

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN



BÀI GIẢNG
PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

Giảng viên: VÕ TRUNG HÙNG

Đà Nẵng, 10-2006

BÀI 1. QUY TRÌNH PHÁT TRIỂN PHẦN MỀM

I. Công nghệ phần mềm (SE – Software Engineering)

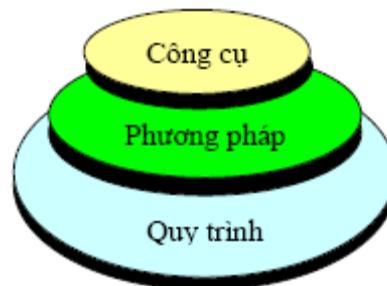
Công nghệ phần mềm là việc áp dụng các công cụ, các kỹ thuật một cách hệ thống trong việc phát triển các ứng dụng dựa trên máy tính. Đó chính là việc áp dụng các quan điểm, các qui trình có kỷ luật và lượng hoá được, có bài bản và hệ thống để phát triển, vận hành và bảo trì phần mềm.

Công nghệ phần mềm bao gồm các hoạt động: phát triển, đưa vào vận hành, bảo trì và loại bỏ phần mềm một cách có hệ thống. Các kỹ sư phần mềm sẽ được cung cấp với các kỹ thuật, công cụ cơ bản nhằm phát triển các hệ thống phần mềm.

Như vậy, công nghệ phần mềm là lĩnh vực nghiên cứu của tin học, nhằm đề xuất các nguyên lý, phương pháp, công cụ, cách tiếp cận và phương tiện phục vụ cho việc thiết kế và cài đặt các sản phẩm phần mềm có chất lượng; là sự áp dụng có hệ thống và kỷ luật những phương pháp có thể định lượng được trong các khâu phát triển, vận hành và bảo trì phần mềm. Nghĩa là áp dụng những qui tắc của công nghệ vào phần mềm (theo IEEE).

Ngày nay, sự phát triển phần mềm ngày càng thực sự khó kiểm soát được; các dự án phần mềm thường kéo dài và vượt quá chi phí cho phép. Những nhà lập trình chuyên nghiệp phải cố gắng hoàn thành các dự án phần mềm một cách có chất lượng, đúng hạn trong chi phí cho phép.

Theo quan điểm của nhiều nhà nghiên cứu, có thể nhìn công nghệ phần mềm là một mô hình được phân theo ba tầng mà tất cả các tầng này đều nhằm tới mục tiêu chất lượng, chi phí, thời hạn phát triển phần mềm.



Hình 1. Mô hình 3 tầng của công nghệ phần mềm

Ở đây tầng quy trình (process) liên quan tới vấn đề quản trị phát triển phần mềm như lập kế hoạch, quản trị chất lượng, tiến độ, chi phí, mua bán sản phẩm phụ, cấu hình phần mềm, quản trị sự thay đổi, quản trị nhân sự (trong môi trường làm việc nhóm), việc chuyển giao, đào tạo, tài liệu.

Tầng phương pháp (methods) hay cách thức, công nghệ, kỹ thuật để làm phần mềm: liên quan đến tất cả các công đoạn phát triển hệ thống như nghiên cứu yêu cầu, thiết kế, lập trình, kiểm thử và bảo trì. Phương pháp dựa trên những nguyên lý cơ bản nhất cho tất cả các lĩnh vực công nghệ kể cả các hoạt động mô hình hoá và kỹ thuật mô tả.

Tầng công cụ (tools) liên quan đến việc cung cấp các phương tiện hỗ trợ tự động hay bán tự động cho các tầng quá trình và phương pháp (công nghệ).

Qua sơ đồ trên, ta thấy rõ công nghệ phần mềm là một khái niệm đề cập không chỉ tới các công nghệ và công cụ phần mềm mà còn tới cả cách thức phối hợp công nghệ, phương pháp và công cụ theo các quy trình nghiêm ngặt để làm ra sản phẩm có chất lượng.

