong đặc tả như thế, kết quả thu được phải được diễn đạt một cách đầy đủ, toàn vẹn, theo dạng đó là cái gì (không phải đó là như thế nào). Một phần điều này là vì kết quả của một hàm (toán học) của đầu vào (phép toán có điểm bắt đầu và điểm kết thúc đã xác định rõ) không bị ảnh hưởng bởi môi trường bao quanh.

**Nguyên lý 2:** *Cần 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 (như trong “hệ thống máy tính nhúng”). Hành vi của nó không thể biểu diễn được ở dạng hàm (toán học) của đầu vào. Thay vì thế, cần phải sử dụng cách biểu diễn khác - cách mô tả hướng tiến trình, trong

đó đặc tả cái gì đã đạt được bằng cách xác định một mô hình các thao tác mong muốn đạt được của hệ thống dưới dạng các công việc đáp ứng chức năng đối với kích thích khác nhau từ môi trường.

Những đặc tả hướng tiến trình như vậy, trình bày một mô hình về hành vi hệ thống, thông thường đã bị loại ra khỏi các ngôn ngữ đặc tả hình thức, nhưng chúng lại là bản chất nếu nhiều tình huống động phức tạp hơn cần phải được đặc tả. Trong thực tế, cần phải thừa nhận rằng trong những tình huống như vậy cả tiến trình cần tự động hoá lẫn môi trường tồn tại của nó đều phải được mô tả một cách hình thức. Tức là, toàn bộ hệ thống các bộ phận tương tác phải được đặc tả chứ không chỉ một thành phần được đặc tả.

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

Một hệ thống bao gồm các thành phần tương tác nhau. Chỉ bên trong hoàn cảnh của hệ thống toàn bộ 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 có thể được mô hình hoá như một tập hợp các sự vật tích cực và thụ động. Những sự vật này có liên quan lẫn nhau và qua thời gian thì 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 phải bao gồm cả môi trường mà hệ thống vận hành.*

Tương tự, môi trường mà trong đó hệ thống vận hành và tương tác với cũng phải được xác định.

May mắn là điều này đơn thuần chỉ cần sự thừa nhận rằng bản thân môi trường cũng là một hệ thống bao gồm các sự vật tương tác, cả tích cực lẫn thụ động, mà trong đó hệ thống 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 khác.

Cần phải chú ý rằng bức tranh đặc tả hệ thống được trình bày ở đây chính là bức tranh của tập hợp các tác nhân xoắn xuýt nhau cao độ phản ứng với những kích thích trong môi trường (thay đổi các sự vật) do các tác nhân đó tạo ra. Chỉ có thông qua những hành động điều phối của tác nhân mà hệ thống mới đạt tới mục tiêu của nó. Sự phụ thuộc lẫn nhau vi phạm vào nguyên lý phân tách (cô lập với các phần khác của hệ thống và môi trường). Nhưng đây là một nguyên lý *thiết kế*, không phải là nguyên lý đặc tả. Thiết kế tuân theo đặc tả, và quan tâm tới việc phân rã một đặc tả thành các mẫu gắn 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 theo một cách thức nhiều chi tiết như các giai đoạn cài đặt và thiết kế cần tới. 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 có để bao quát thật nhiều cho 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. Nó phải mô tả một hệ thống như cộng đồng người sử dụng cảm nhận thấy. Các sự vật mà nó thao tác phải tương ứng với các sự vật của lĩnh vực đó; các tác nhân phải mô hình cho các cá nhân, tổ chức và trang thiết bị trong lĩnh vực đó; còn các hành động họ thực hiện thì phải mô hình cho những hoạt động thực tế xuất hiện trong lĩnh vực.

Đặc tả phải có khả năng tổ hợp vào trong nó những qui tắc hay luật bao trùm các sự vật thuộc lĩnh vực. Một số trong những trường hợp là luật *bài trừ những trạng thái nào đó của hệ thống* (như “hai sự vật không thể đồng thời ở cùng một chỗ và vào cùng một lúc”), và do đó giới hạn hành vi của các tác nhân hay chỉ ra nhu cầu soạn thảo thêm để ngăn cản những trạng thái này khỏi nảy sinh. Các luật khác *mô tả cách các sự vật đáp ứng lại khi bị kích thích* (như luật chuyển động của Newton). Những luật này, biểu thị cho “tính vật lý” của lĩnh vực, là phần cố hữu của đặc tả hệ thống.

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

Đặc tả phải đủ đầy đủ và hình thức để có thể được dùng trong việc xác định liệu một cài đặt được đề nghị có thoả 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 có thể trong số những hành vi phải có của cài đặt được đề nghị. Do đó, theo một nghĩa mở rộng, đặc tả này phải là vận hành ...

**Nguyên lý 7:** *Đặc tả chấp nhận dung sai về tính không đầy đủ.*

Không đặc tả nào có thể là đầy đủ hoàn toàn. Môi trường trong đó nó tồn tại thường quá phức tạp cho điều đó. Một đặc tả bao giờ cũng là một mô hình - một sự trừu tượng hoá - của một tình huống thực (hay được mượn tượng) nào đó. Do đó, nó sẽ không đầy đủ. Hơn thế nữa, như đã được phát biểu nó sẽ tồn tại tại ở nhiều mức chi tiết. Tính vận hành được yêu cầu ở trên không nhất thiết là cần thiết. Các công cụ phân tích được sử dụng để giúp cho người đặc tả và để kiểm thử đặc tả phải có khả năng xử lý với tính không đầy đủ. Một cách tự nhiên điều này làm cho việc phân tích bị yếu đi, khi có thể được thực hiện bằng cách mở rộng phạm vi các hành vi chấp nhận được thoả

mãn cho đặc tả, nhưng một sự suy giảm như vậy phải phản ánh các mức độ bất trắc còn lại.

**Nguyên lý 8:** *Đặc tả phải được cục bộ hoá và được ghép lồng lẻo.*

Các nguyên lý trước xử lý đặc tả như một thực thể tĩnh. Thực thể này nảy sinh từ cái động của đặc tả. Cần phải thừa nhận rằng mặc dầu mục tiêu chính của một đặc tả là để dùng làm cơ sở cho thiết kế và cài đặt một hệ thống nào đó, nó không phải là một sự vật tĩnh dụng sẵn mà là một sự vật động đang trải qua thay đổi đáng kể. Việc thay đổi như thế 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 cho đặc tả, điều mấu chốt là nội dung và cấu trúc của nó được chọn để làm phù hợp hoạt động này. Yêu cầu chính cho sự phù hợp đó là ở chỗ thông tin bên trong đặc tả phải được cục bộ hoá sao cho chỉ một phần nhỏ (một cách lí tưởng) cần phải sửa đổi khi thông tin thay đổi, và ở chỗ đặc tả cần được cấu trúc (ghép) một cách lồng lẻo để 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.

Mặc dầu các nguyên lý được Balzer và Goldman tán thành tập trung vào tác động của đặc tả trên định nghĩa về ngôn ngữ hình thức, những lời bình luận của họ áp dụng được cho cả mọi dạng đặc tả. Tuy nhiên, các nguyên lý cần phải được dịch thành sự thực hiện. Trong mục sau chúng ta sẽ xem xét một tập các hướng dẫn để tạo ra một đặc tả các yêu cầu.

### 3.4. Các mức trừu tượng của đặc tả.

Các đặc tả được thể hiện ở một vài mức trừu tượng khác nhau cùng với mối tương liên giữa các mức ấy. Mỗi mức nhắm đến các đối tượng đọc khác nhau mà họ có quyền quyết định về việc dựa vào đó mà thực hiện đánh giá bản thiết kế của các nhà phát triển phần mềm. Các mức đó là:

**Mức 1:** *Định ra yêu cầu.*

Được thể hiện bằng ngôn ngữ tự nhiên về các dịch vụ mà hệ thống sẽ phải cung cấp. Phần này phải được viết sao cho dễ hiểu đối với khách hàng và người quản lý hợp đồng, người sẽ mua sản phẩm phần mềm và người sẽ sử dụng nó. Kỹ thuật đặc tả phi hình thức là thích hợp cho mức đặc tả này.

**Mức 2:** *Đặc tả yêu cầu.*

Tài liệu nêu ra các dịch vụ một cách chi tiết hơn. Tài liệu này đôi khi còn được gọi là tài liệu đặc tả chức năng. Yêu cầu đối với đặc tả ở mức này là phải chính xác đến mức có thể làm cơ sở cho hợp đồng giữa nhà phát triển phần mềm và khách hàng. Đồng thời cũng cần được viết sao cho dễ hiểu đối với nhân viên kỹ thuật của cả nơi

mua phần mềm và nơi phát triển hệ thống. Kỹ thuật đặc tả hình thức hần là thích hợp cho mức đặc tả như vậy, tuy nhiên cũng còn tùy thuộc vào trình độ kiến thức cơ bản của khách hàng. Tốt hơn cả là ta có thể dùng loại hình hỗn hợp để đặc tả.

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

Dùng làm cơ sở cho việc thiết kế và thực thi. Cần thể hiện một quan hệ rõ ràng giữa tư liệu này và đặc tả yêu cầu. Ta phải xác định rằng: đối tượng đọc ở đây chủ yếu là các kỹ sư phần mềm chứ không phải là người sử dụng hoặc người quản lý. Kỹ thuật đặc tả hình thức là hoàn toàn phù hợp cho mức đặc tả này.

### 3.5. Xét duyệt đặc tả.

Việc xét duyệt bản *Đặc tả 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 kì cẩn thận trong khi tiến hành cuộc họp xét duyệt.

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:

1. Các mục tiêu và mục đích đã được thiết lập cho phần mềm có nhất quán với mục tiêu và mục đích của hệ thống hay không?
2. Những giao diện quan trọng với mọi phần tử hệ thống đã được mô tả chưa?
3. Luồng và cấu trúc thông tin đã được mô tả thích hợp cho lĩnh vực vấn đề chưa?
4. Các biểu đồ có rõ ràng không? Liệu mỗi biểu đồ có thể đứng riêng không lời giải thích không?
5. Các chức năng chính có còn bên trong phạm vi và đã được mô tả thích hợp chưa?
6. Liệu hành vi của phần mềm có nhất quán với thông tin nó phải xử lí và chức năng nó phải thực hiện hay không?
7. Các ràng buộc thiết kế có hiện thực không?
8. Rủi ro công nghệ phát triển là gì?
9. Các yêu cầu phần mềm khác đã được xem xét đến chưa?
10. Các tiêu chuẩn hợp lệ đã được phát biểu chi tiết chưa? Chúng có thích hợp để mô tả một hệ thống thành công không?
11. Liệu có sự không nhất quán, bỏ sót hay dư thừa nào không?
12. Việc tiếp xúc với khách hàng có đầy đủ không?

13. Người dùng đã xét duyệt bản *Tài liệu sơ bộ của người dùng* hay bản mẫu chưa?

14. Các ước lượng về *Kế hoạch dự án phần mềm* bị ảnh hưởng thế nào?

Để đưa ra câu trả lời cho nhiều câu hỏi trên, việc xét duyệt có thể tập trung vào mức *chi tiết*. Tại đây, mỗi quan tâm của chúng ta là vào từ ngữ của bản đặc tả. Chúng ta cố gắng làm lộ ra vấn đề có thể ẩn náu bên trong nội dung đặc tả. Những hướng dẫn sau đây là gợi ý về việc xét duyệt chi tiết bản đặc tả:

- Phải quan sát các mối 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 đó?”
- Theo dõi những thuật ngữ mơ hồ (như “một số”, “đôi khi”, “thường”, “thông thường”, “bình thường”, “phần lớn”, “đa số”); 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”, “nhảy qua”, “xoá bỏ”. Có thể có nhiều cách hiểu về nó.
- Phải nhận biết các đại từ “vu vơ” (như “mô đun vào/ra liên lạc với mô đun kiểm tra tính hợp lệ dữ liệu và đặt cờ báo kiểm soát *của nó*.” Cờ kiểm soát của ai? ).
- 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 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 hai thí dụ.

Một khi việc xét duyệt đã hoàn tất thì bản *bản đặc tả yêu cầu phần mềm* sẽ được cả khách hàng lẫn người phát triển “ký tất”. Bản đặc tả trở thành một “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 huỷ bỏ. Nhưng khách hàng phải lưu ý rằng từng thay đổi sau khi kí đều là một mở rộng của phạm vi phần mềm và do đó có thể làm tăng thêm chi phí và / hoặc kéo dài lịch biểu (thời gian thực hiện).

Ngay cả với những thủ tục xét duyệt tốt nhất tại chỗ thì một số vấn đề đặc tả thông thường vẫn còn lại. Bản đặc tả rất khó “kiểm thử” theo mọi cách có ý nghĩa, và do đó sự không nhất quán hay bỏ sót có thể bị bỏ qua không để ý tới. Trong khi xét duyệt, người ta có thể khuyến cáo những thay đổi cho bản đặc tả. Có thể sẽ cực kì khó khăn để lượng định 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 hoá để giúp giải quyết vấn đề này và sẽ được thảo luận trong các chương sau.

# CHƯƠNG 4

## THIẾT KẾ PHẦN MỀM

### 4.1. Đại cương về thiết kế phần mềm.

Trong đời sống hàng ngày, khi một người nào đó cần xây dựng một ngôi nhà, người đó mời một kỹ sư xây dựng đến, yêu cầu thiết kế cho họ ngôi nhà. Với các số liệu về căn nhà cần xây dựng. Căn cứ vào đó, người kỹ sư sẽ thiết kế ra mô hình ngôi nhà. Đây không phải là ngôi nhà được đã được xây dựng trong thực tế, mà chỉ là trên bản vẽ. Nhưng thông qua mô hình đó, cùng với sự mô tả chi tiết của người kỹ sư, chủ nhà cũng có thể hình dung ra ngôi nhà của mình. Bản thiết kế này rất quan trọng, nó giúp cho chủ nhà cùng với kỹ sư xây dựng hiểu về công việc mình cần làm, nếu có yêu cầu chỉnh sửa thì thực hiện chỉ trên bản vẽ. Còn khi đã bắt tay vào xây dựng thực tế thì việc chỉnh sửa lúc này sẽ rất khó khăn và tốn kém.

Khi sản xuất phần mềm cũng vậy. Rõ ràng, yêu cầu của khách hàng cũng không khác gì yêu cầu cần xây ngôi nhà của chủ nhà họ. Công việc của kỹ sư xây dựng và kỹ sư phần mềm theo từng giai đoạn cũng có nhiều điểm chung. Ta hãy xem xét bảng so sánh sau:

Kỹ sư xây dựng	Kỹ sư phần mềm
<ul style="list-style-type: none"> <li>• Khảo sát địa hình, tìm hiểu nhu cầu của chủ nhà: cần xây nhà bao nhiêu tầng, kích thước bao nhiêu, trang trí như thế nào, ...</li> </ul>	<ul style="list-style-type: none"> <li>• Tìm hiểu nhu cầu khách hàng, khảo sát hệ thống, lấy số liệu, ...</li> </ul>
<ul style="list-style-type: none"> <li>• Thiết kế ngôi nhà trên bản vẽ</li> </ul>	<ul style="list-style-type: none"> <li>• Thiết kế phần mềm, đưa ra mô hình</li> </ul>
<ul style="list-style-type: none"> <li>• Tìm hiểu ý kiến chủ nhà về bản thiết kế</li> </ul>	<ul style="list-style-type: none"> <li>• Duyệt lại với khách hàng</li> </ul>
<ul style="list-style-type: none"> <li>• Thực hiện các chỉnh sửa nếu cần</li> </ul>	<ul style="list-style-type: none"> <li>• Thực hiện các chỉnh sửa nếu cần</li> </ul>
<ul style="list-style-type: none"> <li>• Cho thi công ngôi nhà</li> </ul>	<ul style="list-style-type: none"> <li>• Tiến hành cài đặt chương trình</li> </ul>

Thiết kế là bước đầu tiên trong giai đoạn phát triển cho bất kỳ sản phẩm hay hệ thống công nghệ nào. Nó có thể được định nghĩa là "... tiến trình áp dụng nhiều kỹ thuật và 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 của thiết kế là tạo ra một mô hình hay biểu diễn của một thực thể (sự vật: ngôi nhà, chiếc xe hơi, cái cầu, ...) mà sau này được xây dựng.

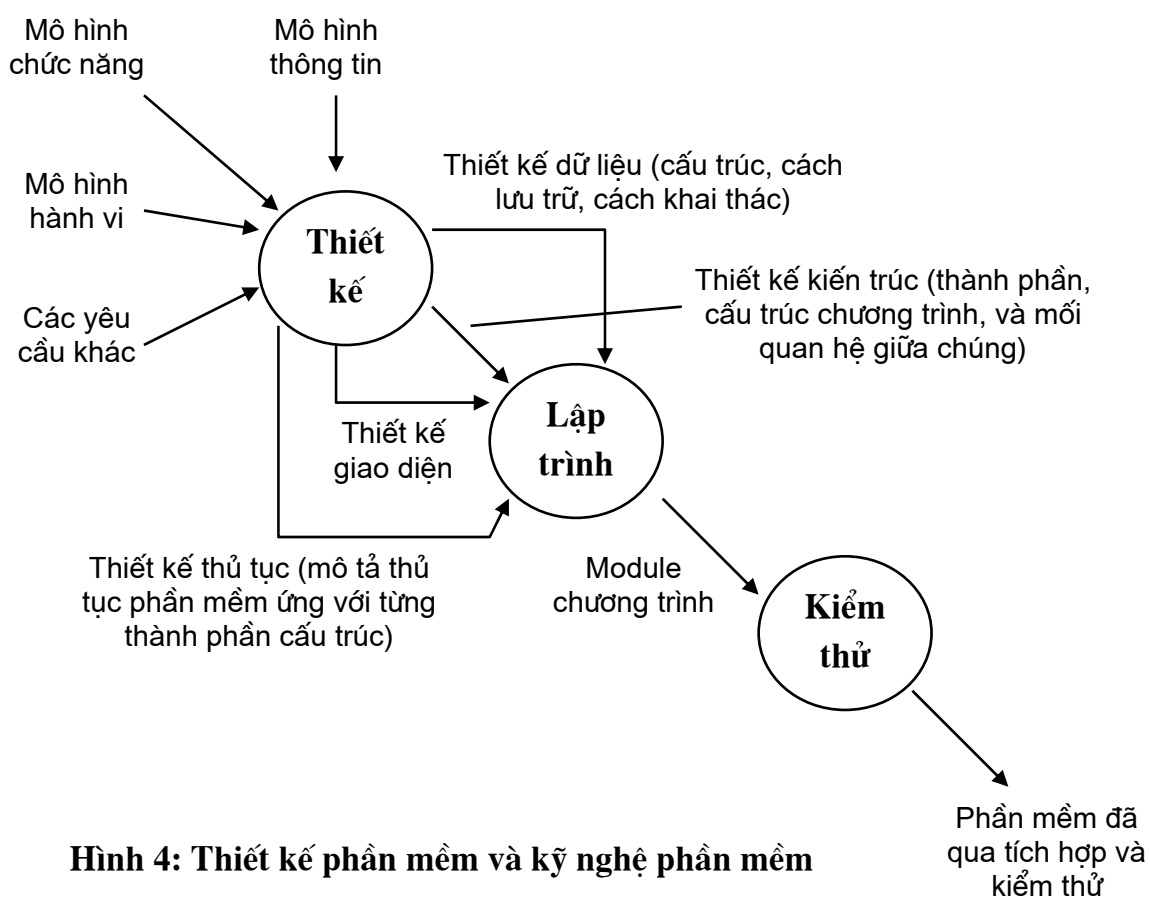
Thiết kế là một quá trình sáng tạo, đòi hỏi kinh nghiệm và sự tinh nhanh của người thiết kế.

Thiết kế phải được thực hành và học bằng kinh nghiệm, bằng khảo sát các hệ thống đang tồn tại, không thể học bằng sách vở (nói đúng ra là không đủ).

Thiết kế phần mềm là một quá trình chuyển hoá các yêu cầu thành một biểu diễn phần mềm. Bước đầu, biểu diễn mô tả toàn bộ về phần mềm. Việc làm mịn tiếp theo sau dẫn tới một biểu diễn thiết kế gần với chương trình gốc.

Thiết kế phần mềm nằm ở trung tâm kỹ thuật của tiến trình kỹ nghệ phần mềm và được áp dụng bất kể khuôn cảnh kỹ nghệ được sử dụng (thác nước, xoáy ốc, bản mẫu, thể hệ thứ 4 - 4GT, ...). Một khi các yêu cầu về phần mềm đã được phân tích và đặc tả thì thiết kế phần mềm là một trong ba hoạt động kỹ thuật - *thiết kế, lập trình, kiểm thử* - những hoạt động cần để xây dựng và kiểm chứng phần mềm. Từng hoạt động này biến đổi thông tin theo cách cuối cùng 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 hoạ trong sơ đồ sau:



**Hình 4: Thiết kế phần mềm và kỹ nghệ phần mềm**

Các yêu cầu phần mềm, được biểu thị bởi 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 tạo ra trong bước phân tích thành cấu trúc 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.

"Hình mẫu thiết kế" có thể được dùng để đạt tới các yêu cầu đã được xác định cho hệ thống, và những ràng buộc ảnh hưởng tới cách mà các hình mẫu thiết kế kiến trúc này có thể được áp dụng. Biểu diễn thiết kế kiến trúc - khuôn khổ của hệ thống dựa trên máy tính - có thể được suy ra từ đặc tả hệ thống, mô hình phân tích và tương tác của các hệ con được định nghĩa bên trong mô hình phân tích.

- **Thiết kế giao diện:** Mô tả cho cách phần mềm trao đổi với chính nó, với hệ thống liên tác với nó, và với người dùng nó. Giao diện bao gồm một luồng thông tin (như dữ liệu và / hoặc điều khiển) và các kiểu hành vi đặc biệt. Do đó, các biểu đồ luồng dữ liệu và điều khiển cung cấp nhiều thông tin cần cho thiết kế giao diện.

- **Thiết kế thủ tục:** Biến đổi các thành phần cấu trúc của kiến trúc phần mềm thành mô tả thủ tục cho các cấu phần phần mềm. Chương trình gốc được sinh ra rồi việc kiểm thử được tiến hành để tích hợp và làm hợp lệ.

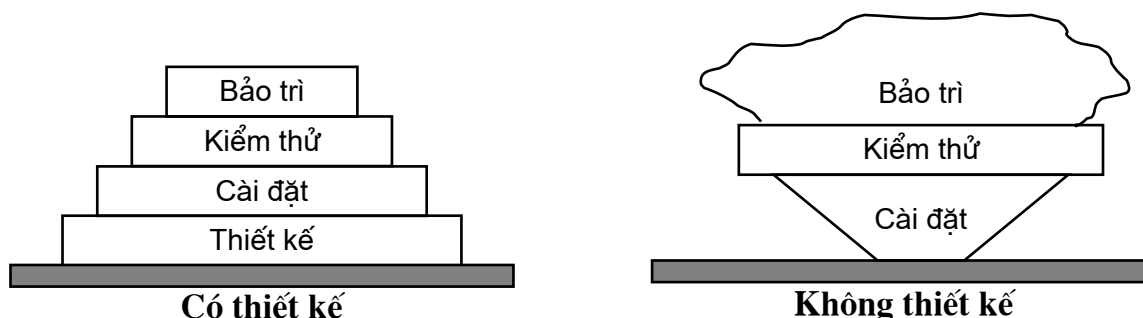
Trong khi thiết kế chúng ta ra các quyết định mà cuối cùng sẽ ảnh hưởng tới sự thành công của việc xây dựng phần mềm và điều quan trọng là ảnh hưởng tới sự dễ dàng bảo trì nó. Nhưng tại sao thiết kế lại quan trọng?

