NHẬP MÔN CÔNG NGHỆ PHẦN MỀM

TÀI LIỆU ĐỌC

TẬP THỂ TÁC GIẢ:

TS. VŨ THỊ HƯƠNG GIANG

TS. BÙI THỊ MAI ANH

TS. NGUYỄN NHẤT HẢI

TS. TRẦN NHẬT HOÁ

TS. TRỊNH THÀNH TRUNG

THS. NGUYỄN MẠNH TUẤN

ĐƠN VỊ: KHOA KHOA HỌC MÁY TÍNH

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HÀ NỘI, 10/2023

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG

Chương 1 TỔNG QUAN VỀ CÔNG NGHỆ PHẦN MỀM

Nội dung:

- Khái niệm và các đặc trưng của phần mềm
- Các khái niệm trong Công nghệ phần mềm
- Vai trò của Công nghệ phần mềm
- Các vấn đề trong Công nghệ phần mềm

1.1 Khái niệm và các đặc trưng của phần mềm

1.1.1 Định nghĩa phần mềm

Trước khi học về các khái niệm tổng quan của công nghệ phần mềm, chúng ta cần phải hiểu rõ về phần mềm là gì. Có lẽ đối với hầu hết các bạn học khóa học này thì đều đã có một hiểu biết nhất định về các khái niệm của phần cứng và phần mềm. Để định nghĩa một cách đơn giản, chúng ta có thể hiểu là: Trong một hệ thống máy tính, nếu bỏ đi phần cứng, là các thiết bị và các loại phụ kiện, thì phần còn lại chính là phần mềm.

Hệ thống máy tính = Software + Hardware

Nếu hiểu theo nghĩa hẹp thì phần mềm là dịch vụ chương trình để tăng khả năng xử lý của phần cứng của máy tính, do bản than phần mềm không có các thứ "cứng" để tương tác với thế giới bên ngoài được. Còn hiểu theo nghĩa rộng thì phần mềm là tất cả các kỹ thuật ứng dụng để thực hiện những dịch vụ chức năng cho mục đích nào đó bằng phần cứng.

Như vậy, có thể hiểu là một hệ thống máy tính, là cái mà con người tương tác trực tiếp đến, là sự kết hợp của cả phần cứng và phần mềm, và cái mà chúng ta tương tác đến là cả phần cứng và phần mềm. Tuy nhiên, vai trò của phần cứng và phần mềm không phải lúc nào cũng như nhau. Ví dụ như hiện nay, con người chú trọng hơn ở vai trò của phần mềm, và phần cứng chỉ là hỗ trợ cho phần mềm. Các bạn có thể giới thiệu cho mọi người có những phần mềm, trò chơi mới này hay lắm. Nhưng thực tế, để có thể chạy được phần mềm đó, chơi được trò chơi đó, bạn vẫn cần phải có sự hỗ trợ của phần cứng. Tối thiểu bạn sẽ cần có màn hình để hiển thị được các hình ảnh ra, ngoài ra bạn

có thể phải cần một lớp cảm ứng dưới màn hình, hay một tay cầm chơi game. Và ngoài ra, còn có sự hỗ trợ tính toán của các CPU, GPU v.v… nữa.

Có thể đối với các bạn, đây là việc hết sức bình thường, nhưng trước đây, phần mềm lại là thứ để hỗ trợ cho phần cứng. Vào thời buổi máy tính mới ra đời, hầu như toàn bộ các chức năng của máy tính là được thực hiện bởi phần cứng. Thời buổi đó, máy tính có thể to bằng các căn phòng, và "phần mềm" được soạn bằng các tấm giấy lớn được đục lỗ để đưa vào máy tính. Khi đấy, các phần mềm chỉ là các chỉ dẫn đơn giản để tận dụng các tính năng có sẵn của phần cứng. Lý do đơn giản là vì các máy tính được tạo ra để phục vụ với đúng cái tên của nó, máy để tính.

Theo định nghĩa như vậy thì chúng ta có thể thấy rất nhiều ví dụ về phần mềm. Không chỉ là các ứng dụng các bạn download ở trên App Store, phần mềm có thể là các website chúng ta đang vào hàng ngày. Bản thân hệ điều hành, như Windows, Android, iOS cũng là các phần mềm. Hay là các trình dịch, cũng là các phần mềm (cụ thể là các phần mềm để tạo ra phần mềm). Một số các ví dụ khác như các phần mềm để hỗ trợ cho các hoạt động của phần cứng, như các phần mềm router, switch,... các phần mềm nhúng như driver, controller..., các phần mềm thiết bị di động như phần mềm cho máy ảnh, GPS, cảm biến...

Ngoài ra thì chúng ta có các ví dụ về phần mềm quen thuộc như các phần mềm phục vụ cho các mục đích cụ thể. Ví dụ các phần mềm xử lý dữ liệu: Hóa đơn điện thoại, dự đoán thị trường tài chính; ứng dụng thời gian thực: Kiểm soát không lưu, phương tiện tự hành; hệ thống thông tin: Quản lý CSDL, thư viện số...; ứng dụng văn phòng: Xử lý văn bản, bảng tính, hội thảo video…; ứng dụng khoa học: Mô phỏng, dự báo thời tiết...

Để định nghĩa một cách chính xác về phần mềm, Roger Pressman đã đưa ra định nghĩa về phần mềm như sau: Một phần mềm bao gồm:

- Các **lệnh** (chương trình máy tính) cung cấp những chức năng và kết quả mong muốn khi được thực hiện.
- Các **cấu trúc dữ liệu** làm cho chương trình thao tác với các thông tin tương ứng.
- Các **tài liệu** mô tả thao tác và cách sử dụng chương trình.

1.1.2 Các đặc trưng của phần mềm

Đầu tiên, cái mà chúng ta có thể thấy được ngay là phần mềm là một loại hàng hóa "*vô hình*", không nhìn thấy được, khác với cả phần cứng là thứ mà chúng ta có thể nhìn thấy, sờ, nắn...

Thứ hai, về chất lượng của phần mềm, phần mềm có xu hướng *tốt hơn* qua thời gian, do các lỗi phần mềm trong quá trình sử dụng được phát hiện và khắc phục. Có thể nhìn thấy thông qua các hệ điều hành điện thoại sau khi chúng ta upgrade lên version mới. Nhiều người sẽ thắc mắc là, có cảm giác như sau khi nâng cấp xong thì còn nhiều lỗi hơn trước, có phải là tốt hơn đâu? Thực tế không phải vậy. Khi hệ điều hành được upgrade lên version mới thì nó sẽ được thêm vào nhiều các tính năng mới. Nói cách khác, đây có thể coi là các phần mềm "con" lần đầu tiên được ra mắt. Sau một thời gian sử dụng, các bạn sẽ tiếp tục được nhận những bản vá và dần dần những lỗi ban đầu cũng sẽ không còn nữa. Khác với phần cứng, ngay cả khi phần mềm không được vá lỗi thì các chức năng của phần mềm cũng không bị "mòn" đi như phần cứng. Nếu như bạn sử dụng một phần mềm mà bạn thấy dùng một thời gian thấy nó "chậm đi" hay có nhiều lỗi hơn, đó chính là vì ngay từ đầu phần mềm đó đã có lỗi như vậy, chứ không phải là phần mềm đó bị "xuống cấp".

Việc này liên quan đến đặc trưng tiếp theo của phần mềm, đó là phần mềm luôn *chứa các lỗi tiềm tàng*. Đó không phải là các lỗi mà ngay từ khi xuất xưởng người dùng và nhà phát triển có thể thấy được ngay, mà là các lỗi phải một thời gian sau khi sử dụng, làm việc với số lượng dữ liệu lớn hơn, đa dạng hơn của người dùng, thì các lỗi đó mới bộc lộ. Và phần mềm nào có quy mô càng lớn thì khả năng chứa lỗi càng cao. Một lý do nữa là do lỗi phần mềm thường dễ được phát hiện bởi người ngoài hơn là các nhà phát triển. Lý do đơn giản là vì nếu nhà phát triển đã nghĩ được đến tất cả các tình huống của người dùng thì đã khắc phục và ngăn chặn các lỗi đó ngay từ khi phát triển rồi. Thường thì phần mềm sẽ được xây dựng theo mạch phát triển của nhà phát triển, do đó khi người khác dùng, họ có thể không theo luồng thông thường, khi đó lỗi phần mềm mới bộc lộ.

Một đặc trưng nữa của phần mềm là chức năng của phần mềm thường có xu hướng *thay đổi*. Việc này có thể do nhiều lý do, ví dụ nhu cầu của thị trường thay đổi, hoặc

thay đổi để áp dụng tại nơi sử dụng mới, khách hàng đổi ý... Đây là một yếu tố hết sức quan trọng trong việc phát triển phần mềm, do đó, trong các nội dung tiếp theo, chúng ta sẽ có những bài học tìm hiểu kỹ hơn về vấn đề này.

1.1.3 Phân loại phần mềm

Thực ra, chúng ta không có một cách phân loại duy nhất nào về phần mềm. Ở góc độ của người dùng, chúng ta có thể phân loại theo mục đích sử dụng của phần mềm. Ví dụ

- Phần mềm **hệ thống** (System software)
- Phần mềm **thời gian thực** (Real-time software)
- Phần mềm **nghiệp vụ** (Business software)
- Phần mềm **khoa học & kỹ thuật** (Engineering & Science software)
- Phần mềm **nhúng** (Embedded software)
- Phần mềm **máy cá nhân** (Personal computer software)
- Phần mềm trên **web** (Web-based software)
- Phần mềm **trí tuệ nhân tạo** (Artificial Intelligent software)

Đối với các nhà phát triển, thường thì phần mềm sẽ được xếp vào 2 phân loại chính như sau:

- **Application** Software – Phần mềm **ứng dụng**
  - Dùng để xử lý nghiệp vụ thích hợp nào đó (quản lý, kế toán,…), phần mềm đóng gói, phần mềm của người dùng,…
- **System** Software – Phần mềm **hệ thống** (Lưu ý phân biệt với software system – hệ thống phần mềm)
  - Trao đổi với phần cứng máy tính và các thiết bị, quản lý các tài nguyên,…
  - *Ví dụ:* Hệ điều hành, driver, trình dịch…

1.2 Các khái niệm trong Công nghệ phần mềm

1.2.1 Định nghĩa

Trước hết, chúng ta sẽ cùng tìm hiểu về khái niệm của công nghệ phần mềm. Tiếng Anh của công nghệ phần mềm là Software Engineering. Ở đây, chữ Software có nghĩa

là "phần mềm", nhưng Engineering thì lại không phải hoàn toàn là "công nghệ", có thể dễ hiểu nhầm thành "technology". Engineering trong Software Engineering có nguồn gốc từ chữ Engineer, tiếng Việt là kỹ sư, do đó đúng ra thì phải dịch Engineering là Kỹ nghệ (từ này hơi hàn lâm và hơi lạ tai đối với một số người) hay Kỹ thuật (cũng không hoàn toàn chính xác vì có thể bị nhầm sang Technique). Chính vì lý do đó, đôi khi các bạn gặp từ Công nghệ phần mềm cũng được dịch là Kỹ nghệ phần mềm hay Kỹ thuật phần mềm. Tuy nhiên từ tiếng Việt phổ biến thì vẫn dùng là Công nghệ phần mềm, nhưng các bạn cũng cần phải hiểu rõ sự khác biệt của công nghệ phần mềm với công nghệ - technology thông thường.

Chúng ta sẽ tìm hiểu một số các định nghĩa chuẩn về công nghệ phần mềm để các bạn có thể hiểu rõ hơn sự khác biệt này:

- Bauer [1969] đã định nghĩa: "Công nghệ phần mềm (CNPM) là việc **thiết lập** và **sử dụng** các **nguyên tắc** kỹ nghệ tiêu chuẩn để có được phần mềm một cách **hiệu quả về kinh tế, đáng tin cậy** và hoạt động **hiệu quả** trên các máy thực"
- Ghezzi [1991] đưa ra định nghĩa của mình: "CNPM là một lĩnh vực của **khoa học máy tính**, liên quan đến xây dựng các **hệ thống phần mềm** vừa lớn vừa phức tạp bởi một hay nhiều nhóm kỹ sư"
- IEEE [1993] đưa ra định nghĩa *"CNPM là*

(1) việc áp dụng phương pháp tiếp cận có hệ thống, bài bản và được lượng hóa trong phát triển, vận hành và bảo trì phần mềm;

(2) **nghiên cứu** các phương pháp tiếp cận được dùng trong (1)"

- Còn theo Pressman [1995], định nghĩa là "CNPM là bộ môn tích hợp các **quy trình**, các **phương pháp**, các **công cụ** để phát triển phần mềm máy tính"
- Sommerville [1995] định nghĩa"CNPM là lĩnh vực liên quan đến **lý thuyết, phương pháp** và **công cụ** dùng cho phát triển phần mềm"
- Còn đối với K. Kawamura [1995], "CNPM là lĩnh vực học vấn về các **kỹ thuật, phương pháp luận** công nghệ học (lý luận và kỹ thuật được hiện thực hóa trên những nguyên tắc, nguyên lý nào đó) trong toàn bộ quy trình phát triển phần mềm nhằm nâng cao cả **chất** và **lượng** của sản xuất phần mềm"

Có thể thấy, với tất cả các định nghĩa, công nghệ phần mềm không phải chỉ là công nghệ để làm ra phần mềm như nhiều người hiểu nhầm. Mà công nghệ phần mềm là một lĩnh vực khoa học, về các phương pháp luận, kỹ thuật và công cụ tích hợp trong quy trình sản xuất và vận hành phần mềm nhằm tạo ra phần mềm với chất lượng mong muốn.

Như vậy, có thể thấy cách hiểu của không ít người, cho rằng "công nghệ phần mềm là lập trình" hay "công nghệ phần mềm là làm ra phần mềm" là không chính xác. Học công nghệ phần mềm là học những lý thuyết đằng sau quá trình "làm ra phần mềm", nhằm hướng đến việc phát triển phần mềm hiệu quả hơn, tạo ra các phần mềm chất lượng hơn.

Tất nhiên, thế nào là một phần mềm "chất lượng" cũng là một vấn đề cần phải phân tích rõ hơn nữa. Có thể hiểu một cách tổng quan là một phần mềm chất lượng là phần mềm ít lỗi, dễ bảo trì, trải nghiệm người dùng tốt và có tính tái sử dụng cao. Những bài học tiếp theo sẽ cung cấp những kiến thức chi tiết hơn về khái niệm "chất lượng phần mềm".

Như vậy, công nghệ phần mềm là một tập hợp các lý thuyết (không phải công cụ) hướng đến:

- Tăng năng suất và chất lượng phần mềm
- Quản lý **kế hoạch** hiệu quả
- Giảm **chi phí** phát triển phần mềm
- Đáp ứng **yêu cầu** và **nhu cầu** của khách hàng
- Tăng cường **quy trình** và **thực hành** kỹ nghệ phần mềm
- **Hỗ trợ** hiệu quả và có hệ thống các hoạt động của kỹ sư phát triển

1.2.2 Các thành phần trong Công nghệ phần mềm

Công nghệ phần mềm là công nghệ phân lớp, bao gồm 3 thành phần chính:

- Quy trình (**Process**)
- Phương pháp (**Method**)
- Công cụ (**Tool**)

Tất cả các thành phần này tập trung vào "chất lượng" của phần mềm.

Quy trình được sử dụng để g*ắn kết* các lớp, là n*ền tảng* cho kỹ thuật phần mềm, mục tiêu nhằm *đảm bảo thời gian* phát triển, t*ạo cơ sở* cho việc kiểm soát, quản lý dự án phần mềm. Quy trình cũng giúp thiết lập bối cảnh cho các phương pháp kỹ thuật được sử dụng, tạo sản phẩm, thiết lập các cột mốc, đảm bảo chất lượng và quản lý thay đổi

Phương pháp nhằm cung cấp các **kỹ thuật** cho xây dựng phần mềm. Phương pháp bao gồm các tác vụ: giao tiếp, phân tích yêu cầu, mô hình thiết kế, xây dựng chương trình, kiểm thử và hỗ trợ. Các phương pháp phải dựa trên các **nguyên tắc** cơ bản, bao gồm các hoạt động mô hình hóa.

Các công cụ ở đây có thể là tự động hoặc bán tự động, hỗ trợ cho quy trình và các phương pháp.

Các thành phần này đều hướng đến yếu tố "tập trung vào chất lượng" – quality focus. Yếu tố này là nền tảng cho CNPM. Chẳng hạn như: bất kỳ cách tiếp cận kỹ thuật nào đều phải dựa trên cam kết về chất lượng. Bản thân các quy trình cũng phải liên tục được cải tiến để phù hợp hơn và hiệu quả hơn....

1.2.3 Các giai đoạn trong Công nghệ phần mềm

Các giai đoạn, hay còn gọi là các "pha" (phase) trong công nghệ phần mềm bao gồm: (1) Pha định nghĩa; (2) Pha phát triển; và (3) Pha hỗ trợ.

a) Pha định nghĩa (definition)

Pha định nghĩa trả lời cho câu hỏi "What", cái gì. Các câu hỏi cần phải trả lời trong pha này bao gồm:

- Thông tin nào được xử lý,
- Chức năng và hiệu quả mong muốn,
- Hành vi mong đợi của hệ thống,
- Các giao điện cần thiết lập,
- Những ràng buộc về thiết kế,
- Và những tiêu chí cần thẩm định.

Đây cũng chính là các yêu cầu chính của hệ thống và phần mềm. Trả lời được các câu hỏi này, nhà phát triển sẽ xác định được các yêu cầu cho hệ thống, phần mềm cần phải phát triển.

b) Pha phát triển (development)

Pha phát triển trả lời cho câu hỏi "How" – như thế nào. Các yếu tố cần xác định trong pha này bao gồm:

- Cách thức dữ liệu được cấu trúc,
- Chức năng được triển khai trong kiến trúc phần mềm,
- Các chi tiết thủ tục được cài đặt,
- Cách xác định các đặc điểm của giao diện,
- Cách chuyển từ thiết kế sang lập trình,
- Và cách thức kiểm thử.

c) Pha hỗ trợ (support)

Pha hỗ trợ là giai đoạn sau khi phần mềm đã được triển khai, và tương ứng với các thay đổi (change) như là Sửa lỗi, Thích nghi với yêu cầu của bối cảnh, Các thay đổi bởi yêu cầu của khách hàng.

Có 4 loại thay đổi cần phải quan tâm:

- Sửa chữa (Correction),
- Thích ứng (Adaptation),
- Cải tiến (Enhancement),
- Phòng ngừa (Prevention).

1.3 Vai trò của Công nghệ phần mềm

1.3.1 Giá trị của phần mềm

Có lẽ các bạn cũng biết, giá trị phần mềm tại thời điểm hiện tại là rất cao. (Không phải tự nhiên mà các doanh nghiệp thuê hàng chục, hàng trăm nhân viên để làm phần mềm, mỗi nhân viên lương hàng chục triệu mỗi tháng) Nhưng giá trị phần mềm không phải lúc nào cũng vậy. Vào thời buổi máy tính mới ra đời, vào khoảng những năm 1940, vai trò của phần mềm rất thấp. (Như chúng ta đã biết ở bài trước) Ban đầu phần cứng có vai trò rất quan trọng, còn phần mềm chỉ có vai trò đưa ra các chỉ lệnh để máy tính thực hiện. Khi đó, giá thành của máy tính (phần cứng) rất cao, chi phí sản xuất máy tính cũng rất lớn. Vào khoảng những năm 1940, Thomas J Watson, chủ tịch của IBM từng

nói một câu nói nổi tiếng: "Tôi nghĩ là thị trường thế giới sẽ cần khoảng 5 cái máy tính".

Rõ ràng điều này giờ đã không còn hợp lý, khi mà mỗi một người đều có ít nhất một chiếc máy tính trên tay, nếu như chúng ta sở hữu một chiếc điện thoại thông minh như Android hay iPhone. Những năm sau đó, sự phát triển của phần mềm đã khiến cho vai trò của phần mềm ngày càng tăng lên. Từ những năm 50, với sự xuất hiện của Fortran, Assembly, việc lập trình bằng "đục lỗ" đã dần biến mất do các phần mềm đã có thể dễ dàng được tạo ra hơn và linh hoạt hơn. Những năm 60 đánh dấu sự ra đời của hệ điều hành, tách riêng các điều khiển phần cứng trực tiếp. Lúc này, phần mềm có thể dễ dàng được tạo ra mà không cần phải quan tâm đến từng loại phần cứng khác nhau, tạo ra một bước phát triển mạnh mẽ trong việc sản xuất phần mềm vào những năm 70. Những năm 80 đánh dấu sự ra đời của các PC – Personal Computer, máy tính cá nhân, mang máy tính đến cho mọi gia đình, và cũng là thời gian mạng máy tính bắt đầu được phát triển. Từ những năm 2000 trở lại đây, chúng ta chứng kiến sự phát triển mạnh mẽ và đa dạng của các loại phần mềm khác nhau, từ VR đến điện toán toàn cầu, xe tự lái,... kèm theo đó là các vấn đề mới về bảo mật. Trong khoảng vài năm trở lại đây, chúng ta lại thấy những bước tiến vượt bậc của trí tuệ nhân tạo – AI, và có lẽ sẽ vẫn là từ khóa "hot" trong nhiều năm tới.

Nhìn vào lịch sử phát triển của phần mềm, chúng ta có thể thấy được vai trò ngày càng tăng của phần mềm. Cùng với một số lượng người sử dụng ngày càng lớn (bất kỳ ai cũng có thể là một người sử dụng phần mềm), giá trị của phần mềm theo đó cũng ngày càng cao. Ngược lại, giá thành của phần cứng ngày càng giảm. Không phải là phần cứng không có sự phát triển gì, mà ngược lại, chính những sự phát triển mạnh mẽ của phần cứng khiến cho giá thành của phần cứng càng dễ tiếp cận hơn, qua đó cũng quay lại thúc đẩy sự phát triển của phần mềm.

Vậy điều gì khiến cho phần mềm đắt đỏ như vậy? Để trả lời cho câu hỏi này, chúng ta nhìn vào phần mềm dưới góc độ kinh tế, của bài toán cung và cầu.

Từ bên cầu, tức là từ góc độ của khách hàng: Nếu xét theo bài toán kinh tế, giả sử khách hàng mua một phần mềm với giá $6,000, nhưng khi khách hàng sử dụng phần mềm đó cho công việc và tiết kiệm được $10,000, thì bản chất là khách hàng đã tiết kiệm được

một khoản tiền không nhỏ là $4,000. Ngoài ra, còn có một số trường hợp việc sử dụng phần mềm là không thể thay thế được, khi đó mọi con số về giá tiền đều là vô nghĩa. Ví dụ như các phần mềm quản lý cho ngân hàng, các phần mềm điều khiển không lưu... Còn nhìn từ bên cung, tức là từ góc độ của nhà phát triển, thì cũng phải thừa nhận một điều là sản xuất ra phần mềm rất đắt đỏ. (Vừa rồi mình có nhắc đến) một ví dụ là chi phí nhân công. Giả sử bạn có một nhóm phát triển phần mềm nhỏ, với 20 nhân viên, và mức lương trả cho nhân viên là 15 triệu / tháng. Khi đấy chi phí nhân công 1 tháng đã là 300 triệu, 1 năm là 3 tỷ 6. Đó là còn chưa kể đến việc thanh toán các chi phí điện, nước, thuê mặt bằng v.v.. Một yếu tố nữa cũng có thể rất tốn kém là các phần mềm để phát triển phần mềm. Ngoài ra còn các khâu khảo sát, lấy ý kiến, quảng cáo, marketing... cũng tốn một khoản tiền không nhỏ.

Nhưng có lẽ một trong các chi phí tốn kém nhất là chi phí hỗ trợ. Nếu các bạn đã từng sử dụng những phần mềm nguồn mở miễn phí và thấy các tính năng của nó không khác gì phần mềm thương mại, thì đó là do các phần mềm miễn phí chỉ được cộng đồng hỗ trợ, khác với các phần mềm thương mại. Đối với các phần mềm thương mại, thực chất là bạn đang trả tiền cho các chi phí hỗ trợ kỹ thuật.

1.3.2 Vai trò của Công nghề phần mềm

Việc áp dụng các kỹ thuật phần mềm cho phép

- Cung cấp một quy trình **dễ hiểu** để phát triển hệ thống. Việc này giúp giảm thiểu các rủi ro có thể phát sinh khi phát triển phần mềm.
- Phát triển các hệ thống và phần mềm có thể **bảo trì** và dễ dàng **thay đổi.** Thứ nhất, phần mềm dễ bảo trì có thể giảm thiểu đáng kể chi phí cho việc hỗ trợ, là một chi phí rất lớn trong phát triển phần mềm. Ngoài ra, việc dễ dàng thay đổi cũng giúp việc phát triển phần mềm linh hoạt hơn, bảo đảm cho việc phần mềm có thể đáp ứng được các yêu cầu.
- Phát triển hệ thống phần mềm **chắc chắn, ổn định.** Một phần mềm chắc chắn, ổn định sẽ ít phát sinh lỗi, khiến khách hàng hay người sử dụng hài lòng hơn với việc sử dụng sản phẩm.

- Cho phép quy trình tạo các hệ thống phần mềm có thể **lặp lại** và **quản lý được**. Việc này nâng cao tính tái sử dụng trong phát triển phần mềm. Chẳng hạn, khi phát triển một dự án mới, nếu nhà phát triển có thể tái sử dụng được các thành phần đã phát triển thì việc đó có thể tiết kiệm được một khoản không hề nhỏ chi phí phát triển phần mềm, đồng thời cũng có thể rút ngắn thời gian phát triển (thực chất cũng là tiền)

Có thể thấy, công nghệ phần mềm có vai trò to lớn trong việc phát triển một phần mềm chất lượng hơn, giúp khách hàng hài lòng với sản phẩm hơn, đồng thời tiết kiệm được chi phí phát triển sản phẩm, rút ngắn thời gian phát triển phần mềm.

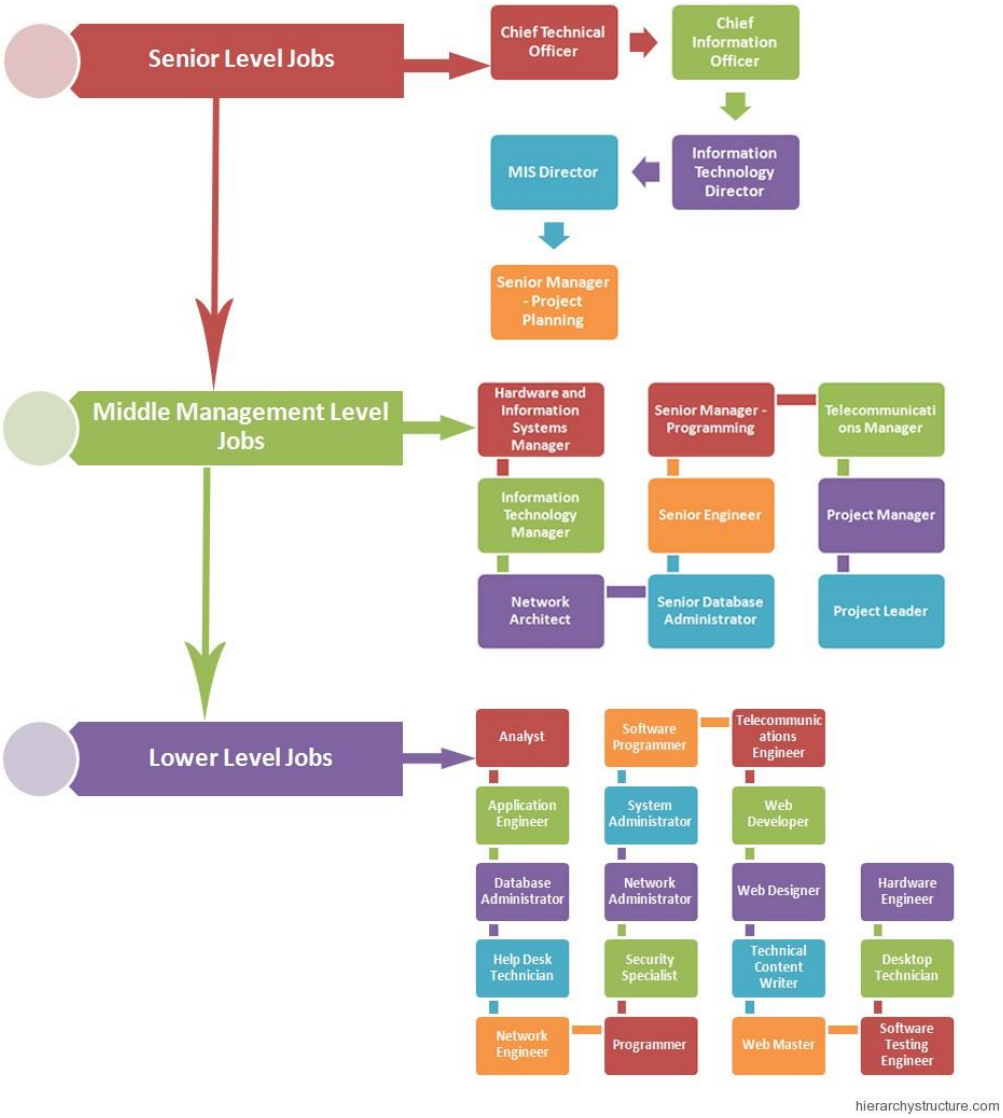
1.3.3 Thị trường của phần mềm

Có thể các bạn đã được nghe nói đến các công việc có liên quan đến phát triển phần mềm, như lập trình viên (programmer), nhà phát triển (developer) hay kỹ sư phần mềm (software engineer), nhưng chưa chắc đã hiểu được sự khác biệt giữa các công việc này.

Để dễ hình dung, có thể coi đây là các mức độ của các công việc liên quan đến phát triển phần mềm. Nếu như bạn chỉ có khả năng viết mã nguồn đơn giản để viết ra một chương trình, các bạn sẽ được coi là newbie. Để có thể được coi là một lập trình viên, các bạn cần phải có khả năng cài đặt các giải thuật để giải quyết một số vấn đề đơn giản. Nhà phát triển phần mềm, developer, là ở một mức cao hơn chút nữa, không chỉ có khả năng cài đặt các giải thuật, mà phải có khả năng tạo ra các ứng dụng có thể sử dụng được (giao diện thuận tiện, không chỉ một mà nhiều người sử dụng) và có thể thu được lợi nhuận từ việc bán phần mềm. Còn nếu mà muốn trở thành kỹ sự phần mềm, software engineer, bạn phải có kỹ năng thiết kế giải pháp, xây dựng các hệ thống và viết mã nguồn để xây dựng các ứng dụng có khả năng mở rộng. Như vậy có thể thấy, một kỹ sư phần mềm là một vị trí công việc cao, bên cạnh việc có thể lập trình hay phát triển sản phẩm được thì cũng phải nắm vững và áp dụng tốt các kỹ thuật công nghệ phần mềm.

Chúng ta có thể nhìn vào Hình 1.1. Bảng phân công các vị trí công việc trong thị trường phần mềm. Có thể thấy, các vị trí bậc trung và bậc cao (thường sẽ có mức lương cao

hơn) đều đòi hỏi các kỹ năng của công nghệ phần mềm. Tất nhiên điều này không có nghĩa là các công việc "lower level job" luôn luôn có mức lương thấp. Nếu các bạn là một lập trình viên giỏi (cộng với một chút may mắn) thì các bạn cũng có thể có một vị trí công việc có mức lương cao. Tuy nhiên, việc nắm được các khái niệm của công nghệ phần mềm (như môn học này đang hướng đến, nhập môn công nghệ phần mềm) chắc chắn sẽ giúp các bạn có cái nhìn bao quát hơn và giúp đỡ rất nhiều cho công việc của bạn. Còn nếu các bạn muốn nhắm đến các công việc bậc cao hơn thì việc nắm vững và biết cách áp dụng các kiến thức liên quan đến công nghệ phần mềm là điều tối thiểu. Ví dụ: Quản trị dự án phần mềm, Phân tích và thiết kế hệ thống, Quản lý chất lượng sản phẩm, Giao diện và trải nghiệm người dùng



Hình 1-1: Các mức độ công việc ngành Công nghệ thông tin

(nguồn: Computer Jobs Hierarchy (2017) Hierarchy Structure. https://hierarchystructure.com/computer-jobs-hierarchy/)

1.4 Các vấn đề trong Công nghệ phần mềm

1.4.1 Xác định đối tượng của phần mềm

Trả lời cho câu hỏi "phần mềm được xây dựng là dành cho những đối tượng nào"? Câu trả lời ở đây là có 3 đối tượng cần phải quan tâm khi phát triển phần mềm:

- Khách hàng (client),
- Người mua (customer), và
- Người sử dụng (user)

Đối tượng đầu tiên là *khách hàng*, client.

Khách hàng hay client có thể là một người (hoặc cũng có thể là một nhóm người, một tổ chức) mà nhóm phát triển phần mềm tạo ra phần mềm cho họ. Khách hàng sẽ đến gặp nhóm phát triển, đưa ra các nhu cầu của mình, mong đợi nhận lại một phần mềm như thế nào. Để đổi lại, khách hàng sẽ cung cấp cho nhóm phát triển tài nguyên để thực hiện công việc phát triển phần mềm (thông thường là một khoản tiền, nhưng đôi khi có thể là nhân lực hay dữ liệu…)

Trong rất nhiều trường hợp, thành công của dự án phần mềm mang ý nghĩa sống còn đối với thành công trong công việc, ví dụ công việc kinh doanh của khách hàng. Lấy ví dụ như một hãng quần áo chuẩn bị ra mắt một chi nhánh mới, cần phải nhanh chóng phát triển hệ thống quản lý logistics để có thể đưa chi nhánh vào hoạt động. Do đó, nếu hệ thống quản lý bị ra mắt chậm hơn so với tiến độ hoàn thành của chi nhánh thì doanh nghiệp sẽ phải gồng một khoản lỗ lớn cho các chi phí cơ sở hạ tầng và nhiều thiệt hại khác.

Đối tượng thứ hai cần quan tâm đến là customer, tiếng Việt đôi khi cũng dịch là khách hàng. Để phân biệt, trong nội dung của môn học này chúng ta sẽ gọi là *người mua*, là người bỏ tiền ra mua phần mềm hoặc lựa chọn nó để cơ quan/doanh nghiệp sử dụng.

Khái niệm này cũng gần giống với khái niệm của khách hàng – client. Điểm khác biệt ở đây là client thường có yếu tố pháp luật. Khi client "đặt hàng" nhà phát triển, thì hai bên sẽ ký hợp đồng và có trách nhiệm thực hiện theo hợp đồng đó. Trong khi đó, customer chỉ đóng vai trò giống như người đi mua hàng: thấy một phần mềm thỏa mãn với các yêu cầu của mình thì sẽ bỏ tiền ra để mua về dùng. Mặc dù vậy, về mặt hành

chính, đôi khi client cũng có thể được coi là một loại customer, do có liên quan đến thao tác "trả tiền"

Đối tượng thứ 3 của phần mềm là *người sử dụng* (user), đôi khi còn được gọi là người dùng cuối (end-user), có thể hiểu đơn giản là người thực sự sử dụng phần mềm.

Với phần mềm cá nhân, người dùng và người mua có thể giống nhau. Ví dụ, nếu bạn lên Google Play hay Apple AppStore để mua một app về sử dụng thì bạn vừa là người mua, vừa là người sử dụng. Tuy nhiên, đối với các tổ chức, người mua và người dùng cuối thường khác nhau. Lấy ví dụ, một trường đại học liên hệ với Microsoft để mua phần mềm Microsoft 365 để sử dụng thì trường đại học đó (hoặc đại diện của trường đại học đó) là người mua, nhưng người dùng cuối lại là các sinh viên, giảng viên của nhà trường.

Vậy khi phát triển một phần mềm, ai sẽ là người mà nhà phát triển phải hướng đến? Có thể hầu hết mọi người sẽ nghĩ ngay đến người dùng cuối, vì đó là những người trực tiếp sử dụng phần mềm, tuy nhiên nếu nhà phát triển chỉ tối ưu cho người dùng cuối, thì sẽ có khả năng các yêu cầu của người mua hay khách hàng không được đảm bảo. Trong rất nhiều trường hợp, khách hàng có những lý do về mặt kinh doanh hoặc về mặt tổ chức mà khó có thể thỏa mãn được các nhu cầu của người dùng cuối.

Tóm lại ở đây là thước đo thành công quan trọng nhất của dự án phần mềm chính là sự hài lòng của khách hàng. Việc làm khách hàng hài lòng ở đây không đồng nghĩa với việc nhất nhất thỏa mãn theo tất cả các yêu cầu của khách hàng, mà thay vào đó là tư vấn hoặc trao đổi với khách hàng nhằm đưa ra giải pháp tối ưu nhất, tìm ra một điểm cân bằng. Rõ ràng là nếu một sản phẩm chỉ làm theo yêu cầu của khách hàng, nhưng dẫn đến việc người dùng không thỏa mãn với sản phẩm thì khách hàng cũng khó mà có thể hài lòng được.

1.4.2 Cân bằng giữa tính năng, chi phí và thời gian

Việc phát triển một dự án phần mềm phải hướng đến:

- Sản phẩm hoạt động như mong đợi (**tính năng**)
- Dưới ngân sách (**chi phí**)
- Giao hàng đúng hoặc trước hạn (**thời gian**)

Tuy vậy, trong thực tế, mọi chức năng bổ sung sẽ tăng thêm chi phí cho việc phát triển, thử nghiệm, bảo trì, v.v. Chính vì vậy, việc tối đa cả 3 tiêu chí tính năng, chi phí và thời gian thường khó có thể đạt được. Ví dụ, nếu muốn một sản phẩm chất lượng, bắt buộc các chi phí cho nhân công hoặc outsource phải tăng lên, hoặc là phải hoãn thời gian giao hàng để chăm chút cho các tính năng của sản phẩm. Hoặc muốn sản phẩm được phát triển với chi phí thấp nhất, thì các tính năng phải chấp nhận kém đi một chút, chấp nhận có thể nhiều lỗi hơn do cắt giảm việc kiểm thử và sửa lỗi, hoặc thời gian giao hàng phải chậm hơn do giới hạn về khả năng của nhân công…

Vì vậy, đối với các dự án thực tế, việc nhóm phát triển cần phải làm là tìm ra một điểm cân bằng giữa các tiêu chí trên. Đối với công nghệ phần mềm, đây là một vấn đề cần phải giải quyết, ví dụ thông qua việc trao đổi với khách hàng để lấy yêu cầu, quản lý các tài nguyên một cách hợp lý…

1.4.3 Rủi ro của dự án

Có rất nhiều rủi ro có thể xảy đến với dự án. Ví dụ như phần lớn chức năng của phần mềm bị đã dành nhiều thời gian và công sức để phát triển nhưng cuối cùng không bao giờ được sử dụng. Việc này khiến cho việc sử dụng các tài nguyên của dự án là rất lãng phí. Trong rất nhiều dự án phần mềm đã bị thất bại, lý do là vì người phát triển phần mềm đã xây dựng sai phần mềm, có thể do hiểu sai ý của khách hàng…

Chính vì vậy, nhóm phát triển phần mềm phải:

- Hiểu những gì khách hàng mong đợi ở phần mềm
- Hiểu những gì tổ chức của khách hàng mong đợi ở khách hàng
- Hiểu những gì khách hàng và người dùng mong đợi ở phần mềm

Một số kỹ thuật trong công nghệ phần mềm hướng đến việc giảm thiểu rủi ro như:

- Nghiên cứu tiền khả thi (có nên bắt đầu dự án không?)
- Tách biệt các yêu cầu (là những gì khách hàng muốn) ra khỏi thiết kế (là cách nhà phát triển đáp ứng các yêu cầu)
- Các mốc quan trọng - milestone (cách nhà phát triển báo cáo hoặc chứng minh tiến độ cho khách hàng) và phát hành – release

- Kiểm thử của người dùng và kiểm thử chấp nhận (cách khách hàng kiểm tra xem phần mềm có đáp ứng yêu cầu không)
- Bàn giao (đảm bảo khách hàng nhận được gói sản phẩm có thể được vận hành và hỗ trợ trong thời gian dàiGiảm thiểu rủi ro
- Nghiên cứu tiền khả thi (có nên bắt đầu dự án không?)
- Tách biệt các yêu cầu (những gì khách hàng muốn) ra khỏi thiết kế
- (cách nhà phát triển đáp ứng các yêu cầu)
- Các mốc quan trọng - milestone (cách nhà phát triển báo cáo hoặc
- chứng minh tiến độ cho khách hàng) và phát hành – release)
- Kiểm thử của người dùng và kiểm thử chấp nhận (cách khách hàng kiểm tra xem phần mềm có đáp ứng yêu cầu không)
- Bàn giao (đảm bảo khách hàng nhận được gói sản phẩm có thể được vận hành và hỗ trợ trong thời gian dài)

Tuy vậy, việc thực hiện các kỹ thuật này thường không dễ dàng. Lý do là vì người phát triển phần mềm thường không giỏi đánh giá tiến độ. Ngoài ra, người phát triển cũng thường quá lạc quan về sự tiến triển của dự án. Bên cạnh đó, các nhà phát triển cũng thường ngại báo cáo, cho rằng việc báo cáo là lãng phí thời gian.

1.4.4 Một số vấn đề khác

Một số các vấn đề khác có thể kể đến như sau:

- Không đáp ứng được các nhu cầu kinh doanh
- Không thỏa mãn các yêu cầu
- Không tích hợp các mô-đun
- Khó khăn khi bảo trì
- Phát hiện muộn các sai sót
- Chất lượng trải nghiệm kém
- Hiệu suất phát triển phần mềm kém
- Không có nỗ lực phối hợp của nhóm
- Các vấn đề về xây dựng và phát hành sản phẩm

- Không có phương pháp **mô tả rõ ràng yêu cầu** của khách hàng, dẫn đến việc dễ phát sinh vấn đề sau khi bàn giao
- Với những phần mềm quy mô lớn, tư liệu đặc tả **cứng nhắc**, khó đáp ứng nhu cầu thay đổi của người dùng
- Phương pháp luận thiết kế **không nhất quán.** Mỗi một thành phần của dự án được thiết kế theo cách riêng, dẫn đến giảm chất lượng phần mềm
- Không có chuẩn về việc tạo **tài liệu quy trình** sản xuất phần mềm. Những đặc tả không rõ ràng cũng dẫn đến việc giảm chất lượng phần mềm
- Không kiểm thử **tính đúng đắn** của phần mềm ở từng giai đoạn mà chỉ kiểm thử ở giai đoạn cuối
- Không đề cao quá trình thiết kế
- Coi thường việc tái sử dụng phần mềm
- Phần lớn các thao tác trong quy trình phát triển phần mềm do con người thực hiện. Việc này có thể khiến năng suất lao động giảm
- Không chứng minh được **tính đúng đắn** của phần mềm, làm giảm độ tin cậy của phần mềm
- Chuẩn về một phần mềm tốt không thể đo được một cách **định lượng, k**hông thể đánh giá được một hệ thống đúng đắn hay không
- Đầu tư số lượng lớn nhân lực vào bảo trì, giảm hiệu suất lao động của nhân viên
- Công việc bảo trì **kéo dài**, làm giảm chất lượng của tư liệu và ảnh hưởng xấu đến những việc khác
- Quản lý dự án **lỏng lẻo.** Quản lý lịch trình sản xuất phần mềm không rõ ràng
- Không có tiêu chuẩn để ước lượng **nhân lực** và **dự toán**, làm kéo dài thời hạn và vượt kinh phí của dự án
- …

Chương 2 VÒNG ĐỜI PHẦN MỀM

Nội dung:

- Tổng quan vòng đời phần mềm
- Tổng quan về quy trình phát triển phần mềm
- Một số mô hình phát triển phần mềm phổ biến
- Ví dụ và bài tập

2.1 Tổng quan vòng đời phần mềm

2.1.1 Một số khái niệm

a. Vòng đời phần mềm

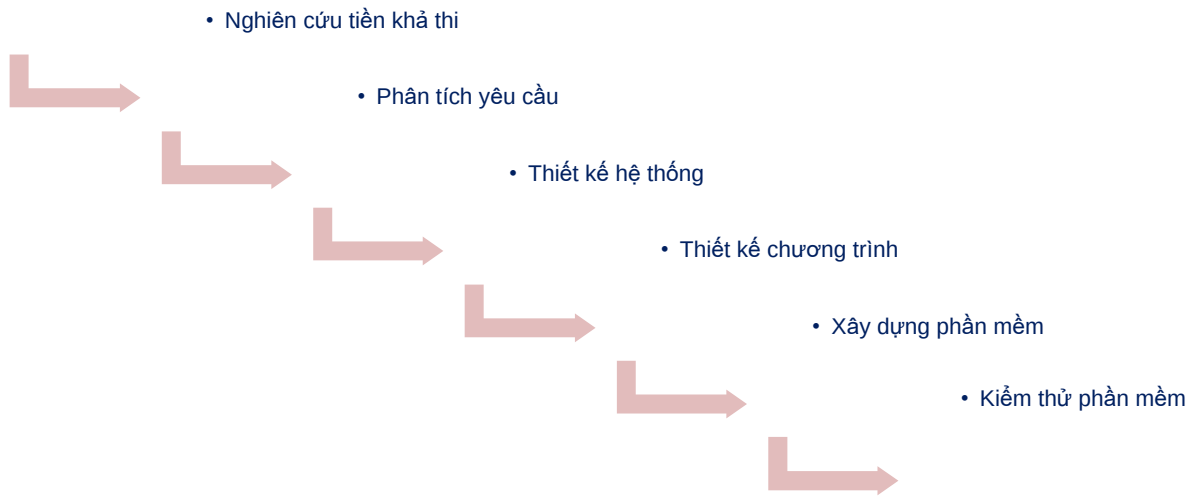
- Tên tiếng Anh: Software life cycle
- Vòng đời phần mềm là thời kỳ tính từ khi phần mềm được sinh (tạo) ra cho đến khi chết đi (từ lúc hình thành đáp ứng yêu cầu, vận hành, bảo dưỡng cho đến khi loại bỏ không đâu dùng)
- Quy trình phần mềm (vòng đời phần mềm) được phân chia thành các pha chính: phân tích, thiết kế, chế tạo, kiểm thử, bảo trì. Biểu diễn các pha có thể khác nhau theo từng mô hình
- Mọi sản phẩm phần mềm đều có vòng đời.
- Vòng đời thường khá dài - một số sản phẩm phần mềm đã "tồn tại" được 30 năm.
- Vòng đời có thể được rút ngắn do tiến bộ công nghệ
- Trong thực tế, khái niệm về vòng đời phần mềm hiện nay ít được nhắc tới. Trong công nghệ phần mềm, khái niệm về vòng đời phát triển phần mềm được nhắc tới nhiều hơn

b. Vòng đời phát triển phần mềm

- Tên tiếng Anh: Software Development Life Cycle - SDLC
- Vòng đời phát triển phần mềm (SDLC) là quy trình về chi phí và thời gian mà các nhóm phát triển sử dụng để thiết kế và xây dựng phần mềm.

- Mục tiêu của SDLC là giảm thiểu rủi ro dự án thông qua việc lập kế hoạch trước để phần mềm đáp ứng mong đợi của khách hàng trong giai đoạn sản xuất và hơn thế nữa.
- Phương pháp này vạch ra một loạt các bước chia quy trình phát triển phần mềm thành các nhiệm vụ mà có thể chỉ định, hoàn thành và đo lường.

2.1.2 Các giai đoạn trong quy trình phát triển phần mềm



Hình 2-1: Ví dụ các bước trong mô hình truyền thống – thác nước

Vòng đời phát triển phần mềm (SDLC) nêu một số bước, nhiệm vụ cần thiết để xây dựng một ứng dụng phần mềm. Quá trình phát triển trải qua nhiều giai đoạn khi nhóm phát triển thêm các tính năng mới và sửa lỗi trong phần mềm. Thông tin chi tiết về quy trình SDLC thay đổi tùy theo nhóm, tùy theo mô hình phát triển phần mềm mà nhóm phát triển áp dụng. Tuy nhiên, nhìn chung đều bao gồm 5 bước chính: Phân tích yêu cầu, thiết kế, phát triển hay viết phần mềm, kiểm thử phần mềm, bảo trì.

- **Yêu cầu**
  - Xác định nhu cầu của khách hàng và các ràng buộc của sản phẩm.
  - Thường bao gồm các nhiệm vụ như phân tích lợi ích chi phí, lập lịch trình, ước tính và phân bổ tài nguyên. Nhóm phát triển thu thập các yêu cầu từ một số bên liên quan như khách hàng, các chuyên gia nội bộ và bên ngoài cũng như các nhà quản lý để tạo ra một tài liệu về thông số kỹ thuật yêu cầu của phần mềm.

- **Thiết kế:** Các kỹ sư phần mềm phân tích các yêu cầu và xác định các giải pháp thích hợp nhất để tạo ra phần mềm. Ví dụ: có thể xem xét việc tích hợp các mô-đun có sẵn, lựa chọn công nghệ và xác định các công cụ phát triển, xem xét cách tích hợp tốt nhất phần mềm mới vào cơ sở hạ tầng CNTT
- **Phát triển - mã hóa**: Trong giai đoạn triển khai, nhóm phát triển mã hóa sản phẩm. Họ phân tích các yêu cầu để xác định các nhiệm vụ viết mã nhỏ hơn mà họ có thể thực hiện hàng ngày để đạt được kết quả cuối cùng.
    - Khi các nhóm phát triển phần mềm, viết mã và thử nghiệm thường trên một bản sao khác của phần mềm chứ không phải bản mà người dùng truy cập. Phần mềm mà khách hàng dùng được gọi là phiên bản chính thức, trong khi các bản sao khác được gọi là môi trường xây dựng, hay môi trường kiểm thử.
    - Việc có các môi trường xây dựng và xuất bản phần mềm riêng giúp đảm bảo khách hàng có thể tiếp tục sử dụng phần mềm ngay cả khi phần mềm đó đang được thay đổi hoặc nâng cấp.
- **Kiểm tra - kiểm thử:** Nhóm phát triển kết hợp quy trình kiểm thử tự động và thủ công để kiểm tra phần mềm xem có lỗi không. Phân tích chất lượng bao gồm việc kiểm tra phần mềm xem có lỗi không và kiểm tra xem phần mềm có đáp ứng các yêu cầu của khách hàng hay không. Giai đoạn thử nghiệm thường chạy song song với giai đoạn phát triển.
- **Bảo trì:** Trong giai đoạn bảo trì, trong số các nhiệm vụ khác, nhóm sửa lỗi, giải quyết các vấn đề của khách hàng và quản lý các thay đổi về phần mềm. Ngoài ra, nhóm giám sát trải nghiệm người dùng, bảo mật và hiệu suất hệ thống chung để xác định các cách mới nhằm cải thiện phần mềm hiện có.

2.1.3 Một số lưu ý

a. Vấn đề bảo mật trong quy trình phát triển phần mềm

Trong quy trình phát triển phần mềm thông thường, kiểm tra bảo mật là một quá trình riêng biệt với vòng đời phát triển phần mềm (SDLC). Nhóm bảo mật phát hiện

ra các lỗ hổng bảo mật chỉ sau khi họ xây dựng phần mềm. Điều này dẫn đến số lượng lỗi lớn vẫn chưa được phát hiện cũng như tăng rủi ro bảo mật.

Hiện nay, bảo mật là một phần không thể thiếu trong vòng đời phát triển của phần mềm. Vấn đề này có thể giải quyết vấn đề bảo mật trong SDLC theo các phương pháp DevSecOps và tiến hành đánh giá bảo mật trong toàn bộ quy trình SDLC

- DevSecOps là viết tắt của development (phát triển), security (bảo mật) và operations (vận hành): xác định vai trò và trách nhiệm khác nhau của các đội ngũ phần mềm khi xây dựng ứng dụng phần mềm.
- DevSecOps là phương pháp tích hợp thử nghiệm bảo mật ở mọi giai đoạn của quá trình phát triển phần mềm. Phương pháp này bao gồm các công cụ và quy trình khuyến khích cộng tác giữa các nhà phát triển, chuyên gia bảo mật và đội ngũ vận hành nhằm xây dựng phần mềm có thể chống lại các mối đe dọa hiện đại.

Ngoài ra, cần lưu ý rằng các hoạt động bảo đảm bảo mật như đánh giá mã lập trình, phân tích kiến trúc và kiểm thử thâm nhập là phần không thể thiếu đối với việc phát triển sản phẩm phần mềm.

b. Phân biệt một số thuật ngữ

➢ **Vòng đời phát triển của hệ thống**

- Hệ thống thường bao gồm một số thành phần phần cứng và phần mềm hoạt động cùng nhau để thực hiện các chức năng phức tạp
- Vòng đời phát triển của phần mềm chỉ xử lý các thành phần phần mềm. Mặt khác, quá trình phát triển hệ thống là một tập hợp rộng hơn liên quan đến việc thiết lập và quản lý phần mềm, phần cứng, con người cũng như các quy trình có thể tạo nên một hệ thống. Quá trình này có thể bao gồm các nhiệm vụ như đào tạo tổ chức và các chính sách quản lý thay đổi không thuộc phạm vi phát triển phần mềm.

➢ **Quản lý vòng đời ứng dụng**

- Quản lý vòng đời ứng dụng (ALM - Application LifeCycle Management) là việc tạo và bảo trì các ứng dụng phần mềm cho đến khi không còn cần sử dụng các ứng dụng đó nữa. Việc này liên quan đến nhiều quy trình, công cụ và nhiều

người làm việc cùng nhau để quản lý mọi khía cạnh của vòng đời, chẳng hạn như ý tưởng, thiết kế và phát triển, kiểm thử, sản xuất, hỗ trợ và dự phòng cuối cùng.

- SDLC mô tả chi tiết hơn về giai đoạn phát triển ứng dụng. Đó là một phần của ALM. ALM bao gồm toàn bộ vòng đời của ứng dụng và không chỉ dừng lại ở SDLC. ALM có thể có nhiều SDLC trong vòng đời của một ứng dụng.

2.2 Tổng quan về quy trình phát triển phần mềm

2.2.1 Đặc thù của các dự án phần mềm

Một số yếu tố liên quan đến các dự án phần mềm:

- **Tình trạng dự án phần mềm không đạt**
  - Phần mềm không hoạt động như dự kiến (tính năng – function)
  - Vượt quá dự toán (chi phí – cost)
  - Thời gian bàn giao phần mềm trễ (thời gian, tiến độ – time)
- **Khó khăn trong quyết định**
  - Mỗi dự án phần mềm đều phải tính toán cân đối giữa 3 yếu tố : tính năng, chi phí, thời gian.
  - Các tính năng bổ sung, thêm mới sẽ làm tăng chi phí cho việc phát triển, thử nghiệm, bảo trì….

Theo thống kê cho thấy: 66% dự án công nghệ (dựa trên phân tích của 50.000 dự án trên toàn cầu) đều thất bại một phần hoặc toàn bộ (*CHAOS 2020, Standish Group*); 31% dự án CNTT của Hoa Kỳ đã bị hủy bỏ hoàn toàn và hiệu suất của 53% dự án đó "đáng lo ngại" (*CHAOS 2020, Standish Group*); 17% dự án CNTT lớn trở nên tồi tệ đến mức đe dọa đến sự tồn tại của công ty (*Theo nghiên cứu của McKinsey, 2020*)

Đi vào phân tích một số nguyên nhân thất bại, chúng ta có thể nhận ra: các dự án phần mềm có tỷ lệ thất bại cao so với nhiều loại dự án khác. Lý do đầu tiên có thể kể tới là việc phát triển phần mềm có tính trừu tượng – không thể thấy hoặc chạm tới được. Sự thất bại của dự án, có thể chia ra thành nguyên nhân bên ngoài và bên trong của dự án

- Nguyên nhân bên trong: kế hoạch không tốt, định nghĩa yêu cầu không tốt, cơ cấu tổ chức của dự án không đạt yêu cầu….

- Nguyên nhân bên ngoài: vấn đề về chất lượng và chức năng, vấn đề về chi phí, vấn đề về tiến độ.

Trong quá trình thực hiện dự án phần mềm, nhóm phát triển phải đồng thời thỏa mãn các tiêu chí khó khăn:

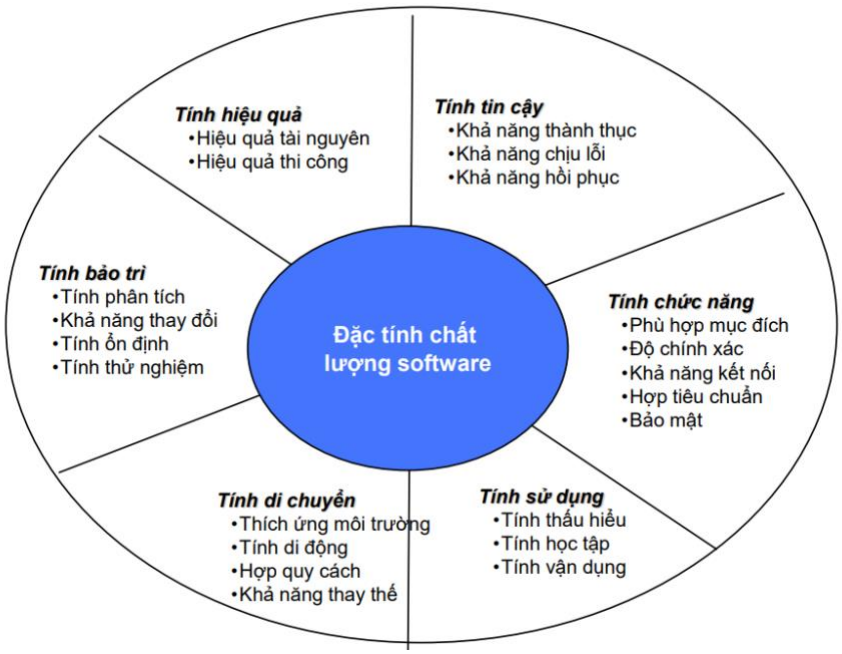
- Hiểu những gì khách hàng mong đợi ở phần mềm
- Hiểu những gì tổ chức của khách hàng mong đợi ở khách hàng
- Hiểu những gì khách hàng và người dùng mong đợi ở phần mềm

2.2.2 Quy trình phần mềm trong thực tế

a. Phân tích đặc tính của chất lượng phần mềm

Liên quan tới chất lượng phần mềm, nhìn chung mục tiêu của quy trình phần mềm là đảm bảo rằng sản phẩm cuối cùng đáp ứng các yêu cầu của khách hàng và đạt được các tiêu chuẩn chất lượng mong muốn. Như vậy, có thể hiểu quy trình phần mềm

- Quy trình tốt dẫn tới phần mềm tốt
- Quy trình tốt giảm thiểu rủi ro
- Quy trình tốt làm rõ ràng, dự án dễ quản lý
- Quy trình tốt cho phép làm việc nhóm



Hình 2-2: Đặc tính chất lượng theo ISO/IEC JT1/SC2 9126, JIS X0129 - 1994

b. Kiểm soát chất lượng phần mềm trong thực tế

Khi tiến hành phân tích các bước trong quy trình phát triển phân mềm cho thấy. Các bước thường bao gồm:

(1) Xác thực các yêu cầu.

(2) Xác nhận hệ thống và thiết kế chương trình.

(3) Kiểm tra khả năng sử dụng.

(4) Kiểm tra chương trình.

(5) Kiểm tra chấp nhận.

(6) Sửa lỗi và bảo trì.

Trong đó các bước (2),(3),(4),(5) ở trên có thể lặp lại nhiều lần tương ứng với các vòng lặp trong chu kỳ phát triển phần mềm. Việc kiểm soát chất lượng trong tất cả chu kỳ phát triển phần mềm và trong suốt quá trình phát triển phần mềm.

Quy trình phát triển phần mềm trong thực tế thường được thiết lập dựa trên các mô hình hoặc phương pháp phát triển phần mềm đã được kiểm chứng và sử dụng rộng rãi. Một số quy trình phát triển phần mềm phổ biến: mô hình Waterfall (thác nước), mô hình Agile, mô hình Incremental (Tăng trưởng), mô hình Spiral (xoáy ốc), mô hình V-Model (Mô hình V), DevOps; mô hình triển khai và tích hợp liên tục (Continuous Integration/Continuous Deployment - CI/CD)

Các quy trình này thường được tùy chỉnh và điều chỉnh để phù hợp với yêu cầu cụ thể của dự án và tổ chức. Chúng cũng có thể được kết hợp hoặc mở rộng để tạo ra một quy trình phát triển độc đáo phù hợp với hoàn cảnh cụ thể.

Không có quy trình phát triển nào là hoàn hảo và phù hợp với tất cả các tình huống. Việc tùy chỉnh và điều chỉnh quy trình để phù hợp với dự án cụ thể và nhóm phát triển là rất quan trọng.

Khi áp dụng quy trình phát triển phần mềm trong thực tế, có một số lưu ý quan trọng:

1. Xác định yêu cầu rõ ràng và chi tiết
2. Liên tục kiểm tra và cập nhật yêu cầu
3. Thực hiện kiểm thử liên tục
4. Luôn có sự giao tiếp và phản hồi tốt
5. Tập trung vào tính ưu tiên và quan trọng

6. Quản lý rủi ro
7. Kiểm soát phiên bản (Version Control)
8. Sử dụng công cụ hỗ trợ phát triển
9. Tạo tài liệu đầy đủ

2.2.3 Cách mô tả quy trình phần mềm

Để mô tả quy trình phát triển phần mềm một cách rõ ràng và dễ hiểu, chúng ta có thể sử dụng một biểu đồ hoặc sơ đồ, hoặc ký hiệu, ngôn ngữ. Dựa trên các ký hiệu chuẩn của ngành công nghiệp phát triển phần mềm. Có một vài cách để mô tả quy trình phần mềm như sau:

- Workflow
    - Theo luồng công việc
    - Trình tự các bước quy trình
    - Đây là cách thông dụng thể hiện quy trình
- Dataflow
    - Theo luồng dữ liệu
    - Xoay quanh 1 đơn vị dữ liệu
- Role/Action – Vai trò/tác nhân
    - Theo vai trò, tác nhân
    - Xoay quanh 1 tác nhân

Chúng ta đi vào phân tích 1 ví dụ, áp dụng cách mô tả quy trình phần mềm theo Workflow:

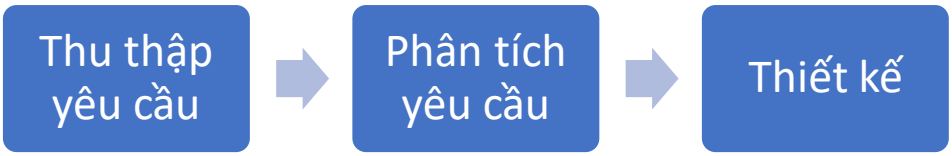
Bước 1: Thu thập yêu cầu (Requirements Gathering)

- Mục tiêu: Xác định yêu cầu từ khách hàng.
- Hoạt động:
    - Khảo sát và ghi lại yêu cầu từ khách hàng.
    - Xác định các tính năng cần thiết.

Bước 2: Phân tích yêu cầu (Requirement Analysis)

- Mục tiêu: Phân tích yêu cầu và chia thành các tính năng con.
- Hoạt động:

- Tách yêu cầu thành các tính năng và chức năng nhỏ hơn.
- Xác định các quy tắc/yêu cầu liên quan tới các tính năng này.



Hình 2-3: Ví dụ mô tả các bước theo workflow

2.3 Một số mô hình phát triển phần mềm phổ biến

2.3.1 Phân tích một số mô hình phần mềm phổ biến

Có nhiều mô hình phát triển phần mềm phổ biến được sử dụng trong ngành công nghiệp phần mềm. Dưới đây là một số mô hình phát triển phần mềm phổ biến:

- Mô hình Waterfall (Thác nước)
  - Mô hình Waterfall là một mô hình phát triển phần mềm tuyến tính và tuần tự. Các giai đoạn (yêu cầu, thiết kế, phát triển, kiểm thử, triển khai, và bảo trì) được thực hiện theo một chuỗi tuyến tính và không thể bước qua giai đoạn trước khi hoàn thành.
  - Mô hình này phù hợp cho các dự án có yêu cầu ổn định và không thay đổi thường xuyên.
- Mô hình Spiral (xoắn/xoáy ốc)
  - Mô hình Spiral kết hợp các phương pháp lặp và tăng dần (incremental). Nó tập trung vào việc kiểm soát rủi ro và đánh giá các rủi ro tiềm ẩn trong quá trình phát triển.
  - Mô hình này phù hợp cho các dự án có rủi ro cao và yêu cầu thay đổi thường xuyên.
- Mô hình Incremental (Tăng dần/tăng trưởng)
  - Mô hình Incremental chia dự án thành các phần nhỏ gọi là "increment" và mỗi increment đại diện cho một phần của sản phẩm hoàn chỉnh. Các

increment được phát triển độc lập và sau đó được kết hợp để tạo thành sản phẩm cuối cùng.

- Mô hình này phù hợp cho các dự án có khả năng chia nhỏ thành các phần nhỏ hơn và phát triển theo từng bước.

- Mô hình Agile
  - Agile không phải là một mô hình cụ thể mà là một triển khai linh hoạt của nhiều mô hình khác nhau như Scrum, Kanban, XP, và nhiều khái niệm khác. Agile tập trung vào việc phát triển phần mềm một cách linh hoạt, tương tác thường xuyên với khách hàng và thích nghi với sự thay đổi.
- Mô hình V-Model (Mô hình V):
  - Mô hình V-Model là một biến thể của mô hình Waterfall. Nó kết hợp các bước kiểm thử với các bước phát triển tương ứng. Mỗi bước phát triển đi kèm với một bước kiểm thử tương ứng.
  - Mô hình này phù hợp cho các dự án có yêu cầu ổn định và quy trình kiểm thử được định rõ.

Mỗi mô hình có ưu điểm và hạn chế của riêng nó, và việc chọn lựa mô hình nào phụ thuộc vào loại dự án, yêu cầu của khách hàng và sự phát triển công nghệ hiện tại.

a. Mô hình thác nước

- Mô hình thác nước là mô hình vòng đời lâu đời nhất; được đề xuất bởi Winston Royce vào năm 1970.
- Mô hình này được gọi là thác nước vì nó thường được vẽ với một chuỗi các hoạt động qua các giai đoạn của vòng đời "xuống dốc" từ trái sang phải: phân tích, yêu cầu, đặc tả, thiết kế, cài đặt, kiểm thử, bảo trì
- Có nhiều phiên bản của mô hình thác nước:
  - Các giai đoạn / hoạt động có thể được cấu trúc theo các mức độ chi tiết khác nhau
  - Phản hồi có thể linh hoạt hơn hoặc ít hơn

Ưu điểm:

- Tách nhiệm vụ trong mỗi giai đoạn 1 cách riêng biệt
- Khả năng hiển thị tốt, dễ theo dõi

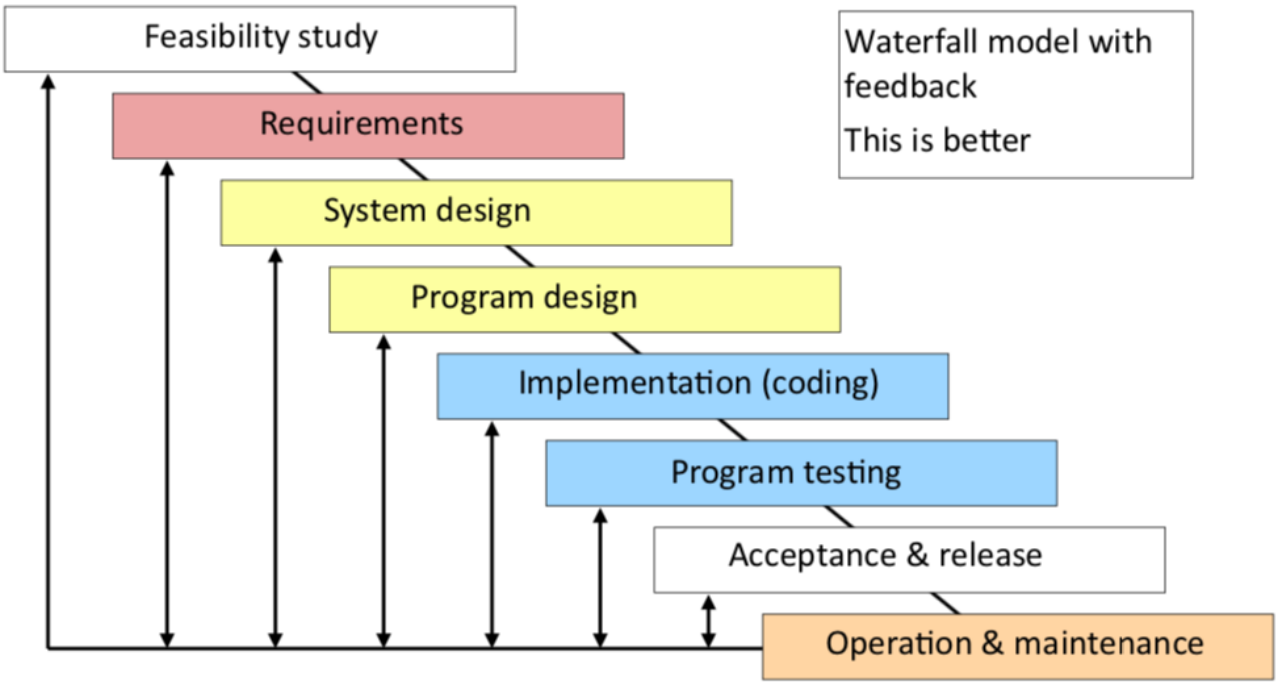
- Quy trình kiểm soát chất lượng ở mỗi bước
- Giám sát chi phí ở từng bước

Nhược điểm:

- Phụ thuộc vào các yêu cầu được xác định sớm từ đầu
- Không khả thi trong một số trường hợp đòi hỏi có nhiều thay đổi
- Trong thực tế, mỗi giai đoạn trong quy trình đều những cải tiến, phản hồi mới về các giai đoạn trước đó, thường đòi hỏi phải sửa đổi, điều chỉnh các giai đoạn trước đó.

Có thể nhận thấy rằng: mô hình thác nước trong nhiều trường hợp không đủ linh hoạt khi áp dụng vào trong các dự án có sự thay đổi thường xuyên về yêu cầu của người dùng.

b. Mô hình thác nước cải tiến



Hình 2-4: Các bước trong mô hình thác nước cải tiến (Modified Waterfall)

Nguồn: Utami Jayadi, software Process Models and the feasibility study

Mô hình thác nước đã sửa đổi hoạt động tốt nhất khi các yêu cầu được hiểu rõ và thiết kế đơn giản. Ví dụ: Chuyển đổi hệ thống xử lý dữ liệu thủ công trong đó các yêu cầu đã được hiểu rõ. Hoặc thích hợp trong trường hợp phiên bản mới của hệ thống có chức năng gần giống với sản phẩm trước đó; hoặc các phần của một hệ thống lớn trong đó

một số thành phần có yêu cầu được xác định rõ ràng và được tách biệt rõ ràng với phần còn lại của hệ thống.

c. Mô hình mẫu thử

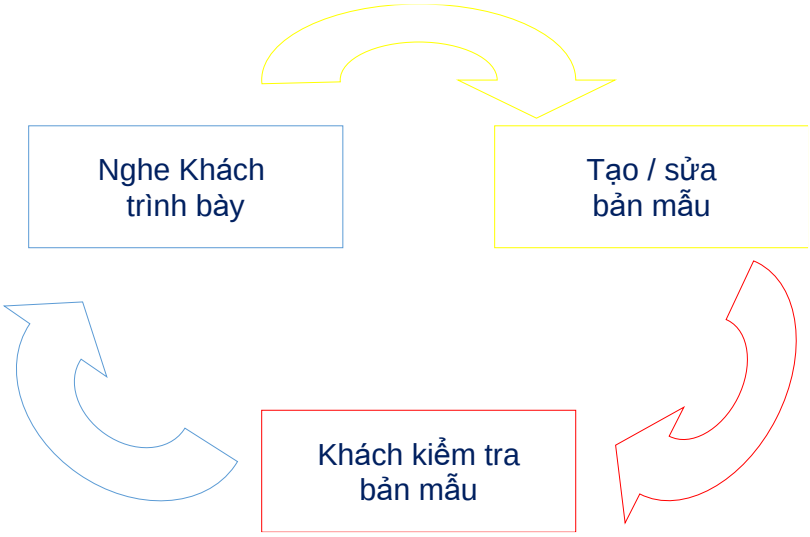
- Khi mới rõ mục đích chung chung của phần mềm, chưa rõ chi tiết đầu vào hay xử lý ra sao hoặc chưa rõ yêu cầu đầu ra
- Dùng để thu thập yêu cầu qua các thiết kế nhanh
- Các giải thuật, giải pháp kỹ thuật dùng làm bản mẫu có thể chưa nhanh, chưa tốt, miễn là có mẫu để thảo luận gợi yêu cầu của người dùng

Mô hình mẫu thử (Prototyping Model) là một trong các mô hình phát triển phần mềm, nơi tạo ra một phiên bản thử nghiệm (prototype) của ứng dụng để giúp hiểu rõ hơn về yêu cầu và cung cấp phản hồi từ khách hàng. Dưới đây là mô tả về các giai đoạn và các yếu tố chính của mô hình mẫu thử:

- Thu thập yêu cầu (Requirements Gathering):
    - Bước đầu tiên là thu thập yêu cầu từ khách hàng. Dự án cần hiểu rõ về mục tiêu, tính năng, và yêu cầu cơ bản.
- Xây dựng prototype (Prototype Development):
    - Trong giai đoạn này, một phiên bản thử nghiệm của ứng dụng được xây dựng. Đây không phải là phiên bản cuối cùng, mà chỉ là một bản ghi lại hình thức hoạt động dự định.
- Kiểm thử mẫu (Prototype Testing):
    - Sau khi xây dựng prototype, nó sẽ được kiểm tra để xác minh tính năng và hiệu suất cơ bản. Phản hồi từ kiểm thử này cung cấp thông tin quý giá để cải thiện phiên bản tiếp theo.
- Thu thập phản hồi (Gather Feedback):
    - Phiên bản thử nghiệm được đưa đến khách hàng để nhận phản hồi. Khách hàng cung cấp ý kiến, sửa đổi, và các yêu cầu bổ sung.
- Sửa chữa và cập nhật (Refinement and Updating):
    - Dựa trên phản hồi từ khách hàng, prototype sẽ được điều chỉnh và cập nhật để phù hợp hơn với yêu cầu thực tế.
- Lặp lại quy trình (Iterate):

- Các bước từ 2 đến 5 có thể lặp đi lặp lại nhiều lần cho đến khi khách hàng cảm thấy hài lòng với phiên bản thử nghiệm.

- Phát triển phiên bản cuối cùng (Final Product Development):
  - Sau khi prototype đã được chấp nhận và phản hồi từ khách hàng hợp lý, phiên bản cuối cùng của ứng dụng được phát triển.

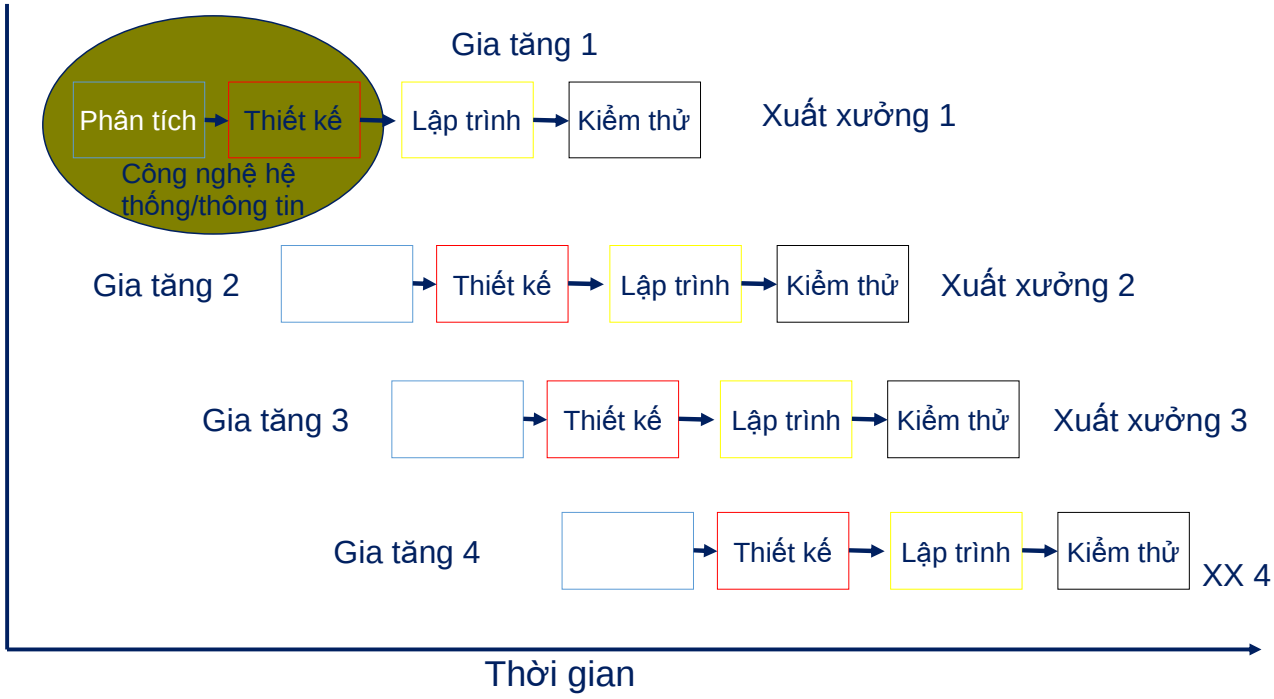


Hình 2-5: Mô hình mẫu thử (prototyping model)

d. Mô hình tăng trưởng (incremental model)

Ý tưởng của mô hình này như sau:

- Kết hợp mô hình tuần tự và ý tưởng lặp lại của chế bản mẫu
- Sản phẩm với những yêu cầu cơ bản nhất của hệ thống được phát triển
- Các chức năng với những yêu cầu khác được phát triển thêm sau (gia tăng)
- Lặp lại quy trình để hoàn thiện dần



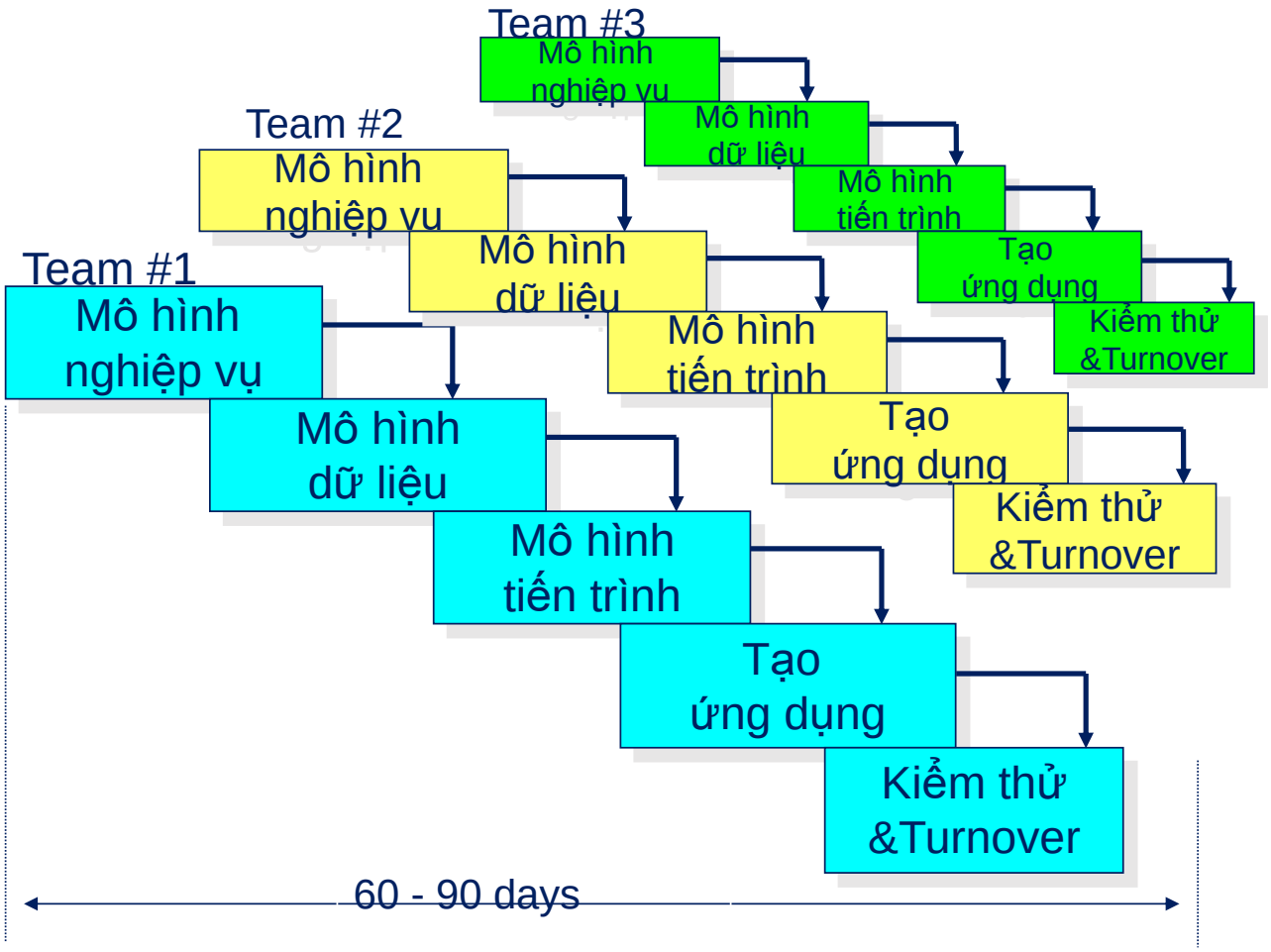
Hình 2-6: Mô hình tăng trưởng (incremental model)

e. Mô hình phát triển ứng dụng nhanh (Rapid Application Development - RAD)

Đây là quy trình phát triển phần mềm mà trong quá trình thực hiện, sản phẩm của qus trình hoạt động sẽ từng bước bồi đắp (Incremental software development) với mỗi chu trình phát triển rất ngắn (60-90 ngày). Thường thường các hệ thống phần mềm được xây dựng dựa trên hướng thành phần (Component-based construction) với khả năng tái sử dụng (reuse).

Mô hình phát triển ứng dụng nhanh là một phương pháp phát triển phần mềm tập trung vào việc tạo ra ứng dụng nhanh chóng bằng cách sử dụng các công cụ và kỹ thuật hỗ trợ quá trình phát triển. RAD tập trung vào việc giảm thiểu thời gian phát triển và tạo ra một phiên bản sớm để thu thập phản hồi từ khách hàng.

Khi phát triển một hệ thống, đội ngũ phát triển có thể gồm một số nhóm (teams), mỗi nhóm làm hoạt động tuân theo quy trình RAD theo các pha: Mô hình nghiệp vụ, Mô hình dữ liệu, Mô hình xử lý, Tạo ứng dụng, Kiểm thử và đánh giá (Business, Data, Process, Appl. Generation, Test).



Hình 2-7: Mô hình phát triển phần mềm nhanh (RAD)

Chương 3 PHƯƠNG PHÁP AGILE

Nội dung:

- Tổng quan về phương pháp Agile
- Các nguyên lý, nguyên tắc cơ bản của Agile
- Ưu – nhược điểm của phương pháp Agile
- Giới thiệu về Scrum

3.1 Tổng quan về phương pháp Agile

3.1.1 Một số định nghĩa và chú ý

Dưới đây là một số định nghĩa phổ biến hay được nhắc tới trong quy trình phát triển phần mềm:

- Phát triển phần mềm linh hoạt hay lập trình linh hoạt (tiếng Anh: Agile software development hay Agile programming) là một phương thức thực hiện các dự án công nghệ phần mềm, phương thức này khuyến khích sự thay đổi khi phát triển dự án và đưa sản phẩm đến tay người dùng sao cho nhanh nhất [wikipedia].
- Agile Software Development là một thuật ngữ chung chỉ tất cả các kỹ thuật và phương pháp phát triển phần mềm theo triết lý Agile.
- The Agile methodology is a project management approach that involves breaking the project into phases and emphasizes continuous collaboration and improvement. Teams follow a cycle of planning, executing, and evaluating [https://www.atlassian.com/agile].

Bên cạnh đó, chúng ta cần chú ý một số vấn đề sau:

- Agile không phải là một công cụ hay một phương pháp duy nhất
    - Agile là triết lý được đưa ra vào năm 2001.
    - Từ triết lý, nguyên lý của Agile có thể áp dụng nhiều phương pháp khác nhau trong quy trình phát triển phần mềm
- Agile thay đổi đáng kể cách tiếp cận phát triển phần mềm
    - Hạn chế việc nặng theo hướng tài liệu (như mô hình thác nước).

- Tương tác thông qua các quy trình và công cụ

3.1.2 Quy trình xây dựng phần mềm nói chung

Quy trình xây dựng phần mềm (Software Development Life Cycle - SDLC) là một loạt các bước và hoạt động được thực hiện để phát triển một ứng dụng phần mềm. Thông thường các bước này đều đã được mô tả hay ít nhiều chuẩn hóa. Dưới đây trình bày quy trình xây dựng phổ biến có 6 giai đoạn chính:

1. Thu thập yêu cầu (Requirements Gathering)
    - Bước đầu tiên trong quy trình SDLC là thu thập thông tin và yêu cầu từ khách hàng hoặc người dùng cuối.
    - Điều này bao gồm việc định rõ các tính năng, chức năng và các yêu cầu kỹ thuật khác của ứng dụng.
    - Trong một số trường hợp, bước này còn có tên là nghiên cứu tiền khả thi (feasibility study)
2. Phân tích yêu cầu (Requirements Analysis)
    - Tại giai đoạn này, nhóm phát triển phân tích và đánh giá yêu cầu để đảm bảo rằng chúng đáp ứng nhu cầu của người dùng.
    - Các yêu cầu được chia thành các mô-đun hoặc tính năng cụ thể để tiện việc quản lý.
3. Thiết kế (Design)
    - Giai đoạn này tập trung vào việc thiết kế kiến trúc tổng thể của hệ thống.
    - Thiết kế có thể bao gồm cả thiết kế cơ sở dữ liệu, giao diện người dùng, và các thành phần chức năng khác.
    - Trong một số tài liệu, bước này có thể được tách thành 2 bộ phận: (1) thiết kế hệ thống và (2) thiết kế chương trình. Trong đó (1) thường tập trung vào kiến trúc tổng thể của hệ thống, các mô-đun, các bộ phận cấu thành nên hệ thống. Còn (2) thì thường nhắc đến việc lập trình và thiên về các thiết kế lớp (trong lập trình hướng đối tượng) hay thiết kế về cơ sở dữ liệu cũng như các tài liệu liên quan.
4. Phát triển (Development/implementation)

- Ở giai đoạn này, mã nguồn thực tế của ứng dụng được viết và các tính năng được triển khai.
- Nhóm phát triển tuân thủ các quy tắc lập trình và tiêu chuẩn để đảm bảo mã nguồn được viết một cách cấu trúc và dễ bảo trì.

5. Kiểm thử (Testing)
    - Ở giai đoạn này, các bài kiểm tra được thực hiện để đảm bảo rằng ứng dụng hoạt động chính xác và đáp ứng các yêu cầu đã đề ra.
    - Các loại kiểm thử bao gồm kiểm thử đơn vị, kiểm thử tích hợp, kiểm thử hệ thống, và kiểm thử chấp nhận người dùng (User Acceptance Testing - UAT).
6. Triển khai và duy trì (Deployment and Maintenance)
    - Ở giai đoạn này, ứng dụng được triển khai và đưa vào sử dụng thực tế.
    - Công việc bảo trì sau triển khai bao gồm việc sửa lỗi, cập nhật và bảo trì hệ thống.
    - Bước này trong một số mô hình còn có tên là kiểm tra và xuất bản phần mềm.

Một số quy trình phát triển phần mềm khác như Agile, Scrum, Waterfall cũng có các bước và hoạt động tương tự nhưng có sự linh hoạt và ưu điểm riêng. Việc chọn lựa quy trình nào phụ thuộc vào dự án cụ thể, yêu cầu của khách hàng và sự phát triển công nghệ hiện tại.

3.1.3 Lịch sử của Agile

Agile là một phương pháp quy trình phát triển phần mềm linh hoạt và tương đối mới so với các quy trình truyền thống như Waterfall. Việc xuất hiện phương pháp này như là một phản ứng với các vấn đề hạn chế của các phương pháp phát triển truyền thống. Chúng ta có thể lướt qua một số mốc quan trọng trong lịch sử phát triển của quy trình phần mềm như sau:

1. *Những năm 1970-1980:* Xuất hiện của các phương pháp phát triển phần mềm đầu tiên (hiện nay được gọi là phương pháp truyền thống) như Waterfall, trong

đó các bước phát triển được thực hiện theo một chuỗi liên tiếp nhau và tuyến tính.

2. *Những năm 1990:* Một số nhà nghiên cứu và chuyên gia trong lĩnh vực phát triển phần mềm như Jeff Sutherland, Ken Schwaber, và Kent Beck, đã bắt đầu nghiên cứu các phương pháp phát triển linh hoạt và linh động hơn.
3. *Những năm 1990 đầu:* Jeff Sutherland và Ken Schwaber phát triển Scrum, một phương pháp quy trình phát triển phần mềm linh hoạt.
4. *Năm 2000-2001:* Nhóm gồm nhiều người đại diện cho các phương pháp phát triển phần mềm như Scrum, Extreme Programming (XP), Crystal, DSDM, FDD, và Adaptive Software Development, đã tổ chức một cuộc hội thảo và ký kết một tuyên bố gọi là "bản tuyên ngôn của Agile - Manifesto for Agile Software Development". Tài liệu này ghi lại các giá trị cốt lõi và phương hướng vận hành của phương pháp Agile.
5. *Sau năm 2001:* Phong trào Agile lan rộng và trở thành một trong những phương pháp phát triển phần mềm phổ biến nhất trên thế giới.

Nếu nghiên cứu, tìm hiểu riêng về Agile thì chúng ta có một số thông tin cụ thể la vào năm 2000, một nhóm mười bảy người cùng nhau tham dự một cuộc họp đặc biệt tại Mỹ. Tại thời điểm họp, tất cả trong số họ đều có mong muốn đề nghị, đề xuất những phương pháp xây dựng và phát triển phần mềm mới gọn nhẹ và bớt nặng nề hơn so với cách làm truyền thống.

Đứng trên quan điểm và mục tiêu đó, họ thấy rằng những phương pháp mới này cần có những điểm chung như sau:

- Rút ngắn thời gian đưa sản phẩm ra thị trường (Speed to market)
- Phản hồi nhanh chóng (Rapid feedback)
- Cải tiến, cải thiện sản phẩm phần mềm liên tục (Continuous improvement)

Các thành viên tham dự đã đưa ra "4 values" - 4 giá trị, điểm chung mà các phương pháp mới khuyến nghị nên có. Bốn giá trị/đặc tính này được gọi là *Agile Manifesto - bản tuyên ngôn của Agile.*

Các phương pháp Agile như Scrum, Kanban, XP, Lean, và nhiều khái niệm khác đã phát triển và mở rộng từ lúc đó. Agile đã trở thành một cách tiếp cận phổ biến trong phát triển phần mềm và được áp dụng trong nhiều ngành công nghiệp khác nhau.