II. Qui trình phần mềm

1. Giới thiệu

Qui trình phát triển phần mềm (còn gọi tắt là qui trình phần mềm) là một tập hợp các hành động phải được thực hiện trong quá trình phát triển và tiến hóa một hệ thống phần mềm.

Qui trình phần mềm xác định một bộ khung và tiêu chuẩn để triển khai công nghệ phần mềm. Những bước thường được thực hiện trong các qui trình phần mềm bao gồm:

- Đặc tả: đặc tả những gì hệ thống phải làm và các ràng buộc trong quá trình xây dựng hệ thống.

- Phát triển: xây dựng hệ thống phần mềm.

- Kiểm thử: kiểm tra xem liệu phần mềm đã thỏa mãn yêu cầu của khách hàng.

- Cải tiến: điều chỉnh và thay đổi phần mềm tương ứng với sự thay đổi yêu cầu.

2. Các giai đoạn của qui trình phần mềm

a. Phân tích và đặc tả yêu cầu

Phân tích và định rõ yêu cầu là bước kỹ thuật đầu tiên trong qui trình công nghệ phần mềm. Công việc ở bước này là tìm hiểu xem chúng ta phải phát triển cái gì, chứ không phải là phát triển như thế nào. Đích cuối cùng của khâu phân tích là tạo ra đặc tả yêu cầu, là tài liệu ràng buộc giữa khách hàng và người phát triển và là cơ sở của hợp đồng. Hoạt động phân tích là hoạt động phối hợp giữa khách hàng và người phân tích (bên phát triển). Khách hàng phát biểu yêu cầu và người phân tích hiểu, cụ thể hóa và biểu diễn lại yêu cầu. Hoạt động phân tích giữ một vai trò đặc biệt quan trọng trong phát triển phần mềm, giúp cho đảm bảo chất lượng của phần mềm (phần mềm đáng tin cậy). Phần mềm đáng tin cậy có nghĩa là phải thực hiện được chính xác, đầy đủ yêu cầu của người sử dụng. Nếu phân tích không tốt dẫn đến hiểu lầm yêu cầu thì việc sửa chữa sẽ trở nên rất tốn kém. Chi phí để sửa chữa sai sót về yêu cầu sẽ tăng lên gấp bội nếu như sai sót đó được phát hiện muộn, ví dụ như ở bước thiết kế hay mã hóa. Việc phân tích, nắm bắt yêu cầu thường gặp các khó khăn như:

Các yêu cầu thường mang tính đặc thù của tổ chức đặt hàng nó, do đó nó thường khó hiểu, khó định nghĩa và không có chuẩn biểu diễn.

Các hệ thống thông tin lớn có nhiều người sử dụng thì các yêu cầu thường rất đa dạng và có các mức ưu tiên khác nhau, thậm chí mâu thuẫn lẫn nhau.

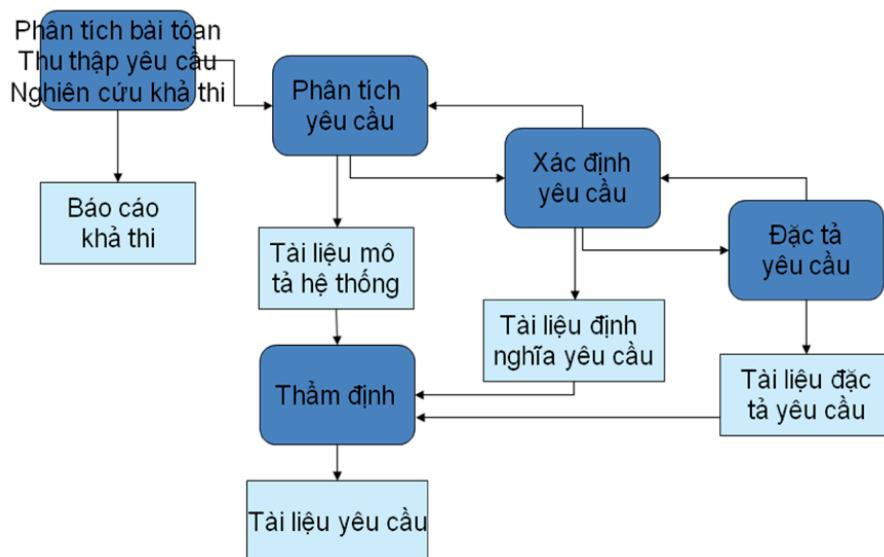
Người đặt hàng nhiều khi là các nhà quản lý, không phải là người dùng thực sự do đó việc phát biểu yêu cầu thường không chính xác.