Tầm quan trọng của thiết kế phần mềm có thể được phát biểu bằng một từ - **chất lượng**. Thiết kế là nơi chất lượng được nuôi dưỡng trong việc phát triển phần mềm: 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 hoá một cách chính xác các yêu cầu của khách hàng 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:

### **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 khó có thể mà thử được; một hệ thống không thể nào xác định được chất lượng chừng nào chưa đến cuối tiến trình kiểm thử, khi thời gian còn rất ngắn mà không ít tiền đã phải chi ra.

- Thiết kế tốt là chìa khoá cho công trình hữu hiệu, không thể hình thức hoá quá trình thiết kế trong bất kỳ một công trình nào. Chú ý rằng RAISE chỉ là một phương pháp nghiêm ngặt để viết ra thiết kế, phát triển nó, kiểm tra nó chứ tuyệt nhiên không phải là một phương pháp hình thức để phát triển thiết kế.



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

**Giai đoạn 1:** Nghiên cứu và hiểu ra vấn đề. Không hiểu rõ vấn đề thì không có thể thiết kế được phần mềm hữu hiệu.

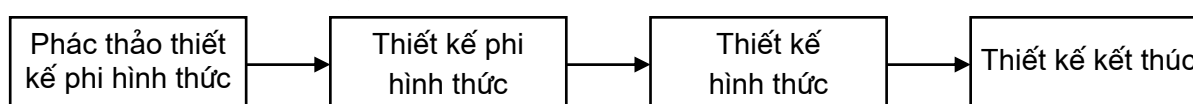
**Giai đoạn 2:** Làm sáng tỏ các đặc điểm lớn của một hoặc một vài giải pháp có thể. Việc chọn giải pháp phụ thuộc vào kinh nghiệm của người thiết kế; phụ thuộc vào các thành phần có thể tái sử dụng và phụ thuộc vào sự đơn giản của các giải pháp trước đó. Kinh nghiệm cho thấy, nếu các nhân tố là tương tự thì nên chọn giải pháp đơn giản nhất.

**Giai đoạn 3:** Mô tả từng điều trừu tượng (chưa rõ ràng) 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 hoá nó. Các sai sót và khiếm khuyết trong mức thiết kế ban đầu sẽ được phát hiện và được điều chỉnh cho phù hợp tại các mức chi tiết thiết kế tiếp theo.

Quá trình khắc phục khiếm khuyết này sẽ được lặp lại cho từng phần trừu tượng từ mức thiết kế ban đầu cho đến khi một đặc tả thiết kế chi tiết cho từng phần trừu tượng kết thúc. *Nên phân chia ra các phần nhỏ ứng với thiết kế rồi tổ hợp lại, sao cho việc mô tả chi tiết các phần nhỏ đó chỉ trong khoảng một trang giấy.*

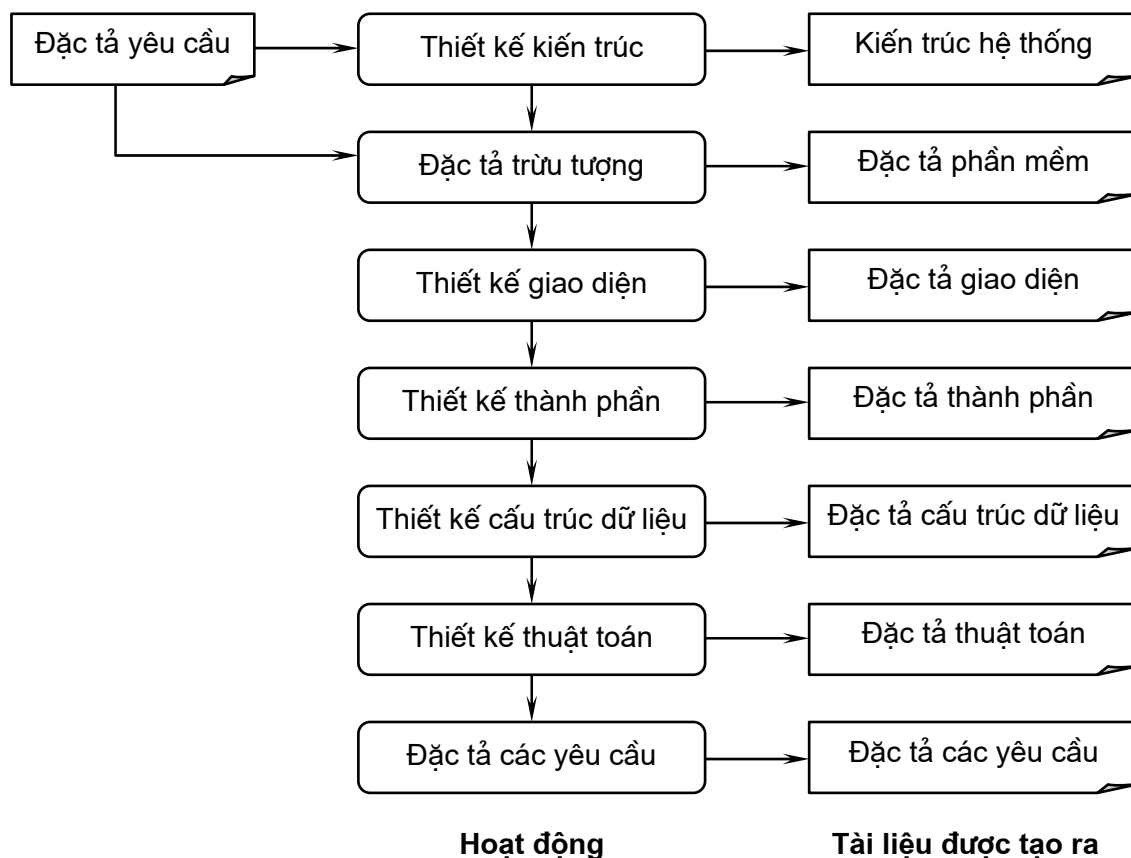
#### ***4.1.1. Quá trình thiết kế.***

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 (từ “hình thức” ở đây có nghĩa là mang tính mô tả được hệ thống trong thực tế) có trước đây của quá trình đó. Nhà thiết kế phải bắt đầu với một bản phác thảo hết sức không hình thức rồi sau đó tinh chế nó, thêm vào đó các thông tin để là cho 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 là 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 ra được những sai sót ở giai đoạn trước. Do đó phải quay trở lại để tinh chế. Thông thường thì người ta bắt đầu giai đoạn sau ngay trước khi giai đoạn trước kết thúc đơn giản là để lui quá trình tinh chế. Hình vẽ dưới đây nêu các hoạt động của quá trình thiết kế và các sản phẩm của nó. Các giai đoạn là khá tùy ý nhưng nó làm cho quá trình thiết kế trở nên nhìn thấy được và từ đó dễ quản lý được.



**Hình 5: Các hoạt động thiết kế và sản phẩm của thiết kế.**

Thành quả của mỗi hoạt động thiết kế là một bản 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 thành phần nào đó của hệ thống phải được thực hiện như thế nào. Khi quá trình thiết kế tiến triển thì các yêu cầu ngày càng được bổ sung vào bản đặc tả đó. Các kết quả cuối cùng là các đặc tả về 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 lại được triển khai ở các mức chi tiết khác nhau trong diễn biến của quá trình thiết kế.

#### **4.1.1.1. Các hoạt động cốt yếu trong việc thiết kế một hệ thống phần mềm lớn**

**1. Thiết kế kiến trúc:** Các hệ con tạo nên hệ tổng thể và các quan hệ của chúng là được phân hoạch rõ ràng và ghi thành tài liệu.

**2. 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 phải tuân theo cũng được hỗ trợ.

**3. Thiết kế giao diện:** Ở đây bạn đọc không nên hiểu “giao diện” chỉ là những gì hiển thị trên màn hình, mà phải hiểu rằng đó có thể là tương tác giữa các thành phần

trong hệ thống với nhau. Giao diện với từng hệ con khác cũng 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 đến những gì được diễn ra bên trong của hệ con đó (theo kiểu “hộp đen”).

**4. Thiết kế các thành phần:** Các dịch vụ được cung cấp bởi hệ con được phân chia thành các thành phần hợp thành của hệ con đó.

**5. Thiết kế cấu trúc dữ liệu:** Các cấu trúc dữ liệu được dùng trong việc thực hiện hệ thống được thiết kế chi tiết và được đặc tả ở đây.

**6. Thiết kế thuật toán:** Các cách thức (phương pháp xử lý) 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 xác định một cách rõ ràng và đều có thể chuyển đổi (ánh xạ) một cách trực tiếp vào các thành phần của ngôn ngữ lập trình, chẳng hạn như các gói (packets), các thủ tục (procedures) và các hàm (functions).

Phương pháp tiếp cận thường xuyên được khuyến khích sử dụng là phương pháp tiếp cận *từ trên xuống* (top down): Vấn đề lớn được phân chia một cách đệ quy thành các vấn đề con cho đến khi các vấn đề dễ giải quyết được xác định rõ ràng. Trong quá trình này người thiết kế không nhất thiết phải phân rã tất cả các thành phần trừu tượng (nghĩa là vấn đề này còn phức tạp mà cách giải quyết là chưa xác định rõ) khi mà bằng kinh nghiệm họ đã biết chắc chắn rằng có thể hoàn toàn xây dựng được. Do đó họ có thể tập trung sức lực vào các thành phần đáng xét nhất.

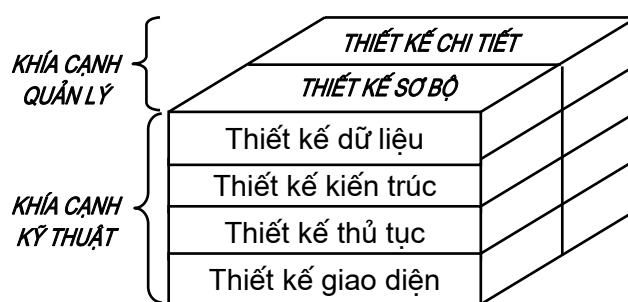
Chú ý rằng khi mà phương pháp hướng đối tượng được chấp nhận thì phương pháp từ trên xuống sẽ ít hiệu quả. Khi đó người thiết kế sử dụng các đối tượng sẵn có để làm khung thiết kế.

*Theo quan điểm quản lý dự án, thiết kế phần mềm được tiến hành theo 2 bước:*

**Bước 1- Thiết kế sơ bộ:** Quan tâm tới việc chuyển hoá các yêu cầu thành kiến trúc dữ liệu và các thành phần phần mềm.

**Bước 2- Thiết kế chi tiết:** Tập trung vào việc làm mịn biểu diễn kiến trúc để dẫn tới cấu trúc dữ liệu chi tiết và biểu diễn các quy trình tính toán và xử lý của 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 phân biệt. Thiết kế giao diện lập ra cách bố trí và cơ chế tương tác người-máy (HCI – human computer interface). Mối quan hệ giữa các khía cạnh kỹ thuật và quản lý của thiết kế được minh hoạ trong hình vẽ dưới đây.



**Hình 6: Quan hệ giữa khía cạnh kỹ thuật và khía cạnh quản lý trong thiết kế**

#### 4.1.1.2. Việc mô tả thiết kế.

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

Thiết kế cần được mô tả sao cho đạt được ở mức độ sau:

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

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

#### 4.1.1.3. Phương pháp thiết kế.

Trong nhiều tổ chức, việc thiết kế phần mềm vẫn còn là một quá trình tự học. Bằng cách cho một tập hợp các yêu cầu (thường là bằng ngôn ngữ tự nhiên) người ta có một thiết kế không hình thức. Bắt đầu việc mã hoá và thiết kế được cải biến trong quá trình thực hiện. Khi giai đoạn thực hiện kết thúc thì thiết kế đã bị biến đổi sản phẩm so với đặc tả ban đầu đến mức mà các tài liệu thiết kế ban đầu là một mô tả hoàn toàn khác với cái được tạo ra.

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

Phương pháp cấu trúc được dùng rộng rãi trong những năm đầu của những năm 1980. Nó đã được dùng thành công trong nhiều dự án lớn, nó làm giảm giá thành một cách đáng kể, sử dụng được các khái niệm chuẩn và đảm bảo rằng việc thiết kế tuân theo một chuẩn nhất định. Các công cụ CASE (Computer Aided Software Engineering – thiết kế phần mềm có máy tính hỗ trợ) đã được dùng để trợ giúp cho phương pháp này.



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 lược đồ cấu trúc.
- *Nhìn nhận quan hệ thực thể:* Mô tả cấu trúc dữ liệu logic thường dùng, đề cập đến đặc tả dữ liệu quan hệ thực thể.
- *Nhìn nhận dòng dữ liệu:* Về lược đồ dòng dữ liệu.

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

Để đảm bảo chất lượng 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 nên đưa ra cách tổ chức theo cấp bậc để dùng cách kiểm soát thông minh trong số các thành phần phần mềm.
2. Thiết kế nên theo các module, tức là phần mềm nên được phân hoạch một cách logic thành các thành phần thực hiện chức năng hay các chức năng con xác định.
3. Thiết kế nên chứa cách biểu diễn phân biệt và tách biệt giữa dữ liệu và thủ tục.
4. Thiết kế nên dẫn tới các module (như chương trình con hay thủ tục) nêu ra các đặc trưng chức năng đặc biệt.
5. Thiết kế nên dẫn đến giao diện là rút gọn độ phức tạp của việc nối ghép lại giữa các module và với môi trường bên ngoài.
6. Thiết kế nên được hướng theo cách dùng một phương pháp lặp lại được điều khiển bởi thông tin có trong 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 xét duyệt thấu đáo.

Như vậy, mỗi phương pháp thiết kế phần mềm đều đưa vào những phương pháp trực cảm và lý pháp duy nhất, cũng như một cách nhìn thiên cận thế nào đó về cái gì đặc trưng cho chất lượng thiết kế

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

1. Một cơ chế để chuyển hoá từ biểu diễn miền thông tin thành biểu diễn thiết kế
2. Một kí pháp để biểu diễn các thành phần chức năng và dao diện của chúng
3. Các trực cảm để làm mịn và 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:

- Trừu tượng
- Modul
- Kiến trúc phần mềm.
- Cấp bậc điều khiển
- Cấu trúc dữ liệu
- Thủ tục phần mềm
- Che dấu thông tin

Tập các khái niệm thiết kế nền tảng đã tiến hoá 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

Theo M.A.Jackson: “Cái bắt đầu của sự khôn ngoan đối với kỹ sư phần mềm là thừa nhận sự bắt đầu làm chương trình và 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”.

##### 4.1.2. Chiến lược thiết kế.

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

##### 4.1.2.1. Thiết kế hướng 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.

Ban đầu, ta coi yêu cầu mức cao nhất của hệ thống là một chức năng duy nhất cần phải thực hiện. Sau đó, ta trả lời cho câu hỏi “Để thực hiện chức năng trên thì cần phải làm các công việc gì?” – từ *công việc* trong câu hỏi trên được coi là chức năng con của chức năng trên. Thực hiện xong các chức năng con cũng là thực hiện xong chức năng cha. Hệ thống được phân rã dần dần, và được làm mịn. Hình ảnh của hệ thống sẽ được xây dựng theo các bước trên.

##### 4.1.2.2. Thiết kế hướng đối tượng

Hệ thống được nhìn nhận như một bộ các đối tượng (chứ không phải là một tập hợp các chức năng). Hệ thống được phân tán, mỗi đối tượng có thông tin và trạng thái

của riêng nó. Đối tượng là một bộ các thuộc tính xác định trạng thái của đối tượng đó và các phép toán thực hiện trên đó. Mỗi đối tượng là một khách thể của một lớp mà *lớp được xác định bởi thuộc tính và các phép toán* của nó. Nó được thừa kế từ một vài lớp đối tượng cấp cao hơn, sao cho định nghĩa nó chỉ cần nêu đủ các khác nhau giữa nó và các lớp cao hơn nó. Các đối tượng liên lạc với nhau chỉ bằng trao đổi các *thông báo*. Trong thực tế, hầu hết các liên lạc được thực hiện giữa các đối tượng bằng cách nối đối tượng này với 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. Gần đây theo cách thiết kế này, người ta đã phát triển nhiều hệ thống cấu tạo bởi nhiều thành phần độc lập và có tương tác với nhau.

Sự thật, các hệ phần mềm lớn 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 trong hệ thống. Chẳng 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 phải là loại bỏ nhau. Kỹ sư phần mềm sẽ chọn ra cách tiếp cận thích hợp nhất trong từng giai đoạn thiết kế. Nhìn ở mức tổng thể thì hệ thống như là một bộ các đối tượng (chứ không phải là một bộ các chức năng), cho nên ở mức trừu tượng cao thì cách tiếp cận hướng đối tượng là thích hợp hơn. Đến mức chi tiết thì một cách tự nhiên hơn nên xem 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 có thể xem nó như là một hệ con.

Rất nhiều hệ thống, đặc biệt là hệ thống thời gian thực được nhúng (vào một hệ thiết bị vật chất có thực) được cấu tạo như một hệ gồm một bộ các quá trình hoạt động song song và 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 phản ứng 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 và kiểm tra và thử nghiệm hơn là các hệ thống song song. Sự phụ thuộc về thời gian giữa các quá trình là khó hình thức hoá, khó khống chế và thử nghiệm.

Do đó, quá trình thiết kế nên được xem như là một hoạt động gồm 2 giai đoạn:

*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 hướng chức năng hoặc cách nhìn hướng đối tượng.

*Giai đoạn 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. Chắc rằng sự quyết định về tính song song nên là ở giai đoạn này chứ không phải là các giai đoạn sớm hơn trong quá trình thiết kế.

### 4.1.3. Chất lượng thiết kế.

Không có cách nào hay để xác định được thế nào là một thiết kế tốt, phụ thuộc vào từng ứng dụng và vào yêu cầu của dự án. *Một thiết kế tốt hẳn là thiết kế mà nó cho phép sản sinh ra mã lệnh hữu hiệu*, nó có thể là một thiết kế tối thiểu mà theo đó việc thực hiện là càng chặt càng tốt, hoặc đó 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 biến chức năng và việc thêm chức năng mới. Do đó, thiết kế phải dễ hiểu và việc thay đổi chỉ có hiệu ứng cục bộ. Các thành phần trong thiết kế phải kết dính (cohesive) theo nghĩa là tất cả các phần trong đó phải có một quan hệ logic chặt chẽ. Các thành phần ấy phải được ghép nối sao cho vẫn có sự linh hoạt mềm dẻo (ghép nối lỏng lẻo). Sự ghép nối là một độ đo tính độc lập của các thành phần. Nối ghép càng lỏng lẻo thì càng dễ thích nghi.

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

#### 4.1.3.1. Sự kết dính (cohesion)

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

Constantine và Yourdon định nghĩa ra 7 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 lại bị bó vào một thành phần.
- 2- *Hội hợp logic*: Các thành phần cùng thực hiện chức năng tương tự, chẳng hạn như vào, xử lý lỗi... là được đặt vào trong một thành phần.
- 3- *Kết dính theo thời điểm*: Tất cả các thành phần cùng hoạt hoá một lúc, chẳng hạn như khởi sự và kết thúc... là được bó vào với nhau.
- 4- *Kết dính thủ tục*: Các phần tử trong thành phần được ghép lại trong một dãy điều khiển.
- 5- *Kết dính truyền thông*: Các phần trong thành phần cùng thao tác trên cùng một dữ liệu và đưa ra cùng một dữ liệu ra.
- 6- *Kết dính tuần tự*: Trong một thành phần, “ra” của phần tử này là “vào” của phần tử khác.
- 7- *Kết dính chức năng*: Mỗi phần của thành phần đều cần thiết để thi hành một chức năng nào đó.

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

Một đối tượng kết dính có thể là một thực thể đơn. Tất cả các phép toán trên thực thể đó đều nằm trong thực thể đó (mang tính nội tại). Do vậy có thể xác định một lớp kết dính nữa là:

8- *Kết dính đối tượng*: Mỗi phép toán cho một chức năng, chức năng này cho phép các thuộc tính của đối tượng được cải biến, kiểm soát và sử dụng như là cơ sở cho việc cung cấp các dịch vụ.

#### 4.1.3.2. 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 chặt chẽ sẽ có độ nối ghép cao giữa các đơn vị, các đơn vị phụ thuộc lẫn nhau. Hệ thống nối ghép lỏng lẻo sẽ làm cho các đơn vị độc lập hoặc tương đối độc lập với nhau.

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

Có lẽ tính 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 đến một dạng khác của ghép nối.

#### 4.1.3.3. Sự hiểu được (understandability)

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

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

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

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

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

Từ “*phức tạp*” ở đây được dùng theo nghĩa không hình thức. Độ phức tạp ám chỉ nhiều quan hệ giữa các thành phần khác nhau của thành phần thiết kế đó và một cấu

trúc logic phức tạp mà nó dính líu đến độ sâu lồng nhau của cấu trúc *if-then-else*. Các thành phần phức tạp là khó hiểu, vì thế người thiết kế hẳn là nên là cho các thiết kế thành phần là càng đơn giản càng tốt.

Đa số công việc để đo chất lượng thiết kế được tập trung vào việc cố gắng đo độ phức tạp của từng thành phần và từ đó thu được một vài độ đo về sự dễ hiểu của thành phần đó. Độ phức tạp, theo một khía cạnh nào đó, phản ánh độ dễ hiểu, chẳng hạn như tổ chức dữ liệu và kiểu mô tả cách thiết kế. Các số đo độ phức tạp có thể chỉ cung cấp một số chỉ số cho độ dễ hiểu của thành phần.

Sự thừa kế trong một thiết kế hướng đối tượng phản ánh độ dễ hiểu. Nếu sự thừa kế được dùng để gắn kết các chi tiết thiết kế thì thiết kế sẽ dễ hiểu hơn. Mặt khác, nếu sử dụng sự thừa kế đòi hỏi người đọc thiết kế phải nhìn nhiều lớp đối tượng khác nhau trong sơ đồ (hay tôn ti) thừa kế thì độ dễ hiểu của thiết kế là được rút gọn.

#### 4.1.3.4. Sự thích nghi được (adaptability)

Nếu một thiết kế nhằm được bảo trì thì nó phải sẵn sàng thích nghi được. Dĩ nhiên điều này say ra rằng các thành phần của nó lên được ghép nối một cách lỏng lẻo (thể hiện tính linh động). Tuy nhiên sự thích nghi được cũng có nghĩa là tư liệu của thiết kế đó phải được viết ra sao cho người dùng dễ đọc.