3.1.4 Bản tuyên ngôn của Agile

a) **Các cá nhân và tương tác** hơn là **các quy trình và công cụ**

b) **Phần mềm chạy tốt** hơn là **tài liệu đầy đủ/toàn diện**

c) **Cộng tác với khách hàng** hơn là **thương thảo hợp đồng**

d) **Đáp ứng với thay đổi** hơn là **bám sát kế hoạch**

Trong đó, các mệnh đề, nội dung ở bên trái được coi trọng hay nhấn mạnh hơn các ý ở bên phải. Một trong những điều cơ bản trong công nghệ phần mềm là không có một cách tốt nhất để xây dựng một phần mềm. Tuyên ngôn Agile đưa ra các giá trị giúp cho các nhóm phát triển dự án tuân theo cách tư duy của Agile. Khi mỗi người áp dụng được tư duy Agile vào trong suy nghĩ, trong công việc của mình, Agile thực sự sẽ giúp ích trong việc xây dựng phần mềm/hệ thống/sản phẩm được tốt hơn.

Nghiên cứu bản tuyên ngôn của Agile, chúng ta có thể hiểu như sau:

a. **Các cá nhân và tương tác** hơn là **các quy trình và công cụ**

- Cá nhân và sự tương tác giữa họ là cốt yếu để nhóm đạt được hiệu suất cao
- Các thành viên trong nhóm thể hiện những hành vi quan trọng sau:
- Tôn trọng giá trị của mỗi cá nhân
- Trung thực trong truyền thông
- Minh bạch về thông tin, dữ liệu, hoạt động
- Tin tưởng vào sự hỗ trợ của mỗi cá nhân với nhóm, cam kết với nhóm và tập trung vào các mục tiêu của nhóm

b. **Phần mềm chạy tốt** hơn là **tài liệu đầy đủ / toàn diện**

- Khuyến nghị các nhóm cung cấp phần mềm chạy tốt sau một khoảng thời gian nhất định.
- Đồng thuận với định nghĩa hoàn thành: “phần mềm chạy tốt”

- Một phần của chức năng hoàn thành chỉ khi các tính năng của chúng vượt qua tất cả các kiểm thử.
- Kiểm thử và có thể: kiểm thử đơn vị (unit test), kiểm thử hệ thống, kiểm thử tích hợp, kiểm thử hiệu năng và kiểm thử chấp nhận của khách hàng,…
- Có thể được vận hành bởi người dùng cuối.

c. **Cộng tác với khách hàng** hơn là **thương thảo hợp đồng**

- Để khách hàng tham gia vào quá trình phát triển phần mềm là hết sức cần thiết để dẫn tới thành công
- Chọn ra một vị đại sứ của khách hàng
- Đại diện tất cả khách hàng bao gồm cả quản lý, dịch vụ khách hàng, và các bên liên quan khác
- Cộng tác với khách hàng (hoặc đại diện của khách hàng) trên cơ sở hàng ngày.
- Tạo ra một vị trí đặc biệt trong đội hình phát triển để dành riêng cho vị khách hàng đại diện này

d. **Đáp ứng với thay đổi hơn là bám sát kế hoạch**

- Tìm kiếm sự phản hồi của khách hàng trong suốt dự án, ngay khi sản phẩm đang được phát triển
- Thay đổi các kế hoạch trong một khoảng thời gian đều đặn dựa trên những thông tin phản hồi từ phía khách hàng
    - Có kế hoạch để thay đổi
    - Thiết lập các quy trình (ví dụ: Sơ kết, Cải tiến) để thay đổi các ưu tiên thường xuyên dựa trên thông tin phản hồi của khách hàng

Bản Tuyên ngôn Agile tập trung vào việc đặt ưu tiên vào việc tạo ra phần mềm có giá trị thực sự và thúc đẩy sự tương tác tích cực giữa các thành viên trong nhóm phát triển và với khách hàng.

3.2 Các nguyên lý, nguyên tắc cơ bản của Agile

3.2.1 Cách tiếp cận của Agile

Khái niệm Agile (viết tắt của Agile Software Development) có nghĩa là phương thức phát triển phần mềm linh hoạt, được ứng dụng trong quy trình phát triển phần mềm với mục tiêu là đưa sản phẩm đến tay người dùng càng nhanh càng tốt.

Thực chất, Agile giống như một phương pháp luận, một triết lý dựa trên nguyên tắc phân đoạn vòng lặp (iterative) và tăng trưởng (incremental).

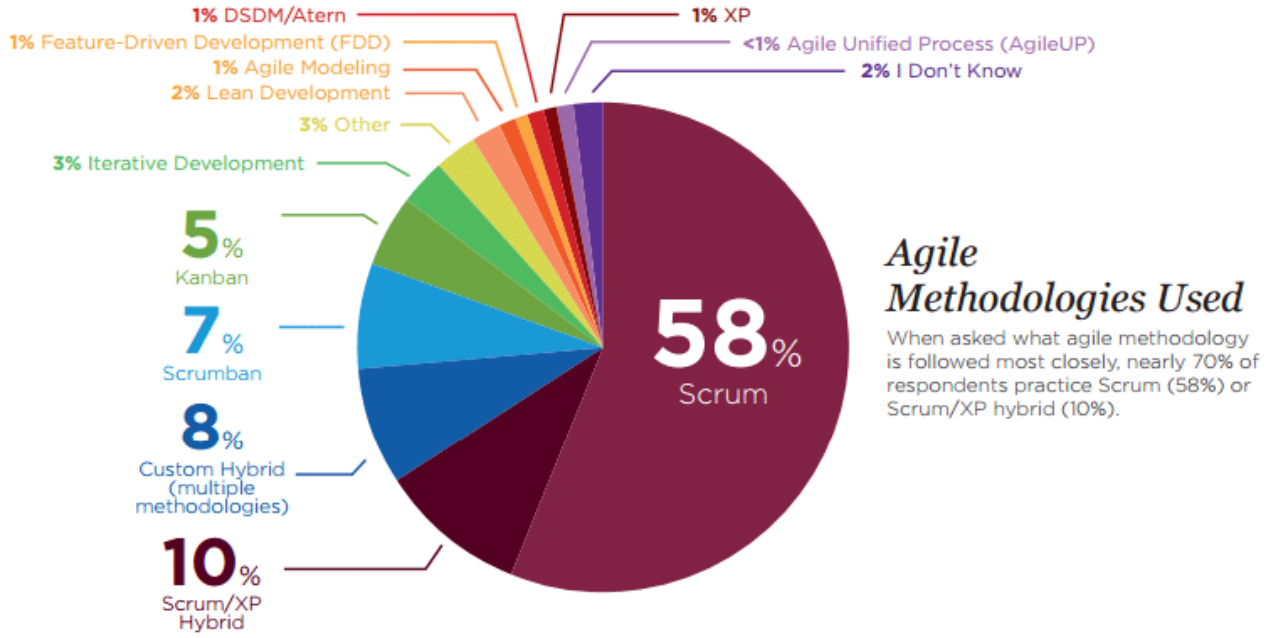
Agile không định nghĩa ra một phương pháp cụ thể nhưng lại có nhiều phương pháp khác nhau thỏa mãn và hướng theo các tiêu chí của nó. Ví dụ như Scum, Kanban, Extreme Programming…

- Agile không định nghĩa ra một phương pháp/cách thức cụ thể
- Có nhiều phương pháp khác nhau thỏa mãn và hướng theo các tiêu chí của Agile
- Thực tế: Nhiều công ty đã kết hợp các phương pháp lại với nhau

3.2.2 Một số ví dụ về phương pháp phát triển phần mềm theo Agile

Có thể kể tên một số cách áp dụng Agile phổ biến như:

- Scum
- Extreme programming (XP)
- Kanban
- Crystal methodology
- Feature-driven development (FDD)



Hình 3-1: Biểu đồ mức độ ưa chuộng sử dụng các phương pháp Agile trong doanh nghiệp

Nguồn: Budiman, Thomas & Suroso, Jarot. (2017). Optimizing IT Infrastructure by Virtualization Approach. IOP Conference Series: Materials Science and Engineering

3.2.3 Đặc trưng của Agile

a. Tính lặp

- Dự án sẽ được thực hiện trong các phân đoạn lặp đi lặp lại (Iteration hoặc Sprint), thường có khung thời gian ngắn (từ 1-4 tuần).
- Trong mỗi phân đoạn, nhóm phát triển thực hiện đầy đủ các công việc cần thiết như lập kế hoạch, phân tích yêu cầu, thiết kế, triển khai, kiểm thử.
- Trong mỗi phân đoạn, nhóm phát triển lựa chọn một số chức năng/công việc trong danh sách. Chúng ta có lhái niệm: Danh sách chức năng/công việc **(Product baclog item)** - là danh sách các chức năng có thể được sắp xếp theo thứ tự ưu tiên mà một sản phẩm nên có

b. Tính tăng trưởng (Increments & Evolutionary)

- Mỗi bước lặp giống như phát triển một phần mềm hoàn chỉnh: xác định yêu cầu, phân tích thiết kế, viết mã, kiểm thử, viết tài liệu
- Kết quả sau mỗi giai đoạn phát triển đều được kiểm tra kĩ và sẵn sàng đưa vào sản phẩm.

- Dự án được chia thành nhiều phần/ giai đoạn phát triển nhỏ (sprint) nối tiếp nhau, sau mỗi giai đoạn, hệ thống được từng bước hoàn thiện
- Các phần/đoạn thời gian này thường kéo dài từ 2 đến 4 tuần
- Theo thời gian, các phần đoạn nối tiếp nhau, sản phẩm của dự án được hoàn thiện dần tiến tới đáp ứng yêu cầu khách hàng
- Quá trình phát triển được thực hiện bởi nhóm nhỏ, thường từ 4 - 9 thành viên

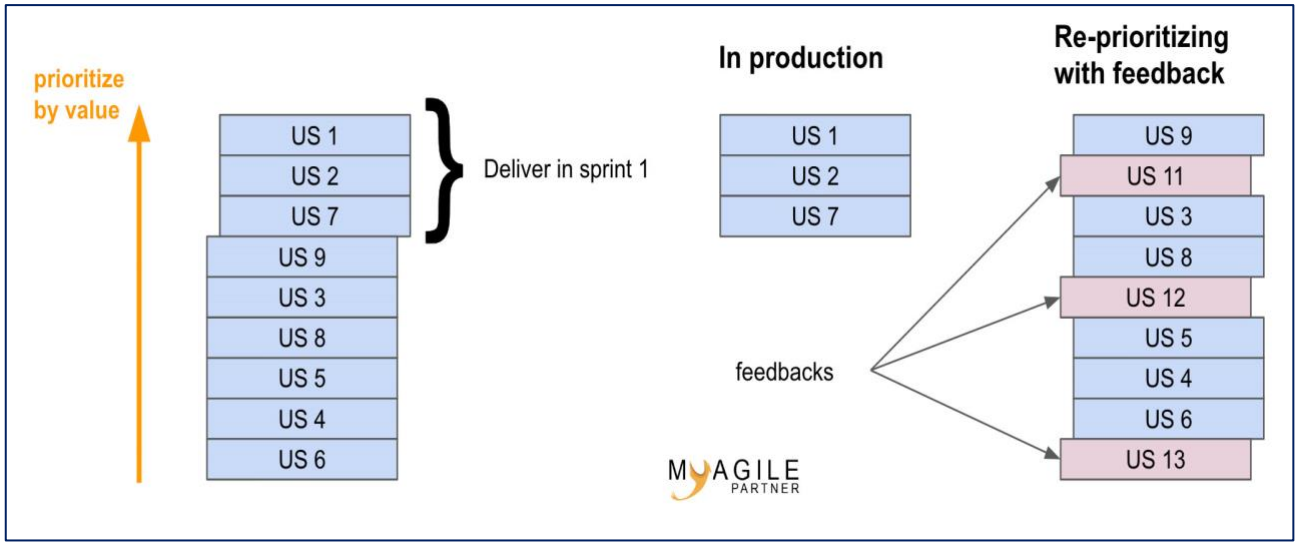
c. Tính thích ứng/thích nghi (Adaptive)
   - Kế hoạch sẽ liên tục được điều chỉnh
   - Phù hợp theo các phân đoạn ngắn của dự án
   - Kịp thời những yêu cầu thay đổi của khách hàng hay những tác động của các vấn đề khác

d. Nhóm tự tổ chức và liên chức năng
   - Nhóm tự tổ chức sẽ chịu trách nhiệm từng mảng công việc riêng biệt theo mỗi phân đoạn của dự án.
   - Phù hợp với công việc được giao để có thể hoàn thành nhiệm vụ

e. Quản lý tiến trình thực nghiệm (Empirical Process Control)
   - Dựa vào dữ liệu thực tế để đưa ra các quyết định cho công việc
   - Rút ngắn thời gian phản hồi và tăng tính linh hoạt

f. Giao tiếp trực tiếp (Face-to-face communication)
   - Agile đánh giá cao việc trao đổi trực tiếp hơn là giao tiếp thông qua giấy tờ.
   - Agile còn khuyến khích nhóm dự án trực tiếp nói chuyện với khách hàng để hiểu rõ điều họ đang cần.

g. Phát triển dựa trên giá trị (Value-based development)



Hình 3-2: Biểu đồ Value Driven-Development (VDD)

Nguồn: Judicaël Paquet, Value Driven-Development (VDD), myagilepartner

- Thay vì theo lý thuyết và kế hoạch giả định , các nhóm trong mô hình Agile sẽ dựa vào dữ liệu thực tế để đưa ra các quyết định cho công việc.
- Đặc điểm này được xây dựng dựa trên nguyên tắc tích cực tương tác với khách hàng. Thông qua việc tương tác đó mà nắm được những yêu cầu có mức độ ưu tiên cao, quan trọng; loại bỏ hay giảm bớt các việc không quan trnọng.

3.2.4 Các nguyên tắc của Agile

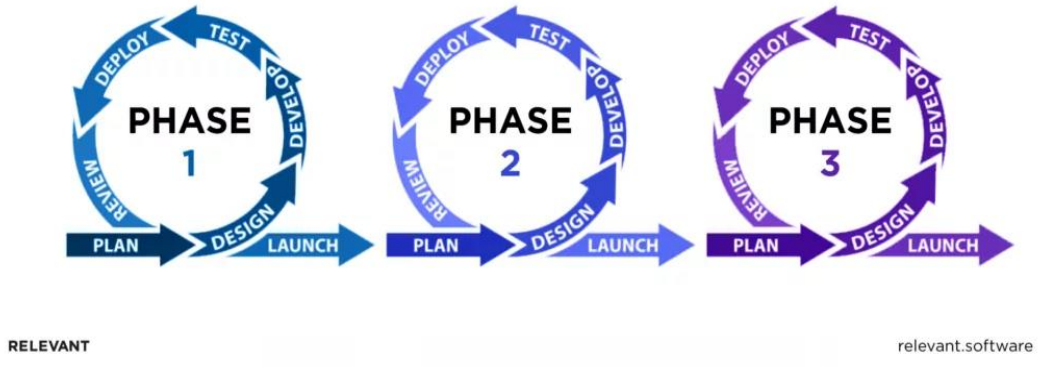
Agile có 12 nguyên tắc như sau

- Sự hài lòng của khách hàng cần đặt lên hàng đầu
- Luôn thay đổi khi cần thiết trong quá trình phát triển
- Thường xuyên ra mắt phần mềm làm việc, ưu tiên trong khung thời gian ngắn
- Luôn hợp tác với những bên liên quan cùng với bộ phận phát triển kinh doanh.
- Những cá nhân có động lực được làm việc trong môi trường có sự hỗ trợ tốt, giúp mang lại hiệu quả công việc
- Trao đổi trực tiếp.
- Phần mềm hoạt động (working software) là thước đo quan trọng nhất để đo sự tiến bộ.
- Thúc đẩy sự phát triển một cách bền vững, có tính ổn định liên tục.
- Nâng cao sự linh hoạt của chi tiết kỹ thuật và thiết kế.

- Đề cao tính đơn giản.
- Kiến trúc, yêu cầu, thiết kế tốt nhất được tạo ra từ các nhóm tự tổ chức (seft-organizaing teams).
- Điều chỉnh hành vi phù hợp.

3.3 Ưu – nhược điểm của phương pháp Agile

3.3.1 Phân tích quy trình Agile



Hình 3-3: Quy trình của Agile

(Nguồn: Anna Dziuba, Relevant Software, 2023)

- Chia dự án tổng thể thành các phân đoạn lặp đi lặp lại (Iteration)
  - Kế hoạch tổng thể (Initial Planning) của dự án được lập trong những tuần đầu tiên
  - Đại diện khách hàng và nhóm phát triển cùng nhau chia dự án thành các phân đoạn, ước lượng thời gian và công sức thực hiện chúng và vạch ra lịch trình phát triển cho từng phần tăng trưởng
- Trong mỗi phân đoạn, thực hiện đầy đủ các công việc cần thiết
  - Lập kế hoạch
  - Phân tích thiết kế
  - Phát triển
  - Kiểm thử
  - Triển khai/bàn giao
  - ….

a. Lập kế hoạch

- Kế hoạch tổng thể sẽ được điều chỉnh tùy theo tình hình của dự án.
- Mỗi phần tăng trưởng có một kế hoạch thực hiện cụ thể được vạch ra vào đầu mỗi vòng lặp. Đội dự án sẽ họp mặt hàng ngày để cập nhật tình hình công việc

b. Phân tích thiết kế

- Phân tích yêu cầu: đại diện của khách hàng sẽ ngồi làm việc chung với đội

  dự án. Các lập trình viên chỉ việc đến trao đổi trực tiếp với người này
- Thiết kế: của sản phẩm được cải tiến liên tục. Hoạt động này được thực hiện nhờ phương pháp phát triển dựa trên hướng phát triển thử nghiệm

c. Lập trình/phát triển

- Các lập trình viên (LTV) tích hợp viết các đoạn mã (vài giờ một lần) và đảm bảo rằng phiên bản mới đủ tiêu chuẩn về mặt kĩ thuật để bàn giao ngay cho khách hàng
- Mã nguồn được coi là sở hữu tập thể. Mọi người được yêu cầu sửa lỗi bất kể lỗi đó do ai gây ra

d. Kiểm thử

- Tất cả các thành viên trong dự án đều có trách nhiệm đảm bảo chất lượng sản phẩm
- Các LTV thực hiện kiểm thử các thành phần riêng lẻ và kiểm thử tích hợp
- Khách hàng kiểm tra công việc của LTV và trợ giúp bằng các kiểm thử của khách hàng
- Tìm ra lỗi, phân tích nguyên nhân, tìm cách cải tiến quy trình để ngăn ngừa lỗi tái diễn

e. Triển khai/ Bàn giao sản phẩm

- Sau mỗi phân đoạn thực hiện, kết quả đạt được dưới dạng sản phẩm sẽ được bàn giao, đưa cho khách hàng xem xét, góp ý.
- Nếu không có yêu cầu thay đổi thì tiến hành thực hiện phân đoạn tiếp theo đến khi sản phẩm hoàn thành thì bàn giao

3.3.2 Điều kiện áp dụng của Agile

Không phải trường hợp nào cũng áp dụng được phương pháp Agile. Để áp dụng được Agile, các dự án của thường có đặc điểm sau:

- Mức độ rủi ro thấp
    - Dự án theo Agile tốn rất nhiều chi phí : Dự án nào có tính khả thi thấp, tức là mức độ rủi ro cao thì không nên áp dụng mô hình này. Ví dụ như khách hàng thường xuyên thay đổi yêu cầu thì bên làm dự án sẽ phải thực hiện đi thực hiện lại cho đến khi phù hợp yêu cầu. Như vậy, nếu thay đổi quá nhiều, rủi ro cuối cùng dự án vẫn không thể hoàn thành thì rất tốn thời gian, công sức và chi phí.
- Thành viên nhóm có kinh nghiệm: Agile tập trung cho các dự án nhỏ và có thời gian hoàn rất ngắn. Do đó:
    - Phương pháp Agile cần có các cá nhân đa năng: có động lực, biết nghiên cứu, phân tích, sáng tạo, và có các kỹ năng giao tiếp cần thiết để hiểu thấu các vấn đề của khách hàng.
    - Làm việc nhóm có tính kỷ luật
    - Có kỹ năng hoạt động theo nhóm tốt
- Nhu cầu thay đổi thường xuyên
    - Tiếp xúc thường xuyên với khách hàng, trao đổi thường xuyên để cùng hoàn thiện dự án.
    - Đối với các dự án có tính ổn định cao, ít thay đổi thì áp dụng phương pháp này là không cần thiết.
- Kích thước nhỏ, các thành viên làm việc cùng một địa điểm
    - Nhóm không cần quá nhiều thành viên,
    - Làm việc tại cùng một địa điểm, trao đổi thông tin có hiệu quả hơn
- Văn hóa công ty phù hợp
    - Các thành viên đưa ra những chính kiến của mình trong quá trình làm việc nhóm
    - Thành viên có thể chủ động linh hoạt công việc

3.3.3 Lợi ích của áp dụng quy trình theo Agile

Khi áp dụng mô hình Agile vào quy trình phát triển dự án phần mềm thì thường có các lợi ích như sau:

- Đảm bảo tiến độ: Mỗi phân đoạn trong Agile (thường từ 1 – 4 tuần. Kỹ thuật này giúp **rút ngắn thời gian** và đảm bảo bảo cung cấp đúng thời điểm. Bên cạnh đó việc thiết kế và triển khai thử nghiệm được thực hiện đồng thời. Mục đích chính là làm giảm thời gian, đưa phần mềm đến tay người dùng sớm nhất. Qua đó, toàn bộ chu trình phát triển phần mềm được đẩy nhanh tiến độ.
- Tiết kiệm chi phí: Chỉ có những thứ thực sự cần thiết mới được phát triển. Nhờ vào định hướng này, tài nguyên và thời gian tiêu hao ít hơn.
- Minh bạch: nhà phát triển và chủ dự án đều giảm sát mọi thứ. Họ thường xuyên đưa ra phản hồi trước và sau một vòng lặp. Việc này nhằm đảm bảo các mục tiêu đều được đáp ứng. Các bên liên quan cập nhập thông tin thông qua báo cáo hàng ngày và minh bạch (trong mô hình Scrum)
- Năng suất: Mỗi thành viên tự chịu trách nhiệm, cam kết chất lượng ở mức cao. Luồng thông tin và giao tiếp nhanh chóng cũng làm tăng hiệu suất hơn.
- Cải thiện chất lượng
- Đem đến sự hài lòng và cam kết cho khách hàng
- Dễ dàng hợp tác nhóm
- Thích ứng nhanh chóng: Phương pháp Agile cho phép dự án thích ứng. Các nhóm phát triển trong quá trình hoạt động hiểu rằng yêu cầu sẽ liên tục biến đổi trong phát triển sản phẩm. Ngay sau khi nhận ra sai lệch nào đó, nhóm phát triển có thể phản ứng nhanh và tiến hành điều chỉnh.

3.3.4 Ưu nhược điểm của Agile

a. Ưu điểm

- Phù hợp với những dự án nhỏ thường có những yêu cầu không được xác định rõ ràng (có thể thay đổi thường xuyên).
- Khách hàng có thể được xem trước từng phần dự án trong quá trình phát triển, luôn sẵn sàng cho bất kỳ thay đổi từ phía khách hàng.

- Agile chia dự án thành những phần nhỏ và giao cho nhóm phát triển, hàng ngày tất cả mọi người phải họp trong khoảng thời gian ngắn để thảo luận về tiến độ và giải quyết những vấn đề nảy sinh nếu có nhằm đảm bảo đúng quy trình phát triển dự án.
- Tỉ lệ thành công của các dự án sử dụng Agile thường cao hơn các quy trình khác.

b. Nhược điểm:

- Khó xác định về loại dự án phần mềm nào phù hợp nhất cho cách tiếp cận Agile
- Nhiều tổ chức lớn gặp khó khăn trong việc chuyển từ phương pháp truyền thống sang một phương pháp Agile (linh hoạt).
- Khi Agile có rủi ro:
    - Phát triển quy mô lớn (> 20 nhà phát triển)
    - Phát triển phân tán (các nhóm không nằm chung)
    - Khách hàng hoặc người liên hệ không đáng tin cậy
    - Bắt buộc quy trình nhanh trong nhóm phát triển
    - Nhà phát triển thiếu kinh nghiệm

3.4 Giới thiệu về Scrum

3.4.1 Giới thiệu chung

a. Khái niệm Scrum

- Tên gọi: Scum là thuật ngữ dùng trong môn bóng bầu dục. Cả đội tập hợp lại theo cái mà họ gọi là scrum làm việc cùng nhau để đưa bóng về phía trước. Như vậy có thể hiểu, Scrum là từ mang nghĩa 1 nhóm người cùng nhau làm việc đưa mục tiêu/sản phẩm phát triển.
- Có nhiều phương pháp khác nhau thỏa mãn và hướng theo các tiêu chí của Agile
- Scrum là một dạng của phương pháp Agile, là Framework phổ biến nhất của Agile hiện nay.
- Scrum được thiết kế để áp dụng cho các dự án cần được xây dựng và cải tiến liên tục

- Ngoài các nguyên lý, nguyên tắc của Agile, Scrum bổ sung chi tiết một số quy định cụ thể hơn về:
- Quy trình hoàn chỉnh trong quá trình phát triển phần mềm
- Vai trò, trách nhiệm của các cá nhân tham gia dự án

b. Lịch sử ra đời của Scrum

- Scrum lần đầu tiên được giới thiệu bởi Hirotaka Takeuchi vào năm 1986 trong bài báo của Harvard Business Review
- Scrum tiếp tục được phát triển và hệ thống hóa bởi Ken Schwaber và Jeff Sutherland vào năm 1995, khi họ xuất bản *Tuyên ngôn Agile*
- Scrum được phát triển mục đích là để thay thế cho waterfall.
  - Waterfall: dự án được chia thành các giai đoạn tuần tự.
  - Scrum: Các lập trình viên có thể tạo ra sản phẩm tốt hơn nếu được chủ động và linh hoạt với các yêu cầu từ khách hàng.

3.4.2 Nguyên tắc của Scrum

1. Về mặt lý thuyết, Scrum đưa ra một số yêu cầu cơ bản như sau

- *Minh bạch*: Thông tin về quy trình, công việc được hiển thị rõ ràng, dễ dàng tiếp cận cho tất cả thành viên trong nhóm và các bên liên quan.
- *Giám sát, kiểm tra:* Giám sát chất lượng sản phẩm và tiến độ dự án liên tục. Thông qua đó điều chỉnh, thích nghi để đạt mục tiêu
- *Thích nghi:* Bất cứ hoạt động, quy trình nào làm ảnh hưởng tới các mục tiêu đã thống nhất từ trước đều cần phải được điều chỉnh sớm nhất có thể.

2. Vai trò của các thành viên trong Scrum

➢ ***Scrum Team:*** là một đội ngũ tinh gọn, bao gồm: 1 Scrum Master, 1 Product Owner và các Developers.

1. Không có phân cấp hay tổ chức con
2. Có tính đa năng: tập hợp tất cả các thành viên sẽ có tất cả kỹ năng cần thiết để tạo ra giá trị sau mỗi vòng lặp.
3. Tự quản: tự ra quyết định ai làm gì, khi nào và như thế nào

4. Chịu trách nhiệm cho tất cả những hoạt động liên quan tới sản phẩm: cộng tác các bên, kiểm định, bảo trì, thử nghiệm, nghiên cứu phát triển…

➢ ***Scrum Master:*** Là người chịu trách nhiệm triển khai Scrum và đảm bảo sự hoạt động hiệu quả của cả nhóm.

Hình 3-4: Vai trò của Scrum Master với các đối tượng khác

➢ ***Product Owner:*** chịu trách nhiệm tối đa hoá giá trị của thành phẩm từ kết quả làm việc của Scrum Team. Product Owner chịu trách nhiệm quản lý Product Backlog một cách hiệu quả, bao gồm:

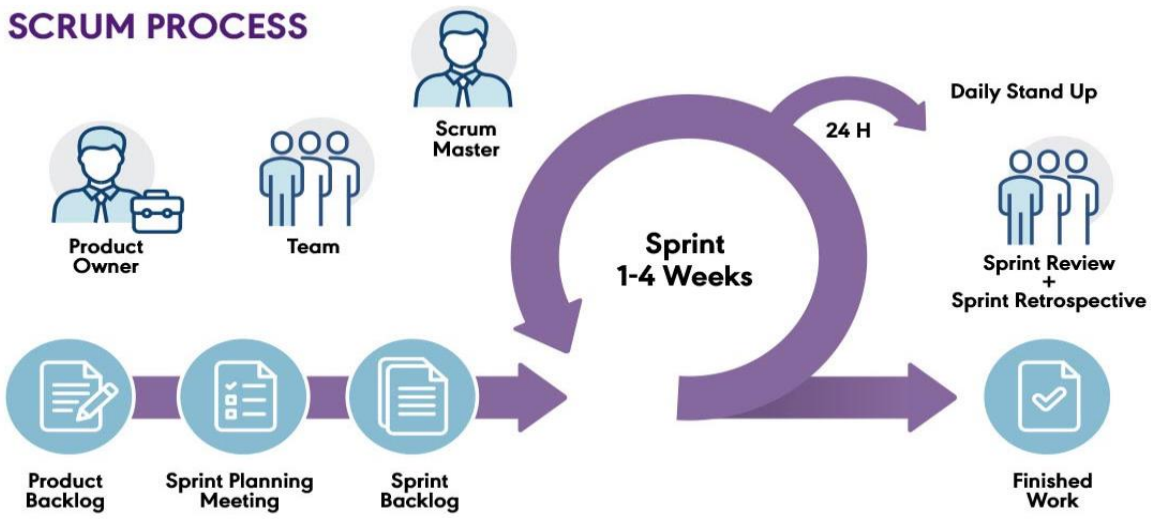
- Phát triển và truyền đạt rõ ràng Product Goal;
- Tạo ra và truyền đạt rõ các hạng mục trong Product Backlog;
- Sắp thứ tự các hạng mục trong Product Backlog;
- Bảo đảm Product Backlog được minh bạch, rõ ràng và dễ hiểu.

- ***Developers (lập trình viên)*:** Bất cứ hoạt động, quy trình nào làm ảnh hưởng không tốt đến kết quả đều cần được điều chỉnh sớm nhất có thể. Developers luôn luôn chịu trách nhiệm:
  - Tạo ra kế hoạch cho Sprint, gọi là Sprint Backlog;
  - Nâng cao chất lượng bằng việc tuân thủ quy định về sự hoàn thành công việc hay nhiệm vụ;
  - Thay đổi để thích ứng kế hoạch hằng ngày nhằm hoàn tất Sprint Goal;
  - Chịu trách nhiệm với nhau như những người chuyên nghiệp.

3. Nguyên tắc phối hợp trong Scrum

- ***Tôn trọng***: để có thể cộng tác tốt , các thành viên cần nhất là sự tôn trọng lẫn nhau, tôn trọng scrum master và quy trình Scrum.
- ***Tập trung***: Team cần tập trung vào những công việc đã được chọn để có thể tạo ra được kết quả như mong đợi ở cuối mỗi sprint
- ***Cởi mở***: Mọi thứ sẽ không bao giờ diễn ra hoàn hảo như kế hoạch. Các thành viên cần giữ được tinh thần cởi mở với những ý tưởng và cơ hội mới, điều này có thể vừa giúp họ phát triển bản thân, vừa có thể giúp cải thiện sản phẩm hoặc quy trình của dự án.
- **Can đảm:** dám tự quyết định các công việc dự kiến thực hiện trong 1 khoảng thời gian nhất định, và mỗi cá nhân tự chịu trách nhiệm cao
- **Cam kết:** Cam kết hoàn thành công việc theo thời gian được giao, có tính trách nhiệm cao

3.4.3 Quy trình của Scrum



Hình 3-5: Vai trò của Scrum Master với các đối tượng khác

(Nguồn: The Agile Journey: A Scrum overview; Pm-parner, 2023)

a. Các sự kiện trong Scrum

- ***SPRINT:*** Thông thường, một sprint sẽ kéo dài hai tuần, có thể ngắn hơn hoặc dài hơn tùy vào bối cảnh cụ thể của team. Để bắt đầu một sprint, leader sẽ cần

xác định công việc cần phải hoàn thành (sprint backlog). Để có một sprint tốt nhất trong khả năng, bạn cần đảm bảo tất cả các task tồn đọng được ghi lại một cách cụ thể ở một nơi. Cân nhắc sử dụng các công cụ quản lý dự án như Asana hay Trello để sắp xếp tất cả thông tin này.

- ***SPRINT PLANNING:*** Trước khi bắt đầu một sprint, cần biết những nhiệm vụ cần tập trung trong sprint và tại sao lại cần tập trung trong sprint này. Sự kiện này sẽ cần xác định các mục tiêu của sprint và truyền đạt cho nhóm lý do tại sao. Từ đó, sẽ xác định sprint backlog (nhiệm vụ tồn đọng) sẽ cần được giải quyết trong sprint này và cách hoàn thành công việc.
- ***DAILY STANDUP:*** Họp nhanh mỗi ngày. Lên kế hoạch để các thành viên gặp nhau 15 phút mỗi ngày. Đây là cơ hội để xem xét lại về những gì đội ngũ đang làm và xác định được các cản trở có thể gặp phải.
- ***SPRINT REVIEW:*** Mỗi khi kết thúc một sprint, nhóm scrum nên cùng nhau xem xét lại sprint. Trong buổi "sprint review" này, các thành viên sẽ trình bày những công việc đã được "Hoàn thành" để các bên liên quan phê duyệt hoặc kiểm tra.
- ***SPRINT RETROSPECTIVE:*** Sau khi kết thúc một sprint, sự kiện này dùng để thảo luận về diễn tiến của sprint đó và những gì có thể được cải thiện trong tương lai. Tôn chỉ của Scrum là hoạt động cải tiến liên tục, vì vậy có thể trao đổi, nhìn nhận để có các cải tiến nếu cần.

b. Đặc điểm trong quy trình Scrum

- Linh hoạt theo tình hình thực tế.
- Tự tổ chức: mặc dù tuân theo vai trò và quy tắc, mọi thành viên đều được trao quyền để làm chủ các nhiệm vụ và công việc. Việc này cho phép nhóm sáng tạo và năng động hơn
- Cộng tác: Các thành viên sẽ tạo ra giá trị lớn nhất khi luôn đồng hành cùng nhau dù là trước, trong hay sau sprint.
- Sắp xếp ưu tiên dựa trên giá trị: mục tiêu của Scrum là tạo ra nhiều giá trị nhất cho doanh nghiệp. Để làm được điều đó, chúng ta phải sắp xếp ưu tiên công việc ngay từ khi bắt đầu quá trình Scrum.

- Có giới hạn về thời gian (Timeboxing): Quy trình Scrum có nhiều hoạt động chẳng hạn như sprint, daily standup …. Thế nên mỗi việc chỉ nên làm trong một thời gian nhất định, hết thời hạn đó thì cần chuyển sang nhiệm vụ tiếp theo. Trong tương lai luôn có thể quay lại để cải thiện kết quả công việc.
- Liên tục cải tiến và phát triển. Sản phẩm đầu tiên sẽ không cần hoàn hảo. Nhưng bằng cách liên tục xây dựng và cải tiến, team sẽ có thể bắt kịp nhu cầu của khách hàng, điều chỉnh sản phẩm và kết quả cuối cùng để tạo ra giá trị cao nhất.

Chương 4 QUẢN LÝ DỰ ÁN PHẦN MỀM

Nội dung:

- Tổng quan về quản lý dự án
- Các quy trình quản lý dự án phần mềm
- Lập kế hoạch dự án phần mềm
- Quản lý rủi ro dự án phần mềm

4.1 Tổng quan về quản lý dự án phần mềm

4.1.1 Các định nghĩa

4.1.1.1 Dự án

Theo PMBoK Guide, dự án là một nỗ lực có thời hạn nhằm tạo ra một sản phẩm, dịch vụ hay kết quả duy nhất.

Theo định nghĩa này, dự án có 2 đặc tính chính:

- Tính thời hạn: một dự án phải có thời điểm bắt đầu, thời điểm kết thúc rõ ràng và vòng đời với các bước phân biệt.Ví dụ, 1 dự án có thể kết thúc khi đã đạt mục tiêu, hoặc khi khách hàng không muốn dự án hoàn thành nữa.

Các dự án là có thời hạn, nhưng sản phẩm của chúng có thể tồn tại sau khi dự án kết thúc.

- Tính duy nhất: Các dự án được thực hiện để hoàn thành các mục tiêu bằng cách tạo ra các sản phẩm bàn giao. Tính duy nhất thể hiện qua sản phẩm bàn giao này. Có 2 cách thể hiện.

Một là, sản phẩm bàn giao phải mang lại yếu tố mới cho đội ngũ thực hiện. Yếu tố mới ở đây có thể thể hiện qua nhiều khía cạnh như sản phẩm hoặc môi trường thực hiện dự án là duy nhất, sử dụng các hoạt động mới, quy trình mới trong quá trình thực hiện.

Hai là, sự khác biệt về kết quả của dự án phải nhận biết được. Ví dụ, một sản phẩm mới khác hẳn sản phẩm khác, hoặc sự kết hợp của nhiều sản phẩm hiện hành thành một sản phẩm có tính năng mới mà các sản phẩm hiện hành không có.

Dự án là một trong các phương thức chủ yếu tạo ra giá trị và lợi ích trong các tổ chức.

Tuy nhiên, quá trình thực hiện dự án luôn xuất hiện các rủi ro hay các yếu tố không chắc chắn làm cho kết quả dự án có thể khác với dự đoán ban đầu.

Do đó, dự án cần được quản lý với giả định sẽ xảy ra thay đổi.

4.1.1.2 Dự án phần mềm

Dự án phần mềm là một chuỗi các hoạt động độc đáo, phức tạp, có tính kết nối, có chung mục đích là phải được hoàn thành trước một thời gian cụ thể, trong phạm vi ngân sách và theo đúng đặc tả kỹ thuật [3].

Dự án phần mềm có 1 số đặc trưng sau đây:

- Do đội ngũ thành viên gồm ít nhất 2 người thực hiện
- Giới hạn về thời gian, ngân sách, và nhân lực
- Sản phẩm là phần mềm mới hoặc phần mềm có sẵn được cải tiến
- Sản phẩm phải góp phần tạo dựng quy trình nghiệp vụ mới, hữu ích, hoặc mang lại lợi ích đáng kể cho quy trình nghiệp vụ hiện có.

4.1.1.3 Quản lý dự án

Quản lý dự án là áp dụng kiến thức, kỹ năng, công cụ và kỹ thuật vào các hoạt động của dự án nhằm đáp ứng yêu cầu của dự án.

- Đạt mục tiêu dự án
- Đạt hoặc vượt các yêu cầu hay kỳ vọng của những bên liên quan (stakeholders)
- Cân bằng giữa các yếu tố: thời gian, chi phí, chất lượng sản phẩm, nguồn nhân lực, v.v.

4.1.2 Các yếu tố ảnh hưởng đến thành công của quản lý dự án

4.1.2.1 Các mức độ thành công của dự án

Một dự án luốn được thực hiện với kỳ vọng là dự án thành công. Thước đo thành công của dự án phụ thuộc vào từng quan điểm. Có 4 mức độ thành công tương ứng với 4 quan điểm thành công (xem Hình 4-1).

Mức 1: Quan điểm quản lý dự án: thành công về chất lượng, chi phí, bàn giao sản phẩm đúng phạm vi đã thỏa thuận.

Quản lý dự án thành công chỉ tính đến hiệu quả nội bộ của việc quản lý dự án bằng cách chủ yếu tập trung vào việc hiện thực hóa thành công các mục tiêu về chi phí, thời gian và hiệu suất của quy trình và nhiệm vụ dự án.

Như vậy, sẽ có những dự án được quản lý thành công một phần, ví dụ:

- Dự án hoàn thành hầu hết các mục tiêu, nhưng lại không đáp ứng được các yêu cầu về thời gian và ngân sách.
- Dự án hoàn thành các mục tiêu đúng thời hạn và/hoặc trong phạm vi ngân sách, nhưng chất lượng lại không như mong đợi.
- Dự án được bàn giao dự án đúng thời hạn, trong ngân sách và chất lượng như mong đợi, nhưng lại không đáp ứng hết các mục tiêu đề ra.

Trong các trường hợp này, chỉ có một hoặc một số quy trình quản lý dự án đã được thực hiện thành công. Việc xem xét xác định thành công của các quy trình được sử dụng cho phép phân tích đánh giá sự phù hợp của chúng với mục đích của dự án cũng như tính tích hợp và hiệu quả của chúng trong việc đóng góp vào kết quả của dự án, từ đó cung cấp phản hồi cho nhóm dự án và tổ chức để học hỏi và cải tiến cho các dự án tiếp theo.

Mức 2: Quan điểm dự án: ghi nhận giá trị của các sản phẩm bàn giao của dự án.

Các tiêu chi mà theo đó sản phẩm bàn giao của dự án được coi là thành công phải được đo lường, đánh giá sau khi sản phẩm của dự án được triển khai, và trong một khoảng thời gian xác định.

Sản phẩm bàn giao của các dự án khác nhau có thể khác nhau, ví dụ như hệ thống thông tin, phần mềm hoặc các dịch vụ phần mềm. Các tiêu chi đo lường bao gồm các tiêu chí liên quan đến chính sản phẩm bàn giao (chẳng hạn như sự phù hợp của nó với thông số kỹ thuật, yêu cầu và mong đợi về chất lượng) và sự hài lòng của khách hàng/người dùng (chẳng hạn như sự chấp nhận, sử dụng và hiệu quả của sản phẩm).

Thành công trong dự án liên quan đến kết quả dài hạn và hướng tới khách hàng hơn. Tại sao lại như vậy? Khách hàng/người dùng có những mối quan tâm và kỳ vọng khác nhau về một dự án. Họ quan tâm đến tính dùng được của sản phẩm, và các lợi ích khác gắn liền với những cải thiện về tính chất và điều kiện làm việc của họ. Một dự án có

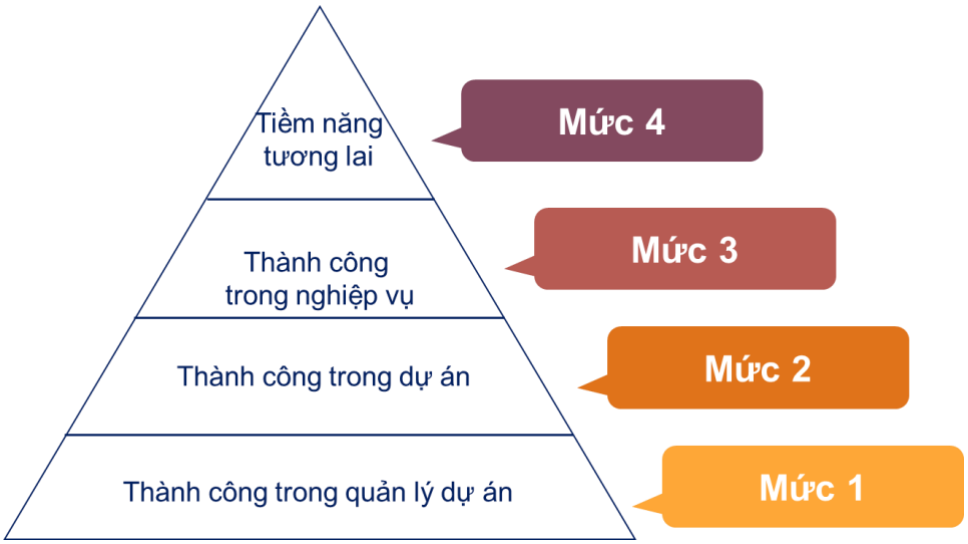
thể thành công trong việc cung cấp một phần mềm "đúng thời gian, trong phạm vi ngân sách và đầy đủ đặc điểm kỹ thuật" nhưng không được người dùng nghiệm thu. Điều này hoàn toàn có thể xảy ra, ví dụ như khi yêu cầu người dùng được đặc tả không đầy đủ hoặc yêu cầu của người dùng thay đổi do bối cảnh nghiệp vụ thay đổi.

Mức 3: Quan điểm tổ chức thực hiện dự án: Thành công trong kinh doanh/nghiệp vụ nhờ kết quả của dự án.

Điều này có nghĩa là dự án cung cấp một sản phẩm mang lại giá trị ròng đáng kể cho tổ chức đầu tư vào dự án hoặc tổ chức, cá nhân thụ hưởng trên các khía cạnh kinh tế, xã hội, đời sống, v.v. Nó cũng có thể bao gồm việc đánh giá sự đóng góp của tổ chức vào kết quả của dự án, trong trường hợp có nhiều tổ chức phối hợp thực hiện dự án. Do đó, các thước đo thường bao gồm mức độ mà dự án đáp ứng các mục tiêu và mục đích thúc đẩy việc phê duyệt đầu tư (thường được nêu rõ trong kế hoạch kinh doanh) và liệu các lợi ích mong đợi có được hiện thực hóa hay không. Chúng cũng có thể bao gồm việc xem xét tính hiệu quả và đóng góp của quản trị doanh nghiệp cho dự án. Cuối cùng, việc đánh giá lợi ích ròng cũng sẽ bao gồm mọi lợi ích ngoài ý muốn và tác động tiêu cực phát sinh từ khoản đầu tư.

Mức 4: Quan điểm của các bên liên quan: Dự án tạo ra tiềm năng tương lai cho các bên liên quan.

Ở mức này, lợi ích của tổ chức được đánh giá bởi các bên liên quan bên ngoài như nhà đầu tư, đối thủ cạnh tranh, nhà phân tích ngành hoặc cơ quan quản lý chứ không phải là người trong nội bộ tổ chức đó. Thành công ở cấp độ này xuất phát từ những cải tiến ròng về vị thế của ngành, tăng trưởng và phát triển kinh doanh, lợi thế cạnh tranh và/hoặc lợi ích chiến lược khác.



Hình 4-1: Các mức độ thành công của dự án

4.1.2.2 Các ràng buộc về chất lượng dự án

Như minh họa trong Hình 4-2, các ràng buộc của dự án được thể hiện trên các khía cạnh phạm vi, tiến độ, chi phí, nguồn lực, chất lượng và rủi ro. Tầm quan trọng của mỗi ràng buộc là khác nhau đối với từng dự án và người quản lý dự án điều chỉnh cách tiếp cận để quản lý những hạn chế này dựa trên môi trường dự án, tổ chức văn hóa, nhu cầu của các bên liên quan và các biến số khác.

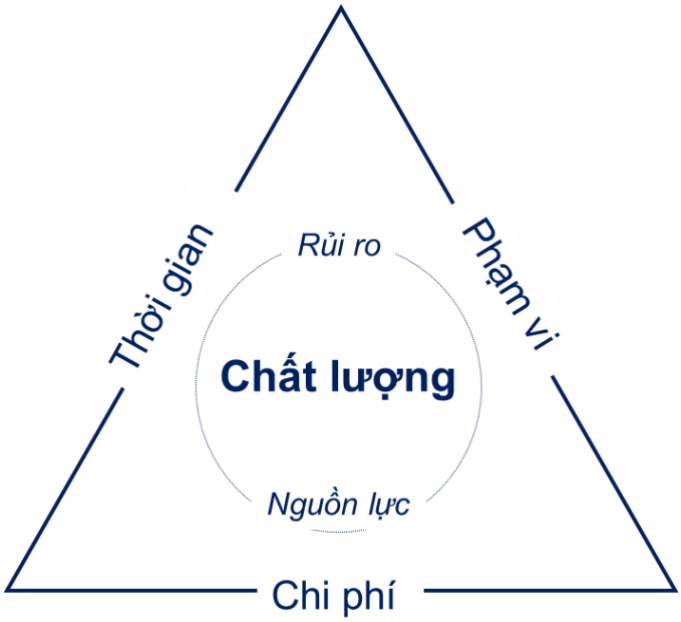
Chi phí: Những hạn chế tài chính của dự án, còn được gọi là ngân sách dự án

Phạm vi: Các nhiệm vụ cần thiết để hoàn thành mục tiêu của dự án

Thời gian: Tiến độ dự án, dựa vào đó để hoàn thành dự án

Rủi ro: Các tình huống, sự cố bất ngờ xảy ra, ảnh hưởng đến dự án. Trong khi hầu hết các rủi ro của dự án là tiêu cực, một số có thể là tích cực.

Nguồn lực: nhân lực, vật lực và các tài nguyên có thể huy động thực hiện dự án



Hình 4-2: Các ràng buộc về chất lượng dự án

Người quản lý dự án có thể tăng hoặc giảm chi phí, thời gian và phạm vi của dự án với sự đánh đổi để giữ cho dự án đúng tiến độ và phù hợp với ngân sách. Ví dụ.

- Thời gian và phạm vi: nếu dự án chậm tiến độ, có thể giảm phạm vi dự án, từ đó giảm thời gian thực hiện dự án. Ngược lại, nếu các bên liên quan của dự án đưa ra các hoạt động bổ sung cho dự án (mở rộng phạm vi), cần tăng thời lượng thực hiện ự án.
- Chi phí và phạm vi: Phạm vi dự án giảm đồng nghĩa với việc thực hiện ít nhiệm vụ hơn, cũng đồng nghĩa với việc giảm chi phí. Ngược lại, phạm vi dự án lớn hơn dẫn tới chi phí cao hơn.
- Chi phí và thời gian: Trong một số dự án, thời gian và chi phí có thể liên quan trực tiếp. Ví dụ: chi phí thuê thiết bị hoặc nhân công tỷ lệ thuận với thời gian sử dụng chúng.
- Nguồn lực và chi phí: Nguồn lực gắn chặt với những hạn chế về chi phí trong dự án của bạn vì những yêu cầu này của dự án sẽ tốn kém chi phí. Nếu không phân bổ nguồn lực hợp lý, chất lượng dự án có thể thấp hơn, ngân sách tăng và chậm trễ về thời gian.

Người quản lý dự án cũng cần xác định, đánh giá và chuẩn bị cho những rủi ro tiềm ẩn của dự án. Với một kế hoạch quản lý rủi ro tốt, người quản lý dự án có thể ngăn chặn

những rủi ro dự án gây thiệt hại nặng nề nhất và chuẩn bị cho mọi rủi ro bất ngờ có thể xảy ra.

4.1.2.3 Các thành phần của quản lý dự án

Khi nói đến quản lý dự án, không thể không đề cập đến bốn yếu tố cực kỳ quan trọng, ảnh hưởng đến sự thành công của quản lý dự án. Trong tiếng Anh, 4 yếu tố này đều có chữ cái đầu tiên là P, nên thường được gọi là 4Ps of project management.

Con người (People)

Con người là yếu tố quan trọng nhất, ảnh hưởng đến thành công của dự án. Đây hầu như là bất kỳ ai có trách nhiệm, quyền lợi, nghĩa vụ liên quan đến dự án, gọi là bên liên quan hay stakeholder.

Các bên liên quan

Các bên liên quan thường được phân thành 2 nhóm chính: các bên liên quan bên trong đơn vị thực hiện dự án, và các bên liên quan bên ngoài đơn vị thực hiện dự án.

Bên trong đơn vị thực hiện dự án, có thể kể tới: người quản lý dự án, chủ đầu tư, dội ngũ quản lý dự án, ví dụ như quản lý tài nguyên, các thành viên dự án, các cá nhân, bộ phận khác trong đơn vị thực hiện như nhà tài trợ dự án, văn phòng dự án, bộ phận tài chính kế toán, …

Các đối tác kinh doanh cũng được gom vào nhóm bên liên quan trong đơn vị thực hiện dự án.

Các bên liên quan bên ngoài đơn vị thực hiện dự án bao gồm: khách hàng, người dùng cuối, các nhà cung cấp, các đối thủ cạnh tranh, và các cơ quan quản lý như cơ quan thuế, bảo hiểm xã hội. Với dự án phần mềm, cần phân biệt rõ khách hàng (Customer), là người đưa ra các yêu cầu cho phần mềm và người sử dụng cuối cùng (End-user), là người tương tác với phần mềm khi nó được phát hành.

Ví dụ về một nhóm phát triển phần mềm điển hình:

- Người quản lý dự án: bao quát việc phát triển, tổ chức và phân phối sản phẩm
- Đội ngũ quản lý dự án:

  - Người quản lý kỹ thuật / Giám đốc kỹ thuật: hướng dẫn nhóm trong quá trình phát triển, tập trung vào khía cạnh kỹ thuật của sản phẩm
  - Người quản lý nhóm / Trưởng nhóm: quản lý nhóm, điều khiển nhóm đi đúng hướng
  - Người quản lý sản phẩm: chịu trách nhiệm quyết định tính năng nào cần có cho sản phẩm.
  - Nếu sản phẩm được phát triển theo một quy trình đặc thù, ví dụ scrum chẳng hạn, thì còn cần đến chủ sở hữu sản phẩm: dẫn dắt việc phát triển sản phẩm để đáp ứng yêu cầu của khách hàng
- Các thành viên nhóm tiêu biểu bao gồm:
  - Người phân tích nghiệp vụ: phân tích nhu cầu của khách hàng và phối hợp với thành viên nhóm để đưa ra phương án giải quyết phù hợp.
  - Kiến trúc sư phần mềm: thiết kế giải pháp tổng thể và đưa ra các tiêu chuẩn kỹ thuật cho các thiết kế chi tiết, đảm bảo phần mềm được cài đặt theo đúng thiết kế
  - Nhà thiết kế UX/UI: thiết kế các tính năng để nâng cao trải nghiệm và tương tác của khách hàng
  - Nhà phát triển phần mềm: thiết kế và phát triển phần mềm phù hợp với nhu cầu của người dùng
  - Người kiểm thử phần mềm: thực hiện các thử nghiệm để đảm bảo các giải pháp phù hợp với nhu cầu về hiệu suất và bảo mật
  - Người quản lý chất lượng: xác nhận giải pháp được phát triển phù hợp với thông số kỹ thuật

Người quản lý dự án

Người quản lý dự án là những chuyên gia có tổ chức, có định hướng mục tiêu, sử dụng niềm đam mê, tính sáng tạo và sự hợp tác để dẫn dắt dự án hướng tới thành công. Người quản lý dự án, cùng với nhóm quản lý dự án, chỉ đạo việc thực hiện các hoạt động dự án đã lên kế hoạch, và là tâm điểm giao tiếp, kết nối về mặt tổ chức và kỹ thuật trong dự án.

Để trở thành người quản lý dự án tốt, cần rèn luyện kỹ năng, kinh nghiệm trong các lĩnh vực sau:

- Lãnh đạo và giao tiếp hiệu quả—người quản lý dự án phải lãnh đạo và giao tiếp hiệu quả với nhóm của họ cũng như các bên liên quan trong toàn bộ vòng đời của dự án.
- Tổ chức và quản lý thời gian - người quản lý dự án phải xử lý việc tổ chức và phân công nhiệm vụ. Họ cũng phải đảm bảo rằng tất cả tài liệu và sản phẩm bàn giao của dự án đều được hoàn thành đúng thời hạn.
- Giải quyết vấn đề sáng tạo và khả năng thích ứng—người quản lý dự án phải hiểu cách giải quyết vấn đề và điều chỉnh dự án của mình một cách sáng tạo để tránh rủi ro và thua lỗ.
- Động lực và quản lý nhóm—người quản lý dự án phải đảm bảo các bên liên quan và thành viên trong nhóm luôn có động lực trong suốt vòng đời của dự án. Hơn nữa, họ phải có khả năng quản lý nhóm của mình để đảm bảo kết quả chất lượng hàng đầu và hoàn thành các sản phẩm dự án đúng thời hạn.

Người quản lý dự án cũng phải biết chu kỳ sống của dự án, các tiến trình của dự án và vai trò của các tiến trình này trong việc thực hiện các công việc ở các pha khác nhau trong chu kỳ sống của dự án. Họ cũng phải nhận biết được sự phức tạp của môi trường thực hiện dự án, và phải được chuẩn bị để đối phó với các mối xung đột khác nhau.

Trong suốt vòng đời dự án, người quản lý dự án chịu trách nhiệm dẫn dắt dự án đến thành công, và thực hiện các công việc như:

Trong suốt vòng đời dự án, một số trách nhiệm cốt lõi của người quản lý dự án bao gồm:

- Xác định mục tiêu, nhu cầu và phạm vi của dự án
- Lập kế hoạch, giám sát dự án
- Giữ đúng lịch trình thực hiện dự án, đảm bảo tất cả các nhiệm vụ, sản phẩm bàn giao và tài liệu dự án được giao kịp thời
- Lập kế hoạch chi phí của dự án và bám sát ngân sách
- Quản lý tất cả các nguồn lực cần thiết (nhân lực, máy móc thiết bị, vật tư…) để thực hiện dự án

- Thúc đẩy giao tiếp hiệu quả với các bên liên quan về tình trạng dự án
- Dự báo, đánh giá và loại bỏ một cách chiến lược các yếu tố cản trở và rủi ro tiềm ẩn
- Xử lý sự cố
- Lập tài liệu về tiến độ của dự án và ghi chép các nhiệm vụ trong suốt dự án bằng các công cụ quản lý dự án khác nhau
- Đảm bảo kết quả chất lượng cao và thành công cho dự án

Sản phẩm (Product)

Yếu tố thứ 2 cần xét đến là sản phẩm của dự án.

Người quản lý dự án phải xác định phạm vi để đảm bảo kết quả thành công, kiểm soát "sự leo thang phạm vi"; cũng như những trở ngại kỹ thuật mà dự án có thể gặp phải.

Có 2 loại phạm vi là phạm vi sản phẩm và phạm vi dự án.

Phạm vi sản phẩm

Phạm vi sản phẩm trả lời câu hỏi: phải tạo ra cái gì. Ví dụ: xây dựng web site bán hàng, có các chức năng tìm kiếm sản phẩm, giới thiệu sản phẩm và quản lý người dùng.

Để có thể định nghĩa tốt phạm vi sản phẩm, cần làm rõ 3 loại thông tin chính:

- Hoàn cảnh: Phần mềm được xây dựng phù hợp với bối cảnh hệ thống, sản phẩm hoặc nghiệp vụ lớn hơn nào? Các ràng buộc nào của hoàn cảnh cần được xem xét?
- Mục tiêu thông tin: Khách hàng thấy được sản phẩm đầu ra của phần mềm là gì? Dữ liêu yêu cầu của đầu vào là gì?
- Chức năng và hiệu suất: Chức năng nào của phần mềm thực hiện biến đổi dữ liệu đầu vào thành đầu ra? Đặc trưng về hiệu suất đặc biệt nào được giải quyết?

Để có phạm vi sản phẩm tốt, cần thực hiện phân tách vấn đề. Khi phạm vi sản phẩm được xác định, nó được phân hoạch thành:

- Các chức năng cấu thành
- Các đối tượng dữ liệu người dùng có thể thấy

- Tập các lớp vấn đề

Quá trình phân hoạch tiếp tục cho đến khi tất cả các chức năng hoặc các lớp vấn đề đã được xác định

Phạm vi dự án:

Phạm vi dự án trả lời câu hỏi: phải làm những gì. Ví dụ: lập kế hoạch quản lý dự án. Quản lý chất lượng. Viết báo cáo tổng hợp.

Phạm vi dự án phải rõ ràng và dễ hiểu tại mỗi mức quản lý và chuyên môn

Các yếu tố kỹ thuật, công nghệ

Ngoài ra, các yếu tố kỹ thuật, công nghệ có thể đặt ra những hạn chế đối với các dự án phần mềm và sản phẩm phần mềm bao gồm:

- Hiện trạng công nghệ phần cứng và phần mềm;
- Nền tảng phần cứng, nền tảng phần mềm, hệ điều hành và giao thức truyền thông;
- Tính toàn vẹn, các hạn chế và giao thức của kiến trúc CNTT; Kiến trúc phần mềm;
- Công cụ phát triển phần mềm;
    - Nguồn gốc các thành phần cấu thành
    - Tái sử dụng các thành phần phần mềm từ thư viện;
    - Sử dụng các thành phần phần mềm nguồn mở hay nguồn đóng;
    - Sử dụng các thành phần phần mềm do khách hàng cung cấp;
- Yêu cầu tương thích xuôi và ngược: giữa các thành phần cấu thành phần mềm;
- Giao diện với phần cứng và phần mềm khác;
- Sáng tạo và sử dụng tài sản trí tuệ.

Quy trình (Process)

Yếu tố thứ 3 cần xem xét là quy trình. Quy trình là tập hợp các hoạt động kiểu mẫu và nhiệm vụ trong lĩnh vực công nghệ phần mềm để hoàn tất công việc. Người quản lý dự án và các thành viên trong nhóm cần có phương pháp, kế hoạch để xây dựng quy trình khung. Quy trình khung trả lời các câu hỏi:

- Ai: Ai chịu trách nhiệm? Họ nằm ở đâu về mặt tổ chức?
- Làm gì: Tại sao hệ thống được phát triển? Những gì sẽ được thực hiện?
- Khi nào: Khi nào nó sẽ được thực hiện?
- Bằng cách nào: Công việc sẽ được thực hiện như thế nào về mặt kỹ thuật và quản lý?
- Bằng cái gì: Mỗi nguồn lực (ví dụ: con người, phần mềm, công cụ, cơ sở dữ liệu) sẽ cần bao nhiêu?
- Trong bao lâu: Thời hạn cho phép để hoàn thành toàn bộ dự án.
- Nộp cái gì: kết quả là gì, xách định thế nào.

Nếu quy trình khung không được xác định rõ ràng, các thành viên trong nhóm dự án sẽ không biết phải làm gì và khi nào tiến hành các hoạt động dự án.

Sử dụng đúng quy trình sẽ tăng tỷ lệ thực hiện thành công dự án, đáp ứng được các mục đích, mục tiêu ban đầu của nó.

Dự án (Project)

Yếu tố cuối cùng của quản lý dự án là dự án, thể hiện tất cả các công việc để sản phẩm trở thành hiện thực. Đây là lúc vai trò và trách nhiệm của người quản lý dự án phát huy tác dụng. Người đó phải hướng dẫn các thành viên trong nhóm đạt được mục tiêu và mục tiêu của dự án. Người quản lý dự án phải giao nhiệm vụ, giúp đỡ các thành viên trong nhóm khi cần thiết và cuối cùng cố gắng hoàn thành tất cả các yêu cầu đặt ra trong phạm vi dự án.

Thực tế, các dự án phần mềm gặp rắc rối trong 1 số trường hợp sau:

- Người làm phần mềm không hiểu nhu cầu của khách hàng
- Phạm vi sản phẩm được xác định kém
- Những thay đổi được quản lý kém
- Công nghệ đã chọn thay đổi
- Nhu cầu nghiệp vụ thay đổi (hoặc không được xác định rõ ràng).
- Thời hạn là không thực tế
- Người dùng không hài lòng
- Tài trợ bị mất (hoặc không bao giờ có được một cách hợp lý)

- Nhóm dự án thiếu người có kỹ năng phù hợp
- Các nhà quản lý và những người thực hiện né tránh những phương pháp hay nhất và bài học kinh nghiệm.

4.2 Quy trình quản lý dự án phần mềm

4.2.1 Giải quyết bài toán quản lý dự án

4.2.1.1 Bài toán quản lý dự án

Theo các định nghĩa về dự án và quản lý dự án đã đề cập ở trên, có thể coi dự án là tập hợp của nhiều bài toán, và quản lý dự án cũng là việc tìm lời giải cho các bài toán đó. Trong lĩnh vực công nghệ thông tin, một bài toán có thể được giải quyết theo mô hình xử lý thông tin IPO (Input – Process – Output): cho một đầu vào, qua 1 tập các bước xử lý sẽ nhận được đầu ra. Như vậy, một bài toán dự án có thể được mô tả như sau: cho một tập nguồn lực của dự án (đầu vào), một tập các mục đích / mục tiêu (đầu ra), qua các bước xử lý (kế hoạch), đầu vào sẽ được biến đổi thành đầu ra.

Đầu vào

Đầu vào ở đây là tập nguồn lực của dự án. Ví dụ, nhân sự làm việc toàn thời gian, bán thời gian, phần cứng, phần mềm, không gian văn phòng, tiền mặt, công nghệ, thông tin v.v.

Các bước xử lý:

Ta cần dùng các nguồn lực đó để thực hiện các hoạt động trong dự án theo các quy trình nhất định nhằm thực hiện mục tiêu dự án. Các quy trình ở đây thể hiện cách thức quản lý, tức là cách kết hợp, sử dụng và xử lý các hoạt động theo nguồn lực với các ràng buộc, điều kiện cụ thể. Cách thức quản lý khác nhau dẫn tới phương pháp, lịch trình, thứ tự thực hiện các hoạt động cũng khác nhau, đảm bảo sử dụng cân bằng các nguồn lực và chuyển đổi suôn sẻ đầu vào thành đầu ra.

Đầu ra:

Sau khi thực hiện các bước xử lý, ta sẽ thu được đầu ra tương ứng với kết quả của các hoạt động của dự án. Nói cách khác, đầu ra của dự án chính là các sản phẩm cần bàn

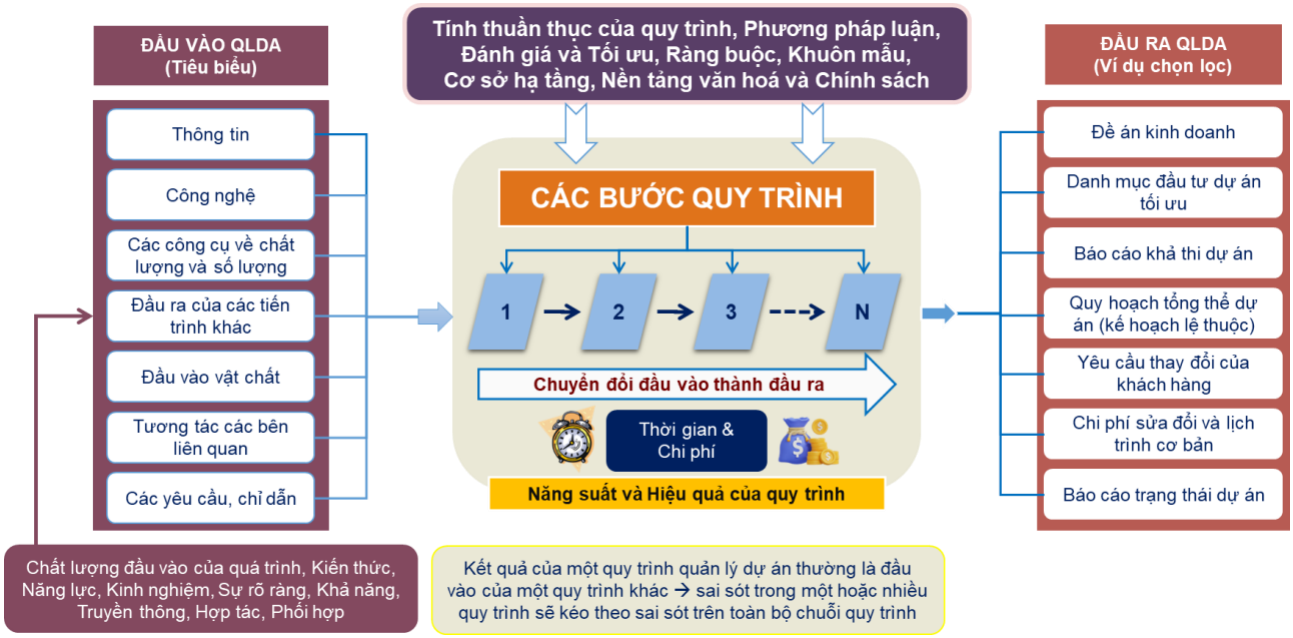
giao. Các sản phẩm này có thể hữu hình hoặc vô hình. Ví dụ, đầu ra có thể là một sản phẩm hoặc dịch vụ phần mềm mới, một bộ tài liệu quy chuẩn mới thay cho bộ tài liệu quy chuẩn cũ, hoặc nhân viên đã được đào tạo nâng cao trình độ sau quá trình thực hiện dự án.

Vấn đề ở đây là kết quả thu được có thể khác biệt với mục tiêu / mục đích hay kế hoạch ban đầu của dự án. Nguyên nhân của vấn đề có thể bắt nguồn từ các yếu tố, đầu vào, quy trình và điều kiện ràng buộc trong quá khứ. Nếu đầu vào sai, dù quy trình thực hiện đúng thì kết quả vẫn sai. Nếu quy trình kém hiệu quả, dù có đầu vào đúng thì kết quả vẫn sai. Các ràng buộc / điều kiện là yếu tố không chắc chắn, có thể xảy ra và thay đổi bất cứ lúc nào.

Đầu vào và quy trình có thể kiểm soát được, nhưng không thể kiểm soát các điều kiện ràng buộc một cách dễ dàng. Đặc biệt, với những điều kiện ràng buộc tuyệt đối như bão, động đất và suy thoái kinh tế, điều duy nhất có thể làm là chỉ nhận kết quả từ nó. Một số điều kiện ràng buộc tương đối như luật pháp, ngân sách, số lượng nhân viên, thời hạn, v.v. có thể bị tác động đến nó bằng một số cách như đàm phán, v.v.

4.2.1.2 Lời giải của bài toán quản lý dự án

Như vậy để giải quyết bài toán quản lý dự án, và đảm bảo là đầu ra khớp với các mục tiêu, mục đích, kế hoạch, ta cần phân tích chi tiết hơn các thành tố trong mô hình IPO (xem Hình 4-3). Theo đó, đầu vào của các quy trình quản lý dự án rất đa dạng, và chất lượng đầu vào phụ thuộc rất nhiều vào người quản lý dự án và các bên liên quan. Quản lý dự án sử dụng các quy trình để tạo ra "sản phẩm giao nộp". Một số quy trình quản lý dự án khá phức tạp và có nguy cơ lỗi cao. Năng suất và hiệu quả của các quy trình còn phụ thuộc vào sự thuần thục của quy trình, các ràng buộc dự án, nguồn lực và cách thức quản lý. Bên cạnh sản phẩm bàn gioa cho khách hàng, đầu ra của quản lý dự án còn là các tài liệu dự án bàn giao cho tổ chức thực hiện dự án.



Hình 4-3: Bài toán quản lý dự án

4.2.2 Các nhóm quy trình quản lý dự án

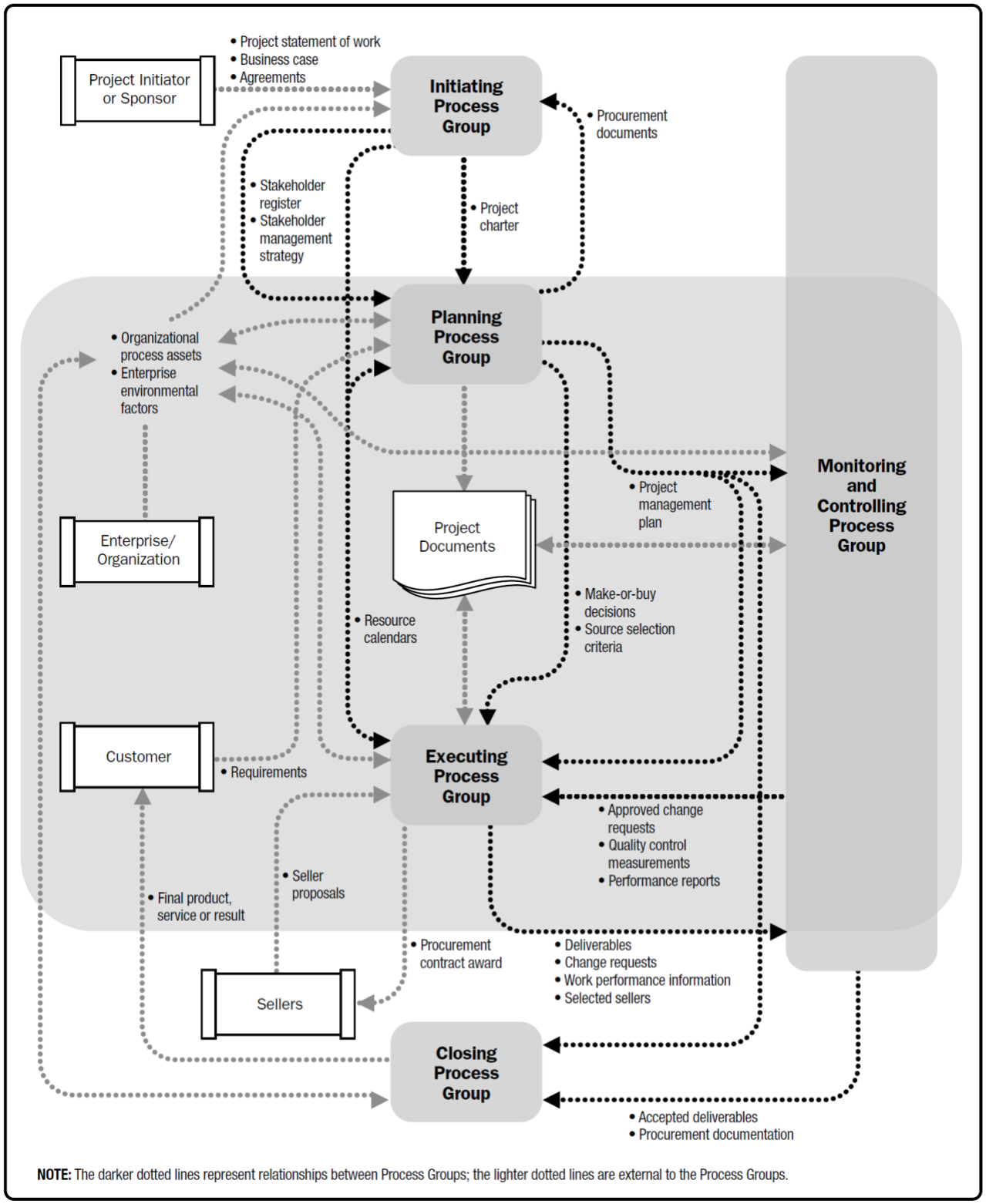
Quản lý dự án được thực thi theo rất nhiều quy trình. PMBoK nhóm các quy trình thành 5 nhóm chính: khởi tạo, lập kế hoạch, thực thi, giám sát và kiểm soát và kết thúc.

Lưu ý: Nhóm quy trình không phải là các pha của dự án. Nếu việc thực thi dự án phức tạp, cần được chia thành các pha, các quy trình trong Nhóm quy trình sẽ tương tác với nhau trong từng pha.

Vì các dự án được chia thành các pha riêng biệt, chẳng hạn như phát triển ý tưởng, nghiên cứu khả thi, thiết kế, tạo mẫu thử, cài đặt, kiểm thử, v.v., các quy trình trong mỗi Nhóm quy trình được lặp lại khi cần thiết trong mỗi pha cho đến khi đáp ứng được tiêu chí hoàn thành cho pha đó.

Nếu dự án được chia theo vòng đời nghiệp vụ, cần tuân thủ tuyệt đối vòng đời nghiệp vụ, từ việc định nghĩa, chứng minh tính khả thi, cho đến khi phân bổ lợi ích cho doanh nghiệp.

Quan hệ giữa các nhóm quy trình quản lý dự án phần mềm được mô tả như sau:



Hình 4-4: Quan hệ giữa các tiến trình quản lý dự án phần mềm

(nguồn: Software Extension to the PMBOK® Guide)

4.2.2.1 Nhóm quy trình khởi tạo

Nhóm quy trình khởi tạo bao gồm (các) quy trình được thực hiện để xác định một dự án mới hoặc một giai đoạn mới của dự án hiện có để được phép bắt đầu dự án hoặc giai đoạn.

Nhóm quy trình khởi tạo đưa ra định nghĩa và thiết kế dự án: Khái niệm, mục tiêu, cách tiếp cận và cách biện minh rằng một dự án đã được định nghĩa đúng, được đồng ý, và được truyền đạt đúng.

Nhóm quy trình này gồm 2 quy trình con: phát triển điều lệ dự án và xác định các bên liên quan.

Phát triển điều lệ dự án (project charter)

Phát triển điều lệ dự án hay tuyên ngôn dự án là quy trình phát triển một tài liệu chính thức cho phép sự tồn tại của một dự án và cung cấp cho người quản lý dự án quyền áp dụng các nguồn lực của tổ chức vào các hoạt động của dự án. Lợi ích chính của quy trình này là nó cung cấp mối liên kết trực tiếp giữa dự án và các mục tiêu chiến lược của tổ chức, tạo ra hồ sơ chính thức về dự án và thể hiện cam kết của tổ chức đối với dự án. Quy trình này được thực hiện một lần hoặc tại các điểm được xác định trước trong dự án.

Xác định các bên liên quan

Đây là quá trình xác định các bên liên quan của dự án một cách thường xuyên và phân tích và ghi lại thông tin liên quan về lợi ích, sự tham gia, sự phụ thuộc lẫn nhau, ảnh hưởng và tác động tiềm tàng của họ đối với sự thành công của dự án. Lợi ích chính của quy trình này là nó cho phép nhóm dự án xác định trọng tâm thích hợp cho sự tham gia của từng bên liên quan hoặc nhóm bên liên quan. Quy trình này được thực hiện định kỳ trong suốt dự án khi cần thiết.

4.2.2.2 Nhóm quy trình lập kế hoạch

Nhóm Quy trình Lập kế hoạch bao gồm (các) quy trình cần thiết để thiết lập phạm vi của dự án, sàng lọc các mục tiêu và xác định tiến trình hành động cần thiết để đạt được các mục tiêu đó.

Mục tiêu tổng thể của việc lập kế hoạch dự án là thiết lập một chiến lược thực tế để kiểm soát, theo dõi và giám sát một dự án kỹ thuật phức tạp.

Đầu ra của việc lập kế hoạch dự án bao gồm các thông tin về các công việc / hoạt động cần thực hiện trong khuôn khổ dự án:

- Tại sao thực hiện
- Thực hiện khi nào
- Cần những nguồn lực nào
- Cần đáp ứng những tiêu chí nào để dự án được coi là thành công

Đây là nhóm quy trình ảnh hưởng rất lớn tới thành công của quản lý dự án và thành công của dự án, nên chúng ta sẽ đi sâu vào các quy trình này trong phần sau.

4.2.2.3 Nhóm quy trình thực thi

Nhóm quy trình thực thi bao gồm (các) quy trình được thực hiện để hoàn thành công việc được xác định trong kế hoạch quản lý dự án nhằm đáp ứng các yêu cầu của dự án. Lợi ích chính của Nhóm quy trình này là các công việc cần thiết để đáp ứng các yêu cầu và mục tiêu của dự án được thực hiện theo kế hoạch.

Các quy trình trong nhóm được phân thành 2 mảng chính:

Chỉ đạo và quản lý công việc dự án

Đây là các quy trình lãnh đạo và thực hiện công việc được xác định trong kế hoạch quản lý dự án và thực hiện các thay đổi đã được phê duyệt để đạt được mục tiêu của dự án. Lợi ích chính của quy trình này là nó cung cấp khả năng quản lý tổng thể công việc và sản phẩm của dự án, do đó cải thiện khả năng thành công của dự án.

Thực thi một dự án có thể trải qua nhiều giai đoạn, mỗi giai đoạn có mục tiêu và kết quả cần đạt khác nhau. Các giai đoạn thường yêu cầu các kỹ năng, cấu trúc và mức độ tài nguyên khác nhau. Vì phần lớn ngân sách, nguồn lực và thời gian của dự án được sử dụng để thực hiện các quy trình trong nhóm này, nên việc lập kế hoạch, ước lượng chi phí và phân bổ tài nguyên riêng cho từng giai đoạn là cần thiết. Lợi ích theo kế hoạch sẽ được đánh giá và theo dõi trong suốt dự án. Tối ưu hóa lợi ích là một trong số các mục tiêu chính của người quản lý dự án. Do đó, nhóm quy trình thực thi bao

gồm việc điều phối các nguồn lực, quản lý sự tham gia của các bên liên quan cũng như tích hợp và thực hiện các hoạt động của dự án theo kế hoạch quản lý dự án.

Quản lý kiến thức dự án

Đây là các quy trình sử dụng kiến thức hiện có và tạo ra kiến thức mới để đạt được mục tiêu của dự án và đóng góp cho việc học hỏi của tổ chức. Lợi ích chính của quá trình này là kiến thức có trước của tổ chức được tận dụng để tạo ra hoặc cải thiện kết quả của dự án và kiến thức do dự án tạo ra có sẵn để hỗ trợ các hoạt động của tổ chức cũng như các dự án hoặc giai đoạn trong tương lai. Quá trình này được thực hiện trong suốt dự án.

Các quy trình trong nhóm quy trình thực thi có thể tạo ra các yêu cầu thay đổi. Nếu được phê duyệt, các yêu cầu thay đổi có thể kích hoạt một hoặc nhiều quy trình lập kế hoạch dẫn đến kế hoạch quản lý, tài liệu dự án được sửa đổi và có thể cả các đường cơ sở mới.

4.2.2.4 Nhóm quy trình giám sát và kiểm soát

Nhóm quy trình giám sát và kiểm soát bao gồm (các) quy trình cần thiết để theo dõi, xem xét và điều chỉnh tiến độ và hiệu suất của dự án; xác định bất kỳ lĩnh vực nào cần thay đổi kế hoạch; và bắt đầu những thay đổi tương ứng. Lợi ích chính của nhóm quy trình này là hiệu suất dự án được đo lường và phân tích định kỳ, các sự kiện thích hợp hoặc khi xảy ra các điều kiện ngoại lệ nhằm xác định và điều chỉnh những khác biệt so với kế hoạch quản lý dự án.

Nhóm quy trình giám sát và kiểm soát bao gồm 2 nhóm chính: giám sát và kiểm soát công việc dự án và thực hiện kiểm soát thay đổi tích hợp.

Giám sát và kiểm soát công việc dự án

Đây là các quy trình theo dõi, xem xét và báo cáo tiến độ tổng thể để đáp ứng các mục tiêu hoạt động được xác định trong kế hoạch quản lý dự án.

Giám sát là thu thập dữ liệu về hiệu suất của dự án, đưa ra các thước đo về hiệu suất, báo cáo và phổ biến thông tin về hiệu suất.

Kiểm soát là so sánh hiệu suất thực tế với hiệu suất theo kế hoạch, phân tích sự khác biệt, đánh giá các xu hướng để thực hiện cải tiến quy trình, đánh giá các khả năng lựa chọn thay thế và đề xuất hành động khắc phục thích hợp nếu cần.

Lợi ích chính của các quy trình này là nó cho phép các bên liên quan hiểu được trạng thái hiện tại của dự án, nhận ra các hành động được thực hiện để giải quyết mọi vấn đề về hiệu suất và có cái nhìn rõ ràng về trạng thái dự án trong tương lai với các dự báo về chi phí và tiến độ. Các quy trình này được thực hiện xuyên suốt dự án.

Thực hiện kiểm soát thay đổi tích hợp

Đây là các quy trình xem xét tất cả các yêu cầu thay đổi; phê duyệt các thay đổi và quản lý các thay đổi đối với sản phẩm bàn giao, tài sản quy trình tổ chức, tài liệu dự án và kế hoạch quản lý dự án; và truyền đạt các quyết định. Các quy trình này xem xét tất cả các yêu cầu thay đổi tài liệu dự án, sản phẩm bàn giao hoặc kế hoạch quản lý dự án và xác định giải pháp cho các yêu cầu thay đổi. Như vậy, có thể tránh được các rủi ro phát sinh từ việc thực hiện thay đổi mà không xem xét đến các mục tiêu hoặc kế hoạch tổng thể của dự án. Các quy trình này được thực hiện xuyên suốt dự án.

Quá trình giám sát và kiểm soát còn bao gồm các nhiệm vụ:

- Đánh giá các yêu cầu thay đổi và quyết định phản hồi phù hợp;
- Đề xuất hành động khắc phục hoặc phòng ngừa trước những vấn đề có thể xảy ra;
- Giám sát các hoạt động đang diễn ra của dự án so với kế hoạch quản lý dự án và đường cơ sở của dự án; và
- Ảnh hưởng đến các yếu tố có thể phá vỡ quá trình kiểm soát thay đổi nên chỉ những thay đổi đã được phê duyệt mới được thực hiện.

Giám sát liên tục cung cấp cho nhóm dự án và các bên liên quan khác cái nhìn sâu sắc về tình trạng của dự án và xác định bất kỳ lĩnh vực nào cần được chú ý thêm.

Nhóm Quy trình Giám sát và Kiểm soát đang được thực hiện trong từng Lĩnh vực Kiến thức, từng Nhóm Quy trình, từng giai đoạn vòng đời và toàn bộ dự án.

4.2.2.5 Nhóm quy trình kết thúc

Nhóm quy trình kết thúc bao gồm (các) quy trình được thực hiện để chính thức hoàn thành hoặc kết thúc một dự án, giai đoạn hoặc hợp đồng. Nhóm Quy trình này xác minh rằng các quy trình được xác định đã được hoàn thành trong tất cả các Nhóm Quy trình để đóng dự án hoặc giai đoạn nếu thích hợp và chính thức xác nhận rằng dự án hoặc giai đoạn dự án đã hoàn tất. Lợi ích chính của Nhóm quy trình này là các giai đoạn, dự án và hợp đồng được kết thúc một cách thích hợp. Mặc dù chỉ có một quy trình trong Nhóm quy trình này nhưng các tổ chức có thể có các quy trình riêng liên quan đến việc kết thúc dự án, giai đoạn hoặc hợp đồng.

Các hoạt động nổi bật bao gồm:

- Kết thúc dự án
    - Chuyển giao công việc, quy trình, kết quả cần đạt cho các bộ phận chuyên ngành khác.
    - Nộp hồ sơ, tài liệu đúng hạn, đầy đủ chi tiết về hoạt động cũng như quá trình kiểm tra giám sát dự án, đây là cơ sở để bảo trì và phát triển dự án trong tương lai.
    - Giải phóng nhân lực, thiết bị và phương tiện
- Phân bổ lợi ích
- Rà xét việc thực hiện dự án
    - Đánh giá mức độ thành công của dự án
    - Xác định các mục cần cải tiến
    - Rút ra bài học kinh nghiệm

4.2.3 Các phân mục kiến thức quản lý dự án

Ngoài Nhóm quy trình, các quy trình cũng được phân loại theo phân mục kiến thức. Một phân mục kiến thức là một lĩnh vực quản lý dự án được xác định bởi các yêu cầu kiến thức của nó và được mô tả dưới dạng các quy trình thành phần của nó, hướng dẫn thực hiện, đầu vào, đầu ra, công cụ và kỹ thuật.

Mặc dù các phân Kiến thức có liên quan với nhau nhưng chúng được xác định riêng biệt từ góc độ quản lý dự án.

Có 10 phân mục kiến thức được sử dụng trong hầu hết các dự án.

Quản lý tổng thể

Bao gồm các quy trình và hoạt động để xác định, xác định, kết hợp, thống nhất và điều phối các quy trình và hoạt động quản lý dự án khác nhau trong Nhóm quy trình quản lý dự án:

- Xây dựng Điều lệ Dự án
- Xây dựng kế hoạch quản lý dự án
- Chỉ đạo và quản lý công việc dự án
- Quản lý kiến thức dự án
- Giám sát và kiểm soát công việc của dự án
- Thực hiện kiểm soát thay đổi tích hợp
- Kết thúc dự án hoặc giai đoạn

Quản lý phạm vi dự án

Bao gồm các quy trình cần thiết để đảm bảo dự án chỉ bao gồm tất cả các công việc cần thiết để hoàn thành dự án một cách thành công:

- Lập kế hoạch quản lý phạm vi
- Thu thập yêu cầu
- Xác định phạm vi
- Tạo cấu trúc phân cấp công việc
- Kiểm tra tính hợp lệ của phạm vi
- Kiểm soát phạm vi

Quản lý tiến độ dự án

Bao gồm các quy trình cần thiết để quản lý việc hoàn thành dự án đúng thời hạn:

- Lập kế hoạch quản lý lịch trình
- Định nghĩa các hoạt động
- Xâu chuỗi các hoạt động

- Ước tính thời gian thực hiện hoạt động
- Lập lịch
- Kiểm soát tiến độ

Quản lý chi phí dự án

Bao gồm các quy trình liên quan đến lập kế hoạch, ước tính, lập ngân sách, cấp vốn, tài trợ, quản lý và kiểm soát chi phí để dự án có thể được hoàn thành trong phạm vi ngân sách được phê duyệt:

- Lập kế hoạch quản lý chi phí
- Ước lượng chi phí
- Xác định kinh phí
- Kiểm soát chi phí

Quản lý chất lượng dự án

Bao gồm các quy trình để kết hợp chính sách chất lượng của tổ chức liên quan đến việc lập kế hoạch, quản lý và kiểm soát các yêu cầu về chất lượng sản phẩm và dự án nhằm đáp ứng mong đợi của các bên liên quan:

- Lập kế hoạch quản lý chất lượng
- Quản lý chất lượng
- Kiểm soát chất lượng

Quản lý nguồn lực dự án (tài nguyên dự án)

Bao gồm các quy trình để xác định, thu thập và quản lý các nguồn lực cần thiết để hoàn thành thành công dự án.

- Lập kế hoạch quản lý nguồn lực
- Ước lượng nguồn lực cho từng hoạt động
- Tìm kiếm nguồn lực
- Phát triển đội ngũ
- Quản lý đội ngũ
- Kiểm soát nguồn lực

Quản lý truyền thông dự án

Bao gồm các quy trình cần thiết để đảm bảo lập kế hoạch, thu thập, tạo, phân phối, lưu trữ, truy xuất, quản lý, kiểm soát, giám sát và xử lý cuối cùng thông tin dự án kịp thời và phù hợp:

- Lập kế hoạch quản lý truyền thông
- Quản lý truyền thông
- Giám sát truyền thông

Quản lý rủi ro dự án

Bao gồm các quy trình tiến hành lập kế hoạch quản lý rủi ro, xác định, phân tích, lập kế hoạch ứng phó, thực hiện ứng phó và giám sát rủi ro đối với một dự án:

- Lập kế hoạch quản lý rủi ro
- Xác định rủi ro
- Thực hiện phân tích rủi ro định tính
- Thực hiện phân tích rủi ro định lượng
- Lập kế hoạch đáp trả rủi ro
- Thực hiện đáp trả rủi ro
- Giám sát rủi ro

Quản lý thuê mua

Bao gồm các quy trình cần thiết để mua hoặc thu được sản phẩm, dịch vụ hoặc kết quả cần thiết từ bên ngoài nhóm dự án.

- Lập kế hoạch quản lý thuê mua
- Thực hiện thuê mua
- Kiểm soát thuê mua

Quản lý các bên liên quan của dự án

Bao gồm các quy trình cần thiết để xác định những người, nhóm hoặc tổ chức có thể tác động hoặc bị ảnh hưởng bởi dự án, để phân tích kỳ vọng của các bên liên quan và tác động của họ đối với dự án, đồng thời phát triển các chiến lược quản lý phù hợp để thu hút hiệu quả các bên liên quan vào các quyết định và thực hiện dự án.

- Xác định các bên liên quan
- Lập kế hoạch cho sự tham gia của các bên liên quan
- Quản lý sự tham gia của các bên liên quan
- Giám sát sự tham gia của các bên liên quan

4.3 Lập kế hoạch dự án phần mềm

4.3.1 Tổng quan

Trong quản lý dự án, phải phân biệt hai loại kế hoạch. Một là các kế hoạch để đạt được các mục tiêu đề ra, ví dụ như báo cáo phạm vi, lịch trình tổng thể, lịch trình chi phí, v.v. Hai là các kế hoạch quản lý tương ứng với 10 phân mục kiến thức đã đề cập ở phần trước: kế hoạch quản lý phạm vi, kế hoạch quản lý thời gian, kế hoạch quản lý chi phí, v.v.

Dưới đây, chúng ta sẽ xem xét chi tiết kế hoạch quản lý phạm vi và các hoạt động ước lượng, lập lịch.