Trong phân tích cần phân biệt giữa yêu cầu và mục tiêu của hệ thống. Yêu cầu là một đòi hỏi mà chúng ta có thể kiểm tra được còn mục tiêu là cái trừu tượng hơn mà chúng ta hướng tới. Ví dụ, giao diện của hệ thống phải thân thiện với người sử dụng là một mục tiêu và nó tương đối không quan trọng và khó kiểm tra. Có nghĩa là với một phát biểu chung chung như vậy thì khách hàng và nhà phát triển khó định ra được một ranh giới rõ ràng để nói rằng phần mềm đã thỏa mãn được đòi hỏi đó. Với một mục tiêu như vậy, một yêu cầu cho nhà phát triển có thể là giao diện đồ họa mà các lệnh phải được chọn bằng menu. Mục đích của giai đoạn phân tích là xác định rõ các yêu cầu của phần mềm cần phát triển. Tài liệu yêu cầu nên dễ hiểu với người dùng, đồng thời phải chặt chẽ để làm cơ sở cho hợp đồng và để cho người phát triển dựa vào đó để xây dựng phần mềm. Do đó yêu cầu thường được mô tả ở nhiều mức chi tiết khác nhau phục vụ cho các đối tượng đọc khác nhau. Các mức đó có thể là:

Định nghĩa (xác định) yêu cầu: mô tả một cách dễ hiểu, vắn tắt về yêu cầu, hướng vào đối tượng người đọc là người sử dụng, người quản lý, ...

Đặc tả yêu cầu: mô tả chi tiết về các yêu cầu, các ràng buộc của hệ thống, phải chính xác sao cho người đọc không hiểu nhầm yêu cầu, hướng vào đối tượng người đọc là các kỹ sư phần mềm (người phát triển), kỹ sư hệ thống (sẽ làm việc bảo trì), ...

Các tài liệu yêu cầu cần được thẩm định để đảm bảo thỏa mãn nhu cầu người dùng. Đây là công việc bắt buộc để đảm bảo chất lượng phần mềm. Đôi khi việc xác định đầy đủ yêu cầu trước khi phát triển hệ thống là không thực tế và khi đó việc xây dựng các bản mẫu để nắm bắt yêu cầu là cần thiết.



Hình 2. Các hoạt động phân tích và đặc tả yêu cầu

1) Nghiên cứu khả thi

Đây là giai đoạn có tầm quan trọng đặc biệt, vì nó liên quan đến việc lựa chọn giải pháp. Trong giai đoạn này người phân tích phải làm rõ được các điểm mạnh và điểm yếu của hệ thống cũ, đánh giá được mức độ, tầm quan trọng của từng vấn đề, định ra các vấn đề cần phải giải quyết (ví dụ: những dịch vụ mới, thời hạn đáp ứng, hiệu quả kinh tế). Sau đó người phân tích phải định ra một vài giải pháp có thể (sơ bộ) và so sánh cân nhắc các điểm tốt và không tốt của các giải pháp đó (như tính năng của hệ thống, giá cả cài đặt, bảo trì, việc đào tạo người sử dụng, ...). Đó là việc tìm ra một điểm cân bằng giữa nhu cầu và khả năng đáp ứng. Mọi dự án đều khả thi khi nguồn tài nguyên vô hạn và thời gian vô hạn. Nhưng việc xây dựng hệ thống lại phải làm với sự hạn hẹp về tài nguyên và khó (nếu không phải là không hiện thực) bảo đảm đúng ngày bàn giao. Phân tích khả thi và rủi ro có liên quan với nhau theo nhiều cách. Nếu rủi ro của dự án là lớn thì tính khả thi của việc chế tạo phần mềm có chất lượng sẽ bị giảm đi.