Một thiết kế dễ thích nghi hẳn là có mức nhìn thấy được cao. Mỗi quan hệ rõ ràng giữa các mức khác nhau của thiết kế. Người đọc khi đọc bản thiết kế phải thấy được các biến hiện liên quan sao cho lược đồ cấu trúc biểu diễn biểu đồ luồng dữ liệu.

Cần phải dễ dàng kết hợp chặt chẽ các biến đổi của thiết kế trong toàn bộ bản thiết kế. Vì nếu không như vậy thì những sự thay đổi của thiết kế có thể sẽ không được đưa vào trong các mô tả liên quan. Tư liệu thiết kế có thể trở nên không thống nhất. Các biến đổi tiếp theo là khó thực hiện (điều này cho thấy thành phần thiết kế này là ít có tính thích nghi được) ví sự cải biến đó không thể đưa vào vì như vậy có thể là mất tính nhất quán của tư liệu thiết kế.

Để có độ thích nghi tối ưu thì một thành phần đều phải tự chứa. Một thành phần có thể là ghép nối lỏng lẻo theo nghĩa chỉ liên hệ với các thành phần khác thông qua việc truyền các thông báo (tín hiệu - message). Điều này không giống như là sự tự chứa vì rằng các thành phần này có thể dựa trên các thành phần khác, chẳng hạn các chức năng hệ thống hoặc các chức năng xử lý lỗi. Sự thích nghi với một thành phần có thể dính líu đến sự thanh đối các thành phần trong đó mà nó dựa trên các chức năng ngoài đối tượng nên đặc tả các chức năng ngoại này cũng phải xét đến sự cải biến đó.

Muốn là tự chứa một cách hoàn toàn thì mỗi thành phần không nên sử dụng các thành phần khác được xác định ngoại lai. Tuy nhiên điều này lại mâu thuẫn với kinh nghiệm cho thấy các thành phần hiện có nên có tính sử dụng lại được (kế thừa). Vậy nên cần có sự cân đối giữa tính ưu việt của sử dụng lại các thành phần và sự mất tính thích nghi của thành phần đó.

Một trong những tính ưu việt nhất của thừa kế trong thiết kế hướng đối tượng là các thành phần này có thể sẵn sàng thích nghi được. Cơ chế thích nghi được này không dựa trên việc cải biến các thành phần đó mà dựa trên việc tạo ra một thành phần mới có thừa kế các thuộc tính và phép toán của thành phần gốc. Chỉ các thuộc tính và phép toán cần phải biến đổi mới được cải biến. 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 vì sao các ngôn ngữ hướng đối tượng là hữu hiệu đến vậy trong việc tạo mẫu nhanh. Tuy nhiên với các hệ thống tồn tại lâu dài thì vấn đề thừa kế là ngày càng có nhiều thay đổi cần thực hiện. Sơ đồ (mạng lưới) kế thừa ngày càng phức tạp. Các chức năng thường là được nhân bản ở các điểm trong mạng là các thành phần khó mà có thể hiểu được. Kinh nghiệm của lập trình hướng đối tượng đã chỉ ra rằng mạng kế thừa phải thường xuyên được rà soát theo chu kỳ và phải được cấu trúc lại nhằm thu gọn độ phức tạp của nó và giảm bớt các bản sao chức năng. Rõ ràng điều đó làm tăng chi phí cho thay đổi hệ thống.

## 4.2. Các mô hình thiết kế và đặc tả phần mềm.

### 4.2.1. Thiết kế hướng đối tượng.

#### 4.2.1.1 Cách tiếp cận hướng đối tượng

Che dấu thông tin là chiến lược thiết kế dấu càng nhiều thông tin trong các thành phần càng hay. Cái đó ngầm hiểu rằng việc kết hợp điều khiển logic và cấu trúc dữ liệu được thực hiện trong thiết kế càng chậm càng tốt. Liên lạc qua các thông tin trạng thái dùng chung (các biến tổng thể) là ít nhất, nhờ vậy khả năng hiểu được nâng lên. Thiết kế là tương đối dễ thay đổi một thành phần không thể không dự kiến các hiệu ứng phụ trên các thành phần khác.

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 chức năng. Các đối tượng có một trạng thái được che dấu và các phép toán trên trạng thái đó. Thiết kế biểu thị các dịch vụ được yêu cầu cùng với những hỗ trợ cung cấp bởi các đối tượng có tương tác với nó.

Ví dụ trong “Hệ thống bán hàng”, khách hàng ta coi là một đối tượng. Khi đó các thuộc tính của đối tượng này đó là: Họ tên, địa chỉ, điện thoại, số tài khoản ... Các dịch vụ (hay các chức năng, phép toán) mà đối tượng này thực hiện là: *Xem hàng, thoả thuận phương thức thanh toán, đặt hàng, mua hàng trực tiếp, nhận hoá đơn* ... Khi đó ta cần thiết kế ra một đối tượng “Khách hàng” có đầy đủ các thuộc tính và chức năng như trên. Đối tượng này cũng có thể tương tác với các đối tượng khác như “Nhân viên bán hàng”, “Thủ kho” ... trong toàn bộ hệ thống. Điều này cho thấy các thao tác trong hệ thống đều xuất phát từ chính bản thân các đối tượng đó chứ không phải là từ chức

năng lớn trong hệ thống yêu cầu. Cách tiếp cận này khác hẳn với cách thiết kế hướng chức năng mà ta sẽ xem xét dưới đây.

#### **4.2.1.2. Đặc trưng của thiết kế hướng đối tượng**

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

#### **4.2.1.3. Các ưu điểm của thiết kế hướng đối tượng.**

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

Các đối tượng là rất tốt cho việc sử dụng lại (do tính độc lập của chúng). Một thiết kế sau có thể dùng lại các đối tượng đã có trong bản thiết kế trước đó.

Có một vài lớp hệ thống thể hiện phản ánh quan hệ rõ ràng giữa các thực thể có thực (chẳng hạn như các thành phần dùng chung) với các đối tượng điều khiển trong hệ thống. Điều này làm cho thiết kế trở lên dễ hiểu, dễ tiếp cận hơn.

#### **4.2.1.4. Nhược điểm của thiết kế hướng đối tượng.**

Sự nhận định về các đối tượng trong hệ thống một cách hợp lý là một điều khó. Cách nhìn nhận tự nhiên nhiều hệ thống là cách nhìn chức năng và thích nghi với cách nhìn hướng đối tượng đôi khi là khó khăn. Đây là một phương thức còn khá mới mẻ trong phân tích và thiết kế hệ thống thông tin.

#### **4.2.1.5. Phân biệt giữa thiết kế hướng đối tượng và lập trình hướng đối tượng.**

*Thứ nhất*, dễ nhầm lẫn hai khái niệm này. Ngôn ngữ lập trình hướng đối tượng cho phép cài đặt thực tế các đối tượng và cung cấp các lớp đối tượng cùng với sự thừa kế.

*Thứ hai*, thiết kế hướng đối tượng là một chiến lược, không phụ thuộc vào một ngôn ngữ thực hiện cụ thể nào. Các ngôn ngữ lập trình hướng đối tượng và các khả năng cung cấp các cơ chế như bao gói, kế thừa, đa xạ, ... giữa các đối tượng làm cho việc thiết kế hướng đối tượng được thực hiện một cách đơn giản hơn. Tuy nhiên, một thiết kế hướng đối tượng cũng có thể được thực hiện trên một ngôn ngữ ít hỗ trợ về lập trình hướng đối tượng như PASCAL chẳng hạn, nó không cần cung cấp cơ chế thực hiện bao gói tốt như ngôn ngữ lập trình JAVA.



Việc chấp nhận thiết kế hướng đối tượng như là một chiến lược hữu hiệu dẫn đến sự phát triển của phương pháp thiết kế hướng đối tượng.

Ví dụ tiếp theo, Ada không phải là ngôn ngữ lập trình hướng đối tượng vì nó không trợ giúp sự kế thừa của các lớp. Nhưng lại có thể thực hiện các đối tượng trong Ada bằng cách sử dụng các gói hoặc các nhiệm vụ (tasks), do đó Ada là ngôn ngữ có thể được dùng để mô tả các thiết kế hướng đối tượng.

*Thứ 3*, phương pháp thiết kế hướng đối tượng vẫn còn là tương đối chưa chín muồi và đang thay đổi nhanh chóng. Phương pháp hiện đang được sử dụng rộng rãi là phương pháp dựa trên sự phân rã chức năng.

#### **4.2.2. Thiết kế hướng chức năng.**

##### **4.2.2.1. Cách tiếp cận hướng chức năng.**

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

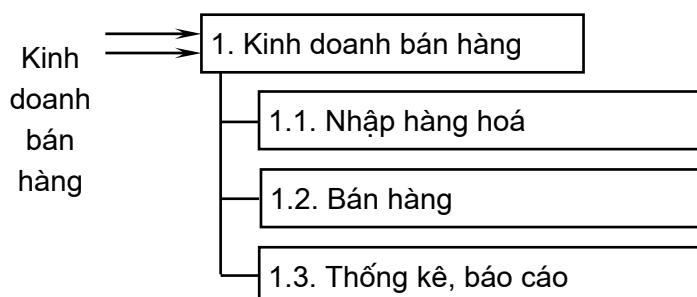
Trong thiết kế hướng chức năng, người ta dùng các *biểu đồ dòng dữ liệu* (mà mô tả việc xử lý dữ liệu logic), *các lược đồ cấu trúc* (chỉ ra cấu trúc của phần mềm) và các *mô tả việc thiết kế chi tiết* (đặc tả thiết kế chi tiết). Khái niệm *dòng dữ liệu* đang hướng tới thích hợp hơn cho việc sử dụng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

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

Xét ví dụ về “Hệ thống quản lý kinh doanh bán hàng”. Như vậy chức năng chính là “Kinh doanh bán hàng”, nhưng để thực hiện chức năng trên thì ta cần phải tiến hành các công việc sau: “Nhập hàng hoá”, “Bán hàng” và “Thống kê báo cáo”. Mỗi công việc đó ta coi chúng như là các chức năng và nó là chức năng con của chức năng “Kinh doanh bán hàng”. Thực hiện xong các chức năng con thì cũng chính là hoàn chỉnh chức năng cha. Tuy nhiên, phân rã như vậy chưa đủ chi tiết và dễ hiểu. Ta cần phải tiến hành việc phân rã ra thành nhiều chức năng con ở mức thấp hơn nữa. Sao cho dẫn tới việc

đặc tả chương trình (phần mềm). Một câu hỏi đặt ra là phân rã đến mức nào thì dừng lại. Theo kinh nghiệm thì việc phân rã có thể dừng lại khi ta đã tách thành các chức năng mà có thể thực hiện được ngay (dễ thực hiện) và việc đặc tả nó trong khuôn khổ một trang giấy, như đã đề cập ở chương 3 (đặc tả).

Sơ đồ phân rã chức năng có thể được vẽ như sau:



Thiết kế hướng chức năng gắn với chi tiết của thuật toán gắn với chức năng đó nhưng các thông tin trạng thái của hệ thống là không bị che dấu. Điều này có thể gây ra vấn đề sau: Vì rằng một chức năng có thể thay đổi trạng thái theo một cách mà chức năng khác không ngờ tới. Do vậy có thể gây ra các tương tác bất ngờ cho các chức năng khác.

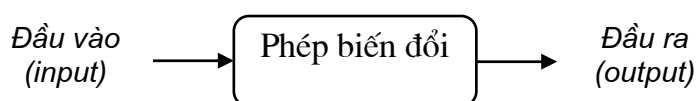
Cách tiếp cận chức năng để thiết kế là tốt nhất khi mà khối lượng thông tin trạng thái hệ thống hệ thống thường được làm nhỏ nhất và thông tin dùng chung trong hệ thống là rõ ràng.

#### 4.2.2.2. Biểu đồ dòng dữ liệu (Data Flow Diagram - DFD).

*Biểu đồ dòng dữ liệu* (còn gọi là biểu đồ luồng tiến trình) chỉ ra cách thức biến đổi dữ liệu và thành dữ liệu ra thông qua một dãy các phép biến đổi. Đó là cách đơn giản và hữu ích nhất để mô tả một hệ thống mà không cần một sự chú thích gì thêm. Bước thứ nhất của thiết kế hướng chức năng là phát triển biểu đồ dòng dữ liệu hệ thống. Biểu đồ này không nhất thiết bao gồm các thông tin điều khiển nhưng ta lên lập tư liệu cho các phép biến đổi này.

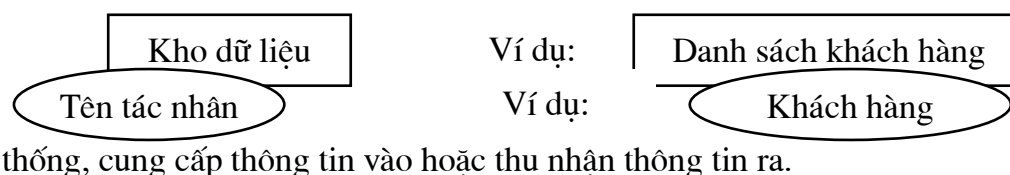
Biểu đồ dòng dữ liệu là một phân hợp nhất của một số các phương pháp thiết kế và các công cụ CASE thường trợ giúp cho việc tạo ra biểu đồ dòng dữ liệu. Thông thường, ta dùng các ký pháp sau (xin đưa ra một số bộ ký pháp thực tế hay được sử dụng):

1. *Các hình chữ nhật góc tròn*: Biểu diễn một *phép biến đổi* (chức năng) dòng dữ liệu vào thành dòng dữ liệu ra, sự biến đổi được chỉ ra bằng chính tên của hình đó.



2. *Hình chữ nhật*: Biểu diễn *kho dữ liệu*, dữ liệu được nói rõ qua tên (nhãn - label) của hình đó.

3. *Các hình tròn hay hình bầu dục*: Biểu diễn *tác nhân* (giao tác) tác động với hệ

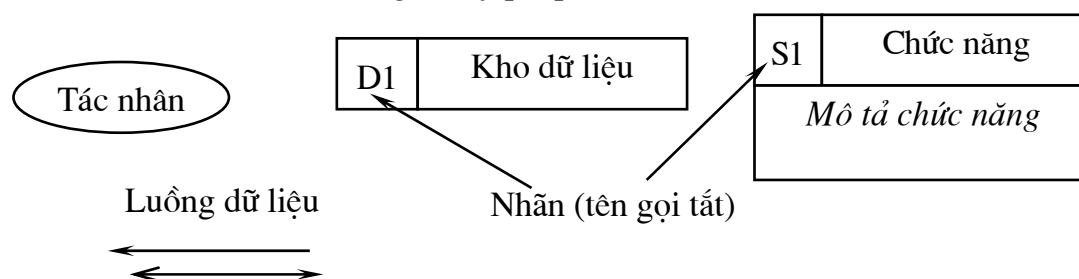


4. *Mũi tên*: Chỉ hướng dòng dữ liệu.

5. *Các khuyên tròn*: Dùng để nối các dòng dữ liệu.

6. *Các lời chú thích đính kèm với các ký pháp trên*: Giải thích cho ký pháp đó.

Một số tài liệu khác thì lại dùng bộ ký pháp sau:



Ví dụ sau đây minh hoạ một sơ đồ luồng dữ liệu của chức năng “Bán hàng” trong hệ thống “Kinh doanh bán hàng”.

Mô tả hoạt động trong thực tế ta thấy: Khi một khách hàng bước vào cửa hàng, thì họ có thể muốn mua hàng hoặc chỉ *xem hàng hoá* (tuy nhiên việc xem và lựa chọn hàng hoá là tất yếu phải có). Sau khi đã lựa chọn xong hàng, khách hàng sẽ tiến hành *thoả thuận thanh toán* với nhân viên bán hàng (trả tiền sau, thanh toán trực tiếp bằng tiền mặt, chuyển khoản, ...). Tiếp theo, nhân viên bán hàng *lập hoá đơn* và *giao hàng* cho khách hàng, cập nhật thông tin về hàng hoá.

Phân tích chi tiết hơn, ta thấy rằng: Hệ thống mà chúng ta xây dựng có thể hỗ trợ được việc “xem hàng hoá” bằng cách cung cấp các thông tin về hàng hoá như: Tên hàng hoá, đơn giá bán, nơi sản xuất, ... điều này có được bằng cách lưu thông tin về hàng hoá trong kho dữ liệu “Hàng hoá”. Thông tin về các mặt hàng được kết xuất bằng cách “đọc” (read) từ kho dữ liệu ra.

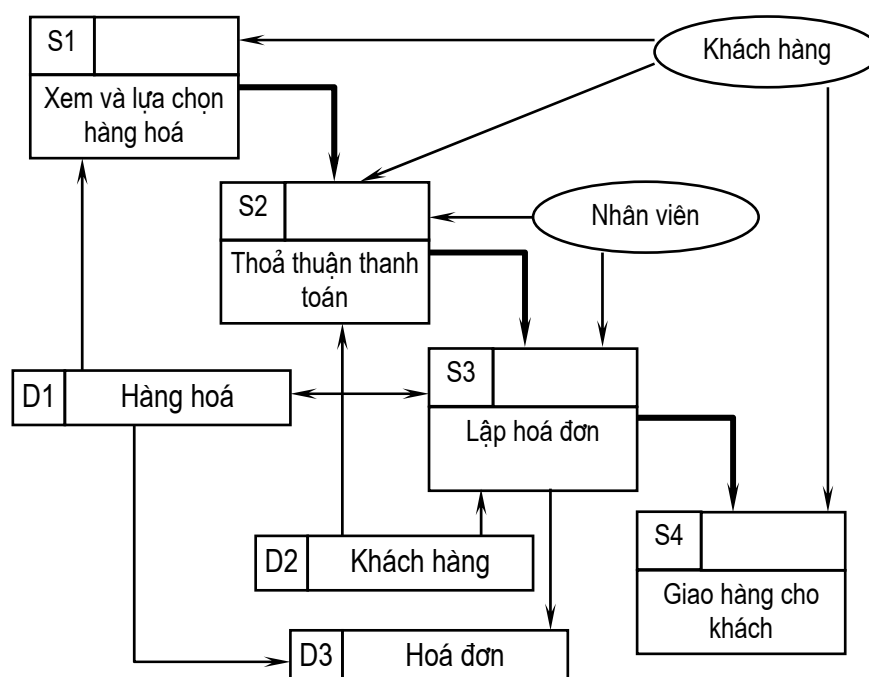
Với chức năng “thoả thuận thanh toán” thì phần lớn là do con người thực hiện. Hệ thống chỉ có thể hỗ trợ ở mức độ sau: Đối với khách hàng quen, thì thông tin về khách hàng đã được lưu trong kho dữ liệu “khách hàng” với các thông tin: họ tên khách hàng,

địa chỉ, điện thoại, số tài khoản, ... và có thể đưa ra lời nhắc nhở về người khách này: nếu trả tiền sau thì tài khoản mà người khách này có đủ khả năng thanh toán không. Còn với khách hàng mới thì chỉ có thể cập nhật thông tin về khách hàng vào trong kho dữ liệu “khách hàng”.

Với chức năng “lập hoá đơn” thì hệ thống hoàn toàn có thể hỗ trợ việc in ra một hoá đơn và cập nhật thông tin về số lượng hàng hoá trong kho do đã bán đi một lượng hàng hoá theo sự lựa chọn của khách hàng.

Với chức năng “Giao hàng cho khách” thì hầu như hệ thống thông tin không hỗ trợ gì nhiều. Chủ yếu vẫn do con người thực hiện.

Sơ đồ luồng dữ liệu có thể vẽ theo dạng sau:



Chú thích về luồng dữ liệu:

- Đi từ S1 đến S2: Thông tin về hàng hoá: tên hàng, số lượng...
- Đi từ S2 đến S3: Thông tin về khách hàng và hàng hoá: tên khách hàng, địa chỉ, tài khoản, tên hàng, số lượng, đơn giá bán...
- Đi từ S3 đến S4: Thông tin về khách hàng và hàng hoá: tên khách hàng, địa chỉ, tài khoản, tên hàng, số lượng, đơn giá bán...
- Đi từ D1 đến S1: Thông tin về hàng hoá: tên hàng, số lượng, đơn giá bán, nơi sản xuất...
- Đi từ D1 đến D3: Thông tin về hàng hoá: tên hàng, số lượng bán, đơn giá bán
- Đi từ D2 đến S2: Thông tin về khách hàng: tên khách hàng, địa chỉ, tài khoản...
- Đi từ D2 đến S3: Thông tin về khách hàng: tên khách hàng, địa chỉ, tài khoản...

- *Đi từ S3 đến D3*: Thông tin về khách hàng và hàng hoá: tên khách hàng, địa chỉ, tài khoản, tên hàng, số lượng, đơn giá bán...
- *Đi từ S3 đến D1*: Thông tin về hàng hoá: tên hàng, số lượng còn lại ...

**Nhận xét:** Với sơ đồ luồng dữ liệu, ta có thể thấy được công việc tuần tự được thực hiện; luồng dữ liệu luân chuyển giữa các chức năng và các kho dữ liệu; thông tin chi tiết về các trường dữ liệu mà ta cần lưu trữ; các trạng thái cập nhật dữ liệu: Tạo mới (creat), đọc (read), cập nhật (update), ... Từ đó mà ta có thể tiến hành các bước tiếp theo là thiết kế dữ liệu và thiết kế module chương trình.

#### **Tạo ra Mô hình luồng dữ liệu (Data Follow Diagram - DFD)**

Biểu đồ luồng dữ liệu làm cho người kỹ sư phần mềm phát triển được các mô hình về miền thông tin và miền chức năng cùng lúc. Khi DFD được làm mịn thành những mức chi tiết lớn hơn, thì người phân tích thực hiện việc phân tách chức năng không tưởng mình về hệ thống, do đó hoàn thành nguyên tắc phân tích vận hành tứ tự cho chức năng. Đồng thời, việc làm mịn DFD còn gây ra việc làm mịn tương ứng cho dữ liệu khi nó chuyển qua các tiến trình cụ thể hoá ứng dụng.

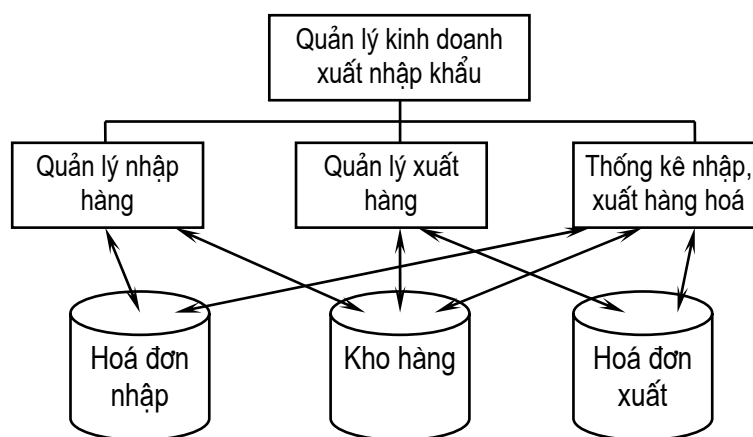
Vài hướng dẫn đơn giản có thể giúp được rất nhiều trong khi dẫn ra biểu đồ luồng dữ liệu: (1) biểu đồ luồng dữ liệu mức 0 nên mô tả cho phần mềm/ hệ thống như một bong bóng; (2) các cái vào và ra chủ yếu nên được ghi chép cẩn thận; (3) việc làm mịn nên bắt đầu bằng việc cô lập các tiến trình, các sự vật dữ liệu và kho ứng cử viên được biểu diễn ở mức tiếp; (4) tất cả các mũi tên và bong bóng nên được đánh nhãn bằng các tên có nghĩa; (5) tính liên tục của luồng thông tin phải được duy trì từ mức nọ sang mức kia, và (6) một lúc nên làm mịn cho một bong bóng. Có một khuynh hướng tự nhiên là ahy làm rối rắm biểu đồ luồng dữ liệu. Điều này xuất hiện khi người phân tích cố gắng bày tỏ quá nhiều chi tiết quá sớm hay biểu diễn các khía cạnh thủ tục của phần mềm thay vì luồng thông tin.