4.3.2 Lập kế hoạch quản lý phạm vi

4.3.2.1 Thiết lập phạm vi

Đối với phần mềm, phạm vi sản phẩm bao gồm các tính năng, các dữ liệu vào ra, các nội dung được trả về cho người dùng khi họ sử dụng phần mềm, và các thuộc tính chất lượng (như hiệu quả hoạt động, ràng buộc, giao diện, và độ tin cậy gắn với hệ thống) mà người dùng, khách hàng và các bên liên quan khác cần và mong muốn. Phạm vi sản phẩm có thể được sử dụng để ước tính phạm vi dự án (tức là tiến độ, ngân sách, nguồn lực và công nghệ). Ngoài ra, các ràng buộc về phạm vi dự án có thể xác định phạm vi sản phẩm (các tính năng và thuộc tính chất lượng). Những ràng buộc về cả phạm vi dự án và phạm vi sản phẩm có thể yêu cầu sự đánh đổi giữa các tính năng, thuộc tính chất lượng, tiến độ, ngân sách, nguồn lực và công nghệ.

Phạm vi dự án và phạm vi sản phẩm xác định công sức (hay còn gọi là số ngày công lao động) để phát triển hoặc sửa đổi một sản phẩm phần mềm. Chi phí nhân công là chi phí chính cho hầu hết các dự án phần mềm, bởi vì phần mềm là sản phẩm trực tiếp của công sức của nhóm dự án. Chi phí bổ sung có thể bao gồm chi phí cho các yếu tố

như đào tạo người dùng, tài liệu sản phẩm, nền tảng phần cứng và phần mềm và có thể cả môi trường thử nghiệm chuyên dụng. Công sức của nhóm cũng được dùng làm cơ sở để xác định tiến độ cho một dự án phần mềm. Ví dụ, một dự án cần 60 ngày công lao động có thể được lên kế hoạch là 3 người làm trong 20 ngày.

Phạm vi phần mềm được mô tả bằng một trong hai kỹ thuật:

- Mô tả tường thuật của phần mềm, được tạo ra sau khi trao đổi với tất cả các bên liên quan.
- Một tập hợp các trường hợp sử dụng (use-case) được người dùng cuối tạo ra.

4.3.2.2 Tạo lập cấu trúc phân cấp công việc

Tạo WBS là quá trình chia nhỏ các sản phẩm bàn giao của dự án và công việc của dự án thành các phần nhỏ hơn, dễ quản lý hơn các thành phần. Lợi ích chính của quá trình này là nó cung cấp một khuôn khổ về những công việc cần thực hiện để tạo ra sản phẩm bàn giao. Nói cách khác, WBS chính là thể hiện của phạm vi dự án. Vì phạm vi dự án phần mềm được thiết lập dựa trên các yêu cầu ban đầu về sản phẩm phần mềm, từ đó xác định kiến trúc phần mềm, và tạo cơ sở cho việc phát triển cấu trúc phân cấp công việc, nên cấu trúc phân cấp công việc phụ thuộc vào mô hình vòng đời phần mềm được sử dụng để quản lý dự án. Nếu các yêu cầu phần mềm đủ ổn định để có thể được phát triển đủ chi tiết tại thời điểm khởi tạo hay lập kế hoạch dự án, quá trình tạo lập WBS được thực hiện một lần duy nhất. Nếu dự án phần mềm dự kiến có nhiều thay đổi, không thể chốt ngay từ đầu, ví dụ như người dùng không chắc chắn họ cần gì, hoặc đội thực hiện không chắc chắn có thể đáp ứng yêu cầu do liên quan đến công nghệ mới (phần cứng mới, phần mềm cơ sở hạ tầng mới) hoặc có sự thay đổi về chính sách, quy định mới, v.v… quá trình tạo lập WBS cần được thực hiện nhiều lần tại các thời điểm được xác định trước trong dự án.

Có 2 dạng cấu trúc phân cấp công việc là dạng bảng hoặc dạng cây. Lá của cây chính là gói công việc. Gói công việc (work package - WP) là công việc theo kế hoạch được chứa ở mức thấp nhất của WBS.

Đối với các dự án phần mềm, ta có thể phân cấp theo quy trình hoặc hoạt động trong vòng đời phần mềm, gọi là WBS hướng hoạt động hoặc theo sản phẩm giao nộp của dự án, gọi là WBS hướng sản phẩm bàn giao.

Phân chia công việc theo sản phẩm

Với cấu trúc phân chia công việc theo định hướng sản phẩm bàn giao, kế hoạch dự án có thể được thể hiện dưới dạng sản phẩm bàn giao và bàn giao phụ. Ví dụ, trong cây phân cấp, các cây con chính là các mô-đun chính cần bàn giao, các lá chính là các chức năng con cần bàn giao.

Lưu ý: Khi tạo lập cấu trúc công việc, cần định danh các phần tử của cấu trúc bằng cách đánh số. Để kiểm chứng xem cấu trúc phân cấp này có đúng chưa, ta sử dụng quy tắc 100%: tất cả các phần tử ở mức thấp hơn đều thuộc về 1 phần tử nào đó ở mức cao hơn.

Phân chia công việc theo quy trình

Cấu trúc phân chia công việc theo định hướng hoạt động là dạng phổ biến nhất đối với các dự án phần mềm. Trong đó, mức cao nhất của cấu trúc tương ứng với quy trình hoặc hoạt động trong vòng đời phần mềm. Công việc và sản phẩm giao nộp được thể hiện dưới dạng đầu ra của các hoạt động và nhiệm vụ ở mức thấp hơn.

Các gói công việc bao gồm đặc tả các hoạt động và các kết quả tương ứng, cũng như các tiêu chí đánh giá kết quả thực hiện.

Một khuyến nghị chung về việc xác định công việc là tất cả các công việc phải có kết quả đầu ra có thể đo lường được, có bằng chứng rõ ràng rằng công việc đã được hoàn thành thành công và thể hiện mục đích chính của công việc. Ví dụ: Trong bảng, công việc “Kiểm thử" có thể tạo ra sản phẩm có tên là “Báo cáo kiểm thử".

- 1 gói công việc xây dựng một thành phần phần mềm được đặc tả như sau:
    - Thời gian dự kiến
    - Thành phần phần mềm cần xây dựng hoặc chỉnh sửa
    - Số nhân sự thực hiện (nhóm theo năng lực, nơi làm việc)
    - Các nguồn lực khác cần sử dụng:
        - Môi trường: công cụ phần mềm, thiết bị phần cứng, tài nguyên mạng
        - Các thành phần phần mềm khác có thể tái sử dụng: mới hoàn toàn, sẵn có trên thị trường, đã từng làm, đã từng làm một phần

- Tiêu chí chấp nhận
- Các yếu tố rủi ro

4.3.3 Ước lượng dự án

Ước lượng dự án là quy trình dự báo thời gian, chi phí và nguồn lực cần thiết để thực hiện một dự án, thường được thực hiện trong quy trình khởi tạo và lập kế hoạch dự án, có tính đến phạm vi, thời hạn và rủi ro tiềm ẩn của dự án.

4.3.3.1 Mục tiêu ước lượng

Các mục tiêu ước lượng thường gặp bao gồm:

Quy mô

Cần ước tính quy mô của sản phẩm bàn giao và công việc, quy trình trong dự án. Đây là một phần thiết yếu của quản lý dự án phần mềm. Có nhiều kỹ thuật được sử dụng để ước tính quy mô dự án như điểm chức năng, điểm trường hợp sử dụng, số dòng code, v.v. Mỗi kỹ thuật này đều có điểm mạnh và điểm yếu và việc lựa chọn kỹ thuật phụ thuộc vào nhiều yếu tố khác nhau như độ phức tạp của dự án, dữ liệu sẵn có và chuyên môn của nhóm.

Năng suất

Cần ước tính yêu cầu về năng suất trong một khoảng thời gian nhất định. Năng suất lao động là khối lượng công việc mà một thành viên nhóm dự án hoặc một nhóm dự án có thể hoàn thành trong một khoảng thời gian nhất định, chẳng hạn như một giờ, một ngày hoặc một tuần. Nó thường được đo bằng tỷ lệ giữa số lượng công việc được thực hiện với số giờ lao động đã sử dụng, hoặc số giờ làm việc hiệu quả hàng ngày. Ví dụ: nếu 1 ngày làm việc (8 giờ), 1 lập trình viên dành 48 phút để viết tài liệu, 48 phút để họp và trao đổi thông tin với các thành viên khác, thời gian còn lại để lập trình thì năng suất lập trình là 0.8.

Số ngày công

Cần ước tính số ngày công (man day) hay công sức (effort) để phát triển phần mềm, quản lý dự án. Giờ công (man hours) xác định khối lượng công việc được thực hiện

bởi một chuyên gia CNTT trung bình trong một giờ làm việc mà không bị gián đoạn. Để ước tính chính xác, cần lưu ý rằng việc xác định giờ công, ngày công là riêng biệt cho từng thành viên nhóm. Lý do là mức lương của các vị trí công việc trong dự án khác nhau đáng kể.

Hai kỹ thuật thường gặp nhất là sử dụng tham số và sử dụng các số liệu từ dự án tương tự trong quá khứ.

Thời gian

Cần ước tính thời gian thực hiện từng nhiệm vụ trong dự án và tổng thời gian thực hiện dự án. Do có nhiều nhân lực thực hiện song song nên có thể rút ngắn hơn nhiều so với số ngày công.

Chi phí

Tổng chi phí cần thiết để thực hiện dự án. Để ước lượng các nguồn lực, chi phí và lịch trình cho một dự án phần mềm, cần có:

- Kinh nghiệm
- Khả năng tiếp cận tốt với thông tin lịch sử (các số liệu)
- Đủ can đảm để cam kết với dự đoán định lượng khi chỉ tồn tại các thông tin định tính

Do có sự không chắc chắn về thông tin đầu vào, kết quả ước lượng có thể khác so với thực tế.

4.3.3.2 Các kỹ thuật ước lượng dự án

Các kỹ thuật ước lượng thường gặp có thể được phân nhóm như sau:

Sử dụng kinh nghiệm từ các dự án tương tự

Ước tính tương tự là một kỹ thuật sử dụng giá trị của các tham số từ dữ liệu lịch sử làm cơ sở để ước tính tham số tương tự cho một hoạt động trong tương lai. Các thông tin liên quan đến chi phí sản xuất từ các dự án tương tự trước đây được xem xét và so sánh với dự án hiện tại. Các tham số thường sử dụng là phạm vi, chi phí và số ngày công. Sự khác biệt giữa chúng được ghi nhận. Đó có thể là những yếu tố như độ phức tạp, quy mô của dự án, ngày giao hàng dự kiến, địa điểm, lạm phát, tỷ giá hối đoái hiện tại,

v.v. Lưu ý, khi quy mô, độ phức tạp của mỗi hoạt động tăng lên thì chi phí và số ngày công phải tăng theo. Giá trị ước lượng được đưa ra cho mỗi hoạt động trên cơ sở phân tích các yếu tố nói trên.

Sử dụng tri thức chuyên gia

Các kỹ thuật thông thường sử dụng tri thức chuyên gia thường sử dụng để:

- Đánh giá quy mô, kích thước
- Phân tích công việc và ước tính công sức

Điểm chức năng (Function point):

Kỹ thuật này liên quan đến việc ước tính quy mô dự án dựa trên chức năng do phần mềm cung cấp. Ví dụ, số đầu vào, đầu ra, số truy vấn, số file sử dụng, hoặc số lượng các giao tiếp cần thiết để thực hiện 1 chức năng.

Điểm trường hợp sử dụng (UC point):

Kỹ thuật này liên quan đến việc ước tính quy mô dự án dựa trên số lượng trường hợp sử dụng mà phần mềm phải hỗ trợ. Điểm trường hợp sử dụng xem xét các yếu tố như độ phức tạp của từng trường hợp sử dụng, số lượng tác nhân tham gia và số lượng trường hợp sử dụng.

Dòng mã (LOC):

Như tên cho thấy, LOC đếm tổng số dòng mã nguồn trong một dự án. Kích thước được ước tính bằng cách so sánh nó với các hệ thống hiện có cùng loại. Các chuyên gia sử dụng nó để dự đoán kích thước cần thiết của các thành phần khác nhau của phần mềm và sau đó cộng chúng lại để có được kích thước tổng thể.

Planning Pokcr

Trong các dự án phần mềm vừa và nhỏ phát triển theo mô hình Agile, người ta hay dùng phương pháp Planning Pocker để thảo luận về số ngày công cần thiết thực hiện mỗi hoạt động. Tất cả các thành viên nhóm dự án đều tham gia ước lượng. Họ dùng một bộ bài, mỗi quân bài có giá trị cho sẵn về số ngày công cần thiết. Với mỗi chức

năng cần xây dựng hay hoạt động cần thực hiện, mỗi thành viên tự chọn một quân bài tương ứng với giá trị mình ước lượng. Khi lật đồng thời các quân bài, nếu cùng giá trị thì đó chính là giá trị ước lượng của chức năng, nếu không, cần bổ sung thông tin và tiếp tục thảo luận, sau đó lặp lại quá trình chọn quân bài / lật quân bài cho đến khi đạt được sự đồng thuận của các thành viên. Quá trình này tiếp tục cho đến khi tất cả các hoạt động được ước lượng.

Sử dụng mô hình thực nghiệm

Có rất nhiều mô hình ước lượng công sức và chi phí thực hiện dự án phần mềm đã được nghiên cứu đề xuất. Người quản lý dự án có thể tìm hiểu và vận dụng các mô hình này cho dự án phần mềm của mình.

Sử dụng các công cụ tự động

Hiện nay, có rất nhiều phần mềm quản lý dự án cung cấp các tính năng hỗ trợ ước lượng dự án. Người quản lý có thể sử dụng các công cụ này.

4.3.3.3 Độ chính xác của ước lượng dự án

Độ chính xác phụ thuộc vào:

- Quy mô của sản phẩm có được ước lượng đúng không
- Số liệu từ các dự án trước về năng suất lao động, thời gian, chi phí cho các nguồn lực có đáng tin cậy hay không
- Khả năng của nhóm dự án có được phản ánh đúng mức trong kế hoạch không
- Sự ổn định của yêu cầu sản phẩm và môi trường hỗ trợ kỹ thuật phần mềm.

Các kỹ thuật ước lượng dự án phần mềm thường có sai sót và nhược điểm cố hữu, do độ chính xác và chắc chắn của dữ liệu đầu vào và những hạn chế về kinh nghiệm và trực giác của con người. Do đó, việc kết hợp các kỹ thuật có thể giúp giảm thiểu rủi ro ước lượng sai.

4.3.4 Lập lịch dự án

Lịch trình dự án là một thời gian biểu sắp xếp các nhiệm vụ, nguồn lực và ngày đến hạn theo một trình tự lý tưởng để dự án có thể được hoàn thành đúng thời hạn. Lịch trình dự án được tạo ra trong giai đoạn lập kế hoạch và bao gồm những nội dung sau:

- Dòng thời gian của dự án với ngày bắt đầu, ngày kết thúc và các mốc quan trọng
- Danh mục nhiệm vụ cần thiết để hoàn thành sản phẩm của dự án
- Chi phí, nguồn lực và sự phụ thuộc liên quan đến từng nhiệm vụ
- Người thực hiện từng nhiệm vụ

Lịch trình dự án được tạo và theo dõi bằng phần mềm lập lịch dự án, phần mềm này có các tính năng chính cho phép người quản lý dự án giám sát tiến độ nhiệm vụ, nguồn lực và chi phí trong thời gian thực. Họ cũng có thể phân công công việc, liên kết các nhiệm vụ phụ thuộc, xem bảng thông tin, phân bổ tài nguyên và hơn thế nữa.

Lập lịch dự án là phương pháp tạo ra lịch trình nói trên.

Để lập lịch, cần liệt kê các nhiệm vụ, xem xét các quan hệ phụ thuộc giữa các nhiệm vụ để thiết lập thứ tự thực hiện phù hợp. Tiếp đó, thực hiện phân bổ chi phí, nguồn lực và phân công người thực hiện công việc.

Các hướng dẫn về lập lịch sẽ được mô tả chi tiết trong phần bài tập.

4.4 Quản lý rủi ro dự án phần mềm

4.4.1 Giới thiệu

4.4.1.1 Rủi ro

Khái niệm rủi ro

Trong bối cảnh quản lý dự án, rủi ro dự án có thể được định nghĩa là khả năng (không chắc chắn) xảy ra một số sự cố nhất định ảnh hưởng tích cực hoặc tiêu cực đến mục tiêu dự án. Ví dụ: những sự kiện có thể làm phá vỡ một dự án, những điều không chắc chắn, những khoản nợ hay những điểm yếu có thể làm cho dự án không đi theo đúng kế hoạch đã định.

Các đặc trưng của rủi ro

Rủi ro dự án được đặc trưng bởi các yếu tố sau:

- Sự kiện hoặc nhận dạng rủi ro, tức là chính xác những gì có thể xảy ra gây tổn hại cho dự án;
- Xác suất rủi ro, tức là khả năng xảy ra sự kiện; Và
- Mức độ tác động, tức là mức độ tổn thất quy ra tiền có thể xảy ra.

Rủi ro trong dự án phần mềm

Với dự án phần mềm, có thể kể đến một số rủi ro như sau:

Rủi ro kỹ thuật:

- Phần mềm hoạt động không đúng yêu cầu do quá nhiều lỗi
- Mã nguồn chất lượng kém dẫn đến phần mềm không mở rộng theo yêu cầu về năng lực hoặc hiệu suất
- Các yêu cầu không được xác định rõ hoặc hiểu sai
- Các mô-đun phần mềm được tích hợp trễ, dẫn tới việc kiểm thử trễ, khi phát hiện lỗi không có đủ thời gian sửa chữa
- Phần mềm không đáp ứng được mong đợi và nhu cầu của khách hàng
- Người dùng cuối khó sử dụng phần mềm
- Lập trình lại hoàn toàn phần mềm hoặc tái cấu trúc phần mềm quá mức do yêu cầu không ổn định, lạm phát yêu cầu hoặc thay đổi trong các tình huống
- Lựa chọn nền tảng phát triển, ngôn ngữ hoặc công cụ mới trong khi số lượng nhân sự hạn chế và thiếu kinh nghiệm
- Phần mềm bị lỗi do không thực hiện đầy đủ các hoạt động quản lý cấu hình cơ sở, kiểm thử các phiên bản phần mềm
- Thay đổi, nâng cấp công nghệ trong quá trình thực hiện dự án
- Sự phụ thuộc bên ngoài vào khả năng của dự án khác trong việc cung cấp đầu vào hữu ích và kịp thời.

Rủi ro an toàn:

- Phần mềm, hệ thống đã phát triển có những khiếm khuyết có thể gây thương tích, tử vong cho người dùng hoặc hủy hoại môi trường.

- Các mối lo ngại về sức khỏe và an toàn liên quan đến công việc, nơi làm việc và các quy trình trong dự án

Rủi ro bảo mật:

- Tính toàn vẹn của hệ thống được phát triển không phù hợp với mức độ quan trọng cần thiết của phần mềm (khả năng xảy ra hậu quả nghiêm trọng do trục trặc)
- Đội phát triển không có kinh nghiệm xử lý các mối đe dọa bảo mật có thể xảy ra đối với phần mềm
- Thiết kế hệ thống không đầy đủ để kiểm soát quyền truy cập, bảo vệ dữ liệu cá nhân hoặc độc quyền khi lưu trữ và truyền tải cũng như bảo vệ hệ thống khỏi phần mềm độc hại và xâm nhập trái phép
- Tái sử dụng mã nguồn chưa rõ nguồn gốc
- Vi phạm chính sách bảo mật làm ảnh hưởng đến sự phát triển hoặc cơ sở hạ tầng sản xuất.

Đội ngũ:

- Thiếu kinh nghiệm về các công cụ, quy trình tổ chức, phương pháp phát triển hoặc yêu cầu nghiệp vụ của khách hàng
- Thiếu nhân lực (thiếu nhân sự thực hiện hoặc nhân sự bị điều động cho các dự án khác, luân chuyển nhân sự)
- Thành viên mệt mỏi, kiệt sức vì cố gắng quá mức
- Các vấn đề liên lạc và phối hợp trong nhóm hoặc với các bên liên quan do nhóm phân tán hoặc chỉ liên lạc từ xa hoặc sự khác biệt về văn hóa
- Khác biệt về kết quả bàn giao của thành viên mới và thành viên giàu kinh nghiệm
- Nhiều lập trình viên làm việc trên cùng một nhánh mã.

Rủi ro lịch trình:

- Lịch trình cơ sở không nhất quán với tốc độ thực hiện dự án thực tế
- Không hoàn thành các chức năng thiết yếu hoặc bắt buộc của phần mềm theo đúng thời hạn bàn giao theo lịch trình

- Phạm vi thay đổi ảnh hưởng đến việc hoàn thành các mục tiêu ban đầu
- Sự chậm trễ trong giai đoạn lập trình dẫn đến áp lực phải rút ngắn thời gian kiểm thử và triển khai thử nghiệm
- Các phép đo hoàn thành dự án (như số dòng code đã viết, phần tram ước tính hoàn thành) không phản ánh đúng tiến độ và hiệu quả dự án
- Các kế hoạch không đề cập đến kiến trúc ban đầu và thiết kế dữ liệu hoặc tài liệu hoặc thử nghiệm tích hợp;
- Chỉ lập lịch kiểm thử một lần duy nhất, bỏ qua việc kiểm thử sau khi sửa đổi mã nguồn

Chi phí:

- Ước tính không chính xác về số lượng nhân công thực hiện từng loại hoạt động như thiết kế, cài đặt, kiểm thử, v.v., và năng suất/tốc độ của họ
- Chi phí thực tế vượt quá nguồn vốn sẵn có, không thể đáp ứng được thách thức về khả năng chi trả.

Khách hàng và bên liên quan:

- Không có dữ liệu về quy trình nghiệp vụ thực tế
- Không có dữ liệu kỹ thuật về các hệ thống được thay thế hoặc kết nối
- Không có tiêu chí chấp nhận (hoặc phân tích nhu cầu thị trường)
- Không có đại diện của khách hàng hoặc người dùng để xác định mức độ ưu tiên các yêu cầu/tính năng, kiểm thử người dùng và chấp nhận hệ thống.

4.4.1.2 Quản lý rủi ro dự án phần mềm

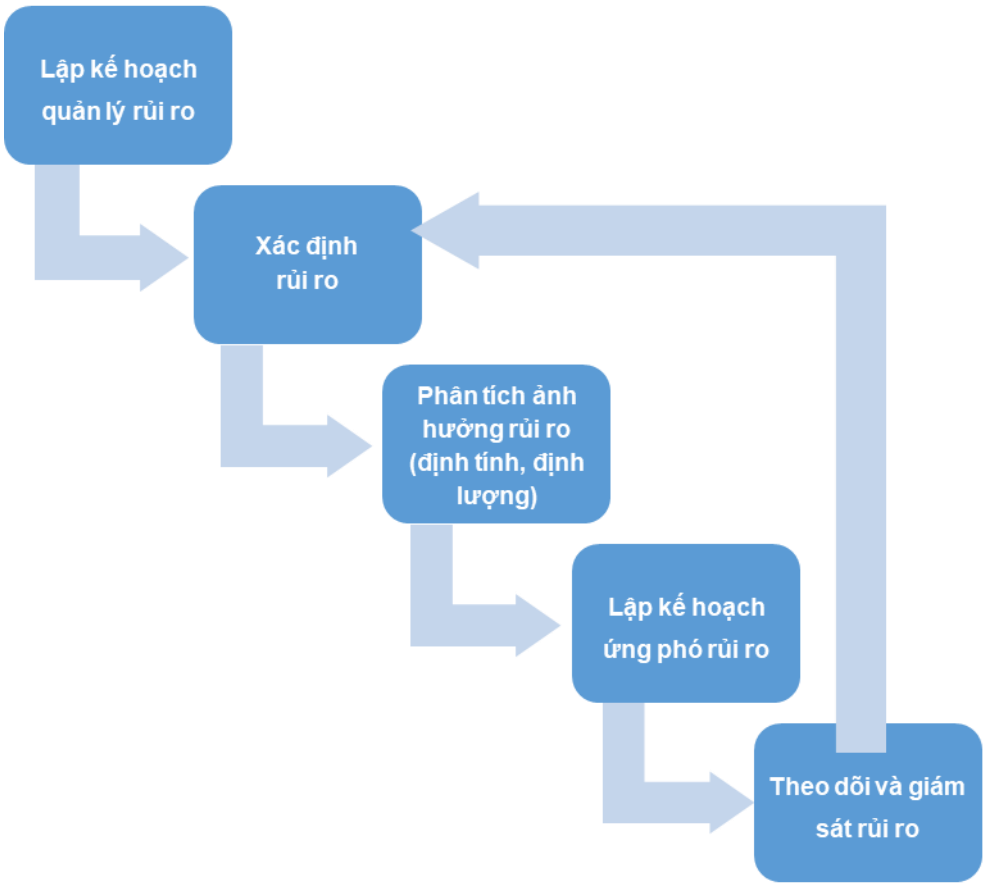
Quản lý rủi ro, trong bối cảnh dự án, là nghệ thuật và khoa học trong việc lập kế hoạch quản lý rủi ro, xác định, phân tích, lập kế hoạch ứng phó và kiểm soát các yếu tố rủi ro trong suốt vòng đời của dự án phần mềm và vì lợi ích tốt nhất của các mục tiêu dự án. Dự án có thành công hay không, có theo đúng kế hoạch hay không phụ thuộc lớn vào việc rủi ro có được kiểm soát tốt hay không. Quản lý rủi ro cho phép tăng khả năng và

tác động của các sự kiện tích cực trong dự án, cũng như giảm xác suất và ảnh hưởng của các sự kiện tiêu cực / sự cố không biết trước trong dự án.

Các thời điểm cần quản lý rủi ro bao gồm:

- Khi lập kế hoạch quản lý
- Khi dự án sẵn sàng thực thi
- Khi khôi phục một dự án đã bỏ dở
- Khi rà xét dự án
- Khi có sự sai lệch lớn so với kế hoạch xảy ra

4.4.2 Quy trình quản lý rủi ro dự án phần mềm



Hình 4-5: Quy trình quản lý rủi ro

4.4.2.1 Xác định rủi ro

Xác định rủi ro là quá trình xác định các rủi ro dự án riêng lẻ cũng như các nguồn rủi ro tổng thể của dự án và ghi lại các đặc điểm của chúng. Lợi ích chính của quá trình

này là ghi lại các rủi ro dự án riêng lẻ hiện có và các nguồn rủi ro tổng thể của dự án. Nó cũng tập hợp thông tin để nhóm dự án có thể ứng phó phù hợp với những rủi ro đã được xác định.

Các nhóm rủi ro thường gặp bao gồm

- Kích thước sản phẩm - rủi ro gắn liền với kích thước tổng thể của phần mềm được xây dựng hoặc chỉnh sửa.
- Tác động kinh doanh - rủi ro gắn liền với các ràng buộc trong quản lý hoặc thị trường.
- Đặc điểm của khách hàng - rủi ro gắn liền với trình độ sử dụng của khách hàng và khả năng của nhà phát triển trong việc giao tiếp với khách hàng một cách kịp thời.
- Định nghĩa quá trình - rủi ro gắn liền với mức độ mà các quá trình phần mềm đã được định nghĩa và được theo dõi bởi tổ chức phát triển.
- Môi trường phát triển - rủi ro gắn liền với sự tiện lợi và chất lượng của các công cụ để xây dựng sản phẩm.
- Công nghệ sử dụng - rủi ro gắn liền với sự phức tạp của hệ thống và "sự mới mẻ" của công nghệ được dùng trong hệ thống.
- Số lượng nhân viên và kinh nghiệm - rủi ro gắn liền với kinh nghiệm kỹ thuật và kinh nghiệm trong dự án của các kỹ sư phần mềm.

Ngoài ra, còn có thể quan tâm đến 2 loại rủi ro khác là:

- Rủi ro hiệu năng - mức độ không chắc chắn rằng sản phẩm sẽ đáp ứng yêu cầu và phù hợp với mục đích sử dụng của nó.
- Rủi ro hỗ trợ - mức độ không chắc chắn rằng phần mềm có được sẽ có thể dễ dàng sửa chữa, hiệu chỉnh và nâng cao.

4.4.2.2 Phân tích rủi ro

Phân tích rủi ro bao gồm việc sử dụng các công cụ và kỹ thuật để xác định khả năng và tác động của các rủi ro dự án đã được xác định trước đó.

Rủi ro được phân tích theo các khía cạnh:

- Xác suất xảy ra rủi ro: Mọi rủi ro đều có xác suất xảy ra nhất định.

- Mức độ tác động của rủi ro đến dự án: Một số rủi ro sẽ mang lại căng thẳng tài chính, trong khi những rủi ro khác có thể liên quan đến các vấn đề quản lý nguồn lực hoặc sự chậm trễ trong tiến độ dự án.
- Thời điểm xảy ra rủi ro: xảy ra tại pha nào, hoạt động nào trong dự án.
- Phân loại rủi ro: rủi ro về kỹ thuật, bảo mật, v.v.

Do đó, phân tích rủi ro giúp các nhà quản lý dự án giải mã sự không chắc chắn của các rủi ro tiềm ẩn và cách chúng tác động đến dự án về mặt tiến độ, chất lượng và chi phí nếu trên thực tế chúng xuất hiện.

Phân tích định tính

Phân tích rủi ro định tính là quá trình thực hiện phân tích các rủi ro theo thứ tự ưu tiên thông qua việc đánh giá khả năng xảy ra và tác động cũng như các đặc điểm khác của chúng. Lợi ích chính của quá trình này là nó tập trung nỗ lực vào những rủi ro có mức độ ưu tiên cao. Quá trình này được thực hiện xuyên suốt dự án.

Phân tích định lượng

Phân tích rủi ro định lượng là quá trình phân tích đánh giá các thông số tác động tổng hợp của các rủi ro đã xác định và các nguồn không chắc chắn khác đối với các mục tiêu tổng thể của dự án.

Đối với các dự án phần mềm, việc xác định rủi ro và phân tích rủi ro cố gắng tập trung vào những rủi ro có thể xảy ra nhất và có tác động cao nhất hơn là tác động tích lũy của chuỗi các rủi ro nhỏ. Ngoài ra, tác động của một số rủi ro có thể khó định lượng về mặt chi phí trực tiếp đối với dự án hoặc tổ chức. Nhưng mục đích của việc phân tích rủi ro định lượng cho các dự án phần mềm thường là đưa ra hành động dựa trên điểm số tương đối hơn là những con số chính xác về số tiền tổn thất. Mục tiêu là đạt được sự đồng thuận với các bên liên quan của dự án phần mềm về những con số hợp lý để sử dụng làm cơ sở cho việc sắp xếp thứ tự ưu tiên chứ không phải báo cáo chi phí trên bảng cân đối kế toán.

4.4.2.3 Lên kế hoạch ứng phó rủi ro

Bảng 4-1: Khuyến cáo ứng phó rủi ro theo Software Extension to the PMBOK® Guide

Loại rủi ro	Cách ứng phó	Mô tả
Kỹ thuật	Tránh	Sử dụng nền tảng và ngôn ngữ phát triển đã được công nhận. Thay đổi các yêu cầu.
	Chuyển	Sử dụng các công cụ và mô-đun có sẵn trên thị trường hoặc sử dụng lại các mô-đun phần mềm hiện có thay vì tạo ra các thiết kế mới (mua thay vì xây dựng).
	Giảm thiểu	Thu hút sự tham gia liên tục của khách hàng và nhà phát triển. Làm việc theo từng giai đoạn ngắn để rủi ro có thể được xác định sớm và quá trình phát triển nhằm giảm thiểu rủi ro có thời gian để tạo ra tác động. Đào tạo nhóm về các phương pháp phát triển mới Có được cam kết của nhà tài trợ dự án đối với những thay đổi. Tiến hành kiểm tra hồi quy để tìm những thay đổi đối với phần mềm quan trọng có thể ảnh hưởng đến các mô-đun tiếp theo hoặc hiệu suất tổng thể.
Bảo mật	Tránh	Mặc dù không có cách nào để tránh tất cả các rủi ro và mối đe dọa bảo mật, nhưng hãy sử dụng các kỹ thuật kiểm soát truy cập và mã hóa an toàn cũng như các kiến trúc được công nhận, hãy tuân theo các tiêu chuẩn bảo mật.
	Chuyển	Nhận bộ phần mềm và công cụ từ các nguồn được công nhận với cam kết khắc phục các lỗ hổng bảo mật. Các nguồn được công nhận bao gồm cộng đồng nguồn mở cũng như các nhà cung cấp phần mềm thương mại độc quyền.

Loại rủi ro	Cách ứng phó	Mô tả
	Giảm thiểu	Đào tạo các nhà phát triển về cách mã hóa an toàn. Thu hút sự tham gia của người kiểm tra thâm nhập phần mềm và phát hiện xâm nhập độc lập để cấp chứng chỉ phần mềm.
Đội ngũ	Tránh	Sử dụng đội ngũ và người quản lý tận tâm, giàu kinh nghiệm cũng như các quy trình tổ chức đã được thiết lập.
	Chuyển	Sử dụng các quy trình hợp tác để không có điểm lỗi duy nhất; Tuyển dụng hoặc ký hợp đồng với các nhà cung cấp lao động để cung cấp nhân viên dự phòng hoặc tăng đột biến. (Lưu ý rằng việc bổ sung nhân viên muộn trong dự án thường làm dự án chậm hơn trong khi nhân viên mới sẽ bắt kịp tốc độ.)
	Giảm thiểu	Cân bằng nhân sự giữa nhân viên cấp cao đắt tiền hơn và nguồn lực cấp dưới ít tốn kém hơn bằng việc huấn luyện và đào tạo. Cải thiện phương pháp giao tiếp nhóm để tránh công việc trùng lặp hoặc làm lại.
Lịch trình	Tránh	Xem lại lịch trình cơ sở để đảm bảo tính chính xác trong việc phân bổ thời gian tương ứng cho các hoạt động, tải nguồn lực và lộ trình quan trọng. Dành thời gian cho việc lập kế hoạch và thiết kế trước khi bắt đầu phát triển quy mô lớn.
	Chuyển	Thu hút khách hàng tham gia vào các quyết định kiểm soát thay đổi tại các điểm kiểm tra của dự án hoặc các ưu tiên và nội dung của nước rút. Hãy để nhóm tham gia vào việc lập kế hoạch và ước tính.
	Giảm thiểu	Bắt đầu sớm các hoạt động quan trọng và có rủi ro cao hơn trong lịch trình để có thời gian tạo nguyên mẫu, thử nghiệm, lặp lại, tích hợp và thử nghiệm lại. Xây dựng dự trữ vào lịch trình. Nhận

Loại rủi ro	Cách ứng phó	Mô tả
		phản hồi sớm về sự khác biệt so với lịch trình và điều chỉnh các kế hoạch lặp lại.
Chi phí	Tránh	Ước tính theo điểm chức năng đã hoàn thành và kiểm tra, thay vì đếm số dòng code hoặc ước tính phần trăm hoàn thành. Sử dụng nhiều kỹ thuật ước tính chi phí.
	Chuyển	Đưa ra các đề xuất thay đổi để khiến khách hàng phải gánh chịu chi phí do các vấn đề không mong muốn hoặc lợi ích của các cơ hội tiết kiệm chi phí.
	Giảm thiểu	Chuyển nguồn lực từ các hoạt động ít quan trọng hơn hoặc giảm bớt mức độ ưu tiên thấp hơn.
Khách hàng và bên liên quan	Tránh	Xây dựng điều lệ dự án, hợp đồng hoặc thỏa thuận làm việc để làm rõ vai trò và trách nhiệm dự kiến của khách hàng.
	Chuyển	Chỉ định một đại diện khách hàng đại diện cho tiếng nói của người dùng với nhiều tổ chức tài trợ.
	Giảm thiểu	Chỉ định các trường hợp dự phòng và giả định trong trường hợp không có dữ liệu khách hàng. Tiến hành các hướng dẫn và nguyên mẫu để xây dựng sự chấp nhận của khách hàng.

4.4.2.4 Theo dõi và giám sát rủi ro

Giám sát và kiểm soát rủi ro cho các dự án phần mềm bao gồm theo dõi các rủi ro đã xác định, giám sát rủi ro còn sót lại, thực hiện kế hoạch xử lý rủi ro và đánh giá hiệu quả của chúng. Trong các dự án phần mềm nhỏ, việc giám sát và kiểm soát rủi ro là một phần nhiệm vụ của người quản lý dự án. Trong các dự án lớn, một cá nhân khác, thường là chuyên gia lập kế hoạch hoặc đảm bảo chất lượng, được chỉ định làm người quản lý rủi ro và được giao trách nhiệm ghi lại các rủi ro mới vào sổ đăng ký rủi ro và

tham khảo ý kiến của người quản lý dự án để đảm bảo rằng các biện pháp giảm thiểu rủi ro đang được thực hiện. và hoàn thành vào ngày hoàn thành đã thỏa thuận.

Theo dõi và giám sát rủi ro liên quan đến 3 câu hỏi chính:

- Ngăn ngừa - Làm thế nào để ngăn ngừa rủi ro?
- Giám sát - Những yếu tố mà ta có thể theo dõi, cho phép ta xác định khả năng của rủi ro tăng lên hay giảm xuống?
- Quản lý - Ta có kế hoạch dự phòng gì nếu rủi ro trở thành hiện thực?

Chương 5 QUẢN LÝ CẤU HÌNH PHẦN MỀM

Nội dung:

- Khái niệm quản lý cấu hình phần mềm
- Quy trình quản lý cấu hình phần mềm
- Quản lý phiên bản

5.1 Khái niệm quản lý cấu hình phần mềm

Trước hết chúng ta sẽ làm quen với các khái niệm cơ bản của quản lý cấu hình phần mềm. Việc hiểu rõ các khái niệm này có vai trò quan trọng để giúp các bạn nắm được ý nghĩa, vai trò và các hoạt động của quản lý cấu hình trong các tiến trình và dự án phát triển phần mềm.

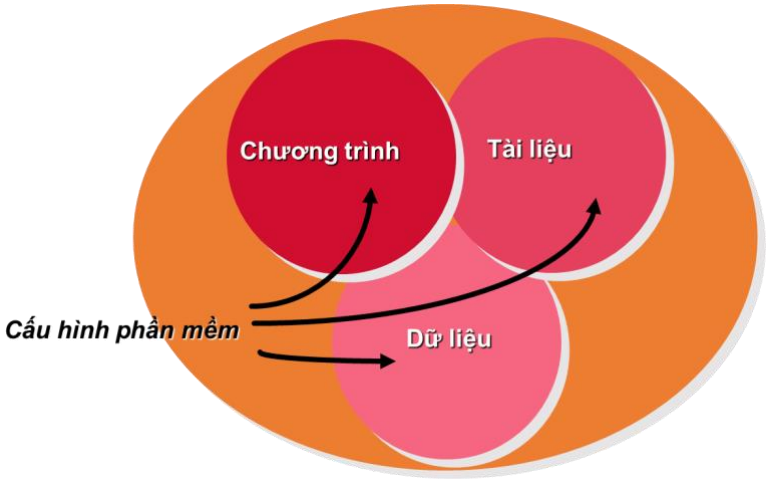
5.1.1 Phần mềm và sự thay đổi

a) Sự tiến hóa của phần mềm

Trong các bài học trước chúng ta đã được giới thiệu về mô hình thác nước, một mô hình phát triển phần mềm lý tưởng theo tính trình tự. Mô hình thác nước có thể phát huy tối đa hiệu quả khi sử dụng trong trường hợp các yêu cầu của phần mềm đã được xác định một cách rõ ràng và có tính ổn định cao. Chúng ta có thể liên hệ với việc "ngắm bắn" trong thực tế, việc bắn trúng mục tiêu cố định luôn dễ dàng hơn là mục tiêu đang di chuyển. Nếu chúng ta có thể kiểm soát và làm giảm tốc độ di chuyển này thì sẽ giúp tăng cơ hội bắn trúng mục tiêu. Quá trình phát triển phần mềm sẽ lý tưởng khi nó được bắt nguồn từ "các yêu cầu ổn định". Tuy nhiên, trong quá trình phát triển các hệ thống phần mềm thực tế, rất hiếm khi không có thay đổi. Các thay đổi có thể đến từ khách hàng hay người sử dụng khi họ thường xuyên đưa ra yêu cầu mới, thay đổi các yêu cầu đã nêu hoặc thậm chí "quên mất" các yêu cầu và chỉ nhớ đến chúng sau này khi tiếp nhận bản thiết kế. Các thay đổi sẽ dẫn đến sự nhầm lẫn và sai sót, đặc biệt trong các dự án phần mềm nơi đồng thời có nhiều thành viên tham gia thực hiện các tác vụ công nghệ phần mềm.

Không những thế, sự thay đổi còn xảy ra ngay trong các giai đoạn phát triển phần mềm thể hiện qua sự "tiến triển" hay "tiến hóa" của các sản phẩm phần mềm. Nhiều yếu tố

khác nhau được tạo ra trong suốt thời gian của dự án phần mềm. Các yếu tố này có thể được chia thành ba nhóm chính như trong hình 5.1: (1) chương trình máy tính (cả mã nguồn và mã dạng thực thi), (2) sản phẩm công việc mô tả chương trình máy tính (các tài liệu dành cho các bên liên quan khác nhau) và (3) dữ liệu hoặc nội dung (bên trong hoặc bên ngoài chương trình).



Hình 5-1: Cấu hình phần mềm

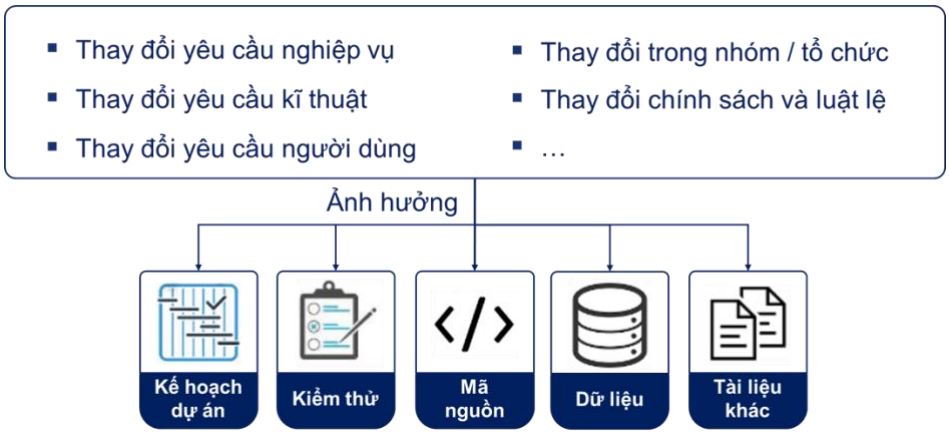
Các mục bao gồm tất cả thông tin được tạo ra như một phần của quy trình phần mềm được gọi chung là ***cấu hình phần mềm (software configuration)***. Các sản phẩm này không được hoàn thành trong một lần duy nhất mà chúng biến đổi và được hoàn thiện dần theo thời gian, từ đó hình thành nên nhiều phiên bản khác nhau. Sự tồn tại đồng thời của nhiều phiên bản cần được quản lý để đảm bảo sự đồng bộ. Bên cạnh đó, để đưa ra một sản phẩm không phải chỉ có một người thực hiện mà cần có sự cộng tác và phối hợp của các thành viên trong nhóm phần mềm. Chúng ta có thể so sánh sự phối hợp này giống như sự vận hành của một ban nhạc, trong đó không phải ai cũng có thể giữ được nhịp điệu. Đôi khi nó là nhằm mục đích tạo ấn tượng nghệ thuật, đôi khi lại là do nhầm lẫn. Trong cả hai trường hợp, người đó cần phải đồng bộ với phần còn lại của ban nhạc. Nếu mọi người cố gắng thích nghi với nhịp điệu của nhau thì cuối cùng sẽ trở nên hỗn loạn khi mọi người cố gắng bắt kịp nhịp của những người khác.

Trong thực tiễn phát triển phần mềm, khi các thay đổi xảy ra chúng ta có thể gặp những sự cố như: Một lỗi nào đó của phần mềm đã tốn công sửa chữa, bỗng nhiên xuất hiện trở lại. Một chức năng nào đó của phần mềm đã được phát triển và kiểm thử cẩn thận bỗng thất lạc hoặc biến mất. Một chương trình đã kiểm tra cẩn thận bỗng nhiên không

chạy được nữa. Sự thay đổi là một trong nhiều lý do khiến cho các dự án phần mềm đi đến thất bại. Khi các yêu cầu người dùng thay đổi, nhóm phần mềm cần điều chỉnh bản thiết kế. Sự điều chỉnh này dẫn đến thay đổi các hoạt động viết mã nguồn và kiểm thử. Sự hỗn loạn và lãng phí thời gian, công sức sẽ xảy ra nếu các thay đổi không được phân tích trước khi chúng thực hiện, được ghi nhận và kiểm tra lại sau khi chúng hoàn thành, được báo cáo cho những người cần biết hoặc được kiểm soát theo cách giúp cải thiện chất lượng và giảm sai sót. Một dự án phần mềm hiệu quả cần phải có chiến lược để giải quyết vấn đề "thay đổi" này.

b) Nguồn gốc của những thay đổi phần mềm

Nguồn gốc của những thay đổi phần mềm là gì? Câu trả lời cho câu hỏi này cũng đa dạng như chính những thay đổi đó.



Hình 5-2: Những thay đổi của phần mềm

Như hình minh họa 5.2, một số nguồn thay đổi có thể kể đến bao gồm:

- Những nghiệp vụ mới dẫn đến những thay đổi trong yêu cầu sản phẩm.
- Việc tổ chức lại hoặc tăng trưởng/thu hẹp quy mô kinh doanh gây ra những thay đổi về mức độ ưu tiên của dự án hoặc cơ cấu nhóm phần mềm.
- Những hạn chế về ngân sách hoặc lịch trình đòi hỏi xác định lại kế hoạch dự án.
- Thay đổi trong chính sách và luật lệ.
- …

c) Thay đổi và kiểm soát

Như vậy phần mềm có thể thay đổi liên tục và chúng ta không thể tránh khỏi sự thay đổi. Tuy nhiên chúng ta có thể đặt ra một kế hoạch giúp đối phó với những thay đổi và quản lý những thay đổi một cách hiệu quả. Tức là thay vì trốn tránh sự thay đổi, chúng

ta cần xây dựng chiến lược kiểm soát và quản lý những thay đổi tồn tại trong bất kỳ hoạt động phát triển phần mềm nào. Các thay đổi cần được kiểm soát, để xác định các thay đổi nào cần được thực hiện và thứ tự thực hiện. Các thay đổi cần được quản lý để theo viết việc thực hiện thay đổi và đảm bảo thay đổi được thực hiện đúng cách. Quản lý sự thay đổi là một phần trong quy trình quản lý dự án phần mềm. Quy trình quản lý đó được gọi là ***Quản lý cấu hình phần mềm - Software Configuration Management (SCM)***.

5.1.2 Quản lý cấu hình phần mềm

a) Khái niệm quản lý cấu hình phần mềm

Chúng ta có thể tìm thấy các mô tả về khái niệm quản lý cấu hình một cách trực tiếp hay gián tiếp trong các bộ tiêu chuẩn hoặc mô hình quy trình phần mềm.

- Các tiêu chuẩn IEEE Standard 828-2012 về kế hoạch quản lý cấu hình phần mềm và IEEE Standard 1042-2012 về hướng dẫn quản lý cấu hình phần mềm đưa ra khái niệm về quản lý cấu hình phần mềm bao gồm "*các nguyên tắc* và *kỹ thuật đánh giá* và *kiểm soát sự thay đổi* đối với các sản phẩm phần mềm trong và sau quá trình kỹ thuật phần mềm."
- Theo ISO 15504: Mục đích của quản lý cấu hình là để thiết lập và bảo đảm tính toàn vẹn của các sản phẩm trung gian cũng như sản phẩm sau cùng của một dự án phần mềm, xuyên suốt chu kỳ sống của dự án phần mềm đó.

Các khái niệm mặc dù được phát biểu khác biệt nhưng vẫn có những điểm chung cơ bản. Quản lý cấu hình là hoạt động bao trùm trong suốt tiến trình phần mềm được áp dụng cho tất cả các giai đoạn của tiến trình. Bởi vì những thay đổi có thể xảy ra vào bất cứ lúc nào, hoạt động quản lý cấu hình được phát triển để xác định thay đổi, kiểm soát thay đổi, đảm bảo rằng những thay đổi đó được thi hành một cách đúng đắn, và báo cáo những thay đổi đó cho những người liên quan. Cụ thể, quản lý cấu hình phần mềm là một quy trình áp dụng một cách tiếp cận nghiêm ngặt để đảm bảo:

- *Các chi tiết* trong hệ thống phần mềm đều được xác định và theo dõi
- *Các thay đổi* với các mục khác nhau được ghi lại và theo dõi
- *Tích hợp* thích hợp tất cả các mô-đun khác nhau

Nói một cách dễ hiểu hơn, điều này có nghĩa là chúng ta cần biết chúng ta phải phát triển *cái gì* (và ghi lại nó), sau đó chúng ta phải *xử lý những thay đổi theo cách nào đó*, chúng ta nên theo dõi mọi thứ *tiến triển như thế nào* và cuối cùng chúng ta phải *kiểm tra* rằng những gì đã phát triển đúng với mục tiêu đặt ra. Tiêu chuẩn IEEE Standard 828-2012 nêu rõ mục đích của quản lý cấu hình phần mềm là:

- Xác định và ghi lại các đặc tính vật lý và chức năng của bất kỳ sản phẩm, thành phần, kết quả hoặc dịch vụ nào
- Kiểm soát mọi thay đổi đối với các đặc điểm đó
- Ghi lại và báo cáo từng thay đổi cũng như tình trạng thực hiện của nó
- Hỗ trợ đánh giá sản phẩm, kết quả, dịch vụ hoặc thành phần để xác minh sự tuân thủ các yêu cầu

Quản lý cấu hình phần mềm có liên quan chặt chẽ đến hoạt động đảm bảo chất lượng phần mềm. Vì thông qua các báo cáo liên quan đến sự thay đổi có thể lượng hoá và đánh giá được qui trình và chất lượng của cả hệ thống phần mềm được xây dựng. Quản lý cấu hình phần mềm giúp kiểm soát sự phát triển và tính toàn vẹn của sản phẩm bằng cách xác định các thành phần, quản lý và kiểm soát sự thay đổi cũng như xác minh, ghi lại và báo cáo về thông tin cấu hình. Từ quan điểm của kỹ sư phần mềm, quy trình này tạo điều kiện thuận lợi cho các hoạt động triển khai thay đổi và phát triển. Nó giúp theo dõi và kiểm soát các thay đổi trong tất cả các khía cạnh của phát triển phần mềm, từ yêu cầu, thiết kế, mã hóa, kiểm thử và làm tài liệu.

b) Sự cần thiết của quản lý cấu hình phần mềm

Quản lý cấu hình tốt sẽ giúp giải quyết được nhiều khó khăn trong các dự án phần mềm, đặc biệt là trong các dự án có quy mô lớn.

- *Thay đổi hay cập nhật đồng thời*: Khi hai hoặc nhiều lập trình viên làm việc độc lập nhưng trên cùng một tệp tin mã nguồn hoặc mô-đun phần mềm, những thay đổi mà người này thực hiện có thể phá vỡ kết quả làm việc của người khác. SCM ngăn ngừa các lỗi có thể phát sinh từ các xung đột do thay đổi.
- *Chia sẻ kết quả (mã nguồn, tài liệu,...)*: Trong các hệ thống lớn, khi các kết quả hay chức năng chung bị thay đổi, tất cả những người liên quan phải được biết.

Không quản lý chia sẻ các kết quả chung tốt thì không đảm bảo tất cả những thay đổi sẽ được phổ biến hay thông báo cho những người liên quan.

- *Quản lý nhiều phiên bản*: Các sản phẩm phần mềm (chương trình, dữ liệu, tài liệu,…) được phát triển với nhiều phiên bản từ thấp đến cao. Tại một thời điểm có thể tồn tại đồng thời nhiều phiên bản, ví dụ một phiên bản đã được phát hành cho khách hàng sử dụng (release), một phiên bản khác đang được kiểm thử (test), và một phiên bản khác nữa đang trong quá trình phát triển, khi có một lỗi xảy ra, việc sửa lỗi phải đồng bộ giữa ba phiên bản này. Ngoài ra nếu có một lỗi do khách hàng phát hiện ra và thông báo, lỗi đó phải được sửa trong tất cả các phiên bản về sau.

5.1.3 Các khái niệm cơ bản trong quản lý cấu hình phần mềm

a) Mục cấu hình (Configuration Item - CI)

Khái niệm về mục cấu hình (Configuration Item - CI)

Mục cấu hình là tập hợp phần cứng, phần mềm hoặc cả hai, *được chỉ định để quản lý cấu hình* và được coi như *một thực thể duy nhất* trong quy trình quản lý cấu hình.

Từ khái niệm mục cấu hình tổng quát, *các mục cấu hình phần mềm (Software Configuration Item - SCI) không chỉ là các đoạn mã chương trình mà là tất cả các loại tài liệu cho sự phát triển phần mềm*.

Như vậy ngoài *mã nguồn*, nhiều mục khác cũng được SCM kiểm soát. Các hạng mục phần mềm có tiềm năng trở thành SCI bao gồm:

- Các kế hoạch,
- Tài liệu phân tích thiết kế,
- Trình điều khiển cho các trường hợp kiểm thử,
- Các tài liệu về cài đặt, bảo trì, vận hành và sử dụng phần mềm
- …

Tìm kiếm các mục cấu hình

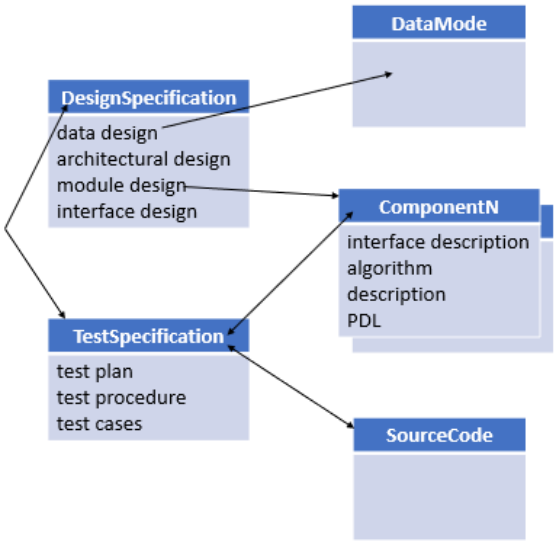
Lựa chọn SCI là một quá trình quan trọng trong đó phải đạt được sự cân bằng giữa việc cung cấp khả năng cho mục đích kiểm soát dự án và cung cấp số lượng hạng mục có

thể quản lý được. Vì các dự án lớn thường tạo ra hàng nghìn thực thể (tệp, tài liệu, ...) phải được xác định duy nhất. Trong đó không phải tất cả các thực thể đều cần được định cấu hình. Hai câu hỏi cần quan tâm khi tìm kiếm các mục cấu hình:

(1) Nên quản lý những gì? Câu trả lời giúp lựa chọn mục cấu hình phù hợp.

(2) Khi nào bắt đầu đặt một thực thể dưới sự kiểm soát cấu hình? Bắt đầu quá sớm dẫn đến sự "áp đặt" mà bắt đầu quá muộn dẫn đến hỗn loạn.

Một số thực thể phải được duy trì trong suốt thời gian tồn tại của phần mềm. Bao gồm giai đoạn khi phần mềm không còn được phát triển nhưng vẫn được sử dụng bởi khách hàng trong nhiều năm sau đó. Với thời gian quản lý dài, chúng ta cần xác định một lược đồ đặt tên thực thể để định danh cho các tài liệu có tên liên quan. Thực tế lựa chọn các mục cấu hình phù hợp là một kỹ năng cần thực hành. Chúng ta có thể sử dụng các kỹ thuật tương tự như mô hình hóa đối tượng để tìm các mục cấu hình. Hình 5.3 minh họa các đối tượng cấu hình phần mềm và quan hệ giữa chúng.



Hình 5-3: Các đối tượng cấu hình phần mềm

b) Đường cơ sở (Baseline)

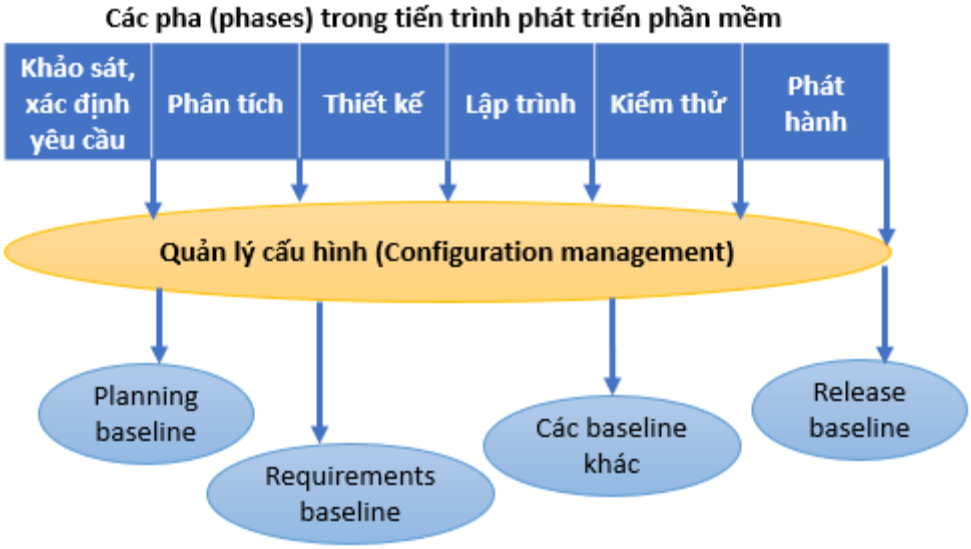
Đường cơ sở hay còn gọi là đường mốc giới (baseline), là đặc tả kỹ thuật hoặc sản phẩm đã được *xem xét* và *thống nhất chính thức*, sau đó được dùng như là một cơ sở để tiếp tục phát triển, và có thể thay đổi chỉ thông qua thủ tục kiểm soát thay đổi chính thức.

Đường cơ sở của phần mềm là một tập hợp các mục cấu hình phần mềm được chỉ định và cố định chính thức tại một thời điểm cụ thể trong vòng đời của phần mềm. Thuật

ngữ này cũng được sử dụng để chỉ một phiên bản cụ thể của hạng mục cấu hình phần mềm đã được thống nhất hay phê duyệt và chỉ có thể được thay đổi thông qua các thủ tục kiểm soát thay đổi chính thức.

Chúng ta có thể hiểu đơn giản, một baseline là một mốc quan trọng trong sự phát triển của phần mềm được đánh dấu bằng việc cung cấp một hoặc nhiều mục cấu hình phần mềm và sự chấp thuận của các mục cấu hình thu được thông qua đánh giá kỹ thuật chính thức. Trước baseline, một mục cấu hình có thể thay đổi nhanh chóng và không chính thức. Tại baseline các mục cấu hình phần mềm đã thống nhất hay phê duyệt được đưa vào cơ sở dữ liệu dự án. Sau baseline cần các thủ tục đặc biệt và chính thức để đánh giá và kiểm soát sự thay đổi cấu hình. Và mọi thay đổi phải được thông báo tới tất cả những người có liên quan.

Khi hệ thống được phát triển, một loạt các đường cơ sở được phát triển, thường là sau khi kết thúc quá trình xem xét hay xét duyệt (review). Baseline có thể được định nghĩa ở bất kỳ mức chi tiết nào, thông thường baseline được tiến hành tại điểm kết thúc của mỗi pha (phase) hay các "mốc" quan trọng trong dự án như hình minh họa 5.4.



Hình 5-4: Các baseline trong tiến trình phần mềm

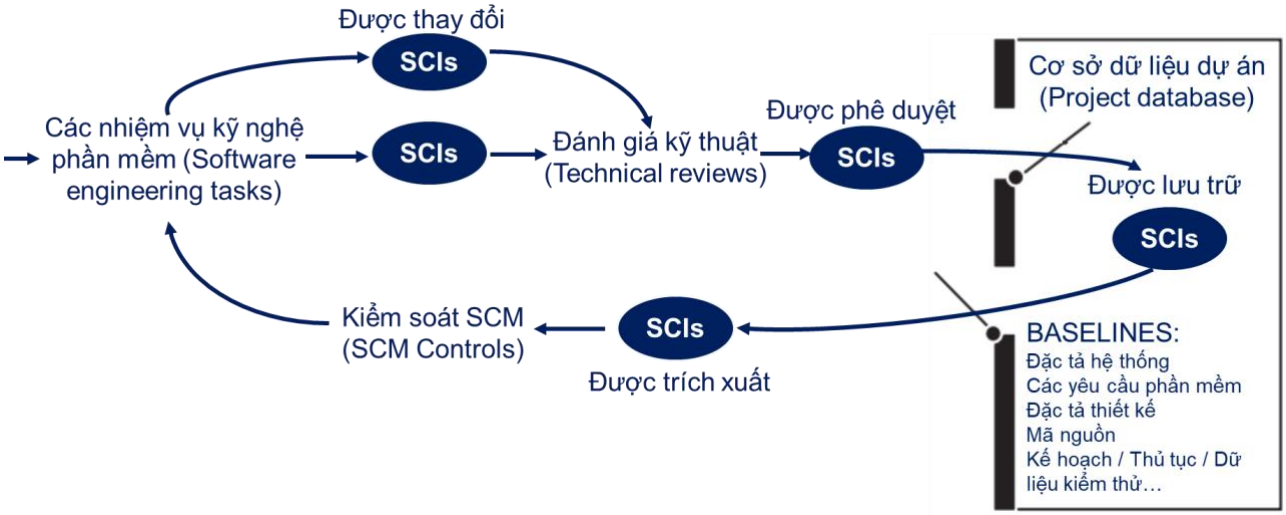
Ví dụ:

- *Baseline A*: được mô tả là API của một chương trình được xác định hoàn toàn, nhưng phần thân của các phương thức trống. đặc tả API được viết thành tài liệu.

Khi tất cả các phần của đặc tả đã được xem xét, sửa cho đúng và sau đó được phê chuẩn thì bản đặc tả API trở thành một baseline.

- *Baseline B*: là cột mốc đánh dấu việc hiện thực hóa các đặc tả API ở trên. Trong đó tất cả các phương thức truy cập dữ liệu được thực hiện và kiểm thử. Lúc này lập trình giao diện đồ họa người dùng có thể bắt đầu.
- *Baseline C*: được định nghĩa là Giao diện đồ họa người dùng được triển khai hoàn thành, và giai đoạn thử nghiệm có thể bắt đầu.

Các mục cấu hình phần mềm (SCI) xác định đường cơ sở (baseline) và cơ sở dữ liệu dự án. Sự tiến triển của các sự kiện dẫn đến đường cơ sở cũng được minh họa như trong trong Hình 5.5. Khi các nhà phát triển thực hiện các nhiệm vụ công nghệ phần mềm tạo ra một hoặc nhiều mục cấu hình phần mềm (SCIs). Sau khi các mục này được xem xét và phê duyệt, chúng sẽ được đặt trong cơ sở dữ liệu dự án (còn được gọi là thư viện dự án hoặc kho phần mềm). Trong cơ sở dữ liệu dự án, tập hợp các mục cấu hình đã được phê duyệt hình thành các baseline.



Hình 5-5: Các mục cấu hình phần mềm (SCIs) và cơ sở dữ liệu dự án

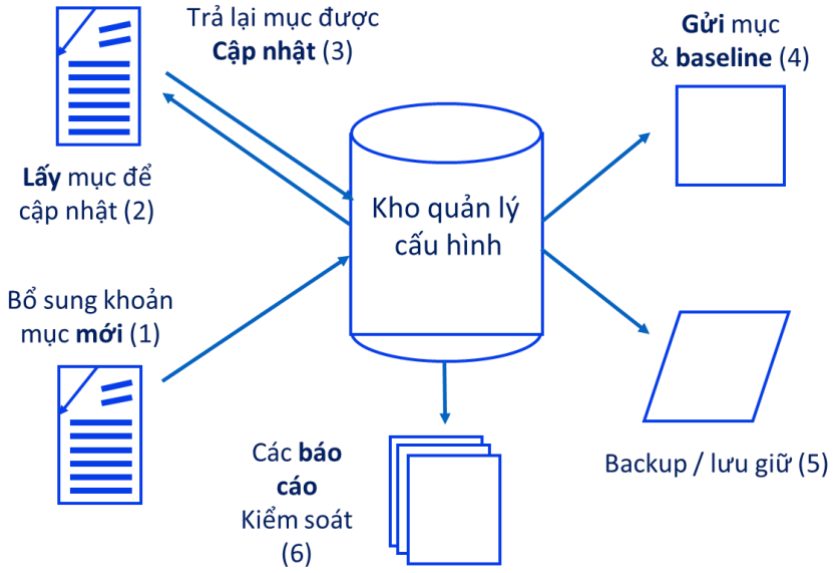
Khi một thành viên của nhóm phần mềm muốn thực hiện sửa đổi đối với mục cấu hình cơ sở, nó sẽ được sao chép từ cơ sở dữ liệu dự án vào không gian làm việc riêng của nhà phát triển đó để thực hiện các thay đổi. Tuy nhiên, SCI được trích xuất này chỉ có thể được sửa đổi nếu tuân thủ các biện pháp kiểm soát quản lý cấu hình phần mềm.

c) Kho lưu trữ SCM (SCM Repository)

Kho lưu trữ cấu hình phần mềm là một tập các cơ chế hoạt động và cấu trúc dữ liệu cho phép một nhóm phát triển phần mềm có thể quản lý thay đổi, phát triển, bảo trì phần mềm một cách hiệu quả. Về cơ bản kho lưu trữ SCM có thể xem là một cơ sở dữ liệu tập trung đảm bảo chia sẻ thông tin nhất quán và chính xác giữa các bên liên quan, đồng thời quản lý các quy trình kiểm soát phiên bản, kiểm soát thay đổi và kiểm soát phát hành. Một kho lưu trữ (repository) có các chức năng:

- Đảm bảo toàn vẹn dữ liệu lưu trữ
- Chia sẻ thông tin cho làm việc nhóm
- Tích hợp công cụ
- Tích hợp dữ liệu
- Thực thi phương pháp luận
- Tiêu chuẩn hoá tài liệu

Các nhiệm vụ của kho lưu trữ SCM được minh họa trong hình 5.6.



Hình 5-6: Kho lưu trữ SCM và các hoạt động

d) Thư mục SCM (SCM Directory)

Các thay đổi bình thường diễn ra trong suốt vòng đời của phần mềm được quản lý thông qua một cơ chế thư viện (library mechanism) hay cấu trúc thư mục (directory structure) được định nghĩa theo tiêu chuẩn IEEE Standard 1042-2012. Các khái niệm bao gồm:

- *Programmer's Directory (IEEE: Dynamic Library)*: Thư viện để chứa các thực thể phần mềm mới được tạo hoặc sửa đổi. Không gian làm việc của lập trình viên chỉ do lập trình viên kiểm soát.
- *Master Directory (IEEE: Controlled Library)*: Quản lý (các) baseline và kiểm soát các thay đổi được thực hiện đối với chúng. Mục nhập được kiểm soát, thường sau khi được xác minh. Các thay đổi phải được cho phép.
- *Software Repository (IEEE: Static Library)*: Lưu trữ cho các baseline khác nhau được phát hành để sử dụng chung. Các bản sao của các baseline này có thể được cung cấp cho các tổ chức yêu cầu.

e) Phiên bản, bản sửa đổi và bản phát hành (Version, Revision and Release)

- *Phiên bản (Version)* là một thực thể mới của một đối tượng cấu hình sau khi đã qua một hoặc nhiều lần xem xét và thay đổi. Phiên bản có thể một bản phát hành ban đầu hoặc tái phát hành một mục cấu hình được liên kết với một bản biên dịch hoặc biên dịch lại hoàn chỉnh của mục đó. Các phiên bản khác nhau phản ánh chức năng khác nhau. Một phiên bản nói dễ hiểu là một thực thể mới của một mục cấu hình phần mềm sau khi đã qua một hoặc nhiều lần xem xét và thay đổi.
- *Bản sửa đổi (Revision)* chứa các thay đổi thành phiên bản chỉ sửa các lỗi trong thiết kế hay mã nguồn, nhưng không ảnh hưởng đến chức năng đã được lập thành tài liệu. Bản sửa đổi thể hiện sự thay đổi đối với nội dung của mục cấu hình hoặc sửa đổi một phần sao cho phần đó vẫn có thể thay thế được hay đảm bảo sự tương thích với biến thể trước đó của nó.
- *Bản phát hành (Release)* mô tả việc phân phối chính thức phiên bản đã được phê duyệt. Quá trình phát triển sản phẩm thường qua nhiều lần tích hợp, kết quả của mỗi lần tích hợp là một bản "build" được hình thành, trong rất nhiều bản "build" đó, một số bản đáp ứng yêu cầu đã định hoặc lập kế hoạch trước (theo yêu cầu khách hàng), sẽ được gửi cho khách hàng để kiểm tra hoặc đánh giá. Các bản "build" này sẽ được gọi là "release", và công việc tạo ra và phân phối các bản

release sẽ được gọi là công việc "release". Sản phẩm sau cùng cũng là một bản release, đôi khi còn gọi là "final release".

5.2 Quy trình quản lý cấu hình phần mềm

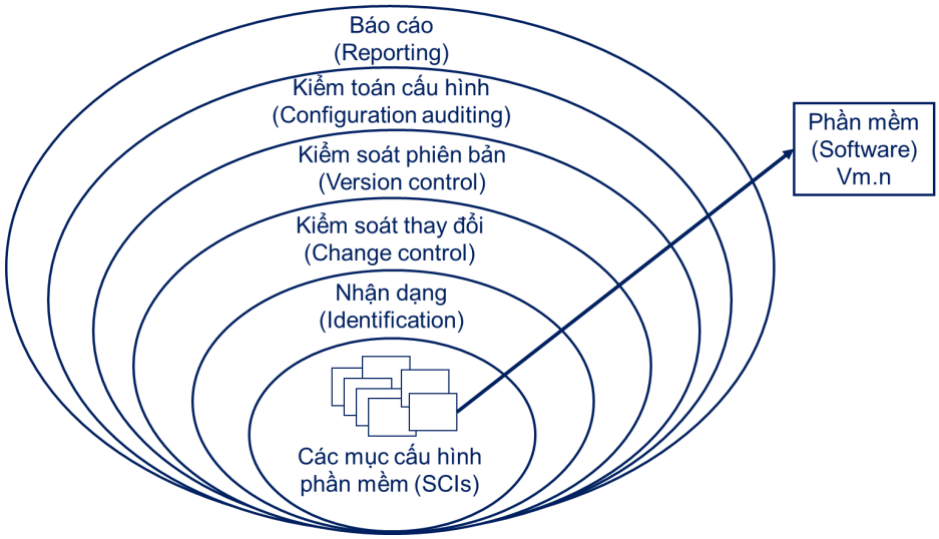
Quy trình quản lý cấu hình phần mềm xác định một trình tự các hoạt động phải được thực hiện dưới sự hỗ trợ của các cơ chế quản lý cấu hình. Quy trình quản lý cấu hình phần mềm xác định một loạt các nhiệm vụ bao gồm bốn mục tiêu chính:

- (1) Xác định tất cả các mục cấu hình phần mềm,
- (2) Quản lý các thay đổi đối với một hoặc nhiều mục này,
- (3) Tạo điều kiện thuận lợi cho việc xây dựng các phiên bản khác nhau của một ứng dụng và
- (4) Đảm bảo chất lượng phần mềm được duy trì khi cấu hình phát triển theo thời gian.

Một quy trình đạt được những mục tiêu này không nhất thiết phải quan liêu và nặng nề, nhưng nó phải có các đặc trưng phù hợp nhằm cho phép nhóm phần mềm trả lời cho một tập hợp các câu hỏi phức tạp.

- Làm thế nào để tổ chức và quản lý được nhiều phiên bản (Version) của phần mềm sao cho những thay đổi đem lại sự hiệu quả?
- Làm sao để tổ chức kiểm soát toàn bộ thay đổi trước và sau khi giao phần mềm cho người sử dụng?
- Ai sẽ chịu trách nhiệm về việc chấp thuận và thiết lập thứ tự ưu tiên cho các thay đổi khi có nhiều thay đổi cùng được đề xuất?
- Làm thế nào đảm bảo được việc thay đổi đã thực hiện đúng?
- Dùng cơ chế nào để đánh giá các thay đổi khác nhau?

Câu trả lời cho những câu hỏi này dẫn chúng ta đến năm nhiệm vụ trong quy trình quản lý cấu hình phần mềm như hình minh họa 5.7, bao gồm: *nhận dạng, kiểm soát phiên bản, kiểm soát thay đổi, kiểm toán cấu hình và báo cáo trạng thái cấu hình.*



Hình 5-7: Các lớp trong quy trình quản lý cấu hình phần mềm

Nhìn vào hình 5.7, chúng ta thấy các nhiệm vụ của quy trình quản lý cấu hình phần mềm được biểu diễn dưới dạng các lớp đồng tâm. Các mục cấu hình phần mềm (SCIs) sẽ đi qua các lớp này trong suốt thời gian chúng tồn tại và được sử dụng. Cuối cùng, trở thành một phần cấu hình phần mềm trong một hoặc nhiều phiên bản của phần mềm hoặc hệ thống. Khi mục cấu hình phần mềm di chuyển qua một lớp, các hành động được ngụ ý trong mỗi nhiệm vụ có thể được áp dụng tùy chọn. Ví dụ: khi một SCI mới được tạo, nó phải được nhận dạng. Tuy nhiên, trong các pha phát triển tiếp theo nếu không có thay đổi nào được thực hiện đối với mục cấu hình này thì lớp kiểm soát thay đổi sẽ không được áp dụng. Một SCI được gán cho một phiên bản cụ thể của phần mềm cần thông qua các cơ chế kiểm soát phiên bản. Bản ghi SCI (tên, ngày tạo, chỉ định phiên bản, v.v.) được lưu giữ cho mục đích kiểm toán cấu hình và báo cáo cho những người cần biết.

Các nhiệm vụ của quản lý cấu hình phần mềm đem lại những ích lợi sau:

- Cung cấp một kho chứa an toàn đối với các kết quả bàn giao. Quản lý được các phiên bản khác nhau của phần mềm và Cung cấp cho người phát triển phiên bản mới nhất của phần mềm và Tạo sự dễ dàng cho phép người phát triển có thể truy cập lại các phiên bản cũ trước đó thông qua hồ sơ hay các báo cáo được thiết lập
- Cho phép việc kiểm soát và tiết lộ có nguyên tắc các kết quả bàn giao thông qua vòng đời của nó, với đầy đủ các dấu tích lịch sử, đảm bảo phiên bản đúng và cập nhật, đã được kiểm tra và phát hành

- Kiểm soát thay đổi cuả các kết quả bàn giao, đảm bảo các kết quả này được lưu theo đúng thứ tự .
- Cung cấp việc lập báo cáo về hiện trạng của các kết quả bàn giao và những thay đổi của chúng. Điều này cho phép kiểm soát được thực trạng của các kết quả bàn giao và mối quan hệ qua lại lẫn nhau giữa các kết quả này.

5.2.1 Nhận dạng mục cấu hình (Configuration item identification)

Nhận dạng mục cấu hình hay *định danh mục cấu hình* là một trong những hoạt động nền tảng của quản lý cấu hình. Được thực hiện bằng việc *mô hình hóa hệ thống như một tập hợp các thành phần / đối tượng đang phát triển*.

Mục đích của nhận dạng là để xác định tính duy nhất của một mục cấu hình, cũng như mối quan hệ giữa nó với các mục cấu hình khác. Nó bao gồm việc mô tả tên, đánh số, đánh dấu đặc trưng, giúp nhận biết và phân biệt mục các cấu hình này với nhau. Để kiểm soát và quản lý các mục cấu hình phần mềm, có thể được tổ chức theo cách tiếp cận của phương pháp hướng đối tượng, bao gồm hai loại đối tượng: đối tượng cơ bản và đối tượng tổng hợp.

- *Đối tượng cơ bản* là một đơn vị thông tin được tạo ra trong quá trình phân tích, thiết kế, viết mã hoặc kiểm tra. Ví dụ: một đối tượng cơ bản có thể là một phần của đặc tả yêu cầu, một phần của mô hình thiết kế, mã nguồn của một thành phần hoặc một bộ trường hợp kiểm thử được sử dụng để thực thi mã.
- *Đối tượng tổng hợp* là một tập hợp các đối tượng cơ bản và các đối tượng tổng hợp khác. Ví dụ: DesignSpecification là một đối tượng tổng hợp. Về mặt khái niệm, nó có thể được xem dưới dạng danh sách các con trỏ được đặt tên (được xác định) chỉ định các đối tượng tổng hợp như ArchitecturalModel và DataModel và các đối tượng cơ bản như ComponentN và UMLDiagramN.

Mỗi mục cấu hình có một tập hợp các đặc trưng riêng biệt giúp xác định nó một cách duy nhất: tên, mô tả, danh sách tài nguyên và "hiện thực hóa của đối tượng".

- *Tên đối tượng*: dùng để định danh và đại diện cho đối tượng. Mỗi mục cấu hình phải có một định danh duy nhất, dạng thường thấy là:

<ProjectId>_<ItemId>_<Version>

Ví dụ: chuỗi định danh *AIM2023_ClassDiag_1.1.2_draft_A* cho biết:

Số ID của dự án: *AIM2023*

Số ID của mục cấu hình: *ClassDiag*

Số phiên bản: *1.1.2_draft_A*

- *Mô tả đối tượng phần mềm* bằng một danh sách các khoản mục, dữ liệu liên quan đến đối tượng:
    - *Kiểu mục cấu hình phần mềm*: là tài liệu, chương trình hay dữ liệu.
    - Xác định vị trí của đối tượng nằm trong giai đoạn nào, phần nào của dự án tổng thể.
    - Thông tin thay đổi hay thông tin liên quan đến phiên bản.
- *Danh sách các nguồn lực*: là tất cả các thực thể được cung cấp, xử lý, tham khảo và các thức khác mà đối tượng cần đến.

Nhận dạng đối tượng cấu hình cũng có thể xem xét các mối quan hệ tồn tại giữa các đối tượng được đặt tên. Mối quan hệ giữa các đối tượng là quan hệ bộ phận hay toàn bộ, thể hiện thông qua đồ thị các đối tượng. Việc theo dõi các mối quan hệ này cũng rất quan trọng để hỗ trợ truy xuất nguồn gốc. Ví dụ một ký pháp đơn giản:

Class diagram <*part-of*> requirements model;

Requirements model <*part-of*> requirements specification;

Chúng ta sẽ tạo ra một cấu trúc phân cấp của các SCIs.

5.2.2 Kiểm soát thay đổi (Change control)

Kiểm soát thay đổi (Change control) nhằm đảm bảo rằng sự phát triển của hệ thống là một *quá trình được quản lý* và ưu tiên dành cho những thay đổi *cấp bách nhất* và *tiết kiệm chi phí*. Việc kiểm soát thay đổi đóng vai trò quan trọng trong quá trình phát triển phần mềm. Ví dụ chỉ cần một sai sót nhỏ trong mã nguồn có thể tạo ra ảnh hưởng cho sự vận hành của phần mềm. Nhưng ở một tình huống khác, một sự thay đổi nhỏ trong mã nguồn lại có thể khắc phục một lỗi lớn hoặc tạo ra những khả năng mới cho sản phẩm. Quá trình kiểm soát thay đổi với nhiều thủ tục quá chặt chẽ có nguy cơ tạo ra sự cứng nhắc, quan liêu trong điều phối sự thay đổi. Nếu không tổ chức và kiểm tra tốt,

sự kiểm soát thay đổi có thể dẫn đến việc cản trở tiến trình phát triển của cả dự án. Kiểm soát quá nhiều gây ra khó khăn cho sự phát triển và hạn chế sự sáng tạo. Ngược lại, nếu kiểm soát quá ít, chúng ta sẽ dẫn đến tình trạng thiếu nhất quán, hỗn độn trong toàn bộ hệ thống và gây ra sự thất bại của phần mềm. Chúng ta cần xác định điểm cân bằng phù hợp. Hầu hết các quy trình phát triển phần mềm đều có cơ chế kiểm soát thay đổi, tạo ra một số tầng kiểm soát khác nhau trên những mức khác nhau trong dự án để việc kiểm soát thay đổi có thể dễ dàng được thực hiện và quản lý từ mức thấp đến mức cao của toàn thể dự án.

Vì vậy quá trình quản lý thay đổi liên quan đến việc *phân tích chi phí và lợi ích của những thay đổi được đề xuất*, *phê duyệt những thay đổi đáng giá* và *theo dõi những thành phần nào trong hệ thống đã được thay đổi*. Mức độ phức tạp của quy trình quản lý thay đổi khác biệt theo từng dự án. Các dự án nhỏ có thể thực hiện các yêu cầu thay đổi một cách không chính thức và nhanh chóng trong khi các dự án phức tạp yêu cầu các biểu mẫu thay đổi chi tiết và sự chấp thuận chính thức của những người quản lý cấp cao khác.

Phương thức được sử dụng để kiểm soát thay đổi là phối hợp cả các qui trình thủ tục, cả con người và các công cụ tự động để tạo ra một cơ chế kiểm soát sự thay đổi hiện quả. Các bước cơ bản trong kiểm soát thay đổi gồm:

1. Nghiên cứu, phân tích: Khi một *yêu cầu thay đổi* được đệ trình cần phải được xác định lại giá trị của yêu cầu thay đổi đó nhằm:
   - Sử dụng kỹ thuật thích hợp để thực hiện thay đổi đó.
   - Xác định các hiệu ứng phụ có thể ảnh hưởng lên tổng thể các đối tượng cấu hình khác và lên các chức năng hệ thống, lên chi phí dành cho dự án có thể diễn ra khi thực hiện sự thay đổi này.

2. Phê chuẩn hoặc không phê chuẩn: Kết quả đánh giá về sự thay đổi được báo cáo cho người có thẩm quyền kiểm soát thay đổi - người có quyền cao nhất trong việc kiểm soát tình trạng và quyết định sự thay đổi đó có được thực hiện hay không.

3. Thực hiện việc thay đổi: Khi một thay đổi được chấp thuận, một lệnh thay đổi kỹ thuật được tạo ra. Lệnh này mô tả các đổi thay cần phải làm, các ràng buộc cần phải

tuân thủ, các tiêu chuẩn rà soát và kiểm toán sẽ được thực hiện sau khi thực hiện thay đổi.

4. Kiểm tra việc thay đổi: đánh giá được ảnh hưởng của thay đổi đối với các yếu tố bên ngoài các khoản mục cấu hình phần mềm.

5. Xác lập baseline mới

5.2.3 Kiểm soát phiên bản (Version control)

Trước hết, chúng ta nhắc lại khái niệm về phiên bản, đây là một thực thể mới của một đối tượng cấu hình sau khi đã qua một hoặc nhiều lần xem xét và thay đổi. Kiểm soát phiên bản hay quản lý phiên bản là tổ hợp các thủ tục và các công cụ để quản lý các phiên bản khác nhau của các đối tượng cấu hình đã được tạo ra trong quy trình xây dựng phần mềm.

Các hệ thống kiểm soát phiên bản bao gồm bốn khả năng chính:

- (1) Cơ sở dữ liệu dự án (kho lưu trữ) lưu trữ tất cả các đối tượng cấu hình có liên quan,
- (2) Khả năng quản lý phiên bản lưu trữ của các đối tượng cấu hình (hoặc cho phép bất kỳ phiên bản nào được xây dựng bằng cách sử dụng những điểm khác biệt so với các phiên bản trước đó),
- (3) Một công cụ (make facility) cho phép thu thập tất cả các đối tượng cấu hình có liên quan và xây dựng một phiên bản cụ thể của phần mềm.
- (4) Khả năng theo dõi vấn đề (còn gọi là theo dõi lỗi) cho phép ghi lại và theo dõi trạng thái của tất cả các vấn đề còn tồn tại liên quan đến từng đối tượng cấu hình.

5.2.4 Kiểm toán cấu hình (Configuration audit)

Mục đích của việc kiểm soát phiên bản, kiểm soát sự thay đổi giúp người phát triển dự án duy trì được trật tự, tránh được tình trạnh hỗn độn trong toàn bộ hệ thống phần mềm. Tuy nhiên, ngay cả các cơ chế kiểm soát thành công nhất cũng chỉ theo dõi được sự thay đổi cho đến khi một đối tượng cấu hình kỹ thuật được sinh ra thay thế cho cấu hình đối tượng trước đó. Như vậy, làm thế nào để nhóm phần mềm có thể đảm bảo

rằng thay đổi đã được thực hiện đúng cách? Câu trả lời gồm hai hoạt động: (1) đánh giá / xét duyệt kỹ thuật (technical reviews) và (2) kiểm toán cấu hình phần mềm (software configuration audit).

- *Xét duyệt kỹ thuật* tập trung vào tính đúng đắn về mặt kỹ thuật của đối tượng cấu hình đã được sửa đổi. Người đánh giá xem xét SCI để xác định tính nhất quán với các SCI khác, những vấn đề còn thiếu sót hoặc các tác dụng phụ tiềm ẩn.
- *Kiểm toán cấu hình phần mềm* bổ sung cho việc xem xét kỹ thuật bằng cách đánh giá đối tượng cấu hình về các đặc điểm thường không được xem xét trong quá trình xét duyệt kỹ thuật.

Kiểm toán trả lời các câu hỏi sau:

- 1. Thay đổi được chỉ định đã được thực hiện chưa? Có bất kỳ sửa đổi bổ sung nào được đưa vào không?
- 2. Đã tiến hành đánh giá tính đúng đắn về mặt kỹ thuật chưa?
- 3. Quy trình phần mềm có được tuân thủ và các tiêu chuẩn kỹ thuật phần mềm có được áp dụng đúng cách không?
- 4. Sự thay đổi có được "làm nổi bật / highlighted" trong SCI không? Ngày thay đổi và tác giả thay đổi đã được chỉ định chưa? Các thuộc tính của đối tượng cấu hình có phản ánh sự thay đổi không?
- 5. Các quy trình SCM về ghi nhận và báo cáo thay đổi có được tuân thủ không?
- 6. Tất cả các SCI liên quan đã được cập nhật đúng cách chưa?

5.2.5 Báo cáo trạng thái cấu hình (Configuration status reporting)

Báo cáo tình trạng cấu hình là một nhiệm vụ trong quản lý cấu hình phần mềm, việc thiết lập báo cáo hiện trạng giúp trả lời những câu hỏi sau:

- Cái gì đã xảy ra?
- Ai làm? Ai đã thực hiện?
- Việc thay đổi đó xảy ra khi nào?
- Thay đổi đó có còn ảnh hưởng nào khác nữa không?

Thông tin báo cáo tình trạng cấu hình gắn chặt với các nhiệm vụ trong phần kiểm soát thay đổi. Mỗi khi một khoản mục cấu hình phần mềm được ấn định mới hoặc được cập nhật các thay đổi thì một nội dung (Entry) sẽ được tạo ra. Mỗi khi một khoản mục cấu hình phần mềm được chấp thuận bởi người có thẩm quyền thay đổi cấu hình (có một mệnh lệnh thay đổi kỹ thuật được tạo ra) thì nội dung cũng được tạo ra.

Các báo cáo tình trạng cấu hình có thể được đặt trên một cơ sở dữ liệu trực tuyến sao cho những người phát triển phần mềm hay bảo trì có thể truy cập các thông tin đã được thay đổi. Báo cáo tình trạng cấu hình phần mềm đóng một vai trò cốt lõi trong thành công của một dự án phát triển phần mềm. Các báo cáo này cũng giúp cải thiện giao tiếp giữa các thành viên và giảm bớt sự lãng phí về mặt thời gian và nhân lực khi cùng một sự thay đổi đối tượng cấu hình lại được thực hiện đi thực hiện lại nhiều lần bởi nhiều nhóm khác nhau.

5.2.6 Các vai trò trong quy trình quản lý cấu hình phần mềm

- *Người quản lý cấu hình*: Chịu trách nhiệm xác định các mục cấu hình (configuration items – CI). Người quản lý cấu hình cũng có thể chịu trách nhiệm xác định các thủ tục để tạo các sự tăng trưởng và các bản phát hành.
- *Thành viên ban kiểm soát thay đổi*: Chịu trách nhiệm phê duyệt hoặc từ chối các yêu cầu thay đổi.
- *Lập trình viên*: Tạo các thay đổi được kích hoạt bởi các yêu cầu. Nhà phát triển kiểm tra các thay đổi và giải quyết xung đột.
- *Kiểm soát viên*: Chịu trách nhiệm về việc lựa chọn và đánh giá các thay đổi để phát hành và đảm bảo tính nhất quán và đầy đủ của bản phát hành.

5.3 Quản lý phiên bản

5.3.1 Khái niệm quản lý phiên bản

Có nhiều định nghĩa và cách hiểu khác nhau về quản lý phiên bản (Version management) hay kiểm soát phiên bản (Version control). Hiểu đơn giản Kiểm soát phiên bản là tổ hợp các thủ tục và các công cụ để quản lý các phiên bản khác nhau của

các đối tượng cấu hình đã được tạo ra trong qui trình phát triển phần mềm và hệ thống mà các thành phần này được sử dụng.

Một phiên bản nói dễ hiểu là một thực thể mới của một mục cấu hình sau khi đã qua một hoặc nhiều lần xem xét và thay đổi. Phiên bản có thể có thể được gán nhãn (label) để đánh dấu nhằm biểu thị một "mốc" quan trọng trong tiến trình phát triển của mục cấu hình phần mềm đó. Kiểm soát phiên bản giúp bảo tồn các phiên bản cũ của các mục cấu hình và đảm bảo các thay đổi do các nhà phát triển khác nhau thực hiện đối với các phiên bản không ảnh hưởng lẫn nhau.

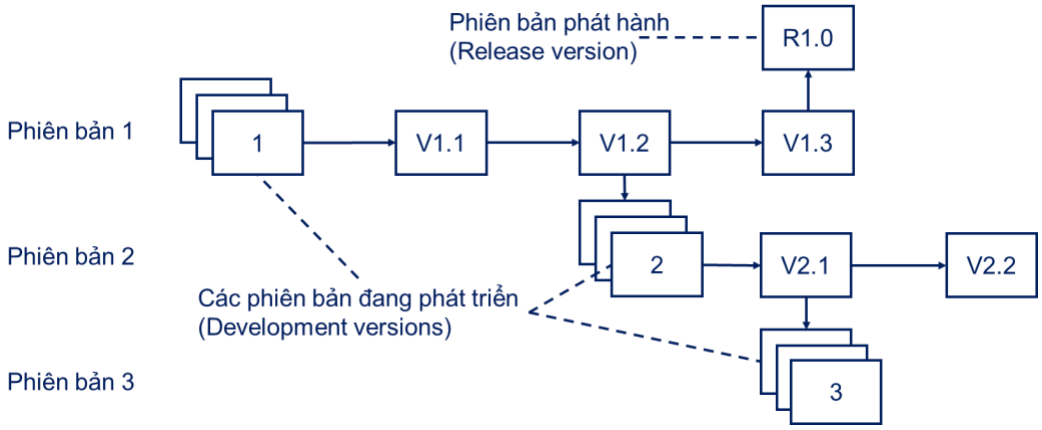
Quản lý phiên bản gắn liền với quản lý baseline, bao gồm:

- (1) Chọn các mục cấu hình phần mềm cho mỗi loại baseline
- (2) Tiến hành "ghim" baseline tại thời điểm sau khi các thay đổi đã được chấp thuận và phê duyệt.