2) Phân tích yêu cầu

- Miền thông tin của vấn đề phải được biểu diễn lại và hiểu rõ.
- Các mô hình mô tả cho thông tin, chức năng và hành vi hệ thống cần phải được xây dựng.
- Các mô hình (và vấn đề) phải được phân hoạch theo cách để lộ ra các chi tiết theo kiểu phân tầng (hay cấp bậc).
- Qui trình phân tích phải đi từ thông tin bản chất hướng tới chi tiết cài đặt.

Bằng cách áp dụng những nguyên lý này, người phân tích tiếp cận tới vấn đề một cách hệ thống. Miền thông tin cần được xem xét sao cho người ta có thể hiểu rõ chức năng một cách đầy đủ. Các mô hình được dùng để cho việc trao đổi thông tin được dễ dàng theo một cách ngắn gọn. Việc phân hoạch vấn đề được sử dụng để làm giảm độ phức tạp. Những cách nhìn nhận cả

từ góc độ bản chất và góc độ cài đặt về phần mềm đều cần thiết để bao hàm được các ràng buộc logic do yêu cầu xử lý áp đặt nên cùng các ràng buộc vật lý do các phần tử hệ thống khác áp đặt nên.

3) Xác định yêu cầu

Xác định yêu cầu là mô tả trùu tượng về các dịch vụ mà hệ thống cần cung cấp và các ràng buộc mà hệ thống cần tuân thủ khi vận hành. Nó chỉ mô tả các hành vi bên ngoài của hệ thống mà không liên quan tới các chi tiết thiết kế. Yêu cầu nên được viết sao cho có thể hiểu mà không cần một kiến thức chuyên môn đặc biệt nào. Các yêu cầu được chia thành hai loại:

- Các yêu cầu chức năng: các dịch vụ mà hệ thống cần cung cấp.
- Các yêu cầu phi chức năng: các ràng buộc mà hệ thống cần tuân thủ.

4) Đặc tả yêu cầu

Tài liệu xác định yêu cầu là mô tả hướng khách hàng và được viết bởi ngôn ngữ của khách hàng. Khi đó có thể dùng ngôn ngữ tự nhiên và các khái niệm trùu tượng. Tài liệu đặc tả yêu cầu (đặc tả chức năng) là mô tả hướng người phát triển, là cơ sở của hợp đồng làm phần mềm. Nó không được phép mơ hồ, nếu không sẽ dẫn đến sự hiểu nhầm bởi khách hàng hoặc người phát triển. Với một yêu cầu mơ hồ thì người phát triển sẽ thực hiện nó một cách rẻ nhất còn khách hàng thì không muốn vậy. Do đó khách hàng có thể đòi hỏi sửa đổi chức năng phần mềm khi nó đã gần hoàn thiện khiến cho chi phí tăng và chậm thời điểm bàn giao. Chi phí cho sửa các sai sót trong phát biểu yêu cầu là rất lớn, đặc biệt là khi các sai sót này được phát hiện khi đã bắt đầu xây dựng hệ thống. Theo một số thống kê thì 85% mã phải viết lại do thay đổi yêu cầu và 12% lỗi phát hiện trong 3 năm đầu sử dụng là do đặc tả yêu cầu không chính xác. Do đó, việc đặc tả chính xác yêu cầu là mối quan tâm được đặt lên hàng đầu.