#### **4.2.2.3. Lược đồ cấu trúc.**

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

Mỗi thành phần chức năng được biểu diễn trên lược đồ cấu trúc như là một hình chữ nhật. Thứ bậc được biểu diễn bằng cách nối các hình chữ nhật với các đường. Thông tin vào và thông tin ra cho một thành phần được chỉ ra bởi việc dùng mũi tên có thông tin trên đó. Mũi tên đi vào một hộp biểu thị thông tin đi vào, mũi tên đi ra từ một hộp biểu thị thông tin ra. Các kho dữ liệu được biểu diễn bằng một khối hình trụ (để tránh nhầm lẫn với các ký hiệu đã dùng trong sơ đồ luồng dữ liệu) và hình bình hành để biểu diễn thông tin vào.

Nhìn chung, lược đồ cấu trúc cho ta thấy được mô hình tổng quan của hệ thống. Lược đồ này nên đưa ra ở mức thiết kế tổng quát. Sau này, với từng mô hình thực tế, nếu đã đủ dễ hiểu thì ta không cần thiết phải đưa ra lược đồ này. Hình vẽ sau minh hoạ một lược đồ cấu trúc.

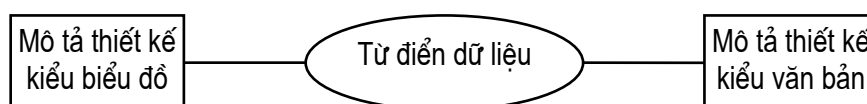


#### 4.2.2.4. Các từ điển dữ liệu.

Từ điển dữ liệu vừa có ích cho việc bảo trì hệ thống vừa có ích trong quá trình thiết kế. Với một đầu vào đã được xác định rõ ràng trong biểu đồ phải có một đường nối vào từ điển dữ liệu cung cấp thông tin về kiểu, chức năng của dữ liệu và một giải thích cơ bản cho việc dữ liệu đi vào. Đôi khi người ta còn gọi đây là mô tả ngắn gọn của chức năng thành phần.

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

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



#### Thiết kế cơ sở dữ liệu

Thiết kế cơ sở dữ liệu được dùng để định nghĩa và rồi xác định cấu trúc của các sự vật nghiệp vụ được dùng trong hệ thống khách/nguồn phục vụ (client/ server). Việc phân tích đòi hỏi định danh các sự vật nghiệp vụ được thực hiện bằng việc dùng các phương pháp kỹ nghệ tiến trình nghiệp vụ đã thảo luận trước. Kí pháp mô hình hoá phân tích qui ước có thể được dùng để định nghĩa các sự vật nghiệp vụ, nhưng một kho cơ sở dữ liệu nên được thiết lập để thu tóm thông tin phụ vốn không thể được làm từ liệu đầy đủ bằng việc dùng kí pháp đồ hoạ như ERD.

##### Tạo ra biểu đồ thực thể/quan hệ (Element Relationship Diagram - ERD)

Biểu đồ thực thể/quan hệ cho phép người kỹ sư phần mềm đặc tả hoàn toàn các sự vật dữ liệu là cái vào và cái ra của hệ thống, các thuộc tính xác định nên các tính chất của những sự vật này, và mối quan hệ của chúng. Giống như hầu hết các phần tử của mô hình phân tích, ERD được xây dựng theo cách lặp. Cách tiếp cận sau đây được chọn:

1. Trong việc kêu gọi yêu cầu, khách hàng được yêu cầu liệt kê ra những "vật" mà ứng dụng hay tiến trình nghiệp vụ đề cập tới. Những "vật" này tiến hoá thành danh sách các sự vật dữ liệu vào và ra cũng như các thực thể ngoài tạo ra hay tiêu thụ thông tin.
2. Mỗi lúc lấy ra các sự vật, người phân tích và khách hàng xác định liệu có tồn tại ghép nối (chưa có tên tại giai đoạn này) hay không giữa sự vật dữ liệu này và sự vật dữ liệu khác.

3. Bất kì khi nào có ghép nối, thì người phân tích và khách hàng tạo ra một hay nhiều cặp sự vật / quan hệ.
4. Với mỗi cặp sự vật / quan hệ, tìm hiểu về bản số và thể thức.
5. Các bước từ 2 tới 4 được lặp lại liên tục cho tới khi tất cả các sự vật/quan hệ đã được xác định. Thường thì sẽ phát hiện ra những bỏ sót khi tiến trình này tiếp diễn. Các sự vật và quan hệ mới lúc nào cũng có thể được bổ sung vào khi số lần lặp tăng lên.
6. Các thuộc tính của từng thực thể được xác định.
7. Biểu đồ thực thể quan hệ được hình thức hoá và xét duyệt
8. Các bước từ 1 tới 7 được lặp lại cho tới khi việc mô hình hoá dữ liệu hoàn tất.

Trong kho này, một sự vật nghiệp vụ được định nghĩa như thông tin vốn thấy được cho người mua và người dùng hệ thống, không phải là người cài đặt của chúng. Thông tin này, được cài đặt bằng việc dùng cơ sở dữ liệu quan hệ, có thể được duy trì trong kho thiết kế. Thông tin thiết kế sau đây được thu thập cho cơ sở dữ liệu client/server:

- *Các thực thể* được định danh bên trong ERD cho hệ thống mới.
- *Các tệp* cài đặt cho các thực thể được định danh bên trong ERD.
- *Mối quan hệ tệp với trường* thiết lập nên cách bố trí cho các tệp bằng việc định danh trường nào được đưa vào tệp nào.
- *Các trường* xác định ra các trường trong thiết kế (từ điển dữ liệu).
- *Mối quan hệ tệp với tệp* định danh các tệp có quan hệ vốn có thể được nối lại để tạo ra cái nhìn hay câu hỏi logic.
- *Làm hợp lệ quan hệ* định danh kiểu mối quan hệ tệp với tệp hay trường với trường được dùng cho kiểm chứng.
- *Kiểu trường* được dùng để cho phép kế thừa các đặc trưng trường từ trường của siêu lớp (như ngày tháng, văn bản, số, giá trị, giá cả).
- *Kiểu dữ liệu* xác định các đặc trưng của dữ liệu được chứa trong một trường.
- *Kiểu tệp* được dùng để định danh vị trí của tệp.
- *Chức năng trường* bao gồm khoá, khoá ngoại, thuộc tính, trường ảo, trường suy dẫn, và những thứ giống thế.
- *Giá trị được phép* định danh các giá trị được phép cho trường kiểu trạng thái.
- *Qui tắc nghiệp vụ* là những qui tắc cho việc soạn thảo, tính toán các trường suy dẫn, ..v..v...

Khuynh hướng nghiêng về việc quản trị dữ liệu phân bố đã tăng tốc khi kiến trúc client/ server đã trở nên phổ cập. Trong các hệ thống client/ server có cài đặt cách tiếp cận này, cấu phần quản trị dữ liệu nằm ở cả phía client và server. Bên trong hoàn cảnh

của thiết kế cơ sở dữ liệu, vấn đề mấu chốt là phân bố dữ liệu. Tức là, làm sao dữ liệu được phân bố giữa client và server và được rải rác qua các nút của mạng?

Hệ cơ sở dữ liệu quan hệ tạo khả năng dễ dàng truy nhập vào dữ liệu phân bố qua việc dùng ngôn ngữ vấn đáp có cấu trúc (SQL). Ưu điểm của SQL trong kiến trúc client/ server là ở chỗ nó là "không dẫn lái". Trong một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS), kiểu của dữ liệu được xác định bằng việc dùng SQL, nhưng không cần tới thông tin dẫn lái. Tất nhiên hậu quả của điều này là ở chỗ RDBMS phải đủ phức tạp để duy trì vị trí của tất cả các dữ liệu và có khả năng xác định con đường tốt nhất cho nó. Trong các hệ thống cơ sở dữ liệu kém phức tạp hơn, một yêu cầu về dữ liệu phải chỉ ra cái gì được truy nhập và nơi có nó. Nếu phần mềm ứng dụng phải duy trì thông tin dẫn lái thì việc quản lý dữ liệu trở thành phức tạp hơn nhiều đối với hệ thống client/ server.

Cũng cần chú ý rằng các kỹ thuật phân bố và quản trị dữ liệu khác cũng có sẵn cho người thiết kế:

**Trích thủ công:** Người dùng được phép sao chép thủ công dữ liệu thích hợp từ nguồn phục vụ về máy khách. Cách tiếp cận này là có ích khi dữ liệu tĩnh được người dùng yêu cầu và điều khiển việc trích này có thể được để lại trong tay người dùng.

**Ảnh chụp nhanh:** Kỹ thuật này tự động việc trích thủ công bằng việc xác định một "ảnh chụp nhanh" về dữ liệu vốn phải được truyền từ server sang client trong những khoảng xác định trước. Cách tiếp cận này có ích cho việc phân bố dữ liệu tương đối tĩnh hơn là yêu cầu cập nhật không thường xuyên.

**Tái tạo:** Kỹ thuật này có thể được dùng khi nhiều bản sao của dữ liệu phải được duy trì tại những vị trí khác nhau (như các server hay client khác nhau). Tại đây, mức độ phức tạp leo thang bởi vì tính nhất quán dữ liệu, việc cập nhật, giữ an toàn, và xử lý tất cả đều phải được điều phối tại nhiều vị trí.

**Phân mảnh:** Trong cách tiếp cận này, cơ sở dữ liệu hệ thống được phân mảnh qua nhiều máy. Mặc dầu hấp dẫn về lý thuyết, việc phân mảnh là khó cài đặt một cách ngoại lệ và thường không hay gặp.

Thiết kế cơ sở dữ liệu và/ hoặc đặc biệt hơn, thiết kế cơ sở dữ liệu cho hệ thống client/ server là chủ đề bên ngoài phạm vi của môn học.

### 4.3. Giao diện người sử dụng.

Trong lời tựa cho cuốn sách về thiết kế giao diện người dùng, Ben Shneiderman có viết:

Chán nản và lo âu là một phần của cuộc sống thường ngày đối với những người dùng các hệ thống tin học hoá. Họ vật lộn để học ngôn ngữ chỉ lệnh (syntax - cú pháp câu lệnh) hay các hệ thống lựa chọn các mục (menu) vốn được xây dựng để giúp cho họ làm việc. Một số người gặp phải các tình huống nghiêm trọng như choáng máy tính, khiếp hãi thiết bị cuối (terminal) - ví



dụ như máy in, máy foto, máy fax, ... - hay cấu bản với mạng đến mức họ tránh né việc dùng các hệ thống tin học hóa.

Vấn đề mà Shneiderman ngụ ý tới là có thật. Tất cả chúng ta đều gặp các “giao diện” khó học, khó dùng, lẫn lộn, không khoan dung và trong nhiều trường hợp, hoàn toàn làm cho người dùng chán nản. Quả vậy, một số người dành thời gian và công sức để xây dựng ra các giao diện này và các thể người xây dựng ra chúng không chủ ý tạo ra các vấn đề như vậy.

Trong phần này ta sẽ xem xét thiết kế giao diện người dùng - một chủ đề đã trở nên ngày càng quan trọng khi việc dùng máy tính càng phát triển. Chúng ta gặp các giao diện "thông minh" khi dùng máy photocopy, lò vi sóng, bộ xử lý văn bản, hay hệ thống thiết kế có máy tính trợ giúp (CASE). Theo quan điểm người dùng, chính giao diện là cho phi công bay được trên con tàu hiện đại, bác sĩ X-quang diễn giải được các kết quả của máy quét CAT, ngân hàng chuyển hàng triệu đô-la qua các lục địa ... Giao diện theo nhiều cách là việc “đóng gói” cho phần mềm máy tính. Nếu nó dễ học, sử dụng đơn giản, trực tiếp và dung thứ thì người dùng sẽ thiên hướng dùng hiệu quả nhưng cái gì có ở bên trong. Nếu nó không có các đặc trưng này thì vấn đề sẽ thường xuyên nảy sinh.

Trong các phần trước, chúng ta đã thảo luận các thiết kế có liên quan đến những cái bên trong của phần mềm. Mặc dầu có tầm quan trọng chủ chốt về chất lượng phần mềm toàn bộ, "thiết kế" vẫn được coi như là bị che dấu với người dùng cuối cùng theo nhiều cách. Thiết kế giao diện lại khác. Nếu nó rất tốt thì người dùng sẽ rơi vào nhịp độ tự nhiên của tương tác. Người đẩy thậm chí có thể quên mất rằng việc trao đổi được tiến hành với máy. Nhưng nếu tồi thì người sử dụng sẽ lập tức biết đến điều đó và sẽ không hài lòng với cách thức "không thân thiện" của tương tác.

Thiết kế giao diện người dùng phải xử lý nhiều nghiên cứu về con người cũng như đã làm với các vấn đề công nghệ. Ai là người dùng? Người dùng học tương tác với hệ thống dựa trên máy tính mới như thế nào? Người dùng diễn giải thông tin do hệ thống tạo ra như thế nào? Người dùng trong đợi gì ở hệ thống? Đó chỉ là một số trong nhiều câu hỏi và phải trả lời như một phần của thiết kế giao diện người dùng.

#### **4.3.1. Nhân tố con người và tương tác người – máy.**

Khi chúng ta xem xét hệ thống tương tác dựa trên phần mềm thì cụm từ “nhân tố con người” mang một số ý nghĩa khác nhau. Tại mức nền tảng, ta nên hiểu là cảm nhận trực quan, tâm lí nhận thức của việc đọc, trí nhớ của con người, lập luận diễn dịch và quy nạp. Tại mức độ khác, chúng ta nên hiểu người dùng và hành vi của người đó. Cuối cùng chúng ta cần phải tìm hiểu các nhiệm vụ mà hệ thống dựa trên phần mềm thực hiện cho người dùng và những nhiệm vụ mà người dùng yêu cầu xem như một phần của tương tác người - máy.

Thiết kế hệ thống máy tính bao gồm một loạt hoạt động từ thiết kế phần cứng tới thiết kế giao diện người dùng. Các kỹ sư điện tử luôn chịu trách nhiệm về thiết kế phần cứng còn các kỹ sư phần mềm chịu trách nhiệm về thiết kế giao diện người sử dụng cũng như thiết kế hệ thống. 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 sử dụng là cơ chế qua đó thiết lập đối thoại giữa chương trình và người dùng. 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 của người dùng.

- Một giao diện khó sử dụng thì sẽ gây ra nhiều sai lầm của người dùng. Trong trường hợp xấu nhất, nó có thể làm cho hệ thống bị huỷ hoại bất chấp chức năng của nó. Còn bình thường thì có thể làm cho người dùng cảm thấy khó chịu, nhầm lẫn với phần mềm trong hệ thống. Hệ thống như vậy là không đáp ứng được nhu cầu của người dùng, họ sẽ nhanh chóng lãng quên và rời xa nó.
- Một giao diện thiết kế kém có thể làm cho người sử dụng gây ra các sai lầm nghiêm trọng. Nếu các thông tin được biểu diễn một cách lẫn lộn và dễ hiểu nhầm thì người dùng có thể hiểu sai lệch ý nghĩa của các khoản mục thông tin và gây ra một dãy các hành vi nguy hiểm. Trong giáo trình này, chỉ nhấn mạnh về giao diện đồ họa.

#### 4.3.1.1. Nền tảng về cảm nhận của con người.

Con người cảm nhận thế giới qua các hệ thống giác quan đã được hiểu tương đối rõ. Khi giao diện người-máy tính (HCI) được xem xét, thì các giác quan nhìn, sờ mó và nghe đóng vai trò hơn cả các giác quan khác. Các giác quan này làm cho người dùng hệ thống dựa trên máy tính cảm nhận được thông tin, ghi nhớ nó trong ký ức (con người) và xử lý nó bằng cách dùng lập luận suy diễn và quy nạp.

Bộ môn vật lý thần kinh về cảm nhận giác quan nằm ngoài phạm vi của quyển sách này. Một thảo luận tuyệt vời về các chủ đề có liên quan, được phát triển tham khảo riêng cho HCI, đã được Monk nêu ra. Để cung cấp một hiểu biết bước đầu cơ sở về các nhân tố con người và mối quan hệ của chúng với thiết kế giao diện người dùng, mục này sẽ trình bày một cách tổng quan ngắn gọn.

Phần lớn HCI đều được thực hiện thông qua trung gian trực quan (như báo cáo in hay đồ họa, màn hình). Mắt và óc làm việc với nhau để nhận và diễn giải thông tin trực quan dựa trên kích cỡ, hình dáng, màu sắc, hướng, chuyển động và các đặc trưng khác. Trao đổi trực quan có tính chất "song song". Nhiều khoản mục thông tin cụ thể được trình bày cụ thể để con người hấp thu. Đặc tả đúng đắn về trao đổi trực quan là phần tử mấu chốt của giao diện "thân thiện người dùng".

Mặc dầu có một khuynh hướng xác định hướng tới trao đổi hình ảnh (đồ hoạ) trong thiết kế HCI, nhiều thông tin trực quan vẫn còn được trình bày dưới dạng văn bản. Đọc - tiến trình trích thông tin từ văn bản - là hoạt động chủ chốt trong phần lớn các giao diện. Con người phải giải mã mẫu hình trực quan và tìm kiếm ý nghĩa của các từ hay cụm từ. Tốc độ của tiến trình này được kiểm soát bởi mẫu hình chuyển động trong mắt quét qua văn bản thông qua việc mắt di động dật qua, tốc độ cao, được gọi là *đảo mắt*. Kích cỡ văn bản, kiểu phong chữ, chiều dài dòng, viết hoa, vị trí, và màu sắc tất cả đều ảnh hưởng tới sự thoải mái để từ đó xuất hiện việc trích lọc thông tin.

Khi thông tin được trích ra từ giao diện, nó phải được ghi nhớ lại để sau lấy ra dùng. Bên cạnh đó người dùng có thể phải nhớ tới các chỉ lệnh, các dãy thao tác các phương án, tình huống sai lỗi và những dữ liệu bí mật khác. Tất cả những thông tin này đều được ghi nhớ trong kí ức con người - một hệ thống cực kì phức tạp mà hiện nay người ta coi nó bao gồm bộ nhớ ngắn hạn (*short term memory* - STM) và bộ nhớ dài hạn (*long term memory* - LTM). Cái vào cảm giác (nhìn, nghe, sờ) được đặt vào trong "bộ đệm" rồi được ghi nhớ vào trong STM nơi nó có thể được dùng lại ngay. Kích cỡ bộ đệm và chiều dài thời gian mà việc dùng lại xuất hiện là có giới hạn. Tri thức được duy trì trong LTM và tạo nên cơ sở cho đáp ứng có học hỏi của chúng ta khi dùng HCI. Cả thông tin cú pháp và ngữ nghĩa (tri thức) đều được ghi nhớ trong LTM. Nếu kỹ sư hệ thống xác định một giao diện người-máy mà giao diện này lại tạo ra những đòi hỏi vô lý về STM hay/ và LTM thì hiệu năng của phần tử con người của hệ thống sẽ bị suy giảm.

Phần lớn mọi người đều không áp dụng cách lập luận diễn dịch hay quy nạp khi phải đương đầu với một vấn đề. Thay vì thế, chúng ta áp dụng một cách trực cảm (theo hướng dẫn, quy tắc và chiến lược) dựa trên cách hiểu của chúng ta về vấn đề tương tự. Tức là một vấn đề giống thế, được gặp phải trong một ngữ cảnh hoàn toàn khác nhau, có thể được áp dụng bằng cách trực cảm khác nhau. Một HCI nên được xác định theo cách thức là cho con người phát triển được trực cảm cho tương tác. Nói chung, những trực cảm này nên giữ nhất quán qua những lần giao diện khác nhau.

#### 4.3.1.2. Mức độ kỹ năng con người và hành vi.

Bên cạnh những yếu tố cơ bản của cảm nhận con người, điều quan trọng cần chú ý tới là sự khác biệt về mức độ kỹ năng cá nhân, sự biến thiên nhân cách và cái khác biệt trong những người dùng hệ thống dựa trên máy tính. Một giao diện hoàn toàn chấp nhận được với kỹ sư có bằng cấp có thể hoàn toàn không thích hợp đối với công nhân không chuyên môn. Một giao diện được hai người dùng cùng mức giáo dục và nền tảng, nhưng với cá tính khác nhau thì dường như nó "thân thiện" với người này mà "không thân thiện" với người kia. Do vậy, giao diện người sử dụng phải tính đến *nhu cầu, cá tính, kinh nghiệm và khả năng* của người sử dụng.

Mức độ mức độ kỹ năng của người dùng cuối cùng (end-user) sẽ có một tác động đáng kể nên khả năng trích lọc thông tin có nghĩa từ HCI, đáp ứng một cách hiệu quả

với các nhiệm vụ tương tác có yêu cầu và ứng dụng một cách hiệu quả các trực cảm sẽ tạo ra nhịp điệu tương tác. Trong nhiều trường hợp, kỹ năng của con người còn hơn cả hiểu biết mang tính sách vở hay trí thông minh (tục ngữ Việt Nam có câu: *"trăm hay không bằng tay quen"*). Chẳng hạn, một thợ cơ khí dùng hệ thống chẩn đoán ô-tô dựa trên máy tính sẽ hiểu biết về lĩnh vực vấn đề và có thể tương tác một cách có hiệu quả thông qua một giao diện được thiết kế chuyên dụng để thích hợp cho người dùng có nền tảng cơ khí. Cùng giao diện này có thể làm cho một bác sĩ bị lú lẫn - cho dù vị bác sĩ này được giáo dục đáng kể và trong thực tế còn thành thạo về máy tính hơn anh thợ cơ khí kể trên.

Mọi người sử dụng máy tính đều có một tính cách duy nhất. Trong phần lớn các trường hợp, một nhân cách cá nhân sẽ ảnh hưởng chặt chẽ đến phong cách nhận biết của mình. Do đó, HCI lý tưởng sẽ được thiết kế để thích hợp cho những khác biệt trong từng nhân cách, hay một cách khác, sẽ được thiết kế để thích hợp với nhân cách "điển hình" trong một lớp người dùng cuối cùng. Shneiderman phác hoạ các lớp tâm lý sau đây: dám nhận mạo hiểm/ tránh mạo hiểm; hướng nội/ hướng ngoại; suy tư/ hấp tấp; hội tụ/ phân tán; lo âu nhiều/ ít; chấp nhận căng thẳng nhiều/ ít; chấp nhận mơ hồ nhiều/ ít; phụ thuộc/ độc lập lĩnh vực; quả quyết/ thụ động; động cơ cao/ thấp; ép buộc cao/ thấp; hướng trí tuệ trái/ phải.