Trong quá trình phát triển tồn tại một số phiên bản của phần mềm ở các giai đoạn khác nhau. Ví dụ:

- *Phiên bản phần mềm đang phát triển* (trong giai đoạn này các chức năng mới của phần mềm sẽ được xây dựng);
- *Phiên bản kiểm thử*, những thay đổi được thực hiện ở giai đoạn này là sửa các lỗi được phát hiện, cải thiện hiệu suất, sự ổn định,... của phần mềm
- *Phiên bản phát hành cho người dùng*: một phiên bản phần mềm vận hành ổn định, khách hàng có thể gửi báo cáo lỗi và yêu cầu mới

Mỗi phiên bản phần mềm này được hình thành từ các phiên bản khác nhau của các mục cấu hình phần mềm. Quản lý phiên bản gắn liền với quản lý baseline, chỉ định các phiên bản thành phần được bao gồm trong hệ thống cụ thể.



Hình 5-8: Phát triển phần mềm nhiều phiên bản

Hình 5.8 minh họa tình huống trong đó ba phiên bản của một phần mềm cùng tồn tại.

- 1. Phiên bản phát triển 1.3 của phần mềm đã hoành thành các chức năng cũng như sửa các lỗi được phát hiện khi kiểm thử. Phiên bản này được đóng gói và phân phối cho khách hành sử dụng. Đây là phiên bản phát hành đầu tiên của hệ thống (R1.0).
- 2. Phiên bản 2.2 đang được kiểm thử nhằm trở thành phiên bản phát hành 2.0 của phần mềm. Không có tính năng mới nào được thêm vào ở giai đoạn này.
- 3. Phiên bản 3 là một phiên bản phát triển trong đó các tính năng mới được thêm vào để đáp ứng yêu cầu thay đổi từ khách hàng. Mục tiêu nhằm đạt được phiên bản phát hành 3.0.

Các hệ thống kiểm soát phiên bản sử dụng *kho lưu trữ dự án (project repository)* và *không gian làm việc riêng (private workspace)* để hỗ trợ phát triển độc lập, không can thiệp lẫn nhau.

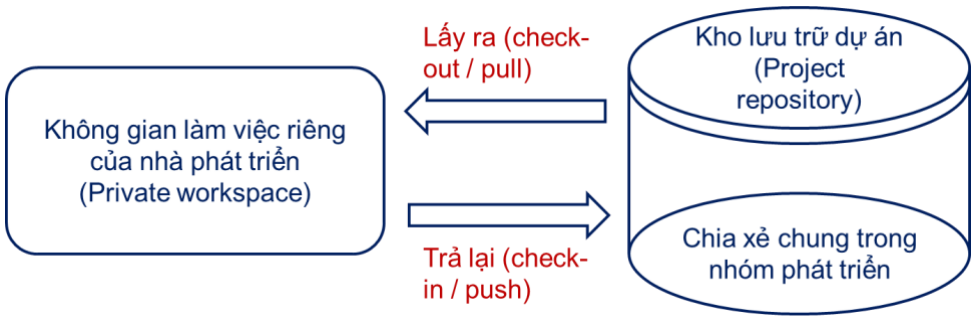
- *Kho lưu trữ dự án (project repository)* duy trì *phiên bản "chính"* của tất cả các thành phần phần mềm. Nó được sử dụng để tạo đường cơ sở cho việc xây dựng hệ thống.
- *Không gian làm việc riêng (private workspace)* chứa *bản sao* của các thành phần từ kho lưu trữ. Nhà phát triển làm việc trên các bản sao này.

Các công cụ kiểm soát phiên bản sử dụng các thủ tục *đưa-vào/tách-ra (check-in/check-out)* để kiểm soát phiên bản như hình 5.9.

- *Thao tác check-out / pull*: nhà phát triển *sao chép* những thành phần cần sử dụng hoặc sửa đổi từ kho lưu trữ vào không gian làm việc riêng.
- *Thao tác check-in / push*: nhà phát triển hoàn thành các thay đổi của mình, các thành phần đã thay đổi sẽ được *trả lại* vào kho lưu trữ.

Ý tưởng cơ bản của các thủ tục này như sau: một mục cấu hình được lưu trữ trong kho lưu trữ dự án là đang ở trong một môi trường có kiểm soát, tức là trạng thái chia xẻ chung mà những người khác vẫn có thể sử dụng được nó. Các thay đổi trên mục cấu hình cần sự cho phép (authorization), bởi vì có thể người khác đang sử dụng nó.

Để thực hiện thay đổi, nhà phát triển phải tách (check-out) mục cấu hình ra khỏi môi trường kiểm soát này. Nghĩa là cần tạo ra một bản sao (copy) của mục cấu hình để khi thực hiện thay đổi sẽ không làm phá hủy phiên bản trước hiện tại.



Hình 5-9: Thao tác với kho lưu trữ dự án và không gian làm việc

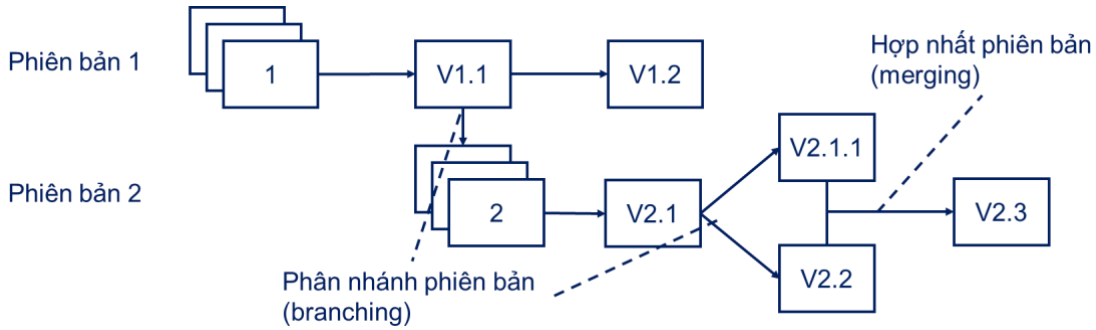
Bản sao này được lưu vào không gian làm việc riêng của nhà phát triển. Sau khi các thay đổi đã được thực hiện, một phiên bản mới của mục cấu hình được tạo ra và nó cần được phản ánh trả về (check-in) môi trường có kiểm soát để những người khác cũng nhận được các thay đổi của phiên bản mới.

Codeline là một chuỗi các phiên bản của một thành phần phần mềm. Trong đó, các phiên bản mới hơn trong chuỗi bắt nguồn từ các phiên bản trước đó. Từ các phiên bản trong codeline sẽ được lựa chọn để tạo baseline, từ là chỉ định các phiên bản thành phần (đã được phê duyệt) được bao gồm trong hệ thống và xác định các thư viện được sử dụng, các tệp cấu hình và những thông tin khác.

Hệ quả của sự phát triển độc lập của các lập trình viên trên cùng một thành phần tạo ra sự *phân nhánh phiên bản (branching)*. Thay vì một chuỗi các phiên bản tuyến tính phản ánh những thay đổi của thành phần theo thời gian, có thể tồn tại một số chuỗi độc

lập. Điều này là bình thường trong quá trình phát triển hệ thống, nơi các nhà phát triển khác nhau làm việc độc lập trên các phiên bản mã nguồn khác nhau và thay đổi mã đó theo những cách khác nhau. Thông thường, khi làm việc trên một hệ thống, khi muốn thử nghiệm một tính năng mới nên tạo một nhánh mới để những thay đổi không vô tình làm hỏng hệ thống đang hoạt động.

Ở một giai đoạn nào đó, có thể cần phải *hợp nhất các nhánh phiên bản (merging)* để tạo phiên bản mới của thành phần bao gồm tất cả các thay đổi đã được thực hiện.



Hình 5-10: Phân nhánh và hợp nhất phiên bản

Hình 5.10 mô tả phiên bản thành phần V2.1.1 và V2.2 được hợp nhất để tạo phiên bản V2.3. Nếu những thay đổi được thực hiện trong các phiên bản này liên quan đến các phần hoàn toàn khác nhau của mã nguồn, thì phiên bản hợp nhất có thể được hệ thống kiểm soát phiên bản tự động kết hợp các thay đổi này. Đây là tình huống thường gặp khi các lập trình viên viết thêm các nội dung mã nguồn mới. Tuy nhiên, những sửa đổi do các nhà phát triển khác nhau thực hiện trên mã nguồn đôi khi chồng chéo lên nhau. Các thay đổi có thể không tương thích và gây ảnh hưởng lẫn nhau. Trong trường hợp này, nhà phát triển phải kiểm tra xung đột và xét duyệt các thay đổi đối với các thành phần để giải quyết sự không tương thích giữa các phiên bản khác nhau.

5.3.2 Hệ thống kiểm soát phiên bản

a) Các đặc trưng của hệ thống kiểm soát phiên bản

- *Định danh phiên bản và bản phát hành*: Các phiên bản được quản lý của một thành phần phần mềm được gán mã định danh duy nhất khi chúng được gửi tới hệ thống. Những mã định danh này cho phép quản lý các phiên bản khác nhau của cùng một thành phần mà không thay đổi tên thành phần đó. Các phiên bản

cũng có thể được gán các thuộc tính, với tập hợp các thuộc tính được sử dụng để nhận dạng duy nhất từng phiên bản.

- *Ghi lại lịch sử thay đổi*: Hệ thống lưu giữ hồ sơ về những thay đổi đã được thực hiện để tạo phiên bản mới của một thành phần từ phiên bản trước đó. Trong một số hệ thống, những thay đổi này có thể được sử dụng để chọn một phiên bản hệ thống cụ thể.
- *Phát triển độc lập*: Các nhà phát triển khác nhau có thể đồng thời làm việc trên cùng một thành phần. Hệ thống kiểm soát phiên bản theo dõi các thành phần đã được kiểm tra để chỉnh sửa và đảm bảo rằng những thay đổi do các nhà phát triển khác nhau thực hiện đối với một thành phần không gây trở ngại.
- *Hỗ trợ phát triển dự án*: Một hệ thống kiểm soát phiên bản có thể hỗ trợ quản lý các thành phần được dùng chung giữa nhiều dự án.
- *Quản lý lưu trữ*: Thay vì duy trì các bản sao riêng biệt của tất cả các phiên bản của một thành phần, hệ thống kiểm soát phiên bản có thể sử dụng các cơ chế hiệu quả để đảm bảo rằng các bản sao trùng lặp của các tệp giống nhau không được tạo ra. Khi chỉ có những khác biệt nhỏ giữa các tệp, hệ thống quản lý phiên bản có thể lưu trữ những khác biệt này thay vì duy trì nhiều bản sao của tệp. Một phiên bản cụ thể có thể được tạo lại tự động bằng cách áp dụng những khác biệt từ phiên bản chính.

b) Phân loại hệ thống kiểm soát phiên bản

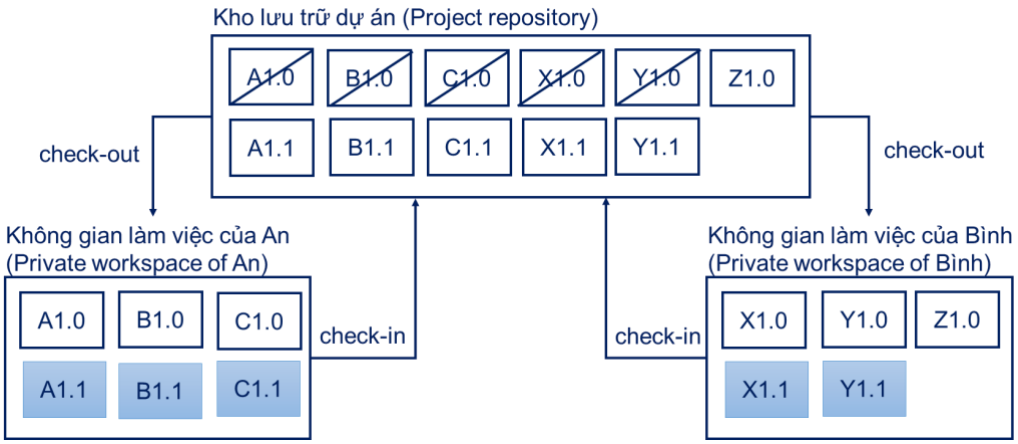
Các hệ thống kiểm soát phiên bản có thể được phân loại theo nhiều cách khác nhau. Ví dụ dựa trên mô hình kho lưu trữ (hệ thống phân tán, hệ thống tập trung) hay chế độ triển khai lưu trữ (tại chỗ, trên đám mây),…

- *Hệ thống tập trung*: có một kho lưu trữ chính duy nhất duy trì tất cả các phiên bản của các thành phần phần mềm đang được phát triển. **Subversion** là một ví dụ về hệ thống kiểm soát phiên bản tập trung.
- *Hệ thống phân tán*: tồn tại nhiều phiên bản của kho thành phần cùng một lúc. **Git** là một ví dụ về hệ thống kiểm soát phiên bản phân tán.

c) Hệ thống kiểm soát phiên bản tập trung

Trong các hệ thống tập trung, các nhà phát triển check-out các thành phần hoặc thư mục của các thành phần từ kho lưu trữ dự án vào không gian làm việc riêng của họ và làm việc trên các bản sao này. Khi các thay đổi được hoàn tất, nhà phát triển sẽ check-in các thành phần trở lại kho lưu trữ. Điều này tạo ra một phiên bản thành phần mới mà sau đó có thể được chia sẻ cho các nhà phát triển khác.

Nếu có hai hoặc nhiều người cùng làm việc trên một thành phần cùng lúc thì mỗi người phải check-out thành phần đó từ kho lưu trữ. Nếu một thành phần đã được check-out, hệ thống kiểm soát phiên bản sẽ cảnh báo những người dùng khác muốn check-out thành phần đó rằng nó đã được người khác check-out. Hệ thống cũng sẽ đảm bảo rằng khi check-in các thành phần đã sửa đổi, các phiên bản khác nhau sẽ được gán các mã định danh phiên bản khác nhau và được lưu trữ riêng biệt.



Hình 5-11: Hệ thống kiểm soát phiên bản tập trung

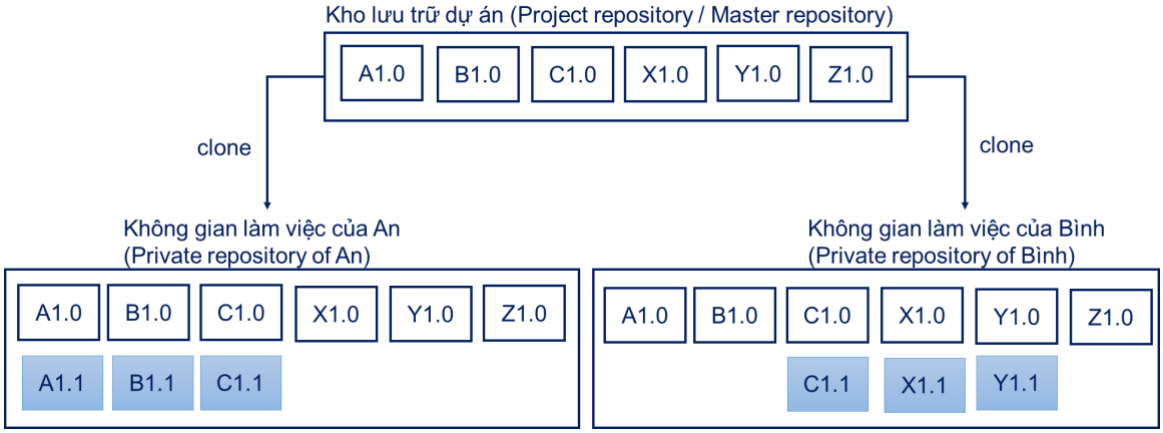
Hệ thống chỉ có một khu lưu trữ chính duy nhất cho các phiên bản của dự án. Các lập trình viên như An và Bình khi làm việc sẽ tạo các bản sao của các thành phần cần sử dụng để thay đổi trên đó. Trong ví dụ này, An đã check-out các phiên bản A1.0, B1.0 và C1.0. Anh ấy đã sửa đổi các phiên bản này và đã tạo ra các phiên bản mới A1.1, B1.1 và C1.1. Sau đó An check-in các phiên bản mới này vào kho lưu trữ. Bình check-out X1.0, Y1.0 và Z1.0. Anh ta cũng tạo ra các phiên bản mới của các thành phần X1.1, Y1.1 và check-in chúng trở lại kho lưu trữ.

Trong một tình huống nếu Bình cũng check-out thành phần C1.0 và sửa đổi để tạo ra thành phần C1.1 của riêng mình thì khi Bình thực hiện check-in, hệ thống sẽ tạo ra một phiên bản C1.2 khác để những thay đổi của An không bị ghi đè.

d) Hệ thống kiểm soát phiên bản phân tán

Hệ thống kiểm soát phiên bản phân tán sử dụng cách tiếp cận ngang hàng để kiểm soát phiên bản, trái ngược với cách tiếp cận máy khách-máy chủ của các hệ thống tập trung. Không có phiên bản trung tâm duy nhất của cơ sở mã; thay vào đó, mỗi người dùng có một bản sao làm việc và toàn bộ lịch sử thay đổi.

Kho lưu trữ "master" được tạo trên máy chủ duy trì các sản phẩm do nhóm phát triển tạo ra. Thay vì chỉ check-out các thành phần cần dùng, một nhà phát triển tạo một bản sao của kho lưu trữ dự án (clone) vào không gian làm việc riêng. Các nhà phát triển làm việc trên các tệp được yêu cầu và duy trì các phiên bản mới trên kho lưu trữ riêng trên máy tính của mình. Khi các thay đổi được thực hiện, nhà phát triển "commit" những thay đổi này và cập nhật kho lưu trữ riêng. Sau đó, họ có thể "push" những thay đổi này vào kho dự án. Hình 5.12 minh họa tình huống cả An và Bình đều đã sao chép kho lưu trữ dự án và có các thành phần được cập nhật. Họ vẫn chưa đẩy những thay đổi này trở lại kho dự án.



Hình 5-12: Tạo bản sao của kho lưu trữ trong Hệ thống kiểm soát phiên bản phân tán

Ưu điểm của hệ thống kiểm soát phiên bản phân tán:

- Cung cấp một cơ chế sao lưu cho kho lưu trữ.
    - Nếu kho lưu trữ bị hỏng, công việc có thể tiếp tục và kho lưu trữ dự án có thể được khôi phục từ các bản sao cục bộ.
- Cho phép hoạt động ngoại tuyến để các nhà phát triển có thể thực hiện các thay đổi nếu không có kết nối mạng.

- Các nhà phát triển có thể biên dịch và thực thi toàn bộ hệ thống trên máy cục bộ để thử nghiệm những thay đổi đã thực hiện.

5.3.3 Đánh số phiên bản

Khi một mục cấu hình bị thay đổi, các mục cũ không được thay thế bằng bản mới; thay vào đó, bản cũ được duy trì (maintained) và một cái mới được tạo ra. Điều này tạo ra nhiều phiên bản của cùng một mục, do đó, các quy ước gán số phiên bản (version number) được cần đến. Đánh số phiên bản hay định danh phiên bản phần mềm (Software versioning)

- Một trong những hoạt động nền tảng của quản lý cấu hình.
- Mục đích của định danh là để xác định tính duy nhất của một mục cấu hình ở mỗi giai đoạn phát triển của nó và mối quan hệ của nó với các mục cấu hình khác.
- Nó bao gồm việc mô tả tên, đánh số, đánh dấu đặc trưng, giúp nhận biết và phân biệt một mục cấu hình với các mục cấu hình hay thành phần khác.

Các thay đổi cũng không giống nhau giữa các lần thực hiện. Có những lần chỉnh sửa chỉ đơn thuần là những hiệu chỉnh nhỏ không đáng kể, những cũng có những lần "nâng cấp" toàn diện khiến cho nó không còn là thứ giống như ban đầu. Ví dụ một số phiên bản của phần mềm hệ điều hành iOS: iOS 17.0.3, iOS 16.6.1, iOS 15.7.9. Ở đây, chúng ta thấy sự khác biệt trong tên gọi các phiên bản là các con số đi kèm, khoảng cách giữa các số thường thể hiện sự cải tiến trong phần mềm. Khi sử dụng các công cụ quản lý phiên bản, việc quản lý việc đánh số phiên bản có thể được hỗ trợ tự động.

Một số phương pháp đánh số phiên bản được dùng phổ biến:

- Đánh số phiên bản bằng các chuỗi số và ký tự (Sequence-based identifiers)
- Đánh số phiên bản dựa theo mức độ ổn định của sản phẩm / các giai đoạn của sản phẩm (Stage-based identifiers)

a) Đánh số phiên bản bằng các chuỗi số và ký tự

Phương pháp này sử dụng các con số (đôi khi kết hợp thêm các chữ cái) để gán số hiệu cho các phiên bản. Công thức đánh số hiệu phiên bản như sau gồm 3 hoặc 4 nhóm chuỗi số tách nhau bằng dấm chấm. Công thức chung:

major.minor.[build [.revision]]

hoặc **major.minor [maintenance[.build]]**

Ý nghĩa các số *major, minor, build, revision* như sau

- *major*: Chuỗi phiên bản chính. Đánh số khi có những thay đổi không tương thích với phiên bản cũ. Chỉ số *major* sẽ tăng mỗi khi: Có sự thay đổi lớn trong "nhân hệ thống" mà theo đó hệ thống mới có thể khác 1 phần hay hoàn toàn hệ thống cũ.
- *minor*: Chuỗi phiên bản phụ. Đánh số khi thêm tính năng mới nhưng vẫn đảm bảo tương thích với các phiên bản cũ. Chỉ số *minor* sẽ tăng mỗi khi: Có sự thay đổi phần "core" của hệ thống mà không làm mất đi hoàn toàn tính tương thích trong cùng phiên bản chính.
- *build*: Chuỗi phiên bản build. Đánh dấu các lần fix bugs, 2 chữ số. chỉ số *build* sẽ tăng mỗi khi: Có đóng gói gửi đi ra ngoài (đội phát triển) nhằm các mục đích phát hành hay thử nghiệm. và đảm bảo tương thích với các bản cũ trước đó.
- *revision*: Lần sửa đổi. đánh dấu số lần sửa đổi của mã nguồn của phiên bản build. Chỉ số *revision* có thể được sử dụng mỗi khi: Cần thay thế code phát hành trước đó mà chưa cần thiết phải thay tên phiên bản. Chỉ số này là lần sửa đổi (revisions) của mã nguồn, nó đánh dấu số lần sửa đổi của mã nguồn và được thường được hệ thống kiểm quản lý mã nguồn gán.

Một số nguyên tắc:

- Khi phát hành một phiên bản mới các chỉ số ***major, minor***, ***build*** phải được tăng ổn định và có thứ tự. Ví dụ 1.9.0 → 1.10.0 → 1.10.1
- Mỗi khi phiên bản mới đã được phát hành, tất cả nội dung (bao gồm mã nguồn, API) của phiên bản đó phải giữ nguyên không được thay đổi. Bất kỳ thay đổi phát sinh nào đều phải được công bố như phát hành một phiên bản mới.

b) Đánh số phiên bản dựa theo mức độ ổn định / giai đoạn của sản phẩm

Tên gọi cho các phiên bản phần mềm khi phát hành gồm: Alpha, Closebeta, Openbeta, Release Candidate, Official version.

Ý nghĩa của từng phiên bản như sau:

- *Alpha*: Giai đoạn này là pha đầu tiên bắt đầu kiểm thử phần mềm trong vòng đời phát hành (alpha là kí tự đầu tiên trong bảng chữ cái Hy Lạp, được sử dụng như số 1). Giai đoạn alpha luôn luôn được kết thúc bằng việc không bổ sung thêm chức năng nào nữa (feature freeze), như vậy có thể nói phần mềm sau giai đoạn này là "đã hoàn chỉnh về chức năng" (feature complete). Các phần mềm trong giai đoạn này đều chưa hoàn chỉnh và có thể gây ra mất dữ liệu hoặc crash, nên những phiên bản phần mềm như vậy thường không được công bố rộng rãi mà chỉ khuyến khích bộ phận kiểm thử hay những người tình nguyện kiểm thử sử dụng nhằm tìm kiếm lỗi.
- *Beta*: Là các phiên bản dùng thử, nhằm tung ra để người dùng public rộng rãi sử dụng và… test lỗi, phản hồi. Bản này có thể có lỗi được phát hành để nhằm mục tiêu hoàn thiện trước khi phát hành chính thức.
    - *Closebeta*: Phiên bản thử nghiệm hạn chế. Đặc điểm: Bản closebeta là bản thử nghiệm các tính năng mới phát triển, nó thường không mang đầy đủ các đặc điểm của hệ thống và dễ dàng thay đổi hoặc bị loại bỏ nếu nhận được các phản ứng không tốt sau khi thử nghiệm.
    - *Openbeta*: Phiên bản thử nghiệm diện rộng. Đặc điểm: Bản openbeta là bản thử nghiệm các tính năng đã phát triển, nó thường mang đầy đủ các đặc điểm của hệ thống và hiếm khi thay đổi hoặc bị loại bỏ khỏi hệ thống trừ khi có phản hồi không tốt từ cộng đồng.
- *Release Candidate*: Phiên bản ứng viên. Là phiên bản cho dùng thử trước khi sản phẩm chính thức ra đời. Đặc điểm: Bản Release Candidate là bản ổn định, là ứng cử viên cho phiên bản chính thức. Các lỗi được phát hiện trong giai đoạn này sẽ tiếp tục được sửa chữa. Phiên bản này được sử dụng như một bản đệm trong thời gian chờ phiên bản chính thức ra mắt nhằm tránh trường hợp một bản chính thức có thể bị lỗi ngay sau khi ra mắt.

- *Official version*: Phiên bản chính thức. Đặc điểm: Bản Official version là bản chính thức đầu tiên của giai đoạn phát triển của dòng phiên bản mới. Official version là tên gọi của lần phát hành phiên bản duy nhất, các phiên bản tiếp theo sau đó sẽ chỉ được gọi tên bằng số phiên bản. Phiên bản này đánh dấu việc ra mắt dòng phiên bản mới và khuyến khích người sử dụng chuyển tiếp lên phiên bản mới.

5.4 Ví dụ và bài tập

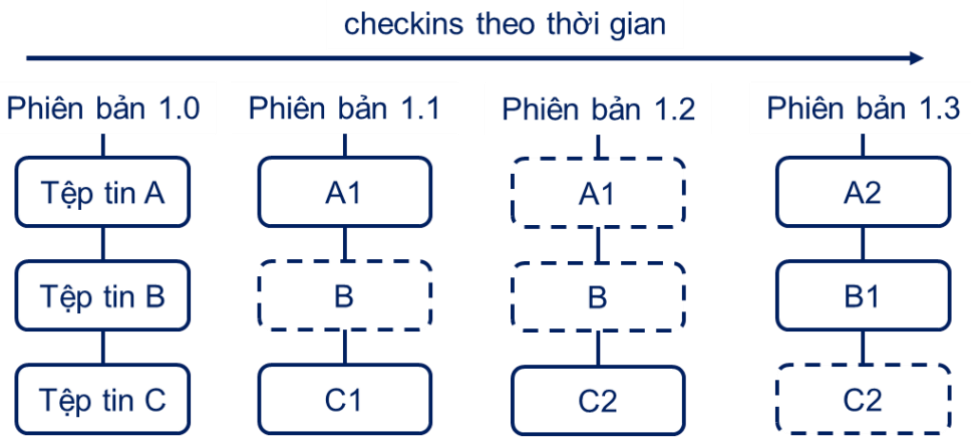
5.4.1 Giới thiệu về hệ thống Git

Git là một hệ thống kiểm soát phiên bản phân tán mã nguồn mở và miễn phí được thiết kế để quản lý phiên bản từ các dự án nhỏ đến rất lớn mang lại tốc độ và hiệu quả. Git vốn được phát triển nhằm quản lý mã nguồn (source code) của Linux ra đời vào năm 2005. Hiện nay git là một trong các hệ thống quản lý phiên bản thông dụng nhất.

Git có đầy đủ các ưu điểm của một hệ thống quản lý phiên bản phân tán bao gồm:

- Lưu lại được các phiên bản khác nhau của mã nguồn dự án phần mềm
- Khôi phục lại mã nguồn từ một phiên bản bất kỳ
- Dễ dàng so sánh giữa các phiên bản
- Phát hiện được ai đã sửa phần nào làm phát sinh lỗi
- Khôi phục lại tập tin bị mất
- Dễ dàng thử nghiệm, mở rộng tính năng của dự án mà không làm ảnh hưởng đến phiên bản chính (master branch)
- Giúp phối hợp thực hiện dự án trong nhóm một cách hiệu quả

Sự khác biệt chính giữa Git và các hệ thống kiểm soát phiên bản khác là cách Git lưu trữ dữ liệu. Về mặt khái niệm, hầu hết các hệ thống khác đều lưu trữ thông tin dưới dạng danh sách các thay đổi dựa trên tệp. Thông tin chúng lưu trữ dưới dạng một tập hợp các tệp và những thay đổi được thực hiện đối với từng tệp theo thời gian.



Hình 5-13: Git lưu trữ dữ liệu dưới dạng ảnh chụp nhanh của dự án theo thời gian

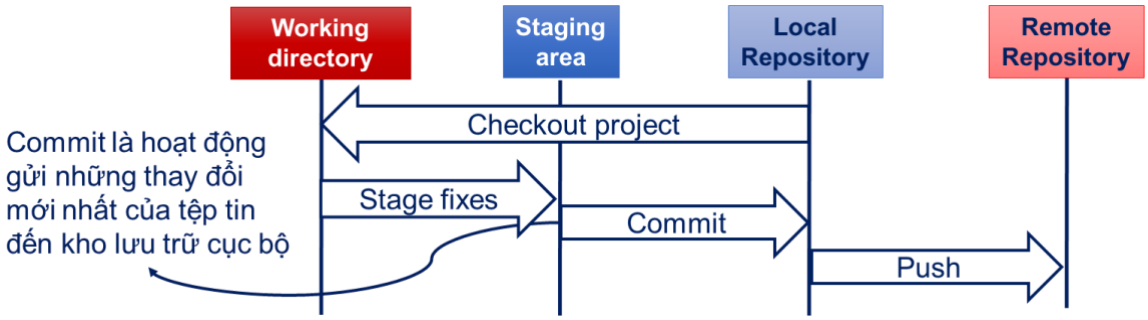
Git coi dữ liệu của nó giống như một loạt ảnh chụp nhanh của một hệ thống tệp thu nhỏ. Với Git, mỗi khi commit hoặc lưu trạng thái dự án, Git sẽ chụp ảnh tất cả các tệp của dự án tại thời điểm đó và lưu trữ một tham chiếu đến ảnh chụp nhanh này. Để hiệu quả, nếu các tệp không thay đổi, Git sẽ không lưu trữ lại tệp mà chỉ lưu trữ một liên kết đến tệp giống hệt trước đó mà nó đã lưu trữ.

Trong các hệ thống quản lý phiên bản thông thường, các dữ liệu sẽ được lưu trữ ở hai nơi, bao gồm thư mục đang làm việc trên máy tính (working directory hay working tree) và kho chứa các kết quả (repository) sau khi đã thực hiện thay đổi.

Kho chứa các kết quả (repository) là nơi Git lưu trữ siêu dữ liệu và cơ sở dữ liệu đối tượng cho dự án. Đây là phần quan trọng nhất của Git và là thứ được sao chép khi thực hiện lệnh clone kho lưu trữ từ máy tính khác hoặc kho lưu trữ từ xa trên đám mây.

- **Local Repository:** là repository nằm trên chính máy tính của lập trình viên
- **Remote Repository:** là repository được cài đặt trên máy chủ chuyên dụng. Ví dụ: GitHub, GitLab, Bitbucket,...

Git có thêm một lựa chọn nữa, đó là khu vực trung gian gọi là **Staging Area** và đây chính là một lợi thế lớn của Git. Staging Area nghĩa là khu vực sẽ lưu trữ những thay đổi trên tập tin để nó có thể được commit, vì muốn commit tập tin nào thì tập tin đó phải nằm trong Staging Area. Một tập tin khi nằm trong Staging Area sẽ có trạng thái là **Stagged.**



Hình 5-14: Các phần chính của một dự án Git

Khi đang ở trong thư mục làm việc, chúng ta có thể chọn các tệp cần được Git theo dõi. Tại sao chúng ta cần điều này? Tại sao chúng ta không theo dõi mọi thứ trong dự án? Đó là vì một số tệp trong dự án như tệp lớp, tệp nhật ký, tệp kết quả và tệp dữ liệu tạm thời được tạo động. Việc theo dõi phiên bản của những tệp này không có ý nghĩa. Trong khi đó, các tệp mã nguồn, tệp dữ liệu, tệp cấu hình và các tạo phẩm khác của dự án chứa logic nghiệp vụ của ứng dụng. Do đó, các tệp này sẽ được Git theo dõi và cần được thêm vào khu vực trung gian. Nói cách khác, khu vực trung gian là nơi chúng ta có thể nhóm, thêm và sắp xếp các tệp được commit cho Git để theo dõi các phiên bản của chúng.

Trạng thái của các tệp tin trong dự án Git:

- *Chưa được theo dõi (Untracked)*: tệp tin chưa được theo dõi phiên bản
- *Đã theo dõi (Tracked)*: tệp tin được đánh dấu để Git quản lý phiên bản. Tại trạng thái này, tệp tin có thể nhận thêm 3 trạng thái cụ thể khác:
    - *Đã chỉnh sửa (Modified)*: tệp tin được thay đổi nhưng chưa được commit vào kho lưu trữ, cần chuyển vào trạng thái Staged
    - *Sẵn sàng để commit (Staged)*: đánh dấu tệp tin đã được chỉnh sửa trong phiên bản hiện tại để đưa vào snapshot lần commit tiếp theo
    - *Đã hoàn thành commit (Committed)*: dữ liệu (các thay đổi) đã được lưu trữ an toàn trong kho lưu trữ cục bộ

5.4.2 Các thao tác trên Git

a) Cài đặt Git và các thiết lập cơ bản

Bài tập: Cài đặt Git vào máy tính và thực hiện các thiết lập cơ bản

Hướng dẫn:

- Trên hệ điều hành Windows và macOS, tải về bộ cài đặt Git và thực hiện cài đặt: https://git-scm.com/downloads
- Trên hệ điều hành Linux (Ubuntu/Debian), có thể sử dụng lệnh

```
$ sudo apt-get install git
```

- Trên hệ điều hành Linux (CentOS/Fedora/RHEL), có thể sử dụng lệnh

```
$ yum install git
```

Bên cạnh đó thì có các GUI hỗ trợ như SourceTree, Git Desktop; một số công cụ soạn thảo mã nguồn cũng có tích hợp Git.

Thao tác với Git được thực hiện thông qua giao diện dòng lệnh hoặc sử dụng các công cụ có giao diện người dùng đồ họa (GUI). Sau khi cài đặt Git vào hệ điều hành Windows, các bạn mở ứng dụng Git Bash để bắt đầu sử dụng các lệnh của Git. Bây giờ bạn mở terminal trên máy tính và chạy mã lệnh **git --version** để kiểm tra phiên bản đã được cài đặt.

Hình 5-15: Giao diện dòng lệnh của Git Bash trên hệ điều hành Windows

Nếu kết quả hiển thị được phiên bản Git chứng tỏ bạn đã cài đặt thành công, nếu không thì git chưa tồn tại và bạn cần cài đặt lại trên máy tính của mình.

Bài tập: Cấu hình thông tin cá nhân, khai báo tên và địa chỉ email

Hướng dẫn:

- Khai báo tên và địa chỉ email vào trong file cấu hình của Git trên máy (tệp tin .gitconfig)

```
$ git config --global user.name "Ten"
$ git config --global user.email "Email"
```

- Xem lại danh sách các thiết lập của Git trên máy hiện tại

```
$ git config --list
```

- Xem nội dung tệp tin cấu hình

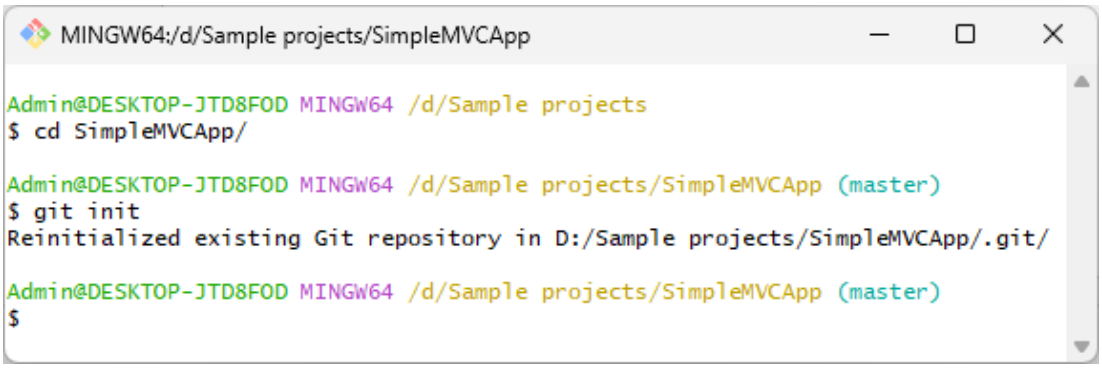
```
$ cat ~/.gitconfig
```

b) Tạo kho lưu trữ (repository)

Trong Git, Repository là nơi lưu trữ, quản lý tất cả những thông tin cần thiết (thư mục, tập tin, ảnh, video, bảng biểu, dữ liệu…) cũng như các sửa đổi và lịch sử của toàn bộ dự án. Khi tạo mới repository, bạn nên tạo thêm tập tin README hoặc một tập tin thông tin giới thiệu về dự án. Trong bài tập này chúng ta sẽ Tạo kho lưu trữ cục bộ (local repository) tại thư mục mã nguồn của dự án, có thể dùng lệnh Git init để tạo ra một git repository trong 1 dự án mới hoặc đã có sẵn. Lệnh này được sử dụng trong thư mục gốc của dự án.

Bài tập: Tạo kho lưu trữ cục bộ (local repository) tại thư mục mã nguồn của dự án

Hướng dẫn:



Hình 5-16: Khởi tạo kho lưu trữ cục bộ

- **Bước 1**: Truy cập vào thư mục mã nguồn, sử dụng lệnh cd
- **Bước 2**: Khởi tạo kho lưu trữ tại thư mục hiên tại, sử dụng lệnh git init
- (Tạo thư mục mới và khởi tạo kho lưu trữ tại thư mục đó, sử dụng lệnh: git init tên_thư_mục)

Bài tập: Tạo kho lưu trữ từ xa (remote repository) trên Github và tạo bản sao về máy cá nhân

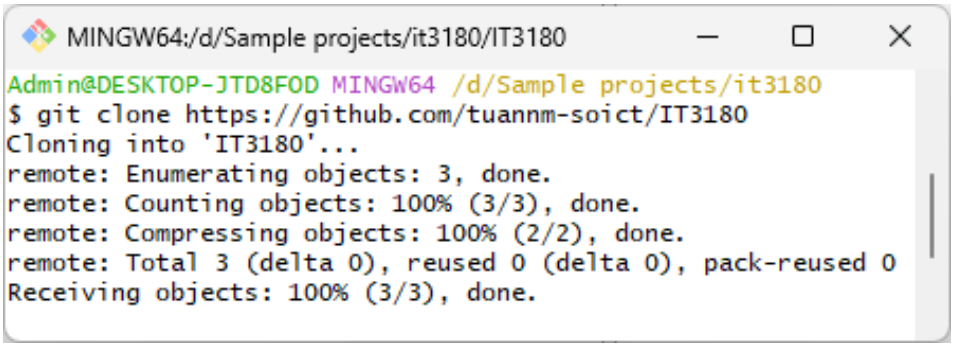
GitHub là một dịch vụ lưu trữ mã nguồn và quản lý phiên bản phân tán (distributed version control system) phổ biến. Nó cho phép các nhà phát triển phát triển phần mềm cộng tác, quản lý và theo dõi sự thay đổi trong mã nguồn của dự án.

GitHub cung cấp một nền tảng trực quan và dễ sử dụng để lưu trữ các dự án phần mềm, quản lý phiên bản và phối hợp công việc giữa các thành viên trong một nhóm phát triển. Được phát triển dựa trên hệ thống quản lý phiên bản Git, GitHub cung cấp nhiều tính năng hữu ích như hệ thống theo dõi lỗi (issue tracking), wiki, công cụ hỗ trợ liên quan đến việc kiểm tra mã (code review), tích hợp liên tục (continuous integration) và nhiều hơn nữa.

Hướng dẫn:

- **Bước 1**: Đăng ký tài khoản miễn phí và đăng nhập vào https://github.com/
- **Bước 2**: Tạo mới kho lưu trữ. Sau khi tạo xong, địa chỉ kho lưu trữ có dạng:

```
https://github.com/$user-name/$repository
```



Hình 5-17: Tạo bản sao từ kho lưu trữ trên Github

- **Bước 3**: Tạo bản sao của kho chứa trên Github về máy, sử dụng lệnh:

```
$ git clone địa_chỉ_kho_lưu_trữ
```

- Ví dụ: `$ git clone https://github.com/tuannm-soict/IT3180`

c) Các lệnh cơ bản

- Lệnh thêm các tệp tin có trong kho chứa để quản lý phiên bản (thiết lập trạng thái tracked). Khi bạn tạo, cập nhật hoặc xóa file, những hành động đó nó chỉ có tác dụng ở trên local, nó không được commit trong lần tiết theo. Chúng ta sử dụng lệnh Git add để chọn ra một hoặc nhiều file được commit vào git trong lần tiếp theo.

```
$ git add tên_file
```

- Lệnh kiểm tra trạng thái các tệp tin được đánh dấu quản lý

```
$ git status
```

- Lệnh ghi lại việc thêm / thay đổi của tệp tin. Đây có lẽ là câu lệnh được sử dụng nhiều nhất trong Git, khi dự án đã đạt được xây dựng đến một giai đoạn nhất định, chúng ta muốn lưu phiên bản đó vào Git. Git commit để lưu lại là một phiên bản dự án và đặc biệt trong tương lai có thể quay trở lại phiên bản đó nếu cần.

```
$ git commit -m "Nội dung lời nhắn"
```

- Với các kho chứa từ xa như Github, cần đẩy các tệp tin đã được commit

```
$ git push origin [branch]
```

- Cập nhật kho lưu trữ: `$ git pull`
- Xem lịch sử của các lần commit trước đó

```
$ git log
```

- Sử dụng tham số -p để hiển thị chi tiết của mỗi lần commit

```
$ git log -p
```

- Một số tùy chọn xem log:
  - `--since, --after`: Xem các lần commit kể từ ngày nhất định
  - `--until`: Xem các lần commit trước từ ngày nhất định
  - `--author`: Xem các lần commit của một người cụ thể
  - `--grep`: Lọc các chuỗi trong log

Branch - Nhánh trong Git cho phép có thể tách riêng các tính năng của dự án, thử nghiệm các tính năng mới hay cũng có thể dùng nhánh để khắc phục, hoàn chỉnh lỗi nào đó của dự án,… Khi bắt đầu khởi tạo một repository hoặc clone một repository, chúng ta sẽ có một nhánh (branch) chính tên là master, đây là branch chứa toàn bộ các mã nguồn chính trong repository. Từ nhánh master này, trong quá trình thực hiện dự án bạn có thể rẽ thêm nhiều nhánh khác tùy theo nhu cầu thực tế. Tất cả các nhánh đều được hệ thống lưu lại lịch sử các lần commit trên nhánh và bạn hoàn toàn quay lại mốc commit nào mà mình muốn. Một lưu ý là tất cả thao tác fetch, push hoặc pull mặc định

thực hiện trên nhánh hiện hành, nên cần lưu ý nhánh đang thao tác là nhánh master hay nhánh nào để tránh sai sót. Các lệnh thao tác với nhánh:

- Lệnh tạo một nhánh: `$ git branch [branch]`
- Lệnh xóa một nhánh: `$ git branch -d [branch]`
- Lệnh chuyển nhánh khác:

```
$ git checkout [branch]
$ git checkout –b [new_branch]
```

- Gộp dữ liệu từ một branch: Hợp nhất các nhánh (Merge): chuyển dữ liệu từ một branch nào đó về branch đang trỏ đến

```
$ git merge [branch]
```

Trong một hệ thống kiểm soát phiên bản như Git, xung đột có thể xảy ra khi hai hoặc nhiều người thay đổi cùng một tệp. Các xung đột có thể xuất hiện tại kho lưu trữ cục bộ của thành viên hoặc kho lưu trữ từ xa Github. Xung đột chỉ ảnh hưởng đến nhà phát triển trong quá trình tiến hành hợp nhất, phần còn lại của nhóm (nghĩa là những người không tham gia vào quá trình thay đổi file này) sẽ không biết gì về xung đột. Nguyên tắc đưa ra, ai gây ra xung đột người đó sẽ chủ động giải quyết trước. Git không thể tự động xác định điều gì là đúng, nó đâu thể biết được nội dung trong file như thế nào là đúng theo ý muốn của người viết. Git sẽ đánh dấu file bị xung đột và tạm dừng quá trình hợp nhất. Sau đó, trách nhiệm của các nhà phát triển là giải quyết xung đột dựa trên những gì Git đã đánh dấu.

Chương 6 KỸ NGHỆ YÊU CẦU PHẦN MỀM

Nội dung:

- Khái niệm và tầm quan trọng của yêu cầu phần mềm
- Quy trình phân tích yêu cầu phần mềm
- Tổng quan về mô hình hoá trong phát triển phần mềm
- Các công cụ mô hình hoá yêu cầu phần mềm
- Bài tập về mô hình hoá yêu cầu phần mềm

6.1 Khái niệm và tầm quan trọng của yêu cầu phần mềm

Các khái niệm sau đây được sử dụng xuyên suốt bài giảng cơ học kỹ thuật, việc hiểu rõ các khái niệm này có vai trò quan trọng để bắt đầu môn học này.

6.1.1 Khái niệm

Các yêu cầu phần mềm diễn tả chức năng của hệ thống đứng từ quan điểm của khách hàng

- Các yêu cầu phần mềm cho phép xác định các tính năng (functionality), các ràng buộc và mục đích của hệ thống
- Yêu cầu phần mềm cần được hiểu rõ bởi cả khách hàng và cả những nhà phát triển
- Tài liệu đặc tả yêu cầu phần mềm (Software Requirement Specification – SRS) chính là sản phẩm đầu ra của pha phân tích yêu cầu phần mềm

6.1.2 Tại sao hiểu rõ yêu cầu khi phát triển phần mềm lại rất quan trọng?

Các nguyên nhân thất bại của các dự án phần mềm:

- Yêu cầu không đầy đủ (13.1%)
- Thiếu sự tham gia của người dùng (12.4%)
- Thiếu tài nguyên (10.6%)
- Mong đợi phi thực tế của khách hàng (9.9%)
- Thiếu sự hỗ trợ nghiệp vụ bên phía khách hàng (9.3%)
- Thay đổi yêu cầu và đặc tả (8.8%)

- Thiếu kế hoạch (8.1%)
- Hệ thống bị lỗi thời (7.5%)

Nguyên nhân lớn nhất đến tự việc định nghĩa yêu cầu không đầy đủ và rõ ràng. Do vậy hiểu rõ yêu cầu phần mềm giúp đội dễ án có cơ hội thành công hơn. Tuy nhiên trong thực tế việc định nghĩa và thu thập yêu cầu phần mềm một cách rõ ràng không dễ dàng. Do đội dự án chỉ có thể xây dựng được một hệ thống trừ khi hiểu rất rõ về hệ thống đó và những gì được yêu cầu tuy nhiên người đưa ra yêu cầu là khách hàng lại chỉ hiểu biết một phần về hệ thống mà họ muốn xây dựng. Kết lại việc thu thập, định nghĩa và xây dựng yêu cầu phần mềm là cả một quá trình đòi hỏi sự cộng tác của nhiều bên: đội dự án, nhà đầu tư, khách hàng, chuyên gia…

6.1.3 Nguồn gốc yêu cầu phần mềm?

- Khách hàng – Người sử dụng (Client – User)
  - Khách hàng cung cấp các yêu cầu nghiệp vụ: cung cấp thông tin về công ty, tổ chức trong đó hệ thống sẽ được vận hành, đặc điểm và các quy định, các điều luật riêng của công ty (privacy)
  - Khách hàng cung cấp yêu cầu người dùng (user requirement): cung cấp thông tin về các chức năng cụ thể mà hệ thống sẽ cung cấp
- Đội phát triển cần làm việc với cả hai nhóm khách hàng nói trên để có thể thu thập được đầy đủ thông tin về yêu cầu của hệ thống

6.1.4 Thách thức trong xác định yêu cầu phần mềm

a) Đặc điểm của khách hàng

Khách hàng có các đặc điểm sau đây:

- Khách hàng không phải là những nhà phát triển
- Khách hàng chỉ có những ý tưởng mơ hồ về phần mềm cần xây dựng
- Khách hàng thường hay thay đổi về yêu cầu đối với hệ thống
  - Khi họ nhìn thấy hệ thống, họ sẽ yêu cầu những tính năng mới
  - Những thay đổi đó có thể dẫn tới việc phải sửa chữa hệ thống rất nhiều
  - Những thay đổi này gây ra các vấn đề cho các mô hình phát triển phần mềm

b) Phân tích yêu cầu trong mô hình Heavyweight

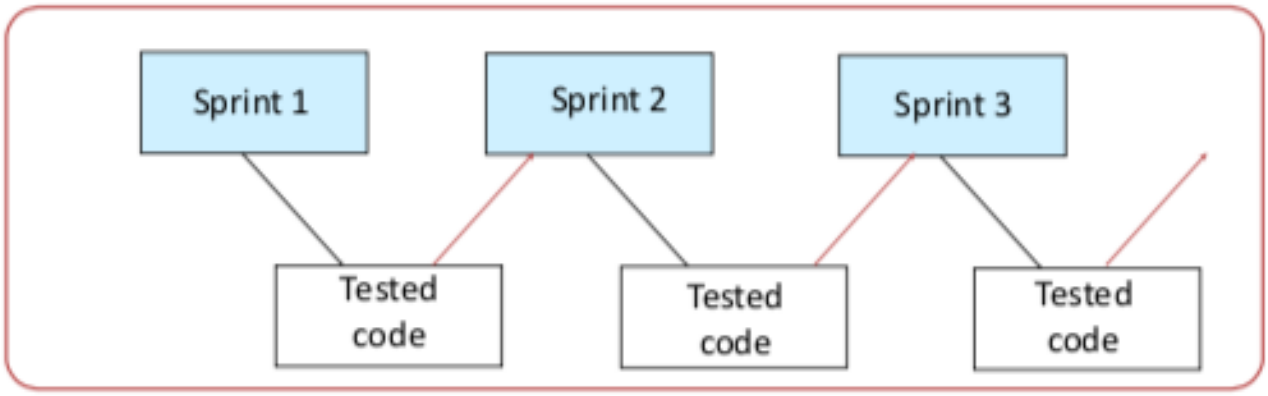
Các mô hình Heavyweight như *mô hình thác nước* yêu cầu những đặc tả rất chi tiết về yêu cầu phần mềm

- Tài liệu đặc tả từng yêu cầu ở mức độ rất chi tiết
- Tài liệu được sử dụng như một bản hợp đồng giữa hai bên
- Tài liệu đặc tả được sử dụng trong giai đoạn kiểm thử chấp nhận sau này (acceptance testing)

Các khó khăn đối với mô hình Heavyweight:

- Xây dựng tài liệu đặc tả tốn rất nhiều thời gian và công sức
- Khó bảo trì hay theo vết thay đổi đối với các đặc tả
- Kiểm chứng chi tiết từng đặc tả cũng là một công việc tốn nhiều thời gian và công sức
- Khách hàng thường không hiểu về quá trình xây dựng đặc tả và những tác động của nó lên toàn bộ quá trình phát triển nên họ có thể có những đòi hỏi vô lý

c) Phân tích yêu cầu trong mô hình Lightweight



Hình 6-1 - Sprint trong phát triển phần mềm nhanh

Những khó khăn trong việc định nghĩa, xây dựng và bảo trì đặc tả yêu cầu ở mức độ chi tiết trong mô hình Heavyweight là một trong những lí do cho rất nhiều công ty phát triển lựa chọn mô hình Lightweight (mô hình có tính lặp, mô hình phát triển nhanh). Mô hình Lightweight phát triển một tập hợp yêu cầu lựa chọn trước trong một sprint:

- Working Code – Mã nguồn phát triển được sử dụng để kiểm tra các yêu cầu

- Khách hàng và nhà phát triển cùng làm việc trên một yêu cầu cụ thể
- Tài liệu đặc tả chi tiết của yêu cầu được giảm thiểu, không yêu cầu mức độ chi tiết như mô hình Heavyweight ➔ giảm thiểu chi phí thời gian
- Tài liệu đầy đủ sẽ được hoàn thiện dần dần trong quá trình phát triển

Các khó khăn trong mô hình Lightweight:

- Một số yêu cầu phần mềm có quy mô hệ thống, không thể được định nghĩa và xây dựng trong 1 sprint
    - Ví dụ: database, kiến trúc bảo mật, thiết kế giao diện người dùng toàn hệ thống
- Các yêu cầu được phát triển trong các sprint trong tương lai có thể đòi hỏi sự thay đổi ở các yêu cầu đã được phát triển trong các sprint trước đó

Cả hai loại mô hình phát triển phần mềm heavyweight và lightweight đều có các khó khăn riêng đối với yêu cầu phần mềm

- Đối với các hệ thống lớn, đội phát triển cần phải **linh hoạt** xử lý
    - Đối với mô hình heavyweight, có thể đặc tả yêu cầu ở mức độ chi tiết **trung bình** và chuẩn bị các phiên bản **tinh chỉnh chi tiết** hơn trong quá trình phát triển, điều đó giúp giảm thiểu thời gian bỏ ra ban đầu và dễ dàng sửa đổi hơn
    - Khi sử dụng mô hình lightweight, có thể đặt **ưu tiên** cho các yêu cầu có mức độ hệ thống ở **giai đoạn sớm** của chu trình phát triển, làm cơ sở cho các sprint sau này

6.1.5 Phân loại yêu cầu phần mềm

a) Phân loại

Theo thành phần phần mềm

- Yêu cầu phần mềm
- Yêu cầu phần cứng
- Yêu cầu dữ liệu
- Yêu cầu về con người (người dùng cuối, tổ chức, khách hàng)

Theo đặc tả phần mềm

- Yêu cầu chức năng
- Yêu cầu phi chức năng
- Các ràng buộc khác

b) Yêu cầu chức năng

Định nghĩa:

- Mô tả các chức năng mà hệ thống sẽ thực hiện
- Phụ thuộc vào kiểu phần mềm và mong đợi của người dùng
    - Tương tác giữa phần mềm và môi trường, độc lập với việc cài đặt
- Có thể bao gồm các nội dung: giao dịch (transactions), dữ liệu (data) và giao diện người dùng (user interfaces)

Các công cụ đặc tả yêu cầu chức năng tiêu biểu:

- Biểu đồ luồng dữ liệu (Data Flow Diagram)
- Máy trạng thái hữu hạn (Finite State Machine)
- Mạng Petri (petri nets)
- Các biểu đồ ca sử dụng, biểu đồ hoạt động hoặc biểu đồ luồng nghiệp vụ của UML
- Kết hợp sử dụng ngôn ngữ tự nhiên để đặc tả chi tiết

c) Yêu cầu phi chức năng

Định nghĩa:

- Các yêu cầu phi chức năng không liên quan trực tiếp đến các chức năng hệ thống thực hiện
- Yêu cầu về sản phẩm: hiệu năng, độ tin cậy, tính khả chuyển
- Yêu cầu về tổ chức: bàn giao, đào tạo sử dụng, các quy chuẩn
- Yêu cầu từ bên ngoài: tương tác với các hệ thống ngoài, legacy

Ví dụ: Library Management System – Hệ thống quản lý thư viện có các yêu cầu phi chức năng sau đây:

- Phần cứng và hệ điều hành: IBM/UNIX
- Hệ quản trị CSDL: Oracle
- Ngôn ngữ lập trình: C/C++

d) Các ràng buộc khác

Được đặt ra bởi khách hàng, tổ chức vận hành và môi trường thực thi phần mềm, bao gồm:

- Các quy định do tổ chức đặt ra: quy chuẩn về tài liệu, các yêu cầu đặc biệt trong bàn giao sản phẩm và vận hành sản phẩm
- Các tiêu chuẩn của tổ chức hoặc các giao thức các điều luật của quốc gia cũng có thể là các ràng buộc phi chức năng

6.2 Quy trình phân tích yêu cầu phần mềm

Mục tiêu của quy trình phân tích yêu cầu phần mềm:

- Hiểu rõ một cách **chi tiết** các yêu cầu phần mềm
- Đảm bảo khách hàng (client) và đội phát triển đều hiểu về yêu cầu phần mềm và tác động của chúng
- Định nghĩa các yêu cầu phần mềm theo cách thức **rõ ràng cụ thể** và **chi tiết** cho khách hàng
- Định nghĩa các yêu cầu phần mềm theo cách thức **rõ ràng** cho những nhà phát triển để có thể **thiết kế** (design), **cài đặt** (implement) và **bảo trì** (maintain) hệ thống

Các bước cần thiết trong kỹ nghệ yêu cầu phần mềm:

- Phân tích (Analysis)
    - Việc phân tích giúp đưa ra các chức năng chính của hệ thống thông qua tham khảo và thảo luận với khách hàng
- Mô hình hoá (Modelling)
    - Mô hình hoá giúp tổ chức các yêu cầu phần mềm một cách có hệ thống và dễ hiểu
- Định nghĩa, ghi nhận và trao đổi, thảo luận và thoả thuận về các yêu cầu phần mềm

6.2.1. Phân tích yêu cầu phần mềm: Phỏng vấn khách hàng

Phỏng vấn khách hàng đóng vai trò vô cùng quan trọng khi phân tích và thu thập yêu cầu phần mềm. Khách hàng có thể chỉ có những hiểu biết mơ hồ về hệ thống họ muốn

phát triển. Quá trình giao tiếp và phân tích yêu cầu hệ thống với khách hàng cần phải được chuẩn bị trước bằng danh sách các câu hỏi:

- Dự án cần đạt được điều gì? Ai là nhà đầu tư?
- Những tác động của thay đổi mang lại từ hệ thống?
- Có các cá nhân, tổ chức và hệ thống nào tham gia vào?
- Dữ liệu có sẵn không?
- Cần áp dụng các quy chuẩn gì để phân tích nghiệp vụ không?

6.2.2. Phân tích yêu cầu phần mềm: Hiểu rõ yêu cầu

Việc hiểu rõ yêu cầu phần mềm được thực hiện thông qua các phiên làm việc sau:

a) Brainstorming session

Các phiên làm việc chung để thảo luận chi tiết về giải pháp khả thi, các ý tưởng giải quyết các vấn đề nghiệp vụ

- Người tham gia: đại diện từ nhiều lĩnh vực khác nhau liên quan đến hệ thống và một số thành viên chủ chốt của đội phát triển
- Output: giải pháp tốt nhất sẽ được lựa chọn

b) Joint Application Development

Các phiên làm việc phát triển ứng dụng:

- Nhiều thành viên tham gia: nhà đầu tư, đội phát triển, chuyên gia phân tích nghiệp vụ, chuyên gia đảm bảo chất lượng
- Các phiên làm việc này nhằm đưa ra:
    - Phạm vi của dự án và mục tiêu cần đạt được cuối cùng
    - Chốt lại các yêu cầu của hệ thống từ quan điểm của người dùng
    - Giải pháp về kiến trúc, nền tảng và các kỹ thuật liên quan cũng sẽ được thảo luận

c) User story mapping

Tổ chức các user-stories một cách có hệ thống giúp khách hàng có thể dễ dàng trải nghiệm được hệ thống.

6.2.3. Phân tích yêu cầu phần mềm: Phân tích các hệ thống và tài liệu có sẵn

So sánh các hệ thống đang tồn tại:

- Đôi khi các hệ thống tương tự đã tồn tại
- Đôi khi khách hàng đang sử dụng một hệ thống sẵn có và muốn mở rộng hoặc thay đổi hoàn toàn hệ thống đó
    - **Hệ thống thay thế:** khi hệ thống cần phát triển sẽ thay thế cho hệ thống sẵn có
- Các tính năng của hệ thống sẵn có cần được tham khảo để dễ dàng chuyển đổi sang hệ thống mới
- **Mở rộng hệ thống:** khi các hệ thống sẵn có không bị thay thế mà sẽ được mở rộng, thay đổi và cập nhật

Khi phân tích các hệ thống tồn tại cần phải làm rõ:

- Các tính năng của hệ thống cũ cần trong hệ thống mới
- Các tính năng của hệ thống cũ không cần trong hệ thống mới
- Đề xuất các tính năng mới

6.2.4. Phân tích yêu cầu phần mềm: Phân tích nhà đầu tư

Xác định rõ ràng các đối tượng và tác động của hệ thống đến các đối tượng đó

- Nhà đầu tư
- Client – Khách hàng
- End-users: Người dùng cuối
- Các vai trò khác nhau của người dùng trong hệ thống

Các đối tượng khách hàng sẽ được nhóm lại và phân tích về tương tác với hệ thống. Ví dụ: Hệ thống quản lý trường đại học:

- Thí sinh đăng ký vào trường đại học
- Sinh viên, Giảng viên
- Nhân viên các phòng ban, trung tâm và quản lý
- Quản trị viên hệ thống…

6.2.5. Phân tích yêu cầu phần mềm: Thoả thuận

Đánh giá tính khả thi của các yêu cầu đã thu thập và phân tích. Trong tình huống một số yêu cầu là bất khả thi với nguồn lực con người và thời gian cũng như ngân sách cần phải thoả thuận với khách hàng:

- Thảo luận về yêu cầu, tại sao lại cần thiết, liệu có thể thay thế yêu cầu bằng yêu cầu khác khả thi hơn?
- Đội phát triển giải thích với khách hàng về lí do, kĩ thuật và chi phí
- Đưa ra các gợi ý

➔ Giải quyết các vấn đề nói trên là mục đích của bước thoả thuận khi có những yêu cầu bất khả thi.

6.2.6. Tài liệu đặc tả yêu cầu phần mềm

Đặc tả yêu cầu phần mềm có thể được thực hiện bằng cách xây dựng tài liệu đặc tả với 2 phương pháp:

- Đặc tả phi hình thức (informal specification) sử dụng ngôn ngữ tự nhiên
- Đặc tả hình thức (formal specification) sử dụng tập các ký pháp có quy định về cú pháp và ngữ nghĩa

Tài liệu đặc tả được đánh giá dựa trên các tiêu chí sau:

- Tiêu chí đánh giá chất lượng hồ sơ đặc tả:
- Tính rõ ràng, chính xác
- Tính phù hợp
- Tính đầy đủ và hoàn thiện

Tài liệu SRS là các phát biểu chính thức về những yêu cầu của hệ thống mà đội phát triển cần tuân thủ và xây dựng:

- Sử dụng các công cụ mô hình hoá để mô tả các yêu cầu phần mềm một cách rõ ràng, có hệ thống và dễ hiểu
- Được sử dụng làm tài liệu kiểm thử chấp nhận sau này
- Được sử dụng để xây dựng kế hoạch kiểm thử cho đội phá triển

Cấu trúc của tài liệu đăc tả:

- Giới thiệu chung về mục đích, phạm vi, từ điển thuật ngữ, tài liệu tham khảo
- Mô tả tổng quan về hệ thống và quy trình nghiệp vụ
- Đặc tả chi tiết các chức năng của hệ thống
- Đặc tả phi chức năng và các ràng buộc nếu có

6.3 Tổng quan về mô hình hóa trong phát triển phần mềm

6.3.1 Thế nào là mô hình hoá?

Trong thực tế để xây dựng được các hệ thống phức tạp chúng ta cần phải hiểu rõ về hệ thống trước khi bắt tay vào xây dựng. Ví dụ để xây dựng được một chiếc máy bay chiến đấu, chúng ta cần bắt tay vào tìm hiểu tính năng, cấu trúc bên ngoài, cấu trúc bên trong của sản phẩm. Nếu chỉ quan sát bên ngoài sẽ không thể hiểu được cấu trúc các thành phần của sản phẩm cũng như nếu chỉ quan sát bên trong thì không thể nắm rõ được sự tương tác giữa các thành phần và cách thức cả hệ thống (máy bay chiến đấu) hoạt động. Do vậy chúng ta cần phải hiểu rõ một hệ thống phức tạp dưới nhiều góc nhìn khác nhau, mỗi góc nhìn cung cấp một đặc tả khác nhau về hệ thống. Tổng hợp lại từ nhiều góc nhìn giúp chúng ta hiểu rõ về hệ thống. Mô hình hoá hệ thống là việc chúng ta mô phỏng lại hệ thống dưới nhiều góc nhìn khác nhau để giúp cho chúng ta hiểu rõ về một hệ thống phức tạp.

6.3.2 Tại sao cần mô hình hoá?

Tầm quan trọng của mô hình hoá gắn liền với quy mô và độ phức tạp của sản phẩm. Ví dụ chúng ta chỉ cần mô phỏng một chiếc máy bay giấy, chúng ta chỉ cần gấp máy bay và thử nghiệm. Nếu máy bay không hoạt động chúng ta hoàn toàn có thể thay thế bằng một sản phẩm khác cho đến khi sản phẩm hoạt động hoàn hảo. Tuy nhiên để sản xuất một máy bay chiến đấu thì chúng ta không thể làm vậy vì chi phí để làm lại sản phẩm và vô cùng lớn và mất thời gian. Do vậy chúng ta cần phải mô hình hoá để hiểu rõ về sản phẩm trước khi bắt tay vào xây dựng nó.

Mô hình hoá giúp đạt được 4 mục đích sau:

- Cho phép **trực quan hóa** một hệ thống cần phải xây dựng
- Cho phép đặc tả **cấu trúc** và **hành vi** của hệ thống
- Đưa ra một **khuôn mẫu** (**template**) hướng dẫn chúng ta xây dựng một hệ thống.
- Tài liệu hóa lại **những quyết định** mà chúng ta đưa ra trong quá trình phát triển.

Mô hình hoá được coi như một bản vẽ thiết kế:

- Tuỳ thuộc vào công cụ, kỹ thuật và nền tảng mà chúng ta sử dụng, các phiên bản khác nhau của hệ thống có thể được tạo ra, tuy nhiên tất cả sẽ đều tuân thủ theo bản thiết kế (mô hình) ban đầu.

Nguyên lý của mô hình hoá:

- Có rất nhiều mô hình khác nhau, mỗi mô hình cung cấp **một góc nhìn riêng** về hệ thống.
- Việc lựa chọn mô hình nào phụ thuộc một cách mật thiết vào việc chúng ta **giải quyết yêu cầu** đặt ra như thế nào?
- Môt mô hình **duy nhất** không thể biểu diễn được cả một hệ thống.
- Mọi mô hình đều có thể được biểu diễn ở các **mức độ chi tiết** khác nhau.
- Những mô hình tốt giúp kết nối hệ thống với thực tế.

6.3.3 Ngôn ngữ mô hình hoá UML

Nguồn gốc ra đời của UML:

- 1980s: phân tích và thiết kế hướng cấu trúc
- 1990s: phân tích và thiết kế hướng đối tượng
- Giữa những năm 1990: nhiều hơn 50 phương pháp hướng đối tượng với rất nhiều định dạng khác nhau ra đời.
    - Fusion, Shlaer-Mellor, ROOM, Class-Relation, Wirfs-Brock, MOSES, Syntropy, BOOM, OOSD, OSA, BON, Catalysis, COMMA, HOOD, Ooram, DOORS…

➔ Một ngôn ngữ mô hình hóa chung nhất do đó được mong đợi ra đời. Và đó chính là ngôn ngữ UML ngày nay mà chúng ta sử dụng.

a) Định nghĩa

UML viết tắt của Unified Modeling Language là ngôn ngữ để:

- Trực quan hoá
- Đặc tả
- Xây dựng
- Tài liệu hóa mọi thành phần của một hệ thống phần mềm

b) Trực quan hoá

- Việc giao tiếp bằng các mô hình mang tính khái niệm (khái quát) thường dễ mắc lỗi trừ khi tất cả mọi người tham gia đều có chung một hệ quy chiếu về khái niệm.
- Có những thành phần của hệ thống phần mềm mà chúng ta không thể nắm rõ được cho đến khi chúng ta xây dựng mô hình cho chúng.
- Mô hình trong UML được biểu diễn một cách trực quan và mang ngữ nghĩa rõ ràng giúp việc giao tiếp dễ dàng giữa đội phát triển và khách hàng hoặc giữa các thành viên trong đội phát triển.

c) Đặc tả

Các mô hình UML mang ngữ nghĩa chính xác, tường minh và đầy đủ thông tin.

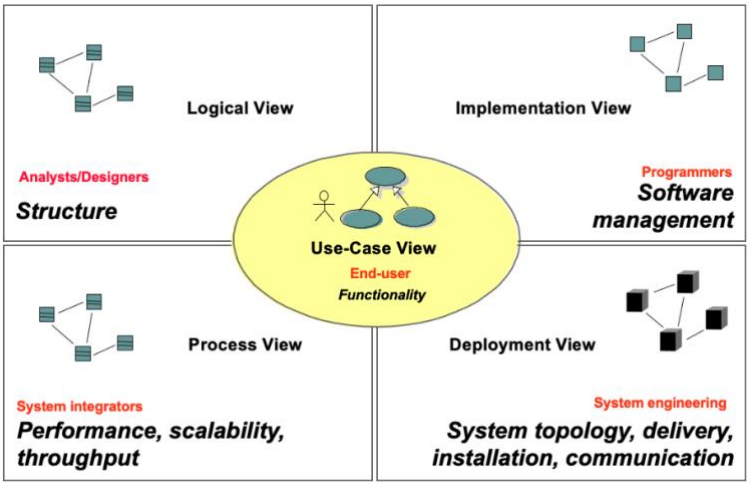
d) Xây dựng

Các mô hình UML có thể kết nối trực tiếp với nhiều loại ngôn ngữ lập trình khác nhau, ví dụ Java, C++, Visual Basic, bảng trong các hệ quản trị dữ liệu quan hệ hoặc kho lưu trữ lâu dài trong các hệ quản trị dữ liệu hướng đối tượng.

e) Tài liệu hóa

Ngôn ngữ UML hướng tới xây dựng tài liệu cho một hệ thống phần mềm, từ kiến trúc tới yêu cầu, kiểm thử, lập kế hoạch project và quản lý phiên bản.

Một mô hình duy nhất không thể mô hình hoá được toàn bộ hệ thống. Do vậy khi sử dụng UML để mô hình hoá hệ thống chúng ta đưa ra khái niệm **4+1 view.**



Hình 6-2 : 4+1 view trong mô hình hoá với UML[1]

[1] *Nguồn: The Rational Unified Process: An Introduction. Philippe Kruchten.*

6.3.4 Các thành phần của UML

Các thành phần của UML giúp mô hình hoá hai khía cạnh của hệ thống:

- Cấu trúc tĩnh (static structure) biểu diễn các thành phần của phần mềm (component, package, class) và các quan hệ giữa các thành phần đó.
- Hành vi động (dynamic behavior) biểu diễn cách các thành phần tương tác với nhau để thực hiện các thay đổi về trạng thái của hệ thống theo thời gian.

Các biểu đồ thông dụng của UML bao gồm:

- Use-case diagram: biểu đồ ca sử dụng
- Class diagram: biểu đồ class
- Object diagram: biểu đồ đối tượng
- State machine diagram: biểu đồ máy hữu hạn trạng thái
- Activity diagram: biểu đồ hoạt động
- Interaction diagram: biểu đồ tương tác
- Deployment diagram: biểu đồ triển khai

6.4 Các công cụ mô hình hoá yêu cầu phần mềm

Các công cụ mô hình hoá giúp biểu diễn nhiều khía cạnh khác nhau của hệ thống phần mềm với các công cụ thông dụng sau:

- Mô hình hoá cách thức dữ liệu được trung chuyển và xử lý: sử dụng **biểu đồ luồng dữ liệu**
- Mô hình hoá sự thay đổi trạng thái của các đối tượng dữ liệu trong hệ thống: sử dụng **biểu đồ máy hữu hạn trạng thái**
- Biểu diễn mối quan hệ giữa các đối tượng dữ liệu: **biểu đồ thực thể quan hệ**
- Mô hình hoá tương tác giữa người dùng và hệ thống: **biểu đồ ca sử dụng** và **biểu đồ hoạt động**

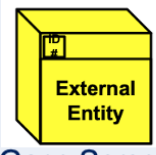
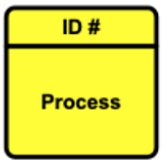
6.4.1 Biểu đồ luồng dữ liệu

Biểu đồ luồng dữ liệu (Data Flow Diagram – DFD) được sử dụng như một cách thức chính thống để biểu diễn cách thức một nghiệp vụ diễn ra trong thực tế.

- DFD biểu diễn các tiến trình nghiệp vụ (**business processes**) và cách thức dữ liệu trung chuyển giữa các tiến trình đó
- Tồn tại 2 loại tiến trình:
    - Tiến trình logic: chỉ biểu diễn nghiệp vụ mà không chứa các chi tiết về cài đặt hay triển khai trong thực tế
    - Tiến trình vật lý: chứa thêm thông tin triển khai trong thực tế

a) Các thành phần trong biểu đồ luồng dữ liệu

Bảng 6-1 : Các thành phần của biểu đồ DFD

Kho dữ liệu (data store)	- Số thứ tự - Tên (danh từ) - Mô tả - Có thể có 1 hoặc nhiều luồng dữ liệu đến - Có thể có 1 hoặc nhiều luồng dữ liệu ra	 Visio 2000  Gane Sarson DFD
Thực thể ngoài (external entity)	- Tên (danh từ) - Mô tả	 Visio 2000  Gane Sarson DFD
Tiến trình (process)	- Số thứ tự - Tên (cụm động từ) - Mô tả - Có thể có 1 hoặc nhiều luồng dữ liệu đến - Có thể có 1 hoặc nhiều luồng dữ liệu ra	 Visio 2000  Gane Sarson DFD
Luồng dữ liệu (data flow)	- Tên (danh từ) - Mô tả - Một hoặc nhiều liên kết đến các tiến trình	

b) Ví dụ về cách sử dụng biểu đồ luồng dữ liệu

Biểu diễn luồng dữ liệu cho tiến trình nghiệp vụ đăng ký khoá học mới với đặc tả như sau:

- Sinh viên đăng ký khoá học mới, hệ thống sẽ kiểm trả các khoá học sẵn có

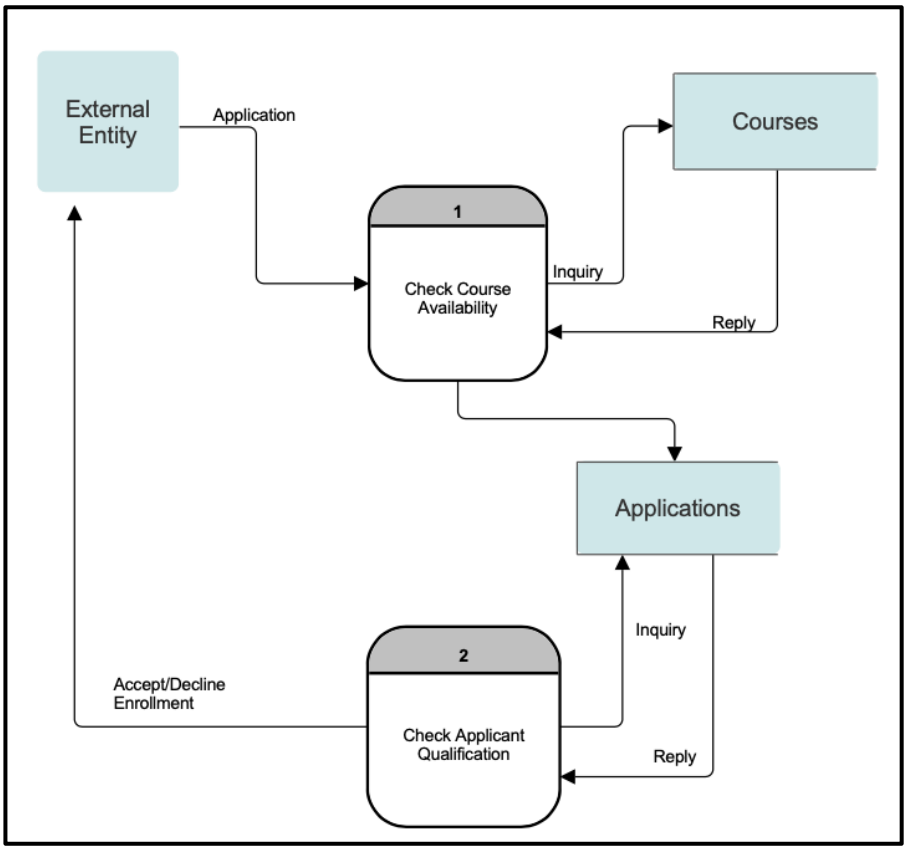
- Đơn đăng ký học của sinh viên sẽ được lưu vào kho lưu trữ
- Hệ thống đánh giá hồ sơ đăng ký học và phản hồi lại với sinh viên

Thực thể ngoài: Sinh viên

Tiến trình: kiểm tra khoá học và đánh giá hồ sơ

Kho dữ liệu: khoá học và đơn đăng ký

Biểu đồ luồng dữ liệu được thể hiện trên Hình 6-3.



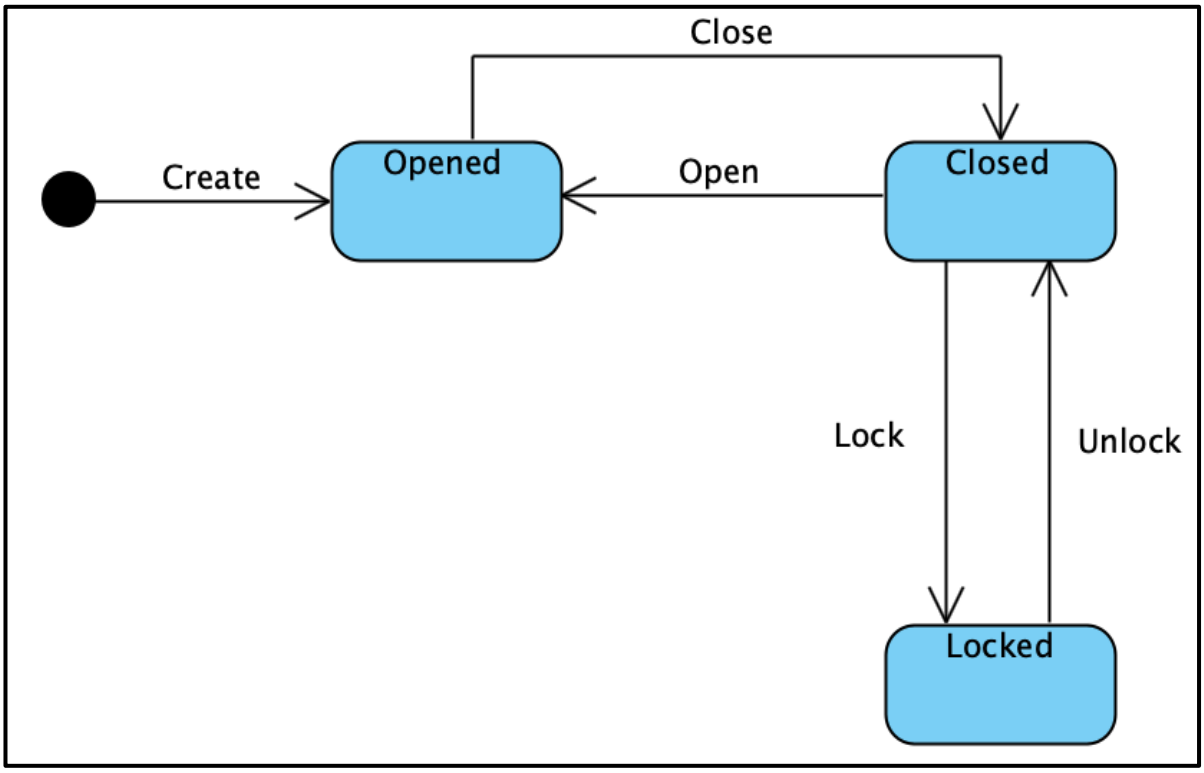
Hình 6-3 : Biểu đồ DFD đăng ký khoá học mới

6.4.2 Biểu đồ máy trạng thái

Biểu đồ máy trạng thái (State Machine Diagram – SMD) được sử dụng để mô hình hoá qúa trình chuyển trạng trạng thái của một đối tượng đơn (single object) thông qua một chuỗi các sự kiện. Ví dụ, mô hình hoá trạng thái của 1 cánh cửa.

- Đối tượng: Cửa (Door)
- Trạng thái của đối tượng: **closed** / **opened** / **locked**
- Sự kiện dẫn đến sự thay đổi trạng thái: **Open** / **Close** / **Lock** / **Unlock**

Sơ đồ máy trạng thái biểu diễn trạng thái của cánh cửa được thể hiện trên Hình 6-4.

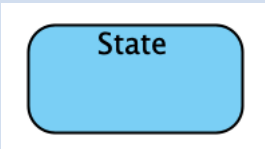


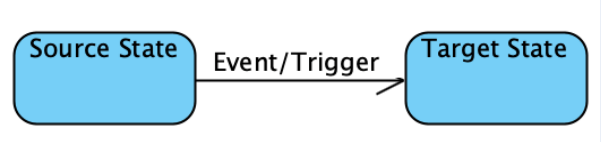
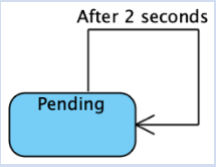
Hình 6-4 : Sơ đồ máy trạng thái của đối tượng Door

Có thể quan sát thấy, với một số trạng thái một số sự kiện không thể áp dụng. Ví dụ, Sự kiện Lock/Unlock chỉ áp dụng cho trạng thái Closed/Locked mà không áp dụng cho trạng thái Opened.

a) Các thành phần của biểu đồ máy trạng thái

Bảng 6-2 - Các thành phần của biểu đồ máy trạng thái

Thành phần	**Thông tin chính**	**Biểu tượng trong UML**
Trạng thái (State)	Tên của trạng thái	
Trạng thái bắt đầu (initial state)	Trạng thái bắt đầu có thể không cần đặt tên cụ thể	
Trạng thái kết thúc (final state)	Trạng thái kết thúc	

Chuyển trạng thái (Transition)	Trạng thái nguồn, trạng thái đích, sự kiện/ràng buộc điều kiện dẫn tới việc chuyển trạng thái	
Self-transition	Trạng thái nguồn và trạng thái đích trùng nhau	

Với các chuyển trạng thái (Transition): Việc phát sinh sự kiện có thể cần phải thực hiện dưới một số điều kiện ràng buộc khi đó các điều kiện sẽ được thêm vào bên cạnh các sự kiện.

b) Ví dụ

Yêu cầu: mô hình hoá cho trạng thái của thang máy theo đặc tả sau đây:

- Ban đầu thang máy được thiết lập ở tầng 1, trạng thái: ***On first floor***
- Thang máy có thể ở trạng thái đang di chuyển lên trên (***Moving up***) khi có lệnh yêu cầu đi lên (go up)
- Thang máy có thể ở trạng thái đang di chuyển xuống (***Moving down***) khi có lệnh yêu cầu đi xuống (go down)
- Thang có thể ở trạng thái rỗi (***Idle***) khi không có lệnh yêu cầu gì. Thang máy không thể chuyển trạng thái từ đang đi lên (***Moving up***) sang đang đi xuống (***Moving down***)
- Sau một khoảng thời gian nào đó (timeout) thì thang máy sẽ tự di chuyển về tầng 1 (***Moving to first floor***)

Các trạng thái của thang máy:

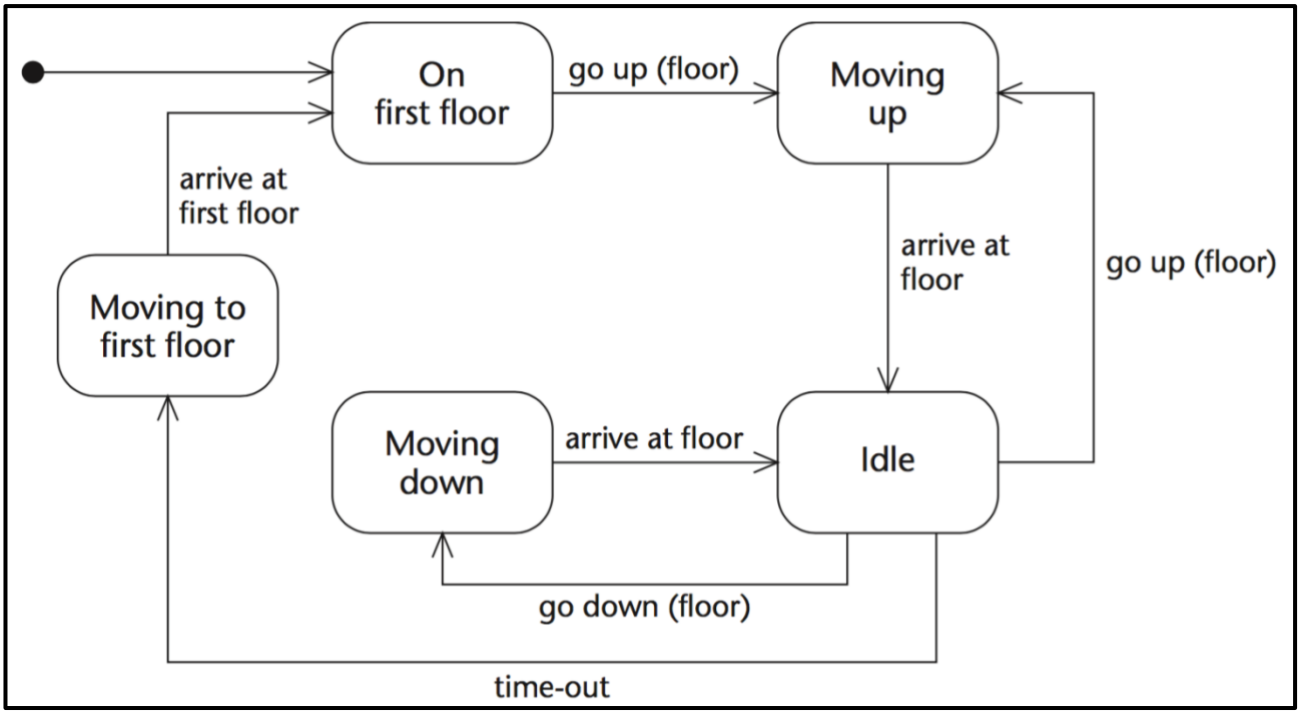
- On first floor
- Moving up
- Moving down
- Idle

- Moving to first floor

Các sự kiện kích hoạt:

- Go up
- Go down
- Arrive at floor
- Time out
- Arrive at first floor

Từ đó chúng ta vẽ được biểu đồ máy trạng thái như trên Hình 6-5.



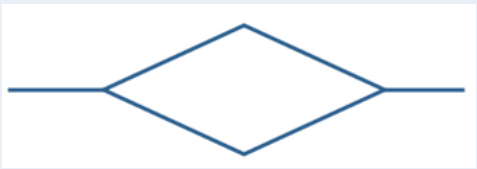
Hình 6-5 : Biểu đồ máy trạng thái thang máy

6.4.3 Biểu đồ mối quan hệ thực thể

Biểu đồ mối quan hệ thực thể (Entity Relationship Diagram – ERD) Được sử dụng để mô hình hoá các thực thể dữ liệu và mối quan hệ giữa chúng trong các hệ cơ sở dữ liệu quan hệ. ERD có thể được sử dụng để đặc tả yêu cầu về dữ liệu cũng như được sử dụng trong pha thiết kế (design).

a) Các thành phần

Bảng 6-3 : Các thành phần ERD

Thành phần	**Thông tin**	**Biểu tượng**
Thực thể (enity)	Tên thực thể (danh từ)	
Mối quan hệ giữa các thực thể	Tên quan hệ (động từ)	
Thuộc tính thực thể hoặc thuộc tính quan hệ	Tên thuộc tính (danh từ)	

b) Ví dụ

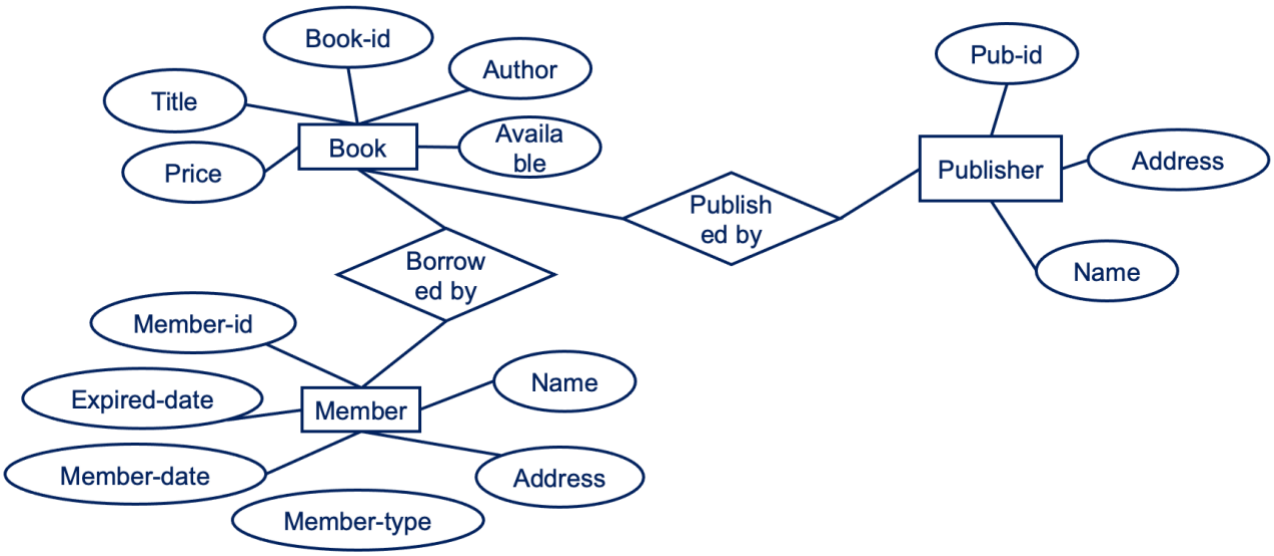
Hệ thống quản lý thư viện yêu cầu quản lý các đối tượng dữ liệu sau:

- Book: book-id, title, author, price, availble
- Publisher: pub-id, address, name
- Member: member-id, member-date, expired-date, member-type, address, name

Quan hệ giữa các thực thể:

- Book – published by – Publisher
- Book – borrowed by – Member

Biểu đồ mối quan hệ thực thể giữa các đối tượng trong hệ thống quản lý thư viện nói trên được biểu diễn trong Hình 6-6.