Có hai phương pháp đặc tả là:

- Đặc tả phi hình thức: là cách đặc tả bằng ngôn ngữ tự nhiên
- Đặc tả hình thức: là cách đặc tả bằng các ngôn ngữ nhân tạo (ngôn ngữ đặc tả), các công thức và biểu đồ

5) Thẩm định yêu cầu

Một khi các yêu cầu đã được thiết lập thì cần thẩm định xem chúng có thỏa mãn các nhu cầu của khách hàng hay không. Nếu thẩm định không được tiến hành chặt chẽ thì các sai sót có thể lan truyền sang các giai đoạn thiết kế và mã hóa khiến cho việc sửa đổi hệ thống trở nên rất tốn kém. Mục tiêu của thẩm định là kiểm tra xem yêu cầu mà người phân tích xác định có thỏa mãn 4 yếu tố sau không:

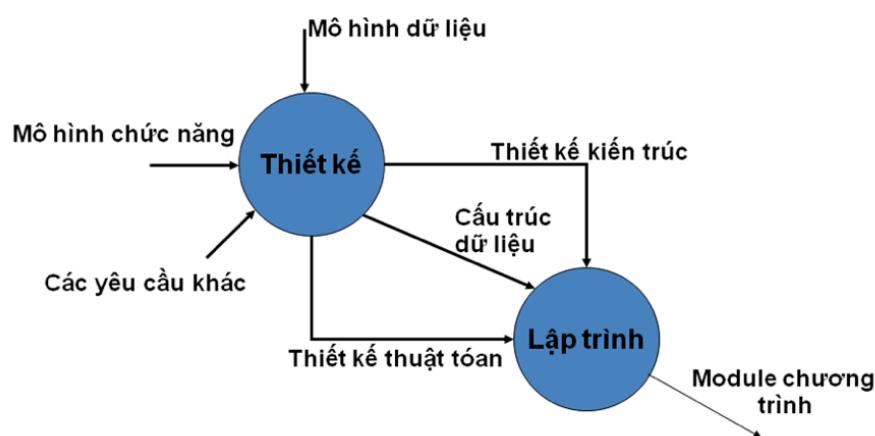
- Thỏa mãn nhu cầu của người dùng: cần phải thỏa mãn được các nhu cầu bản chất của người dùng (khách hàng).
- Các yêu cầu không được mâu thuẫn nhau: với các hệ thống lớn các yêu cầu rất đa dạng và có khả năng sẽ mâu thuẫn nhau. Khi đó người phân tích phải loại bỏ các yêu cầu (không chủ chốt) để sau cho các yêu cầu được mô tả trong tài liệu yêu cầu không mâu thuẫn nhau.
- Các yêu cầu phải đầy đủ: cần chứa mọi chức năng và ràng buộc mà người dùng đã nhầm đến.
- Các yêu cầu phải hiện thực: yêu cầu phải hiện thực về các khía cạnh kỹ thuật, kinh tế và pháp lý.

b. Thiết kế và mã hóa

1) Khái niệm

Có thể định nghĩa thiết kế là một quá trình áp dụng nhiều kỹ thuật và các nguyên lý để tạo ra mô hình của một thiết bị, một qui trình hay một hệ thống đủ chi tiết mà theo đó có thể chế tạo ra sản phẩm vật lý tương ứng với nó. 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. Mục tiêu thiết kế là để tạo ra một mô hình biểu diễn của một thực thể mà sau này sẽ được xây dựng. Mô hình chung của một thiết kế phần mềm là một đồ thị có hướng, các nút biểu diễn các thực thể có trong thiết kế, các liên kết biểu diễn các mối quan hệ giữa các thực thể đó. Hoạt động thiết kế là một loại hoạt động đặc biệt:

- Là một quá trình sáng tạo, đòi hỏi có kinh nghiệm, sự nhanh nhẹn và sáng tạo.
- Cần phải được thực hành và học bằng kinh nghiệm, bằng khảo sát các hệ đang tồn tại, chỉ học bằng sách vở là không đủ.