Vẫn còn tương đối ít các kinh nghiệm thực tế để giúp cho người thiết kế HCI tạo ra một giao diện thích hợp cho nhiều người dùng có cá tính riêng. Tuy nhiên, gần như không có vấn đề là các giao diện nào đó sẽ dễ dàng được người dùng với kiểu tính cách cá nhân này chấp nhận hơn là với kiểu người dùng có tính cách cá nhân khác.

Sự phát triển nhanh chóng của các hệ thống dựa trên máy tính đã làm nảy nở một hiện tượng được gọi là "sợ kỹ thuật" - một nỗi sợ vô lý về các sản phẩm và hệ thống công nghệ cao. Khi chúng ta mở rộng hiểu biết của mình về nhân tố con người trong các hệ thống này và xác định HCI theo cách thích hợp cho nhu cầu con người thì rất có thể là mức độ sợ kỹ thuật sẽ dần giảm đi.

#### 4.3.1.3. Nhiệm vụ và nhân tố con người.

Một hệ thống dựa trên máy tính tương tác hiếm khi có thể để người dùng làm được một cái gì đó hoàn toàn mới. Trong phần lớn các hệ thống sẽ được xây dựng để tự động hoá (và do đó nâng cao) một số nhiệm vụ nào đó mà trước đây vẫn làm thủ công hay một cách tiếp cận khác. Một cách lý tưởng, công nghệ mới làm cho người dùng thực hiện nhiệm vụ một tốt hơn, nhanh hơn, hiệu quả hơn, chính xác hơn hay ít tốn kém hơn. Nhưng những nhiệm vụ nền tảng vẫn thế, và một HCI phải cung cấp cho người dùng cuối cùng một môi trường tự nhiên, dễ dàng để tiến hành nhiệm vụ đó.

Mặc dầu các nhiệm vụ cho từng ứng dụng là khác nhau, vẫn có thể có một phân loại tổng thể. Dù ta có xem xét cách làm việc "cũ" hay cách tiếp cận "mới" dùng hệ thống dựa trên máy tính tương tác thì các nhiệm vụ chung sau đây bao giờ cũng phải được thực hiện:

*Nhiệm vụ trao đổi:* Các hoạt động làm cho thông tin được chuyển từ nơi sản xuất đến nơi tiêu thụ.

*Nhiệm vụ đối thoại:* Các hoạt động là cho người dùng định hướng và điều khiển tương tác với hệ thống dựa trên máy tính.

*Nhiệm vụ nhận biết:* Các hoạt động thực hiện một khi đã thu được thông tin; các hoạt động liên kết với hệ thống.

*Nhiệm vụ điều khiển:* Các hoạt động cho phép người dùng kiểm soát thông tin và nhận biết và ra lệnh cho tiến trình, thông qua đó các nhiệm vụ tổng quát khác xuất hiện.

Để phát triển các thể nghiệm đặc biệt của những nhiệm vụ tổng quát này, ta sẽ dùng một kỹ thuật thiết kế giao diện người dùng, gọi là phân tích và mô hình hoá nhiệm vụ. Ta sẽ thảo luận mục này trong phần sau.

#### **4.3.2. Phong cách tương tác người-máy.**

Phong cách tương tác người-máy bao quát cả loạt các tùy chọn có quan hệ chặt chẽ với tiến hoá lịch sử của máy tính (và các thiết bị tương tác có liên qua khác) và tới khuynh hướng HCI. Khi phần cứng trở nên phức tạp hơn thì các tùy chọn cho phong cách tương tác cũng phát triển. Trong nhiều thể nghiệm, các hệ thống dựa trên máy tính hiện đại vẫn dùng một phong cách tương tác đã được thiết kế nguyên thủy cho môi trường phần cứng hoàn toàn lạc hậu từ 25 năm nay.

Trong những ngày đầu của tin học (trước khi có những thiết bị hiển thị đồ hoạ, chuột, trạm làm việc tốc độ cao hay các thiết bị tương tự) thì chỉ có một chế độ tương tác người máy duy nhất đó là giao diện chỉ thị lệnh và các câu hỏi (thế hệ I). Trao đổi thuần tuý văn bản thông qua các chỉ thị lệnh và các đáp ứng câu hỏi do hệ thống sinh ra. Người dùng có thể trao đổi với hệ thống bằng cách xác định một chỉ lệnh như:

```
>run progr1.exe/debug='on'/out=p1/in=t1 alloc=1000K
```

```
* RUN ALLOCATION TO BE QUEUED?>>yes
```

```
* AUTOMATIC CHECKPOINTING INTERVAL?>>5
```

Mặc dù chỉ lệnh và câu hỏi bí mật như trên rất chính xác, chúng vẫn có khả năng sinh lỗi, không cho người ta có khả năng khôi phục lại (nếu có phạm lỗi), và hơn thế nữa là rất khó học. Nói chung với phương cách giao tiếp như vậy với người dùng là không thể chấp nhận được và không hiệu quả.

Một cải tiến thêm và giao diện chỉ lệnh là đưa ra các mục để ta lựa chọn, ví dụ:

**Hãy tùy chọn chương trình mong muốn:**

**1 = Nhập dữ liệu từ bàn phím.**

**2 = Vào dữ liệu từ tệp tin đã có.**

3 = Thực hiện phân tích đơn giản.

4 = Thực hiện phân tích chi tiết.

5 = Tạo đầu ra dưới dạng bảng biểu.

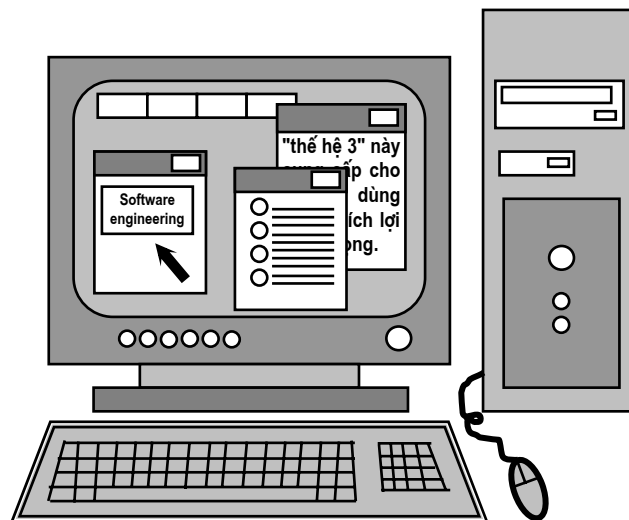
6 = Tạo đầu ra dạng đồ hoạ

7 = Các tùy chọn khác

**Bạn chọn: -**

Menu đơn giản cung cấp cho người dùng một ngữ cảnh tổng thể và ít sinh lỗi hơn trong chỉ thị dòng lệnh, nhưng cũng một khi dùng. Chẳng hạn, tùy chọn 7 trong thí dụ trên kéo theo rằng có thể cần đến các tùy chọn phụ (đơn). Người dùng không thể đi trực tiếp tới tùy chọn khác mà phải làm việc theo từng mức menu cho tới khi đạt tới tùy chọn mong muốn. Điều này rất chán và sẽ không hiệu quả.

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 của chúng tới thiết kế giao diện thì giao diện *trở* và *nhặt* (drop and drag), *hướng của sổ* bắt đầu tiến hoá. Khái niệm về *màn hình nền* (desktop) xuất hiện (được minh hoạ trong hình sau). Giao diện "thế hệ 3" này cung cấp cho người dùng một số ích lợi quan trọng.



1. Có thể hiển thị nhiều kiểu thông tin khác nhau, cho phép người dùng chuyển hoàn cảnh (như cho phép viết mã lệnh ở cửa sổ này; xem kết quả ở cửa sổ khác; viết cập nhật với lời thuật xử lý ở cửa sổ thứ ba) mà không mất một 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 nhầm lẫn.
2. Nhiều nhiệm vụ tương tác khác nhau có sẵn sơ đồ đơn kéo xuống, do vậy nó cho phép người dùng thực hiện các nhiệm vụ kiểm soát và đối thoại một cách dễ dàng.
3. Việc dùng nhiều biểu tượng đồ hoạ, mục lựa chọn (menu) kéo xuống, nút ấn và lý thuật cuộn màn hình làm giảm khối lượng gõ. Điều này có thể tăng tính hiệu

quả đối với những người không phải là chuyên viên gõ máy tính và có thể làm cho máy tính trở lên dễ tiếp cận đối với những người sợ bàn phím.

Thế hệ HCI hiện tại nổi bật cả các thuộc tính của giao diện thế hệ 3 với siêu văn bản và đa nhiệm - khả năng thực hiện một số nhiệm vụ khác nhau đồng thời (tất nhiên là theo quan điểm của người dùng). Do đó, một tác giả dùng hệ thống xuất bản trên màn hình nền (desktop) có thể gọi chương trình kiểm tra chính tả cho một chương dài, trong khi vẫn thực hiện việc tìm kiếm dữ liệu tham khảo về một chủ đề mới và vẫn có thể tiếp tục gõ văn bản cho chương mới. Trong khi đó máy tính (workstation) của tác giả này vẫn là một nhiệm vụ có tính "hậu trường" như theo dõi thư điện tử và/ hoặc duy trì lịch hẹn gặp hàng ngày với báo thức để chỉ ra một cuộc họp sắp tới. Các giao diện thế hệ 4 này hiện đã có ở trên nhiều trạm làm việc và cả PC.

Hiện nay, người ta đang nghiên cứu và phát triển giao diện theo âm thanh, ví dụ như: đã thực hiện được việc ra một số lệnh đơn giản cho máy tính thực hiện bằng giọng nói (có thể thực hiện việc ghi âm trước và lưu vào trong cơ sở dữ liệu, sau đó thực hiện phép so sánh với âm thanh vừa thu nhận được) hoặc kết hợp điều khiển từ xa...

Mỗi một trong những giao diện vừa được mô tả trên, chúng ta đều vẫn gặp trong mọi lĩnh vực ứng dụng. Không hoài nghi gì rằng khuynh hướng đang đi về các giao diện nhỏ nhất, hướng cửa sổ, đa nhiệm. Nhưng lời chú thích thì vẫn cần. Các giao diện thế hệ thứ 3 và thứ 4 thật sự đã làm cho HCI dễ dàng hơn và thân thiện hơn, nhưng chỉ khi ta tiến hành thiết kế cẩn thận cho giao diện.

#### **4.3.3. Mô hình thiết kế giao diện người-máy.**

Thiết kế giao diện người-máy làm một phần của một chủ đề lớn hơn mà chúng ta đang nói đến đó là *thiết kế phần mềm*. Các phương pháp thiết kế phần mềm mà chúng ta đã đề cập trong phần trước xử dụng luồng dữ liệu, các sự vật thể giới thực hay các cấu trúc dữ liệu và đưa vào ký pháp mô hình hoá để biểu diễn thiết kế. Các phương pháp thiết kế cho HCI còn chưa được dùng rộng rãi, mặc dù các đặc tả đồ hoạ và dựa trên ngôn ngữ cho giao diện người dùng quả đã có tồn tại.

Toàn bộ tiến trình thiết kế giao diện người dùng bắt đầu với việc tạo ra các mô hình khác nhau về chức năng của hệ thống (như được cảm nhận từ bên ngoài). Ta phải phân tích nhiệm vụ hường con người và máy tính cần để đạt tới chức năng hệ thống, xem xét các vấn đề thiết kế áp dụng cho mọi thiết kế giao diện, rồi sử dụng các công cụ làm bản mẫu và cuối cùng là cài đặt cho mô hình thiết kế và đánh giá kết quả về chất lượng.

Có bốn mô hình khác nhau khi thiết kế HCI. Người kỹ sư phần mềm tạo ra *mô hình thiết kế*; người kỹ sư con người (hay là kỹ sư phần mềm) lập ra *mô hình người dùng*, người dùng cuối cùng xây dựng một hình ảnh tinh thần, thường được gọi là *mô hình người dùng* hay *cảm nhận hệ thống*, và người cài đặt hệ thống tạo ra *hình ảnh của hệ thống*. Điều không may là các mô hình này có thể khác nhau đáng kể. Vai trò của

thiết kế giao diện là điều hoà những sự khác biệt này và đưa ra một cách biểu diễn nhất quán cho giao diện.

*Mô hình thiết kế* toàn bộ hệ thống tổ hợp cá biểu diễn dữ liệu, kiến trúc và thủ tục phần mềm. Đặc tả các yêu cầu có thể thiết lập nên một số ràng buộc giúp cho việc định nghĩa các hoạt động của người dùng trong hệ thống, nhưng thiết kế giao diện chỉ ngẫu nhiên là mô hình thiết kế.

*Mô hình người dùng* mô tả sơ lược cho người dùng cuối cùng của hệ thống. Để xây dựng một giao diện người dùng có hiệu quả, "mọi thiết kế nên bắt đầu với một sự hiểu biết về người dùng được dự định, kể cả các thông tin về tuổi tác, giới tính, khả năng thể chất, nền tảng giáo dục, văn hoá hay tôn giáo, động cơ, mục đích và cá nhân cách". Bên cạnh đó, người dùng có thể được phân loại thành:

- *Người mới học* - nói chung không hiểu biết về cú pháp của hệ thống và gần như không biết về ngữ nghĩa của ứng dụng hay việc dùng máy tính.
- *Người đôi khi dùng, có hiểu biết* - biết ngữ nghĩa của ứng dụng nhưng ít nhớ được thông tin cú pháp cần thiết để dùng giao diện.
- *Người dùng thường xuyên, có hiểu biết* - biết rõ về cú pháp và ngữ nghĩa, thường đi tới "triệu chứng người dùng siêu", tức là người dùng tìm những đường tắt và thích hợp với cách thức vận tác của giao diện.

*Cảm nhận hệ thống* (mô hình của người dùng) là hình ảnh của hệ thống mà người dùng mang trong đầu. Chẳng hạn, nếu người dùng sử dụng một chương trình xử lý văn bản đặc biệt được yêu cầu mô tả lại vận hành của chương trình thì cảm nhận về hệ thống sẽ giúp cho họ việc trả lời. Độ chính xác về mô tả sẽ tùy thuộc vào lược sử của người dùng (người mới học có đáp ứng ngần ngại) và sự quen thuộc nói chung với các phần mềm trong lĩnh vực ứng dụng. Một người dùng hiểu đầy đủ về bộ xử lý văn bản, nhưng chỉ làm việc một lần với bộ xử lý đó thì thực tế có thể đưa ra một mô tả đầy đủ về chức năng của nó hơn là người dùng mới học đã dành cả tuần để học hệ thống đó.

*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 hệ thống. Khi hình ảnh hệ thống trùng với cảm nhận hệ thống thì nhìn chung người dùng cảm thấy thoải mái với phần mềm và sử dụng nó một cách hiệu quả. Để thực hiện được tính sự tan chảy này của các mô hình thì mô hình thiết kế phải được xây dựng phù hợp với thông tin được chứa trong mô hình người dùng và hình ảnh hệ thống phải phản ánh được chính xác các thông tin cú pháp và ngữ nghĩa của giao diện.

#### **4.3.4. Hướng dẫn thiết kế giao diện người-máy.**

Thiết kế giao diện người - máy đòi hỏi nhiều kinh nghiệm của người thiết kế và kinh nghiệm giao thoại được trình bày trong nhiều bài báo kỹ thuật và rất nhiều cuốn



sách khác. nhiều nguồn tài liệu đã trình bày một tập các hướng dẫn thiết kế HCI để tạo ra giao diện “thân thiện” và hiệu quả. Trong mục này sẽ trình bày một số hướng dẫn thiết kế HCI quan trọng nhất.

Chúng ta gợi ý ba phạm trù của hướng dẫn HCI: tương tác chung, hiển thị dữ liệu và vào dữ liệu.

#### 4.3.4.1. Tương tác chung.

Hướng dẫn về tương tác chung thường xuyên qua biên giới của hiển thị thông tin, vào dữ liệu và điều khiển toàn bộ hệ thống. Do đó chúng là bao quát và rất hay bị bỏ qua. Những hướng dẫn sau đây tập trung vào tương tác chung:

*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 HCI.

*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 soát mọi hành động phá huỷ không tầm thường:* Nếu người dùng yêu cầu xoá một tệp tin, 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 nhiều hành động:* Các chức năng UNDO (hoàn tác) hay REVERSE (đảo ngược) đã giúp cho hàng nghìn người dùng khỏi mất đi hàng nghìn giờ làm việc. Khả năng lần ngược nên có trong mọi ứng dụng tương tác.

*Giảm thiểu khối lượng thông tin phải ghi 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 chức năng thiết kế 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 dùng các phím, cần xem xét chuột phải đi qua giữa các điểm thiết kế bố trí màn hình, đừng đầu người dùng vào tình huống phải tự hỏi, “Cái này là gì nhỉ?”.

*Dung thứ cho sai lầm:* Hệ thống nên tự bảo vệ khỏi lỗi của người dùng để khỏi bị chết, hỏng.

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

*Cung cấp tiện nghi cung trợ giúp cảm ngữ cảnh:* Trợ giúp mang tính trực tiếp, tức thời. Ví dụ khi người dùng di con trỏ chuột đến nơi có biểu tượng (icon) có dạng “X”

thì ngay lập tức, bên dưới xuất hiện dòng chú thích “xoá” (delete) hoặc “đóng” (close) (trong một số ngôn ngữ lập trình hiện nay thì đó là “tooltip”).

*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 ồng thì khó nhận dạng và nhớ. Nó cũng có thể chiếm không gian không cần thiết trong danh sách đơn.

*Dùng các động từ đơn giản hay cụm từ ngắn để đặt tên cho chỉ lệnh (command).* Tên của chỉ lệnh dài dòng thì khó nhận dạng và ghi nhớ. Nó cũng có thể chiếm nhiều không gian trong danh sách đơn.

#### 4.3.4.2. Hiện thị thông tin.

Nếu thông tin được HCI trình bày không đầy đủ, mơ hồ hay không dễ hiểu thì ứng dụng sẽ không thoả mãn nhu cầu người dùng. Thông tin được “hiển thị” theo nhiều cách khác nhau: với văn bản, hình ảnh và âm thanh; bằng cách sắp đặt, di chuyển và kích cỡ, dùng màu sắc, độ phân giải; và thậm chí cả việc bỏ lửng. Các hướng dẫn sau đây chỉ tập trung vào hiển thị thông tin:

*Chỉ hiển thị thông tin có liên đến ngữ cảnh hiện tại.* Người dùng không cần khó nhọc để lần qua dữ liệu, đơn giản và sử dụng đồ hoạ phụ để thu được chức năng hệ thống riêng.

*Đừng chôn vùi người dùng dưới dữ liệu, hãy dùng định dạng trình bày cho phép hấp thụ thông tin nhanh chóng.* Đồ hoạ hay sơ đồ có thể thay thế cho các bảng lớn.

*Dùng nhãn nhất quán, cách viết tắt chuẩn và màu sắc có thể dự đoán trước được thông tin trình bày.* Ý nghĩa của thông tin không cần phải tham khảo tới thông tin bên ngoài.

*Cho phép người dùng duy trì ngữ cảnh trước quan.* Nếu việc hiển thị đồ hoạ máy tính được thay đổi thì tỷ lệ hình ảnh gốc nên được thường xuyên (giao diện chính - main menu- được rút gọn ở góc màn hình) để cho người dùng hiểu được vị trí tương đối của phần hình ảnh hiện đang được xét.

*Đưa ra các thông báo lỗi có nghĩa.* Thông tin đưa ra ngắn gọn nhưng cũng cần phải chú ý đến ngữ nghĩa trọn vẹn của nó. Ví dụ, một thông báo sau là vô nghĩa “Có lỗi”. Vấn đề của ta cần biết là lỗi gì, sửa nó như thế nào.

*Dùng chữ hoa, chữ thường, thụt 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 hình dạng văn bản cũng có tác động đáng kể đến sự thoải mái để người dùng hấp thụ 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 đạt tới dễ dàng.

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

*Xem xét hiển thị thông tin trên màn hình và dùng nó một cách hiệu quả.* Khi dùng nhiều cửa sổ, ít nhất lên có những không gian để chỉ ra một phần của cửa sổ này. Bên cạnh đó, kích cỡ màn hình (vấn đề công nghệ hệ thống) nên được lựa chọn để hoà hợp với kiểu ứng dụng được cài đặt. Một kỹ năng mà chúng ta hay dùng, đó là cửa sổ xếp chồng nhiều, giả sử có 5 cửa sổ, nếu xác suất từ cửa sổ thứ 5 về cửa sổ thứ 1 nhiều hơn thì nên bố trí chúng ở cạnh nhau để đạt được hiệu quả trong thao tác.

#### **4.3.4.3. Vào dữ liệu.**

Phần lớn thời gian của người dùng được dành cho việc chọn lựa các chỉ lệnh, nhập vào dữ liệu và cung cấp cái vào cho hệ thống. Trong nhiều ứng dụng hiện nay, bàn phím vẫn còn là phương tiện nhập dữ liệu có chính, nhưng chuột, bộ số hoá và thậm chí cả hệ thống nhận dạng tiếng nói đang nhanh chóng trở thành các phương tiện có hiệu quả. Những hướng dẫn sau đây tập trung vào việc đưa vào dữ liệu:

*Tối thiểu hoá hành động cần đưa vào mà người dùng cần thực hiện.* Thông thường, không nên quá 15 thao tác trên một giao diện. Vì đó là ngưỡng thao tác chính xác của con người. Việc rút gọn khối lượng gõ vào là yêu cầu trước hết. Điều này có thể được thực hiện bằng cách dùng chuột để chọn từ một tập cái vào đã được xác định (có thể là hộp lựa chọn - selection box - ví dụ: Chọn tên quốc gia); dùng “thanh trượt” để xác định cái vào trong một miền giá trị; dùng “macro” làm cho chỉ một phím cũng chuyển thành một tập dữ liệu và phức tạp hơn (ví dụ: chức năng “tự động hoàn thiện” - Autocorrect - trong Microsoft Words).

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

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

*Tương tác nên mềm dẻo, nhưng cũng nên hoà hợp với một đưa vào đang được ưa thích.* Mô hình theo quan điểm xuất phát từ yêu cầu của người dùng sẽ trợ giúp cho việc xác định một đưa vào nào là ưa thích. Một thư ký có lẽ rất thích hợp với cách đưa vào từ bàn phím, trong khi người quản lý lại thoải mái khi dùng các thiết bị trở và nhảy như chuột.

*Khử kích hoạt các chỉ lệnh không thích hợp với hoàn cảnh hiện tại.* Điều này bảo vệ cho người dùng khỏi phải dùng thêm một hành động nào đó mà nó có thể 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 của hành động yêu cầu (khi có thể được trong hoàn cảnh ứng dụng), và khôi phục được từ các điều kiện lỗi mà không cần phải ra khỏi chương trình.