Hình 6-6 : Sơ đồ quan hệ thực thể hệ thống quản lý thư viện

6.4.4 Biểu đồ ca sử dụng

Biểu đồ ca sử dụng (Usecase Diagram – UC) thường được dùng để mô hình hoá khía cạnh động của hệ thống phần mềm. Mô tả các yêu cầu chức năng của phần mềm về các trường hợp sử dụng qua tương tác với các tác nhân của hệ thống. Biểu đồ ca sử dụng thường được dùng để:

- Giao tiếp
- Xác định yêu cầu
- Tài liệu

Các thành phần trong biểu đồ ca sử dụng:

- Tác nhân (Actor) là bất kỳ cái gì có thể tương tác với hệ thống. Tên của tác nhân: Danh từ chung, không số nhiều
- Use-case biểu diễn 1 chuỗi các sự kiện (hành động) thực hiện bởi phần mềm qua tương tác với tác nhân và trả về 1 kết quả thực hiện nào đó
    - Tên của UC: Động từ + Danh từ
    - Xác định rõ ràng chức năng cụ thể của hệ thống
    - Ngoại lệ: login, logout, sign-in, sign-out

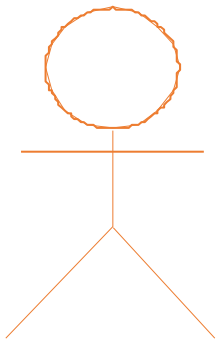
a) Tác nhân

- Các tác nhân thể hiện các vai trò khác nhau mà một đối tượng người dùng hoặc một hệ thống khác có thể thực hiện với hệ thống phần mềm

- Tác nhân có thể là con người, một thiết bị máy móc hoặc 1 hệ thống phần mềm
- Tác nhân cũng có thể là 1 thiết bị ngoại vi hoặc 1 cơ sở dữ liệu

- Tác nhân có thể trao đổi thông tin và tương tác với hệ thống phần mềm
  - Có thể là đối tượng đưa thông tin
  - Cũng có thể là đối tượng nhận thông tin
- **Quan trọng**: Tác nhân phải **nằm ngoài** hệ thống đang xét

Kí hiệu của tác nhân trong biểu đồ UC UML như trên Hình 6-7.



Hình 6-7 : Kí hiệu Actor

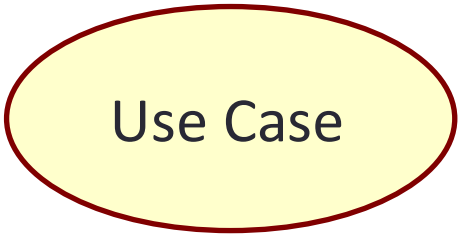
Tác nhân cũng thể hiện một vai trò cụ thể đối với hệ thống. Ví dụ một cá nhân cụ thể X, có thể vừa đóng vai trò là Professor, vừa đóng vai trò là Student trong hệ thống quản lý trường Đại học H. Khi đó cá nhân X có thể thực hiện các tương tác với hệ thống X với tư cách là một Professor đồng thời cũng có thể đóng vai trò là một Student.

b) Ca sử dụng

Ca sử dụng Use-case (UC) thể hiện một chức năng của hệ thống có thể thực hiện để tương tác với tác nhân:

- Mỗi UC mô hình hoá một hội thoại giữa một hoặc nhiều tác nhân và hệ thống phần mềm
- Thể hiện một **chuỗi** các hành động mà hệ thống thực hiện và **trả về một kết quả** nào đó cho tác nhân

Kí hiệu UC trong biểu đồ UC của UML như trên Hình 6-8.



Hình 6-8 : Biểu tượng UC

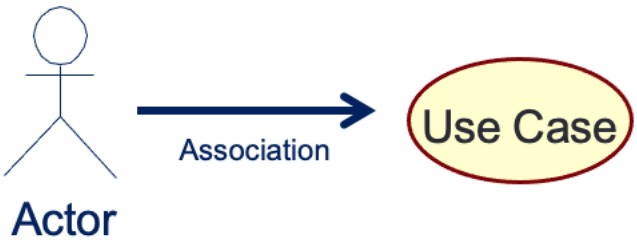
c) Mối quan hệ giữa các thành phần trong biêu đồ UC

Trong biểu đồ UC chúng ta sẽ xét đến các mối quan hệ sau đây:

- Mối quan hệ giữa tác nhân – usecase
- Mối quan hệ giữa usecase và usecase
- Mối quan hệ giữa tác nhân và tác nhân

Mối quan hệ giữa tác nhân và usecase

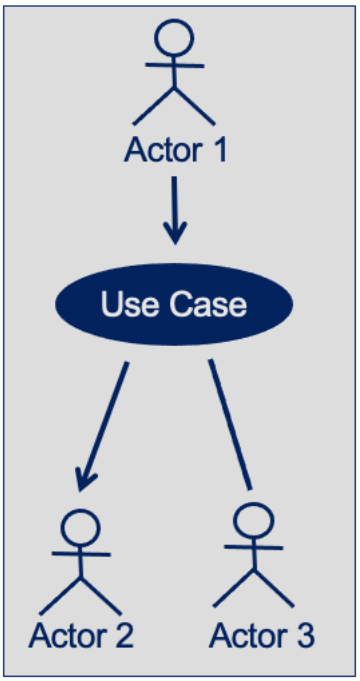
- Thiết lập tương tác giữa các tác nhân với các ca sử dụng có liên quan
- Liên kết: Association
- Thể hiện giao tiếp giữa tác nhân và ca sử dụng
- Giao tiếp có thể thực hiện theo hai chiều
    - Tác nhân gửi thông tin đến hệ thống
    - Hệ thống xử lý thông tin và phản hồi lại tác nhân
- Biểu diễn bằng một đường thẳng liền nét, có thể có hướng hoặc không có hướng
- Khi có hướng, liên kết sẽ thể hiện đối tượng khởi tạo quá trình giao tiếp



Hình 6-9 : Association - Liên kết giữa tác nhân và ca sử dụng

Khi liên kết là có hướng:

- Chiều mũi tên sẽ thể hiện ai là người bắt đầu quá trình tương tác



Hình 6-10 : Chiều tương tác liên kết

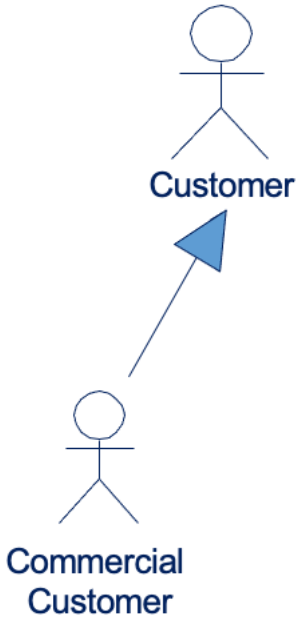
Ví dụ trên Hình 6-10:

- Actor 1 là người bắt đầu quá trình tương tác với hệ thống thông qua Use Case
- Actor 2 là người tham gia vào quá trình tương tác sau này, do yêu cầu của hệ thống
- Actor 3 có thể tham gia vào quá trình tương tác tại một thời điểm nào đó

➔ **Quan trọng**: Khi có nhiều hơn 1 actor tham gia tương tác, cần xác định: actor chính (**primary**) và các actor phụ (**secondary**).

Mối quan hệ giữa tác nhân và tác nhân

Quan hệ kế thừa (Generalization) thể hiện sự mở rộng của tác nhân. Tác nhân con kế thừa các liên kết mà tác nhân cha có và có thể thay thế cha thực hiện các hành động đó. Quan hệ kế thừa thể hiện bằng mũi tên có đầu là một tam giác, hướng về tác nhân cha. Tác nhân con đặc biệt hơn cha và có thể có thêm các tương tác mà cha không có với hệ thống.

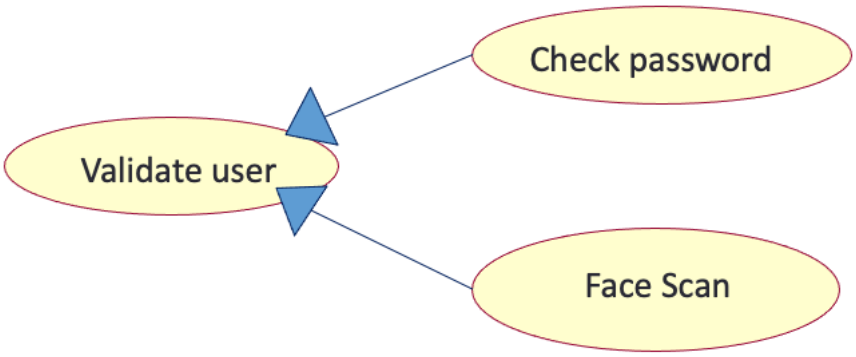


Hình 6-11 : Quan hệ kế thừa giữa hai tác nhân

Mối quan hệ giữa ca sử dụng và ca sử dụng

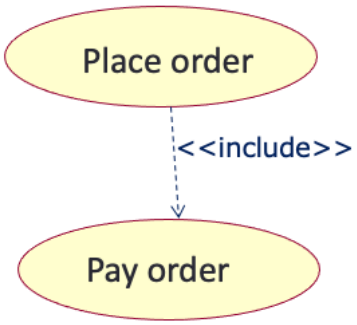
Giữa hai ca sử dụng có thể có các quan hệ sau

- Quan hệ kế thừa: UC con kế thừa mọi tương tác mà UC cha có. UC con đặc biệt hơn UC cha do có thể có những thay đổi như: các hành động của hệ thống, tương tác với các tác nhân khác ngoài tác nhân chính của UC cha.



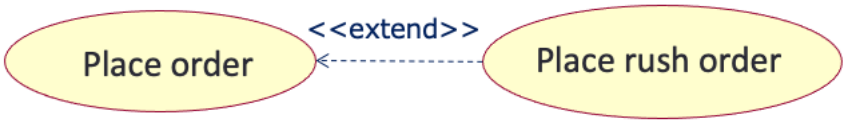
Hình 6-12 : Quan hệ kế thừa giữa hai UC

- Quan hệ bao gồm: Biểu diễn bằng một đường nét đứt có hướng. Chiều mũi tên hướng về UC được gọi tới. Kí hiệu thêm <<include>> trên quan hệ. Ý nghĩa của quan hệ bao gồm thể hiện sự bắt buộc phải gọi tới một UC khác trong quá trình thực hiện UC cơ sở.



Hình 6-13 : Quan hệ bao gồm

- Quan hệ mở rộng biểu diễn bằng một đường nét đứt có hướng, chiều mũi tên hướng về UC cơ sở và có thêm kí hiệu <<extend>> trên quan hệ. Quan hệ mở rộng thể hiện sự tuỳ chọn, có thể thực hiện hoặc có thể không thực hiện trong quá trình thực hiện UC chính.



Hình 6-14 : Quan hệ mở rộng

Biểu đồ ca sử dụng chỉ thể hiện sự tương tác giữa các thành phần bên ngoài hệ thống và hệ thống mà không thể hiện một cách chi tiết cách thức thực hiện các tương tác này. Để biểu diễn cho điều đó, các ca sử dụng thường có thêm thông tin chi tiết về luồng thực hiện hành động tương tác giữa tác nhân và hệ thống: **đặc tả chi tiết usecase**.

Luồng hành động này chia thành 3 loại:

- Luồng hành động chính
- Luồng hành động thay thế
- Luồng xử lý lỗi

Ví dụ về đặc tả chi tiết ca sử dụng đăng nhập (Bảng 6-4).

Bảng 6-4 : Đặc tả chi tiết UC Đăng nhập

<table>
<tr><td>Mã Use case</td><td>UC001</td><td>Tên Use case</td><td>Đăng nhập</td></tr>
<tr><td>Tác nhân</td><td colspan="3">Khách</td></tr>
<tr><td>Tiền điều kiện</td><td colspan="3">Không</td></tr>
<tr><td>Luồng sự kiện chính
(Thành công)</td><td colspan="3">
<table>
<tr><th>STT</th><th>Thực hiện bởi</th><th>Hành động</th></tr>
<tr><td>1.</td><td>Khách</td><td>chọn chức năng Đăng nhập</td></tr>
<tr><td>2.</td><td>Hệ thống</td><td>hiển thị giao diện đăng nhập</td></tr>
<tr><td>3.</td><td>Khách</td><td>nhập email và mật khẩu (mô tả phía dưới *)</td></tr>
<tr><td>4.</td><td>Khách</td><td>yêu cầu đăng nhập</td></tr>
<tr><td>5.</td><td>Hệ thống</td><td>kiểm tra xem khách đã nhập các trường bắt buộc nhập hay chưa</td></tr>
<tr><td>6.</td><td>Hệ thống</td><td>kiểm tra email và mật khẩu có hợp lệ do khách nhập trong hệ thống hay không</td></tr>
<tr><td>7.</td><td>Hệ thống</td><td>gọi use case “Tạo menu” kèm email của khách đăng nhập</td></tr>
</table>
</td></tr>
<tr><td>Luồng sự kiện thay thế</td><td colspan="3">
<table>
<tr><th>STT</th><th>Thực hiện bởi</th><th>Hành động</th></tr>
<tr><td>6a.</td><td>Hệ thống</td><td>thông báo lỗi: Cần nhập các trường bắt buộc nhập nếu khách nhập thiếu</td></tr>
<tr><td>7a.</td><td>Hệ thống</td><td>thông báo lỗi: Email và/hoặc mật khẩu chưa đúng nếu không tìm thấy email và mật khẩu trong hệ thống</td></tr>
<tr><td>7b.</td><td>Hệ thống</td><td>thông báo lỗi: Tài khoản bị khoá, nếu email/mật khẩu đúng như tài khoản đang bị admin khoá.</td></tr>
<tr><td>7c1.</td><td>Hệ thống</td><td>gọi use case “Thay đổi mật khẩu theo yêu cầu” nếu đúng email và mật khẩu; nhưng người dùng được đánh dấu là cần thay đổi mật khẩu</td></tr>
<tr><td>7c2.</td><td>Hệ thống</td><td>gọi use case “Tạo menu” kèm email của khách đăng nhập</td></tr>
</table>
</td></tr>
<tr><td>Hậu điều kiện</td><td colspan="3">Không</td></tr>
</table>

6.4.5 Biểu đồ hoạt động

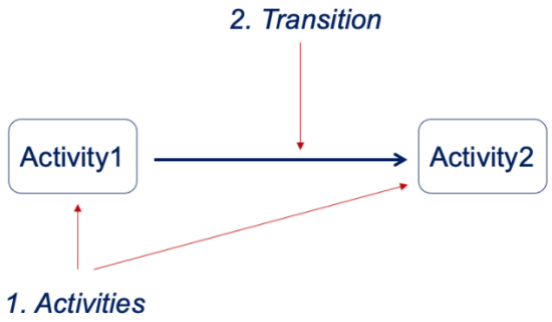
Biểu đồ hoạt động (Activity Diagram (AD))

- Biểu đồ hoạt động biểu diễn các khía cạnh động (dynamic aspects) của hệ thống phần mềm, thể hiện các hành vi cụ thể phần mềm tương tác với các tác nhân bên ngoài
- Biểu đồ hoạt động thể hiện luồng hoạt động của hệ thống và tác nhân liên quan (flow of activities)
- Biểu đồ hoạt động giúp làm rõ luồng sự kiện và các hoạt động của từng ca sử dụng hoặc giữa các ca sử dụng với nhau
- Thường được sử dụng để mô hình hoá các tiến trình nghiệp vụ trong giai đoạn phân tích và xây dựng yêu cầu phần mềm

- Biểu đồ hoạt động là một trong những công cụ hữu ích để phân tích và hiểu rõ về các luồng nghiệp vụ của hệ thống phần mềm

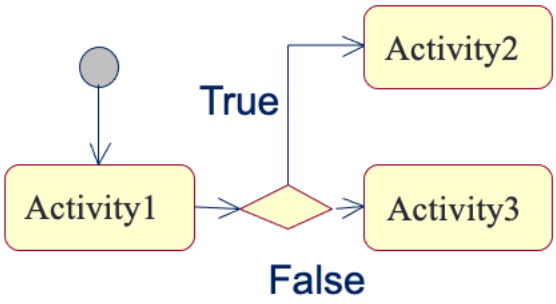
a) Các thành phần của biểu đồ hoạt động

- Biểu đồ hoạt động được biểu diễn như một cấu trúc đồ thị với các đỉnh và các cạnh
- Đỉnh trong đồ thị:
    - Hoạt động (Activity)



Hình 6-15 : Hoạt động

- Một hoạt động được định nghĩa là một nhiệm vụ nào đó cần phải được thực hiện. Mỗi hoạt động có thể được theo sau bởi một hoạt động khác (theo chuỗi các hoạt động kế tiếp nhau). Mỗi hoạt động có thể được kích hoạt bằng một hoặc nhiều hơn một sự kiện. Kết quả trả về của mỗi hoạt động có thể kích hoạt các hoạt động khác hoặc kích hoạt một tiến trình khác
- Điểm quyết định (Decision Node): Giao điểm quyết định cũng là một hoạt động trong biểu đồ AD. Tại giao điểm quyết định có nhiều hơn một nhánh rẽ, gắn liền với một điều kiện/ràng buộc. Tuỳ thuộc vào giá trị của điều kiện tại giao điểm quyết định mà hoạt động tương ứng sẽ được kích hoạt. Kí hiệu bằng hình thoi với các nhánh rẽ tương ứng với các trường hợp kiểm chứng của điều kiện.



Hình 6-16 : Giao điểm quyết định

- Cạnh (chuyển tiếp / transition) nối các đỉnh với nhau để tạo nên luồng hoạt động mô tả cho từng nghiệp vụ của hệ thống
- Điểm chia và điểm nối
    - Thể hiện sự đồng bộ của các luồng hoạt động trong biểu đồ AD
    - Nút chia (Fork): là một nút điều khiển trên biểu đồ hoạt động giúp chia luồng thành nhiều luồng hoạt động đồng thời
        - 1 luồng đến
        - Nhiều luồng ra đồng thời
    - Nút ghép/nối (Join): là một nút điều khiển giúp đồng bộ hoá nhiều luồng
        - Nhiều luồng đến
        - 1 luồng ra

Tham chiếu / Gọi các biểu đồ hoạt động con (sub-activity diagram). Một số hoạt động trong biểu đồ AD là một nhóm các luồng hoạt động khác. Tham chiếu tới một biểu đồ hoạt động con giúp chúng ta tổ chức biểu đồ AD tốt hơn, đặc biệt khi luồng nghiệp vụ của chức năng tương ứng là phức tạp và cồng kềnh.

b) Ví dụ

Ca sử dụng Xử lý đơn hàng (Process Order):

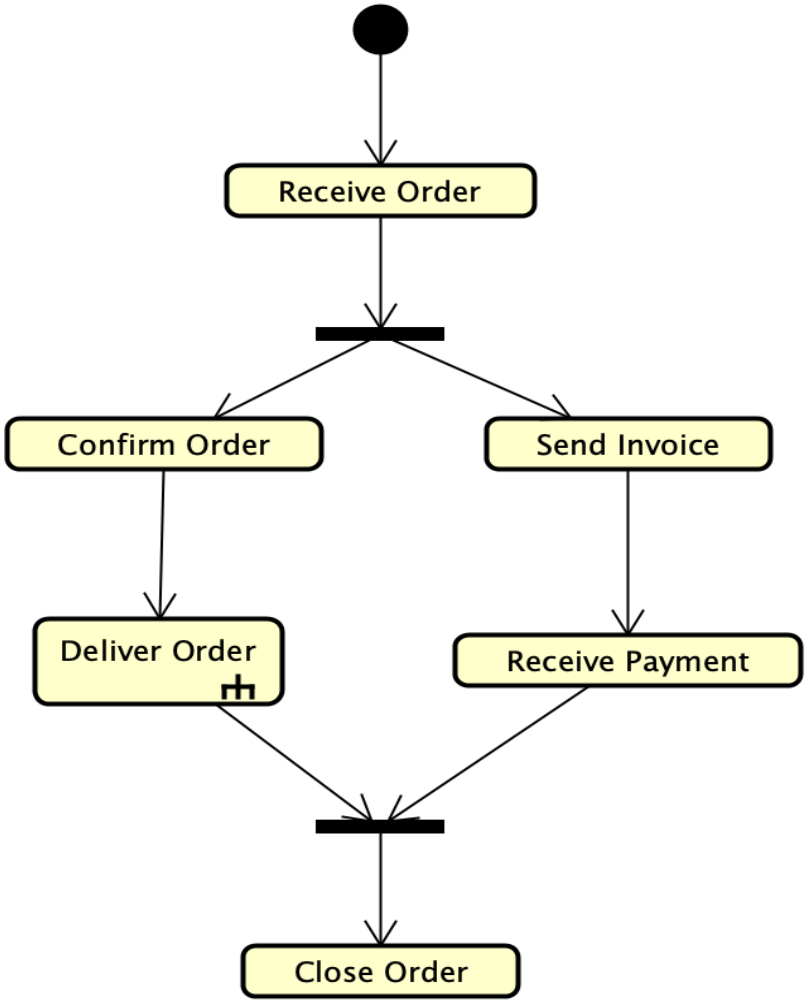
Quá trình xử lý đơn hàng được mô tả nghiệp vụ như sau:

1. Hệ thống nhận được đơn hàng mới
2. Hệ thống gửi xác nhận đơn hàng đồng thời tạo hoá đơn (invoice) gửi cho khách hàng
3. Hệ thống xác nhận thanh toán đồng thời tiến hành quy trình giao hang

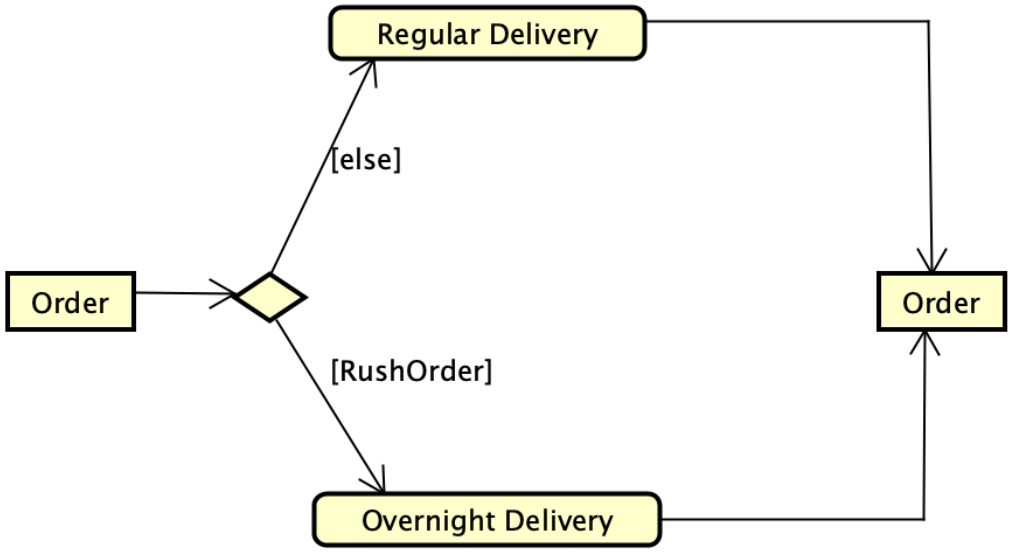
4. Khi đơn hàng được bàn giao thành công quá trình xử lý đơn hàng kết thúc

Quy trình giao hàng:

1. Nếu khách hàng lựa chọn hình thức giao hàng hoả tốc (Rush Order). Quá trình giao hàng tiến hành ngay trong đêm
2. Nếu không phải là hình thức giao hàng hoả tốc thì tiến hành giao hàng thông thường



Hình 6-17 : Biểu đồ hoạt động UC Process Order



Hình 6-18 : Luồng hành động giao hàng

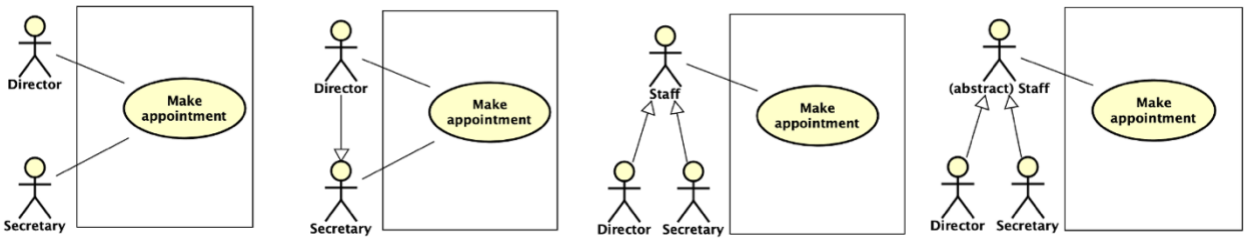
6.5 Hướng dẫn bài tập: Mô hình hoá yêu cầu phần mềm bằng biểu đồ UC

6.5.1 Sử dụng quan hệ giữa các tác nhân

a) Câu hỏi:

Trong công ty, chỉ giám đốc hoặc thư ký là được phép tạo cuộc hẹn phỏng vấn, ngoài ra thì không ai khác được phép. Trong các biểu đồ sau đây, biểu đồ nào phản ánh đúng yêu cầu nói trên?

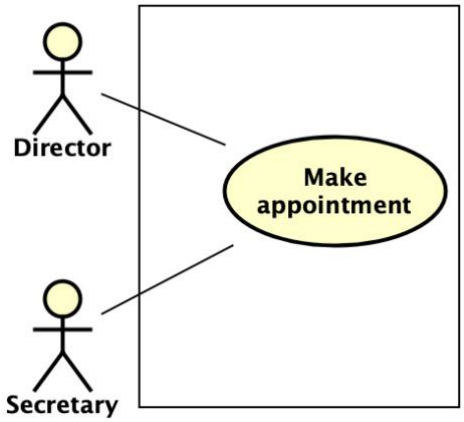
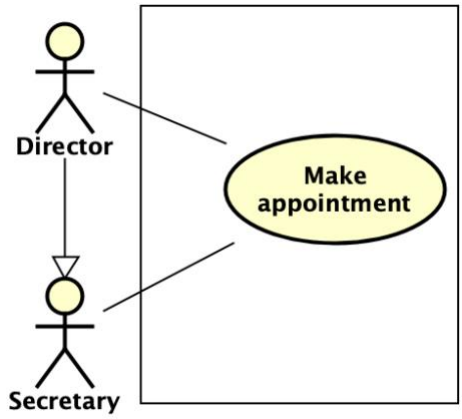
Trong các biểu đồ UC dưới đây, biểu đồ nào cho phép thể hiện đúng tình huống này?

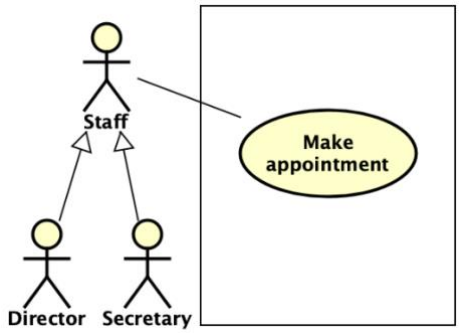
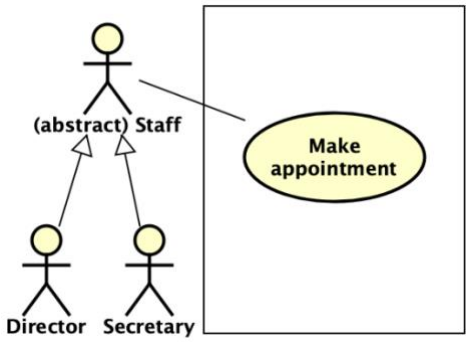


Hình 6-19 : Các tình huống a) b) c) d)

b) Phân tích

Bảng 6-5 - Phân tích tình huống Hình 6-19

Biểu đồ UC	Quan sát và phân tích
	2 tác nhân cùng có liên kết với UC Make Appointment ▪ Director và Secretary cùng tham gia thực hiện Make Appointment SAI so với tình huống đưa ra: chỉ Director HOẶC Secretary được phép thực hiện Make Appointement
	2 tác nhân cùng có liên kết với UC Make Appointment. Tác nhân Director kế thừa tác nhân Secretary. ▪ Director và Secretary cùng tham gia thực hiện Make Appointment ▪ Director kế thừa Secretary nên kế thừa mọi quan hệ mà Secretary có. Director cũng có thể thực hiện Make Appointment thay thế cho Secretary ➔ 2 vai trò Director có thể đồng thời gia thực hiện Make Appointement ▪ Vậy theo biểu đồ này, để thực hiện UC Make Appointement • Director ⋀ Director • Director ⋀ Secretary

	***SAI** so với tình huống đưa ra: chỉ Director **HOẶC** Secretary được phép thực hiện Make Appointement*
	2 tác nhân kế thừa một tác nhân khác Staff, Staff có thể thực hiện Make Appointment ▪ Director và Secretary cùng kế thừa Staff nên đều có thể thực hiện Make Appointement ▪ Chỉ một vai trò hoặc Staff hoặc Director hoặc Secretary thực hiện Make Appointement ▪ Vậy theo biểu đồ này, để thực hiện UC Make Appointement o Director ⋁ Secretary ⋁ Staff ***SAI** so với tình huống đưa ra: chỉ Director **HOẶC** Secretary được phép thực hiện Make Appointement*
	2 tác nhân kế thừa một tác nhân khác Staff, Staff có thể thực hiện Make Appointment ▪ Director và Secretary cùng kế thừa Staff nên đều có thể thực hiện Make Appointement ▪ Staff được định nghĩa là 1 tác nhân trừu tượng (stereotype) cho nên về

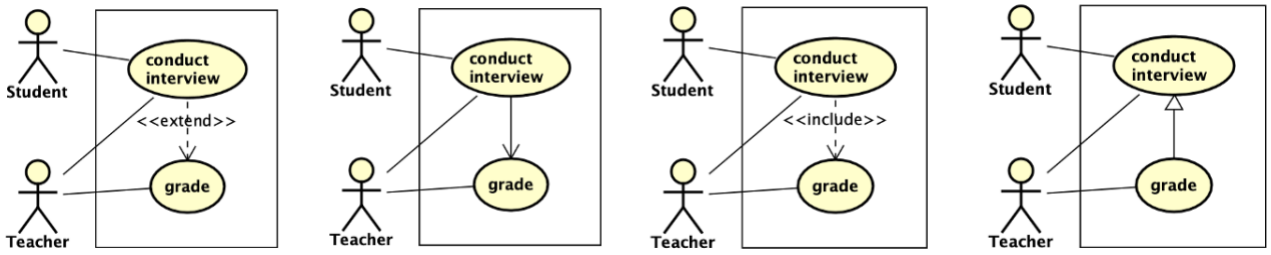
	bản chất không tồn tại một vai trò cụ thể là Staff trong hệ thống ▪ Chỉ một vai trò hoặc Director hoặc Secretary thực hiện Make Appointement ▪ Vậy theo biểu đồ này, để thực hiện UC Make Appointement • Director ⋁ Secretary ***ĐÚNG*** *so với tình huống đưa ra: chỉ Director* ***HOẶC*** *Secretary được phép thực hiện Make Appointement.*

6.5.2 Sử dụng quan hệ <<include>> giữa các ca sử dụng

a) Câu hỏi:

Một giáo viên tiến hành phỏng vấn sinh viên, trong quá trình này giáo viên được yêu cầu sẽ phải đánh giá và cho điểm sinh viên.

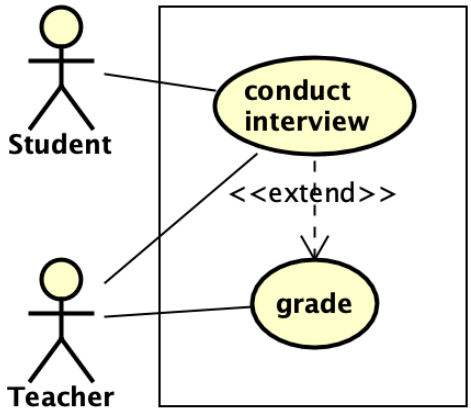
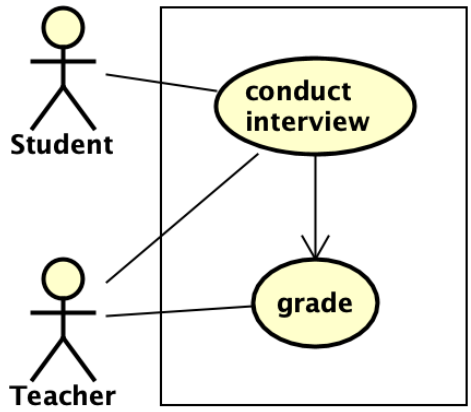
Trong các biểu đồ UC dưới đây, biểu đồ nào cho phép thể hiện đúng tình huống này?

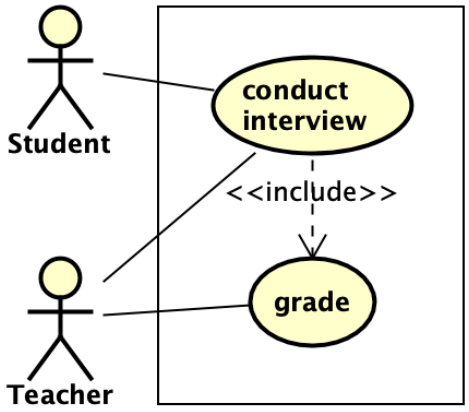
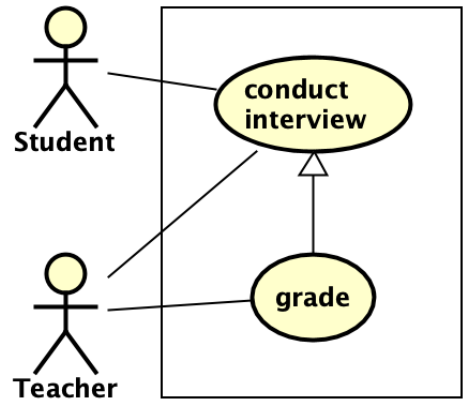


Hình 6-20 : Các biểu đồ a) b) c) d)

b) Phân tích

Bảng 6-6 - Phân tích các tình huống trong Hình 6-20

Biểu đồ UC	Quan sát và phân tích
	2 tác nhân Student và Teacher liên kết với UC Conduct Interview. UC Conduct Interview có quan hệ mở rộng với UC Grade ▪ UC Conduct Interview là UC cơ bản ▪ UC Grade là UC mở rộng ▪ Student và Teacher cùng tham gia thực hiện UC Conduct Interview ▪ Trong quá trình thực hiện UC Conduct Interview, có thể Teacher sẽ thực hiện UC Grade ***SAI** so với tình huống đưa ra: Quá trình thực hiện UC Conduct Interview, Teacher luôn phải thực hiện Grade.*
	2 tác nhân Student và Teacher liên kết với UC Conduct Interview UC Conduct Interview có quan hệ liên kết với UC Grade ▪ Không tồn tại quan hệ liên kết (association) giữa 2 UC ▪ Student và Teacher cùng tham gia thực hiện UC Conduct Interview ▪ Teacher tham gia thực hiện UC Grade

	***SAI** so với tình huống đưa ra: Quá trình thực hiện UC Conduct Interview, Teacher luôn phải thực hiện Grade*
	2 tác nhân Student và Teacher liên kết với UC Conduct Interview UC Conduct Interview có quan hệ bao gồm với UC Grade ▪ UC Conduct Interview là UC cơ bản ▪ UC Grade là UC được bao gồm ▪ Student và Teacher cùng tham gia thực hiện UC Conduct Interview ▪ Trong quá trình thực hiện UC Conduct Interview, Teacher sẽ phải thực hiện UC Grade ***ĐÚNG** so với tình huống đưa ra: Quá trình thực hiện UC Conduct Interview, Teacher luôn phải thực hiện Grade.*
	2 tác nhân Student và Teacher liên kết với UC Conduct Interview UC Grade kế thừa từ UC Conduct Interview ▪ UC Grade kế thừa mọi quan hệ mà UC Conduct Interview có và kế thừa mọi hoạt động của Conduct

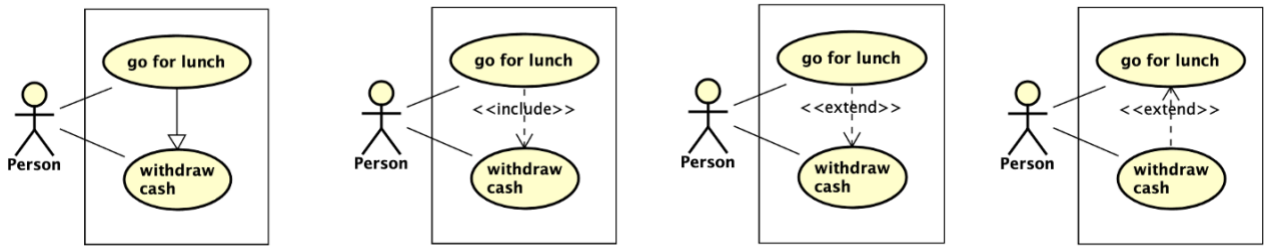
	Interview và có thể có thêm những hoạt động khác ▪ UC Grade có thể được **thực hiện bởi Student** do Student có liên kết với UC Conduct Interview, UC Grade kế thừa từ UC Conduct Interview nên cũng kế thừa liên kết này ▪ Student và Teacher cùng tham gia thực hiện UC Conduct Interview ▪ Teacher tham gia thực hiện UC Grade ***SAI** so với tình huống đưa ra: Quá trình thực hiện UC Conduct Interview, Teacher luôn phải thực hiện Grade.*

6.5.3 Sử dụng quan hệ <<extend> giữa các ca sử dụng

a) Câu hỏi:

Một người đi ăn trưa, trong quá trình đó có thể cần thiết phải rút tiền từ máy ATM

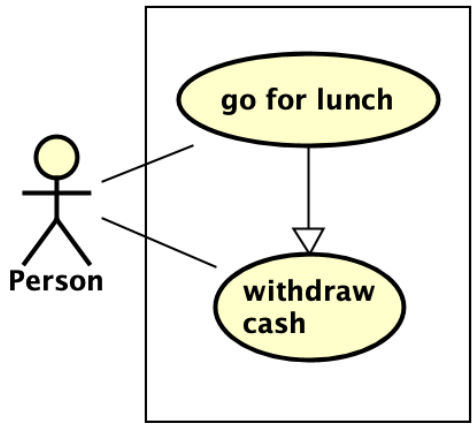
Trong các biểu đồ UC dưới đây, biểu đồ nào cho phép thể hiện đúng tình huống này?

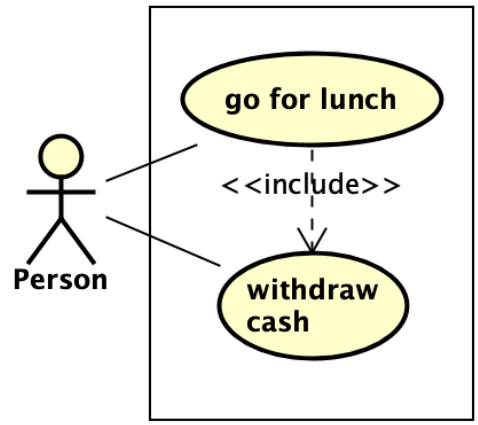
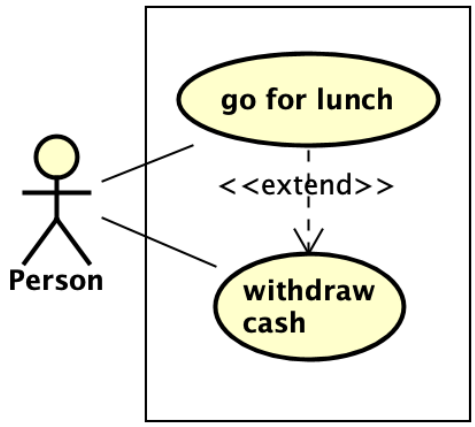
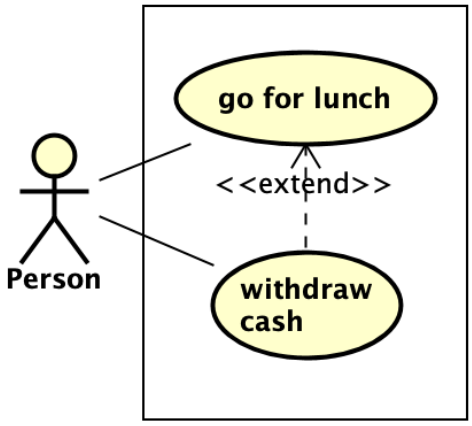


Hình 6-21: Các tình huống a) b) c) d)

b) Phân tích

Bảng 6-7 - Phân tích các tình huống trong Hình 6-21

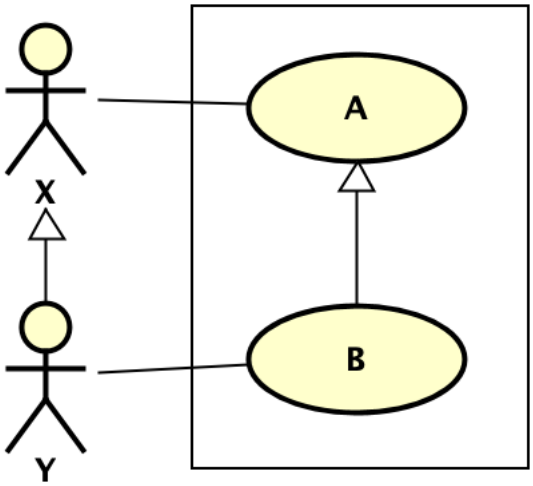
Biểu đồ UC	Quan sát và phân tích
	Tác nhân Person có quan hệ liên kết với 2 UC Go for lunch và Withdraw cash. UC Go for lunch kế thừa từ UC Withdraw cash. ▪ UC Go for lunch kế thừa UC Withdraw cash nên kế thừa mọi quan hệ và hành vi của UC Withdraw cash • Quá trình thực hiện Go for lunch có thể bao hàm cả quá trình rút tiền từ máy ATM • UC Go for lunch sẽ kế thừa luôn quan hệ liên kết của UC Withdraw cash với Person ➔ Khi thực hiện Go for lunch cần 2 tác nhân vai trò Person ***SAI*** *so với tình huống đưa ra: Quá trình thực hiện Go for lunch và Withdraw cash chỉ do 1 tác nhân Person tham gia.*

	Tác nhân Person có quan hệ liên kết với 2 UC Go for lunch và Withdraw cash. UC Go for lunch bao gồm UC Withdraw cash. ▪ Quá trình thực hiện Go for lunch sẽ bắt buộc gọi tới UC Withdraw cash ***SAI*** *so với tình huống đưa ra: Quá trình thực hiện Go for lunch và Withdraw cash chỉ do 1 tác nhân Person tham gia.*
	Tác nhân Person có quan hệ liên kết với 2 UC Go for lunch và Withdraw cash. UC Withdraw cash mở rộng từ UC Go for lunch ▪ Quá trình thực hiện UC Withdraw cash, có thể sẽ cần gọi tới UC Go for lunch ***SAI*** *so với tình huống đưa ra: Quá trình thực hiện Go for lunch và Withdraw cash chỉ do 1 tác nhân Person tham gia.*
	Tác nhân Person có quan hệ liên kết với 2 UC Go for lunch và Withdraw cash. UC Go for lunch mở rộng từ UC Withdraw cash ▪ Quá trình thực hiện UC Go for lunch, có thể sẽ cần gọi tới UC Withdraw cash

	***ĐÚNG** so với tình huống đưa ra: Quá trình thực hiện Go for lunch, **có thể** sẽ cần phải thực hiện Withdraw cash.*

6.5.4 Sử dụng quan hệ kế thừa giữa các ca sử dụng

Cho sơ đồ UC như trên Hình 6-22.



Hình 6-22 : Sơ đồ UC minh hoạ quan hệ kế thừa

Quan sát:

- Tác nhân Y kế thừa từ tác nhân X
- Tác nhân X liên kết với UC A
- Tác nhân Y liên kết với UC B
- UC B kế thừa từ UC A

Ý nghĩa của các quan hệ sử dụng trong biểu đồ

- Tác nhân Y kế thừa từ tác nhân X nên kế thừa mọi quan hệ mà X có
  - Y sẽ kế thừa liên kết với UC A từ X
- Y có thể thay thế X để thực hiện A
- UC B kế thừa từ UC A do đó cũng kế thừa mọi quan hệ mà A có
  - UC B sẽ kế thừa liên kết với X thông qua X liên kết với A
- Tác nhân X sẽ tham gia thực hiện UC B thông qua liên kết với B kế thừa từ A
- Tác nhân Y kế thừa từ X nên sẽ có thể thay thế X tương tác với B

Các tác nhân tham gia thực hiện UC B:

- Y ∧ X
- Y ∧ Y

6.6 Hướng dẫn bài tập: Mô hình hoá yêu cầu phần mềm bằng biểu đồ AD

6.6.1 Xác định các tác nhân và usecase từ đặc tả

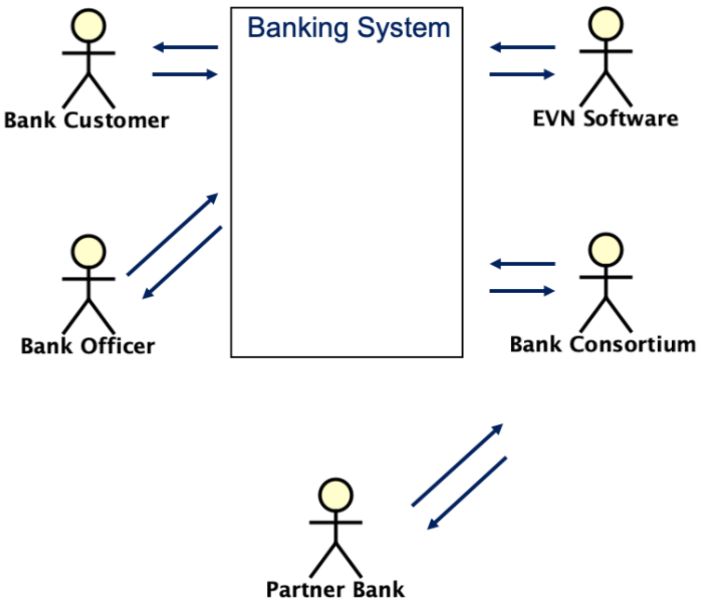
a) Đặc tả hệ thống

- Hệ thống Ngân hàng Trực tuyến, cho phép kết nối mạng liên ngân hàng, giao tiếp với khách hàng của ngân hàng thông qua ứng dụng web. Để thực hiện giao dịch, khách hàng phải đăng nhập vào phần mềm. Khách hàng có thể thay đổi mật khẩu hoặc xem thông tin cá nhân.
- Khách hàng có thể lựa chọn bất kỳ hình thức giao dịch nào: chuyển khoản (mạng nội bộ và liên ngân hàng), truy vấn số dư, truy vấn lịch sử giao dịch, thanh toán hóa đơn tiền điện (qua phần mềm của Tập đoàn Điện lực).
- Trong giao dịch chuyển khoản liên ngân hàng, sau khi nhận đủ thông tin từ khách hàng, phần mềm sẽ yêu cầu hệ thống liên ngân hàng xử lý yêu cầu. Hệ thống liên ngân hàng chuyển tiếp yêu cầu đến ngân hàng thích hợp. Sau đó, ngân hàng đó xử lý và phản hồi cho hệ thống liên ngân hàng, từ đó thông báo kết quả cho phần mềm.
- Nhân viên ngân hàng có thể tạo tài khoản mới cho khách hàng, đặt lại mật khẩu, khóa/mở khóa tài khoản.

b) Xác định tác nhân và ca sử dụng

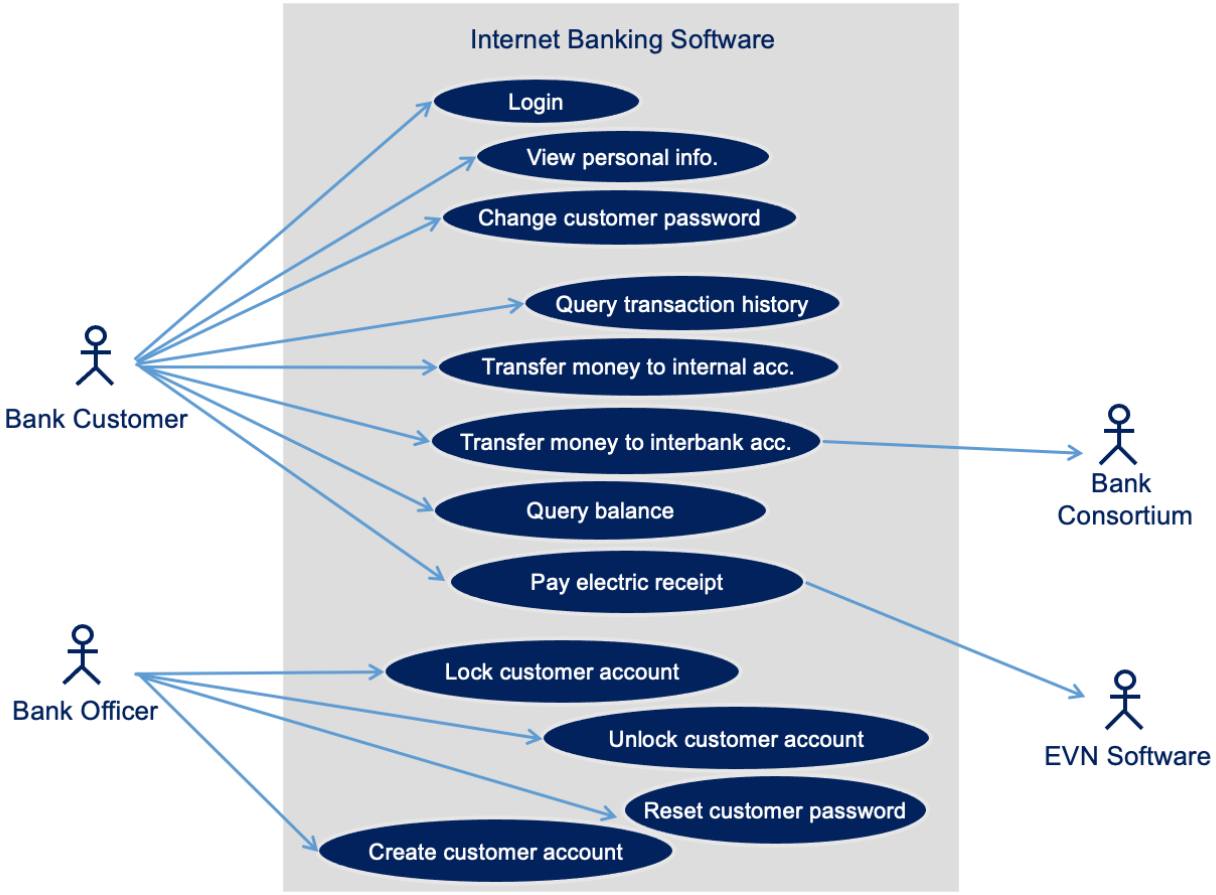
Tác nhân:

- Bank Customer
- Bank Officer
- Bank Consortium
- EVN Software



Hình 6-23 : Tương tác giữa các tác nhân và hệ thống Banking

Xác định các ca sử dụng và vẽ biểu đồ ca sử dụng như Hình 6-24.



Hình 6-24 : Biểu đồ UC hệ thống ngân hàng

6.6.2 Đặc tả chi tiết UC

UC – Interbanking Transfer được đặc tả chi tiết như sau:

1. Khách hàng chọn "Chuyển khoản liên ngân hàng"
2. Phần mềm hiển thị màn hình chuyển khoản liên ngân hàng
3. Khách hàng chọn ngân hàng đích và nhập tài khoản đích
4. Phần mềm yêu cầu hệ thống liên ngân hàng lấy tên tài khoản đích
5. Phần mềm hiển thị tên tài khoản đích
6. Khách hàng nhập số tiền cần chuyển và yêu cầu chuyển.
7. Phần mềm kiểm tra số dư của khách hàng nếu tiền không đủ. Nếu không đủ sẽ thông báo "Không đủ tiền"
8. Ngược lại phần mềm sẽ gửi mã OTP cho khách hàng
9. Khách hàng nhập số OTP và xác nhận chuyển khoản
10. Phần mềm xác minh số OTP có hợp lệ không

11. Nếu hợp lệ, phần mềm sẽ yêu cầu liên ngân hàng xử lý chuyển khoản đến tài khoản đích và giảm số tiền chuyển vào tài khoản khách hàng.
12. Phần mềm thông báo chuyển thành công.

Bảng 6-8 - Đặc tả chi tiết UC Transfer Inter-Bank

<table>
<tr><td>Mã Use case</td><td>UC002</td><td>Tên Use case</td><td>Interbank-Transfer</td></tr>
<tr><td>Tác nhân chính</td><td colspan="3">Bank Customer</td></tr>
<tr><td>Tác nhân phụ</td><td colspan="3">Bank Consortium</td></tr>
<tr><td>Tiền điều kiện</td><td colspan="3">Khách hàng đã đăng nhập thành công vào hệ thống</td></tr>
<tr><td>Luồng sự kiện chính (Thành công)</td><td colspan="3">
<table>
<tr><th>STT</th><th>Thực hiện bởi</th><th>Hành động</th></tr>
<tr><td>1.</td><td>Khách hàng</td><td>chọn hình thức Chuyển khoản liên ngân hàng</td></tr>
<tr><td>2.</td><td>Hệ thống</td><td>hiển thị màn hình chuyển khoản liên ngân hàng</td></tr>
<tr><td>3.</td><td>Khách hàng</td><td>chọn ngân hàng đích và nhập tài khoản đích</td></tr>
<tr><td>4.</td><td>Hệ thống</td><td>gửi yêu cầu tới hệ thống liên ngân hàng để lấy thông tin tài khoản đích</td></tr>
<tr><td>5.</td><td>Hệ thống liên ngân hàng</td><td>xử lý yêu cầu và gửi trả lại thông tin cho hệ thống</td></tr>
<tr><td>6.</td><td>Hệ thống</td><td>hiển thị tên tài khoản đích</td></tr>
<tr><td>7.</td><td>Khách hàng</td><td>nhập số tiền cần chuyển và yêu cầu chuyển khoản</td></tr>
<tr><td>8.</td><td>Hệ thống</td><td>kiểm tra số dư của khách hàng và gửi mã OTP cho khách hàng nếu số dư thoả mãn</td></tr>
<tr><td>9.</td><td>Khách hàng</td><td>nhập số OTP và xác nhận chuyển khoản</td></tr>
<tr><td>10.</td><td>Hệ thống</td><td>Xác minh số OTP hợp lệ và yêu cầu hệ thống liên ngân hàng xử lý chuyển khoản</td></tr>
<tr><td>11.</td><td>Hệ thống</td><td>Thông báo chuyển khoản thành công</td></tr>
</table>
</td></tr>
<tr><td>Luồng sự kiện thay thế</td><td colspan="3">
<table>
<tr><th>STT</th><th>Thực hiện bởi</th><th>Hành động</th></tr>
<tr><td>6a.</td><td>Hệ thống</td><td>Thông báo tài khoản đích không hợp lệ</td></tr>
<tr><td>9a.</td><td>Hệ thống</td><td>Thông báo số dư của khách hàng không đủ</td></tr>
<tr><td>11a.</td><td>Hệ thống</td><td>Thông báo mã OTP không hợp lệ</td></tr>
</table>
</td></tr>
<tr><td>Hậu điều kiện</td><td colspan="3">Tài khoản của khách hàng bị trừ số tiền tương ứng đã chuyển khoản</td></tr>
</table>

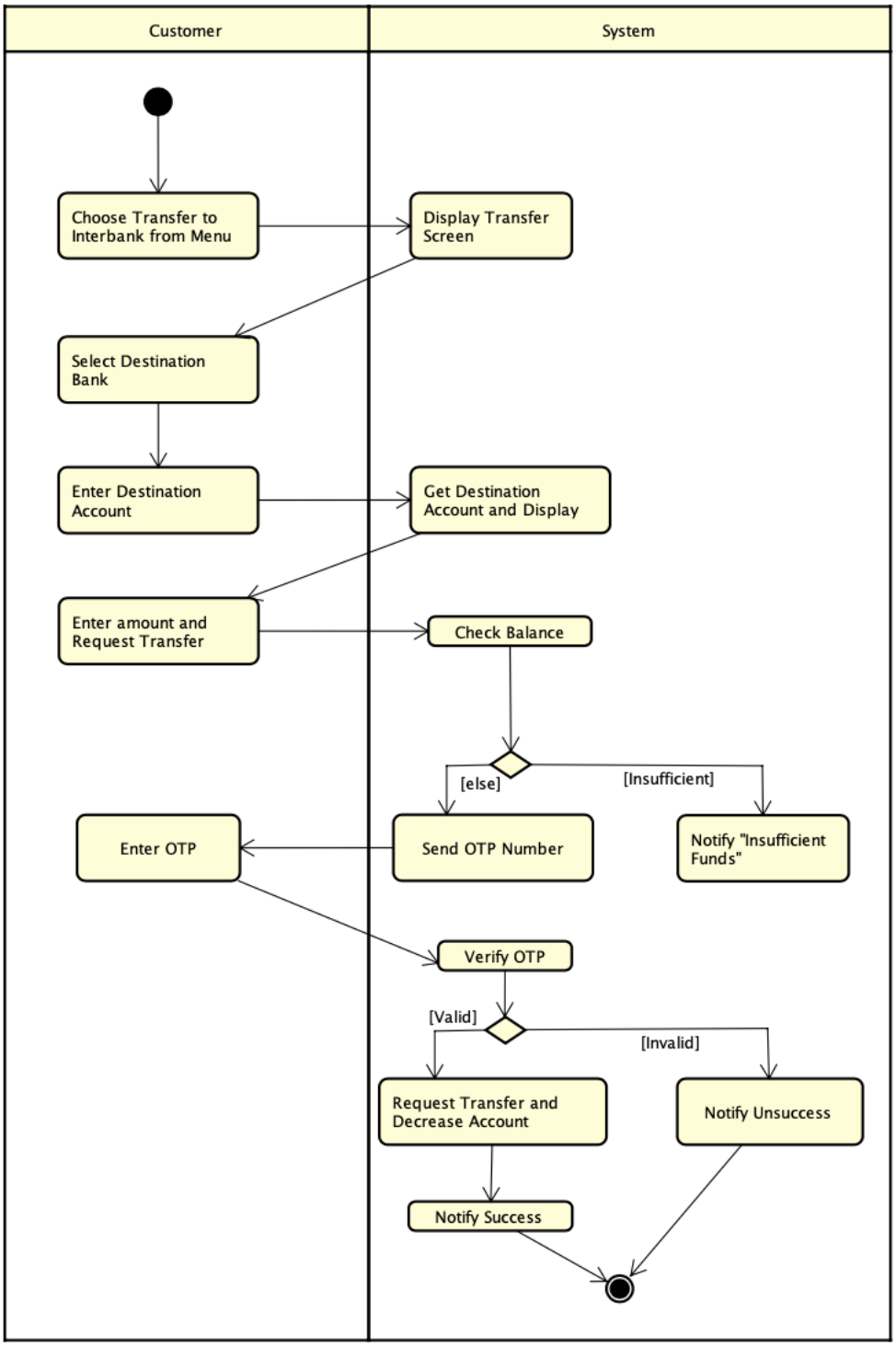
6.6.3 Biểu đồ hoạt động trực quan hoá luồng sự kiện UC

Xác định các thành phần trong biểu đồ hoạt động cho UC Transfer Inter-Bank

- Swimlanes
  - Customer / System

- Activities
  - Xác định các hoạt động ứng với các bước tương ứng trong đặc tả chi tiết UC
  - Bước 1 – bước 11
  - Bước 6a, 9a, 11a
- Decision nodes
  - Xác định các điểm quyết định tương ứng với các bước có điều kiện cần kiểm chứng
  - Bước 5, 8 và 10

Xây dựng biểu đồ hoạt động như trên Hình 6-25.



Hình 6-25 : Biểu đồ hoạt động UC Transfer Inter-Bank

Chương 7 THIẾT KẾ PHẦN MỀM

Nội dung:

- Các khái niệm trong thiết kế phần mềm
- Các tiêu chí chất lượng cho thiết kế phần mềm
- Thiết kế kiến trúc và thiết kế chi tiết
- Thiết kế giao diện người dùng

7.1 Tổng quan về thiết kế phần mềm

7.1.1 Khái niệm về thiết kế phần mềm

a) Thiết kế phần mềm là gì?

Sau khi phân tích yêu cầu, nhóm phát triển chuyển sang pha thiết kế. Đây là giai đoạn rất quan trọng góp phần tạo nên sự thành công của phần mềm. Thiết kế sẽ xác định cách thức thực hiện những gì đã được đặt ra ở phần phân tích. Bản chất thiết kế phần mềm là một quá trình chuyển hóa các yêu cầu phần mềm thành một biểu diễn thiết kế. Từ những mô tả quan niệm về toàn bộ phần mềm, việc làm mịn (chi tiết hóa) liên tục dẫn tới một biểu diễn thiết kế rất gần với cách biểu diễn của chương trình nguồn để có thể ánh xạ vào một ngôn ngữ lập trình cụ thể.

Thiết kế là quá trình tạo ra *các biểu diễn* và *dữ kiện* của hệ thống phần mềm cần xây dựng *từ kết quả phân tích yêu cầu* để có thể tiếp tục thực hiện giai đoạn tiếp theo trong tiến trình phát triển phần mềm.

Kết quả của pha thiết kế là *mô hình thiết kế của phần mềm*

- Mô hình thiết kế đủ chi tiết để giai đoạn lập trình có thể thực hiện
- Là phương tiện để trao đổi thông tin và đảm bảo chất lượng
- Mô hình thiết kế dễ sửa đổi hơn mã chương trình, cung cấp cái nhìn tổng thể, đồng thời có nhiều mức chi tiết

Các mô hình hay biểu diễn của phần mềm sau này sẽ được xây dựng hay thực thi (Implementation) ở giai đoạn tiếp theo. Chúng ta có thể phân tính và đánh giá những mô hình này có hay không phù hợp với những yêu cầu khác nhau. Chúng ta có thể sử dụng các mô hình này để kiểm tra và thẩm định nhằm tìm những giải pháp thay thế và

xem xét những đánh đổi (tradeoffs). Cuối cùng chúng ta sử dụng những mô hình kết quả là phương tiện để trao đổi thông tin và đảm bảo chất lượng. Các mô hình được sử dụng như là đầu vào hay là điểm bắt đầu của quá trình xây dựng và kiểm thử phần mềm.

b) Từ mô hình phân tích đến mô hình thiết kế

Trên thực tế, phân tích và thiết kế không có sự phân tách riêng biệt mà hai hoạt động này được tiến hành phối hợp và bổ sung cho nhau. Bởi vì phân tích là tìm ra những yêu cầu của khách hàng, thiết kế sẽ quyết định cách thức để thực hiện những yêu cầu đó, mô tả một hoặc nhiều giải pháp, giúp đánh giá các giải pháp, lựa chọn giải pháp tốt nhất. Quá trình này chuyển hoá các yêu cầu thành một biểu diễn phần mềm.

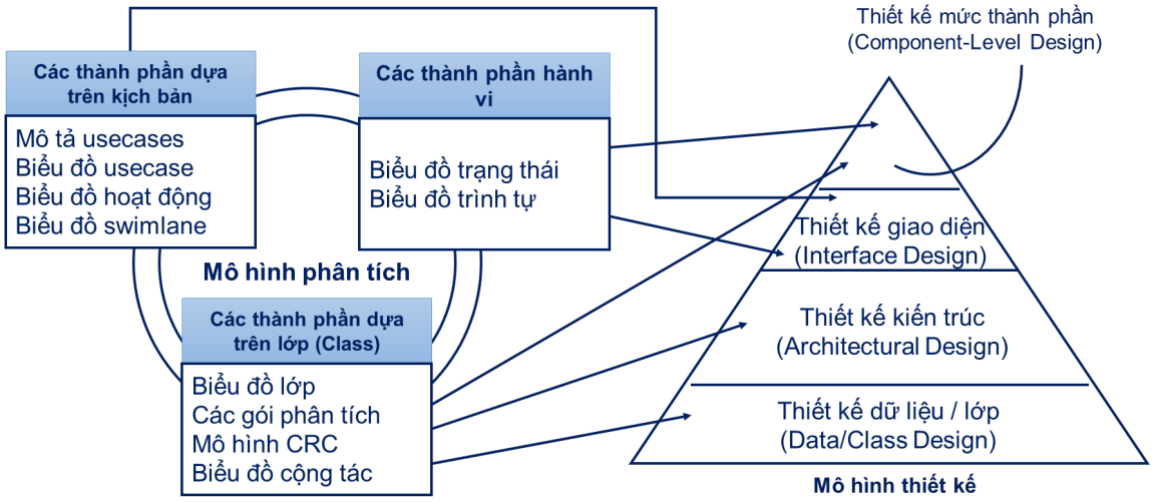
Hãy liên hệ một ví dụ về quá trình xây dựng một ngôi nhà. Căn cứ vào yêu cầu của chủ nhà và các số liệu về căn nhà cần xây dựng, người kỹ sư sẽ thiết kế ra mô hình ngôi nhà. Đây chưa phải là ngôi nhà thật đã được xây dựng trong thực tế, mà chỉ là trên bản vẽ. Nhưng thông qua bản vẽ đó, cùng với sự mô 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 chỉ cần thực hiện 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. Theo lời của Frank Lloyd Wright, "Thay vì búa tạ trên công trường xây dựng, bạn có thể sử dụng một cục tẩy trên bàn soạn thảo."

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 nọ. Bản vẽ thiết kế của ngôi nhà tương ứng chính với các mô hình để biểu diễn cho phần mềm.

Đầu ra của thiết kế phần mềm sẽ được sử dụng cho quá trình xây dựng và kiểm thử nên việc đánh giá một thiết kế có phù hợp hay không rất quan trọng, nếu một thiết kế sai sẽ dẫn đến tất cả các quá trình sau đó cũng sai và cần phải chỉnh sửa nếu thiết kế được chỉnh sửa. Điều này sẽ làm tăng công sức viết mã chương trình, tăng công sức bảo trì và khó khăn khi sửa đổi, mở rộng.

Hoạt động chính của pha thiết kế là tiến hóa tập biểu diễn phân tích thành tập biểu diễn thiết kế. Mỗi thành phần của mô hình yêu cầu cung cấp thông tin để tạo ra bốn mô hình

thiết kế cần thiết cho một đặc tả thiết kế hoàn chỉnh. Luồng thông tin trong quá trình thiết kế phần mềm được minh họa trong hình 7.1.



Hình 7-1: Từ mô hình phân tích đến mô hình thiết kế

Mô hình yêu cầu, được thể hiện bằng các yếu tố dựa trên kịch bản, dựa trên lớp và hành vi, tạo ra thiết kế dữ liệu/lớp, thiết kế kiến trúc, thiết kế giao diện và thiết kế thành phần.

- *Thiết kế dữ liệu/lớp* chuyển đổi các mô hình lớp thành các lớp thiết kế và các cấu trúc dữ liệu cần thiết để triển khai phần mềm.
- *Thiết kế kiến trúc* xác định mối quan hệ giữa các thành phần cấu trúc chính của phần mềm dựa trên các kiểu kiến trúc và mẫu kiến trúc.
- *Thiết kế giao diện* mô tả cách phần mềm giao tiếp với các hệ thống tương tác và với người sử dụng. Giao diện ngụ ý về luồng thông tin (ví dụ: dữ liệu và/hoặc điều khiển) và loại hành vi cụ thể. Vì vậy, các kịch bản sử dụng và mô hình hành vi cung cấp nhiều thông tin cần thiết cho việc thiết kế giao diện.
- *Thiết kế mức thành phần* chuyển đổi các thành phần cấu trúc của kiến trúc phần mềm thành mô tả thủ tục của các thành phần phần mềm. Thông tin thu được từ các mô hình dựa trên lớp và mô hình hành vi làm cơ sở cho việc thiết kế thành phần.

c) Các giai đoạn thiết kế

Trong danh sách tiêu chuẩn các quy trình vòng đời phần mềm như Quy trình vòng đời phần mềm ISO/IEC/IEEE 12207, thiết kế phần mềm bao gồm hai giai đoạn:

- *Thiết kế kiến trúc phần mềm (Software architectural design) (còn được gọi là thiết kế mức cao / top-level design)*: mô tả cấu trúc và tổ chức cấp cao nhất của phần mềm và xác định các thành phần khác nhau. Các hạng mục chính bao gồm: (1) cấu trúc cấp cao của phần mềm và các thành phần xây dựng nên phần mềm, (2) một thiết kế cấp cao cho các giao diện bên ngoài phần mềm và giữa các thành phần phần mềm, (3) một thiết kế cấp cao cho cơ sở dữ liệu.
- *Thiết kế chi tiết phần mềm (Software detailed design)*: mô tả từng thành phần một cách đầy đủ để cho phép xây dựng nó. Trong thiết kế chi tiết, các mục sau được phát triển: (1) mỗi thành phần (component) được tinh chỉnh thành các *đơn vị phần mềm (software units)* có thể được mã hóa, biên dịch và kiểm thử, (2) các giao diện bên ngoài thành phần phần mềm, giữa các thành phần phần mềm và giữa các đơn vị phần mềm.

Có nhiều chiến lược hỗ trợ cho quá trình thiết kế qua các phương pháp tiếp cận khác nhau. Tuy nhiên không có một chiến lược nào là toàn năng cho tất cả các dự án. Hai chiến lược thiết kế được sử dụng phổ biến là *thiết kế hướng chức năng* và *thiết kế hướng đối tượng*. Mỗi chiến lược đều có những ưu, nhược điểm riêng phụ thuộc vào ứng dụng phát triển và nhóm phát triển phần mềm. Hai cách tiếp cận này là bổ sung và hỗ trợ cho nhau chứ không đối kháng nhau. Ngoài ra còn có các phương pháp tiếp cận thiết kế khác: thiết kế hướng thành phần, thiết kế hướng dịch vụ,…

- *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. Đây thường được xem là phương pháp cổ điển. Người ta dùng các biểu đồ luồng dữ liệu mô tả việc xử lý dữ liệu logic, các lược đồ cấu trúc để chỉ ra cấu trúc của phần mềm và mối quan hệ giữa các thành phần.
- *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. Thiết kế hướng đối tượng dựa trên ý tưởng che dấu thông tin và có thể dùng lại một số thành phần của đối tượng đã được thiết kế trước đó. Cách nhìn tự nhiên nhiều hệ thống là cách nhìn chức năng nên việc thích nghi với cách nhìn đối tượng đôi khi là khó khăn. Làm sao để tìm ra các đối tượng thích hợp trong một hệ thống cũng là một vấn đề.

- *Thiết kế hướng thành phần*: một thành phần của phần mềm là một đơn vị độc lập, có giao diện và có thể được triển khai một cách độc lập. Chiến lược thiết kế hướng thành phần dựa trên các vấn đề liên quan đến việc cung cấp, phát triển, và tích hợp các thành phần như vậy để cải thiện, tái sử dụng.
- *Thiết kế hướng dịch vụ*: phần mềm gồm nhiều thành phần độc lập được thể hiện thành những dịch vụ (service), mỗi dịch vụ thực hiện quy trình nghiệp vụ nào đó. Các dịch vụ kết nối "mềm dẻo" với nhau, có giao tiếp được định nghĩa rõ ràng, có tính kế thừa các thành phần đang tồn tại, và sự tương tác giữa chúng không cần quan tâm đến việc chúng được phát triển trên nền tảng công nghệ nào. Điều này khiến hệ thống có thể mở rộng và tích hợp một cách dễ dàng.

7.1.2 Chất lượng của thiết kế

a) Đặc trưng của một thiết kế tốt

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. 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*.

Để đánh giá các đặc trưng của một thiết kế tốt, người ta tiến hành thiết lập một số độ đo chất lượng thiết kế. Theo Mitch Kapor, là tác giả của Tuyên ngôn thiết kế phần mềm thì Thiết kế phần mềm tốt (good design) nên thể hiện ở 3 khía cạnh: *Sự ổn định (Firmness), Tiện nghi (Commodity), và Sự hài lòng (Delight)*.

- *Sự ổn định (Firmness)*: Một chương trình không nên có bất cứ lỗi nào làm hạn chế chức năng của nó
- *Sự tiện nghi (Commodity)*: Một chương trình nên phù hợp với mục đích đã định của nó
- *Sự hài lòng (Delight)*: Trải nghiệm sử dụng chương trình nên làm hài lòng người dùng

Và đối ngược với các đặc trưng này là ba đặc điểm quan trọng của một *thiết kế xấu nên tránh*:

- *Cứng nhắc (Rigidity)* – Khó có thể thay đổi vì mỗi thay đổi ảnh hưởng quá nhiều đến các phần khác của hệ thống.
- *Bất ổn định (Fragility)* – Khi thực hiện một sự thay đổi, phần thay đổi bất ngờ đó sẽ có thể phá vỡ hệ thống.
- *Kém linh hoạt (Immobility)* – Khó có thể tái sử dụng lại trong các ứng dụng khác bởi nó không thể tách rời khỏi các ứng dụng hiện hành.

Trong suốt quá trình thiết kế, chất lượng của thiết kế đang phát triển được đánh giá bằng một loạt các đánh giá kỹ thuật. Ba đặc điểm dùng làm hướng dẫn đánh giá một thiết kế tốt:

- *Thiết kế phải thực hiện tất cả các yêu cầu rõ ràng* chứa trong mô hình phân tích, và nó phải đáp ứng tất cả các yêu cầu tiềm ẩn khách hàng mong muốn.
- *Thiết kế phải là một hướng dẫn dễ hiểu, dễ đọc* cho những người tạo ra code và cho những người kiểm thử và sau đó hỗ trợ cho phần mềm.
- *Thiết kế nên cung cấp một bức tranh hoàn chỉnh của phần mềm,* giải quyết vấn đề dữ liệu, chức năng, và hành vi từ một quan điểm thực thi.

Mỗi đặc điểm này thực sự là một mục tiêu quan trọng của quá trình thiết kế. Nhưng làm thế nào để đạt được các mục tiêu này? Chúng ta cần thiết lập các tiêu chí chất lượng để hướng dẫn cho quá trình thực hiện thiết kế.

b) Các tiêu chí cho thiết kế tốt

- *Một thiết kế nên thể hiện một kiến trúc* mà:
  - (1) đã được tạo ra bằng cách sử dụng phong cách kiến trúc hoặc các pattern được công nhận,
  - (2) bao gồm các thành phần mang những đặc tính thiết kế tốt và,
  - (3) có thể được thực hiện một cách tiến hóa,
  - Đối với các hệ thống nhỏ hơn, thiết kế đôi khi có thể được phát triển tuyến tính.
- *Một thiết kế nên mô-đun hoá*, đó là phần mềm nên được phân chia thành các thành phần hợp lý hoặc hệ thống con

- *Một thiết kế cần có biểu diễn riêng biệt* của dữ liệu, kiến trúc, giao diện, và các thành phần
- *Một thiết kế nên dẫn đến các cấu trúc dữ liệu thích hợp* cho các lớp sẽ được thực thi và được rút ra từ mô hình dữ liệu có thể nhận biết
- Một thiết kế nên dẫn đến *các thành phần mang những đặc tính chức năng độc lập*
- Một thiết kế nên dẫn đến *giao diện mà giảm sự phức tạp* của các kết nối giữa các thành phần và với môi trường bên ngoài
- Một thiết kế nên *được chuyển hoá bằng cách sử dụng một phương pháp lặp lại* được dẫn dắt bởi các thông tin thu được trong quá trình phân tích các yêu cầu phần mềm
- Một thiết kế nên được *đại diện bằng một ký hiệu truyền đạt hiệu quả ý nghĩa* của nó

Những hướng dẫn thiết kế này không xuất hiện một cách tình cờ. Chúng đạt được thông qua việc áp dụng các nguyên tắc thiết kế cơ bản, các phương pháp có hệ thống và được xem xét kỹ lưỡng.

7.1.3 Nguyên tắc thiết kế phần mềm

- Quá trình thiết kế *không nên* mắc phải "tunnel vision" / “tầm nhìn đường hầm”. Một nhà thiết kế giỏi nên xem xét các phương pháp tiếp cận khác nhau, đánh giá từng phương pháp dựa trên yêu cầu của vấn đề, nguồn lực sẵn có để thực hiện công việc.
- Việc thiết kế nên *có thể truy ngược* về mô hình phân tích. Bởi vì một yếu tố duy nhất của mô hình thiết kế thường liên quan đến nhiều yêu cầu, nên cần có một phương tiện để theo dõi xem mô hình thiết kế đã đáp ứng các yêu cầu như thế nào.
- Việc thiết kế *không nên* "phát minh lại bánh xe". Các hệ thống được xây dựng bằng cách sử dụng một tập hợp các mẫu thiết kế, nhiều mẫu trong số đó có thể đã từng gặp trước đây. Những mẫu này phải luôn được chọn làm giải pháp thay

thế cho việc sáng tạo lại. Thời gian rất ngắn và nguồn lực có hạn! Thời gian thiết kế nên được đầu tư vào việc thể hiện những ý tưởng thực sự mới và tích hợp những mẫu đã tồn tại.

- Việc thiết kế nên "*giảm thiểu khoảng cách trí tuệ*" giữa phần mềm và bài toán như nó tồn tại trong thế giới thực. Nghĩa là, cấu trúc của thiết kế phần mềm nên (bất cứ khi nào có thể) bắt chước cấu trúc của miền vấn đề.
- Việc thiết kế nên biểu lộ *tính đồng nhất và tích hợp*. Một thiết kế được coi là đồng nhất nếu có vẻ như chỉ có một người phát triển toàn bộ. Các quy tắc về phong cách và định dạng phải được xác định cho nhóm thiết kế trước khi bắt đầu công việc thiết kế. Một thiết kế được tích hợp nếu sự cẩn thận được thực hiện trong việc xác định các giao diện giữa các thành phần thiết kế.
- Việc thiết kế nên được cấu trúc để *thích ứng với thay đổi*. Một thiết kế có thể thích nghi với việc cải biên các chức năng và việc thêm các chức năng mới. Một thiết kế như thế phải dễ hiểu và việc sửa đổi chỉ có hiệu ứng cục bộ. Các thành phần thiết kế phải là kết dính (cohesive) theo nghĩa là tất cả các bộ phận trong thành phần phải có một quan hệ logic chặt chẽ, các thành phần ghép nối (coupling) với nhau là lỏng lẻo. Ghép nối càng lỏng lẻo thì càng dễ thích nghi, nghĩa là càng dễ sửa đổi để phù hợp với hoàn cảnh mới.
- Việc thiết kế nên được cấu trúc để *làm suy thoái (degrade) nhẹ nhàng*, ngay cả khi đang gặp phải dữ liệu, các sự kiện, hoặc điều kiện hoạt động bất thường. Phần mềm được thiết kế tốt sẽ không bao giờ bị "đánh bom". Nó phải được thiết kế để phù hợp với những trường hợp bất thường và nếu nó phải chấm dứt quá trình xử lý, hãy thực hiện điều đó một cách nhẹ nhàng.
- Thiết kế không phải là lập trình, lập trình không phải là thiết kế. Ngay cả khi các thiết kế thủ tục chi tiết được tạo ra cho các thành phần chương trình, *mức độ trừu tượng của mô hình thiết kế vẫn cao hơn mã nguồn*. Các quyết định thiết kế duy nhất được đưa ra ở cấp độ mã hóa nhằm giải quyết các chi tiết triển khai nhỏ cho phép thiết kế thủ tục được mã hóa.

- Việc thiết kế nên được *đánh giá về chất lượng khi nó được tạo ra, chứ không phải sau thực tế*. Có nhiều khái niệm thiết kế và biện pháp thiết kế khác nhau để hỗ trợ người thiết kế đánh giá chất lượng.
- Việc thiết kế cần được xem xét để *giảm thiểu lỗi khái niệm* (ngữ nghĩa). Đôi khi nhà phát triển có xu hướng tập trung vào những chi tiết vụn vặt khi xem xét thiết kế, bỏ sót rừng cây. Nhóm thiết kế phải đảm bảo rằng các yếu tố khái niệm chính của thiết kế (thiếu sót, mơ hồ, không nhất quán) đã được giải quyết trước khi lo lắng về cú pháp của mô hình thiết kế.

7.2 Các khái niệm trong thiết kế phần mềm

Một số phương pháp thiết kế đã được nghiên cứu và được áp dụng trong thực tiễn. Mỗi phương pháp đều cung cấp cho người thiết kế phần mềm một nền tảng để từ đó có thể áp dụng hoặc xây dựng phương pháp thiết kế phức tạp hơn. Nền tảng này giúp thực hiện các nhiệm vụ thiết kế phần mềm, ví dụ: phân vùng phần mềm thành các thành phần riêng lẻ, tách biệt hoặc xây dựng chi tiết cấu trúc dữ liệu, v.v… Qua đó thiết lập các tiêu chí thống nhất xác định chất lượng của thiết kế. Các phương pháp này đều có một số đặc điểm chung:

(1) cơ chế chuyển đổi mô hình yêu cầu thành bản trình bày thiết kế,

(2) ký hiệu để biểu diễn các thành phần chức năng và giao diện của chúng,

(3) phương pháp heuristics để tinh chế và phân vùng, và

(4) hướng dẫn đánh giá chất lượng.

Bất kể phương pháp thiết kế nào được sử dụng, chúng đều được xây dựng dựa trên một tập các khái niệm cơ bản cho thiết kế ở các cấp độ khác nhau: dữ liệu, kiến trúc, giao diện và thành phần. Mặc dù mức độ quan tâm đến những khái niệm này đã thay đổi qua nhiều năm, nhưng mỗi khái niệm đều đã đứng vững trước thử thách của thời gian. Những khái niệm này sẽ được trình bày trong nội dung tiếp theo.

7.2.1 Các khái niệm cơ bản trong thiết kế phần mềm

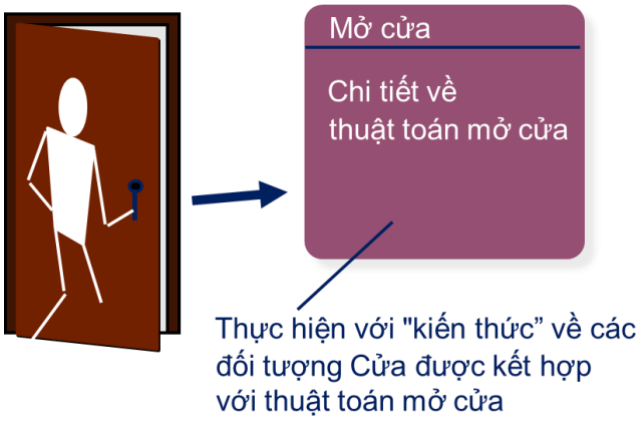
a) Trừu tượng hóa (Abstraction)

Trừu tượng hóa là quá trình ánh xạ một sự vật/hiện tượng của thế giới thực thành một khái niệm logic. Quá trình thiết kế trải qua nhiều mức trừu tượng hoá khác nhau.

- Mức trừu tượng cao nhất là các vấn đề được mô tả một cách tổng quát và có sử dụng thuật ngữ trong miền lĩnh vực của bài toán.
- Mức độ trừu tượng thấp hơn thì mô tả chi tiết hơn về giải pháp sẽ được cung cấp, hướng đến thủ tục xử lý chi tiết; kết hợp các thuật ngữ hướng đến hiện thực.
- Mức độ trừu tượng thấp nhất, giải pháp được nêu theo cách có thể triển khai trực tiếp.

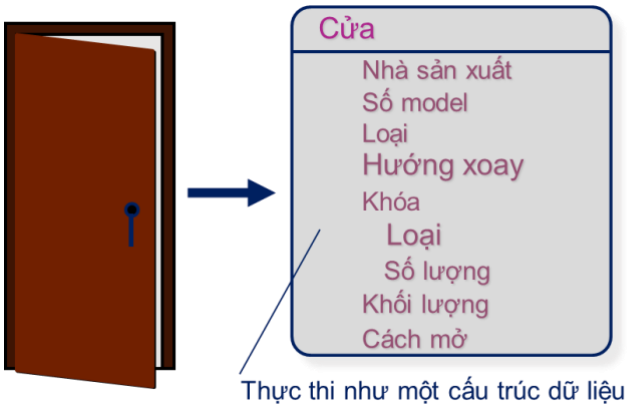
Bao gồm *trừu tượng hóa về thủ tục và dữ liệu*:

Trừu tượng hoá thủ tục là quá trình chia nhỏ một vấn đề phức tạp thành các bước nhỏ hơn, dễ quản lý hơn. Bằng cách xác định các hành động thiết yếu cần được thực hiện để giải quyết vấn đề, việc trừu tượng hóa thủ tục có thể giúp làm cho quá trình giải quyết vấn đề trở nên hiệu quả hơn, cho phép xây dựng các ứng dụng phức tạp từ những thao tác đơn giản. Ví dụ: Thủ tục mở cửa (bao gồm đi đến cửa, cầm lấy tay nắm, xoay tay nắm, kéo cánh cửa, đi vào…);



Hình 7-2: Trừu tượng hóa thủ tục mở cửa

Trừu tượng hoá dữ liệu là tổ hợp dữ liệu mô tả một đối tượng dữ liệu. Quá trình này cho phép rút gọn một phần dữ liệu cụ thể thành một đại diện đơn giản cho toàn bộ. Nói chung, trừu tượng dữ liệu là quá trình loại bỏ các đặc điểm chi tiết để tạo thành một tập hợp các phần tử thiết yếu, giúp đơn giản hóa dữ liệu cơ bản, đồng thời ẩn đi sự phức tạp liên quan của nó. (Ví dụ tập hợp các thuộc tính mô tả đối tượng cánh cửa)



Hình 7-3: Trừu tượng hóa dữ liệu của cánh cửa

b) Kiến trúc (Architecture)

Ở dạng đơn giản nhất, kiến trúc là cấu trúc hoặc tổ chức của các thành phần chương trình (mô-đun), cách thức mà các thành phần này tương tác và cấu trúc dữ liệu được các thành phần đó sử dụng. Tuy nhiên, theo nghĩa rộng hơn, các thành phần có thể được khái quát hóa để biểu diễn các thành phần chính của hệ thống và sự tương tác của chúng.

Một mục tiêu của thiết kế phần mềm là tạo ra bản vẽ kiến trúc của một hệ thống. Kết xuất này đóng vai trò như một khuôn khổ để từ đó các hoạt động thiết kế chi tiết hơn được tiến hành. Một tập hợp các mẫu kiến trúc cho phép kỹ sư phần mềm sử dụng lại các khái niệm ở mức thiết kế.

c) Mẫu thiết kế (Design patterns)

Mỗi chúng ta đều đã trải qua vấn đề thiết kế và từng nghĩ: "Liệu đã có ai phát triển một lời giải cho vấn đề này chưa?" Điều gì sẽ xảy ra nếu đã có một cách tiêu chuẩn để mô tả một vấn đề, và một phương pháp có tổ chức để trình bày lời giải cho vấn đề đó?

Mẫu thiết kế là một phương pháp có hệ thống để mô tả các vấn đề và giải pháp, cho phép các cộng đồng công nghệ phần mềm có thể nắm bắt kiến thức thiết kế theo cách có thể tái sử dụng.

Nói theo cách khác, mẫu thiết kế mô tả một cấu trúc thiết kế giải quyết một vấn đề thiết kế cụ thể trong một bối cảnh cụ thể và trong bối cảnh "các lực" có thể có tác động đến cách áp dụng và sử dụng mẫu đó. "Chuyển tải các tinh túy" của một giải pháp thiết kế đã được chứng minh.

Một số mô tả khác về khái niệm mẫu thiết kế:

- "Mỗi khuôn mẫu mô tả một vấn đề xảy ra hết lần này đến lần khác trong môi trường của chúng ta và sau đó mô tả cốt lõi của giải pháp cho vấn đề đó theo một cách mà bạn có thể sử dụng nó hàng triệu lần mà không bao giờ phải làm lại một việc lần thứ hai."
- "Một quy tắc ba phần diễn tả một mối quan hệ giữa một ngữ cảnh, một vấn đề, và một giải pháp."

Mục đích của mỗi mẫu thiết kế là cung cấp một mô tả cho phép người thiết kế xác định (1) liệu mẫu đó có thể áp dụng cho công việc hiện tại hay không, (2) liệu mẫu đó có thể được sử dụng lại hay không (do đó, tiết kiệm thời gian thiết kế) và (3) liệu mẫu có thể dùng làm hướng dẫn để phát triển một mẫu tương tự nhưng khác về chức năng hoặc cấu trúc hay không.

Các loại mẫu thiết kế:

- *Architectural patterns*, mẫu kiến trúc mô tả các vấn đề thiết kế trên diện rộng được giải quyết bằng cách tiếp cận cấu trúc
- *Data patterns*, mẫu dữ liệu mô tả vấn đề dữ liệu và các giải pháp mô hình dữ liệu có thể được dùng để giải quyết vấn đề trên
- *Component patterns*, mẫu thành phần nhằm đến các vấn đề liên quan với việc phát triển các hệ thống con và các thành phần, cách thức chúng giao tiếp với nhau, và vị trí của chúng trong một kiến trúc lớn hơn
- *Interface design patterns*, mẫu thiết kế giao diện mô tả các vấn đề giao diện người dùng thông thường và giải pháp bao gồm các đặc trưng cụ thể của người dùng cuối

d) Phân tách mối quan tâm (Separation of concerns)

Tách biệt các mối quan tâm là một khái niệm thiết kế gợi ý rằng bất kỳ vấn đề phức tạp nào cũng có thể được xử lý dễ dàng hơn nếu nó được chia thành các phần mà mỗi phần có thể được giải quyết và/hoặc tối ưu hóa một cách độc lập. *Mối quan tâm là một tính năng hoặc hành vi được chỉ định như một phần của mô hình yêu cầu đối với phần mềm.*

Bằng cách tách các mối quan tâm thành các phần nhỏ hơn và do đó dễ quản lý hơn, một vấn đề sẽ tốn ít công sức và thời gian hơn để giải quyết. Việc phân tách chương trình thành nhiều phần, mỗi phần tập trung xử lý một vấn đề giúp ta dễ dàng quản lý code, dễ dàng mở rộng và thay đổi chương trình. Việc đọc code, tìm hiểu luồng chạy cũng sẽ đơn giản hơn, vì mỗi mô-đun, mỗi phần nhỏ chỉ tập trung giải quyết một vấn đề riêng biệt. Ngoài ra, Separation of Concern còn là tiền đề cho rất nhiều khái niệm, nguyên lý thiết kế khác.

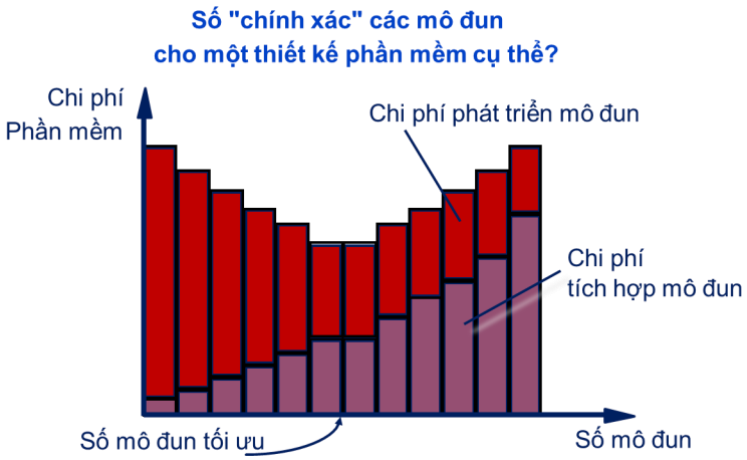
e) Mô đun hoá (Modularity)

Tính mô đun hay mô đun hóa là biểu hiện phổ biến nhất của việc phân tách các mối quan tâm. Phần mềm được chia thành các thành phần rời rạc và độc lập nhau, còn được gọi là mô-đun, được tích hợp để đáp ứng các yêu cầu của vấn đề.