Hình 3. Hoạt động thiết kế phần mềm

Thiết kế là nơi chất lượng phần mềm được nuôi dưỡng trong quá trình phát triển: cung cấp cách biểu diễn phần mềm có thể được xác nhận về chất lượng, là cách duy nhất mà chúng ta có thể chuyển hóa một cách chính xác các yêu cầu của khách hàng thành sản phẩm hay hệ thống phần mềm cuối cùng. Thiết kế phần mềm là công cụ giao tiếp làm cơ sở để có thể mô tả một cách đầy đủ các dịch vụ của hệ thống, để quản lý các rủi ro và lựa chọn giải pháp thích hợp. Thiết kế phần mềm phục vụ như một nền tảng cho mọi bước công nghệ phần mềm và bảo trì. Không có thiết kế có nguy cơ sản sinh một hệ thống không ổn định - một hệ thống sẽ thất bại. Một hệ thống phần mềm rất khó xác định được chất lượng chừng nào chưa đến bước kiểm thử. Thiết kế tốt là một trong những bước quan trọng để đảm bảo chất lượng phần mềm.

2) Qui trình thiết kế

Thiết kế phần mềm là quá trình chuyển các đặc tả yêu cầu dịch vụ thông tin của hệ thống thành đặc tả hệ thống phần mềm. Thiết kế phần mềm trải qua một số giai đoạn chính sau:

1. Nghiên cứu để hiểu ra vấn đề.

Không hiểu rõ vấn đề thì không thể có được thiết kế hữu hiệu.

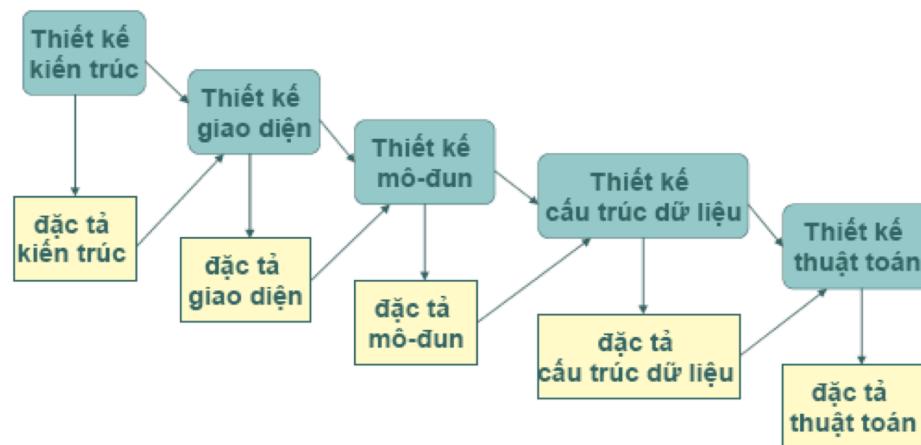
2. Chọn một (hay một số) giải pháp thiết kế và xác định các đặc điểm thô của nó.

Chọn giải pháp phụ thuộc vào kinh nghiệm của người thiết kế, vào các cấu kiện dùng lại được và vào sự đơn giản của các giải pháp kéo theo. Nếu các nhân tố khác là tương tự thì nên chọn giải pháp đơn giản nhất.