*Cung cấp trợ giúp cho mọi hành động đưa vào.* Thực tế cho thấy, chi phí để xây dựng cho phần trợ giúp (help) của hệ thống thường chiếm tới 1/3 trên tổng chi phí. Nhưng đa số người dùng hiện nay ít dùng nó. Không phải người dùng không biết là có trợ giúp, nhưng vì bản tính ngại khó, và cũng chưa biết cách tra cứu. Trợ giúp trực tiếp (tooltip) có vẻ như thuận lợi hơn cả. Tuy nhiên cũng cần phải cung cấp cho người dùng một chức năng bật/ tắt tooltip, vì đối với người dùng thành thạo thì tooltip đôi khi gây phiền toái và mất thời gian.

Tuy nhiên, hiện nay thiết kế giao diện người - máy vẫn chưa có một tiêu chuẩn nào, mà chủ yếu theo khả năng và còn mang tính nghệ thuật. Bởi giao diện của chương trình được cảm nhận qua chính người dùng, đây là đa số và phụ thuộc nhiều vào tâm lý, sở thích, trình độ của người dùng. Điều này đòi hỏi các nhà xây dựng phải tìm hiểu nhu cầu của người dùng một cách cụ thể, đầy đủ. Từ đó có thể đưa ra được một giao diện phù hợp với đa số người dùng.

# CHƯƠNG 5

## THU THẬP DỮ LIỆU VÀ QUẢN LÝ DỰ ÁN

### 5.1. Thu thập dữ liệu

#### 5.1.1. Thu thập dữ liệu từ người dùng

Mỗi giai đoạn phát triển hệ thống đòi hỏi sự trao đổi giữa nhà phát triển và người dùng để nhận được thông tin có ích. Mỗi giai đoạn cần tìm kiếm một dải rộng các câu hỏi về ứng dụng. Ví dụ: Khi phân tích tính khả thi, các câu hỏi tương đối rộng và tổng quát:

- Đây là phạm vi của vấn đề?
- Cách tốt nhất để tự động hoá là gì?
- Công ty có cố gắng để phát triển ứng dụng này hay không?
- Công ty có thể hỗ trợ việc phát triển ứng dụng không?

Khi phân tích yêu cầu chúng ta tìm hiểu các thông tin có liên quan đến ứng dụng là gì. Ví dụ:

- Các dữ liệu cần thiết là gì?
- Các xử lý nào được tiến hành và các thông tin chi tiết liên quan?

Khi thiết kế chúng ta phát triển thêm: **Làm thế nào** thông tin có liên quan tới ứng dụng:

- Làm thế nào chuyển ứng dụng vào môi trường đã chọn?
- Làm thế nào thiết kế dữ liệu logic được chuyển vào thiết kế dữ liệu vật lý?
- Các module chương trình được phối hợp với nhau như thế nào?

Các thông tin đó không xuất phát từ đâu khác ngoài chính từ yêu cầu của người dùng. Nhiệm vụ của nhà phát triển là phải nắm bắt được các thông tin trên. Có nhiều cách để thu thập dữ liệu: *Phỏng vấn - họp nhóm - quan sát - giới thiệu trước chương trình sau đó xin ý kiến - ấn định công việc tạm thời - làm việc chung - xem xét tài liệu nội bộ, tài liệu ngoài...* Mỗi phương pháp có ưu, nhược điểm riêng (chúng ta sẽ thảo luận sau). Nhà phát triển phần mềm phải biết vận dụng linh hoạt các phương pháp trên để thu được thông tin một cách hiệu quả nhất.

#### 5.1.2. Các tính chất của dữ liệu.

Các dữ liệu được phân biệt theo một vài khía cạnh:

- Định hướng thời gian.
- Cấu trúc.
- Nhập nhằng.
- Ngữ nghĩa.
- Độ lớn.

Mỗi yếu tố trên đều quan trọng trong việc xác định các đặc tả của ứng dụng bởi vì chúng hướng dẫn cho công nghệ phần mềm biết số lượng và kiểu thông tin nên được chọn. Cũng vậy, các kiểu dữ liệu khác nhau có liên quan tới các loại ứng dụng khác nhau và đòi hỏi các kỹ thuật khai thác thông tin khác nhau. Không chú ý tới các đặc tính của dữ liệu sẽ dẫn tới lỗi phân tích thiết kế.

Bên cạnh việc thu thập thông tin, chúng ta cũng cần sử dụng các kỹ thuật định lượng thông tin và biên dịch và ứng dụng đề ra.

**Tính chất 1:** *Hướng thời gian.*

Tính hướng thời gian của dữ liệu đề cập tới *quá khứ, hiện tại* hoặc các *đòi hỏi tương lai* của ứng dụng đề ra.

Các *dữ liệu quá khứ*, ví dụ, có thể mô tả công việc đã được biến đổi thế nào qua thời gian, các quy định ảnh hưởng thế nào tới nhiệm vụ, vị trí của nó trong tổ chức và nhiệm vụ. Các thông tin quá khứ là chính xác, đầy đủ và xác đáng.

Các *thông tin hiện tại* là các thông tin và cái gì đang xảy ra. Ví dụ thông tin ứng dụng hiện tại liên quan tới quá trình hoạt động của công ty, số lượng các lệnh được thực hiện trong ngày hoặc số lượng các hàng hoá được sản xuất, các chính sách, sản phẩm, đòi hỏi nghiệp vụ, yêu cầu pháp quy hiện tại hoặc các ràng buộc khác cũng rất cần thiết cho việc phát triển ứng dụng. Các thông tin hiện tại nên được chuyển thành các tư liệu cho phù hợp với đội ngũ phát triển để tăng sự hiểu biết của họ về ứng dụng và phạm vi của bài toán

Các *đòi hỏi trong tương lai* liên quan đến các sự thay đổi sẽ diễn ra, chúng không chính xác và rất khó kiểm tra. Các dự đoán kinh tế, khuynh hướng tiếp thị, kinh doanh là các ví dụ.

**Tính chất 2:** *Tính có cấu trúc.*

Thông tin chúng ta thu thập được là những thông tin được tổ chức theo một cấu trúc (khuôn mẫu) nhất định; có như vậy mới thể hiện một ý nghĩa phản ánh một đối tượng nào đó, điều này là hiển nhiên. Tuy nhiên, trong quá trình thu thập dữ liệu, chúng ta có khi không hiểu được cấu trúc của thông tin phản ánh, mà rất có thể hiểu

theo hướng khác (điều này đã được đề cập ở phần *các lỗi có thể mắc phải trong quá trình phát triển hệ thống* - Chương 2).

*Cấu trúc của thông tin* định hướng về phân mở rộng theo đó thông tin có thể được phân loại theo một cách nào đó. Cấu trúc có thể tham chiếu tới các hàm, môi trường hoặc dạng dữ liệu hay hình thức xử lý. Các thông tin thay đổi từ phi cấu trúc cho tới cấu trúc mà phần cấu trúc được xác định bởi công nghệ phần mềm (SE).

Một ví dụ thực tế khi phân tích chức năng của nghiệp vụ. Các chức năng của nghiệp vụ nếu theo người quản lý hệ thống thì không thể kể ra hết vì đó là các công việc của từng bộ phận, của từng nhân viên. Do vậy ta chỉ nắm được những cái tổng quan (có tính trừu tượng cao - không rõ ràng, cụ thể). Còn các chức năng nghiệp vụ của từng bộ phận, từng nhân viên thì rất nhỏ lẻ. Và đứng giữa một danh sách các chức năng như vậy thì khó có thể thấy được tính cấu trúc của nó. Các nhà phân tích lại phải "ngồi lại" với nhau và tổ chức lại các chức năng nghiệp vụ đó. Có như vậy thì khi xây dựng chương trình, ta tránh phải làm đi làm lại các chức năng giống nhau giữa các bộ phận trong thực tế. Mà ta chỉ cần nêu ra một liên kết (link) từ bộ phận (module) này đến bộ phận khác.

Tính "không chuẩn" của dữ liệu thể hiện rõ nhất ở thông tin trong một tờ "hoá đơn". Hoá đơn thanh toán thể hiện rất nhiều thông tin, như: *Số HD, Tên HD, Tên khách hàng, Địa chỉ khách hàng, ...* và sau đó là một bảng liệt kê chi tiết *tên các mặt hàng, đơn giá, số lượng, thành tiền ...* nhưng trong thực tế, không một bảng dữ liệu có khuôn dạng giống như một hoá đơn nào có mặt trong kho dữ liệu của hệ thống. Điều này là do liên kết dữ liệu từ các bảng khác mà thành, tránh lưu trữ trùng lặp quá nhiều thông tin. Do vậy, các nhà thiết kế dữ liệu đã tổ chức lại cấu trúc của dữ liệu cần lưu trữ.

### **Tính chất 3: Đầy đủ.**

Hơn lúc nào hết, khi tìm hiểu về một đối tượng hay lĩnh vực nào đó, ta luôn cần thông tin phản ánh về nó một cách đầy đủ và chính xác nhất có thể có. Về mặt lý thuyết thì không bao giờ ta có được toàn bộ thông tin về đối tượng hay lĩnh vực mà ta xử lý. Trong thực tế cũng như vậy, thông tin mà ta có chỉ là tạm đủ để ta có thể xử lý mà thôi.

Các thông tin có thể xếp theo cấp độ tính đầy đủ mà cao nhất là mọi thông tin cần thiết sẽ được biểu diễn. Mỗi kiểu ứng dụng đòi hỏi một mức độ đầy đủ khác nhau. Các hệ thống xử lý giao dịch luôn tiếp cận các thông tin đầy đủ và chính xác (ví dụ hệ thống bán vé máy bay). Tuy nhiên các hệ thống xây dựng theo kiến trúc hệ chuyên gia hay trí tuệ nhân tạo (AI) là minh hoạ tốt nhất việc xử lý thông tin không đầy đủ.

### **Tính chất 4: Nhập nhằng.**

Tính nhập nhằng là một thuộc tính của dữ liệu không trong sáng về nghĩa hoặc có nhiều nghĩa một cách hữu ý (có chủ định). Tính chất này liên quan đến mức độ ngữ

nghĩa. Ví dụ, nhìn thấy một cửa hiệu có thể đề biển “Giặt là hấp”, thì một cậu bé có thể hỏi bố một câu hỏi như sau: “Tại sao giặt lại là hấp?”, vào hoàn cảnh này, ông bố sẽ phải mất rất nhiều công sức để giải thích cho con hiểu. Như vậy có hiện tượng “ông nói gà, bà hoá cuốc”. Để giải quyết vấn đề này cần căn cứ vào ngữ cảnh.

**Tính chất 5: Ngữ nghĩa.**

Mọi người trong một tổ chức đều có một tập hợp các định nghĩa được chia sẻ cho biết các thuật ngữ, chính sách hoặc các hành động được biểu hiện như thế nào.

Ngữ nghĩa rất quan trọng với việc phát triển ứng dụng và với chính bản thân ứng dụng đó. Nếu mọi người dùng chung một thuật ngữ mà có cách hiểu khác nhau thì sẽ dẫn đến không thể trao đổi thông tin được. Đối với ứng dụng thì dữ liệu sẽ không bao giờ xử lý được cho đến khi người sử dụng hiểu được ngữ nghĩa của dữ liệu này. Các ứng dụng sẽ có ý nghĩa xác định với mục dữ liệu được định tính thông qua việc đào tạo và sử dụng lâu dài. Khi các cán bộ chủ chốt chuyển công tác, thì khả năng chuyển hoá ngữ nghĩa dễ mất. Việc đánh mất ngữ nghĩa của một công ty có thể gây tổn thất rất lớn cho công ty đó.

**Tính chất 6: Độ lớn (volume).**

*Volume* là số lượng các sự kiện nghiệp vụ hệ thống phải tiến hành trong một chu kỳ nào đó. Volume của tạo mới hay thay đổi khách hàng được tiến hành theo tháng hoặc năm, trong đó volume của giao dịch được tiến hành theo ngày giờ hoặc là theo peak volume (peak volume là số các giao dịch hoặc các sự kiện được thực hiện trong thời kỳ bận nhất). Thời kỳ cao điểm có thể là cuối năm hoặc cuối các quý, ví dụ chuẩn bị cho báo cáo nộp thuế. Volume của dữ liệu là một nguồn thông tin phức tạp bởi vì số lượng thời gian cần thiết với một giao dịch đơn lẻ có thể trở thành rất quan trọng đối với lượng lớn dữ liệu cần xử lý sau này.

**5.1.3. Các kỹ thuật thu thập dữ liệu.**

Các kỹ thuật thu thập dữ liệu có thể kể ra là: phỏng vấn, họp nhóm, quan sát ấn định công việc tạm thời, xem xét tài liệu, xem xét phần mềm. Mỗi kỹ thuật đều có điểm mạnh và hạn chế và số lượng và kiểu dữ liệu ta thu được khi sử dụng chúng. Chúng ta hãy bàn luận về các kỹ năng này.

**5.1.3.1. Phỏng vấn.**

Phỏng vấn là việc tập hợp một nhóm người số lượng ít trong một khoảng thời gian cố định với một mục đích cụ thể. Phỏng vấn thường được tiến hành với 1 hoặc 2 người hỏi đối với 1 người được phỏng vấn. Trong quá trình phỏng vấn, các câu hỏi có thể được thay đổi. Bạn có thể đánh giá được cảm nhận của họ, động cơ và thói quen với các bộ phận, quá trình quản lý hoặc các thông tin về thực thể khác đáng chú ý. Kiểu của phỏng vấn là kiểu của thông tin yêu cầu. Phỏng vấn được dẫn dắt sao cho cả 2 bên tham gia đều cảm thấy thoải mái với kết quả của nó. Cuộc phỏng vấn được chuẩn bị kỹ



đồng nghĩa với việc hiểu được về người đang được phỏng vấn. Do đó bạn không là cho họ bối rối và bạn có thể hỏi vài câu ban đầu được chuẩn bị cho dù không phải là tất cả.

Một cuộc phỏng vấn bao giờ cũng có bắt đầu, đoạn giữa và kết thúc.

- Lúc bắt đầu, bạn tự giới thiệu và đặt các câu hỏi đơn giản. Nên bắt đầu với các câu hỏi tổng quát vì không đòi hỏi các trả lời mang tính quan điểm cá nhân. Hãy chú ý đến kết quả trả lời để tìm ra mối các câu hỏi tiếp theo và tính trung thực, thái độ của người được phỏng vấn.
- Vào giữa buổi, nên tập trung vào chủ đề. Hãy lấy mọi thông tin bạn cần lưu ý, sử dụng các kỹ thuật mà bạn đã chọn ban đầu. Nếu thấy một và thông tin qua trọng, hãy hỏi xem bạn có thể được thảo luận sau này.
- Vào lúc kết thúc, hãy tóm tắt các thứ mà bạn đã nghe và nói những gì sẽ phỏng vấn tiếp. Bạn có thể ghi chép và đề nghị người được hỏi xem xét lại. Tốt nhất là trong thời gian 48 giờ và có sự chấp nhận của người dùng theo ngày xác định.

Phỏng vấn có thể sử dụng 2 loại câu hỏi:

- *Câu hỏi mở*: Là câu hỏi có nhiều cách trả lời khác nhau, câu hỏi mở thích hợp cho các chức năng ứng dụng hiện tại cũng như đang đề nghị và cho việc xác định cảm nhận ý kiến, và mong đợi về ứng dụng được đề ra. Một ví dụ là: “Ông có thể nói cho tôi về ...”, “Ông có thể mô tả làm thế nào ...”.
- *Câu hỏi đóng*: là câu hỏi mà chỉ trả lời “có” hoặc “không” hoặc một câu trả lời cụ thể. Các câu hỏi đóng tốt cho khai thác thông tin thực tế hoặc bắt người dùng tập trung vào phỏng vấn. Ví dụ, câu hỏi có thể là: “Bạn có dùng các báo cáo hàng tháng hay không?”. Với các câu trả lời “Có” thì có thể được tiếp nối bằng câu hỏi mở: “Ông có thể giải thích ...”

#### Các bước tiến hành phỏng vấn thành công

Tiến hành đặt cuộc hẹn phù hợp với thời gian của phỏng vấn.

Chuẩn bị tốt, tìm hiểu kỹ về người được phỏng vấn.

Đúng giờ.

Có kế hoạch mở đầu

- Giới thiệu bản thân, mục đích.
- Sử dụng câu hỏi mở để bắt đầu.
- Luôn lưu ý vào câu trả lời.
- Có kế hoạch cho nội dung chính.
- Kết hợp câu hỏi đóng và mở.

- Luôn bám sát các cách trình bày và phát triển chi tiết.
- Luôn cung cấp thông tin phản hồi, ví dụ: “Cho phép tôi trình lại điều ông vừa nói ...”.
- Hạn chế ghi chép nếu thấy không tiện.
- Có kế hoạch kết thúc.
- Tóm tắt nội dung, yêu cầu hiệu chỉnh.
- Yêu cầu xác thực lại nội dung, đánh giá lại ghi chép.
- Cho biết ngày tháng họ sẽ nhận được báo cáo.
- Thống nhất ngày tháng lấy bản hiệu chỉnh.
- Xác nhận lại lịch làm việc.

Các câu hỏi có thể đưa ra theo kiểu có cấu trúc hay phi cấu trúc.

- *Phỏng vấn có cấu trúc* là phỏng vấn trong đó người được phỏng vấn đã có danh sách các mục cần duyệt qua, các câu hỏi xác định và các thông tin cần tìm hiểu đã được xác định trước.
- *Phỏng vấn không cấu trúc* là phỏng vấn được định hướng bởi câu trả lời. Các câu hỏi phần lớn là câu hỏi mở, không có một kế hoạch ban đầu. Do vậy người đi phỏng vấn biết các thông tin cần thiết sẽ dùng từ các câu hỏi mở để phát triển chi tiết hơn về chủ đề.

Phỏng vấn có cấu trúc thích hợp khi bạn biết về các thông tin cần thiết trước khi phỏng vấn. Ngược lại, phỏng vấn phi cấu trúc thích hợp khi bạn không thể đoán trước được chủ đề, hay chưa có thông tin gì về người được phỏng vấn. Các trường hợp điển hình của phỏng vấn là người khách hàng bắt đầu với phỏng vấn phi cấu trúc để cho hai bên nhận thức được về miền của bài toán (hiểu sơ lược vấn đề). Sau đó, phỏng vấn dần dần trở thành có cấu trúc và tập trung vào các thông tin bạn cần để hoàn chỉnh phân tích.

Các kết quả phỏng vấn người sử dụng lên được trao đổi lại với người được phỏng vấn trong một thời gian ngắn. Người được phỏng vấn phải được báo trước về thời hạn đối với việc phỏng vấn. Tuy nhiên, có thể xin bố trí bổ sung phỏng vấn trong trường hợp còn nhiều điều cần hỏi hoặc nhiều người cần gặp.

Bảng sau so sánh phỏng vấn có cấu trúc và phỏng vấn phi cấu trúc.

	<b>Phỏng vấn có cấu trúc</b>	<b>Phỏng vấn phi cấu trúc</b>
<b>Ưu</b>	Dùng dạng chuẩn cho nhiều câu hỏi Dễ quản lý và đánh giá	Có khả năng mềm dẻo nhất Cần chăm chú nghe và có kỹ năng mở rộng câu hỏi.

<b>điểm</b>	Đánh giá được nhiều mục đích. Không cần đào tạo nhiều. Có kết quả trong các phỏng vấn.	Có thể bao được những thông tin chưa biết Đòi hỏi có thực hành.
<b>Nhược điểm</b>	Chi phí chuẩn bị lớn. Tính có cấu trúc có thể không thích hợp cho mọi tình huống. Giảm tính chủ động của người đi phỏng vấn.	Lãng phí thời gian phỏng vấn. Người được phỏng vấn có thể định kiến với các câu hỏi. Tốn thời gian lựa chọn và phân tích thông tin.

Một kỹ năng tốt là phát triển các sơ đồ như là một phần của tài liệu phỏng vấn. Khi bắt đầu một cuộc phỏng vấn mới, nên bàn bạc về các sơ đồ và đưa cho họ bản ghi chép để họ có thể kiểm tra sau này. Bạn sẽ nhận được ngay ý kiến phản hồi về tính chính xác của sơ đồ và hiểu biết của bạn về ứng dụng. Lợi ích của cách tiếp cận này thể hiện cả mặt kỹ năng và tâm lý. Từ khía cạnh kỹ thuật, bạn thường xuyên được kiểm tra lại các vấn đề mà bạn được nghe. Cho tới khi thời gian phân tích kết thúc, cả bạn và khách hàng đều tin chắc rằng quá trình xử lý ứng dụng là đầy đủ. Từ khía cạnh tâm lý, bạn làm tăng niềm tin của khách hàng vào khả năng phân tích bằng cách trình bày các hiểu biết của mình. Mỗi khi bạn cải thiện sơ đồ và đi vào phân tích, bạn cũng tăng được niềm tin của người sử dụng rằng bạn có thể xây dựng được ứng dụng đáp ứng được nhu cầu của họ.

Phỏng vấn thích hợp cho việc nhận thông tin đảm bảo cả số lượng lẫn chất lượng:

Các kiểu thông tin định tính là: các ý kiến, niềm tin, thói quen, chính sách và mô tả.

Các kiểu thông tin định lượng bao gồm: tần suất, số lượng, định lượng các mục được dùng trong ứng dụng.

Phỏng vấn là một dạng khác của thu thập dữ liệu có thể làm bạn lạc lối, thiếu chính xác hoặc thông tin không thích hợp. Bạn cần học cách đọc ngôn ngữ bằng cử chỉ, thói quen để quyết định được các điều kiện cần thiết cho cùng một thông tin.

Trong khi phỏng vấn, chúng ta cần chú ý đến hành động của người được phỏng vấn để có cách ứng xử thích hợp. Bảng sau liệt kê một vài tình huống và kinh nghiệm xử lý.

Hành vi của người được phỏng vấn.	Đáp ứng của người đi phỏng vấn.
Đoán các câu trả lời chứ không thừa nhận là không biết	Sau phỏng vấn, kiểm tra chéo các câu trả lời.
Cố nói những điều lọt tai người đi phỏng vấn, sai sự thật.	Tránh các câu hỏi dễ đoán được câu trả lời, kiểm tra chéo các câu hỏi
Cho thông tin không đầy đủ	Kiên trì hỏi để đạt mục đích.
Dừng trình bày khi người đi phỏng vấn ghi chép	Ghi nhanh nhất có thể, chỉ hỏi các câu quan trọng

Vội vã hay trả lời rời rạc, uể oải	Nhanh chóng kết thúc, đề nghị bố trí buổi khác
Thể hiện sự không quan tâm, trả lời đứt quãng	Nói chuyện vui sau đó chuyển đề tài khác
Không muốn thay đổi môi trường hiện tại	Động viên cải thiện môi trường hiện tại và so sánh 2 khuynh hướng.
Không hợp tác, từ chối trả lời	Lấy nguồn tin khác và hỏi: “Ông có quan tâm về những điều người khác nói về ông hay không?”. Nếu câu trả lời là “Không” thì thôi phỏng vấn.
Phàn nàn về vị trí công tác, lương, ...	Tìm ra mấu chốt vấn đề. Cố gắng dẫn dắt về chủ đề chính, ví dụ: “Dường như cơ quan ông có rất nhiều vấn đề, có thể ứng dụng mới mà chúng tôi đề xuất sẽ giải quyết được các vấn đề trên”.
Là người thích thú về công nghệ	Chọn lọc các thông tin cần thiết, không để bị lôi cuốn vào các vấn đề công nghệ.