Các mô đun sẽ có khả năng thực hiện công việc một cách độc lập. Những mô đun mới này có thể làm việc như những cấu trúc cơ bản cho toàn bộ phần mềm. Người thiết kế muốn thiết kế các mô đun như thế để chúng có thể thực thi, biên dịch một cách riêng biệt và độc lập được.

Phần mềm nguyên khối (ví dụ, một chương trình lớn gồm một mô-đun duy nhất) có thể không được dễ dàng nắm bắt bởi số lượng các luồng điều khiển, số lượng các biến, và độ phức tạp tổng thể sẽ làm cho việc hiểu được gần như không thể. Trong hầu hết các trường hợp, ta nên phá vỡ thiết kế thành nhiều mô-đun, sẽ làm cho việc hiểu biết dễ dàng hơn và như một hệ quả, giảm chi phí cần thiết để xây dựng các phần mềm. Ngoài ra, hệ thống được chia thành các thành phần nhỏ hơn, điều này sẽ gia tăng tính cục bộ giữa các mô đun, giúp cho việc sửa đổi, bảo trì được dễ dàng.

Phần mềm được xây dựng bằng cách phân chia thành nhiều mô đun để cho việc quản lý phần mềm khoa học hơn. Về sau các mô đun này phải được tích hợp lại với nhau. Khi đó, nhà phát triển cần chi phí tích hợp và giải quyết những khó khăn trong quá trình tích hợp mô đun. Vì vậy cần xác định số mô đun tối ưu trong quá trình phân chia (số lượng mô đun không quá nhiều, không quá ít). Hình 7.4 cho thấy những thông tin cần xem xét và cân nhắc trong việc phân chia mô đun.



Hình 7-4: Sự đánh đổi giữa số lượng mô đun và chi phí phần mềm

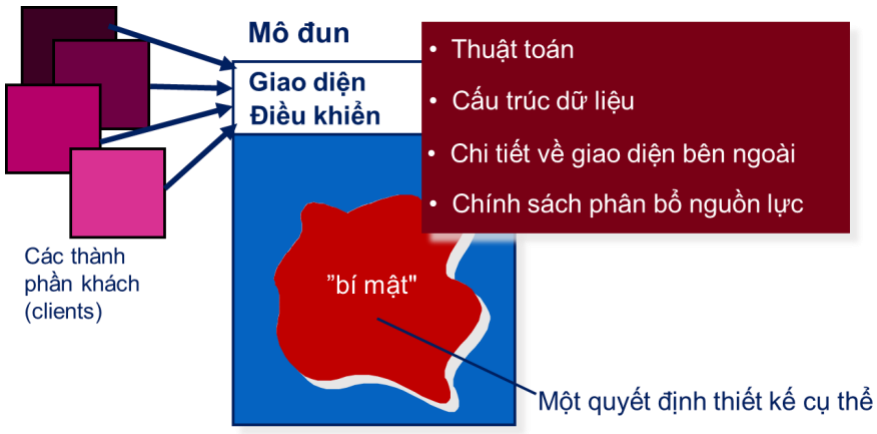
e) Che giấu thông tin (Information hiding)

Che dấu thông tin là một trong những nguyên lý quan trọng của việc phân chia mô đun. Các mô đun giao tiếp với nhau bằng những thông tin thật sự cần thiết. Những thông tin về thủ tục và dữ liệu cục bộ của mỗi mô đun phải được che dấu khỏi các mô đun khác. Một trong những lợi ích của việc che giấu thông tin là kiểm soát được thay đổi và sửa lỗi dễ dàng.

Việc che giấu thông tin có thể mang lại nhiều lợi ích như:

- Giảm hiệu ứng phụ khi sửa đổi thông tin
- Giảm tác động của thiết kế tổng thể lên thiết kế cục bộ
- Nhấn mạnh trao đổi thông tin thông qua giao diện
- Loại bỏ việc sử dụng dữ liệu dùng chung
- Hướng tới sự đóng gói chức năng – một thuộc tính của thiết kế tốt.

Khi xây dựng một hệ thống phần mềm, người sử dụng không quan tâm đến cấu trúc bên trong của hệ thống là đơn giản hay phức tạp, không cần biết cách thức cài đặt của nó thông qua thuật toán gì, cấu trúc dữ liệu như thế nào, có tài nguyên hệ thống ra sao. Họ chỉ quan tâm đến giao diện có những tham số và giá trị nhập vào, trả lại như thế nào. Sử dụng mô đun thông qua các giao diện là cách thức che giấu thông tin.



Hình 7-5: Che giấu thông tin trong thiết kế mô đun

f) Tái cấu trúc (Refactoring)

Tái cấu trúc là một hoạt động thiết kế quan trọng được đề xuất cho nhiều phương pháp phát triển phần mềm linh hoạt (Agile). Tái cấu trúc là quá trình thay đổi một hệ thống phần mềm theo cách mà nó không làm thay đổi hành vi bên ngoài của mã [thiết kế] nhưng cải thiện cấu trúc bên trong của nó.

Khi phần mềm được tái cấu trúc, thiết kế hiện tại sẽ được kiểm tra xem có dư thừa, các yếu tố thiết kế không được sử dụng, thuật toán không hiệu quả hoặc không cần thiết, cấu trúc dữ liệu được xây dựng kém hoặc không phù hợp hoặc bất kỳ lỗi thiết kế nào khác có thể được sửa chữa để mang lại thiết kế tốt hơn.

g) Độc lập chức năng (Functional independence)

Khái niệm về tính độc lập về chức năng là kết quả trực tiếp của việc phân tách các mối quan tâm, tính mô đun và các khái niệm về trừu tượng hóa và che giấu thông tin.

Sự độc lập về chức năng đạt được bằng cách phát triển các mô-đun có chức năng "duy nhất" và "không thích" tương tác quá mức với các mô-đun khác. Nói cách khác, chúng ta nên thiết kế phần mềm sao cho mỗi mô-đun giải quyết một tập hợp con các yêu cầu cụ thể và có giao diện đơn giản khi xem từ các phần khác của cấu trúc chương trình.

Tính độc lập được đánh giá bằng hai tiêu chí: *sự gắn kết hay kết dính (cohesion) và sự ghép nối (coupling).*

7.2.2 Các khái niệm trong thiết kế hướng đối tượng

Mô hình phân tích xác định một tập hợp các lớp phân tích. Mỗi lớp này mô tả một số thành phần của bài toán, tập trung vào các khía cạnh của vấn đề mà người dùng có thể nhìn thấy được. Mức độ trừu tượng của lớp phân tích tương đối cao.

Khi mô hình thiết kế phát triển, chúng ta sẽ xác định một tập hợp các lớp thiết kế để tinh chỉnh các lớp phân tích bằng cách cung cấp chi tiết thiết kế cho phép các lớp được triển khai và cơ sở hạ tầng phần mềm hỗ trợ giải pháp nghiệp vụ.

Các lớp thiết kế được xác định từ 3 loại lớp phân tích là: *Các lớp biên, Các lớp thực thể và Các lớp điều khiển*. Ba loại lớp này được hình thành từ những khía cạnh có thể thay đổi của hệ thống bao gồm: *Ranh giới giữa hệ thống và các tác nhân, Thông tin hệ thống sử dụng và Logic điều khiển của hệ thống*.

Sự hình thành của các lớp thiết kế từ 3 loại lớp này như sau:

- Các lớp Persistent classes được tinh chỉnh trong quá trình thiết kế từ các lớp thực thể
- Các lớp biên phát triển trong thiết kế để tạo ra giao diện interface classes (ví dụ, màn hình tương tác hoặc báo cáo) mà người dùng thấy và tương tác với với phần mềm. Các lớp biên được thiết kế với trách nhiệm quản lý các đối tượng thực thể được đại diện cho người sử dụng.
- Các lớp điều khiển được thiết kế để quản lý:
    - Việc tạo ra hoặc cập nhật các đối tượng thực thể;
    - Sự tức thời của các đối tượng lớp biên khi chúng có được thông tin từ các đối tượng thực thể;
    - Truyền thông phức tạp giữa các tập các đối tượng;
    - Xác nhận của dữ liệu trao đổi giữa các đối tượng hoặc giữa người sử dụng và với ứng dụng.

Ngoài ra các nguyên lý khác của hướng đối tượng cũng được áp dụng trong thiết kế để trình bày chi tiết về khía cạnh kỹ thuật như một hướng dẫn cho việc lập trình.

- *Sự thừa kế* - tất cả các trách nhiệm của một lớp cha được thừa kế bởi các lớp con

- *Thông điệp* - khuyến khích một số hành vi xảy ra trong đối tượng nhận
- *Đa hình* - giúp giảm đáng kể nỗ lực cần thiết để mở rộng thiết kế

7.2.3 Các mô hình thiết kế

a) Các yếu tố thiết kế dữ liệu (Data design elements)

Giống như các hoạt động công nghệ phần mềm khác, thiết kế dữ liệu (đôi khi được gọi là kiến trúc dữ liệu) tạo ra mô hình dữ liệu và/hoặc thông tin được thể hiện ở mức độ trừu tượng cao (quan điểm của khách hàng/người dùng về dữ liệu). Sau đó, mô hình dữ liệu này được tinh chỉnh thành các biểu diễn cụ thể hơn về cách triển khai mà hệ thống dựa trên máy tính có thể xử lý được. Trong nhiều ứng dụng phần mềm, kiến trúc của dữ liệu sẽ có ảnh hưởng sâu sắc đến kiến trúc của phần mềm phải xử lý dữ liệu đó.

b) Các yếu tố thiết kế kiến trúc (Architectural design elements)

Mô hình kiến trúc được lấy từ ba nguồn: (1) thông tin về miền ứng dụng cho phần mềm sẽ được xây dựng; (2) các thành phần của mô hình yêu cầu cụ thể như trường hợp sử dụng hoặc lớp phân tích, mối quan hệ và sự cộng tác của chúng đối với vấn đề hiện tại; và (3) sự sẵn có của các phong cách kiến trúc.

Yếu tố thiết kế kiến trúc thường được mô tả như một tập hợp các hệ thống con được kết nối với nhau, thường bắt nguồn từ các gói phân tích bên trong mô hình yêu cầu. Mỗi hệ thống con có thể có kiến trúc riêng (ví dụ: giao diện đồ họa người dùng có thể được cấu trúc theo kiểu kiến trúc có sẵn cho giao diện người dùng).

c) Các yếu tố thiết kế giao diện (Interface design elements)

Có ba yếu tố quan trọng của thiết kế giao diện: (1) giao diện người dùng (UI), (2) giao diện bên ngoài với các hệ thống, thiết bị, mạng khác hoặc nhà sản xuất hoặc người tiêu dùng thông tin khác và (3) giao diện nội bộ giữa các thành phần thiết kế khác nhau. Các yếu tố thiết kế giao diện này cho phép phần mềm giao tiếp với bên ngoài và cho phép giao tiếp và cộng tác nội bộ giữa các thành phần tạo nên kiến trúc phần mềm.

Thiết kế giao diện người dùng (ngày càng được gọi là thiết kế khả năng sử dụng) là một hoạt động kỹ thuật phần mềm quan trọng. Thiết kế khả năng sử dụng kết hợp các yếu tố thẩm mỹ (ví dụ: bố cục, màu sắc, đồ họa, cơ chế tương tác), các yếu tố công thái

học (ví dụ: bố cục và vị trí thông tin, ẩn dụ, điều hướng giao diện người dùng) và các yếu tố kỹ thuật (ví dụ: mẫu giao diện người dùng, các thành phần có thể tái sử dụng). Việc thiết kế các giao diện bên ngoài yêu cầu thông tin chính xác về thực thể mà thông tin được gửi hoặc nhận. Thiết kế các giao diện bên ngoài cần kết hợp việc kiểm tra lỗi và các tính năng bảo mật thích hợp.

Thiết kế giao diện bên trong được liên kết chặt chẽ với thiết kế cấp thành phần. Việc thực hiện thiết kế của các lớp phân tích thể hiện tất cả các hoạt động và sơ đồ thông điệp cần thiết để cho phép giao tiếp và cộng tác giữa các hoạt động trong các lớp khác nhau. Mỗi thông báo phải được thiết kế để đáp ứng việc truyền thông tin cần thiết và các yêu cầu chức năng cụ thể của hoạt động đã được yêu cầu.

d) Các yếu tố thiết kế mức thành phần (Component-level design elements)

Thiết kế cấp thành phần cho phần mềm mô tả đầy đủ chi tiết bên trong của từng thành phần phần mềm. Để thực hiện điều này, thiết kế cấp thành phần xác định cấu trúc dữ liệu cho tất cả các đối tượng dữ liệu cục bộ và chi tiết thuật toán cho tất cả quá trình xử lý xảy ra trong một thành phần và giao diện cho phép truy cập vào tất cả các hoạt động (hành vi) của thành phần.

Các chi tiết thiết kế của một thành phần có thể được mô hình hóa ở nhiều mức độ trừu tượng khác nhau. Sơ đồ hoạt động UML có thể được sử dụng để thể hiện logic xử lý. Luồng thủ tục chi tiết cho một thành phần có thể được biểu diễn bằng mã giả (biểu diễn giống như ngôn ngữ lập trình) hoặc một số dạng sơ đồ khác (ví dụ: flowchart or box diagram). Cấu trúc thuật toán tuân theo các quy tắc được thiết lập cho lập trình có cấu trúc (tức là một tập hợp các cấu trúc thủ tục bị ràng buộc). Cấu trúc dữ liệu, được chọn dựa trên tính chất của đối tượng dữ liệu cần xử lý, thường được mô hình hóa bằng mã giả hoặc ngôn ngữ lập trình được sử dụng để triển khai.

c) Các yếu tố thiết kế mức triển khai (Deployment-level design elements)

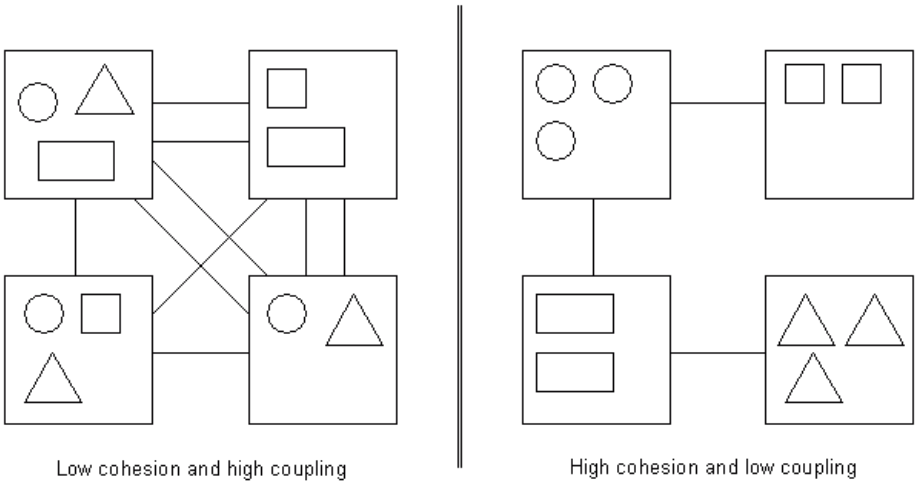
Các phần tử thiết kế ở mức triển khai chỉ ra cách phân bổ chức năng phần mềm và các hệ thống con trong môi trường máy tính vật lý sẽ hỗ trợ phần mềm. Trong quá trình thiết kế, sơ đồ triển khai UML được phát triển và sau đó được tinh chỉnh để thể hiện mô hình triển khai.

7.3 Tính móc nối (Coupling) và tính kết dính (Cohesion)

Sự kết dính của một mô-đun (Cohesion) là độ đo về tính khớp lại với nhau của các phần trong mô-đun đó. Nếu một mô-đun chỉ thực hiện một chức năng logic hoặc là một thực thể logic, tức là tất cả các bộ phận của mô-đun đó đều tham gia vào việc thực hiện một công việc thì độ kết dính là cao. Nếu một hoặc nhiều bộ phận không tham gia trực tiếp vào việc chức năng logic đó thì mức độ kết dính của nó là thấp. Thiết kế là tốt khi độ kết dính cao. Khi đó chúng ta sẽ dễ dàng hiểu được từng mô-đun và việc sửa chữa một mô-đun sẽ không (ít) ảnh hưởng tới các mô-đun khác.

Ghép nối (Coupling) là độ đo sự nối ghép với nhau giữa các đơn vị (mô-đun) của hệ thống. Hệ thống có nối ghép cao thì các mô-đun phụ thuộc lẫn nhau lớn. Hệ thống nối ghép lỏng lẻo thì các mô-đun là độc lập hoặc là tương đối độc lập với nhau và chúng ta sẽ dễ bảo trì nó. Các mô đun được ghép nối chặt chẽ nếu chúng dùng các biến chung và nếu chúng trao đổi các thông tin điều khiển (ghép nối chung nhau và ghép nối điều khiển). Ghép nối lỏng lẻo đạt được khi bảo đảm rằng các thông tin cục bộ được che dấu trong các mô-đun và các mô-đun trao đổi thông tin thông qua danh sách tham số (giao diện) xác định.

Trong thiết kế và xây dựng chương trình sự kết dính và ghép nối được thể hiện qua nhiều mức độ khác nhau và một thiết kế tốt được thể hiện qua: ***"high cohesion và low coupling"***



Hình 7-6: Minh họa high cohesion và low coupling

7.4 Thiết kế kiến trúc phần mềm

7.4.1 Tại sao cần thiết kế kiến trúc?

Thuật ngữ "kiến trúc" (architecture) đã có từ lâu trong lĩnh vực xây dựng để miêu tả cấu trúc tổng quát của công trình xây dựng. Trong lĩnh vực công nghệ thông tin, chúng ta cũng có các khái niệm "kiến trúc" ví dụ như kiến trúc hệ thống, kiến trúc phần cứng máy tính và kiến trúc phần mềm.

Thuật ngữ "Kiến trúc phần mềm" (software architecture) được dùng để miêu tả cấu trúc logic của các phần mềm. Nó không phải là phần mềm hoạt động. Thay vào đó, nó là một đại diện cho phép một kỹ sư phần mềm:

(1) Phân tích hiệu quả của thiết kế trong việc đáp ứng các yêu cầu đề ra,

(2) Xem xét lựa chọn thay thế kiến trúc khi thay đổi thiết kế vẫn tương đối dễ dàng,

(3) Giảm thiểu rủi ro gắn liền với việc xây dựng các phần mềm

Sự quan trọng của kiến trúc phần mềm, được lý giải qua có ba nội dung chính:

1. Hỗ trợ việc giao tiếp

Hỗ trợ việc giao tiếp với các thành viên trong dự án. Kiến trúc phần mềm tái hiện một vẻ bề ngoài trừu tượng của hệ thống. Với sự trừu tượng hóa hệ thống với các khái niệm dễ hiểu, những thành viên trong dự án sẽ chỉ cần vận dụng các kiến thức cơ bản của mình về hệ thống trong việc tìm hiểu, dàn xếp, phối hợp làm việc, và bàn bạc trao đổi với nhau.

2. Giúp ra quyết định sớm hơn

Việc ra quyết định được thực hiện sớm hơn. Kiến trúc phần mềm biểu thị các quyết định thiết kế dành cho hệ thống. Như vậy các đội tham gia phát triển, triển khai, kiểm thử và bảo trì phần mềm cũng như các nhóm người dùng và các cấp quản lý sẽ có cái nhìn tổng quan hơn cũng như sớm hơn về hệ thống ngay từ khi nó còn sơ khai. Mỗi đội đó sẽ có các đóng góp ý kiến của mình, các đề xuất cũng như các phản bác của mình khi mọi chuyện chưa quá muộn. Nếu không có quyết định sớm, thì khi phần mềm đã được xây dựng hoàn chỉnh hoặc khá hoàn chỉnh mà đột ngột xuất hiện các yêu cầu thay đổi từ phía nhóm này hoặc nhóm khác, ngoài việc gây trì trệ cho tiến độ công việc mà còn có thể gây ra tâm lý căng thẳng, mâu thuẫn giữa các đội tham gia trong dự án.

3. Biểu diễn mô hình của phần mềm

Kiến trúc phần mềm không phụ thuộc vào một ngôn ngữ cụ thể nào cả mà chỉ tuân theo một số chuẩn của các ngôn ngữ đặc tả nó. Ngoài ra, kiến trúc phần mềm khi được xây dựng, nó tạo thành một mô hình tương đối nhỏ, dễ nắm bắt về cách hệ thống được cấu trúc và cách các thành phần của nó hoạt động cùng nhau. Kiến trúc phần mềm còn chỉ ra cách thức mà phần mềm làm việc với hệ thống. Do vậy, khi ta muốn chuyển phần mềm sang làm việc ở các hệ thống khác có những điểm tương đồng nhất định với hệ thống cũ thì phần mềm này cũng sẽ có các thuộc tính chất lượng và các yêu cầu chức năng được đảm bảo là không quá khác so với khi tồn tại ở hệ thống cũ.

7.4.2 Mô tả kiến trúc

Theo nội dung của tiêu chuẩn IEEE 1471, Một mô tả kiến trúc (Architectural description) là "*một tập hợp các sản phẩm để tài liệu hoá một kiến trúc*". Mô tả kiến trúc biểu diễn nhiều khung nhìn, trong đó mỗi khung nhìn là "sự thể hiện của toàn bộ hệ thống từ quan điểm của một tập hợp các mối quan tâm của [các bên liên quan]".

Một số khung nhìn phổ biến là:

- Khung nhìn theo các chức năng/view logic (Functional/logic view)
- Khung nhìn theo mã nguồn (Code view)
- Khung nhìn theo tư tưởng phát triển/ cấu trúc (Development/structural view)
- Khung nhìn về xử lý đồng thời /tiến trình / thread (Concurrency/process/thread view)
- Khung nhìn vật lý /view triển khai / (Physical/deployment view)
- Khung nhìn theo hành động người sử dụng (User action/feedback view)

Mô tả kiến trúc cần ngắn gọn và dễ hiểu vì nó tạo cơ sở cho việc đàm phán, đặc biệt trong việc xác định ranh giới hệ thống.

- Các nhà phát triển muốn có hướng dẫn rõ ràng, dứt khoát về cách tiến hành thiết kế.

- Khách hàng muốn hiểu rõ ràng về những thay đổi môi trường phải xảy ra và đảm bảo rằng kiến trúc sẽ đáp ứng nhu cầu kinh doanh của họ.
- Các kiến trúc sư khác muốn có sự hiểu biết rõ ràng, nổi bật về các khía cạnh then chốt của kiến trúc.

7.4.3 Kiểu kiến trúc

Mỗi kiểu kiến trúc hay phong cách kiến trúc (architectural style) mô tả một loại hệ thống tương tự như trong các thể loại của các tòa nhà, chúng ta có những phong cách chung như: nhà ở, căn hộ, chung cư, cao ốc văn phòng, tòa nhà công nghiệp, nhà kho, vv. Mỗi kiểu kiến trúc hay phong cách kiến trúc (architectural style) mô tả một loại hệ thống bao gồm:

(1) Một *tập hợp các thành phần* (ví dụ: một cơ sở dữ liệu, mô đun tính toán,…) thực hiện một chức năng cần thiết của một hệ thống,

(2) Một *tập hợp các kết nối* cho phép "truyền thông, phối hợp và hợp tác" giữa các thành phần.

(3) *Các ràng buộc* xác định cách các thành phần có thể được tích hợp để tạo thành hệ thống,

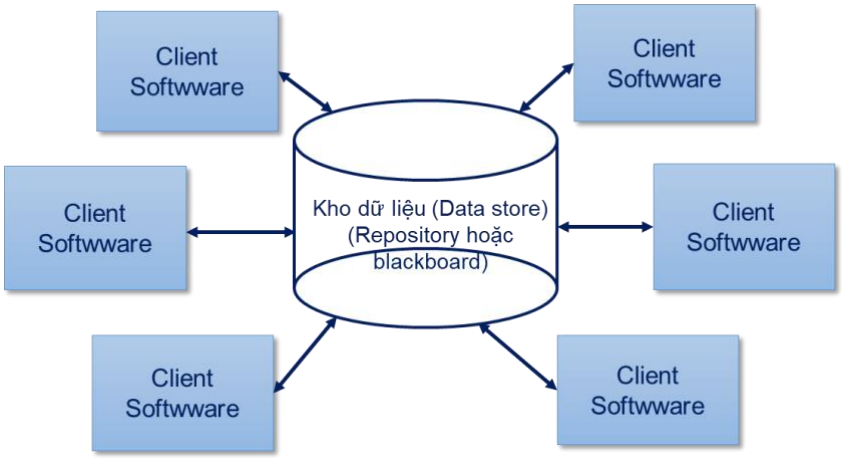
(4) *Các mô hình ngữ nghĩa* cho phép một nhà thiết kế hiểu được tính chất tổng thể của hệ thống bằng cách phân tích các đặc tính đã biết của các bộ phận cấu thành của nó.

Một số Phân loại ngắn gọn về kiểu kiến trúc:

- Kiến trúc lấy dữ liệu làm trung tâm (Data-Centered Architectures)
- Kiến trúc luồng dữ liệu (Data-Flow Architectures)
- Kiến trúc gọi và trả về (Call and Return Architectures)
- Kiến trúc phân lớp (Layered Architectures)
- Kiến trúc hướng đối tượng (Object-Oriented Architectures)
- …

a) Kiến trúc lấy dữ liệu làm trung tâm (Data-Centered Architectures)

Tất cả dữ liệu của hệ thống được quản lý trong một kho chứa tập trung, mọi thành phần chức năng của hệ thống đều có thể truy xuất kho chứa này. Các thành phần không tương tác trực tiếp với nhau, chỉ thông qua kho chứa tập trung.



Hình 7-7: Kiến trúc lấy dữ liệu làm trung tâm

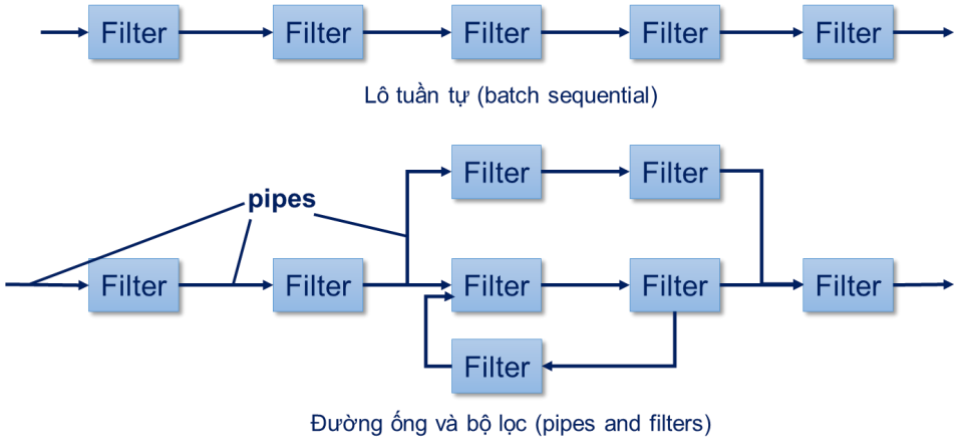
Tình huống nên dùng : khi hệ thống tạo và chứa một lượng rất lớn thông tin trong thời gian dài, hay trong các hệ thống dựa vào dữ liệu, ở đó việc chứa thông tin vào kho sẽ kích hoạt một chức năng hoạt động.

Ưu điểm: các thành phần độc lập nhau, không biết về nhau.

Hạn chế: kho là điểm yếu nhất, nếu có lỗi sẽ ảnh hưởng toàn bộ các thành phần chức năng. Có vấn đề về truy xuất đồng thời kho, phân tán kho trên nhiều máy cũng khó khăn.

b) Kiến trúc luồng dữ liệu (Data-Flow Architectures)

Phần mềm gồm nhiều thành phần độc lập. Mỗi thành phần được gọi là filter, nó xử lý dữ liệu đầu vào theo định dạng xác định rồi tạo kết quả đầu ra theo định dạng xác định.

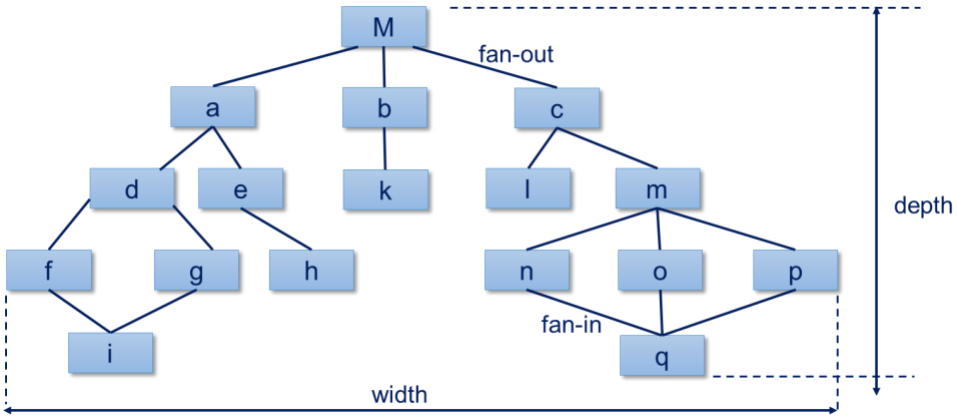


Hình 7-8: Kiến trúc luồng dữ liệu

Kiến trúc này có thể áp dụng cho trong các ứng dụng xử lý dữ liệu mà dữ liệu nhập cần được xử lý bởi nhiều công đoạn khác nhau và có tính độc lập cao trước khi tạo ra kết quả cuối cùng. Ưu điểm là dễ dàng thay đổi/bảo trì/dùng lại từng filter của hệ thống, phù hợp với nhiều hoạt động nghiệp vụ, dễ dàng nâng cấp bằng cách thêm filter mới. Tuy nhiên 2 filter kề nhau cần tuân thủ định dạng dữ liệu chung.

c) Kiến trúc gọi và trả về (Call and Return Architectures)

Phong cách kiến trúc gọi và trả lại được đặc trưng bằng cách xem hệ thống như một thực thể chính có thể gọi các thực thể phụ nhỏ hơn để thực hiện một hành động. Kết quả của hành động này được trả về thực thể chính. Thể hiện bằng cách phân rã theo dạng cây phân cấp: dựa trên mối quan hệ định nghĩa-sử dụng. Các thủ tục có mối quan hệ mật thiết thường đượcgộp thành mô-đun.



Hình 7-9: Kiến trúc gọi và trả về

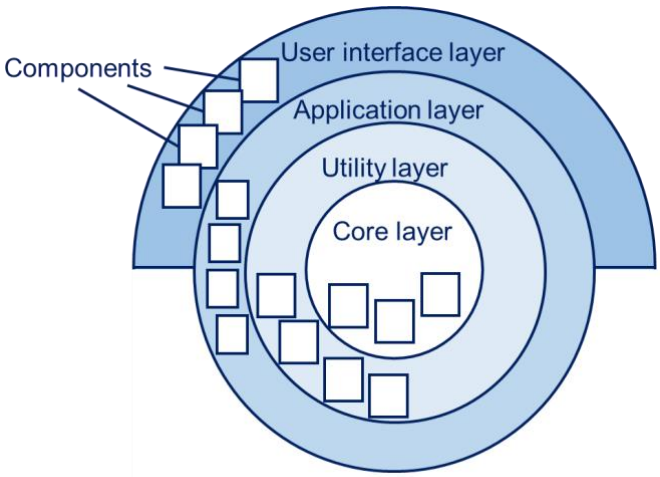
Các thuộc tính chất lượng liên quan đến phong cách kiến trúc này là:

– Tính có thể sửa đổi, chỉ ra rằng một trong các hệ thống con có thể được sửa đổi dễ dàng mà không ảnh hưởng đến các hệ thống (con) khác.

– Khả năng mở rộng, cho biết hệ thống có thể xử lý việc tăng hoặc giảm số lượng người dùng mà không ảnh hưởng đến chức năng của hệ thống.

d) Kiến trúc phân lớp (Layered Architectures)

Hệ thống gồm nhiều lớp chức năng dạng chồng lên nhau, mỗi layer có chức năng cụ thể, rõ ràng và cung cấp các dịch vụ cho layer ngay trên mình. Layer cấp thấp nhất chứa các dịch vụ cơ bản nhất và được dùng cho toàn hệ thống.



Hình 7-10: Kiến trúc phân lớp

Kiến trúc này phù hợp cho các tình huống khi xây dựng thêm khả năng mới trên hệ thống có sẵn, hay khi có nhiều nhóm phát triển khác nhau, mỗi nhóm chịu trách nhiệm về một layer chức năng cụ thể, hay khi có yêu cầu bảo mật nhiều cấp.

Ưu điểm là cho phép hiệu chỉnh bên trong layer bất kỳ sao cho interface không đổi. Có thể giải quyết một chức năng nào đó ở nhiều cấp theo cách thức tăng dần. Tuy nhiên kiến trúc này lại khó tách bạch chức năng của từng cấp, layer trên khó tương tác với layer phía dưới nó nhưng không liền kề. Hiệu quả có thể giảm sút khi nhiều layer phải tương tác nhau để giải quyết một chức năng chung.

7.4.4 Các bước thiết kế kiến trúc

Phần mềm phải được đặt trong bối cảnh: Thiết kế cần xác định các thực thể bên ngoài (các hệ thống khác, thiết bị, con người) mà phần mềm tương tác với và bản chất của sự tương tác. Một tập hợp các nguyên mẫu kiến trúc cần được xác định: Một nguyên mẫu là một trừu tượng (tương tự như một lớp) đại diện cho một phần tử của hệ thống hành vi. Các nhà thiết kế xác định cấu trúc của hệ thống bằng cách xác định và tinh chỉnh các thành phần phần mềm thực hiện từng nguyên mẫu.

- **Bước 1:** Thu thập các kịch bản.
- **Bước 2:** Gợi ý các yêu cầu, ràng buộc, và đặc tả môi trường.
- **Bước 3:** Mô tả các phong cách / mô hình kiến trúc đã được lựa chọn để giải quyết các tình huống và yêu cầu:

  - Quan điểm mô đun
  - Góc nhìn quá trình
  - Quan điểm luồng dữ liệu

- **Bước 4:** Đánh giá chất lượng thuộc tính bằng cách xem xét thuộc tính trong sự cô lập.
- **Bước 5:** Xác định mức độ nhạy cảm của các thuộc tính chất lượng với các thuộc tính kiến trúc khác nhau cho một phong cách kiến trúc cụ thể.
- **Bước 6:** Đánh giá kiến trúc ứng cử viên (được phát triển trong bước 3) bằng cách sử dụng phân tích độ nhạy được thực hiện ở bước 5.

7.5 Thiết kế chi tiết phần mềm

7.5.1 Khái niệm về thiết kế chi tiết

Thiết kế chi tiết là chỉ ra chi tiết và đầy đủ về thành phần, tạo điều kiện xây dựng phần mềm trong pha sau, còn gọi là *thiết kế mức thành phần*. Mục đích: "Cung cấp một thiết kế cho phần mềm triển khai và có thể được xác minh theo các yêu cầu và kiến trúc phần mềm và đủ chi tiết để cho phép mã hóa và thử nghiệm"

Một thiết kế chi tiết thành công là:

- Mỗi thành phần được tinh chỉnh thành các đơn vị phần mềm có thể được mã hóa, biên dịch và kiểm thử
- Các external interfaces của mỗi đơn vị phần mềm được xác định
- Tính nhất quán và nguồn gốc được thiết lập giữa thiết kế chi tiết và các yêu cầu và thiết kế kiến trúc.

Các hoạt động thiết kế chi tiết:

- Thiết kế chi tiết lớp, mô-đun,
- Thiết kế thuật toán,
- Thiết kế dữ liệu,
- Thiết kế giao diện,
- …

Khái niệm về một thành phần (component) là một phần có tính mô-đun, có thể triển khai và thay thế của một hệ thống. Nó đóng gói việc thực thi và đưa ra một tập các giao diện.

- *Góc nhìn hướng đối tượng*: Một thành phần bao gồm một tập các lớp cộng tác
- *Góc nhìn quy ước*: Một thành phần bao gồm logic xử lý, cấu trúc dữ liệu bên trong cần thiết để triển khai logic đó và một giao diện cho phép thành phần được gọi và dữ liệu được truyền đến nó

7.5.2 Nguyên lý cơ bản về thiết kế chi tiết

a) Nhóm nguyên lý SOLID

SOLID là viết tắt của 5 chữ cái đầu trong 5 nguyên tắc thiết kế hướng đối tượng, giúp cho nhà phát triển xây dựng thiết kế để giúp viết ra những đoạn mã nguồn dễ đọc, dễ hiểu, dễ bảo trì, được đưa ra bởi Bob Martin và Michael Feathers.

Nguyên tắc 1: Single responsibility principle (SRP)

Nguyên tắc đầu tiên trong bộ 5 nguyên tắc đó là chữ cái S, với ý nghĩa là một class với trách nhiệm duy nhất, nếu như một class có quá nhiều chắc năng thì nó sẽ trở nên khó hiểu, cồng kềnh và cũng khiến cho các lập trình viên khó có thể duy trì hết toàn bộ.

Nguyên tắc 2: Open/Closed Principle (OCP)

Nguyên tắc thứ hai trong bộ 5 nguyên tắc đó là chữ cái O, Một mô đun (thành phần) nên được mở cho sự mở rộng nhưng nên đóng cho các thay đổi. Thông thường muốn mở rộng chức năng thì phải viết thêm code để thiết kế mô-đun và có thể dễ dàng mở rộng, hạn chế thay đổi code cần thiết. Giải pháp là tách các bộ phận dễ thay thế khỏi các bộ phận khó thay thế, đảm bảo rằng những bộ phận khác không bị ảnh hưởng.

Nguyên tắc 3: Liskov Substitution Principle (LSP)

có ý nghĩa chính là các đối tượng kiểu class con hoàn toàn có thể thay đổi những đối tượng class cha mà không gây ra lỗi gì.

Nguyên tắc 4: Interface Segregation Principle (ISP)

Nguyên tắc này rất dễ hiểu. Hãy tưởng tượng rằng chúng ta hiện đang có được một giao diện của người dùng lớn cùng với khoảng 100 phương thức. Bên cạnh đó việc dư thừa cũng rất dễ dàng bị xáy ra vì lý do không cần sử dụng tất cả 100 phương thức.

Chia giao diện người dùng thành nhiều giao diện nhỏ với các phương thức liên quan giúp việc triển khai và quản lý dễ dàng hơn.

Nguyên tắc 5: Dependency Inversion Principle (DIP)

Nguyên tắc cuối cùng với ý nghĩa:

- Các mô-đun cấp cao thì không nên phụ thuộc vào những nhóm mô-đun cấp thấp, và hơn hết cả 2 nên phụ thuộc vào abstraction.
- Abstraction thì không nên phụ thuộc vào những chi tiết mà nên ngược lại.

Nguyên tắc này hoàn toàn có thể được hiểu như sau: các thành phần của chương trình chỉ nên có sự phụ thuộc vào sự trừu tượng hóa. Bên cạnh đó những thành phần trừu tượng không nên phụ thuộc vào các thành phần cụ thể, mà ngược lại.

Các phần trừu tượng ít thay đổi và đa dạng, tập hợp các đặc tính chung nhất. Những sự vật cụ thể, dù khác nhau đến đâu, đều tuân theo những quy luật chung của cái trừu tượng. Tính trừu tượng phụ thuộc giúp chương trình linh hoạt và thích nghi tốt với sự thay đổi liên tục.

b) Nhóm nguyên lý gắn kết bên trong của thành phần

Một câu hỏi khó trong giai đoạn thiết kế chi tiết đối với nhà phát triển là một lớp sẽ thuộc về thành phần (component) phần mềm nào? Đây là một quyết định quan trọng và cần có sự cân đối từ các nguyên tắc về kỹ thuật phần mềm tốt. Không may, trong nhiều trường hợp, quyết định này đã được đưa ra theo cách bộc phát gần như hoàn toàn dựa trên bối cảnh. Ở đây chúng ta sẽ đề cập về ba nguyên tắc chính của component cohesion (sự gắn kết component):

Nguyên lý phát hành và tái sử dụng tương đương (REP - **Reuse/Release Equivalence Principle**): "Nguyên tắc tái sử dụng là nguyên tắc phát hành". Nói cách khác, nếu một thành phần được coi là có thể tái sử dụng thì nó phải là một đơn vị có thể thay thế được. Hay các class và mô-đun mà được nhóm lại với nhau trong một component nên xử lý làm sao cho có thể release cùng nhau.

Nguyên lý đóng chung (CCP): Các lớp thay đổi cùng nhau thì thuộc về nhau. Gộp vào component các class mà có sự thay đổi với cùng một lý do và thời điểm. Tách ra các component khác các class mà có sự thay đổi với những lý do khác và khác thời điểm. Đây cũng có thể hiểu là nguyên tắc Single Responsibility (SRP, nguyên tắc đầu

tiên trong SOLID) nhưng được phát biểu lại cho *component*. Nếu như SRP nói rằng một *class* không nên bao hàm nhiều lý do để bị thay đổi, thì CCP nói rằng một component không nên có nhiều lý do để bị thay đổi.

Nguyên lý tái sử dụng chung (CRP): Nguyên tắc Common Reuse (CRP) là một nguyên tắc khác giúp chúng ta quyết định xem class và mô-đun nào nên được đặt trong một thành phần. Nó phát biểu rằng các class và mô-đun có xu hướng tái sử dụng cùng nhau thì nên thuộc về cùng một thành phần. Các lớp không được tái sử dụng cùng nhau thì không nên nhóm lại với nhau. Đừng bắt buộc những người dùng của một thành phần phải phụ thuộc vào những thứ họ không cần.

7.5.3 Hướng dẫn thiết kế chi tiết

Hướng dẫn chung:

- *Thành phần*: Quy ước đặt tên nên được thiết lập cho các thành phần được xác định như là một phần của mô hình kiến trúc rồi sau đó tinh chỉnh và xây dựng như là một phần của mô hình mức thành phần
- *Giao diện*: Giao diện cung cấp thông tin quan trọng về sự giao tiếp và cộng tác (đồng thời cũng giúp chúng ta đạt được OCP)
- *Phụ thuộc và kế thừa*: Việc mô hình hóa sự phụ thuộc từ trái sang phải và sự kế thừa từ dưới lên trên

Thiết kế thành phần quy ước

- Việc thiết kế các xử lý logic được quy định bởi các nguyên tắc cơ bản của thiết kế thuật toán và lập trình có cấu trúc
- Việc thiết kế các cấu trúc dữ liệu được định nghĩa bởi các mô hình dữ liệu được phát triển cho hệ thống
- Việc thiết kế các giao diện được quy định bởi sự hợp tác mà một thành phần phải có ảnh hưởng

Thiết kế thuật toán

- Là hoạt động thiết kế sát nhất với việc lập trình
- Cách tiếp cận:

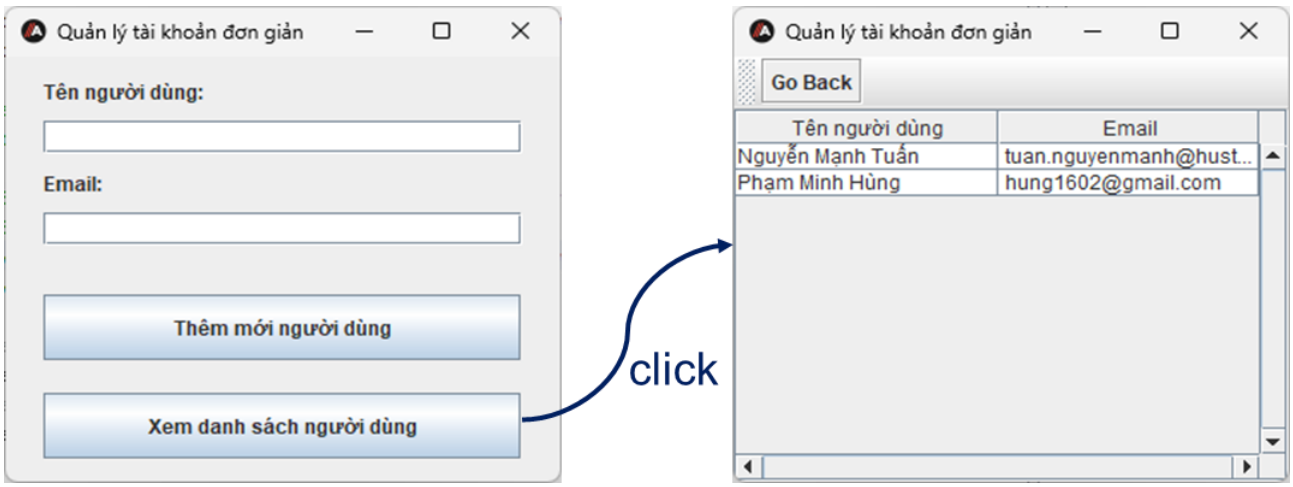
  - Xem xét mô tả thiết kế cho các thành phần
  - Sử dụng tinh chỉnh từng bước để phát triển thuật toán
  - Sử dụng lập trình có cấu trúc để thực hiện logic có tính thủ tục
  - Sử dụng 'phương pháp chính thống' để chứng minh logic
- Biểu diễn thuật toán: Đồ họa (ví dụ flowchart, box diagram), Mã giả (ví dụ PDL), Bảng quyết định, Ngôn ngữ lập trình,…

Thiết kế dữ liệu

- Tìm kiếm biểu diễn logic cho các phần tử dữ liệu đã được nhận diện trong giai đoạn phân tích yêu cầu.
- Thiết kế các cấu trúc dữ liệu của chương trình và cơ sở dữ liệu
- Các tiêu chí thiết kế dữ liệu
  - (1) Không dư thừa dữ liệu và (2) Tối ưu hoá không gian lưu trữ
- Được biểu diễn ở các mức trừu tượng khác nhau
  - Mô hình dữ liệu quan niệm, mô hình dữ liệu logic, mô hình dữ liệu vật lý

7.6 Ví dụ và bài tập

Bài tập: Một phần mềm đơn giản quản lý thông tin người dùng gồm: tên người dùng và địa chỉ email. Hai chức năng: (1)Thêm mới một mục thông tin người dùng vào cơ sở dữ liệu và (2) Liệt kê danh sách thông tin người dùng. Hãy chọn một mẫu thiết kế kiến trúc phù hợp cho phần mềm và biểu diễn mô hình kiến trúc đó. Hình 7.11 minh họa hai màn hình giao diện của phần mềm.



Hình 7-11: Minh họa giao diện phần mềm quản lý thông tin người dùng

Hướng dẫn:

Trong bài tập này, chúng ta Lựa chọn mẫu kiến trúc MVC (Model-View-Controller), giúp tách rời hệ thống phần mềm thành 3 phần logic tương tác lẫn nhau. Mỗi thành phần đều có một **nhiệm vụ riêng** của nó và **độc lập** với các thành phần khác. Tên gọi 3 thành phần:

- **Model (dữ liệu)**: Quản lí xử lí các dữ liệu.
- **View (giao diện)**: Nới hiển thị dữ liệu cho người dùng.
- **Controller (bộ điều khiển)**: Điều khiển sự tương tác của hai thành phần **Model** và **View**.

Mô hình MVC thường được dùng để **phát triển các phần mềm có giao diện** người dùng. Mô hình này giúp đạt được sự tách biệt các mối quan tâm.

Trong bài tập này phần mềm cần phát triển bao gồm 2 kịch bản sử dụng. Với mỗi kịch bản sử dụng chúng ta xác định các lớp cần thiết và đặt nó vào vai trò tương ứng của thành phần trong MVC.

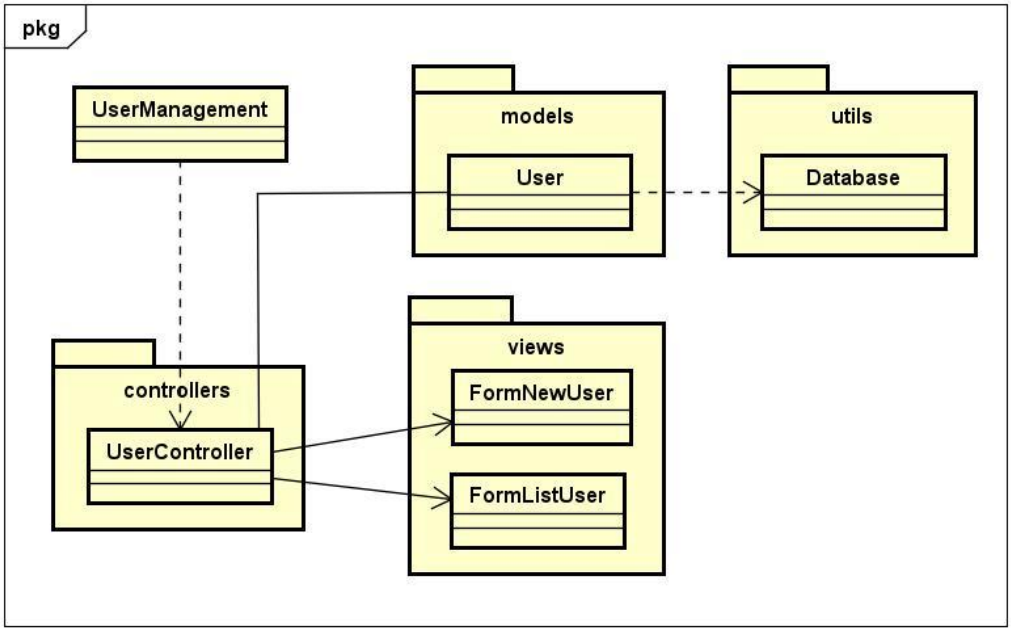
Cụ thể:

- Lớp User đóng gói các thông tin của người dùng
- Lớp FormNewUser và FormListUsers thể hiện các màn hình tương tác với người dùng
- Lớp UserController chứa xử lý logic của ứng dụng cho 2 chức năng: thêm mới và liệt kê danh sách

- Lớp tiện ích Database cung cấp các thao tác với cơ sở dữ liệu và lớp UserManagement là điểm bắt đầu thực thi ứng dụng

Từ các kết quả phân tích vừa rồi, tiếp theo chúng ta sẽ biểu diễn kiến trúc của phần mềm qua sơ đồ gói của UML. Trước tiên chúng ta nhắc lại khái niệm về gói hay package; Các gói trong UML được sử dụng để nhóm các phần tử và cung cấp không gian tên cho các phần tử được nhóm. Một gói có thể chứa các gói khác, tạo ra một cấu trúc phân cấp các gói. Điều này giống như các thư mục trong một hệ thống tệp tịn. Việc phân chia một hệ thống thành nhiều gói làm cho hệ thống trở nên dễ hiểu, đặc biệt là nếu từng gói đại diện cho một phần cụ thể của hệ thống. Một sơ đồ gói hay biểu đồ gói cho thấy sự sắp xếp và tổ chức của các yếu tố mô hình trong một dự án. Nó cho phép thể hiển thị cả cấu trúc và sự phụ thuộc giữa các thành phần của một hệ thống phần mềm.

Trong bài toán này chúng ta xây dựng các packge như minh họa trong hình 7.12.



Hình 7-12: Sơ đồ gói biểu diễn mô hình kiến trúc của phần mềm

Bài tập: Hãy xem xét một hệ thống bao gồm một máy chủ Web và hai máy chủ cơ sở dữ liệu. Cả hai máy chủ cơ sở dữ liệu đều giống hệt nhau: Máy chủ đầu tiên hoạt động như một máy chủ chính, trong khi máy chủ thứ hai hoạt động như một bản sao lưu dự phòng trong trường hợp máy chủ đầu tiên bị lỗi. Người dùng sử dụng trình duyệt Web

để truy cập dữ liệu thông qua máy chủ Web. Hãy biểu diễn ánh xạ các thành phần phần cứng/phần mềm của hệ thống này.

Hướng dẫn:

Bài tập này giới thiệu về việc biểu diễn kiến trúc của hệ thống qua sơ đồ triển khai của UML; Sơ đồ triển khai (Deployment Diagram) là bản vẽ giúp chúng ta xác định sẽ triển khai hệ thống phần mềm như thế nào. Đồng thời, xác định chúng ta sẽ đặt các thành phần phần mềm (component) lên hệ thống ra sao.

Các thành phần của một sơ đồ triển khai bao gồm:

- **Node:** Node là một thành phần vật lý, nó có thể là thiết bị phần cứng hoặc một môi trường nào đó mà các thành phần phần mềm được thực hiện.
- **Relationship:** Deployment Diagram sử dụng quan hệ Association và Dependence để thể hiện mối quan hệ giữa các node với nhau.

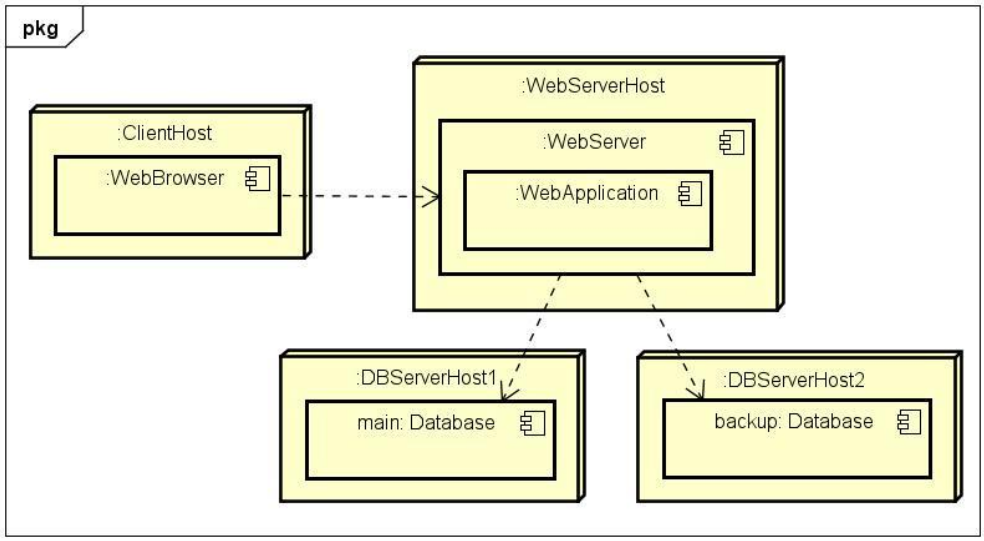
Deployment Diagram có thể ứng dụng vào các trường hợp sau:

– Làm tài liệu để triển khai hệ thống.

– Sử dụng trong thiết kế kiến trúc cho hệ thống.

– Dùng trong giao tiếp với khách hàng, các nhà đầu tư.

Kết quả phân tích các thành phần của hệ thống khi triển khai, chúng ta xây dựng sơ đồ như hình 7.13.

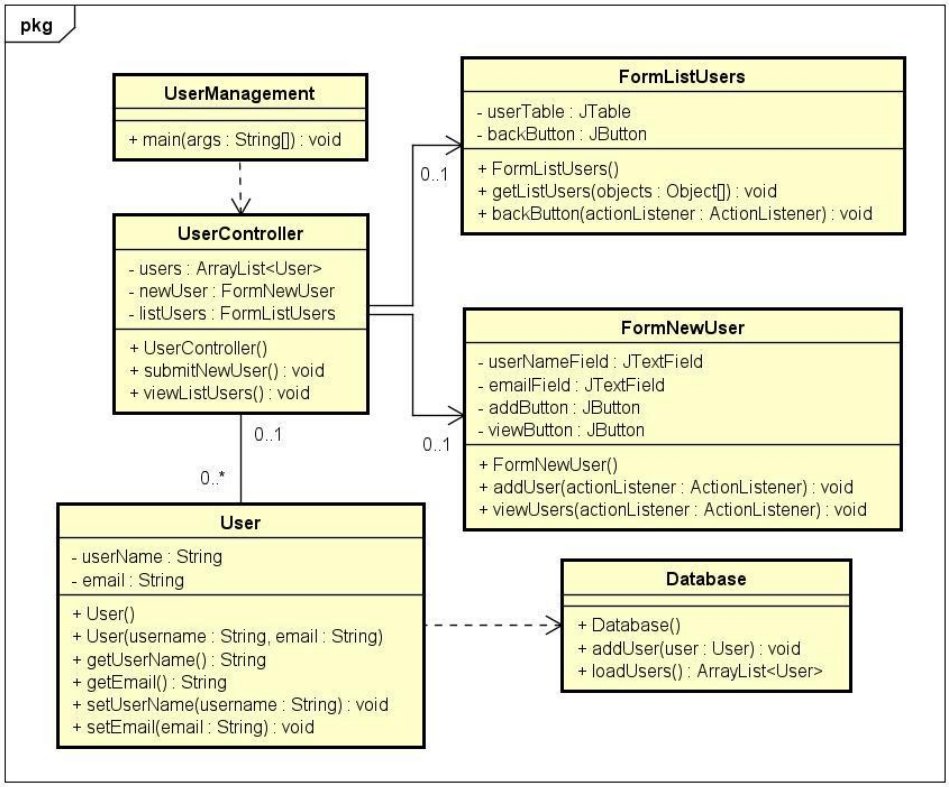


Hình 7-13: Sơ đồ triển khai (deployment diagram)

Bài tập: Xây dựng thiết kế lớp chi tiết cho phần mềm đơn giản quản lý thông tin người dùng ở phần trước.

Hướng dẫn:

- Xác định các thao tác/phương thức
- Xác định mối quan hệ giữa các lớp
- Xác định các thuộc tính
- Xây dựng sơ đồ lớp
- Viết đặc tả chi tiết cho các lớp



Hình 7-14: Sơ đồ lớp (class diagram)

Minh họa viết đặc tả chi tiết cho lớp FormListUsers: Các thuộc tính và phương thức

#	Tên	Kiểu dữ liệu	Giá trị mặc định	Mô tả
1	userTable	JTable	null	Bảng hiển thị danh sách thông tin người dùng
2	backButton	JButton	Null	Nút bấm quay lại màn hình trước

#	Tên	Kiểu trả về	Tham số	Mô tả
1	FormListUser			Phương thức khởi tạo
2	getListUsers	void	objects : Object[]	Hiển thị danh sách thông tin người dùng lên bảng
3	backButton	void	ActionListener actionListener	Xử lý sự kiện nhấn nút Back để quay lại màn hình trước

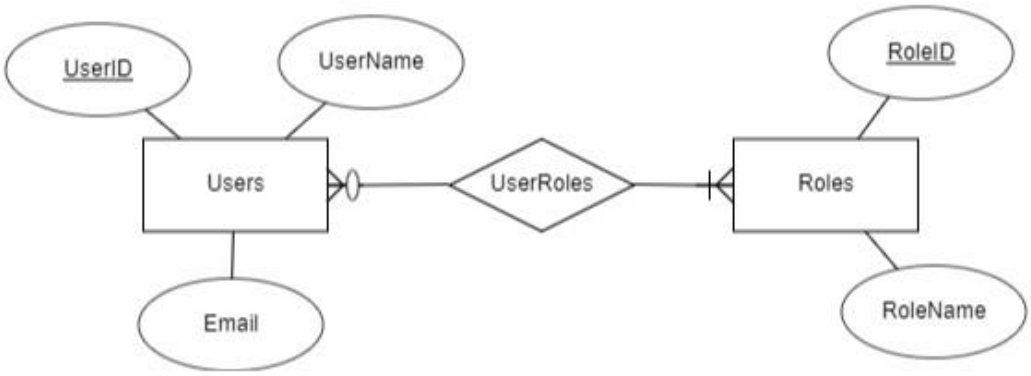
Bài tập: Xây dựng thiết kế dữ liệu cho phần mềm quản lý thông tin người dùng đơn giản.

- Thông tin người dùng gồm có tên người dùng và địa chỉ email
- Mỗi người dùng có thể nhận các vai trò khác nhau trong hệ thống và luôn phải có ít nhất 1 vai trò cụ thể
- Mỗi vai trò được mô tả bằng một tên gọi
- Một vai trò có thể không được gán hoặc gán cho nhiều người dùng khác nhau

Hướng dẫn:

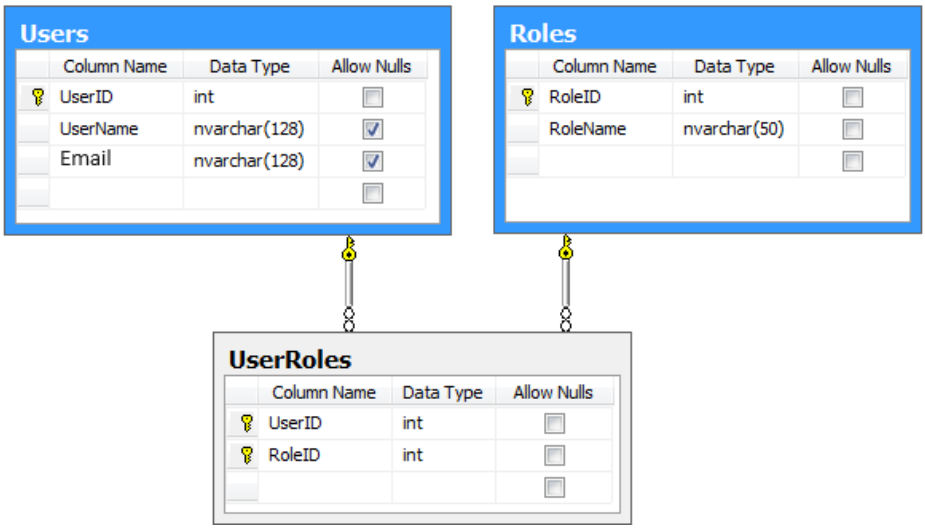
Biểu diễn mô hình dữ liệu mức quan niệm: sử dụng sơ đồ thực thể - mối quan hệ (Entity Relationship (ER) Diagram)

- Xác định các thực thể và thuộc tính:
    - Người dùng: UserID, UserName,Email
    - Vai trò: RoleID, RoleName
- Xác định mối quan hệ: UserRoles (người dùng được gán vai trò)
- Xác định loại quan hệ: nhiều – nhiều
- Xây dựng sơ đồ ER



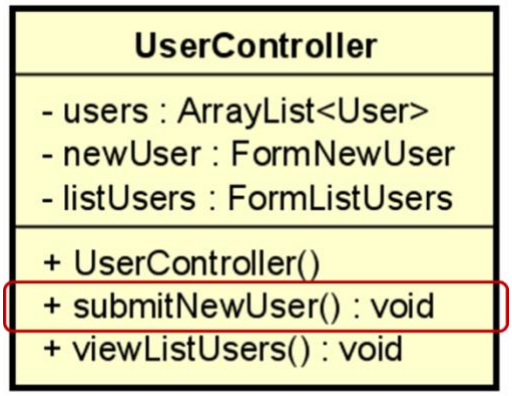
Hình 7-15: Sơ đồ thực thể - mối quan hệ

Biểu diễn mô hình dữ liệu logic: sử dụng mô hình dữ liệu quan hệ, dữ liệu được thể hiện qua các bảng và liên kết giữa các bảng



Hình 7-16: Mô hình dữ liệu quan hệ

Bài tập: Xây dựng biểu diễn thiết kế thuật toán xử lý của phương thức `submitNewUser()` nằm trong lớp `UserController`

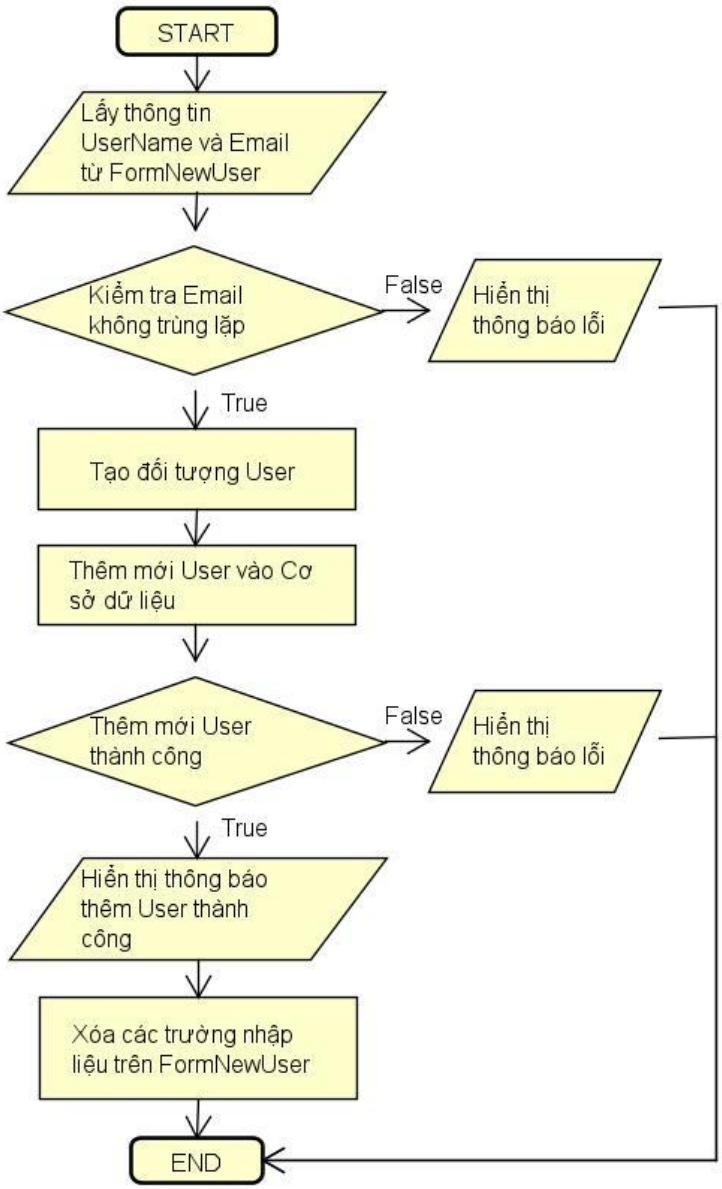


Hình 7-17: Lớp UserController

Hướng dẫn:

Biểu diễn thuật toán bằng lưu đồ (flowchart) bao gồm các xử lý:

- Nhận dữ liệu từ các trường dữ liệu nhập trên View
- Kiểm tra email trùng lặp
- Tạo và thêm mới thông tin User
- Kiểm tra kết quả thực hiện truy vấn



Hình 7-18: Lưu đồ biểu diễn các xử lý của phương thức submitNewUser()

7.7 Thiết kế giao diện người dùng

7.7.1 Tổng quan về thiết kế giao diện người dùng

7.7.1.1 Khái niệm

Giao diện người dùng

Theo ISO 9241-110:2020: Giao diện người dùng là tập hợp tất cả các thành phần của một hệ tương tác, cung cấp thông tin và điều khiển cho người dùng để họ hoàn thành các nhiệm vụ cụ thể với hệ tương tác.

Giao diện người dùng có thể có nhiều dạng nhưng luôn hoàn thành hai nhiệm vụ cơ bản: truyền đạt thông tin từ hệ tương tác tới người dùng, và truyền đạt thông tin từ người dùng đến hệ tương tác. Do đó, giao diện người dùng là một trong số các thành phần quan trọng nhất của bất kỳ một hệ thống máy tính nào.

Các đặc tính mong muốn của giao diện người dùng

Trước đây, khi xây dựng phần mềm, người ta không đặt quá nhiều yêu cầu cho giao diện. Theo đó, giao diện chỉ cần hoạt động được, tức là cung cấp được chức năng tối thiểu để người dùng hoàn thành nhiệm vụ.

Sau đó, yêu cầu của người dùng tăng lên, giao diện không những phải hoạt động được, mà còn phải có tính dùng được và tính dễ truy cập.

Tính dùng được tham chiếu tới năng suất, hiệu quả, sự hài lòng của người dùng khi sử dụng một phần mềm để đạt được các mục tiêu nhất định trong một bối cảnh cụ thể. Năng suất thể hiện qua tính chính xác và đầy đủ mà nhờ đó người dùng đạt được các mục tiêu đã xác định. Hiệu quả thể hiện qua các nguồn lực được sử dụng liên quan đến kết quả đạt được. Sự hài lòng thể hiện qua mức độ mà các phản ứng về thể chất, nhận thức và cảm xúc của người dùng bắt nguồn từ việc sử dụng hệ thống, sản phẩm hoặc dịch vụ đáp ứng được nhu cầu và mong đợi của người dùng.

Tính dễ truy cập: người dùng có nhu cầu, khả năng, tính cách khác nhau đều có thể dễ dàng sử dụng một phần mềm để đạt được các mục tiêu nhất định trong một bối cảnh cụ thể.

Và hiện nay, người dùng còn yêu cầu cao hơn nữa: không những dùng được, dễ truy cập, mà còn phải có lợi và có cảm xúc tích cực khi tương tác với phần mềm qua giao diện. Đó là yêu cầu về trải nghiệm người dùng, tức là hận thức và phản hồi của người dùng, bao gồm cảm xúc, niềm tin, sở thích, nhận thức, sự thoải mái, hành vi và thành tích, có được do việc sử dụng và/hoặc dự kiến sử dụng một phần mềm.

Ứng dụng công thái học trong thiết kế giao diện cho phép dễ dàng tạo ra các giao diện đảm bảo tính dùng được cho người dùng và máy tính. Về phía người dùng, người dùng ít động não, ít động chân tay và có hứng thú khi tương tác. Về phía máy tính, giảm

được bộ nhớ làm việc, giảm sự phụ thuộc vào công nghệ và tăng tính hiệu quả của các tài nguyên sẵn có.

Các lỗi thiết kế giao diện người dùng thường gặp

- Thiếu nhất quán
- Quá nhiều thứ cần nhớ
- Không có hướng dẫn / giúp đỡ
- Không nhạy cảm với ngữ cảnh
- Đáp ứng kém
- Phức tạp / không thân thiện

7.7.1.2 Nguyên tắc thiết kế giao diện

Dễ học

Phần mềm cần phải dễ học cách sử dụng, do đó người dùng có thể nhanh chóng bắt đầu làm việc với phần mềm đó

Quen thuộc

Giao diện nên dùng các thuật ngữ và khái niệm rút ra từ kinh nghiệm của những người sẽ dùng hệ thống nhiều nhất

Nhất quán

Giao diện cần nhất quán sao cho các thao tác gần giống nhau có thể được kích hoạt theo cùng kiểu, phong cách thiết kế các màn hình giao diện phải thống nhất

Ngạc nhiên tối thiểu

Người dùng không bao giờ bị bất ngờ về hành vi của hệ thống

Khôi phục được

Giao diện nên có các cơ chế cho phép người dùng khôi phục lại tình trạng hoạt động bình thường sau khi gặp lỗi

Hướng dẫn người dùng

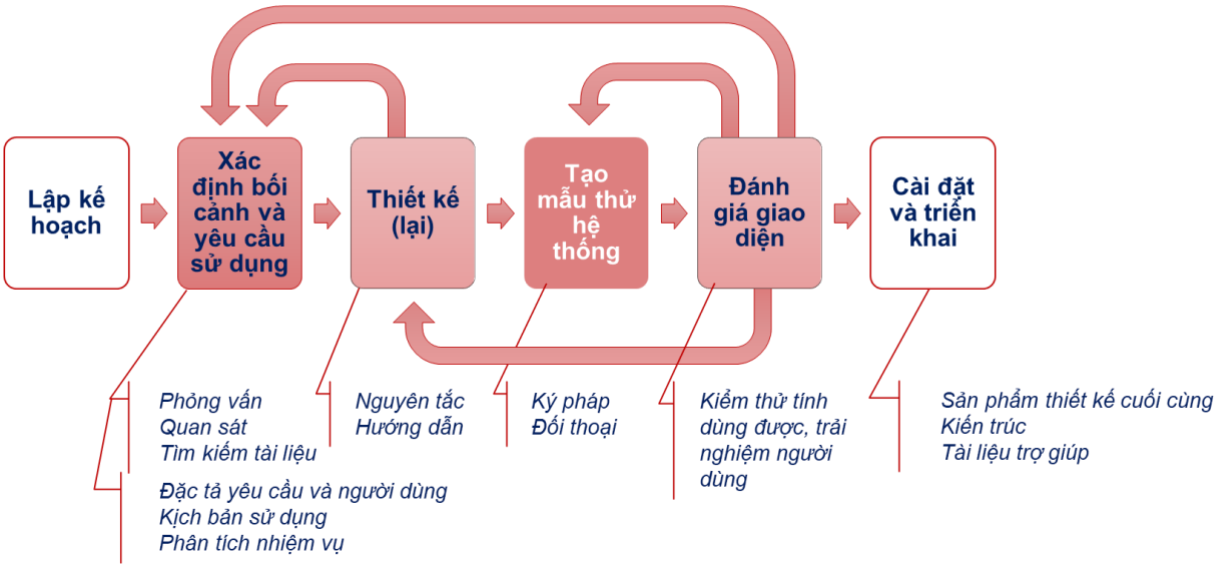
Giao diện nên có phản hồi có nghĩa khi xảy ra lỗi và cung cấp các tiện ích trợ giúp theo ngữ cảnh

Phù hợp với sự khác biệt của người dùng

Giao diện nên cung cấp các tiện ích tương tác thích hợp cho các loại người dùng hệ thống khác nhau

7.7.1.3 Quy trình thiết kế giao diện

Hình 7-19 minh họa quy trình thiết kế mẫu thử giao diện



Hình 7-19: Quy trình thiết kế mẫu thử giao diện

Người dùng có vai trò như thế nào trong quy trình này?

Người dùng là trung tâm trong các hoạt động của quy trình thiết kế giao diện.

Trong Hình 7-20, mỗi vòng tròn tượng trưng cho một hoạt động cần có sự tham gia của người dùng. Các vòng tròn này giao nhau, vì các hoạt động không tách rời mà chồng chéo về thời gian và phạm vi. Phần giao nhau của các vòng tròn thể hiện vai trò của người dùng. Họ tham gia vào việc định nghĩa nhiệm vụ, phản ánh trải nghiệm người dùng, kiểm thử tính dùng được và đánh giá sơ kết. Kết quả của mỗi hoạt động có sự tham gia của người dùng là các tài liệu và sản phẩm trung gian cung cấp thông tin đầu vào cho hoạt động tiếp theo.

Hình 7-20: Vai trò của người dùng

Ví dụ:

Kết quả của hoạt động xác định bối cảnh sử dụng là tài liệu mô tả bối cảnh sử dụng, báo cáo về nhu cầu người dùng.

Kết quả của hoạt động xác định yêu cầu người dùng là tài liệu đặc tả yêu cầu người dùng

Kết quả của giải pháp thiết kế là các tài liệu đặc tả giao diện người dùng và mẫu thử giao diện, tài liệu đặc tả tương tác người dùng và mẫu thử tương tác.

Kết quả của hoạt động đánh giá là các báo cáo mẫu thử, báo cáo đánh giá, báo cáo kiểm thử tính dùng được, báo cáo dữ liệu hiện trường.

Xác định bối cảnh và yêu cầu sử dụng

Bối cảnh sử dụng bao gồm sự kết hợp của người dùng, mục tiêu, nhiệm vụ, tài nguyên và môi trường kỹ thuật, vật lý và xã hội, văn hóa và tổ chức trong đó hệ thống, sản phẩm hoặc dịch vụ được sử dụng.

Giai đoạn này bao gồm hoạt động phân tích trong đó nhóm thiết kế tìm hiểu xem họ sẽ thiết kế phần mềm cho ai và họ sẽ sử dụng nó như thế nào.

Cần trả lời được các câu hỏi:

- Ai là người sẽ tương tác với phần mềm thông qua giao diện (người dùng)? Các bên liên quan khác?
- Đặc tính của người dùng?

- Các nhiệm vụ mà người dùng phải thực hiện để đạt mục tiêu của họ?
- Các nội dung được trình bày như là một phần của giao diện?
- Môi trường trong đó các nhiệm vụ này sẽ được tiến hành?

Có rất nhiều hoạt động có thể tạo điều kiện thuận lợi cho việc này, bao gồm:

- Quan sát công việc đang thực hiện
- Phỏng vấn người dùng
- Xem xét tài liệu đào tạo và hướng dẫn sử dụng
- Thu thập tài liệu và hiện vật được sử dụng
- Các nhóm tập trung và hội thảo
- Khảo sát
- Đánh giá các sản phẩm hiện có (đối thủ cạnh tranh)
- Đánh giá các nghiên cứu đã công bố về lĩnh vực nhiệm vụ
- Người thiết kế tự thực hiện nhiệm vụ và rút ra nhận xét

Đặc tả yêu cầu người dùng

Đặc tả yêu cầu người dùng cung cấp cơ sở cho việc thiết kế và đánh giá các hệ thống tương tác để đáp ứng nhu cầu của người dùng

Dựa trên các mô tả về bối cảnh sử dụng, người dùng được phân thành các nhóm dựa trên hồ sơ của họ, tức là sự hiểu biết, kỹ năng và kiến thức, loại người dùng, v.v..

Yêu cầu được thu thập theo từng nhóm. Dựa trên yêu cầu, nhà phát triển hiểu cách phát triển giao diện.

Khi các yêu cầu đã được thu thập hết, cần tiến hành phân tích chi tiết các yêu cầu này. Việc phân tích yêu cầu có thể được thực hiện bởi 1 trong 2 phương pháp: xây dựng kịch bản tương tác và phân tích nhiệm vụ.

Trong phần phân tích, các nhiệm vụ mà người dùng thực hiện để thiết lập mục tiêu của hệ thống được xác định, mô tả và xây dựng chi tiết.

Việc phân tích môi trường người dùng tập trung vào môi trường làm việc vật lý.

Trong số các câu hỏi được hỏi là:

- Giao diện sẽ được đặt ở đâu về mặt vật lý?

- Người dùng sẽ ngồi, đứng hay thực hiện các tác vụ khác không liên quan đến giao diện?
- Phần cứng giao diện có đáp ứng được các hạn chế về không gian, ánh sáng hoặc tiếng ồn không?
- Có những yếu tố con người đặc biệt nào được cân nhắc bởi các yếu tố môi trường không?

Thiết kế

Mục tiêu

Mục tiêu của giai đoạn này là:

- Xác định tập hợp các đối tượng và hành động giao diện, tức là các cơ chế điều khiển cho phép người dùng thực hiện các tác vụ mong muốn.
- Chỉ ra cách các cơ chế kiểm soát này ảnh hưởng đến hệ thống.
- Chỉ định chuỗi hành động của các nhiệm vụ và nhiệm vụ phụ, còn được gọi là kịch bản người dùng.
- Cho biết trạng thái của hệ thống khi người dùng thực hiện một tác vụ cụ thể.

Mặc dù tất cả quy trình là thiết kế, tuy nhiên: thiết kế giao diện là khâu trọng yếu của quy trình, là nền tảng cho giai đoạn lập trình.

Công việc

Tạo tài liệu thiết kế thể hiện:

- Nội dung (content)
- Kết xuất nội dung (rendering)
- Điều hướng (navigation)
- Hỗ trợ tương tác (interaction support)

Phương pháp

Các phương pháp thiết kế dựa trên: luật tương tác, nguyên lý thiết kế, hướng dẫn phong cách thiết kế.

Các vấn đề về thiết kế nếu có như thời gian đáp ứng, cấu trúc lệnh và hành động, xử lý lỗi và các phương tiện trợ giúp sẽ được xem xét cải tiến trong các vòng lặp của quy trình.

Tạo mẫu thử

Nếu như chúng ta xây dựng hoàn thiện phần mềm rồi mới đưa cho người dùng xem, và họ muốn chỉnh sửa thì sẽ rất tốn thời gian và công sức. Vậy, chúng ta có thể lấy ý kiến của người dùng sớm hơn không? Câu trả lời là có. Chúng ta sẽ cung cấp cho người dùng một sản phẩm chưa hoàn chỉnh, nhưng người dùng có đủ manh mối để đánh giá về sản phẩm và yêu cầu chỉnh sửa cho hợp ý họ. Sản phẩm đó chính là mẫu thử (prototype). Có 2 loại mẫu thử là mẫu thử giao diện và mẫu thử tương tác. Mẫu thử giao diện cho người dùng biết ứng dụng sẽ cung cấp những chức năng gì, nhận thông tin gì từ người dùng và trả về những thông tin gì cho người dùng. Mẫu thử tương tác cho người dùng biết cần làm những gì, theo trình tự nào để có được những thông tin đó.

Mẫu thử giao diện

Trong bước này, mẫu thử giao diện được thiết kế dựa trên đặc tả giao diện người dùng. Đặc tả giao diện người dùng gồm các nội dung như sau:

- Phần tử giao diện biểu diễn nhiệm vụ và hệ thống
- Đặc tính, hành vi, mối quan hệ giữa các nhiệm vụ và hệ thống
- Kỹ thuật đối thoại được sử dụng cho các nhiệm vụ cụ thể (ví dụ: menu, đối thoại dựa trên biểu mẫu, đối thoại lệnh, kết hợp các đối thoại đó)
- Phương thức biểu diễn các phần tử giao diện
    - Điều khiển đầu vào: hộp kiểm, nút radio, danh sách thả xuống, hộp danh sách, nút, chuyển đổi, trường văn bản, trường ngày
    - Thành phần điều hướng: đường dẫn, thanh trượt, trường tìm kiếm, phân trang, thanh trượt, thẻ, biểu tượng
    - Thành phần thông tin: chú giải công cụ, biểu tượng, thanh tiến trình, thông báo, hộp thông báo, cửa sổ phương thức

- Nhóm thông tin: khoảng trắng, đường viền.

Theo đó, mẫu thử giao diện tương ứng với các nội dung thông tin vào ra của ứng dụng và cách thức kết xuất nội dung đó trên giao diện. Các nội dung thông tin này thường không thể kết xuất trong một màn hình duy nhất, do đó giao diện của một ứng dụng gồm nhiều màn hình hoặc nhiều trang, mỗi màn hình chứa một tập các phần tử giao diện.

Các bước thiết kế mẫu thử giao diện bao gồm:

- Thiết kế bố cục
- Thiết kế bản đơn sắc
- Thiết kế bản phối màu

Mẫu thử tương tác

Loại mẫu thử tiếp theo mà chúng ta cần xây dựng là mẫu thử tương tác. Mẫu thử tương tác thể hiện các hoạt động điều hướng của người dùng và cách thức phần mềm hỗ trợ người dùng tương tác. Các hoạt động điều hướng bao gồm điều hướng giữa các màn hình và điều hướng trong từng màn hình để có thể truy cập vào từng phần tử giao diện. Mẫu thử tương tác dựa trên đặc tả tương tác người dùng. Bản đặc tả này bao gôm 5 nội dung:

- Thiết kế quy trình làm việc của người dùng, chỉ ra mối quan hệ tương hỗ tổng thể (bao gồm trình tự) giữa các nhiệm vụ và các thành phần phần mềm, thể hiện trách nhiệm và vai trò của người dùng và của phần mềm.
- Thiết kế nhiệm vụ: tất cả các nhiệm vụ được chia thành các nhóm nhiệm vụ con và phân bổ các nhiệm vụ phụ cho người dùng và phần mềm và yêu cầu liên quan
- Mục tiêu khả năng sử dụng chi tiết theo nhiệm vụ cụ thể
- Mô hình đối thoại: với mỗi nhiệm vụ, thông tin trao đổi giữa người dùng và hệ thống bao gồm trình tự và thời gian cũng như các đối tượng tương tác liên quan và lựa chọn kỹ thuật đối thoại ở cấp độ cao
- Kiến trúc thông tin: từ góc nhìn của người dùng

Đánh giá giao diện

Giai đoạn này tập trung vào việc kiểm tra giao diện. Chúng ta không chờ đợi có thể có một thiết kế hoàn hảo ngay lần đầu tiên. Vì thế cần phải đánh giá để xem mẫu thử tốt đến đâu, và chỗ nào có thể cải thiện được.

Sản phẩm của bước này bao gồm:

Báo cáo đánh giá tính dùng được

Báo cáo này bao gồm các nội dung sau đây:

- Các vấn đề về tính dùng được và khuyến nghị khắc phục
- Mức độ thỏa mãn yêu cầu tính dùng được tối thiểu của sản phẩm
- Sự khác biệt về tính dùng được cho các sản phẩm tương tự
- Thử nghiệm sự phù hợp với yêu cầu người dùng

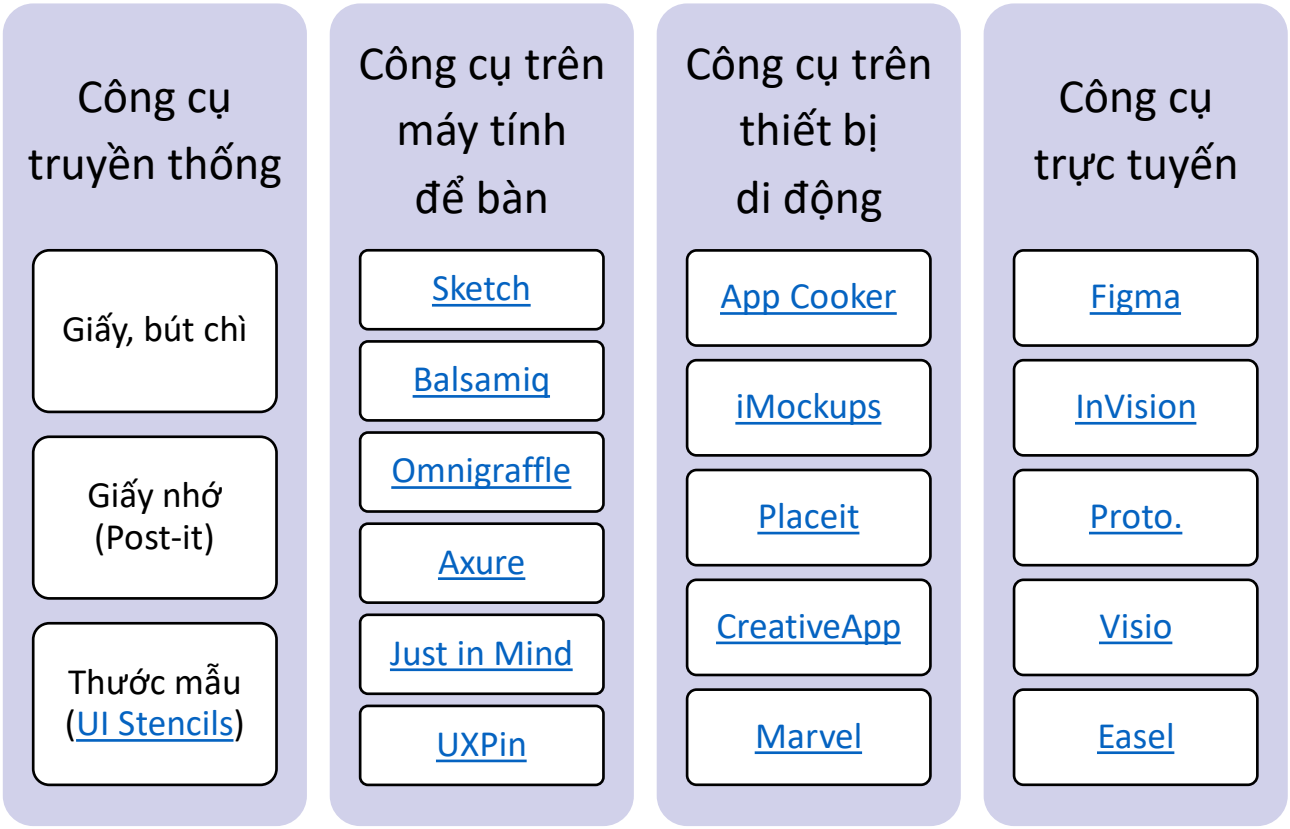
Báo cáo dữ liệu hiện trường

Báo cáo này bao gồm các nội dung sau đây:

- Dữ liệu về cách sử dụng thực tế của sản phẩm (so với mục đích sử dụng sản phẩm). Nguồn dữ liệu hiện trường có thể bao gồm quan sát việc sử dụng, khảo sát mức độ hài lòng của người dùng, số liệu thống kê sử dụng và dữ liệu bàn trợ giúp.
- Bối cảnh sử dụng thực tế
- Phương tiện thu thập dữ liệu
- Lý do thu thập dữ liệu
- Nhu cầu của người dùng đã được xác định
- Yêu cầu bắt nguồn từ người dùng

7.7.1.4 Công cụ hỗ trợ thiết kế

Các công cụ hỗ trợ thiết kế chia thành 4 nhóm:



Hình 7-21: Các nhóm công cụ hỗ trợ thiết kế

7.7.2 Bài tập: Xác định bối cảnh và yêu cầu sử dụng

7.7.2.1 Hiểu và đặc tả bối cảnh sử dụng

Mô tả bối cảnh sử dụng

Bản mô tả bối cảnh sử dụng là một trong các kết quả của hoạt động xác định Bối cảnh và yêu cầu sử dụng.

Bản mô tả bối cảnh sử dụng

Cấu trúc bản mô tả gồm 7 phần:

- Mục tiêu chung của phần mềm
- Các bên liên quan sử dụng phần mềm hoặc chịu tác động của đầu ra của phần mềm trong suốt vòng đời phần mềm
- Đặc điểm của người dùng
- Mục tiêu và đặc điểm của nhiệm vụ
- Thông tin được xử lý trong khi thực hiện nhiệm vụ
- Môi trường kỹ thuật (phần cứng, phần mềm và thiết bị)

- Môi trường vật lý và xã hội

Ví dụ: Một số thông tin về bối cảnh sử dụng của phần mềm quản lý và thu phí ở chung cư Blue Moon

- Bên liên quan: nhân viên quản lý, ban quản trị tòa nhà, ban quản lý tòa nhà, cư dân,
  cơ quan chức năng.
- Môi trường: nhân viên quản lý sẽ truy cập phần mềm từ văn phòng ban quản trị tòa nhà.
- Thiết bị, nền tảng: phần mềm hoạt động trên máy tính cá nhân sử dụng hệ điều hành Windows.
- Tương tác với người dùng: phần mềm tương tác qua giao diện desktop, sử dụng bàn phím, chuột, màn hình cảm ứng, tương thích với một số tiện ích hỗ trợ truy cập như phần mềm phóng to màn hình, phần mềm đọc màn hình….

Tài liệu phân tích người dùng

Tài liệu phân tích người dùng cũng là một trong các kết quả của hoạt động xác định bối cảnh và yêu cầu sử dụng nhằm mô tả đặc điểm của người dùng. Tài liệu này trả lời các câu hỏi sau.

- Người dùng có phải là các chuyên gia đã qua đào tạo, kỹ thuật viên, văn thư hay là công nhân sản xuất?
- Trình độ học vấn trung bình của người dùng là gì?
- Người dùng có thể tự học thông qua các tài liệu đọc hay họ cần theo một khóa đào tạo?
- Người dùng là những chuyên gia đánh máy hay họ sợ bàn phím?
- Độ tuổi trung bình của cộng đồng người dùng là bao nhiêu?
- Giới tính của đại đa số người dùng?
- Người dùng được trả công cho công việc họ thực hiện như thế nào?
- Người dùng chỉ làm việc trong giờ công sở hay họ sẽ làm việc đến khi công việc hoàn thành?