3. Mô tả trừu tượng cho mỗi nội dung trong giải pháp.

Trước khi tạo ra các tư liệu chính thức người thiết kế cần phải xây dựng một mô tả ban đầu sơ khai rồi chi tiết hóa nó. Các sai sót và khiếm khuyết trong mỗi mức thiết kế trước đó được phát hiện và phải được chỉnh sửa trước khi lập tư liệu thiết kế. Kết quả của mỗi hoạt động thiết kế là một đặc tả thiết kế. Đặc tả này có thể là một đặc tả trùu tượng, hình thức và được tạo ra để làm rõ các yêu cầu, nó cũng có thể là một đặc tả về một phần nào đó của hệ thống phải được thực hiện như thế nào. Khi quá trình thiết kế tiến triển thì các chi tiết được bổ sung vào đặc tả đó. Các kết quả cuối cùng là các đặc tả về các thuật toán và các cấu trúc dữ liệu được dùng làm cơ sở cho việc thực hiện hệ thống.

Các hoạt động thiết kế chính trong một hệ thống phần mềm lớn:



Hình 4. Qui trình thiết kế phần mềm

- Thiết kế kiến trúc: Xác định hệ thống tổng thể phần mềm bao gồm các hệ con và các quan hệ giữa chúng và ghi thành tài liệu.
- Đặc tả trùu tượng: các đặc tả trùu tượng cho mỗi hệ con về các dịch vụ mà nó cung cấp cũng như các ràng buộc chúng phải tuân thủ.
- Thiết kế giao diện: giao diện của từng hệ con với các hệ con khác được thiết kế và ghi thành tài liệu; đặc tả giao diện không được mơ hồ và cho phép sử dụng hệ con đó mà không cần biết về thiết kế nội tại của nó.
- Thiết kế các thành phần: các dịch vụ mà một hệ con cung cấp được phân chia cho các thành phần hợp thành của nó.
- Thiết kế cấu trúc dữ liệu: thiết kế chi tiết và đặc tả các cấu trúc dữ liệu (các mô hình vè thê giới thực cần xử lý) được dùng trong việc thực hiện hệ thống.
- Thiết kế thuật toán: các thuật toán được dùng cho các dịch vụ được thiết kế chi tiết và được đặc tả. Quá trình này được lặp lại cho đến khi các thành phần hợp thành của mỗi hệ con được xác định đều có thể ánh xạ trực tiếp vào các thành phần ngôn ngữ lập trình, chẳng hạn như các gói, các thủ tục và các hàm.

4) Chiến lược thiết kế

Do các hệ phần mềm lớn là phức tạp nên người ta thường dùng các phương pháp tiếp cận khác nhau trong việc thiết kế các phần khác nhau của một hệ thống. Chẳng có một chiến lược tốt nhất nào cho các dự án. Hai chiến lược thiết kế hiện đang được dùng rộng rãi trong việc phát triển phần mềm đó là thiết kế hướng chức năng và thiết kế hướng đối tượng. Mỗi chiến lược thiết kế đều có ưu và nhược điểm riêng phụ thuộc vào ứng dụng phát triển và nhóm phát triển phần mềm.

Cách tiếp cận hướng chức năng hay hướng đối tượng là bổ sung và hỗ trợ cho nhau chứ không phải là đối kháng nhau. Kỹ sư phần mềm sẽ chọn cách tiếp cận thích hợp nhất cho từng giai đoạn thiết kế.

Thiết kế hướng chức năng

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

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

Thiết kế hướng đối tượng

Hệ thống được nhìn nhận như một bộ các đối tượng (chứ không phải là bộ các chức năng). Hệ thống được phân tán, mỗi đối tượng có những thông tin trạng thái riêng của nó. Đối tượng là một bộ các thuộc tính xác định trạng thái của đối tượng đó và các phép toán của nó. Nó được thừa kế từ một vài lớp đối tượng lớp cao hơn, sao cho dễ định nghĩa nó chỉ cần nêu đủ các khác nhau giữa nó và các lớp cao hơn nó.