Phỏng vấn và gặp gỡ phù hợp với mọi loại kiểu dữ liệu do đó chúng thường xuyên được sử dụng.

***Ưu điểm của phỏng vấn:***

- Nhận được cả thông tin chất lượng và số lượng.
- Nhận được cả thông tin đầy đủ và chi tiết.
- Là phương pháp tốt cho các yêu cầu bên ngoài.

***Nhược điểm của phỏng vấn:***

- Đòi hỏi có kỹ năng giao tiếp.
- Có thể có kết quả thiên vị vì mang tính chủ quan của người được phỏng vấn.
- Có thể dẫn đến các thông tin sai lạc, không liên quan, thiếu chính xác.
- Đòi hỏi phải có 3 người để kiểm tra kết quả.
- Không thích hợp với số lượng lớn người.

**5.1.3.2. Quan sát.**

Quan sát có thể tiến hành thủ công hoặc tự động.

- *Theo cách thủ công*, người quan sát ngồi tại chỗ và ghi chép lại các hoạt động, các bước xử lý công việc. Các băng video đôi khi có thể được dùng. Ghi chép hoặc băng ghi hình được phân tích cho các sự kiện, các mô tả động từ chính, hoặc các hoạt động chỉ rõ lý do, công việc, hoặc các thông tin về công việc.
- *Theo cách tự động*, máy tính sẽ lưu trữ chương trình thường trú, lưu lại vết của các chương trình được sử dụng, email và các hoạt động khác được xử lý bởi máy. Các file nhật ký của máy sẽ được phân tích để mô tả công việc.

### ***Ưu điểm của quan sát:***

- Bao trùm được các tiêu chuẩn quyết định, quy trình suy luận, các thủ tục khớp nối (mang tính thực hành).
- Kỹ sư phần mềm sẽ không bị định kiến (không bị ảnh hưởng bởi người khác) mà hoàn toàn tập trung vào vấn đề của mình.
- Quan sát sẽ khắc phục ngăn cách giữa kỹ sư phần mềm và người được phỏng vấn.
- Nhận được các hiểu biết tốt về môi trường công tác hiện tại, vấn đề và quá trình xử lý thông qua quan sát.

### ***Nhược điểm của quan sát:***

- Thời gian quan sát có thể không biểu diễn cho các công việc diễn ra thông thường.
- Thói quen dễ thay đổi do biết mình bị quan sát (người bị quan sát sẽ mất tự nhiên, hành động có thể bị ghò ép).
- Mất nhiều thời gian.

Người đi quan sát nên xác định cái gì sẽ được quan sát. Nên xác định thời gian cần thiết cho việc quan sát, hãy xin sự chấp thuận của cả người quản lý và cá nhân trước khi tiến hành quan sát.

#### **5.1.3.3. Ấn định công việc tạm thời.**

Không có gì thay thế được kinh nghiệm. Với một công việc tạm thời, bạn có được nhận thức đầy đủ hơn về các nhiệm vụ. Cũng vậy, đầu tiên bạn học các thuật ngữ hoàn cảnh sử dụng nó. Thời gian kéo dài từ 2 tuần đến 1 tháng đủ dài để bạn có thể quen với phần lớn các công việc thông thường và các tình huống ngoại lệ nhưng không được quá dài để trở thành chuyên gia thực sự đối với công việc.

Công việc tạm thời cho bạn cơ sở hình thức hoá các câu hỏi về chức năng nào của phương pháp hiện thời của công việc sẽ được giữ lại và cái nào sẽ bị loại trừ hoặc thay đổi, nghiên cứu được ngữ cảnh hiện tại. Có thể bằng công việc để thay thế cho các câu hỏi không thực hiện được. Bất lợi của công việc tạm thời là tốn thời gian và sự lựa chọn về thời gian có thể làm tối thiểu hoá vấn đề, không bao hết được các hoạt động hoặc thời gian. Một nhược điểm khác nữa là kỹ sư phần mềm có thể thiên kiến hoá về quá trình xử lý công việc (do tự mình đã làm), nội dung làm ảnh hưởng đến công việc thiết kế sau này.

#### **5.1.3.4. Họp nhóm (meeting)**

Meeting là việc tập trung từ 3 người trở lên trong một khoảng thời gian để thảo luận về một chủ đề nhất định. Meeting có thể vừa bổ sung vừa thay thế phỏng vấn bằng cách cho phép các thành viên kiểm tra lại các kết quả phỏng vấn cá nhân. Nó có thể thay thế phỏng vấn bằng cách cung cấp một diễn đàn cho các thành viên cùng tìm ra các yêu cầu và các giải pháp cho ứng dụng.

Meeting có thể làm lãng phí thời gian. Nói chung nếu meeting càng lớn thì càng ít ý kiến nhất trí và thời gian để đi đến quyết định sẽ kéo dài. Do vậy nên có kế hoạch ban đầu cho meeting. Lịch trình nên cung cấp trước cho các thành viên. Số lượng chủ đề cần thảo luận chỉ nên thấp hơn 5 chủ đề. Meeting nên có thời gian cố định và có địa điểm thống nhất cụ thể với các quyết định cần thiết.

Meeting không nên kéo dài quá 2 giờ để có thể đảm bảo được sự tập trung, chú ý của các thành viên.

***Ưu điểm của họp nhóm :***

- Có thể ra quyết định mà các thành viên đều phải tuân theo (đa số).
- Nhận được cả thông tin tổng hợp và chi tiết.
- Là phương pháp tốt cho các yêu cầu bên ngoài.
- Tập hợp được nhiều người dùng liên quan.

***Nhược điểm của họp nhóm:***

- Mất nhiều công sức thời gian và tiền bạc để chuẩn bị.
- Nếu số đại biểu nhiều sẽ tốn thời gian để ra được quyết định.
- Các ngắt quãng trong cuộc họp dễ làm mọi người phân tán.
- Dễ chuyển sang các chủ đề ít liên quan như : chính trị, thể thao, thời trang ...
- Mời không đúng thành viên dẫn đến chậm có kết quả.

**5.1.3.5. Điều tra qua bản câu hỏi**

Được ứng dụng khi cần lấy ý kiến của đại đa số người dùng về một số thông tin để có thể tập hợp số liệu thống kê mà không có điều kiện gặp trực tiếp. Với cách này, người thu thập dữ liệu sẽ soạn trước một bản câu hỏi, có thể có sẵn các phương án lựa chọn để người dùng lựa chọn đánh dấu vào, sau đó thu lại và thống kê kết quả.

Ví dụ, các câu hỏi có thể như sau :

Bạn thường ứng dụng máy tính vào các lĩnh vực nào sau đây ?

- A. Giải trí.      B. Công việc.      C. Do ý thích.      D. Không dùng.

Với cách thức này, người thu thập không cần mất thời gian gặp trực tiếp (như phỏng vấn hoặc họp nhóm) mà vẫn thu được thông tin, không đòi hỏi kỹ năng giao tiếp. Các câu hỏi trong danh sách có thể là dạng phỏng vấn trên giấy hoặc máy tính. Ưu điểm chính của câu hỏi là nếu như không cần phải chỉ rõ tên của người trả lời thì thông tin các câu trả lời sẽ có tính trung thực cao hơn. Cũng vậy, các câu hỏi chuẩn xác cung cấp các dữ liệu thực mà theo đó các quyết định có thể được dựa vào. Các mục câu hỏi, như là phỏng vấn có thể là câu hỏi mở hoặc đóng.

***Ưu điểm của bản câu hỏi :***

- Người cho ý kiến có thể không cần biết tên do vậy cho quan điểm và cảm nhận có tính trung thực cao, có thể dựa vào đó để ra quyết định.
- Có thể tiến hành với nhiều người.
- Thích hợp với các câu hỏi đóng và hữu hạn.
- Phù hợp với công ty đa chức năng và có thể tùy biến theo địa phương.

***Nhược điểm của bản câu hỏi :***

- Khó thực hiện lại được.
- Các câu hỏi không được trả lời không có nghĩa là không có thông tin.
- Các câu hỏi có thể khó hiểu do yêu cầu cần phải ngắn gọn
- Thực hiện đánh giá có thể chậm.
- Người dùng ít có khả năng đưa ra ý kiến khác (do tính đóng của các câu hỏi).
- Không thể bổ xung thêm thông tin khi đã tiến hành công bố các bản câu hỏi.

### 5.1.3.6. Xem xét tài liệu

Khái niệm tài liệu ám chỉ các cảm nang, quy định, các thao tác chuẩn mà tổ chức cung cấp như là hướng dẫn cho các nhà quản lý và nhân viên.

Các tài liệu không phải luôn nằm trong đơn vị đó. Tài liệu có thể là tài liệu nội bộ, có thể là các ấn phẩm kỹ thuật, các báo cáo nghiên cứu, ... Các tài liệu thực sự có ý nghĩa với kỹ sư phần mềm để tìm hiểu các lĩnh vực mà họ chưa từng có kinh nghiệm. Nó hữu ích cho việc xác định các câu hỏi về quá trình thao tác và sản xuất. Tài liệu đưa ra các thông tin mang tính khách quan.

***Tài liệu nội bộ*** mô tả được ngữ cảnh hiện thời ; phù hợp với việc nghiên cứu có tính lịch sử (quá trình hoạt động lâu dài). Tuy nhiên việc phải cung cấp tài liệu nội bộ làm cho người dùng e ngại, gây thành kiến ; khó có thể nhận biết được quan điểm, động cơ tiến hành công việc.

**Tài liệu ngoài** cho ta xác định được các khuynh hướng công nghiệp, ý kiến các chuyên gia, các kinh nghiệm của các công ty khác về thông tin, kỹ thuật. Tuy nhiên thông tin có thể không xác đáng, thiếu chính xác và có thể gây thành kiến.

### 5.1.3.7. Xem xét phần mềm

Một cách thường xuyên, các ứng dụng phải thay thế các phần mềm cũ. Hệ thống hiện tại có thể đã có phần mềm hỗ trợ từ trước. Nghiên cứu các phần mềm đã tồn tại cung cấp cho chúng ta các thông tin về quá trình xử lý công việc hiện thời và các mở rộng có ràng buộc bởi thiết kế phần mềm.

Khiếm khuyết của việc thu nhận thông tin từ việc xem xét phần mềm là tài liệu có thể không chính xác hoặc kịp thời, mà có thể không đọc được và thời gian có thể lãng phí nếu ứng dụng đã bị xóa bỏ.

### Kết luận

Thu thập dữ liệu là bước khởi đầu vô cùng quan trọng trong quá trình phát triển phần mềm cho hệ thống. Những thông tin thu thập được sẽ là căn cứ để xây dựng phần mềm và là bằng chứng xác thực các yêu cầu của người dùng có được đề cập và có được đáp ứng hay không? Thu thập dữ liệu có thể được tiến hành trong mọi giai đoạn của quá trình phát triển ứng dụng nhưng có các mục đích khác nhau. Các đặc tính cần lưu ý của dữ liệu cần thu thập là: *tính hướng thời gian; tính có cấu trúc; tính đầy đủ; tính không nhầm lẫn; ngữ nghĩa và độ lớn.*

Thu thập dữ liệu có thể theo nhiều kỹ năng: *phỏng vấn; điều tra qua bản câu hỏi; quan sát; hội họp; làm việc chung; ấn định công việc tạm thời; xem xét tài liệu và xem xét phần mềm hiện tại.* Mỗi kỹ năng có ưu điểm và nhược điểm riêng. Tuy nhiên ưu điểm của kỹ năng này có thể khắc phục nhược điểm của kỹ năng kia (ví dụ: các thông tin không thể hỏi được hoặc diễn đạt không rõ khi phỏng vấn thì có thể thim được trong quá trình làm việc chung). Tùy từng điều kiện hoàn cảnh cụ thể mà người đi thu thập tài liệu có thể áp dụng kỹ năng cho phù hợp. Mục đích chính vẫn là thu thập được nhiều thông tin có tính chân thực cao làm căn cứ cho các công việc sau này.

## 5.2. Quản lý dự án

**Quản lý dự án** là một từ mang sức nặng, được nói nhiều mà mang nghĩa cũng nhiều. Quản lý dự án cung cấp cho bạn các công cụ, tri thức và kỹ thuật để tiến hành xác định, lập kế hoạch, tổ chức, kiểm soát và kết thúc dự án. Nhưng *dự án là gì?*

Dự án có một *ngày bắt đầu* và một *ngày kết thúc*. Mọi dự án đều phải bắt đầu tại một điểm xác định trong thời gian và phải hoàn thành một lúc nào đó trong tương lai.



Cái gọi là "dự án" mà không có ngày kết thúc thì không phải là dự án; chúng không là gì khác hơn một sự liên tục vô tận của công việc thường lệ.

Dự án bao gồm các hoạt động để chuyển giao một sản phẩm cuối cùng. Con đường đi tới sản phẩm cuối này bao gồm việc chỉ ra các hoạt động để đi từ điểm A tới điểm Z. Một số bước phải được thực hiện theo trình tự logic, không ngẫu nhiên. Để làm bước B, trước hết bạn phải làm bước A. Dự án không có chỗ cho trình tự phi logic bởi vì nó chịu một loạt các ràng buộc áp chế như chi phí, ngân sách và lịch biểu.

Dự án công nghệ thông tin hiểu theo một khía cạnh nào đó là việc xây dựng một hệ thống thông tin cho một tổ chức. Đầu tiên là *xác định dự án*, xác định mục đích cuối cùng đạt được của dự án, xác định các vai trò và trách nhiệm của những người tham dự. Bước tiếp theo là *lập kế hoạch của dự án*, xác định cách nó sẽ hoàn thành các mục đích và mục tiêu của mình. Cách thức để hoàn thành mục đích và mục tiêu là tạo ra cấu trúc phân việc, xây dựng các ước lượng thời gian, xây dựng lịch biểu, cấp phát tài nguyên, tính chi phí, và quản lý rủi ro. Căn cứ vào đó, người ta sẽ *tổ chức thực hiện dự án*. Đồng thời cũng phải *kiểm soát dự án*, việc này đảm bảo đảm bảo rằng dự án được tiến hành theo kế hoạch. *Kết thúc dự án*, chúng ta cần thu thập toàn bộ tài liệu thống kê, hợp rút kinh nghiệm và thực hiện một số công việc để điều hoà lại toàn bộ hoạt động của nhóm phát triển hệ thống. Chuẩn bị cho dự án tiếp theo.

Một dự án công nghệ thông tin bao gồm các giai đoạn theo mô hình tuần tự như sau:

Giai đoạn	Mục đích	Các hoạt động chính	Tài liệu, điểm mốc	Công sức qlda
1. <b>Xác định</b>	Hiểu vấn đề và có ước lượng ban đầu	- Vấn đề - Mục tiêu - Kết quả - Đánh giá rủi ro	- Đề cương dự án và tài liệu khả thi - Bản đặc tả yêu cầu - Bảng các rủi ro - Bản kế hoạch ban đầu - Đề xuất giải pháp	90%
2. <b>Phân tích</b>	Hệ thống tổng thể cần phải làm gì	- Khảo sát, phân tích hệ thống - Thiết kế mức tổng thể - Đánh giá lại	- Bản đặc tả chức năng - Kế hoạch triển khai	60%
3. <b>Thiết kế</b>	Từng cấu phần hệ thống, cách hệ thống làm việc	- Thiết kế hệ thống - Quyết định mua hay làm - Duyệt xét chi tiết	- Bản đặc tả thiết kế - Bản kế hoạch chấp nhận - Bản kế hoạch đã được thông qua	30%

Giai đoạn	Mục đích	Các hoạt động chính	Tài liệu, điểm mốc	Công sức qlda
4. Thực hiện	Xây dựng các cấu phần	<ul style="list-style-type: none"> <li>- Lập trình</li> <li>- Mua phần mềm</li> <li>- Chuyên biệt hoá</li> <li>- Kế hoạch kiểm thử</li> </ul>	<ul style="list-style-type: none"> <li>- Bản thiết kế cho từng cấu phần</li> <li>- Kế hoạch kiểm thử hệ thống</li> <li>- Tài liệu cho người dùng</li> </ul>	10%
5. Kiểm thử hệ thống	Hệ thống làm việc tốt, không có lỗi	<ul style="list-style-type: none"> <li>- Kiểm thử từng phần</li> <li>- Kiểm thử hệ thống</li> <li>- Đảm bảo chất lượng</li> </ul>	<ul style="list-style-type: none"> <li>- Báo cáo kết quả tích hợp hệ thống</li> </ul>	10%
6. Kiểm thử chấp nhận	Người dùng chấp nhận hệ thống	Thực hiện qui trình demo đã định.	<ul style="list-style-type: none"> <li>- Báo cáo kết quả qui trình demo</li> </ul>	40%
7. Vận hành	Vận hành và hoàn thiện	<ul style="list-style-type: none"> <li>- Vận hành</li> <li>- Chuyển đổi</li> <li>- Đào tạo</li> <li>- Hỗ trợ</li> <li>- Rút kinh nghiệm</li> </ul>	<ul style="list-style-type: none"> <li>- Bản kế hoạch hỗ trợ</li> <li>- Báo cáo kết quả đào tạo</li> <li>- Kinh nghiệm đúc kết</li> </ul>	20%

### 5.2.1. Vai trò của cán bộ quản lý dự án.

Trong mục này chúng ta sẽ đề cập đến các vị trí cán bộ đóng vai trò quan trọng trong một dự án công nghệ thông tin. Đó là kỹ sư phần mềm và quản trị viên dự án. Vai trò của kỹ sư phần mềm (*SE – Software Engineer*) khác so với quản trị viên dự án (*PM – Project Manager*) bởi SE chuyên về mặt công nghệ trong đó PM chuyên về mặt tổ chức. PM chịu trách nhiệm duy nhất về sự liên lạc trong tổ chức, quản lý nhân viên của dự án, giám sát và điều khiển dự án. Tùy vào quy mô của tổ chức và nhóm dự án, một cá nhân có thể đảm nhận một hoặc cả hai vai trò trên. Với nhóm dự án nhỏ (khoảng 5 người) và tổ chức với nhóm phát triển phần mềm hữu hạn (ít hơn 10 người) thì một người sẽ thường đảm nhận cả hai vai trò trên. Còn với tổ chức lớn hơn, các chức năng có khả năng được chia nhỏ và các kinh nghiệm của cá nhân cần tăng thêm tính bao quát.

Việc quản lý dự án là hoạt động mang nặng tính con người, và bởi lý do này, những người hành nghề có khả năng thường là người lãnh đạo tổ tối. Họ đơn giản không có sự trộn lẫn đúng kỹ năng con người. Vậy mà như Edgemon đã phát biểu: "Không may và quá thường xuyên là các cá nhân chỉ rơi vào trong vai trò người quản trị dự án và ngẫu nhiên trở thành người quản trị dự án."

Trong một cuốn sách tuyệt vời về quyền lãnh đạo kỹ thuật, Jerry Weinberg gợi ý một mô hình MOI về quyền lãnh đạo:

**Động viên.** Khả năng cổ vũ (bằng "kéo" hay "đẩy") người kỹ thuật vào sản xuất với khả năng tốt nhất của họ.

**Tổ chức.** Khả năng tạo khuôn cho các tiến trình hiện có (hay phát minh ra tiến trình mới) vốn làm cho khái niệm ban đầu được dịch thành sản phẩm cuối cùng.

**Ý tưởng hay đổi mới.** Khả năng động viên mọi người tạo ra và cảm thấy sáng tạo ngay cả khi họ phải làm việc trong những giới hạn được thiết lập cho sản phẩm hay ứng dụng phần mềm đặc biệt.

Weinberg gợi ý rằng người lãnh đạo dự án thành công áp dụng một phong cách quản lý việc giải quyết vấn đề. Tức là, người quản lý dự án phần mềm phải tập trung vào việc hiểu vấn đề cần được giải quyết, quản lý luồng tư tưởng, và đồng thời để cho mọi người trong tổ biết (bằng lời nói và - điều quan trọng hơn nhiều - bằng hành động) rằng cần tính tới chất lượng và điều đó sẽ không bị thỏa hiệp.

Một quan điểm khác về các đặc trưng vốn xác định ra người quản trị dự án có hiệu quả nhấn mạnh vào bốn nét chính sau:

**5.2.2.4. Giải quyết vấn đề.** Người quản lý dự án phần mềm hiệu quả có thể chẩn đoán các vấn đề kỹ thuật và tổ chức vốn là cấu trúc liên quan nhất, có tính hệ thống tới giải pháp hay động viên đúng đắn những người hành nghề khác để phát triển giải pháp, áp dụng các bài học đã biết từ các dự án quá khứ vào những tình huống mới, và vẫn còn đủ linh hoạt để thay đổi chiều hướng nếu những nỗ lực ban đầu về giải pháp vấn đề không có kết quả.

**5.2.2.5. Tư cách quản lý.** Người quản lý dự án tốt phải nhận trách nhiệm về dự án. Người đó phải có sự tự tin để đảm bảo kiểm soát khi cần và đảm bảo cho phép người kỹ thuật giỏi đi theo bản năng của mình.

**5.2.2.6. Thành tựu.** Làm tối ưu hoá hiệu suất của tổ dự án, người quản trị phải thưởng cho sáng kiến và việc hoàn thành và biểu lộ qua hành động riêng của người đó rằng việc nhận rủi ro có kiểm soát sẽ không bị phạt.

**5.2.2.7. Ảnh hưởng và xây dựng tổ (nhóm).** Người quản trị dự án có hiệu quả phải có khả năng "đọc" mọi người. Người đó phải có khả năng hiểu các tín hiệu lời hay không lời và phản ứng với nhu cầu của người gửi thông báo đó. Người quản trị vẫn còn phải kiểm soát được trong tình huống có sự dồn nén cao.

### **5.2.3. Các hoạt động chuẩn bị dự án**

Tham gia hoạt động chuẩn bị dự án của quản trị viên dự án và kỹ sư phần mềm bao gồm lên kế hoạch và điều khiển dự án, đăng ký đội ngũ nhân viên làm nhiệm vụ và lựa chọn giữa một hay nhiều giải pháp khác nhau.

Người quản trị viên dự án trước hết cần lập kế hoạch cho dự án (project planning), quản trị viên cần làm việc với kỹ sư phần mềm để xác định nhân tố con người, máy tính và các tài nguyên tổ chức được yêu cầu để phát triển ứng dụng.

Một kế hoạch dự án chính là một sơ đồ của các nhiệm vụ, thời gian và các mối quan hệ giữa chúng. Nó có thể rất chung hoặc rất riêng biệt. Phương pháp lập kế hoạch thường được dùng là lập sơ đồ tuần tự tương tác, biểu đồ đường găng và biểu đồ tương tác mô hình mạng.