- Phần mềm sẽ trở thành một phần quan trọng trong công việc của người dùng hay nó sẽ chỉ thỉnh thoảng được sử dụng?
- Ngôn ngữ chính người dùng sử dụng là ngôn ngữ nào?
- Sẽ có những hệ quả nào nếu người dùng gây ra lỗi khi sử dụng hệ thống?
- Liệu người dùng có phải là chuyên gia trong lĩnh vực liên quan được xử lý bởi hệ thống?
- Liệu người dùng có muốn biết về công nghệ phía sau giao diện?
- …

Sau khi phân tích, các kết quả thu nhận được từ pha xác định nhu cầu sẽ được sắp xếp theo cách thức nào đó để đưa ra các vấn đề chính và trao đổi với các khâu sau của quá trình thiết kế

Báo cáo về nhu cầu người dùng

Báo cáo nhu cầu người dùng gồm 2 loại nội dung chính

- Nhu cầu của người dùng được xác định, nêu rõ, suy ra và ngụ ý (nhận thức, sinh lý, xã hội) trên tất cả các nhóm người dùng được xác định
- Kết quả phân tích nhu cầu của người dùng liên quan đến bối cảnh sử dụng được mô tả và các hạn chế phát triển của nó đối với nhiệm vụ của từng nhóm người dùng bị ảnh hưởng, bao gồm mọi vấn đề hoặc rủi ro phát sinh từ hệ thống con người

Phân tích và mô hình hóa nhiệm vụ: trả lời các câu hỏi

- Người dùng sẽ thực hiện công việc nào trong những trường hợp cụ thể?
- Nhiệm vụ hay nhiệm vụ con nào sẽ được thực hiện khi người dùng thực hiện công việc?
- Người dùng sẽ thao tác với những nhóm nội dung nào khi công việc được thực hiện?
- Quy trình công việc / Trình tự thực hiện các nhiệm vụ là gì?
- Các nhiệm vụ được phân cấp theo thứ tự nào?

Ví dụ: Cô Nguyễn Thị A.

Hành vi và sở thích:

- Chưa từng dùng phần mềm để xử lý việc thu phí.
- Thích một giao diện người dùng trực quan và tiết kiệm công sức.

Kịch bản sử dụng:

- Cô Nguyễn Thị A, nhân viên quản lý, cần quản lý các khoản thanh toán phí dịch vụ hàng tháng cho khu chung cư. Đầu tháng, cô đăng nhập vào phần mềm, lập danh sách các khoản phí cần đóng cho từng căn hộ và gửi thông báo thu tiền. Mỗi khi cư dân nộp tiền mặt thanh toán các khoản phí, cô lại đăng nhập vào phần mềm, xem xét danh sách các khoản thu đang chờ xử lý, xác nhận các khoản thu đã nhận. Cuối tháng cô dùng phần mềm để tạo báo cáo thống kê giúp Ban quản trị nắm được hiện trạng các khoản thu. Cô A thích trải nghiệm người dùng liền mạch và đơn giản để quản lý hiệu quả các khoản thu.

7.7.2.2 Đặc tả yêu cầu người dùng về giao diện

Có 4 loại yêu cầu chính cần đặc tả.

- Yêu cầu xuất phát từ nhu cầu của người dùng và bối cảnh sử dụng
- Yêu cầu phát sinh từ các tiêu chuẩn, hướng dẫn, kiến thức về công thái học và giao diện người dùng
- Yêu cầu và mục tiêu về tính dùng được, bao gồm các tiêu chí về năng suất, hiệu quả, hài lòng, có thể đo lường được trong bối cảnh sử dụng cụ thể
- Yêu cầu của tổ chức / cộng đồng, có ảnh hưởng trực tiếp đến người dùng

7.7.3 Thiết kế nội dung và kết xuất nội dung

7.7.3.1 Nội dung

Có 2 loại nội dung thông tin.

Nội dung một chiều

Nội dung 1 chiều là các thông tin trả về cho người dùng, như:

- Văn bản
- Ký pháp toán học, khoa học
- Hình ảnh, bản đồ ảnh
- Nội dung đa phương tiện

- Lưới thông tin
- Bảng

Nội dung tương tác

Là những nội dung người dùng có thể tương tác hoặc nhập liệu

- Liên kết
- Thực đơn
- Phím tương tác
- Biểu mẫu nhập liệu
- …

Phần tử giao diện

Phần tử giao diện (UI element) là một thành phần hoặc phần tử cơ bản trong giao diện người dùng (UI) của một ứng dụng. Mỗi phần tử giao diện tương ứng với một nội dung 1 chiều hoặc nội dung tương tác, biểu diễn thông tin mà người dùng cần trao đổi với hệ thống hoặc ngược lại. Các phần tử giao diện đều phải có ý nghĩa đối với người dùng: trợ giúp người dùng thực hiện nhiệm vụ. Như vậy, khi xác định nội dung, chúng ta cũng xác định luôn ý nghĩa cần thể hiện qua phần tử giao diện.

Với mỗi phần tử giao diện, việc trợ giúp người dùng sẽ được thực hiện thông qua một vài yếu tố sau đây:

- Cơ chế điều khiển: Phần tử giao diện tương tác được thiết kế để tương tác với người dùng và cung cấp các chức năng hoặc thao tác cụ thể.
- Văn bản:
- Tổ chức màn hình: Ví dụ, nội dung nào cần đặt cạnh nội dung nào để tăng ý nghĩa.
- Nhấn mạnh: Ví dụ, nội dung nào cần thu hút sự chú ý của người dùng hơn các nội dung khác
- Màu sắc: Ví dụ, dùng màu sắc để nhóm các thông tin như thế nào

- Đồ họa, hoạt họa: Ví dụ, nội dung nào cần có hiệu ứng đồ họa, hoạt họa để thu hút sự chú ý. Có cho phép người dùng tắt, bật hiệu ứng theo yêu cầu của người sử dụng, hay bật tự động.
- Thông điệp: Ví dụ, mỗi biểu mẫu phải có nút xác nhận với nhãn của nút chỉ đúng mục đích. Mỗi trang màn hình phải có tiêu đề tương ứng với thông điệp của trang.
- Thông tin phản hồi: Phải thông báo thành công khi người dùng thực hiện thành công bất kỳ công việc gì.

Các phần tử giao diện được sắp xếp theo nguyên tắc sắp xếp thứ tự thông tin trên màn hình, cụ thể như sau:

- Phân chia thông tin thành các phần logic, có ý nghĩa và dễ cảm nhận
- Tổ chức thông tin theo các cấp độ quan hệ của chúng
- Sắp xếp thông tin theo kỳ vọng và nhu cầu người dùng
- Tạo các nhóm thông tin thỏa mãn các thứ tự sắp xếp phổ biến: Quy ước, Trình tự sử dụng, Tần suất sử dụng, Chức năng, Mức độ quan trọng, Mức độ tổng quát
- Các thông tin cần so sánh phải xuất hiện cùng lúc
- Chỉ các thông tin liên quan đến nhiệm vụ hay nhu cầu của người dùng mới xuất hiện trên màn hình

7.7.3.2 Kết xuất nội dung

Bố cục

Màn hình

Màn hình tham chiếu tới khung cửa sổ hình chữ nhật, có kích thước lớn nhất bằng kích thước màn hình máy tính của người dùng, chứa toàn bộ các thông tin của ứng dụng mà người dùng có thể quan sát tại 1 thời điểm.

Màn hình được thiết kế tốt nếu:

- Phản ánh được năng lực, nhu cầu và nhiệm vụ của người dùng
- Phù hợp với các ràng buộc vật lý của thiết bị hiện thị.
- Sử dụng hiệu quả khả năng của các phần mềm điều khiển

- Đạt mục tiêu nghiệp vụ của hệ thống.

Như vậy, khi thiết kế bố cục, ta cần quan tâm đến các vấn đề sau:

- Số lượng tin cần giới thiệu trên màn hình
- Cách tổ chức màn hình và phân tách các phần thông tin
- Ngôn ngữ
- Sự nhất quán giữa các màn hình. Nếu bố cục các màn hình nhất quán sẽ tiết kiệm được rất nhiều thời gian và công sức thiết kế, cũng như người dùng nhanh chóng làm quen và dễ nhớ.

Cửa sổ

Cửa sổ là một vùng trên màn hình, thường có hình chữ nhật, được xác định bởi một đường viền chứa chế độ xem cụ thể của một số khu vực trên máy tính hoặc một phần nào đó trong hộp thoại của một người với máy tính.

Nó có thể được di chuyển và hiển thị độc lập trên màn hình. Một cửa sổ có thể nhỏ, chứa một thông báo ngắn hoặc một trường đơn lẻ, hoặc có thể lớn, chiếm phần lớn hoặc toàn bộ không gian hiển thị có sẵn. Một màn hình có thể chứa một, hai hoặc nhiều cửa sổ trong ranh giới của nó. Trong bước này, cần lựa chọn kích thước của màn hình theo tỉ lệ thiết kế cho sẵn hoặc theo kinh nghiệm cá nhân.

Phân tích cách kết xuất nội dung

- Liệu các loại dữ liệu khác nhau có được đặt vào vị trí cố định trên màn hình (ví dụ, hình ảnh luôn luôn xuất hiện ở góc trên bên phải)?
- Liệu người dùng có thể tùy chỉnh vị trí màn hình cho nội dung?
- Liệu các nhận dạng phù hợp có được gán cho tất cả nội dung?
- Nếu một báo cáo lớn được trình bày, nó sẽ được phân chia như thế nào cho dễ hiểu?
- Liệu có cơ chế để di chuyển trực tiếp tới thông tin tóm tắt cho lượng dữ liệu lớn?
- Liệu các đầu ra đồ họa có được căn chỉnh để vừa vặn với các giới hạn của thiết bị hiển thị được sử dụng?
- Màu sắc sẽ được sử dụng như thế nào để tăng tính dễ hiểu?

- Thông báo lỗi và cảnh báo sẽ được trình bày tới người dùng như thế nào?

Dạng hiển thị thông tin

Chọn phần tử điều khiển phù hợp trên thiết bị

Chọn phần tử điều khiển phù hợp trên màn hình

Thiết kế đồ họa, biểu tượng, hình ảnh có ý nghĩa phù hợp, tuân thủ hướng dẫn phong cách thiết kế ứng dụng cụ thể.

Cách diễn đạt

Viết văn bản và thông điệp rõ ràng

Sử dụng ngôn ngữ nhất quán

Các nội dung phi văn bản phải kèm theo mô tả

Các nội dung văn bản và mô tả phải ngắn gọn, trong sáng, sử dụng ngôn từ đơn giản, dễ hiểu

Màu sắc

Chọn màu sắc phù hợp, tuân thủ hướng dẫn phong cách thiết kế ứng dụng

Chỉ sử dụng màu sắc đủ để làm nổi bật thông điệp gửi đến người dùng, không hơn.

7.7.3.3 Tạo mẫu thử giao diện

Cách thức thực hiện

- Căn cứ vào các nội dung thông tin đã xác định, tiếp tục xác định các phần tử giao diện và hành động (hoạt động) trên các phần tử giao diện.
- Mô tả từng trạng thái giao diện mà người dùng quan sát được khi thực hiện các hành động trên các phần tử giao diện.
- Xác định cách người dùng diễn giải các trạng thái của phần mềm từ thông tin được cung cấp thông qua giao diện.

Ví dụ: Phác thảo bố cục

Mục đích

Định hình bố cục trang màn hình

Cách thực hiện:

Dựa vào kinh nghiệm của bản thân đưa ra các tiêu chuẩn nên có.

Chia màn hình làm 2 vùng:

Vùng nội dung tương tác: chứa các nội dung rất ít khi thay đổi

Vùng nội dung một chiều: các màn hình khác nhau có nội dung khác nhau

Quy chuẩn các đối tượng để dễ trao đổi và lập trình

- Ảnh là hình chữ nhật: đánh dấu X
- Dòng văn bản: đường kẻ ngang

Hình 7-22: Ví dụ của việc phác thảo bố cục

Ví dụ: Bản đơn sắc

Mục đích

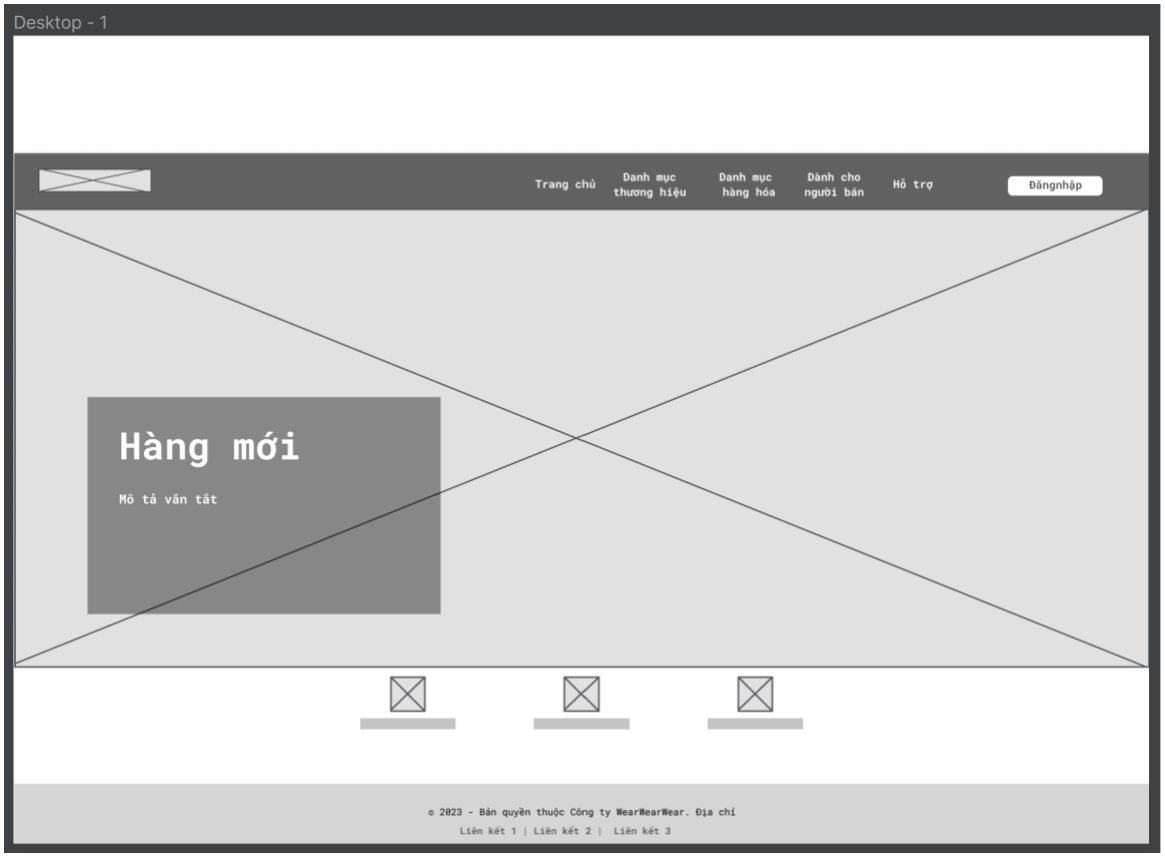
Đánh giá bản phác thảo khi chuyển sang thiết kế đồ họa máy tính có phù hợp với yêu cầu của khách hàng hay không

Cách thực hiện

Sử dụng các công cụ thiết kế đồ họa như: Photoshop, Figma,… để thiết kế mẫu thử giao diện

Chưa thực hiện phối màu cho các thành phần, để ở màu xám

Tuyệt đối không sử dụng hai màu đen, trắng ở vùng muốn phối màu



Hình 7-23: Bản đơn sắc giao diện web bán hàng

Ví dụ: Bản phối màu

Mục đích:

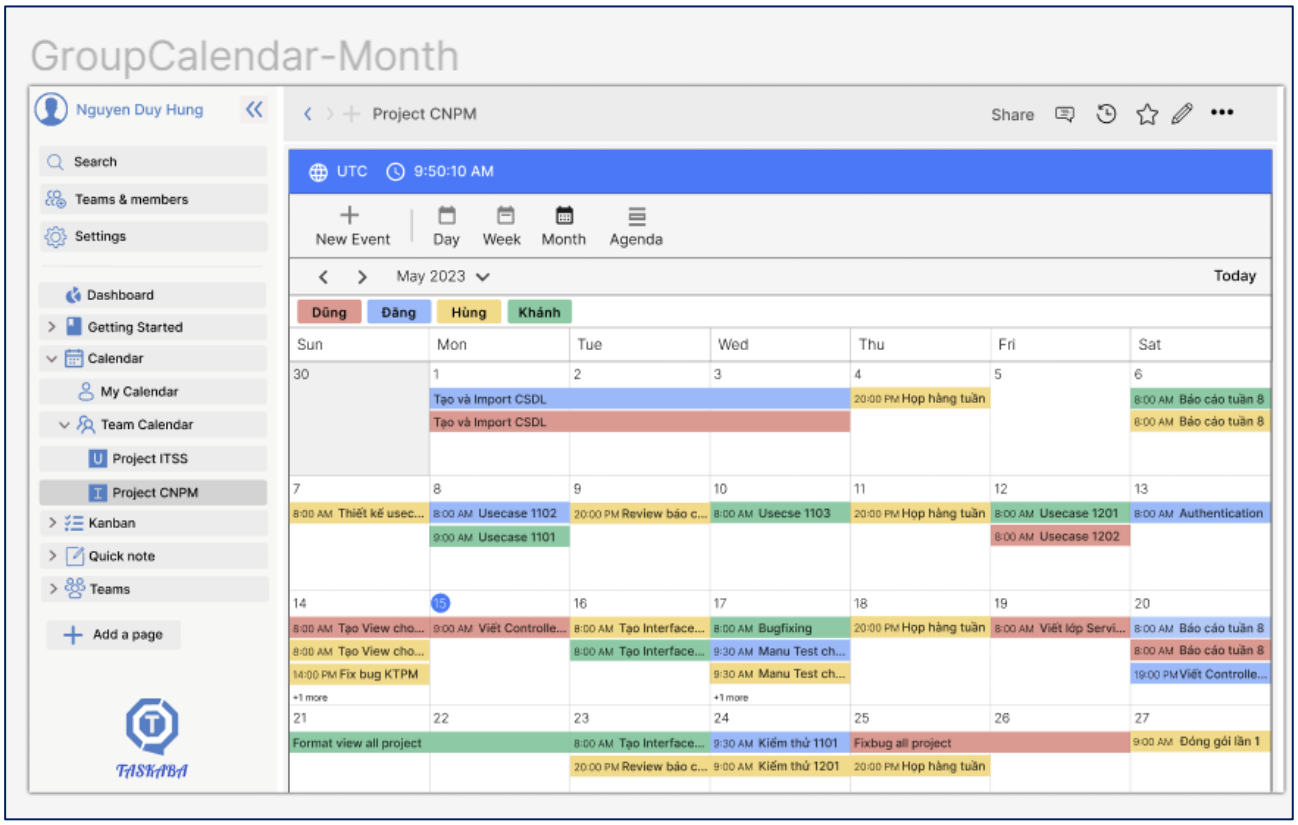
Phối màu cho các thành phần đơn sắc

Cách thực hiện:

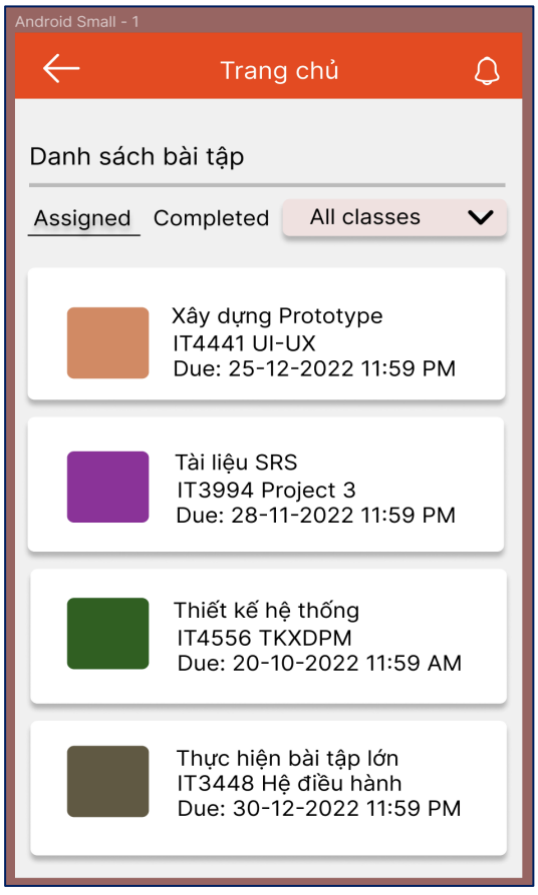
Từ yêu cầu khách hàng đưa ra 1 màu chủ đạo, 1 màu thứ cấp và một mảng các màu hỗ trợ để tăng tính sinh động.

Với văn bản chỉ nên có tối đa 3 màu, 3 kiểu chữ.

Giai đoạn phối màu rất dễ bị ảnh hưởng bởi màu của ảnh, nên chọn ảnh truyền đạt chính xác thông điệp của màn hình.



Hình 7-24: Bản phối màu giao diện web quản lý công việc (nguồn: bài tập lớn của sinh viên)



Hình 7-25: Bản phối màu giao diện di động quản lý công việc (nguồn: bài tập lớn của sinh viên)

7.7.4 Thiết kế điều hướng và hỗ trợ tương tác

7.7.4.1 Điều hướng

Luồng duyệt tin trên màn hình

Người dùng điều hướng bằng cách kết hợp 2 bộ phận: tay và mắt. Mắt để duyệt và tìm thông tin, tay để dịch chuyển con trỏ, hoặc thao tác trên màn hình cảm ứng nhằm truy cập thông tin.

Mắt người thường dịch chuyển từ trái sang phải, từ trên xuống dưới và theo chiều kim đồng hồ. Tay người dễ thao tác nhất theo chiều hướng trên – dưới, trái – phải theo chiều kim đồng hồ.

Nghiên cứu cho thấy, khi phân chia màn hình thành 9 vùng như trên, thì khả năng tìm kiếm thông tin bằng mắt và khả năng truy cập bằng tay sẽ có độ khó dễ khác nhau.

Hình 7-26: Khung màn hình, luồng duyệt tin bằng mắt (đỏ) và luồng duyệt tin bằng tay (xanh) của người dùng

Như vậy, cần tổ chức màn hình sao cho:

- Mắt người dùng duyệt qua các thông tin một các nhịp nhàng, có định hướng
- Tôn trọng cách dịch chuyển tự nhiên của mắt

- Tối thiểu hóa khoảng cách dịch chuyển giữa con trỏ và mắt. Không nên ép buộc hoặc khiến mắt hoặc con trỏ phải di chuyển một quãng đường dài quanh màn hình để tìm mục tiếp theo.

Luồng duyệt tin theo chuyển động của mắt

Khi thiết lập luồng duyệt tin theo chuyển động của mắt qua màn hình, cũng cần lưu ý rằng mắt có xu hướng di chuyển tuần tự, ví dụ:

- Từ vùng tối đến vùng sáng. Từ vật lớn đến vật nhỏ.
- Từ những hình dạng khác thường đến những hình dạng thông thường.
- Từ màu có độ bão hòa cao đến màu không bão hòa.

Đối với các màn hình chứa dữ liệu, hãy xác định vị trí các điều khiển màn hình quan trọng nhất hoặc được sử dụng thường xuyên nhất ở phía trên bên trái màn hình, nơi thường hướng sự chú ý ban đầu.

Luồng duyệt tin theo điều khiển của tay

Việc đặt các điều khiển quan trọng, tần suất sử dụng cao ở góc trên bên trái màn hình cũng sẽ làm giảm tổng số chuyển động của mắt và điều khiển bằng tay cần thiết để làm việc với màn hình.

Sau đó, duy trì luồng thông tin:

- Từ trái sang phải, từ trên xuống dưới cho nội dung 1 chiều
- Từ trên xuống dưới, từ trái sang phải cho nội dung tương tác

Hỗ trợ duyệt tin

Điều hướng trên màn hình phải rõ ràng và dễ thực hiện.

Để hỗ trợ người dùng duyệt tin, ta có thể gióng hàng các phần tử giao diện, nhóm chúng lại, cũng như sử dụng các khung viền để hướng sự chú ý của người dùng.

Việc căn chỉnh các phần tử sẽ giảm thiểu chuyển động quét và điều hướng trên màn hình.

Việc nhóm các phần tử giao diện có thể được thực hiện bằng khoảng trắng. Mắt có thể được hướng dẫn qua màn hình bằng các đường được hình thành từ các khoảng trắng này.

Trình tự sử dụng có thể được thể hiện rõ ràng hơn thông qua việc kết hợp các đường viền xung quanh các nhóm thông tin liên quan hoặc điều khiển màn hình.

Khung viền cung cấp các tín hiệu trực quan liên quan đến việc sắp xếp các thành phần trên màn hình, vì mắt sẽ có xu hướng tập trung bên trong khung để hoàn thành một tác vụ.

Tập trung và nhấn mạnh vào các phần tử giao diện:

- Quan trọng
- Thứ cấp
- Bổ sung

Tuần tự, hướng sự chú ý của một người đến các yếu tố theo tầm quan trọng của chúng. Sử dụng các kỹ thuật hiển thị khác nhau, tập trung sự chú ý vào những phần quan trọng nhất của màn hình. Luôn tab qua màn hình theo thứ tự hợp lý của thông tin được hiển thị và xác định vị trí các nút lệnh ở cuối chuỗi thứ tự tab. Những hướng dẫn để hoàn thành tất cả các mục tiêu chung này sẽ được trình bày ở các trang tiếp theo.

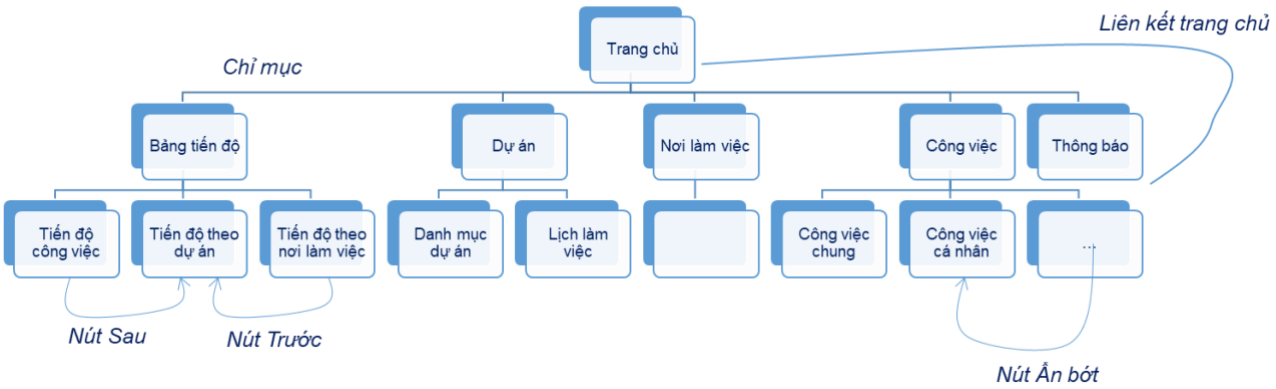
Khi một nhóm các thông tin bị hiện thị trên vài màn hình khác nhau, cần cung cấp điểm ngắt thông tin rõ ràng trong luồng thông tin.

Điều hướng giữa nhiều màn hình

Tương tự như luồng duyệt tin trong một màn hình, việc điều hướng giữa các màn hình cùng cần dễ dàng, nhất quán và đồng điệu. Cần xây dựng một bản đồ ứng dụng, tức là liệt kê toàn bộ các màn hình/trang thuộc ứng dụng, và liệt kê các quy tắc điều hướng có thể được chấp nhận giữa các màn hình đó, sao cho người dùng có thể thực hiện được các tương tác với ứng dụng theo luồng nhiệm vụ và luồng người dùng.

Ví dụ, Hình 7-27 thể hiện bản đồ (site map) của một trang web quản lý công việc. Bản đồ này thể hiện cấu trúc phân cấp của các trang con trong website, kèm theo các quy tắc điều hướng:

- Từ trang chủ có chỉ mục để truy cập đến các trang như bảng tiến độ, dự án, nơi làm việc, công việc, thông báo
- Từ trang bảng tiến độ có thể truy cập đến các trang con như tiến độ công việc, tiến độ theo dự án, tiến độ theo nơi làm việc
- Từ trang dự án có thể truy cập đến trang danh mục dự án, lịch làm việc
- Từ bất kỳ trang con nào cũng đều có thể quay về trang chủ, chuyển sang trang tiếp theo hoặc trước đó
- Với trang công việc cá nhân, có thể xem chi tiết hoặc ẩn bớt thông tin chi tiết.



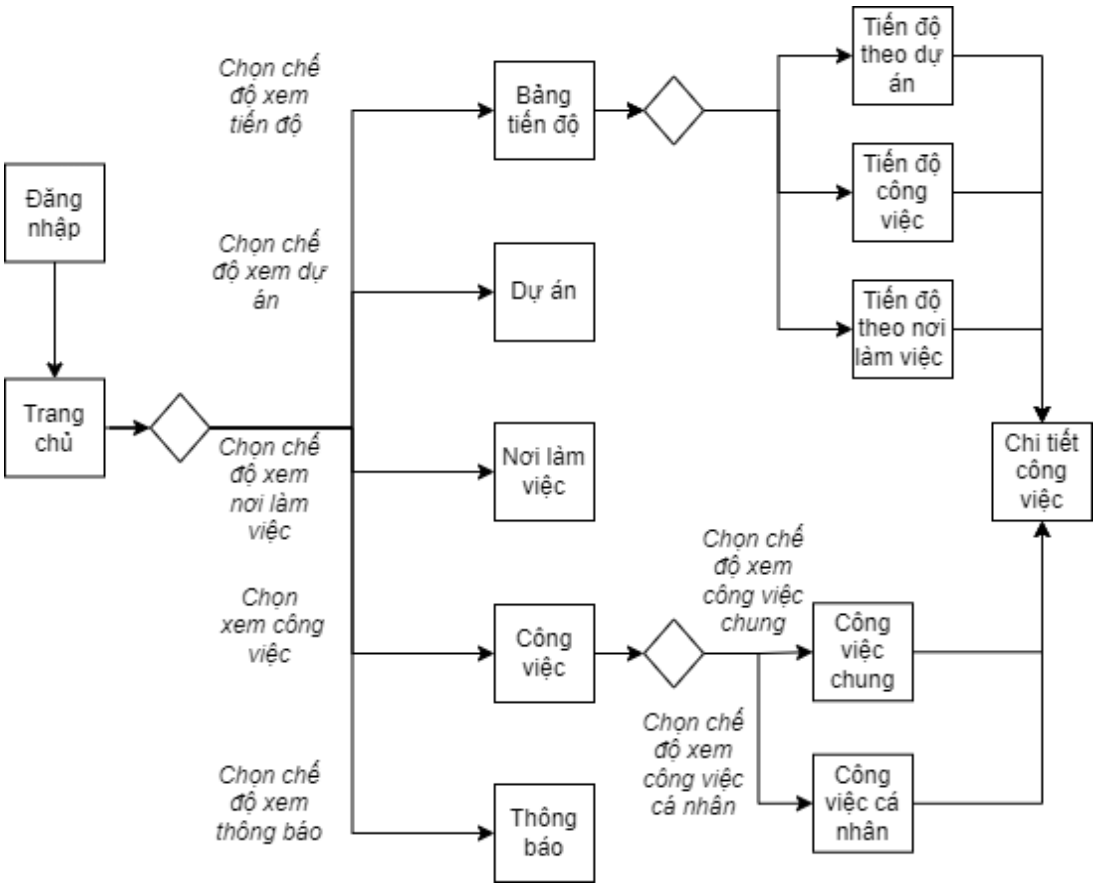
Hình 7-27: Bản đồ trang web quản lý công việc

Luồng nhiệm vụ

Luồng nhiệm vụ (Task flow): Trình tự tương tác người dùng cần thực hiện để hoàn thành một nhiệm vụ hoặc mục tiêu cụ thể trong hệ thống. Mỗi luồng nhiệm vụ tương ứng với 1 đường đi từ gốc đến lá trong bản đồ ứng dụng.

Luồng người dùng

Luồng người dùng (User flow): biểu diễn lộ trình của một cá nhân cụ thể thông qua thiết kế hiện có. Luồng người dùng bao gồm các điểm quyết định trong đó hành trình của cá nhân đến mục tiêu mong muốn có thể khác nhau, dựa trên các quyết định mà cá nhân cần thực hiện khi tương tác với thiết kế. Hình 7-28 thể hiện luồng người dùng trong website quản lý công việc. Nói cách khác, luồng người dùng thể hiện tất cả các đường đi hợp lệ trong bản đồ ứng dụng.



Hình 7-28: Luồng người dùng trong website quản lý công việc

7.7.4.2 Hỗ trợ tương tác

Định vị, tìm kiếm và liên kết thông tin

Định vị

Có cơ chế giúp người dùng xác định được vị trí màn hình hiện tại và quan hệ với các màn hình khác

Có danh sách tổng hợp các nội dung thông tin được cung cấp

Tìm kiếm

Có dịch vụ tìm kiếm nội dung

Liên kết thông tin

Có cơ chế liên kết đến các màn hình có liên quan theo quy trình nghiệp vụ

Có cơ chế để có thể thay đổi, hủy bỏ thao tác điều hướng trước đó.

Trợ giúp

Cần có trợ giúp theo ngữ cảnh.

Ví dụ: Đơn xin việc trực tuyến: Một số câu hỏi có thể khó hiểu cho những người tìm việc mới. Một liên kết giúp đỡ cạnh mỗi câu hỏi sẽ cung cấp chỉ dẫn và giải thích cho mỗi câu hỏi.

Chỉ dẫn

Cung cấp đủ thông tin cho người dùng để thực hiện điều hướng mà không gây nhầm lẫn

Ví dụ: Cung cấp nhãn cho các ô nhập liệu. Ô nhập ngày tháng năm sinh có sẵn văn bản mẫu chỉ ra định dạng đúng cần nhập liệu.

7.7.4.3 Tạo mẫu thử tương tác

Cách thức thực hiện

Căn cứ vào các phần tử giao diện đã xác định khi thiết kế mẫu thử giao diện, xác định hành động (hoạt động) trên các phần tử giao diện.

Xác định các sự kiện (các hành động người dùng) sẽ gây ra sự thay đổi trạng thái của giao diện người dùng. Mô hình hóa hành vi này.

Xác định cách người dùng tương tác với hệ thống thông qua chuỗi hành vi này để đạt được mục đích.

Kiểm tra các chỉ dẫn điều hướng, hỗ trợ tương tác trên giao diện

Giao diện cần

- Cung cấp dấu hiệu truy cập
- Thông báo cho người sử dụng vị trí của họ trong nội dung phân cấp

Giao diện luôn giúp người dùng hiểu tùy chọn hiện tại của họ

- Những chức năng nào khả dụng
- Những liên kết nào còn sử dụng được
- Những nội dung nào có liên quan

Giao diện cần tạo điều kiện chuyển hướng

- Cung cấp một “bản đồ” dễ hiểu để người dùng có thể chuyển hướng trong ứng dụng

Chương 8 XÂY DỰNG PHẦN MỀM

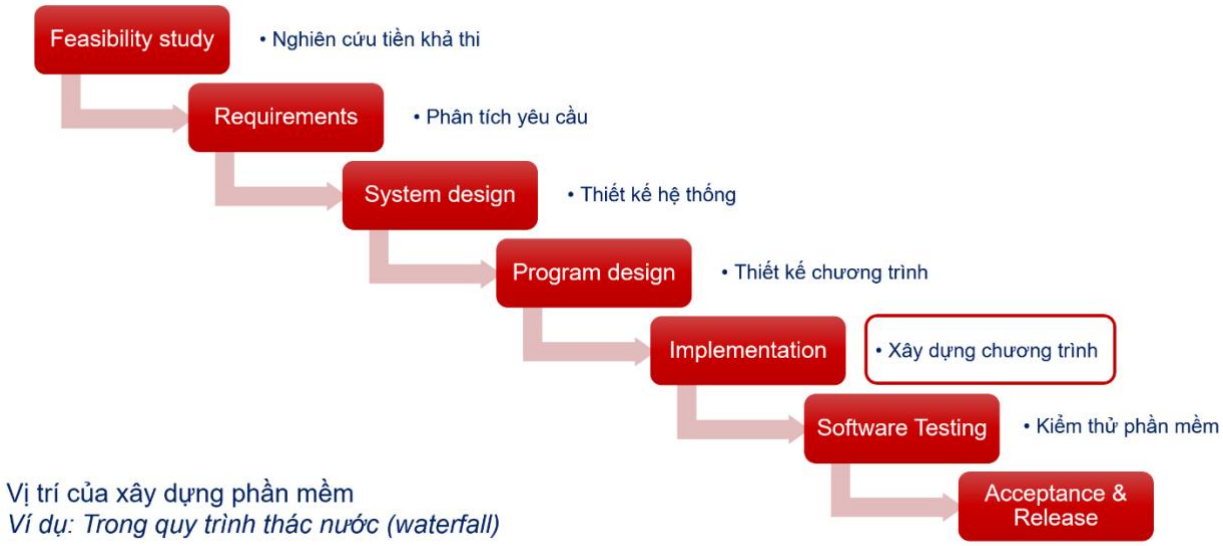
Nội dung:

- Các khái niệm trong xây dựng phần mềm
- Phong cách lập trình
- Tái cấu trúc mã nguồn

8.1 Các khái niệm trong xây dựng phần mềm

Các khái niệm sau đây được sử dụng xuyên suốt bài giảng cơ học kỹ thuật, việc hiểu rõ các khái niệm này có vai trò quan trọng để bắt đầu môn học này.

8.1.1 Xây dựng phần mềm



Hình 8-1: Vị trí của xây dựng phần mềm trong quy trình waterfall

Trong bất kỳ mô hình quy trình phát triển phần mềm (Software Development Process, SDP) nào, thì bước Implementation – xây dựng chương trình, cũng là một bước quan trọng để tạo ra phần mềm. Đối với nhiều lập trình viên, nếu không muốn nói là hầu hết các lập trình viên, đây được coi là là bước quan trọng nhất. Cũng không khó hiểu, vì công việc chính của lập trình viên là lập trình, và lập trình cũng chính là công việc chủ đạo của bước xây dựng phần mềm này. Đối với những người chưa có nhiều kinh

nghiệm trong phát triển phần mềm, nhiều người còn cho rằng lập trình là công việc duy nhất của phát triển phần mềm.

Chúng ta cũng có thể thấy điều đó là không đúng. Thực chất, việc lập trình hay quy trình xây dựng chương trình cũng như các quy trình khác trong phát triển phần mềm, đều có vai trò hết sức quan trọng. Tùy vào các mô hình phát triển phần mềm khác nhau mà vai trò của bước này có thể nhiều hơn hay ít hơn các bước còn lại.

Trong các quy trình phát triển phần mềm khác nhau thì công việc xây dựng phần mềm này có thể nằm ở các giai đoạn khác nhau. Tuy nhiên, nhìn chung thì đây là bước tiếp theo của bước thiết kế phần mềm. Sau khi chúng ta đã có một thiết kế của phần mềm, chúng ta sẽ dựa vào đó để lập trình hay phát triển các chức năng đã thiết kế.

Nhìn vào Hình 8.1, chúng ta có thể nhìn thấy vị trí của bước Xây dựng chương trình trong Quy trình thác nước - waterfall SDP. Ở quy trình này, chúng ta sẽ thiết kế toàn bộ hệ thống / phần mềm của chúng ta ở mức kiến trúc, sau đó chúng ta sẽ tiến hành thiết kế chi tiết cho toàn bộ TẤT CẢ các mô-đun, thành phần, thậm chí cả các lớp và các bảng cơ sở dữ liệu... trước khi bắt tay vào lập trình cho các chức năng đó hay xây dựng chương trình. Việc này cũng tương tự như xây một ngôi nhà, chúng ta vẽ thiết kế ở mức tổng quan, sau đó thiết kế cụ thể từng phòng, thiết kế đường điện đường nước, thiết kế màu sắc, chất liệu nội ngoại thất... Chỉ sau khi chúng ta thiết kế xong thì lúc đó chúng ta mới bắt tay vào xây dựng, sơn trát...

Đối với các quy trình phát triển phần mềm khác, đôi khi việc thiết kế không đi vào quá chi tiết mà những lập trình viên sẽ chủ động hơn trong việc xây dựng chương trình như thế nào cho phù hợp. Sau đó, chương trình sẽ có thể được tinh chỉnh lại trong các pha tiếp theo.

Theo các bạn, vai trò của bước xây dựng phần mềm trong quy trình thác nước (waterfall) là lớn hơn hay nhỏ hơn các quy trình khác như Scrum hay Spiral? Đáp án là Nhỏ hơn, vì hầu hết sự "sáng tạo" đã được định nghĩa ở trong pha Thiết kế chương trình, các lập trình viên ở trong bước này chỉ tập trình vào bước viết mã nguồn cho các thiết kế đó. Trong nhiều trường hợp, các công cụ thiết kế cũng có các tính năng sinh mã nguồn tự động (hầu hết), giảm vai trò của việc lập trình đi rất nhiều. Trong tương

lai, có thể sẽ là một công cụ mô hình ngôn ngữ lớn, hay Large Language Model như ChatGPT có thể làm thay công việc của lập trình viên trong trường hợp này?

8.1.2 Các khái niệm

a) Một số công việc trong xây dựng phần mềm

Coding – viết mã nguồn

Công việc chủ đạo trong bước Xây dựng phần mềm là Coding – viết mã nguồn, hay như nhiều người quan niệm là Programming – lập trình. Thực ra việc này cũng không hoàn toàn chính xác. Nếu như Programming hay lập trình chỉ là gọi chung chung viết mã nguồn ra các chương trình thì việc viết coding hay viết mã nguồn thường được chỉ đến việc "chuyển đổi từ một thiết kế chi tiết – đã có sẵn – của một hệ thống thành mã nguồn." Tức là coding chỉ tập trung vào viết mã thôi. Cũng giống như trong xây dựng, có bản thiết kế rồi thì các bác thợ xây hay thợ nề cũng chỉ cần dựa vào đó và làm theo thiết kế thôi. Chính vì thế mà đôi khi những lập trình viên, đặc biệt ở trong những dự án lớn, theo mô hình thác đổ, thường là ở những công ty outsource, chỉ chủ yếu là làm công việc coding này và hay được gọi bằng cái tên là "thợ code".

Nói như vậy không phải để hạ thấp vai trò của "thợ code". Thợ gì thì muốn giỏi cũng đều đòi hỏi các kỹ thuật rất phức tạp. Ví dụ ngay cả thợ nề, cùng với một thiết kế, thợ nề giỏi có thể xây những bức tường chắc chắn hơn, trát những bức tường phẳng, vuông vức hơn. Và nói chung thì một thợ nề giỏi lương cũng có thể cao hơn nhiều một ông kiến trúc sư quèn. Đối với "thợ code" cũng thế, đối với các thợ code giỏi, họ cũng cần phải nắm được nhiều các kỹ thuật code, hiểu và vận dụng được các mô thức lập trình hay áp dụng được các phong cách lập trình tốt giúp cho mã nguồn đẹp hơn, dễ đọc, dễ bảo trì hơn v.v..

Thông thường, việc viết mã nguồn sẽ được tiến hành với từng mô-đun, thành phần hay các đơn vị - unit đơn lẻ. Sau đó, các mô-đun này sẽ được tích hợp vào nhau để thành các mô-đun lớn hơn hay thành một phần mềm hoàn chỉnh.

Documenting - biên soạn tài liệu đi kèm với mã nguồn

Chúng ta có một khái niệm nữa là "documenting" hay "tài liệu hóa", là công việc biên soạn tài liệu cùng với mã nguồn. Đây là một công việc rất quan trọng, nhưng đôi khi hay bị bỏ qua hoặc thực hiện một cách đại khái trong giai đoạn xây dựng phần mềm. Lý do cũng dễ hiểu, là do hầu hết những người tham gia vào bước xây dựng phần mềm đều là các lập trình viên, và nói chung thì thường chú trọng vào lập trình mà "ngại" việc viết tài liệu. Tuy nhiên, khi làm việc trong một dự án lớn, việc viết tài liệu vô cùng quan trọng, vì nó giúp xác minh được sự phù hợp giữa mã nguồn của chương trình với bản đặc tả của các mô-đun, thành phần hay unit đó, tránh gây ra các conflict giữa các công việc trong project, từ đó có thể đảm bảo được chất lượng của mã nguồn. Ngay cả khi các bạn không làm việc với ai, việc biên soạn tài liệu cũng giúp cho chính các bạn sau này khi cần sử dụng lại hoặc phát triển tiếp mã nguồn của mình.

Ở đây, việc "tài liệu hóa" không phải chỉ là viết ra một "hướng dẫn sử dụng" ở một file Word chẳng hạn, mà có thể là xây dựng một trang web, hoặc viết một cái wiki cho mã nguồn của mình. Những tài liệu – documentation kiểu này được gọi là tài liệu ngoài. Ngoài ra chúng ta còn có khái niệm tài liệu trong, chính là các chú thích hay comment của lập trình viên ở trong mã nguồn. Có một số cách viết chú thích được quy chuẩn, ví dụ như Javadocs, cho phép các IDE đọc và hỗ trợ hướng dẫn lập trình viên ngay trong quá trình lập trình. Chẳng hạn như, ở trước mỗi phương thức, chúng ta có thể chú thích mục đích của phương thức là gì, phương thức này có những tham số đầu vào là gì, input và output của phương thức là gì v.v.. Khi đó, mỗi khi chúng ta sử dụng các phương thức này, các IDE sẽ ngay lập tức hiển thị cho lập trình viên các thông tin về phương thức này và tất cả các phương thức được nạp chồng để lập trình viên có thể dễ dàng chọn lựa và sử dụng. Ngoài ra, một số công cụ (vd. Doxygen) còn có thể tự động sinh ra các tài liệu ngoài từ các chú thích này, giúp các lập trình viên có thể dễ dàng trao đổi với nhau các công việc.

Unit Test - kiểm thử đơn vị

Một khái niệm nữa, có thể chúng ta đã được nghe đến nhiều, là Unit Test – kiểm thử đơn vị. Đây là công việc chúng ta thực hiện các kiểm thử trên từng unit mà chúng ta

đã viết, có thể bằng các giá trị đầu vào đầu ra, hoặc có thể viết các chương trình tự động kiểm thử... Có thể nhiều người cho rằng, việc viết các unit test phải là ở bước tiếp theo của Quy trình phát triển phần mềm, bước Đảm bảo chất lượng hay Kiểm thử. Tuy nhiên trong thực tế thì việc viết các Unit test thường được thực hiện trong bước này do việc này liên quan trực tiếp đến nhóm xây dựng phần mềm và thông thường cũng đòi hỏi việc phải lập trình.

b) Xây dựng chương trình & hệ thống

So sánh việc xây dựng phần mềm với việc "viết chương trình" mà nhiều bạn chúng ta đã quen thuộc, thì thì việc xây dựng phần mềm hay xây dựng hệ thống có một số điểm khác biệt.

Đối với việc viết chương trình, chúng ta chỉ cần nắm được cấu trúc dữ liệu và giải thuật của một chương trình, là từ đó chúng ta đã có thể viết được chương trình rồi. Trong các bạn đang theo học ở đây, có thể có một số bạn là sinh viên các ngành như Cơ khí, điện điện tử, môi trường v.v.. thì có lẽ rất nhiều bạn cũng cần phải có khả năng lập trình và xây dựng được chương trình rồi, để phục vụ cho các công việc tính toán chẳng hạn. Có thể là tính được tải trọng tối đa của một kết cấu nào đó, hay tính được mức độ ô nhiễm của không khí với một lượng xả thải nào đó v.v.. Thì đó chính là viết chương trình hay xây dựng chương trình.

Tuy nhiên, đối với việc xây dựng một hệ thống, thông thường sẽ là công việc trong tương lai của các bạn sinh viên các ngành công nghệ thông tin và truyền thông, là một kỹ sư phần mềm, thì việc nắm vững các cấu trúc dữ liệu và giải thuật là chưa đủ. Lý do là vì một hệ thống phần mềm thường có quy mô lớn, có thể bao gồm nhiều chương trình, mô-đun, nhiều thành phần, và các thành phần này kết nối, tương tác đến nhau. Điều này cũng đồng nghĩa với việc là bất cứ khi nào một thành phần có thay đổi có thể ảnh hưởng đến toàn bộ các thành phần khác hay thậm chí cả hệ thống. Việc này khiến cho việc xây dựng một hệ thống phần mềm, cũng như xây dựng một tòa nhà lớn, không chỉ đơn thuần là xếp gạch lên là được. Chính vì vậy, khi các bạn xây dựng phần mềm, bên cạnh việc tuân thủ theo các thiết kế, chúng ta cũng cần phải chú ý đến các tính dễ

nâng cấp, bảo trì sau này, cũng như các tính phụ thuộc của thành phần với các thành phần khác.

Trong thực tế, các hệ thống lớn thường sử dụng các thư viện hay các framework có sẵn để xây dựng, nhằm mục đích tận dụng các thành phần được xây dựng sẵn. Trong quá trình học trên trường đại học, đôi khi các bạn sinh viên thường được khuyến khích không sử dụng các framework có sẵn, cũng là hợp lý vì các bạn đang trong quá trình học. Tuy nhiên đối với các hệ thống trong thực tế, việc sử dụng các thư viện hay các framework có sẵn là hoàn toàn bình thường và được khuyến khích, trừ phi nó đem lại bất lợi nào đó cho hệ thống (chi phí vận hành, chi phí sử dụng lớn, hay không đáp ứng đủ nhu cầu, hay quá cồng kềnh gây ảnh hưởng chung đến hệ thống…).

c) Mô thức lập trình

Một khái niệm mà chúng ta có thể được nghe đến trong quá trình xây dựng phần mềm là "mô thức lập trình" – programming paradigm. Một mô thức lập trình, có thể hiểu là một mô hình sử dụng một ngôn ngữ lập trình theo một cách tiếp cận hay một phong cách nhất định để giải quyết các vấn đề.

Trong một số trường hợp (vd. mô hình thác đổ), việc thiết kế chương trình kể cả thiết kế hệ thống có thể cũng phải dựa trên một mô thức lập trình nhất định (chẳng hạn như mô thức lập trình hướng đối tượng). Tuy nhiên, đối với một số quy trình phát triển phần mềm, người xây dựng phần mềm có quyền chủ động hơn trong việc lựa chọn mô thức lập trình phù hợp để phát triển tính năng cho hệ thống.

Các mô thức lập trình lập trình thường được chia làm 2 loại: imperative – hướng mệnh lệnh, và declarative – hướng khai báo. Đối với mô thức hướng mệnh lệnh, lập trình viên cung cấp cho máy tính từng bước để hướng dẫn máy tính làm một công việc gì đó. Đối với mô thức hướng khai báo, lập trình viên đưa ra các đặc điểm của kết quả muốn có.

Ví dụ một số mô thức lập trình phổ biến:

- *Mô thức hướng mệnh lệnh*
    - Procedural paradigm: Các câu lệnh được đưa ra lần lượt. Máy tính sẽ thực hiện các lệnh từ trên xuống, giống như đọc một cuốn tiểu thuyết

  - Structure paradigm: Các câu lệnh được tổ chức một cách có cấu trúc (vd. thành các hàm), tương tự như tư tưởng "chia để trị" (divide and conquer).
  - Object oriented programming: Các thành phần của một chương trình được coi là các đối tượng, và có thể tương tác với nhau bằng cách "truyền thông điệp"
- *Mô thức hướng khai báo*
  - Database oriented: Sử dụng các câu truy vấn, query, để yêu cầu máy tính thực hiện một công việc. (vd. CẬP NHẬT Bảng SV, ĐẶT Họ tên là Nguyễn Văn A, TẠI mã SV là 12345)

Trong các mô thức lập trình, mô thức lập trình hướng đối tượng có lẽ là mô thức lập trình phổ biến nhất hiện nay. Mô thức này có các đặc trưng là: Tập trung vào các khái niệm (concepts) thay vì các thao tác, hàm chức năng (operations/functions), và; Mỗi khái niệm có thể biểu diễn một đối tượng dữ liệu trong chương trình (Book, Category, Product etc.). Mô thức lập trình hướng đối tượng có nhiều ưu điểm có thể kể đến như: (1) Dễ dàng tái sử dụng các khái niệm trong các chương trình khác. Ví dụ Book trong chương trình quản lý thư viện và Book trong BookStore có thể được tái sử dụng dễ dàng; (2) Dễ dàng mở rộng các khái niệm cũ (old concepts) thành các khái niệm mới hơn cho các chương trình khác. Ví dụ: MusicPlayer → MP3Player, WinampPlayer, DVDPlayer v.v…; (3) Dễ dàng hơn trong việc định vị các khu vực thay đổi khi các đối tượng dữ liệu liên quan có thay đổi.

8.1.3 Quy trình xây dựng phần mềm

Như có thể thấy ở trong Hình 8.2, ở bên trong bước 4. Xây dựng phần mềm, chúng ta cũng có một quy trình khép kín từ Thiết kế, phân tích, xác thực thiết kế, cài đặt thiết kế, kiểm thử giải pháp và tài liệu hóa giải pháp.

Bước 1: Phân tích yêu cầu - Phân tích hệ thống dựa trên các hệ thống có thực (do con người vận hành hoặc hệ thống tự động). Việc này do các nhà phân tích hệ thống tiến hành, sẽ hiệu quả hơn nếu phỏng vấn người dùng. Mục tiêu là xác định xem hệ thống hiện tại đã làm được những gì, làm như thế nào, còn tồn tại các vấn đề gì. Từ đó quyết định xem có nên thực hiện bước tiếp theo hay không (Return-on-Investment – ROI estimation).

Các công việc chính trong bước này bao gồm

- Thiết lập các requirements
- Gặp các nhà phân tích hệ thống và users
- Xác định input, output, processing, và các thành phần dữ liệu. Từ đó xây dựng ra bảng IPO.

Bước 2: Thiết kế giải pháp - Phân chia hệ thống từng bước thành các thủ tục để giải quyết vấn đề. Có hai hướng tiếp cận là thiết kế hướng đối tượng - thường thực hiện theo bottom-up, và thiết kế cấu trúc - còn gọi là thiết kế top-down, các lập trình viên bắt đầu với thiết kế tổng thể rồi đi đến thiết kế chi tiết.

Ví dụ, lập trình viên có thể tiếp cận theo hướng cấu trúc, bằng cách thiết kế Sơ đồ phân cấp chức năng (hierarchy chart) hay còn gọi là sơ đồ cấu trúc để trực quan hóa các mô-đun chương trình.

Đối với tiếp cận theo hướng đối tượng, lập trình viên đóng gói dữ liệu và các thủ tục xử lý dữ liệu trong đối tượng (object). Các đối tượng được phân loại thành các lớp (classes). Từ đó, thiết kế các biểu đồ lớp thể hiện trực quan các quan hệ phân cấp quan hệ của các lớp.

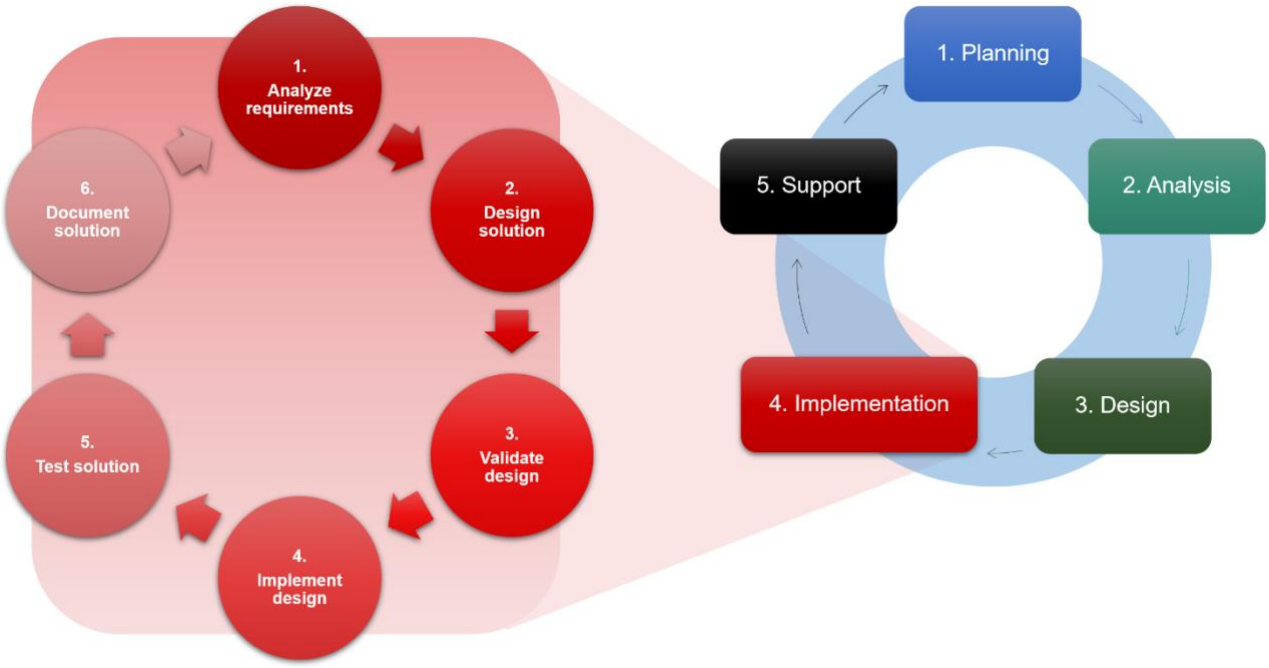
Nếu giải pháp đang cần xây dựng có quy mô không quá lớn, người xây dựng chương trình có thể tiến hành lập trình luôn. Trong trường hợp các bước để cài đặt vẫn còn phức tạp, ta có thể tiến hành thiết kế giải thuật, là một tập các chỉ thị miêu tả cho máy tính nhiệm vụ cần làm và thứ tự thực hiện các nhiệm vụ đó.

Bước 3: Chứng thực thiết kế - Kiểm tra độ chính xác của chương trình. Người lập trình viên có thể kiểm tra tính đúng đắn bằng cách tìm các lỗi logic (Logic Error) - các sai sót khi thiết kế gây ra những kết quả không chính xác. Cách khác là người lập trình có thể dùng các dữ liệu thử nghiệm (Test data) để kiểm tra chương trình. Test data là các dữ liệu thử nghiệm giống như số liệu thực mà chương trình sẽ thực hiện. Phương pháp này còn gọi là desk check.

Bước 4: Cài đặt thiết kế - Dịch từ thiết kế thành chương trình.

Bước 5: Kiểm thử giải pháp - Đảm bảo chương trình chạy thông và cho kết quả chính xác. Trong giai đoạn Xây dựng phần mềm, phương pháp chủ yếu được sử dụng là Debugging - Tìm và sửa các lỗi syntax và logic. Đặc biệt trước khi tung tính năng cho người dùng, cần phải thực hiện kiểm tra phiên bản beta, giao cho người dùng dùng thử và thu thập phản hồi.

Bước 6: Viết tài liệu cho giải pháp. Công việc chính là rà soát lại mã nguồn chương trình - loại bỏ các **dead code**, tức các lệnh mà chương trình không bao giờ gọi đến. Xây dựng và hoàn thiện các tài liệu trong (chính là các chú thích) và tài liệu ngoài của giải pháp.



Hình 8-2: Quy trình phát triển phần mềm

Tất nhiên, đối với các quy trình phát triển phần mềm khác nhau thì việc tổ chức các công việc này có thể không hoàn toàn theo thứ tự này, cũng không nhất thiết phải thực hiện toàn bộ trong bước 4. Ví dụ, trong bước 2. Analysis của dự án, có thể kết hợp trao đổi một số thông tin với khách hàng để lấy các requirements cho việc thực thi. Đối với các mô hình Agile, khi mà client cùng tham gia vào các pha của dự án với nhóm phát triển thì các bước như Phân tích yêu cầu, Xác thực yêu cầu, Kiểm thử giải pháp… sẽ liên tục được thực hiện cùng với client trong suốt vòng đời phát triển của dự án.

8.2 Phong cách lập trình

8.2.1 Khái niệm

a) Thế nào là phong cách lập trình

Để có thể hiểu được khái niệm về phong cách lập trình, trước hết, chúng ta phải trả lời được câu hỏi: "Mã nguồn viết ra là để cho ai đọc?".

Trước hết, rõ ràng là mã nguồn sẽ được đọc bởi các compiler. Đối với các compiler, việc duy nhất quan trọng là mã nguồn phải đúng với cú pháp của ngôn ngữ lập trình. Ví dụ đối với các ngôn ngữ lập trình tựa C thì compiler chỉ quan tâm đến việc các bạn viết đúng cú pháp, đặt dấu ; để kết thúc mỗi câu lệnh. Còn việc các bạn xuống bao nhiêu dòng, cách bao nhiêu khoảng trắng... thì không ảnh hưởng gì đến việc chương trình có biên dịch được hay không. Đối với một số complier của các ngôn ngữ kịch bản như Python, Ruby... Thì compiler có thể quan tâm thêm đến việc bạn thụt dòng bao nhiêu khoảng trắng nữa.

Tuy nhiên, đối với lập trình viên, còn nhiều yếu tố lập trình viên sẽ chú ý khi đọc một mã nguồn, như cách mã nguồn được trình bày thế nào, các tên hàm, tên biến được đặt tên ra sao, hay là cách các lớp, các gói được tổ chức như thế nào... Các yếu tố này hoàn toc không ảnh hưởng được đến việc mã nguồn vẫn có thể biên dịch được hay không, tuy nhiên lại có ý nghĩa to lớn trong việc giúp lập trình viên có thể hiểu rõ được mã nguồn. Chính các yếu tố như vậy được gọi là "phong cách lập trình"

Kernighan, trong cuốn sách "Các thành tố về phong cách lập trình" – The elements of programming style, đã định nghĩa phong cách lập trình như sau:

"Phong cách lập trình (programming style) hay phong cách viết mã nguồn (coding style) là một tập hợp các quy tắc hoặc hướng dẫn được sử dụng khi viết mã nguồn cho chương trình máy tính. Tuân thủ một phong cách lập trình cụ thể sẽ giúp lập trình viên đọc và hiểu mã nguồn phù hợp với phong cách đó, đồng thời giúp tránh phát sinh lỗi"

Chúng ta có thể so sánh phong cách lập trình với thời trang. Trong cuộc sống hàng ngày, nói đến phong cách là chúng ta nói đến thời trang. Đối với phong cách lập trình cũng vậy, có thể coi nó như là "thời trang" ở trong lập trình. Thử xét một ví dụ về thời

trang như sau: Giả sử một bạn giai, giữa mùa hè nhưng mặc váy, khoác áo lông thú ra ngoài đường, thì có vấn đề gì không? Chẳng sao cả, vì luật pháp Việt Nam không quy định cấm cản gì việc này cả. Tuy nhiên, rõ ràng là nếu bạn ăn mặc như vậy thì sẽ nhận được những ánh nhìn xăm xoi của những người đi đường. Tương tự, nếu bạn viết một đoạn mã nguồn, dù vẫn biên dịch được, nhưng nếu nó "xấu" thì cũng khiến cho bạn gặp nhiều khó khăn khi phải làm việc chung với người khác. Tất nhiên, đối với trường hợp bạn không làm việc với ai cả thì tùy bạn, thích viết mã kiểu gì thì viết, cũng như khi bạn ở nhà một mình thì thích mắc gì cũng được. Mặc dù vậy, tốt nhất bạn vẫn nên có một phong cách lập trình tốt để giúp cho chính bản thân bạn sau này khi đọc lại mã nguồn mình đã viết.

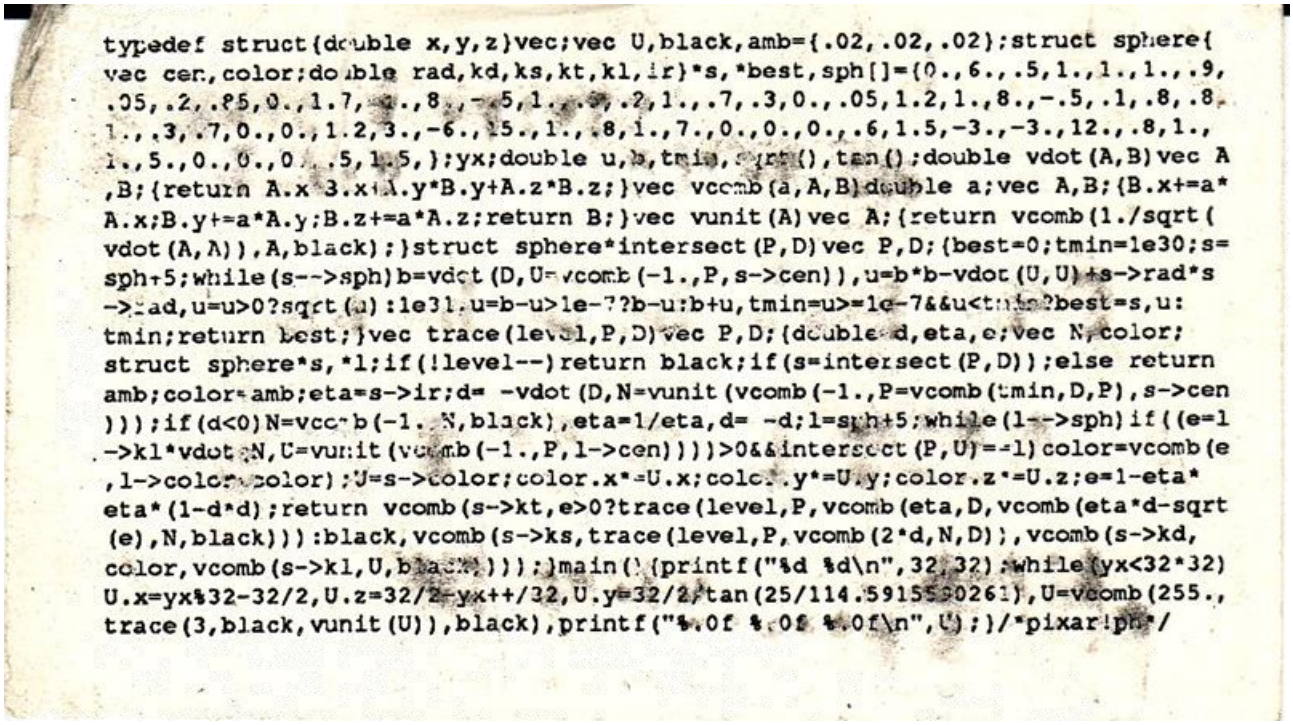
Tất nhiên, đã nói đến thời trang là nói đến vấn đề văn hóa, vì văn hóa mỗi nơi một khác. Cũng ví dụ về "bạn giai, mặc váy, khoác áo lông thú" ở trên, nếu mà không phải ở Việt Nam mà ở Scotland chẳng hạn, thì lại không có vấn đề gì cả. Đối với phong cách lập trình cũng vậy, với mỗi ngôn ngữ lập trình khác nhau, doanh nghiệp khác nhau, hay ở các quốc gia khác nhau v.v.. thì phong cách lập trình cũng có thể khác nhau. Lấy một ví dụ đơn giản, cùng là ngôn ngữ lập trình tựa C, nhưng nếu như C thường thì lập trình viên sẽ xuống dòng rồi mới mở ngoặc nhọn bắt đầu một block mới, thì như Java hay C#, các lập trình viên lại mở ngoặc nhọn ở trong cùng dòng với khai báo. Hoặc như trong Java, các phương thức có thể bắt đầu bằng chữ thường, thì trong C#, các phương thức lại bắt đầu bằng chữ hoa. Trong cùng một ngôn ngữ lập trình, các cộng đồng khác nhau (vd. Cộng đồng mã nguồn mở hay Cộng đồng phát triển phần mềm trong doanh nghiệp) cũng có thể có các phong cách viết mã nguồn khác nhau.

b) Tại sao phải có phong cách lập trình

Phong cách lập trình tốt sẽ giúp cho chương trình của chúng ta dễ debug để phát hiện ra các lỗi trong mã nguồn hơn, dễ nâng cấp và bảo trì hệ thống hơn. Thứ nhất là việc phát hiện ra các lỗi trong mã nguồn. Lý do khiến mã nguồn có lỗi, hầu hết xảy ra do sự nhầm lẫn của lập trình viên, có thể là nhầm lẫn trong việc xác định biến này dùng để làm gì, hay hàm này được gọi như thế nào. Chẳng hạn, khi nhìn thấy một biến i, lập trình viên có thể tự động coi đó là biến đếm và sử dụng, trong khi thực tế đó có thể

không phải. Đối với các hàm, nếu không rõ ràng mục đích của các tham số hay giá trị trả về, có thể lập trình viên sẽ hiểu sai các thức sử dụng các hàm đó. Như vậy, một mã nguồn muốn tốt thì bắt buộc phải là mã nguồn dễ đọc. Việc này cũng sẽ giúp ích rất nhiều cho việc bảo trì sau này, hoặc nâng cấp các tính năng của hệ thống, thông thường có thể được thực hiện bởi các lập trình viên khác không phải là người viết ra mã nguồn đó từ ban đầu. Trong khi đó, như chúng ta đã được học, việc bảo trì phần mềm là một công việc rất tốn kém, do đó việc có một phong cách lập trình tốt có thể giúp tiết kiệm một số tiền lớn cho chi phí phát triển phần mềm.

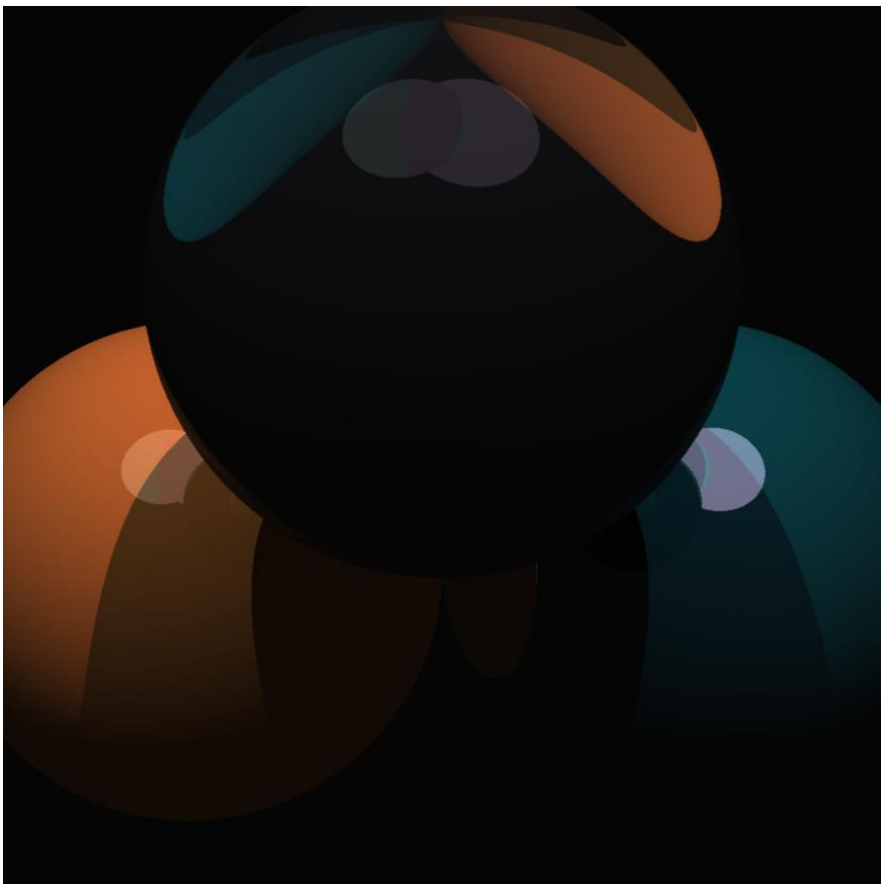
Tất nhiên, trong trường hợp ngược lại, nếu mà lập trình viên vì một lý do gì đó không muốn người khác có thể dễ dàng hiểu được mã nguồn của mình, hay không quan tâm đến việc đó thì có thể làm ngược lại: Làm cho mã nguồn của mình khó đọc hơn. Ví dụ ở đây là mã nguồn của một ray-tracer tối giản của Paul Heckbert, là một kỹ thuật render được sử dụng trong công nghệ đồ họa. Lý do khiến Paul viết mã nguồn kiểu này là để... mã nguồn ngắn gọn hết sức có thể, đủ để đưa vào mặt sau của một tấm card visit.



Hình 8-3: Mã nguồn ray-tracer tối giản, viết trên mặt sau card visit của Paul Heckbert

(nguồn: https://www.realtimerendering.com/)

Mã nguồn này khi chạy sẽ render ra màn hình hình ảnh như Hình 8.4.



Hình 8-4: Ray-tracer của Paul Heckbert

Một ví dụ khác là khi viết các plugin cho các browser, do các plugin này có thể dễ dàng giải nén và đọc được mã nguồn nên các lập trình viên thường cố gắng khiến cho mã nguồn trở nên khó đọc. Thực ra, khi lập trình, các lập trình viên vẫn áp dụng nghiêm chỉnh các nguyên tắc lập trình cơ bản, tuy nhiên trước khi deploy thì các lập trình viên sẽ chạy công cụ để Replace All tất cả các khoảng trắng, tên biến... khiến cho mã nguồn trở nên rối rắm hơn.

Tất nhiên, đây chỉ là các trường hợp đặc biệt. Còn trong thực tế, có phong cách lập trình tốt sẽ mang lại rất nhiều lợi ích cho việc phát triển phần mềm.

8.2.2 Quy ước viết mã nguồn

Quy ước viết mã nguồn (coding convention) hay quy ước mã nguồn (code convention) là một tập hợp các hướng dẫn, guideline, chung để giúp cho mã nguồn dễ đọc, dễ hiểu hơn, đồng thời giúp cho các nhà phát triển có thể cùng làm theo, giúp cho dự án phần mềm có phong cách lập trình tốt hơn. Quy ước viết mã nguồn đôi khi cũng được gọi là phong cách viết mã (coding style).

a) Một số quy tắc chung

Quy ước viết mã nguồn không phải là yêu cầu bắt buộc, và cũng không phải chỉ có một quy ước duy nhất mà phụ thuộc vào từng lập trình viên, cơ quan, tổ chức hay một cộng đồng nhất định. Do đó, trên thực tế có vô số các quy ước viết mã nguồn. Các quy ước viết mã nguồn của các tổ chức lớn có thể kể đến như APACHE coding convention, GNU coding convention, Sun Java coding convention, FreeBSD coding convention, Linux kernel, Google styleguide v.v… Mỗi cá nhân cũng có thể tự xác định cho mình một số quy ước riêng. Tuy nhiên, có một số quy tắc chung mà bất kỳ quy ước viết mã nguồn nào cũng nên dùng.

- Thứ nhất, đấy là phải **nhất quán**. Tức là khi các bạn đã sử dụng quy ước nào cho thành phần gì thì quy ước đó cũng phải được áp dụng cho các thành phần đó trong toàn bộ mã nguồn. Ví dụ, quy tắc đặt tên sẽ phải được tuân thủ trong toàn bộ mã nguồn, việc sử dụng các biến phải được nhất quán. Chẳng hạn như nếu tên thuộc tính đã viết hoa chữ cái đầu thì trong toàn bộ mã nguồn của mình, tên thuộc tính nào cũng phải được viết hoa chữ cái đầu. Việc này giúp cho lập trình viên không bị nhầm lẫn giữa thuộc tính và các biến dữ liệu.
- Thứ hai là tính **khúc triết**. Ví dụ, mỗi phương thức có một mục tiêu rõ ràng, tránh trường hợp một phương thức làm hai hay nhiều việc cùng một lúc. Các phương thức cũng không nên quá dài, cần phải đủ ngắn để người đọc có thể dễ dàng hiểu được. Số lượng tham số của phương thức là đủ nhỏ, ví dụ thông thường mỗi phương thức nên có dưới 6 tham số.
- Thứ ba là mọi thứ cần phải **rõ ràng**, ví dụ như có các chú thích rõ ràng đầu mỗi phương thức hay trước mỗi đoạn để chỉ ra phương thức này hay đoạn này đang chuẩn bị làm việc gì.
- Thứ tư là tính **bao đóng**, tức là những mã nguồn viết ra cần có đầu vào đầu ra rõ ràng, và không làm ảnh hưởng đến các giá trị khác. Ví dụ như đối với hàm, giá trị duy nhất mà hàm nên tác động tới là giá trị trả về, không nên làm thay đổi bất kỳ giá trị nào khác trong thân hàm. Hay đối với các vòng lặp, chúng ta không được thay đổi các giá trị của biến. Ví dụ như ở dưới đây, biến i chỉ nên sử dụng làm biến đếm. Việc thay đổi giá trị i ở bất kỳ chỗ nào trong vòng lặp có thể khiến cho lập trình viên khi rà soát mã nguồn không nhận ra được điều này, và khiến cho chương trình trở nên khó khăn hơn nhiều khi bảo trì.

Sau đây chúng ta sẽ tìm hiểu một số các quy ước viết mã nguồn thông dụng. Ví dụ, đối với cấu trúc mã nguồn, chúng ta có thể có một số các quy ước về khoảng trắng, cách lề, cách đoạn...

b) Cấu trúc mã nguồn

Đối với khoảng trắng, các compiler không quan tâm đó là 1 space, nhiều space hay tab... Do đó, các lập trình viên cần chủ động sử dụng các khoảng trắng làm sao cho dễ đọc và nhất quán. Ví dụ một vòng lặp thông thường

```
for (j=0;j<100;j++) a[j]=j;
```

Trong ví dụ ở trên, các mã nguồn líu ríu vào với nhau, rất khó để nhận ra các khối lệnh. Bằng cách sử dụng các khoảng trắng hợp lý như sau, các khối lệnh trở nên rõ ràng, dễ đọc hơn:

```
for (j=0; j<100; j++)
   a[j] = j;
```

Về các khoảng trắng, các IDE hiện nay hỗ trợ rất tốt các tính năng "auto indenting", cách lề tự động. Lập trình viên nên tận dụng hiệu quả tính năng này, có thể thay đổi trong Cài đặt của IDE để cho phù hợp hơn với yêu cầu của dự án.

Cách lề (indentation) cũng là một dạng khoảng trắng. Đối với hầu hết các ngôn ngữ lập trình, việc cách lề giúp xác định rõ cấu trúc của chương trình (Ngoại trừ các ngôn ngữ lập trình kịch bản như Python hay Ruby thì việc cách lề là bắt buộc để xác định cấu trúc của khối lệnh). Việc cách lề không hợp lý có thể khiến cho mã nguồn bị hiểu sai về mặt cấu trúc. Như có thể thấy ở trong ví dụ ở đây, là một đoạn mã kiểm tra giá trị ngày, tháng, năm có hợp lệ hay không, trong trường hợp ngày của tháng 2 có thể là 28 hoặc 29 ngày.

```
if (month == FEB) {
   if (year % 4 == 0)
      if (day > 29)
         legal = FALSE;
   else
      if (day > 28)
         legal = FALSE;
}
```

Trong đoạn mã này, từ khóa else được đặt thẳng hàng với từ khóa if đầu tiên, nhưng thực tế đối với compiler, từ khóa else này là của if thứ hai. Do đó đoạn mã này bị sai về mặt cấu trúc. Có thể thấy hoàn toàn đoạn mã cho việc nếu year không chia hết cho 4 (không phải năm nhuận) không hề được gọi. Còn trong trường hợp năm nhuận, nếu day = 29 thì sẽ rơi vào trường hợp của else, và do đó legal = FALSE.

Đây là một ví dụ cho thấy một lỗi lập trình có thể gặp phải khi không tuân thủ các phong cách lập trình. Và việc rà soát mã nguồn để debug lỗi này cũng không hề đơn giản. Cách tốt nhất là chúng ta cần phải tuân thủ theo các quy tắc cách lề và sử dụng khối hợp lý, ví dụ:

```
if (month == FEB) {
   if (year % 4 == 0) {
      if (day > 29)
         legal = FALSE;
   }
   else {
      if (day > 28)
         legal = FALSE;
   }
}
```

Một cách đơn giản để mã nguồn có các cấu trúc dễ đọc là phân ra thành cách đoạn, đơn giản bằng cách chèn thêm một dòng trống để chia mã nguồn ra thành các thành phần nhỏ. Ví dụ như ở đây, chúng ta có thể thấy rõ, đoạn 1 dùng để khai báo biến, đoạn 2 dùng để nhập dữ liệu, đoạn 3 dùng để tính toán và đoạn 4 dùng để in thông tin ra màn hình.

```
#include <stdio.h>
#include <stdlib.h>

int main(void)

/* Read a circle's radius from stdin, and compute and write its
   diameter and circumference to stdout. Return 0 if successful. */

{
   const double PI = 3.14159;
   int radius;
   int diam;
   double circum;

   printf("Enter the circle's radius:\n");
   if (scanf("%d", &radius) != 1)
   {
```

```
        fprintf(stderr, "Error: Not a number\n");
        exit(EXIT_FAILURE);  /* or:  return EXIT_FAILURE; */
    }

    diam = 2 * radius;
    circum = PI * (double)diam;

    printf("A circle with radius %d has diameter %d\n",
           radius, diam);
    printf("and circumference %f.\n", circum);

    return 0;
}
```

b) Cấu trúc mã nguồn

*** Đối với các biểu thức**

Nói chung, chúng ta nên tránh dùng các biểu thức phức tạp, mà tìm cách đơn giản và dễ hiểu để viết biểu thức đó. Còn đối với các biểu thức có các tham số phức tạp, chúng ta có thể sắp xếp thành các cơ cấu song song để mã nguồn dễ nhìn hơn. Ví dụ, trong đoạn mã nguồn kiểm tra xem một ký tự c có phải là ký tự tương ứng với chữ cái đầu tiên của các tháng trong năm hay không, nếu chúng ta trình bày như đoạn mã sau, rất khó để có thể nhìn được cấu trúc của biểu thức.

```
if ((c == 'J') || (c == 'F') || (c ==
'M') || (c == 'A') || (c == 'S') || (c
== 'O') || (c == 'N') || (c == 'D'))
```

Thay vào đó, bằng cách tổ chức như sau, biểu thức trở nên rõ ràng hơn rất nhiều.

```
if ((c == 'J') || (c == 'F') ||
    (c == 'M') || (c == 'A') ||
    (c == 'S') || (c == 'O') ||
    (c == 'N') || (c == 'D'))
```

Một yếu tố nữa khi viết các biểu thức là dùng các dấu ngoặc đơn để tránh nhầm lẫn. Như các bạn đã biết, các toán tử có các độ ưu tiên khác nhau. Trong ví dụ về đoạn mã nguồn đọc và in các ký tự cho đến khi kết thúc tệp (EOF – EndOfFile) ở đây, chúng ta có 2 toán tử là toán tử gán = và toán tử logic !=.

```
while (c = getchar() != EOF)
   putchar(c);
```

Ở đây, toán tử logic != có độ ưu tiên cao hơn. Do đó, nếu không sử dụng ngoặc đơn để làm rõ, mã nguồn có thể bị viết sai như trong ví dụ này. Ngay cả trong trường hợp các lập trình viên sử dụng đúng thứ tự của các toán tử, việc sử dụng dấu ngoặc hợp lý cũng giúp cho mã nguồn trở nên rõ ràng hơn. Ngoài ra, cũng cần phải lưu ý là độ ưu tiên của các toán tử không phải ngôn ngữ nào cũng giống nhau. Do đó, trong nhiều trường hợp cần port mã nguồn từ ngôn ngữ này sang ngôn ngữ khác, các lập trình viên có thể bỏ qua vấn đề này và khiến cho chương trình không chạy đúng. Việc rà soát lỗi này là vô cùng khó khăn, và có thể tránh được nếu lập trình viên sử dụng hợp lý dấu ngoặc, ví dụ:

```
while ((c = getchar()) != EOF)
   putchar(c);
```

c) Đặt tên và chú thích

Về đặt tên, chúng ta có một số quy ước chung như sau:

- Dùng tên gợi nhớ, có tính miêu tả cho các biến và hàm. VD: **hovaten, CONTROL, CAPACITY**
- Dùng tên nhất quán cho các biến cục bộ. VD: **i** (không dùng **arrayIndex**) cho biến chạy vòng lặp
- Dùng chữ hoa, chữ thường nhất quán. VD: Chữ cái đầu của tên lớp thường viết hoa.
- Dùng phong cách nhất quán khi ghép từ. VD: **frontsize**, **frontSize, front_size**
- Dùng động từ cho tên hàm, danh từ cho tên biến và tên lớp. VD: **DocSoLieu(), InKetQua(), Check_Octal()**

Về các quy tắc viết chú thích, lập trình viên nên lưu ý:

- Không cần phải chú thích nếu mã nguồn đủ dễ hiểu
- Không viết chú thích chỉ để thêm thông tin, ví dụ

  ```
  i++;   /* add one to i */
  ```

- Chú thích các đoạn mã nguồn, đừng chú thích từng dòng, ví dụ

  ```
  // Sort array in ascending order
  ```

- Chú thích các dữ liệu toàn cục
- Viết chú thích tương ứng với mã nguồn

- Cập nhật lại chú thích khi mã nguồn thay đổi

Đối với các chú thích cho hàm:

- Mô tả những gì cần thiết để gọi hàm 1 cách chính xác
- Mô tả hàm làm gì, chứ không phải làm như thế nào
- Bên trong hàm, không cần chú thích khi mã nguồn đã rõ ràng.
- Mô tả đầu vào: Tham số truyền vào, đọc file gì, biến toàn cục được dùng
- Mô tả đầu ra: giá trị trả về, tham số truyền ra, ghi ra file gì, các biến toàn cục bị ảnh hưởng

Lấy ví dụ về chú thích hàm làm như thế nào thay vì chỉ ra hàm làm công việc gì.

```
/* decomment.c */
int main(void) {
/* Đọc 1 ký tự.  Dựa trên ký tự ấy và trạng thái DFA hiện thời, gọi
hàm xử lý trạng thái tương ứng.  Lặp cho đến hết tệp end-of-file. */
   …
}
```

Ở đây, chúng ta có thể sửa lại như sau

```
/* decomment.c */
int main(void) {
/* Đọc 1 chương trình C qua stdin.
   Ghi ra stdout với mỗi chú thích thay bằng 1 dấu cách.
   Trả về 0 nếu thành công, EXIT_FAILURE nếu không thành công. */
   …
}
```

8.3 Tái cấu trúc mã nguồn

8.3.1 Khái niệm

8.3.1.1 Định nghĩa

Tái cấu trúc mã nguồn (refactoring) được định nghĩa là quá trình sửa đổi mã nguồn để cải thiện tính dễ đọc, khả năng bảo trì và khả năng mở rộng mà không thay đổi những gì mà đoạn mã nguồn đó làm được. Ở đây, cần nhấn mạnh ở yếu tố "không thay đổi" ở đây, có nghĩa là không làm thay đổi hành vi bên ngoài của chương trình và không thêm vào các tính năng mới mà chỉ thay đổi cấu trúc bên trong của chương trình. Có thể hình dung một chương trình giống như một cái hộp đen, việc tái cấu trúc mã nguồn thay đổi các cơ cấu hoạt động bên trong của cái hộp đen đó, giúp cho việc bảo trì, sửa

chữa... Dễ dàng hơn, nhưng cái máy móc bên đó vẫn hoạt động đúng chức năng như thiết kế ban đầu của nó.

Nói theo một cách nào đó thì tái cấu trúc mã nguồn là biến từ một mã nguồn bẩn thành mã nguồn sạch, nhằm mục đích loại bỏ các lỗi tiềm ẩn và làm cho thiết kế trở nên đơn giản hơn.

8.3.1.2 Mục tiêu của tái cấu trúc mã nguồn

Mã nguồn "bẩn" ở đây có thể là do nhiều yếu tố. Hầu hết mọi người sẽ nghĩ ngay đến yếu tố là do lập trình viên thiếu kinh nghiệm. Điều này hoàn toàn đúng, tuy nhiên, đây không phải là lý do duy nhất. Trong các dự án thực tế, ngay cả các lập trình viên có kinh nghiệm cũng thường xuyên viết các mã nguồn kém chất lượng (ví dụ như copy paste các đoạn code...). Lý do chính là vì viết các mã nguồn "sạch" thường rất vất vả, trong khi mã nguồn lại thường xuyên phải thay đổi do các yêu cầu liên tục thay đổi. Ngay cả những mã nguồn "sạch" đã được viết cũng có thể sẽ không còn được sử dụng với yêu cầu mới của khách hàng, khiến lập trình viên sẽ có cảm giác "ngại" hơn khi phải dành thời gian và công sức để viết. Trong khi đó, nếu quản lý không sát sao, lập trình viên sẽ càng dễ có cảm giác "lười biếng" và tạo ra nhiều mã nguồn bẩn hơn. Đôi khi, việc này cũng có thể xảy ra khi nhóm phát triển phải "chạy deadline", không có nhiều thời gian để chăm chút cho mã nguồn sạch. Hoặc một trường hợp nữa là khi các quyết định có tính chất tạm thời. Ví dụ nhóm phát triển không chắc chắn là phải chọn phương án nào nên làm nhanh vài phương án tạm thời. Sau đó, những phương án không được chọn sẽ bị bỏ đi, nhưng phương án được chon sẽ mang tính tạm thời. Ngoài ra, việc khách hàng thay đổi các yêu cầu, như đã tìm hiểu ở các bài trước, cũng là một yếu tố dẫn đến việc lập trình viên phải viết mã nguồn "bẩn" hơn.

Vậy mục tiêu của tái cấu trúc mã nguồn là biến những mã nguồn như thế này trở thành các mã nguồn sạch: Mã dễ đọc, dễ hiểu, dễ bảo trì; giúp quá trình phát triển phần mềm dễ dự đoán được; và tăng chất lượng sản phẩm phần mềm.

Khi tái cấu trúc mã nguồn, có một điểm cần phải chú ý: **Tái cấu trúc có thể tạo ra vấn đề mới, vì bất cứ khi nào bạn sửa đổi phần mềm, bạn đều có thể tạo ra lỗi!**

Nói như vậy có phải nghĩa là việc refactor làm tăng rủi ro cho quá trình phát triển phần mềm? Chưa kể đến chuyện việc refactor tốn nhiều thời gian và công sức, tương ứng với việc tốn chi phí (tiền lương, tiền điện nước…), trong khi nhìn từ bên ngoài vào thì các chức năng hoàn toàn không khác gì? Ngoài ra, khi chúng ta sửa bất kỳ cái gì trong mã nguồn thì chúng ta cũng phải thực hiện lại các kiểm thử để đảm bảo là chương trình của chúng ta không tạo ra lỗi mới, cũng là một tác vụ tốn kém?

Đúng là như vậy. Các bạn có thể đặt câu hỏi, "vậy refactor để làm gì?".

Lý do ở đây là vì refactor đem lại rất nhiều ưu điểm lớn, như là: Mã khó hiểu được cấu trúc lại **đơn giản, dễ hiểu** hơn. Nhờ đó có thể dễ phát hiện ra các lỗi tiềm ẩn hơn mà nếu không refactor có thể không bao giờ phát hiện ra được. Refactor cũng sẽ giúp mã nguồn phù hợp hơn với thiết kế. Nhờ đó, giai đoạn refactor ban đầu có thể mất một chút thời gian, nhưng sau khi mã nguồn đã được tái cấu trúc, tốc độ lập trình có thể được tăng lên đáng kể, kèm với đó là cải thiện đáng kể khả năng bảo trì.

Việc này là bắt buộc do yêu cầu luôn **thay đổi**. Trong một số trường hợp, nếu không tái cấu trúc mã nguồn thì việc nâng cấp, bảo trì mã nguồn trở nên vô cùng khó khăn nếu không muốn nói là không thể.

8.3.1.3 Khi nào cần tái cấu trúc mã nguồn

Một là khi cần thêm 1 tính năng mới vào một code sẵn có (vd. do người khác phát triển). Refactoring lại code để giúp lập trình viên dễ hiểu code hơn. Thứ hai là khi sửa lỗi (debug). Bug thường tập trung ở những chỗ phức tạp, khó tìm ra trong mã nguồn. Đôi khi chỉ cần làm sạch mã là các bug sẽ "tự xuất hiện". Trường hợp thứ ba là trong quá trình rà soát mã nguồn. Đây là cơ hội cuối để làm sạch mã trước khi được public cho toàn bộ đội phát triển.

8.3.1.4 Checklist của tái cấu trúc mã nguồn

Sau quá trình tái cấu trúc mã nguồn, cần phải bảo đảm các yêu cầu sau được thỏa mãn:

- Kiểm tra xem code đã **dễ đọc, dễ hiểu** hơn chưa, các phương thức quá dài đã được tách nhỏ hơn chưa?
- Trong quá trình refactor, đảm bảo **không có chức năng mới** nào thêm vào
- Tất cả các **kiểm thử đều phải pass** sau khi refactor.

8.3.2 Kỹ thuật tái cấu trúc mã nguồn

8.3.2.1 Xác định các mã nguồn "bốc mùi"

Kỹ thuật đầu tiên và cũng là quan trọng nhất là kỹ thuật xác định các mã nguồn "bốc mùi" – code smell (đôi khi còn gọi là mã nguồn "thối" hay mã nguồn "xấu"). Đây là khái niệm được Martin Fowler đưa ra, ám chỉ những mã nguồn có khả năng là mã bẩn, tương ứng với các vấn đề tiềm ẩn ở sâu bên trong hệ thống. Thực ra các code smell không nhất thiết lúc nào cũng là mã nguồn "xấu", mà chỉ là các mã nguồn mà các nhà phát triển có thể "đánh hơi" thấy khi thấy có khả năng có vấn đề tiềm ẩn. Lấy ví dụ như khi nhìn thấy một phương thức dài, người lập trình viên có thể nghi ngờ đoạn mã có vấn đề, tuy nhiên đôi khi một số phương thức dài cũng không có vấn đề gì cả. Cần phải nhớ một điều là: bản thân các code smell không phải là vấn đề, mà thường đó chỉ là các chỉ báo của vấn đề tiềm ẩn bên trong.

Tương tự như design pattern là các "khuôn mẫu thiết kế", chúng ta có thể coi các code smell như các pattern tệ mà các đoạn mã nguồn "bẩn" hay áp dụng trong quá trình xây dựng phần mềm. Những code smell, khác với các vấn đề thực sự, thường có thể phát hiện ra một cách dễ dàng và nhanh chóng hơn nhiều.

Có thể tìm hiểu một số loại code smell thông dụng:

- *Mã trùng lặp,* có thể do lập trình viên lười nên Copy & Paste mà không cấu trúc lại chương trình. Mã nguồn như vậy sẽ trở nên tệ nếu sửa đổi một phiên bản của mã trùng lặp mà không sửa đổi các đoạn mã khác thì (có thể) tạo ra lỗi!
- *Phương thức dài.* Các phương thức dài thường khó hiểu hơn và do đó khi rà soát lỗi có thể bỏ sót một số lỗi tiềm ẩn. Một số lập trình viên có thói quen không muốn viết các phương thức ngắn để tối ưu về mặt hiệu suất. Tuy nhiên hiện nay máy tính đã đủ mạnh để không cần quá quan tâm đến vấn đề hiệu suất khi sử dụng các phương thức ngắn hơn.
- *Lớp lớn*, là lớp hướng đến làm quá nhiều việc. Điều này có thể làm giảm kết dính (cohesion).
- *Quá nhiều tham số* trong phương thức. Việc này cũng có thể khiến cho phương thức trở nên khó hiểu, giảm tính nhất quán
- *Thay đổi khác biệt* (*Divergent Change*). Triệu chứng của code smell này là khi có một thay đổi loại này thì phải thay đổi một tập hợp con các phương thức; thay

đổi một loại khác lại phải thay đổi một tập hợp con khác. (Mối quan hệ many-to-one: Nhiều thay đổi tác động đến 1 lớp). Ngược lại với code smell này là *Shotgun Surgery*, một thay đổi tác động đến nhiều lớp.

- *Lớp lười (Lazy Class)*: Lớp không còn thực hiện nhiệm vụ ban đầu, có thể sau rất nhiều lần thay đổi mã nguồn.
- *Tổng quát hóa suy đoán (Speculative Generality):* Các tính năng được xây dựng để sử dụng "lúc nào đó", nhưng mãi mãi không bao giờ dùng đến.
- *Trường tạm thời (Temporary Field)*

Đây là một số ví dụ code smell tiêu biểu. Còn nhiều code smell nữa, các bạn có thể tự tìm hiểu trên mạng, ví dụ như tại *https://refactoring.guru/refactoring/smells*

8.3.2.2 Một số kỹ thuật refactor

Khi đã phát hiện ra các code smell, chúng ta sẽ tìm hiểu kỹ hơn ở trong mã nguồn để xem các code smell đó có thực sự là các vấn đề không. Nếu đúng đó là các vấn đề, tùy vào từng loại, chúng ta có thể áp dụng một số kỹ thuật refactor để khắc phục, ví dụ như

- Di chuyển một phương thức từ lớp này sang lớp khác
- Thêm/Xóa các tham số
- Chuyển từ delegation sang kế thừa và ngược lại
- …

Tương tự như các code smell, các bạn cũng có thể tìm hiểu thêm về các kỹ thuật này tại các nguồn khác trên mạng, ví dụ *https://refactoring.guru/refactoring/techniques*

8.3.3 Rà soát mã nguồn

Như đã nói ở trên, quá trình rà soát mã nguồn là giai đoạn rất thích hợp để thực hiện công việc refactoring.

Rà soát mã nguồn là quá trình mà mã nguồn (có thể cả thiết kế) được xem xét lại bởi một người (hoặc một nhóm người) **khác** thay vì chính tác giả mã nguồn

Mục đích của quá trình này là Xem xét một số tính chất của mã nguồn như: Tính bảo trì được – Maintainability; Tính tái sử dụng được – Reusability; và Tính dễ dàng mở rộng được – Extensibility. Bên cạnh đó, quá trình này cũng giúp xem xét độ tin cậy và tính tối ưu của thiết kế, tài liệu...

Quá trình này đem lại các lợi ích lớn như: Mang đến một **góc nhìn khác** với người viết mã nguồn, do lỗi thường dễ dàng được phát hiện bởi người khác hơn là chính tác giả. Mục đích thứ hai là **chia sẻ kiến thức**, ví dụ như các thiết kế và ý tưởng. Mục đích thứ 3 là **Phát hiện lỗi** sớm, nhờ đó hạn chế việc tốn công sức phải làm lại. Có một số vấn đề chỉ có thể phát hiện được trong quá trình rà soát mã nguồn thay vì kiểm thử.

Có hai kỹ thuật rà soát mã nguồn chính.

- Quy trình chính thống (Inspection / Formal technical review). Người tham gia là các QA lead, QA tester, developer, project manager, team leader. Với quy trình này, cần chuẩn bị trước một review checklist. Sẽ tổ chức một Formal meeting, lead bởi một người không phải tác giả. Sau đó, sẽ có một formal follow-up: Gửi kết quả review cho tất cả các bên liên quan
- Walkthrough, quy trình không chính thống. Với quy trình này thì không cần chuẩn bị trước gì. Tác giả là người lead cuộc họp. Kỹ thuật này có chi phí thấp, nên thường mang tính chất đào tạo là chính.

8.4 Ví dụ và bài tập

8.4.1 Bài 8.1

Viết chú thích khai báo header mô tả các thông tin tại đầu tệp mã nguồn. Các thông tin cần khai báo bao gồm:

- Tên người lập trình
- Ngày
- Tên của dự án đã lưu
- Tên lớp
- Tên của bất kỳ ai đã giúp bạn
- Mô tả ngắn gọn về những gì chương trình thực hiện

Hướng dẫn:

Người học có thể tham khảo chú thích sau

Sample Header

```
/**
 * MyClass <br>
 *
 * This class is merely for illustrative purposes. <br>
 *
 * Revision History:<br>
 * 1.1 - Added javadoc headers <br>
 * 1.0 - Original release<br>
 * Class: 111678, Teacher's Name:...
 * @author Ng.Van A
 * @version 1.1, 31/07/2020
 */
public class MyClass {
    . . .
}
```

8.4.2 Bài 8.2

Viết chú thích tài liệu cho các phương thức

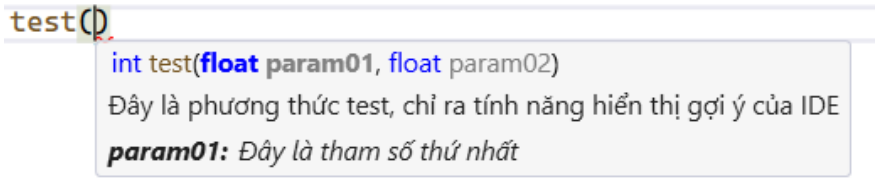
Hướng dẫn

Trước phương thức, khi gõ `///` hoặc `/**`, IDE sẽ tự động sinh ra các gợi ý để viết chú thích

Ví dụ:

```
/// <summary>
///
/// </summary>
/// <param name="param01"></param>
/// <param name="param02"></param>
/// <returns></returns>
int test(float param01, float param02) {
}
```

Bên cạnh việc sinh ra các tài liệu chương trình, các chú thích này cũng hỗ trợ cho việc gợi ý viết mã khi sử dụng phương thức



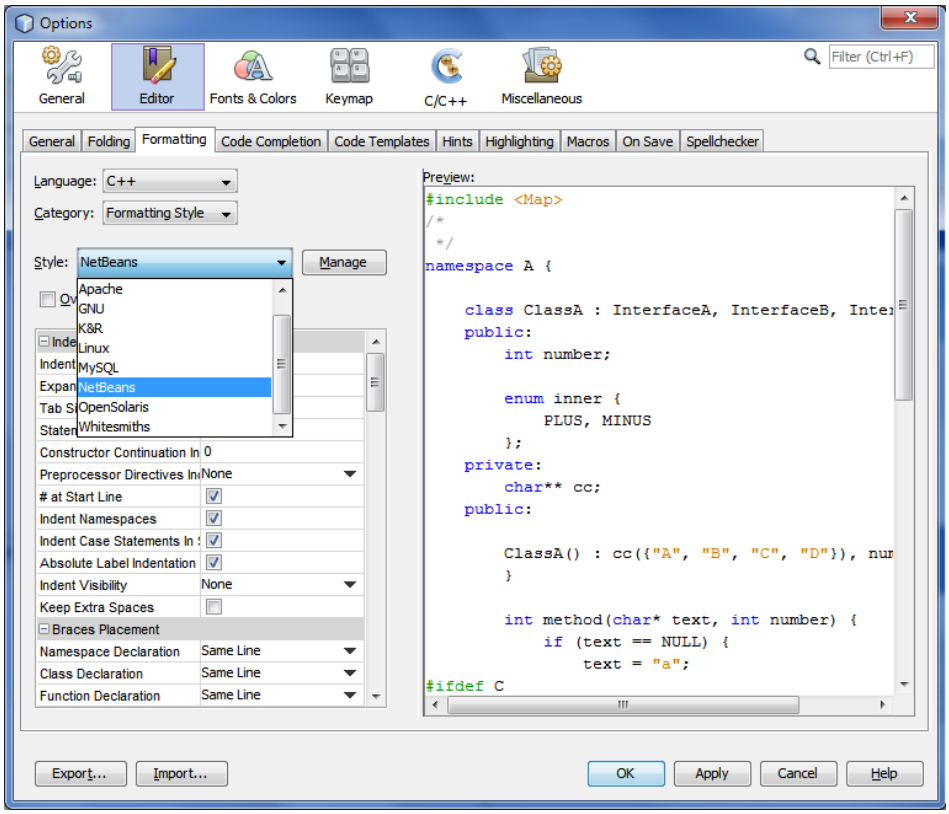
Hình 8-5 Gợi ý viết mã trong IDE

8.4.3 Bài 8.3

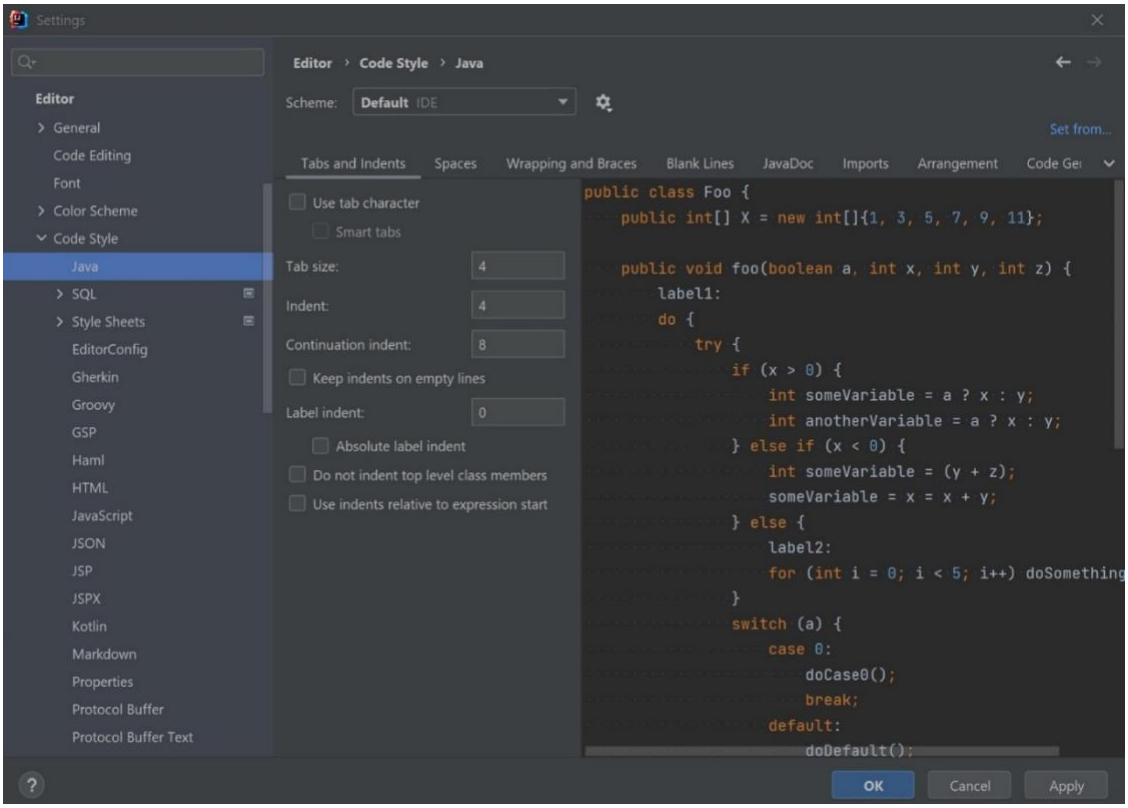
Tìm hiểu các công cụ hỗ trợ phong cách lập trình trong các IDE người học sử dụng

Hướng dẫn:

Các IDE hiện đại đều có các công cụ hỗ trợ phong cách lập trình. Người học tìm trong Settings của các IDE, ví dụ như Hình 8.2 và 8.3



Hình 8-6 Lựa chọn các phong cách lập trình C++ trong NetBeans



Hình 8-7 Thiết lập các phong cách lập trình trong IntelliJ IDEA

8.4.4 Bài 8.4

Sử dụng Null Object để thay thế đoạn mã nguồn sau

```
Customer c = findCustomer(...);
...
   if (customer == null) {
      name = "occupant";
   }
   else {
      name = customer.getName();
   }
if (customer == null) {
...
```

Hướng dẫn:

Có rất nhiều mã nguồn kiểm tra giá trị null. Do đó, có thể thay thế giá trị null bằng một đối tượng NullCustomer

```
public class NullCustomer extends Customer {
```

```
   public String getName() {
      return "occupant";
   }
...
Customer              c              =              findCustomer()
name = c.getName()
```

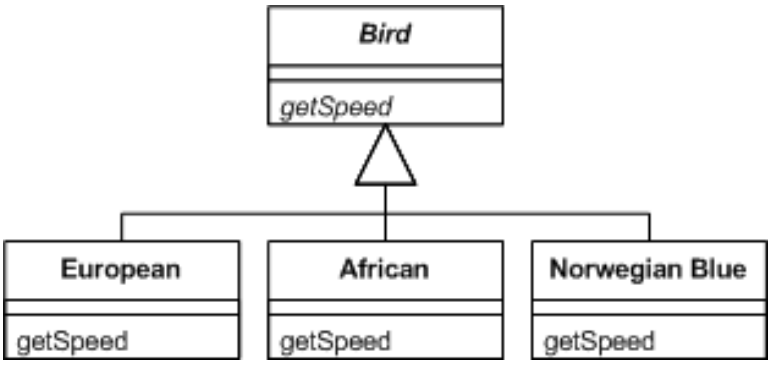
8.4.5 Bài 8.5

Tái cấu trúc đoạn mã nguồn sau bằng cách thay thế các kiểm thử điều kiện bằng đa hình.

```
double                          getSpeed()                          {
    switch                          (_type)                         {
      case                                                      EUROPEAN:
           return                                          getBaseSpeed();
      case                                                       AFRICAN:
           return getBaseSpeed() -
                    getLoadFactor()*numberOfCoconuts;
      case                                                 NORWEGIAN_BLUE:
             return (_isNailed) ? 0 :
                    getBaseSpeed(_voltage);
    }
    throw new RuntimeException ("Should be unreachable");
}
```

Hướng dẫn:

Di chuyển từng nhánh của điều kiện sang một phương thức ghi đè trong mỗi lớp con. Thay đổi phương thức gốc thành phương thức trừu tượng.



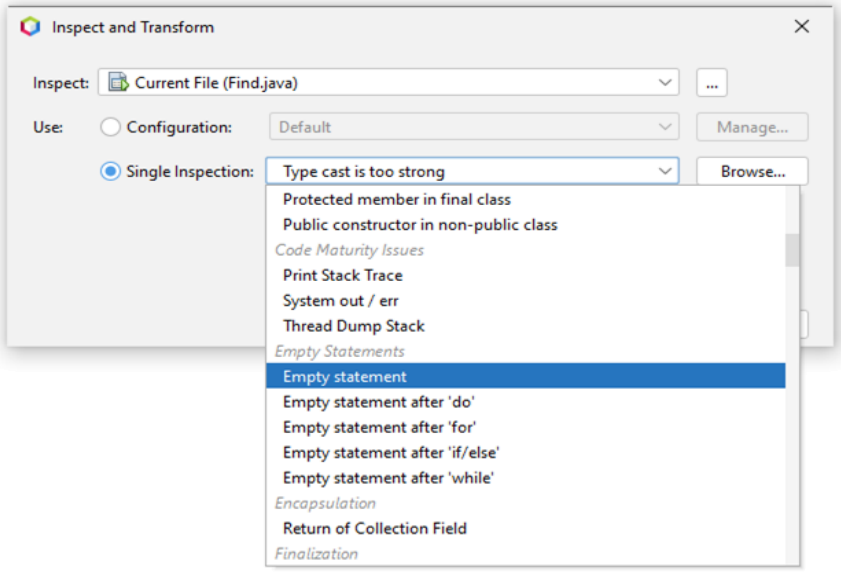
Hình 8-8 Biểu đồ lớp sau khi refactor

8.4.6 Bài 8.6

Tìm hiểu các công cụ hỗ trợ refactor của IDE người học đang sử dụng

Hướng dẫn:

Ví dụ trong NetBeans: Nhấp phải vào một file bất kỳ, sau đó chọn *Refactor* | *Inspect and Transform*. Xem Hình 8.5



- *Hình 8.5. Công cụ Inspect and Transform trong NetBeans*

Chương 9 ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM

Nội dung:

- Các khái niệm cơ bản về chất lượng phần mềm, mô hình chữ V, và kiểm thử phần mềm;
- Khái niệm về kiểm thử hộp trắng và phương pháp kiểm thử luồng điều kiển;
- Khái niệm về kiểm thử hộp đen và phương pháp kiểm thử với phân lớp tương đương;
- Quy trình đảm bảo chất lượng phần mềm;
- Bảo trì phần mềm.

9.1 Các khái niệm cơ bản về đảm bảo chất lượng phần mềm

Các khái niệm sau đây được sử dụng xuyên suốt trong chương học, hiểu rõ các khái niệm này đóng vai trò quan trọng trong việc hiểu được chất lượng và đảm bảo chất lượng quá trình xây dựng phần mềm.

9.1.1 Chất lượng phần mềm

Có thể nhìn nhận chất lượng phần mềm dưới nhiều góc độ.

- Thứ nhất là quan điểm người dùng: Người dùng bao giờ cũng có một cái nhìn rất cụ thể về chất lượng của sản phẩm mà họ muốn. Như vậy, một phần mềm có chất lượng là khi đạt được các mục đích của nó. Đánh giá chất lượng theo quan điểm của người dùng thì chúng ta cần phải trả lời câu hỏi: phần mềm có thoả mãn các yêu cầu và mong đợi của người dùng hay không?
- Thứ hai là quan điểm sản xuất: chất lượng phần mềm được hiểu là những gì tương thích với đặc tả/mô tả của sản phẩm.
- Thứ ba là quan điểm sản phẩm: chất lượng được xem là gắn liền với các đặc tính cố hữu hay là vốn có của sản phẩm.
- Cuối cùng, quan điểm về giá trị kinh tế: chất lượng phụ thuộc vào số tiền mà khách hàng có thể bỏ ra để trả cho sản phẩm của mình.

Một cách chung nhất thì chất lượng của một sản phẩm có nghĩa là phải đáp ứng được các yêu cầu kỹ thuật của nó, có nghĩa là sản phẩm đạt được tất cả các yêu cầu đề ra trong đặc tả.

Theo định nghĩa của IEEE thì chất lượng phần mềm gồm hai ý:

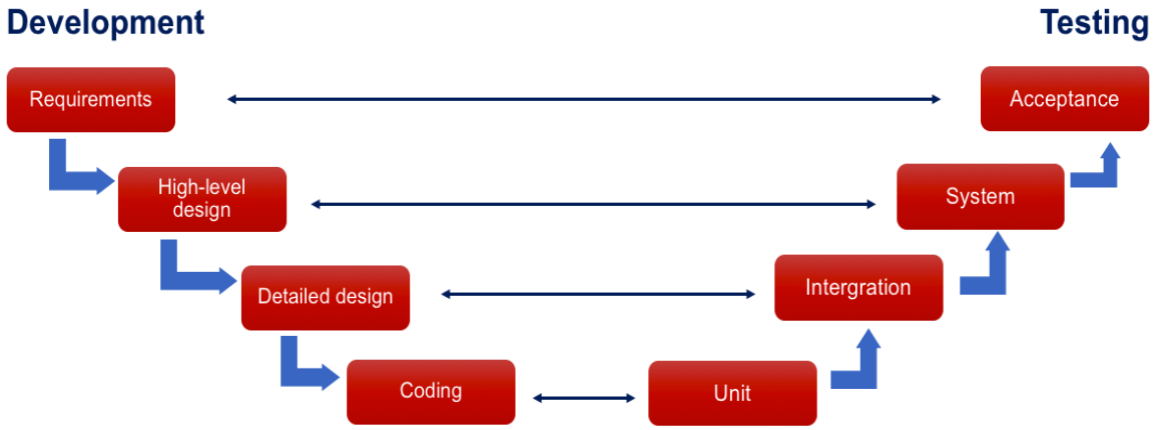
- Thứ nhất, đó là mức độ một hệ thống hay một thành phần hay một tiến trình đạt được các yêu cầu đặt ra;
- Thứ hai, là mức độ một hệ thống hay một thành phần hay một tiến trình đạt được những mong đợi của khách hàng hay người dùng của hệ thống đó.

Như vậy chất lượng của phần mềm theo định nghĩa của IEEE được đánh giá từ hai quan điểm: quan điểm của người phát triển phần mềm (internal view) và quan điểm của người dùng (external view).

9.1.2 Mô hình chữ V

Mô hình chữ V là mô hình cho hoạt động phát triển và đánh giá chất lượng phần mềm. Mô hình này được biết đến với tên gọi là mô hình xác minh (Verification Model) hay là mô hình xác thực (Validation Model). Ở đây ta có hai khái niệm:

- *Verification:* được gọi là xác minh xem hệ thống có được xây dựng đúng hay không (*verify if the system is built right*). Đây là hoạt động đánh giá phần mềm bằng cách xác định xem hệ thống có thoả mãn các yêu cầu nêu ra ban đầu hay không (từ góc nhìn của người phát triển).
- *Validation:* gọi là kiểm tra xem ta đã xây dựng đúng hệ thống hay không (*validate if we built a right system*). Đây là hoạt động kiểm tra xem sản phẩm phần mềm có thoả mãn các yêu cầu của người dùng hay không (từ góc nhìn của người dùng).



Hình 9-1: Mô hình chữ V

Hình vẽ trên là mô hình chữ V. Theo đó, việc kiểm thử sẽ được thực hiện trên từng giai đoạn một cách tuần tự để đảm bảo chất lượng của phần mềm. Quá trình phát triển bao gồm các pha như lấy yêu cầu, thiết kế, thiết kế chi tiết và lập trình. Tương ứng với các bước này thì việc kiểm thử được thực hiện ở các mức khác nhau, bao gồm:

- Kiểm thử đơn vị
- Kiểm thử tích hợp
- Kiểm thử hệ thống
- Kiểm thử xác nhận

9.1.3 Kiểm thử phần mềm

a. Các thuật ngữ về kiểm thử:

- Kiểm thử (software testing): là một quy trình phân tích và đánh giá một thành phần phần mềm (software item) để phát hiện sự khác biệt giữa thành phần đó với các yêu cầu đặt ra (gọi là lỗi).
- Ca kiểm thử (*test case*): là một cặp <*input, expected outcome*>. Khi thực hiện một ca kiểm thử người ta sẽ chạy chương trình với dữ liệu đầu vào và kiểm tra kết quả trả về của hệ thống. Theo đó:
  - Nếu sau khi thực hiện một ca kiểm thử thu được kết quả trả về tương ứng với kết quả mong đợi thì được gọi là đạt (*passed test*)
  - Nếu ngược lại gọi là không đạt (*failed test*).
- Tập các ca kiểm thử (*test suite*): là một tập hợp các test case được xây dựng để kiểm thử cho 1 chức năng hoặc 1 nhóm chức năng của phần mềm

b. Một số khó khăn trong kiểm thử phần mềm có thể gặp phải là:

- Chúng ta mong muốn nâng cao chất lượng phần mềm nhưng không thể vượt quá chất lượng khi thiết kế vì chỉ phát hiện các lỗi tiềm tàng và sửa chúng;
- Việc phát hiện lỗi bị hạn chế do được thực hiện thủ công là chính;
- Thông thường người kiểm thử dễ bị ảnh hưởng về mặt tâm lý trong quá trình thực hiện;
- Cuối cùng, rất khó đảm bảo tính đầy đủ của việc kiểm thử.

c. Một số lưu ý khi kiểm thử:

- Chất lượng do khâu thiết kế quyết định là chủ yếu, chứ không phải khâu kiểm thử;
- Tính dễ kiểm thử phụ thuộc vào cấu trúc của chương trình;
- Người kiểm thử và người phát triển nên là hai đối tượng khác nhau;
- Nếu dữ liệu thử cho kết quả bình thường thì nó không mang lại nhiều ý nghĩa, như vậy cần có những dữ liệu kiểm thử để phát hiện ra lỗi;
- Khi thiết kế trường hợp thử, không chỉ cần đến dữ liệu đầu vào, mà phải thiết kế trước cả dữ liệu kết quả sẽ có;
- Khi phát sinh thêm các trường hợp thử thì nên thử lại những trường hợp kiểm thử trước đó để tránh ảnh hưởng lan truyền.

d. Một số kĩ thuật kiểm thử bao gồm:

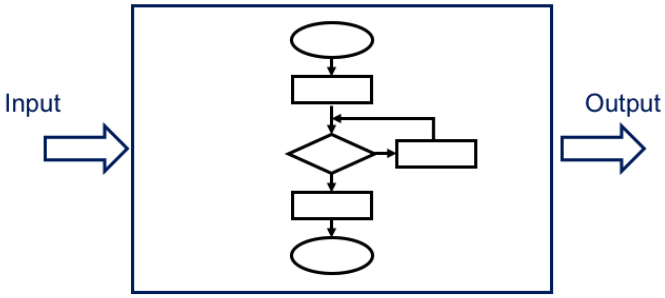
Kiểm thử hộp đen – black box testing và kiểm thử hộp trắng – white box testing. Cả hai kỹ thuật này đều yêu cầu phải thực thi chương trình để xem xét kết quả. Yêu cầu cần thiết kế các ca kiểm thử để có thể phát hiện nhiều nhất các lỗi có thể trong phần mềm. Việc sử dụng kỹ thuật nào tùy thuộc vào các giai đoạn (các mức kiểm thử khác nhau) trong quá trình phát triển phần mềm.

9.2 Phương pháp kiểm thử hộp trắng

9.2.1 Khái niệm kiểm thử hộp trắng

Kiểm thử hộp trắng là chiến lược kiểm thử dựa trên cấu trúc chức năng (gọi là structural testing) nhằm kiểm tra về mặt thuật toán và cấu trúc bên trong của sản phẩm phần mềm.

Kiểm thử hộp trắng thường được thực hiện bởi lập trình viên và có thể được triển khai bởi các kiểm thử viên chủ yếu ở giai đoạn kiểm thử đơn vị và kiểm thử tích hợp.



Hình 9-2: Kiểm thử hộp trắng

Một số phương pháp thiết kế ca kiểm thử với kiểm thử hộp trắng bao gồm:

- Kiểm thử luồng điều khiển: gồm các kĩ thuật phủ tất cả đường dẫn, bao phủ lệnh, bao phủ nhánh;
- Kiểm thử luồng dữ liệu: gồm kĩ thuật phủ tất cả các điểm định nghĩa của biến, bao phủ tất cả các điểm sử dụng của biến.

9.2.2 Kiểm thử luồng điều khiển

Đồ thị luồng điều khiển (Control Flow Graph - CFG): Đồ thị luồng điều khiển thể hiện tập hợp tất cả các đường dẫn từ điểm bắt đầu tới điểm kết thúc của chương trình được biểu diễn bằng đồ thị. Luồng điều khiển biểu diễn đồ thị cấu trúc của một đơn vị chương trình.

Trong đồ thị luồng điều khiển, một đường dẫn (path) của chương trình là một chuỗi các câu lệnh (statement) từ điểm bắt đầu cho đến điểm kết thúc của chương trình. Với dữ liệu đầu vào khác nhau, chương trình có thể thực thi theo các đường dẫn khác nhau. Mục đích của việc thiết kế các ca kiểm thử trong kiểm thử cấu trúc là tìm ra những đường dẫn (path) có lỗi hay xảy ra hoặc tiềm ẩn nhiều lỗi nhất có thể.

Các tiêu chí lựa chọn đường dẫn chương trình bao gồm:

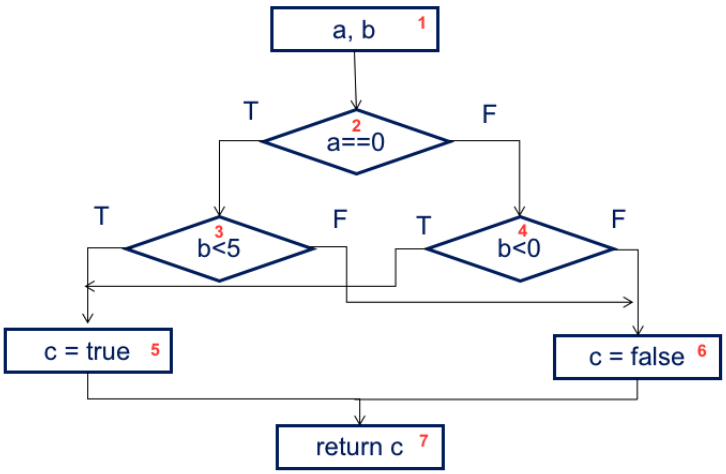
- Bao phủ toàn bộ các đường dẫn (vét cạn): Nếu chương trình có 10 lệnh rẽ nhánh thi số đường dẫn tối đa là $2^{10} = 1024$ đường dẫn nên có 1024 test case;
- Bao phủ lệnh: Đảm bảo mọi câu lệnh trong chương trình đều được kiểm tra;
- Bao phủ nhánh: Đảm bảo mọi nhánh rẽ true/false ở các câu lệnh điều kiện trong chương trình đều được kiểm thử.

Ví dụ: Cho đoạn chương trình như trong hình 9.3. Đoạn chương trình này gồm hai tham số đầu vào là *a* và *b*. Chương trình sẽ kiểm tra các giá trị của các tham số này để trả về kết quả của các biểu thức điều kiện tương ứng.

```
bool foo(int a, int b) {
 bool c;
 if (a==0)
  c = b < 5;
else
  c = b < 0;
return c;
}
```

Hình 9-3: Chương trình ví dụ kiểm thử hộp trắng

Ở dạng bài toán này, bước đầu tiên, ta cần vẽ sơ luồng điều khiển cho đoạn chương trình trên. Kết quả cho đoạn chương trình trên được thể hiện trong hình vẽ 9.4. Các node 1, 5 và 6 tương ứng với việc xác lập các giá trị cho các biến. Còn các node 2, 3 và 4 tương ứng với các điều kiện cho nhánh rẽ.



Hình 9-4: Sơ đồ luồng điều khiển tương ứng với chương trình ví dụ

a. Bao phủ đường dẫn: Ta cần lựa chọn tất cả các đường dẫn (path) có thể có trong chương trình. Mỗi path sẽ tương ứng với 1 test case. Ở đây, ta có 4 đường dẫn qua các node, bao gồm:

- 1-2-3-5-7
- 1-2-3-6-7
- 1-2-4-5-7

- 1-2-4-6-7

Với tiêu chí phủ đường dẫn, ta có thể lập bảng các giá trị cho các biểu thức điều kiện tạo ra các nhánh rẽ (kết quả được chỉ ra trong bảng 9.1). Ở đây có hai trường hợp ở hàng 3 & 6, các điều kiện cho giá trị của b mâu thuẫn nhau nên chúng không thể xảy ra.

Bảng 9-1: Giá trị của các biểu thức

	a==0	b<5	b<0	return
1	T	T	T	T
2	T	T	F	T
3	T	F	T	-
4	T	F	F	F
5	F	T	T	T
6	F	F	T	-
7	F	T	F	F
8	F	F	F	F

Ngoài ra, ta cũng có thể gom nhóm các trường hợp (chỉ ra ở các hàng) mà kết quả trả về không phụ thuộc vào giá trị của biểu thức logic như ở trường hợp 1&2, 7&8 vì các giá trị ở biểu thức điều kiện (T hoặc F) đều cho kết quả cuối cùng giống nhau. Từ đó, ta có thể tổng hợp thành 4 trường hợp và thu được 4 test case như sau:

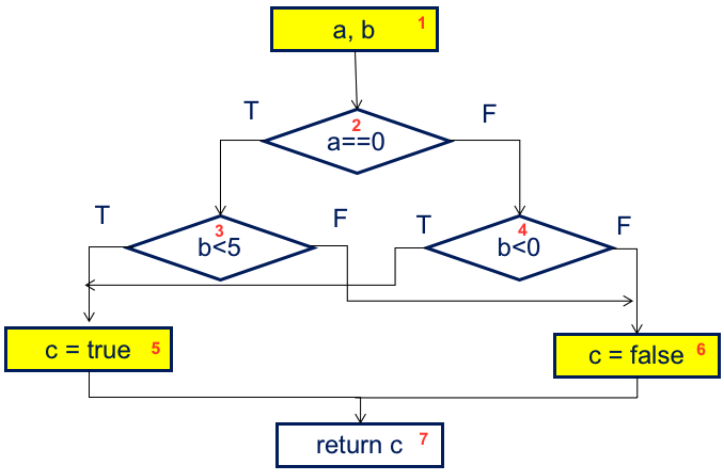
1. <T,T,*> 1-2(T)-3(T)-5-7 => TC: foo(0,3) => true
2. <T,F,F> 1-2(T)-3(F)-6-7 => TC: foo(0,6) => false
3. <F,T,T> 1-2(F)-4(T)-5-7 => TC: foo(1,-2) => true
4. <F,*,F> 1-2(F)-4(F)-6-7 => TC: foo(1,3) => false

b. Bao phủ lệnh: Mục tiêu của kĩ thuật bao phủ lệnh là đảm bảo mỗi câu lệnh đều được thực thi ít nhất 1 lần. Do đó, trong các đường dẫn của chương trình thì mọi đỉnh của đồ thị luồng điều khiển đều được xuất hiện ít nhất 1 lần.

Ta xem xét đoạn chương trình và đồ thị luồng điều khiển đã cho ở ví dụ trước (được thể hiện cụ thể hơn trong hình 9.5). Trong đoạn chương trình này có 2 câu lệnh gán giá trị cho biến *c* được biểu diễn ở đỉnh 5 và 6 trên đồ thị. Bao phủ lệnh có nghĩa là phải thiết kế các test case để đảm bảo cho hai câu lệnh trên được thực thi. Ở đây ta có thể xác định 2 test case với các giá trị tương ứng của *a* và *b* như trên màn hình.

1. TC foo(0,3) => true
2. TC foo(1,3) => false

```
bool foo(int a, int b) {
 bool c;
 if (a==0)
  c = b < 5;
else
  c = b < 0;
return c;
}
```



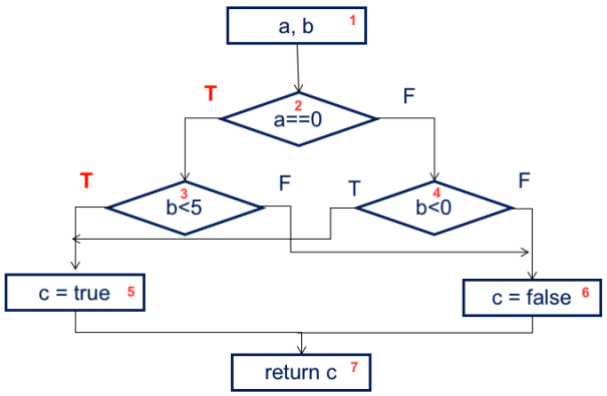
Hình 9-5: Chương trình và đồ thị luồng điều khiển cho bao phủ lệnh

c. Bao phủ nhánh: Ở kĩ thuật này, mục tiêu thiết kế các test case là đảm bảo cho mỗi điểm quyết định trên đồ thị (node tại đó có nhánh rẽ) đều được kiểm thử ít nhất 1 lần với nhánh tương ứng với giá trị điều kiện là true và 1 lần với nhánh có giá trị điều kiện là false.

Xem xét lại ví dụ trước với kĩ thuật bao phủ nhánh.

- Trường hợp 1: điều kiện a==0 là đúng, xét tiếp điều kiện điều kiện b < 5 là đúng, lấy các giá trị cho a & b, ta có test case: foo(0,3) => true.

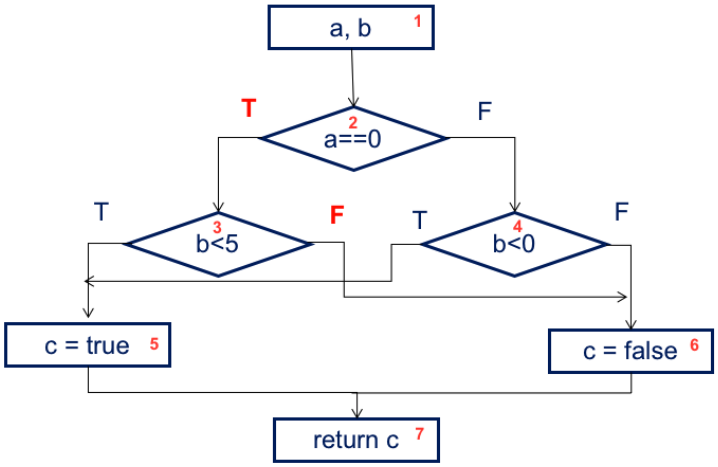
```
bool foo(int a, int b) {
 bool c;
 if (a==0)        T
  c = b < 5;      T
 else
  c = b < 0;
return c;
}
        TC: foo(0,3) => true
```



Hình 9-6: Trường hợp a==0 (đúng) và b<5 (đúng)

- Trường hợp 2: điều kiện a==0 là đúng, và xét điều kiện điều kiện b < 5 là sai, tương tự lấy các giá trị tương ứng cho a & b, ta có test case: foo(0,6) => false.

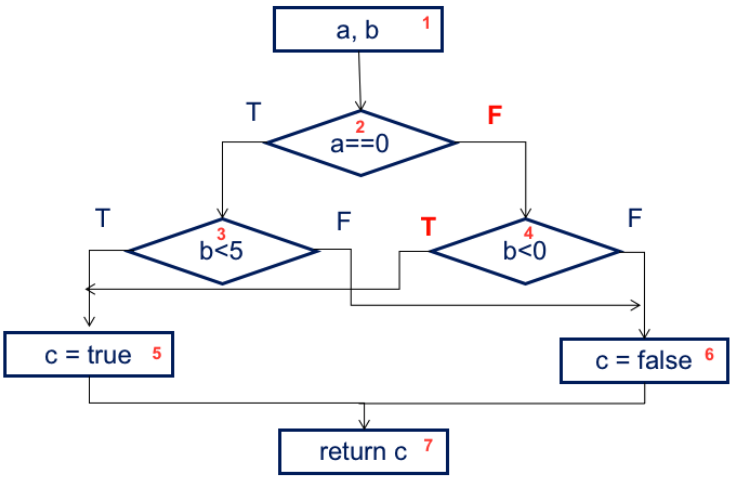
```
bool foo(int a, int b) {
 bool c;
 if (a==0)
  c = b < 5;
 else
  c = b < 0;
return c;
}
```
TC: foo(0,6) => false



Hình 9-7: a==0 (đúng) và b<5 (sai)

- Trường hợp 3: điều kiện a==0 là sai, và xét điều kiện điều kiện b < 0 là đúng ta có test case: foo(1,-2) => true;

```
bool foo(int a, int b) {
 bool c;
 if (a==0)
  c = b < 5;
 else
  c = b < 0;
return c;
}
```
TC: foo(1,-2) => true



Hình 9-8: a==0 (đúng) và b<0 (đúng)

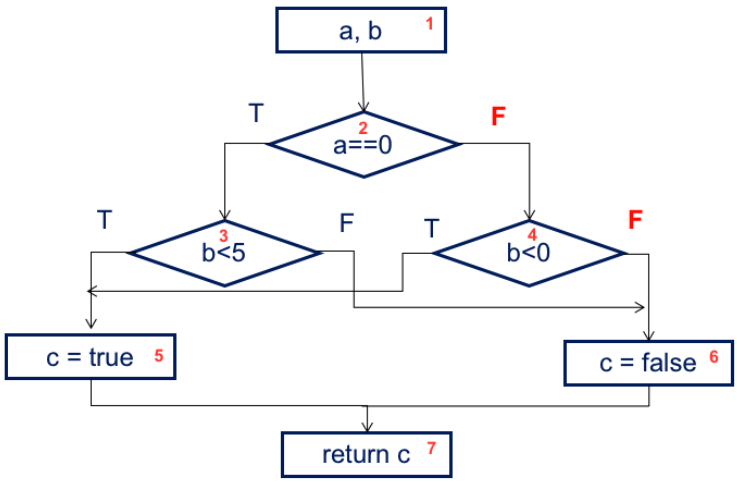
- Trường hợp 4: điều kiện a==0 là sai, và xét điều kiện điều kiện b < 0 là sai ta có test case: foo(1,3) => true.

```
bool foo(int a, int b) {
 bool c;
 if (a==0)
  c = b < 5;
 else
  c = b < 0;
return c;
}
```

F

F

TC: foo(1,3) => false



Hình 9-9: a==0 (sai) và b<0 (sai)

d. So sánh các tiêu chí: Thứ nhất, tiêu chí mạnh nhất sẽ là bao phủ mọi đường dẫn. Trong khi đó tiêu chí yếu nhất là bao phủ mọi câu lệnh với các câu lệnh rẽ nhánh có thể chỉ cần kiểm thử 1 nhánh true hoặc false. Do đó, thường được kết hợp thêm tiêu chí bao phủ điểm quyết định.

9.3 Bài tập kiểm thử hộp trắng với luồng điều khiển

Một bài toán về kiểm thử hộp trắng cần đến mã nguồn của chương trình. Yêu cầu cần thiết kế các ca kiểm thử theo kĩ thuật kiểm thử luồng điều khiển, đảm bảo bao phủ đường dẫn (phủ nhánh, phủ câu lệnh).

Cách thực hiện:

- Bước 1: Vẽ đồ thị CFG tương ứng với đoạn chương trình đã cho.
- Bước 2: Xác định các đường dẫn (nhánh rẽ/câu lệnh) tương ứng theo yêu cầu của đề bài.
- Bước 3: Xác định các giá trị đầu vào tương ứng theo các đường dẫn (nhánh rẽ/câu lệnh) ở bước 2.
- Bước 4: Xác định các test case tương ứng với các giá trị ở bước 3.

Ví dụ:

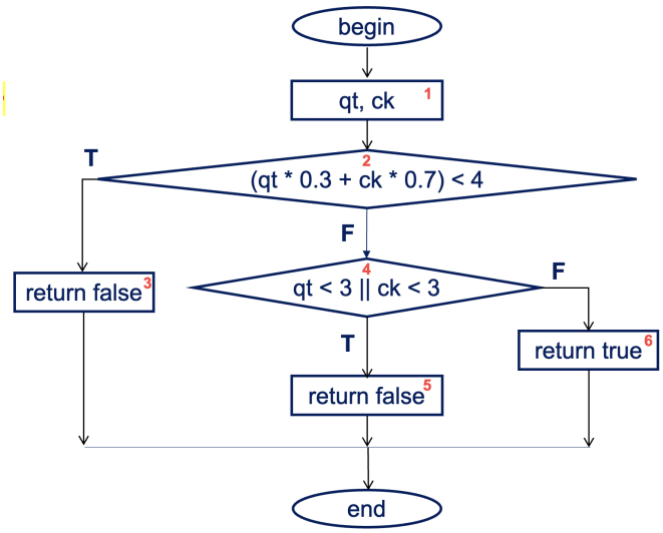
- Cho đoạn chương trình xét kết quả điểm học phần của sinh viên với mô tả như sau: Đầu vào của chương trình là điểm quá trình và điểm cuối kì (*qt* và *ck*) - giả

thiết thuộc khoảng [0,10]; Và đầu ra: *'đạt' nếu (qt * 0.3 + ck * 0.7) >= 4 và không có điểm liệt (< 3); 'không đạt' nếu ngược lại.*

Đoạn mã của chương trình được thể hiện như trên hình vẽ 9.10, đầu vào sẽ là điểm quá trình và cuối kì với giả thiết là cả hai giá trị đều thuộc khoảng [0,10]. Kết quả đầu ra sẽ là đạt (true) hoặc không đạt (false).

Để giải quyết bài toán này bước đầu tiên ta vẽ sơ đồ luồng điều khiển cho đoạn chương trình trên kết quả được chỉ ra trong hình 9.10. Các node 2 & 4 tương ứng với các điều kiện cần kiểm tra. Các node 3, 5 và 6 trên đồ thị tương ứng với các giá trị kết quả có thể trả về.

```
bool kq(float qt, float ck) {
 if ((qt * 0.3 + ck * 0.7) < 4)
  return false;
 else
  if (qt < 3 || ck < 3)
   return false;
  else
   return true;
}
```



Hình 9-10: Kiểm thử hộp trắng với luồng điều khiển

Về phủ đường dẫn, ta xác định có tất cả 3 đường dẫn, bao gồm:

- 1 – 2 – 3
- 1 – 2 – 4 – 5
- 1 – 2 – 4 – 6

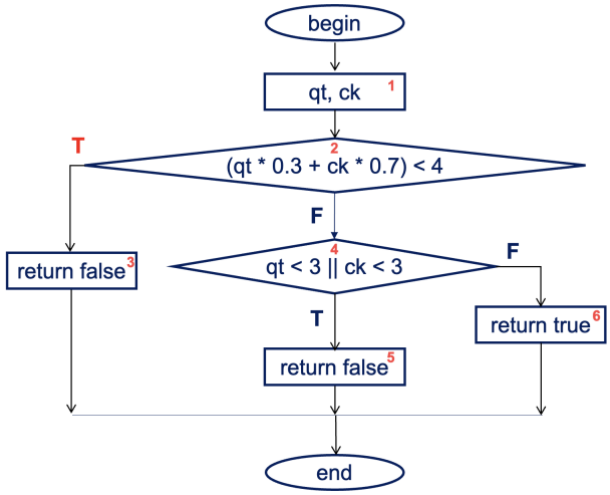
Qua đó, ta có thể lấy các giá trị tương ứng cho điểm quá trình và điểm cuối kì để thõa mãn các đường dẫn ở trên. Chúng ta sẽ có tất cả là 3 test case như bên dưới.

- (1 – 2 – 3): TC <1, 5, false>
- (1 – 2 – 4 – 5): **TC <2, 8, false>**
- (1 – 2 – 4 – 6): **TC <4, 8, true>**

Về bao phủ nhánh: ta xét các nhánh tương ứng với các điều kiện

- Trường hợp 1: Với tổng điểm quá trình và cuối kì có nhân hệ số < 4, kết quả trả về là false, tương ứng ta có thể chọn giá trị cho các điểm quá trình và cuối kì để tạo ra test case như sau **TC <1, 5, false>**.

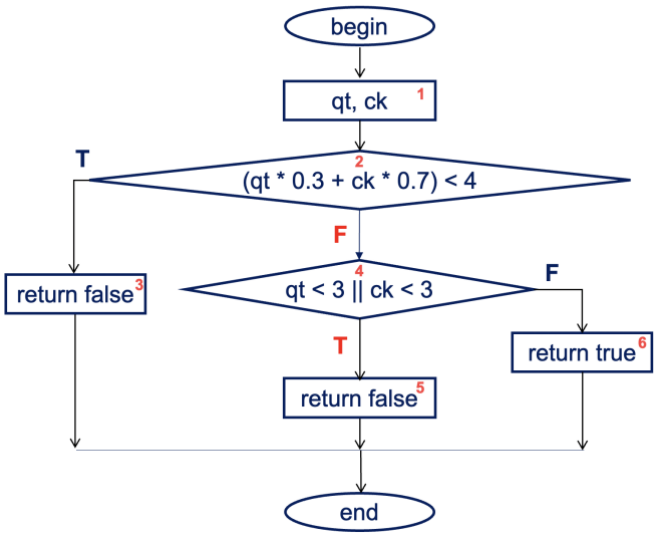
```
bool kq(float qt, float ck) {
 if ((qt * 0.3 + ck * 0.7) < 4)
  return false;
 else
  if (qt < 3 || ck < 3)
   return false;
 else
   return true;
}
```



Hình 9-11: Luồng điều khiển với trường hợp (qt * 0.3 + ck * 0.7) < 4 (đúng)

- Trường hợp 2: Với tổng điểm quá trình và cuối kì có nhân hệ số > 4, nhưng tồn tại một điểm < 3, kết quả trả về sẽ là false, tương ứng ta có thể chọn giá trị cho các điểm quá trình và cuối kì để tạo ra test case sau **TC <2, 8, false>**.

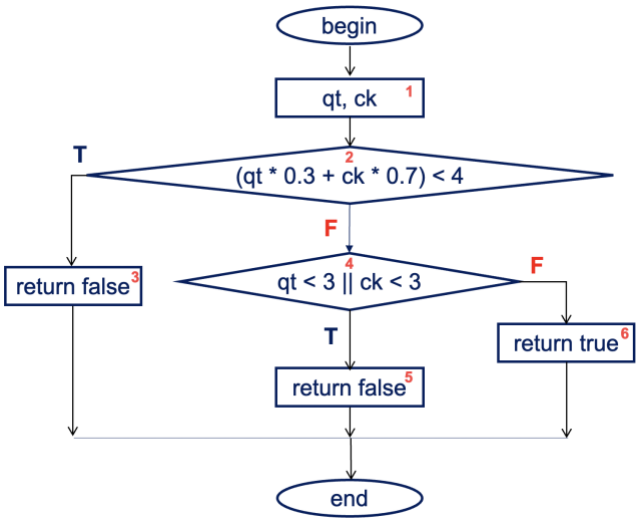
```
bool kq(float qt, float ck) {
 if ((qt * 0.3 + ck * 0.7) < 4)
  return false;
 else
  if (qt < 3 || ck < 3)
   return false;
 else
   return true;
}
```



Hình 9-12: Luồng điều khiển với trường hợp (qt * 0.3 + ck * 0.7) < 4 (sai) và (qt < 3 || ck <3) (đúng)

- Trường hợp 3 (cuối): Với tổng điểm quá trình và cuối kì có nhân hệ số > 4, và không có điểm nào < 3, kết quả trả về là true, tương ứng ta có thể chọn giá trị cho các điểm quá trình và cuối kì để tạo ra test case sau **TC <4, 8, true>**.

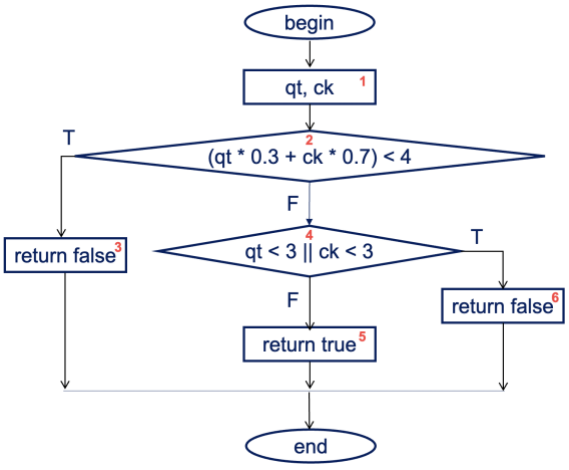
```
bool kq(float qt, float ck) {
 if ((qt * 0.3 + ck * 0.7) < 4)
  return false;
 else
  if (qt < 3 || ck < 3)
   return false;
  else
   return true;
}
```



Hình 9-13: Luồng điều khiển với trường hợp (qt * 0.3 + ck * 0.7) < 4 (sai) và (qt < 3 || ck <3) (sai)

Về bao phủ lệnh: ta có 3 câu lệnh trả về kết quả ở các node 3, 5, 6 (tương ứng với các câu lệnh 1, 2, 3) như trên hình vẽ. Theo đó ta có thể thiết lập các test case như ở dưới.

```
bool kq(float qt, float ck) {
 if ((qt * 0.3 + ck * 0.7) < 4)
  return false; //1
 else
  if (qt < 3 || ck < 3)
   return false; //2
  else
   return true; //3
}
```



Hình 9-14: Minh họa bao phủ lệnh

- //1: TC <1, 5, false>
- //2: TC <2, 8, false>
- //3: TC <4, 8, true>

Nhận xét: Trong bài tập này cả 3 test case tìm được trong ví dụ ở trên đều có thể áp dụng với cùng 3 tiêu chí đã cho. Như vậy không có sự phân biệt rõ ràng tiêu chí nào là mạnh nhất hay yếu nhất.

9.4 Phương pháp kiểm thử hộp đen

9.4.1 Khái niệm kiểm thử hộp đen

Kiểm thử hộp đen là một chiến lược trong đó việc kiểm thử được thực hiện dựa trên yêu cầu và đặc tả của chức năng, trong đó cần xác minh đầu ra của chức năng mà không quan tâm tới cấu trúc bên trong của chương trình. Với kĩ thuật kiểm thử này, ta sẽ không biết lỗi được kiểm soát bên trong như thế nào, cũng không thể đảm bảo kiểm thử được tất cả các đường dẫn như các chiến lược kiểm thử hộp trắng.

Kiểm thử hộp đen được xem là kiểm thử chức năng trong khi kiểm thử hộp trắng là kiểm thử cấu trúc. Kĩ thuật kiểm thử hộp đen, thường được thực hiện bởi các kiểm thử viên, kĩ sư đảm bảo chất lượng, và chủ yếu được sử dụng ở giai đoạn kiểm thử hệ thống và kiểm thử chấp nhận.

Hình 9-15: Kiểm thử hộp đen

- Đầu vào của kiểm thử hộp đen là những đặc tả chi tiết của chức năng. Đầu ra là các kết quả hoạt động của hệ thống tương ứng với từng loại dữ liệu đầu vào và kịch bản kiểm thử.
- Với kiểm thử hộp đen, thông thường các phương pháp thiết kế ca kiểm thử được sử dụng là:
    - Phân chia lớp tương đương
    - Phân tích giá trị biên
    - Sử dụng bảng quyết định

- Kiểm thử trạng thái

9.4.2 Kĩ thuật phân lớp tương đương

Kĩ thuật phân lớp tương đương, ta dựa vào miền dữ liệu trong đặc tả của đầu vào (input) và đầu ra (output) để xác định các phân lớp tương đương. Ở đây, một phân lớp tương đương được định nghĩa là một miền dữ liệu (input hoặc output) trong đó với mọi dữ liệu thuộc miền này thì chức năng được kiểm thử sẽ thực hiện cùng một hành vi.

Như vậy ta cần phân chia miền đầu vào thành các lớp dữ liệu. Một lớp tương đương sẽ bao gồm một tập hợp dữ liệu mà hệ thống sẽ hoạt động theo cùng một cách. Điều đó có nghĩa là mọi giá trị dữ liệu trong lớp đều tạo ra các hoạt động tương đương.

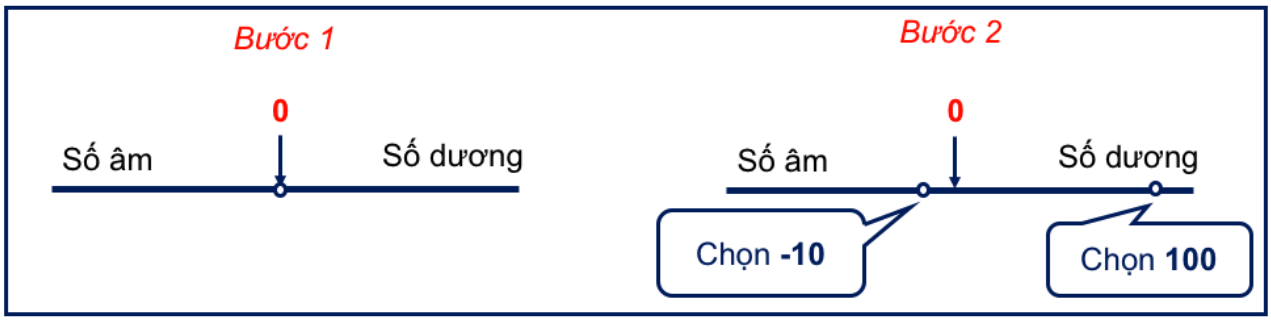
Lấy một ví dụ đơn giản là có thể dễ dàng phân dữ liệu thành hai lớp: một lớp là dữ liệu hợp lệ, còn lớp kia là dữ liệu không hợp lệ.

Ta xem xét ví dụ sau: Cho một hàm thực hiện việc kiểm tra một số nguyên là âm hay dương. Đầu vào của chương trình sẽ là một số nguyên, đầu ra sẽ là giá trị logic true nếu đó là số nguyên âm, hoặc false nếu số đó là số nguyên dương.

Trong kĩ thuật phân lớp tương đương cho bài toán này được giải quyết qua hai bước.

- Bước 1: Xác định 2 lớp tương đương ứng với input > 0 và input < 0
- Bước 2: Ở mỗi phân lớp tương đương lấy 1 giá trị đại diện để xây dựng ca kiểm thử.

Hình vẽ bên dưới mô tả cách thực hiện hai bước trên. Từ đây ta xác định ra hai test case là <-10, true> và <100, false>



Hình 9-16: Kĩ thuật phân lớp tương đương

Ở ví dụ tiếp theo, ta xem xét hệ thống xác nhận đăng kí hồ sơ bảo hiểm với mô tả như sau:

- Hệ thống sẽ từ chối hoặc chấp nhận hồ sơ đăng ký bảo hiểm

- Từ chối hồ sơ của nam trên 80 tuổi
- Từ chối hồ sơ của nữ trên 85 tuổi

Yêu cầu của bài toán là cách thức xác định các lớp tương đương như thế nào?

Phân tích bài toán: Với bài toán này, ta có input là giới tính (nam/nữ) và độ tuổi. Giá trị output là chấp nhận hay từ chối.

Dựa vào đặc tả của bài toán ta có thể phân thành các lớp ứng với giới tính và độ tuổi và từ đó ta có thể chọn các giá trị cụ thể và thu được các test case như trong bảng sau.

Bảng 9-2: Giá trị và test case cho các lớp

Lớp	Test Case
C1: Input: Nam, trên 80 tuổi	T1: nam, 83, từ chối
C2: Input: Nam, nhỏ hơn hoặc bằng 80 tuổi	T2: nam, 56, chấp nhận
C3: Input: Nữ, nhỏ hơn hoặc bằng 85 tuổi	T3: nữ, 83, chấp nhận
C4: Input: Nữ, trên 85 tuổi	T4: nữ, 87, từ chối

Đánh giá:

- Ưu điểm: Ta có thể thấy, ưu điểm của kĩ thuật phân lớp tương đương là giảm được số lượng ca kiểm thử, thay vì phải quét toàn bộ miền dữ liệu đầu vào. Và đảm bảo được rằng mỗi lớp đều được kiểm thử.
- Nhược điểm: Kĩ thuật phân lớp tương đương cũng có một số nhược điểm như việc phân chia các lớp như thế nào phụ thuộc vào kinh nghiệm của kiểm thử viên. Có thể một số phân lớp chứa dữ dữ liệu không hợp lệ lại không được xét đến. Tiếp theo là không xem xét các giá trị biên của các lớp mà lỗi thường được xuất hiện ở các giá trị biên đó. Cuối cùng là nếu mối quan hệ của dữ liệu đầu vào không còn độc lập nữa thì việc phân chia thành các lớp rõ ràng sẽ khó. Ở đây cần kết hợp thêm bảng kiểm thử để xây dựng các ca kiểm thử.

9.5 Bài tập kiểm thử hộp đen với phân lớp tương đương

Bài toán kiểm thử hộp đen với kĩ thuật phân lớp tương đương cần có mô tả cho bài toán bao gồm dữ liệu đầu vào là gì, miền dữ liệu tương ứng cho dữ liệu đầu vào và các yêu

cầu về kết quả đầu ra. Dựa vào các mô tả đó, yêu cầu cần thiết kế các ca kiểm thử tương ứng.

- Nhắc lại kiến thức về lớp tương đương. Ở đây ta cần phân chia miền đầu vào thành các lớp dữ liệu. Ở đó, một lớp tương đương sẽ bao gồm một tập hợp dữ liệu làm cho hệ thống hoạt động theo cùng một cách (hành vi) và mọi giá trị dữ liệu trong lớp đều tương đương dưới góc độ hành vi đó. Từ đó, ta cần dựa vào miền dữ liệu trong đặc tả của đầu vào (input) và đầu ra (output) để xác định các phân lớp tương đương. Một phân lớp tương đương được định nghĩa là một miền dữ liệu (input) trong đó với mọi điểm dữ liệu thuộc miền, chức năng được kiểm tra sẽ thực hiện cùng một hành vi.

Cách thực hiện: Với bài toán kiểm thử lớp tương đương, cách thức giải quyết được tiến hành qua các bước như sau:

- Bước 1: Xác định các dữ liệu đầu vào và miền giá trị tương ứng
- Bước 2: Phân các lớp cho miền giá trị của dữ liệu đầu vào
- Bước 3: Lấy giá trị cho các dữ liệu đầu vào dựa trên các lớp đã được phân ở bước 2.
- Bước 4: Xác định các test case tương ứng ở bước 3.

Ví dụ: Ta xem xét bài tập áp dụng sau.

Mô tả: Chương trình nhận đầu vào là giá trị điểm thi của sinh viên ở hai môn toán và vật lý.

Sinh viên sẽ vượt qua kì thi nếu thõa mãn điều kiện:

- Cả điểm toán và điểm vật lý >= 70 (xét theo thang điểm 100)
- Điểm trung bình của toán và vật lý >= 80

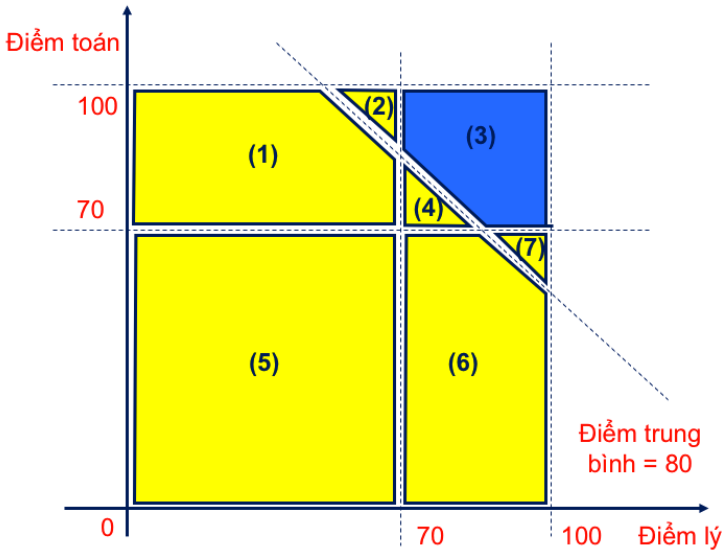
Sinh viên không đạt trong kì thi nếu có kết quả ngược lại.

Thực hiện: Bước đầu tiên, ta cần xác định ra các lớp dựa trên miền dữ liệu của các giá trị đầu vào. Dễ thấy, các lớp được giới hạn bởi các đường.

- Toán = 100
- Toán = 70
- Lý = 100
- Lý = 70

- Và trung bình của Toán và Lý là 80

Như vậy, ta có 7 vùng dữ liệu (gọi là lớp) như hình vẽ 9.17 bên dưới.



Hình 9-17: Vùng dữ liệu với các lớp tương đương

Tiếp theo, với mỗi vùng dữ liệu, ta lấy một giá trị trong đó. Với 7 lớp, ta chọn 7 giá trị cho dữ liệu đầu vào và đầu ra như ở bảng trên. Từ đó, ta có thể tạo ra 7 test case tương ứng với các giá trị đã chọn như được .

Bảng 9-3: Các giá trị cho test case

TC	Toán	Lý	Kết quả
1	80	50	Không đạt
2	96	66	Không đạt
3	86	86	Đạt
4	75	75	Không đạt
5	50	50	Không đạt
6	66	96	Không đạt
7	50	80	Không đạt

Nhận xét: Bài toán sẽ phức tạp hơn nếu số lượng lớp tương đương tăng lên! Một câu hỏi đặt ra là sẽ phải giải quyết như thế nào nếu ngoài điểm toán, điểm lý còn có các điểm của những môn khác nữa. Việc phân chia miền dữ liệu hay là lớp tương đương sẽ phức tạp hơn rất nhiều.

9.6 Quy trình đảm bảo chất lượng phần mềm

9.6.1 Các vấn đề trong đảm bảo chất lượng phần mềm

Chất lượng phần mềm được định nghĩa là mức độ hệ thống hay thành phần của hệ thống đạt được các yêu cầu đặt ra và hệ thống hay thành phần đạt được những mong đợi của khách hàng hay người dùng hệ thống đó. Như vậy, chất lượng phần mềm được nhìn nhận trên 2 quan điểm, đó là: quan điểm của người phát triển và quan điểm của người dùng. Trong quá trình phát triển sản phẩm, có một số vấn đề về đảm bảo chất lượng phần mềm bao gồm:

- Thứ nhất, quan điểm về chất lượng của người dùng và đội phát triển là khác nhau
- Đội phát triển thì quan tâm đến khía cạnh phát triển hệ thống: có tái sử dụng, dễ kiểm thử, dễ bảo trì được hay không. Còn người dùng thì quan tâm đến: hiệu quả, hiệu suất và độ tin cậy của phần mềm;
- Tuy nhiên, đôi khi để đảm bảo cho code sạch, dễ tái sử dụng, dễ bảo trì thì các tiêu chí chất lượng khác như hiệu quả, độ tin cậy có thể giảm đi.
- Thứ 2, đặc tả phần mềm không bao giờ là đầy đủ và nhất quán cho nên rất khó để quy chuẩn được chất lượng phần mềm ngay từ đầu.

9.6.2 Quy trình đảm bảo chất lượng phần mềm

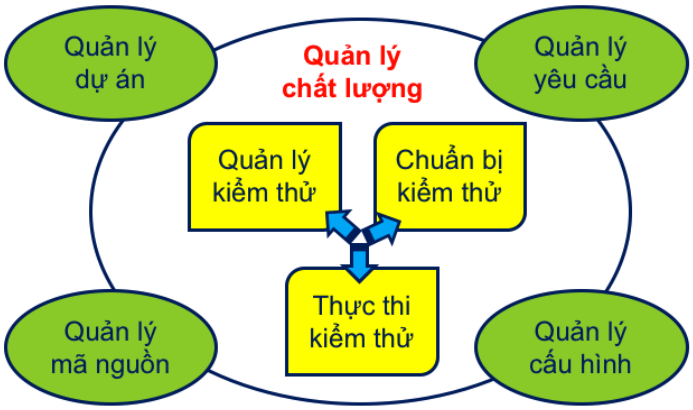
Quy trình đảm bảo chất lượng được thực hiện để xác định xem sản phẩm hoặc dịch vụ đang được phát triển có đạt được các yêu cầu theo đặc tả ban đầu hay không. Đây là một quy trình mang tính hệ thống, cần được tích hợp với tất cả các quy trình khác trong toàn bộ vòng đời phát triển của phần mềm, bao gồm:

- Quản lý dự án
- Quản lý yêu cầu
- Quản lý cấu hình
- Quản lý mã nguồn

Kiểm thử là nền tảng của quy trình đảm bảo chất lượng phần mềm và bao gồm 3 hoạt động chính:

- Quản lý kiểm thử

- Chuẩn bị kiểm thử / Kế hoạch kiểm thử
- Thực thi kiểm thử



Hình 9-18: Quy trình quản lý chất lượng phần mềm

Quản lý dự án: Ngay tại bước lập kế hoạch cho dự án, quy trình đảm bảo chất lượng cần được khởi động:

- Thứ nhất, xác định các mục tiêu và tiêu chí chất lượng cần quan tâm và kiểm chứng trong dự án;
- Thứ hai, xác định các thủ tục kiểm thử và đảm bảo chất lượng cần được áp dụng;
- Thứ ba, chỉ định trước các nguồn lực, bao gồm: con người, phần cứng, hệ thống cơ sở hạ tầng và môi trường cần thiết để thực hiện kiểm thử và đảm bảo chất lượng.

Quản lý yêu cầu: Để cho quy trình đảm bảo chất lượng được thực hiện xuyên suốt dự án thì việc đảm bảo cho các yêu cầu của sản phẩm cần được tổ chức một cách có hệ thống. Quy trình đảm bảo chất lượng cần thực hiện các bước sau:

- Thứ nhất, xác định tất cả các yếu tố quan trọng của các chức năng của hệ thống và cần nhận được xác nhận từ phía khách hàng;
- Thứ hai, xác định các hạn chế và ngoại lệ có thể có từ các yêu cầu của khách hàng;
- Thứ ba, thiết lập tất cả các hiểu biết chung về dự án cho tất cả các thành viên của đội dự án, để đảm bảo đội dự án nắm được các vấn đề của dự án;

- Thứ tư, cần cung cấp cơ sở cho các ước tính, phát triển mã nguồn và các quy trình kiểm chứng.

Đây là điểm khởi đầu cho việc giao tiếp và quản lý thay đổi một cách hiệu quả trong suốt quy trình phát triển sản phẩm.

Quản lý cấu hình: Việc quản lý cấu hình phần mềm là bắt buộc trong mọi dự án phát triển phần mềm. Đảm bảo chất lượng phần mềm yêu cầu phải quản lý cấu hình một cách chặt chẽ và tuân theo các quy trình cụ thể. Điều đó nhằm mục đích:

- Để đảm bảo cho các nhóm phát triển phân tán có thể làm việc trên cùng mã nguồn;
- Đảm bảo cho các phiên bản cơ sở (baseline version) có thể độc lập mà không bị ảnh hưởng bởi bất kỳ sự phá vỡ cấu trúc bởi một lỗi nào;
- Để biết được ở phiên bản mã nguồn nào lỗi xuất hiện lần đầu tiên và xác định được những thay đổi nào làm xuất hiện lỗi đó;
- Việc quản lý cấu hình tốt sẽ đảm bảo việc kiểm thử tự động được thực hiện tốt nhất ở mỗi giai đoạn mà phiên bản mã nguồn mới được ra đời.

Quản lý mã nguồn: Áp dụng quản lý mã nguồn sẽ làm tăng hiệu quả của quy trình đảm bảo chất lượng phần mềm. Theo đó:

- Phát triển mã nguồn thống nhất sẽ làm tăng hiệu quả và làm cho việc bảo trì sản phẩm dễ dàng hơn mà không cần lệ thuộc cứng vào một hoặc một nhóm thành viên của dự án;
- Cho phép phát hiện lỗi sớm nhất;
- Cho phép thiết kế mã nguồn theo hướng kiểm thử;
- Cho phép cục bộ hoá lỗi và giảm thiểu các tác động lên các thành phần phụ thuộc khác;
- Chia sẻ tri thức giữa các thành viên trong đội dự án và cho phép cải tiến kĩ năng cá nhân của các thành viên.

Kiểm thử: Việc kiểm thử có thể được thực hiện trong các pha riêng biệt hoặc các pha lồng nhau.

- Kiểm thử ở nhiều mức độ khác nhau của sản phẩm như: kiểm thử đơn vị, tính năng, tương tác, hệ thống;
- Kiểm thử trong suốt quá trình bảo trì sản phẩm.
- Kiểm thử các khía cạnh như: hiệu năng, hiệu suất, độ ổn định, khả năng tương tác, mức độ tuân thủ các chuẩn/quy định.

Quản lý kiểm thử: Về quản lý kiểm thử cần được thực hiện ngay từ giai đoạn đầu tiên và kéo dài đến khi dự án hoàn toàn kết thúc nhằm đạt được các mục tiêu về cải tiến chất lượng sản phẩm. Các mục đích chính của quản lý kiểm thử bao gồm:

- Xác định các chiến lược kiểm thử và phạm vi các chức năng của dự án;
- Các ràng buộc trong quá trình kiểm thử;
- Các thành phần, các tính năng hoặc các chức năng cần phải kiểm thử và không cần phải kiểm thử;
- Kiểm thử xem các mục nào thông qua và mục nào bị lỗi cần phải sửa chữa, thay đổi. Quy trình kiểm thử và các tiêu chí quay lại tiếp tục kiểm thử sau khi đã sửa lỗi;
- Các công việc kiểm thử, các yêu cầu về môi trường và triển khai cần thiết để tiến hành kiểm thử;
- Các trách nhiệm, phân công công việc và việc đào tạo cần thiết hoặc kết hợp với các đội nhóm phát triển khác trong quá trình kiểm thử;
- Lên kế hoạch kiểm thử cụ thể và xác định các rủi ro có thể có;
- Đánh giá bằng các độ đo chất lượng và viết báo cáo.

Chuẩn bị kiểm thử: Mục tiêu của việc chuẩn bị kiểm thử là để xác định được các loại kiểm thử và môi trường kiểm thử cần thiết cho từng pha phát triển dự án theo kế hoạch kiểm thử. Các hoạt động chính của chuẩn bị kiểm thử gồm có:

- Phân tích các yêu cầu chức năng, các đặc tả và phạm vi của các chức năng;
- Phát triển ma trận theo vết yêu cầu (RTM – Requirement Traceability Matrix);
- Thiết kế và phát triển các trường hợp kiểm thử cho tất cả các pha đã lên kế hoạch;

- Sử dụng các cách tiếp cận và các kĩ thuật để đảm bảo được một tập hợp các ca kiểm thử hiệu quả và tối ưu;
- Phát triển hoặc sử dụng thêm các công cụ để thực hiện kiểm thử tự động;
- Thiết lập các môi trường đặc biệt để có thể thực hiện kiểm thử phi chức năng (ví dụ: thiết lập các kết nối để đo lường hiệu năng và độ tin cậy của hệ thống khi có nhiều truy nhập đến).

Thực hiện kiểm thử: Áp dụng trực tiếp các định nghĩa đã xác định của kiểm thử được đưa ra trong pha chuẩn bị kiểm thử vào quá trình phát triển sản phẩm. Các hoạt động thực hiện kiểm thử thông thường bao gồm:

- Chuẩn bị môi trường thực thi kiểm thử;
- Thực thi các ca kiểm thử;
- Phân tích các kết quả thực thi;
- Thu thập và các kết quả kiểm thử và ghi lại các lỗi;
- Hỗ trợ đội phát triển để tái hiện lại các lỗi và phân tích các lỗi;
- Kiểm tra các ca kiểm thử đã được sửa chữa;
- Phân tích các lỗi có thể bị bỏ sót;
- Cập nhật các ca kiểm thử và môi trường kiểm thử theo kết quả.

9.7 Bảo trì phần mềm

Quy trình đảm bảo chất lượng được thực hiện để xác định xem sản phẩm hoặc dịch vụ đang được phát triển có đạt được các yêu cầu theo đặc tả ban đầu hay không. Đây là một quy trình mang tính hệ thống, cần được tích hợp với tất cả các quy trình khác trong toàn bộ vòng đời phát triển của phần mềm, bao gồm:

9.7.1 Khái niệm bảo trì phần mềm

Bảo trì phần mềm là công việc sửa đổi (tu sửa hoặc thay đổi) một phần mềm đã được phát triển (gồm chương trình, dữ liệu, và tài liệu), sau khi đã bàn giao để chỉnh các lỗi phát sinh, cải thiện hiệu năng hoặc các thuộc tính, hoặc làm cho phần mềm thích ứng trong một môi trường đã bị thay đổi.

9.7.2 Các hình thái bảo trì phần mềm

Chúng ta có các hình thức bảo trì, bao gồm: tu sửa, thích nghi, cải tiến, phòng ngừa.

a) **Bảo trì để tu sửa (sửa lỗi):** Đây là hình thức bảo trì nhằm mục đích khắc phục những khiếm khuyết có trong phần mềm. Một số nguyên nhân điển hình, bao gồm:
   - Kỹ sư phần mềm và khách hàng hiểu nhầm nhau;
   - Lỗi tiềm ẩn của phần mềm do sơ ý của lập trình hoặc quá trình kiểm thử chưa được bao quát hết;
   - Vấn đề tính năng của phần mềm: không đáp ứng được yêu cầu về bộ nhớ, tệp,…
   - Thiết kế sai, biên tập sai,. . .
   - Thiếu chuẩn hóa trong phát triển phần mềm (trước đó).

Để thực hiện được hình thức bảo trì này, ta cần kiểm tra lại thiết kế để tu sửa (gọi là kĩ nghệ ngược – reverse engineering).

b) **Bảo trì để thích hợp:** là hình thức tu chỉnh phần mềm theo thay đổi của môi trường bên ngoài nhằm duy trì và quản lý phần mềm theo vòng đời của nó. Thay đổi phần mềm thích nghi với môi trường gồm phần cứng, môi trường phần mềm. Những nguyên nhân chính cho hình thức bảo trì này gồm:
   - Thay đổi về phần cứng (ngoại vi, máy chủ,. . .);
   - Thay đổi về phần mềm (môi trường): thay đổi hệ điều hành (OS);
   - Thay đổi cấu trúc tệp hoặc mở rộng CSDL.

c) **Bảo trì để cải tiến:** là việc tu chỉnh phần mềm theo các yêu cầu để ngày càng hoàn thiện hơn, đầy đủ hơn, hợp lý hơn. Những nguyên nhân chính cho hình thức bảo trì này là:
   - Do muốn nâng cao hiệu suất nên cần cải tiến;
   - Muốn mở rộng thêm chức năng mới cho hệ thống;
   - Cải tiến nghiệp vụ kéo theo tư liệu vận hành và trình tự công việc;
   - Thay đổi người dùng hoặc thay đổi thao tác.

Hình thức này còn gọi là tái kỹ nghệ (re-engineering). Mục đích: đưa ra một thiết kế cùng chức năng nhưng có chất lượng cao hơn.

d) **Bảo trì để phòng ngừa:** Đây là công việc tu chỉnh chương trình có tính đến việc mở rộng và thay đổi trong tương lai của phần mềm.

- Thực ra trong thiết kế phần mềm đã phải tính đến tính mở rộng của nó, nên thực tế ít khi ta gặp bảo trì phòng ngừa nếu như phần mềm được thiết kế tốt.
- Mục đích của hình thức này là sửa đổi để thích hợp với yêu cầu thay đổi sẽ có của người dùng.
- Thường sẽ thực hiện những thay đổi trên thiết kế không rõ ràng.
- Ta cần hiểu hoạt động bên trong chương trình và cần thiết kế / lập trình lại.

9.7.3 Quy trình bảo trì phần mềm

Quy trình bảo trì phần mềm cần tuân theo các pha phân tích, thiết kế, phát triển và kiểm thử từ khi phát sinh vấn đề cho đến khi giải quyết xong. Các nhiệm vụ bảo trì gồm có:

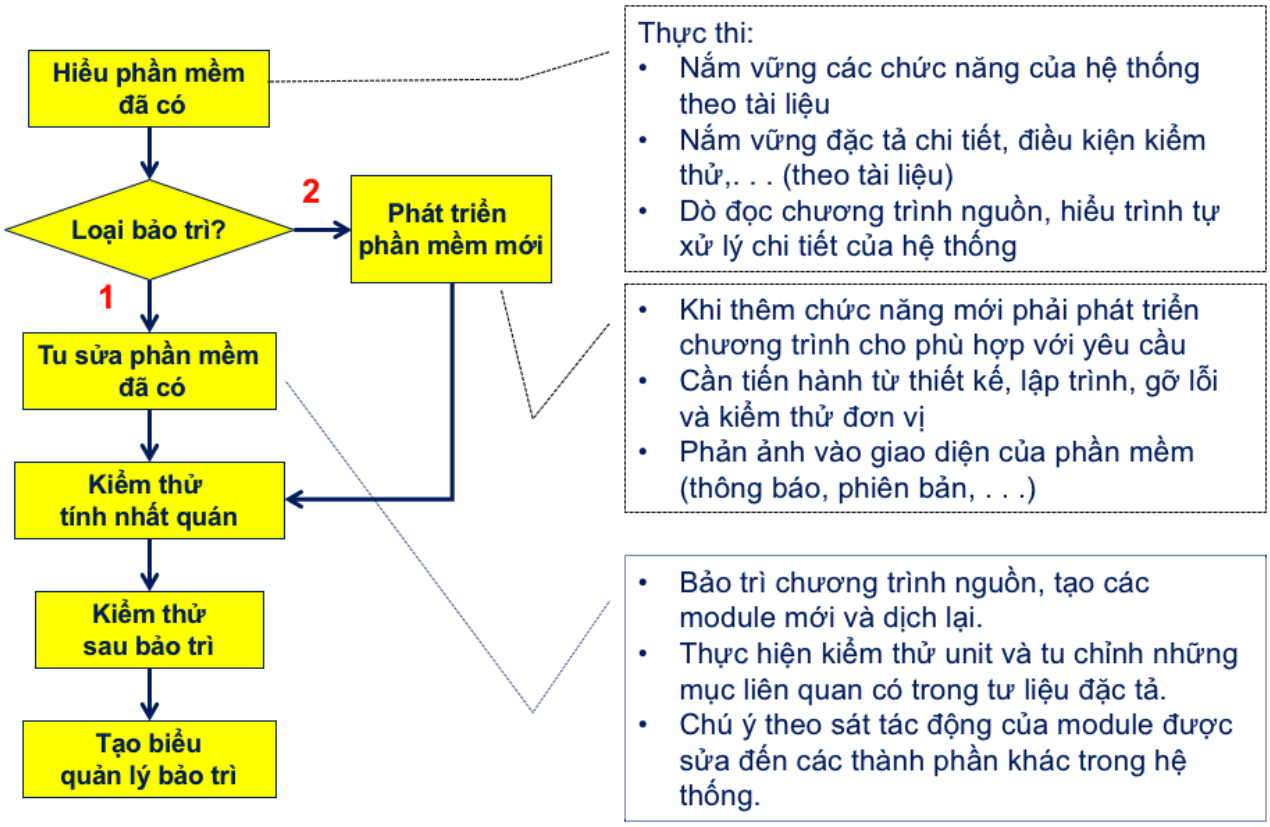
- phân tích tác động, phân tích những giá trị lợi ích, và cô lập các thành phần cần bảo trì;
- thiết kế lại hệ thống (ở đây cần phải biết cách tu sửa, thay đổi);
- thay thế mã nguồn và kiểm soát từng đơn vị thành phần hệ thống, có tính đến thời gian lập trình.

Có 2 loại thao tác bảo trì:

- Thứ nhất, tu chỉnh cái đã có (loại 1);
- Thứ hai, thêm cái mới (loại 2).

Trên hình 9.x là sơ đồ quy trình bảo trì. Để bảo trì, trước hết ta cần hiểu về phần mềm đã có, tức là nắm vững về các chức năng, đặc tả, chương trình nguồn.

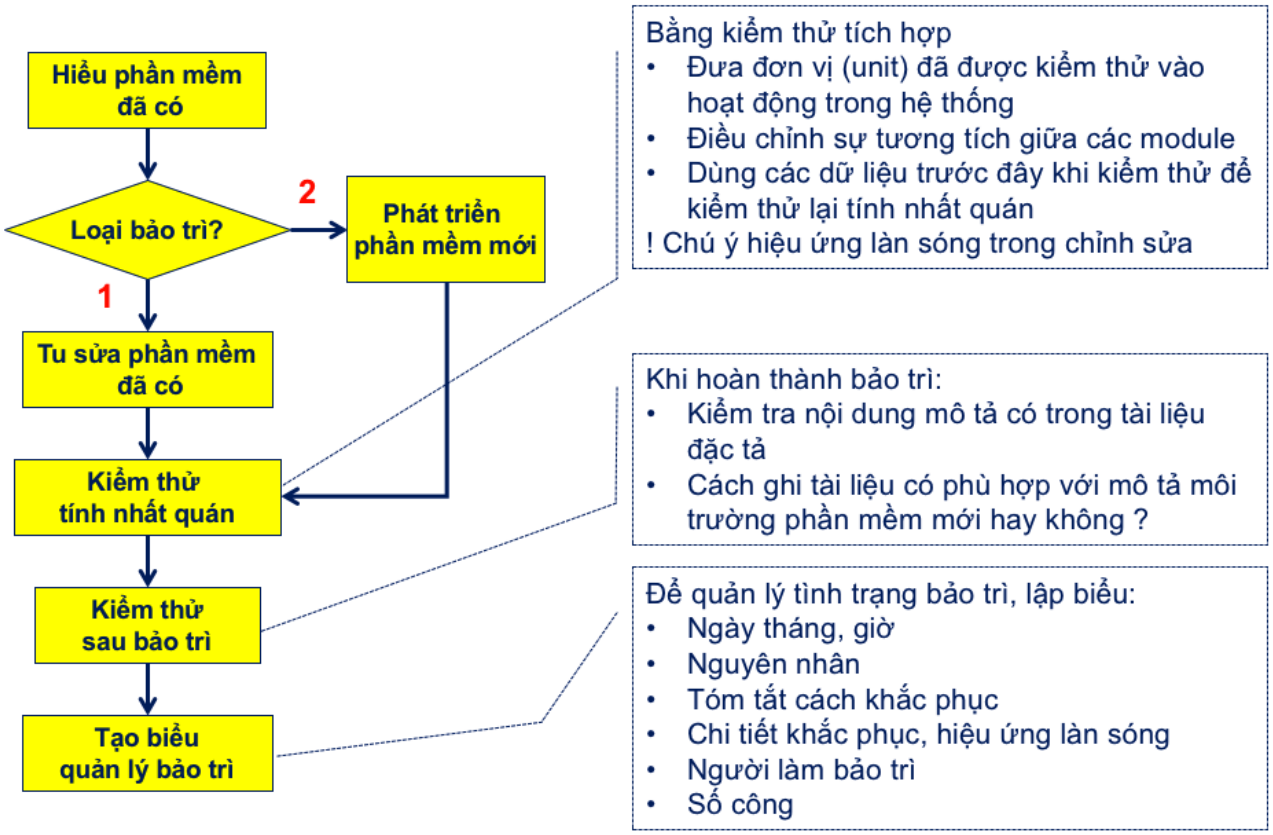
- Khi thêm chức năng mới, phải phát triển chương trình cho phù hợp với yêu cầu, cần tiến hành từ bước thiết kế, lập trình, gỡ lỗi và kiểm thử đơn vị.
- Để tu sửa phần mềm đã có cần bảo trì chương trình nguồn, tạo các mô-đun mới và dịch lại. Thực hiện kiểm thử đơn vị và tu chỉnh những mục liên quan có trong tư liệu đặc tả. Cần chú ý theo sát tác động của các mô-đun được sửa đến các thành phần khác trong hệ thống.



Hình 9-19: Các hình thức bảo trì

Quá trình bảo trì cần kiểm thử tính tính nhất quán bằng kiểm thử tích hợp, ở đây đưa đơn vị đã được kiểm thử vào hoạt động trong hệ thống, điều chỉnh sự tương tích giữa các mô-đun, dùng các dữ liệu trước đây khi kiểm thử để kiểm tra lại tính nhất quán và cần chú ý hiệu ứng làn sóng trong chỉnh sửa;

Khi hoàn thành công tác bảo trì, cần kiểm tra nội dung mô tả có trong tài liệu đặc tả. Cách ghi tài liệu có phù hợp với mô tả môi trường phần mềm mới hay không?



Hình 9-20: Các công việc cho kiểm thử và quản lý bảo trì

TÀI LIỆU THAM KHẢO

1. R. Pressman, Software Engineering: A Practitioner's Approach. 8th Ed., McGraw-Hill, 2016.
2. I. Sommerville, Software Engineering. 10th Ed., AddisonWesley, 2017.
3. Pankaj Jalote, An Integrated Approach to Software Engineering, 3rd Ed., Springer.
4. Shari Lawrence Pleeger, Joanne M.Atlee, Software Engineering theory and practice. 4th Ed., Pearson, 2009
5. UML 2 Toolkit. Hans-Erik Eriksson and Magnus Penker. Wiley Publishing Inc. Available at: http://www.ges.dc.ufscar.br/posgraduacao/UML_2_Toolkit.pdf.
6. Astah Manual. Available at: http://astah.net/tutorial/astah%20professional%20referencemanual.pdf
7. PMBOK® Guide – Seventh Edition. PMI, 2021.
8. Software extension to the PMBOK® Guide 5th Edition. PMI. 2013.
9. Robert K. Wysocki, Effective Software Project Management. Wiley. 2006.
10. ISO 9241: Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts, 2018. Part 110: Interaction principles, 2020. Part 100: Overview of ISO 9241 software ergonomic standards, 2023. Part 171: Guidance on software accessibility, 2008.
11. ISO/IEC 25060 Series: Common industry format for usability-related information (CIF).
12. Fowler, M. (2009). Refactoring improving the design of existing code. Addison-Wesley.
13. Kernighan, B.W. and Plauger, P.J. (1978) The elements of programming style. New York: McGraw-Hill
14. Shvets, A. Refactoring and Design Patterns. Available at: https://refactoring.guru/

15. Sun Microsystems (1999), Java code conventions. Available at: https://www.oracle.com/technetwork/java/codeconventions-150003.pdf
16. ORACLE, How to Write Doc Comments for the Javadoc Tool, How to write Doc comments for the javadoc tool. Available at: https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html
17. Wake, B. (2000) Java Coding Standard. Available at: https://xp123.com/articles/java-coding-standard/
18. UX Writing Best Practices – URL: https://m3.material.io/foundations/content-design/style-guide/ux-writing-best-practices
19. Figma – URL: https://www.figma.com/community/file/1228357783685927211