Phương pháp luận chung trong việc lên kế hoạch gồm các bước sau:

- *Liệt kê các nhiệm vụ:* Bao gồm các nhiệm vụ phát triển ứng dụng, các nhiệm vụ đặc trưng của dự án, các nhiệm vụ về tổ chức giao diện, sự xem xét lại và các phê chuẩn.
- *Xác định sự phụ thuộc giữa các công việc.* Các công việc có thể có liên quan nhau theo các hình thức: dây truyền kết nối (kết quả của công việc này là đầu vào của công việc tiếp theo) hoặc là theo kiểu hỗ trợ lẫn nhau.
- *Ấn định nhân viên tùy theo tên hoặc kỹ năng và mức kinh nghiệm:* Căn cứ và yêu cầu công việc, khả năng đáp ứng của đội ngũ những người phát triển phần mềm,... quản trị viên dự án phải xác định số lượng, chất lượng những người cần cho dự án (yêu cầu về nhân viên đã được đề cập ở chương 2).
- *Lập lịch biểu:* Ấn định thời gian hoàn thành cho công việc; tính toán hợp lý nhất cho các công việc: căn cứ vào yêu cầu của khách hàng, khả năng của từng nhân viên trong đội ngũ phát triển, quản trị viên dự án phải tiến hành lập lịch biểu cho dự án (chính là xác định mốc thời gian bắt đầu và thời gian kết thúc của mỗi công đoạn trong quá trình phát triển hệ thống). Lịch biểu giúp cho tất cả thành viên biết được giới hạn về thời gian cho từng công đoạn và lượng công việc cần hoàn thành trong khoảng thời gian xác định, tuân thủ theo mốc thời gian đã định sẵn giúp đội ngũ phát triển hoạt động nhịp nhàng, đồng bộ.
- *Xác định hướng đi tới hạn:* ở đây có thể nêu ra kết quả cuối cùng cần đạt đến. Công suất (năng lực) của hệ thống phần mềm có thể đạt được.

Những công việc trong quá trình tổ chức gồm các phần sau:

- Xem xét các tài liệu theo khía cạnh đầy đủ, nội dung, sự tin cậy và độ chắc chắn;
- Thương lượng, thỏa thuận và cam kết ngày bắt đầu và ngày kết thúc cho công việc;

- Xác định mọi cách thức tương tác (giao diện) giữa các ứng dụng cần thiết; đặt kế hoạch cho các thiết kế cách thức tương tác chi tiết.

Tất cả các tài liệu, kế hoạch và công việc thiết kế của một đội ngũ thiết kế là phụ thuộc vào người sử dụng. Nhiều bộ phận hay tổ chức khác có thể phải xem xét lại một số hoặc tất cả các công việc trên. Những tổ chức này bao gồm nhà quản lý hệ thống thông tin (Information System - IS), người sử dụng, kiểm toán, các nhà làm luật chính phủ hay các nhà làm luật trong ngành... Mỗi một tổ chức có thể đưa ra những kiến thức chuyên môn của mình vào những tài liệu ứng dụng một cách phù hợp.

Nhiệm vụ thứ 2 là để đạt được sự đồng ý, cam kết từ các ngành, phòng ban bên ngoài. Thông thường các nguồn tài liệu là do các phòng ban khác cung cấp (từ giai đoạn khảo sát hệ thống). Ví dụ như, sự hỗ trợ của nhân viên có thể từ phòng quản lý hành chính.

#### **5.2.4. Giám sát và điều khiển dự án.**

Về mặt lý thuyết hầu hết (nếu không nói là tất cả) các hoạt động quản lý dự án đã thảo luận trong các chương trước đều áp dụng được cho dự án kỹ nghệ phần mềm. Nhưng trong thực hành, cách tiếp cận kỹ nghệ phần mềm tới việc quản lý dự án là khác biệt đáng kể.

##### **5.2.4.1. Khởi đầu dự án.**

Cho dù việc khoán ngoài là chiến lược được chọn cho việc phát triển ứng dụng phần mềm (UDPM), một tổ chức phải thực hiện một số các nhiệm vụ trước khi tìm nhà cung cấp khoán ngoài để làm công việc này:

*Nhiều hoạt động phân tích đã thảo luận trong phần trước nên được thực hiện nội bộ.* Người dùng của ứng dụng phần mềm nên được xác định rõ; những người bảo trợ nội bộ, người có thể có mối quan tâm tới UDPM được liệt kê ra; những mục tiêu tổng thể cho UDPM được xác định và xem xét; thông tin và dịch vụ được chuyển giao bởi UDPM được xác định; các phần mềm cạnh tranh được lưu ý tới; và "cách đo" định tính và định lượng về UDPM thành công được xác định. Thông tin này nên được làm tư liệu trong đặc tả sản phẩm.

*Thiết kế đại thể cho UDPM nên được phát triển nội bộ.* Hiển nhiên, người phát triển phần mềm cỡ chuyên gia sẽ tạo ra một thiết kế đầy đủ, nhưng thời gian và chi phí có thể được tiết kiệm nếu cái nhìn chung về UDPM được vạch ra cho người làm khoán ngoài (điều này bao giờ cũng có thể được sửa đổi trong giai đoạn sơ bộ của dự án). Thiết kế phải bao hàm một chỉ báo về kiểu và khối lượng nội dung được UDPM trình bày và kiểu xử lý tương tác (như các mẫu, đưa vào đơn) cần được thực hiện. Thông tin này nên được bổ sung vào đặc tả sản phẩm.

*Một lịch dự án đại thể, không những chứa ngày tháng chuyển giao cuối cùng mà còn cả những ngày tháng cột mốc, phải được xây dựng ra. Cột mốc nên được gắn vào phiên bản chuyển giao được của UDPM khi nó tiến hoá.*

*Mức độ giám sát và tương tác bởi người hợp đồng với nhà cung cấp nên được định rõ. Điều này nên bao hàm việc chỉ tên người liên hệ của nhà cung cấp và xác định trách nhiệm cùng chủ quyền của người liên hệ, xác định các điểm xét duyệt chất lượng khi việc phát triển được thực hiện, và trách nhiệm của nhà cung cấp đối với truyền thông liên tổ chức.*

#### **5.2.4.2. Lựa chọn nhà cung cấp khoá ngoài ứng cử viên**

Trong những năm gần đây, hàng nghìn công ty “thiết kế phần mềm” đã nổi lên để giúp các doanh nghiệp thiết lập sự hiện diện phần mềm hay tham gia vào thương mại điện tử. Nhiều người đã trở thành lão luyện với tiến trình kỹ nghệ phần mềm, nhưng nhiều người khác vẫn chỉ như một người mê say. Để tuyển người phát triển phần mềm ứng cử viên, người có hợp đồng phải siêng năng thực hiện:

1. Phỏng vấn khách hàng quá khứ để xác định tính chuyên nghiệp của người cung cấp phần mềm, khả năng đáp ứng những cam kết lịch biểu và chi phí, và khả năng trao đổi có hiệu quả.
2. Xác định tên của kỹ sư phần mềm chính của nhà cung cấp về những dự án quá khứ thành công (và về sau, hãy chắc chắn rằng người này có nghĩa vụ về mặt hợp đồng để được tham gia vào dự án của bạn).
3. Xem xét cẩn thận các mẫu công việc của nhà cung cấp vốn tương tự về cái nhìn và cảm giác (và miền nghiệp vụ) với UDPM dự định ký hợp đồng. Thậm chí trước khi yêu cầu về giá được đưa ra, cuộc họp mặt đối mặt có thể cung cấp cái nhìn chủ chốt vào "sự thích hợp" giữa người có hợp đồng và nhà cung cấp.

#### **5.2.4.3. Thẩm định tính hợp lệ về giá cả và độ tin cậy của các ước lượng**

Bởi vì còn tương đối ít dữ liệu lịch sử và phạm vi của UDPM còn hay thay đổi dễ sợ, nên việc ước lượng mang tính rủi ro cố hữu. Bởi lý do này, một số nhà cung cấp sẽ giữ lấy lẽ an toàn chủ yếu trong giá cho dự án. Điều này vừa là hiểu được và thích hợp.

*Mức độ quản lý dự án bạn có thể trông đợi hay thực hiện. Tính hình thức liên kết với các nhiệm vụ quản lý dự án (được thực hiện bởi cả người cung cấp và người hợp đồng) tỉ lệ với kích thước, chi phí và độ phức tạp của UDPM. Với các dự án phức tạp, lớn, một lịch biểu dự án chi tiết vốn xác định ra các nhiệm vụ công việc, các điểm kiểm tra, các sản phẩm công việc đã làm, các điểm xét duyệt của khách hàng, và những cột mốc chính nên được phát triển. Người cung cấp và người hợp đồng nên thẩm định các rủi ro cùng nhau và phát triển những kế hoạch để di chuyển, điều phối và quản lý những rủi ro có vẻ quan trọng. Các cơ chế cho việc đảm bảo chất lượng và kiểm soát*

thay đổi nên được xác định tường minh dưới dạng văn bản viết. Các phương pháp cho việc trao đổi hiệu quả giữa người hợp đồng và người cung cấp nên được thiết lập.

*Thẩm định lịch phát triển.* Bởi vì lịch phát triển UDPM trải ra trong một thời kì tương đối ngắn (thường ít hơn một hay hai tháng), nên lịch phát triển nên có độ bao gói (packet) cao. Tức là các nhiệm vụ công việc và những mốc làm việc nhỏ nên được lập lịch hàng ngày. Việc bao gói (khoán) thời gian này cho phép cả nhà cung cấp và người hợp đồng nhận ra việc trượt lịch trước khi nó đe dọa tới ngày tháng hoàn thành.

#### **5.2.4.4. Quản lí phạm vi.**

Bởi vì rất có thể là phạm vi sẽ thay đổi khi dự án UDPM tiến triển, nên mô hình tiến trình kĩ nghệ phần mềm nên là tăng dần (mô hình xoáy ốc). Điều này cho phép tổ phát triển "làm đông cứng" phạm vi cho một lần tăng để cho việc đưa UDPM ra vận hành có thể được tạo ra. Lần tăng tiếp có thể đề cập tới những thay đổi phạm vi được gợi ý bởi việc duyệt xét lần tăng trước, nhưng một khi lần tăng thứ hai bắt đầu, thì phạm vi lại bị làm đông cứng tạm thời. Cách tiếp cận này tạo khả năng tổ kĩ nghệ phần mềm làm việc không phải điều tiết luồng thay đổi liên tục nhưng vẫn nhận ra đặc trưng tiến hoá liên tục của hầu hết UDPM.

Những hướng dẫn này không có chủ ý làm sách ướng dẫn cho việc tạo ra UDPM giá thấp, đúng hạn. Tuy nhiên, chúng sẽ giúp cho cả người hợp đồng và nhà cung cấp khởi đầu công việc một cách trôi chảy với tối thiểu việc hiểu lầm.

#### **5.2.4.5. Phân công nhiệm vụ cho nhân viên.**

Phân công nhiệm vụ phải minh bạch và công bằng. Vấn đề chính là phải xác định các nhiệm vụ và kỹ năng cần có để thực hiện các nhiệm vụ. Quản trị viên lập danh sách các kỹ năng và khả năng thực thi các nhiệm vụ của nhân viên có thể làm trong dự án rồi dựa vào đó mà phân công nhiệm vụ đó cho từng người (điều này có thể căn cứ vào kết quả thực hiện các dự án trước đó hoặc bản trích lý lịch). Quản trị viên dự án bắt đầu thảo luận thực sự với các thành viên có thể khi họ đang vạch kế hoạch dự án và ướm thử nhiệm vụ cho từng người. Sau đó, công việc thực sự của quản trị viên dự án mới bắt đầu.

Phần khó khăn nhất của phân công nhiệm vụ là sự đánh giá cần thiết để xác định nhiệm vụ cho từng người mà kỹ năng công việc của họ khó mà xác định được một cách chính xác; đây là một vấn đề thường gặp.

Quản trị viên dự án nên tìm hiểu để biết rõ về các thành viên của nhóm. Điều này có nghĩa là đánh giá vị trí của họ trong công ty; mong đợi của họ đối với dự án; vai trò, công việc mà họ ưa thích; thời điểm để bắt đầu và kết thúc công việc; những tính cách và vấn đề cá nhân có thể ảnh hưởng tới công việc của họ. Nhiều trong số các thông tin

này có thể lấy từ các tài liệu đã ghi chép trước đó. Nhưng nhất thiết và không có gì thay thế được việc thảo luận trực tiếp với nhân viên hoặc trưởng nhóm.

Người quản trị viên dự án chịu trách nhiệm trước giám đốc, trước người bảo trợ khách hàng và những thành viên còn lại của dự án là phải có được đội ngũ tốt nhất, có đủ điều kiện nhất có thể được. Để đạt được điều đó, quản trị viên dự án phải thảo luận một cách trung thực, thẳng thắn về mọi vấn đề với nhân viên, tất cả các khúc mắc về mặt cá nhân mà có thể gây ảnh hưởng đến sự tập trung của nhân viên với công việc, bất cứ điều gì ngoài công việc, hay bất cứ trách nhiệm gì khác có thể gây hại tới công việc của chính họ. Nhân viên và quản trị viên dự án phải tạo cơ hội cho nhau để có thể chấp nhận hay loại bỏ các khả năng về công việc. Thậm chí ngay cả khi không tìm được cách giải quyết thì trách nhiệm của quản trị viên dự án là phải làm rõ được những mong muốn của nhân viên về chất lượng và khối lượng công việc. Nếu nhân viên không nói điều này trong cuộc họp trực tiếp với quản trị viên dự án thì có thể báo cáo sau cuộc họp. Theo cách này, mọi người có thể biết được chính xác vấn đề gì đã được đề cập và trách nhiệm nào đã được đảm nhận.

Sau đây là những kinh nghiệm hay các nguyên tắc giải quyết vấn đề phân công công việc cho nhân viên:

1. *Phân công người tốt nhất cho công việc quan trọng và phức tạp nhất.* Phân công tất cả các công việc quan trọng. Sau đó theo độ giảm dần của kinh nghiệm và trình độ kỹ năng của các nhân viên mà phân công các công việc ít phức tạp và nhỏ hơn. Không giao bất cứ việc gì quan trọng cho nhân viên mới, ít thâm niên hay chưa có đủ điều kiện. Giao những công việc quan trọng cho nhân viên có thâm niên sẽ làm giảm khả năng không hoàn thành nhiệm vụ đúng hạn định.
2. *Xác lập một chuỗi công việc cho nhân viên để họ có thể ở lại với công việc tới chừng nào kỹ năng của họ còn cần thiết cho dự án.* Cố gắng phân công những công việc cho phép phát triển kỹ năng của nhân viên.
3. *Không giao cho bất kỳ người nào khối lượng công việc quá tải so với thời gian làm việc của họ.* Đảm bảo rằng mỗi người đều phải làm nhiều việc nhưng phải kết thúc công việc này trước khi bắt đầu một công việc khác.
4. *Cho phép những khoảng thời gian ngừng làm việc ngẫu nhiên cho mỗi người (2÷5%);* điều này góp phần tạo hứng thú cho nhân viên. Tuy nhiên không để bất kỳ nhân viên nào nghỉ liên tục 8 giờ đồng hồ (một ngày làm việc).
5. *Không nên đặt kế hoạch làm việc ngoài giờ.* Điều này gây ra những căng thẳng bất thường trong công việc nghề nghiệp và cá nhân của nhân viên, đồng thời cũng là hợp lý vì phải tôn trọng thời gian làm việc chính thức của nhân viên và tránh cho nhân viên cũng lên “kế hoạch” để từ chối.



Cuối cùng, quản trị viên dự án phải đảm bảo rằng mọi người đều hiểu được trách nhiệm và nghĩa vụ được giao của mình

### 5.2.5. Quản lý nhân sự

Việc quản lý nhân sự trong dự án là việc quản trị viên dự án thuê nhân công, sa thải, hướng dẫn, tạo động lực thúc đẩy, lập kế hoạch, đào tạo và huấn luyện và thẩm định các nhân viên tham gia dự án [1].

#### 5.2.5.1. Thuê mượn nhân sự

Việc thuê mượn nhân sự thường được điều hành thông qua phòng nhân sự. Phòng này xem xét các nhân sự thuê, không chỉ trong một dự án mà có thể nhiều dự án. Căn cứ kế hoạch của dự án đòi hỏi về nhân sự, phòng nhân sự tiến hành *quảng cáo qua báo chí* hoặc các phương tiện thông tin khác, qua đó nhận được hồi âm từ phía các ứng viên (các quảng cáo tìm người thông thường mất từ 1 đến 2 tuần).

Phòng nhân sự tiến hành *sàng lọc công khai, rành mạch các hồ sơ xin việc*, việc phân loại này có thể dựa theo các yếu tố: trình độ, bằng cấp, kinh nghiệm, độ tuổi, giới tính,... Sau đó phòng nhân sự sẽ đệ trình lên quản trị viên dự án thông tin về các thí sinh và *chuẩn bị cho các cuộc thi tuyển, phỏng vấn*.

Thời gian cho việc tuyển người thường mất khoảng 7 tuần hoặc lâu hơn. Thời gian phỏng vấn có thể tiến hành tùy thuộc vào sự sắp xếp của ban quản lý và còn phụ thuộc vào lịch biểu của ban giám khảo cũng như các ứng viên. Thông thường những ứng viên hiện đang có việc làm không muốn tiêu tốn nhiều thời gian cho việc thi tuyển. Nếu một người được xem là xứng đáng thì quản trị viên dự án có thể phải thay đổi lịch biểu phỏng vấn cho phù hợp với ứng viên đó (theo phương châm coi trọng người tài).

#### 5.2.5.2. Sa thải

Giữ lại một người trong công việc mà họ không phù hợp sẽ làm hại cho người quản lý, con người và dự án hơn là bạn nghĩ. Những quản trị viên dự án bị thiệt hại bởi vì anh ta không nghĩ gì khác mà chỉ lo lắng về một quyết định còn lâu mới thực sự cần thiết. Mọi người cũng thường biết rằng họ sẽ bị sa thải nếu như không hoàn thành công việc của mình; họ phải được thông báo bằng văn bản, trước ngày chấm dứt công việc.

Kéo dài thời gian ra quyết định chấm dứt công việc làm thiệt hại đến cả người bị sa thải bởi vì sẽ đưa họ đến tâm trạng thất bại, làm cho họ mất niềm tin vào mọi người như những gì họ mô tả và có thể gây ảnh hưởng tiêu cực đến các thành viên khác trong dự án. Người bị sa thải có thể than phiền với các thành viên khác về tình trạng của họ và làm gián đoạn công việc (vì nhiều người biết sẽ càng tiêu tốn thời gian hơn). Những người còn lại sẽ có tâm trạng hoang mang, lo lắng và có thể mất lòng tin vào quản trị viên dự án.

Thông thường có một giai đoạn triệu chứng trước khi nảy sinh vấn đề. Vào thời gian này, quản trị viên dự án lên thảo luận với nhân viên về tình hình vấn đề. Tạo điều

kiện cho nhân viên có thể trình bày nguyên nhân. Nếu thấy nguyên nhân là hợp lý thì ít nhất cũng phải có một thư cảnh cáo và được đưa vào trong hồ sơ nhân sự của họ. Tiếp đó có thể khiển trách, nói rõ sai phạm và các lý do để phán xét. Khi khiển trách cũng nên nói rõ việc sa thải sẽ diễn ra nếu một số công việc (được lên danh sách) không được hoàn thành vào thời gian tới... Tất cả đề phải thể hiện bằng văn bản, được người quản lý hệ thống thông tin và người điều hành nhân sự xem xét và tán thành, chúng sẽ là cơ sở cho những tranh chấp về pháp luật trong tương lai với người làm công.

Nếu công việc được hoàn thành, việc sa thải sẽ bãi bỏ và ngược lại họ sẽ bị sa thải. Việc sa thải ra khỏi dự án không có nghĩa là sa thải họ khỏi công ty. Nếu họ phù hợp với công việc, vẫn có thể tuyển chọn vào dự án khác, vì họ vẫn có thể là nhân viên giỏi. Một quản trị viên dự án giỏi là người biết sắp xếp nhân sự vào vị trí đúng với khả năng và sở thích của người lao động.

### 5.2.5.3. Tạo động lực thúc đẩy

Động lực thúc đẩy (ĐLTĐ) mang những khía cạnh riêng tư và chuyên nghiệp. Động lực chuyên nghiệp phát sinh từ mong muốn làm một công việc tốt. Mọi người được khuyến khích làm một tốt khi họ được đối xử như một người có khả năng làm việc chuyên nghiệp và được làm công việc thú vị, có ý nghĩa, được đưa ra các quyết định tự do và các phát kiến sáng tạo.

ĐLTĐ cá nhân xuất phát từ mong muốn cải thiện vị trí của người đó trong cuộc sống. Vị trí đó có thể mang tính rất riêng tư ví dụ như có thể kiếm được nhiều tiền hơn, mua được ngôi nhà to hơn, trở thành nhà phân tích hoặc một nhà quản lý v.v..

Cách thức quản lý dự án là nhân tố quyết định của ĐLTĐ cá nhân. Một quản trị viên dự án biết tạo điều kiện cho nhân viên làm việc, kiểm soát, hạn chế rủi ro và cho phép mọi nhu cầu cá nhân (hợp lý) đều được đáp ứng thì sẽ nhận được sự trung thành bất tận nơi nhân viên. Một quản trị viên dự án mà chỉ coi nhân viên của mình là những người kém cỏi, lười biếng và chậm chạp thì cũng có thể nhận được những kết quả mong muốn nhưng phải thông qua sự hăm dọa và ép buộc, đồng thời gây thành kiến với nhân viên.

Quản trị viên dự án phải sâu sát, tường tận từng nhân viên để có thể động viên khen thưởng và phân chia công việc một cách hợp lý nhằm giúp họ đạt được mục đích. Cam kết giúp đỡ nhân viên đạt được mục đích cá nhân của quản trị viên dự án sẽ quyết định ĐLTĐ chuyên nghiệp của các nhân viên. ĐLTĐ có 3 khía cạnh:

- *Thứ nhất*, bản thân các công việc trong dự án có thể được sử dụng cho các mục tiêu chuyên nghiệp khác nữa bao gồm các công việc mới và tiến tới một vị trí mới, kinh nghiệm mới và trách nhiệm mới.

- *Thứ hai*, quản trị viên dự án phải thật cẩn thận trong việc thưởng phạt để đáp ứng nhu cầu công việc, phải đánh giá khách quan mức độ và tầm quan trọng các đóng góp của nhân viên cho công việc.
- *Thứ ba*, nhân viên phải cam kết làm việc tích cực hơn nữa để nhận được khen thưởng.

#### **5.2.5.4. Hoạch định nghề nghiệp**

Nhân viên phải được biết kế hoạch của đơn vị mình trong thời gian tới và có định hướng rõ ràng, hiểu được dự kiến về công việc trong tương lai. Họ làm xong công việc này thì có thể thực hiện tiếp các công việc gì. Từ đó nhân viên có thể xác định khoảng thời gian có thể ở lại với dự án.

Quản trị viên phải đưa ra kế hoạch cụ thể rõ ràng. Tuân theo một lộ trình nhất định: Ví dụ, trong 6 tháng đầu làm gì, tiếp theo làm gì...

#### **5.2.5.5. Đào tạo, huấn luyện**

#### **5.2.5.6. Thẩm định**