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

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

Thiết kế hướng đối tượng có ba đặc trưng:

- Vùng dữ liệu dùng chung là bị loại bỏ. Các đối tượng liên lạc với nhau bằng cách trao đổi thông báo chứ không phải bằng các biến dùng chung.
- Các đối tượng là các thực thể độc lập mà chúng sẵn sàng được thay đổi vì tất cả các trạng thái và các thông tin biểu diễn là chỉ ảnh hưởng trong phạm vi chính đối tượng đó thôi. Các thay đổi về biểu diễn thông tin có thể được thực hiện không cần sự tham khảo tới các đối tượng hệ thống khác.
- Các đối tượng có thể phân tán và có thể hành động tuần tự hoặc song song. Không cần có quyết định về tính song song ngay từ một giai đoạn sớm của quá trình thiết kế.

Các ưu điểm:

- Dễ bảo trì vì các đối tượng là độc lập. Các đối tượng có thể hiểu và cải biên như là một thực thể độc lập. Thay đổi trong thực hiện một đối tượng hoặc thêm các dịch vụ sẽ không làm ảnh hưởng tới các đối tượng hệ thống khác.
- Các đối tượng là các thành phần dùng lại được thích hợp (do tính độc lập của chúng). Một thiết kế có thể dùng lại được các đối tượng đã được thiết kế trong các bản thiết kế trước đó.

- Đối với một vài lớp hệ thống, có một phản ánh rõ ràng giữa các thực thể có thực (chẳng hạn như các thành phần phần cứng) với các đối tượng điều khiển nó trong hệ thống. Điều này cải thiện được tính dễ hiểu của thiết kế.

Nhược điểm:

- Việc xác định các đối tượng hệ thống cho thật sự thích hợp là khó khăn. Cách nhìn tự nhiên nhiều hệ thống là cách nhìn chức năng và việc thích nghi với cách nhìn hướng đối tượng đôi khi là khó khăn.
- Phương pháp thiết kế hướng đối tượng đang có nhiều thay đổi.

c. Mã hóa (Lập trình)

Bước lập trình bắt đầu sau khi thiết kế chi tiết đã được xác định, xét duyệt và sửa đổi nếu cần. Về lý thuyết, việc sinh chương trình gốc từ một đặc tả chi tiết nên là trực tiếp. Để dịch thiết kế sang chương trình, cần đưa ra một chỉ dẫn về việc một ngôn ngữ lập trình phản xạ gần gũi đến mức nào cho một biểu diễn thiết kế. Một ngôn ngữ cài đặt trực tiếp cho các kết cấu có cấu trúc, các cấu trúc dữ liệu phức tạp, vào/ra đặc biệt, khả năng thao tác bit, và các kết cấu hướng sự vật sẽ làm cho việc dịch từ thiết kế sang chương trình gốc dễ hơn nhiều (nếu các thuộc tính này được xác định trong thiết kế).

d. Kiểm thử

Xác minh và thẩm định một hệ phần mềm là một quá trình liên tục xuyên suốt mọi giai đoạn của quá trình phần mềm. Xác minh và thẩm định là từ chung cho các quá trình kiểm tra để đảm bảo rằng phần mềm thỏa mãn các yêu cầu của chúng và các yêu cầu đó thỏa mãn các nhu cầu của người sắm phần mềm.

Xác minh và thẩm định là một quá trình kéo dài suốt vòng đời. Nó bắt đầu khi duyệt xét yêu cầu. Xác minh và thẩm định có hai mục tiêu:

- i) Phát hiện các khiếm khuyết trong hệ thống.
- ii) Đánh giá xem hệ thống liệu có dùng được hay không?

Sự khác nhau giữa xác minh và thẩm định là:

i) Thẩm định: Xem xét cái được xây dựng có là sản phẩm đúng không? Tức là kiểm tra xem chương trình có được như mong đợi của người dùng hay không.

ii) Xác minh: Xem xét cái được xây dựng có đúng là sản phẩm không? Như thế, xác minh là kiểm tra chương trình có phù hợp với đặc tả hay không.

Như trên, kiểm thử là một quá trình của tìm kiếm lỗi. Một kiểm thử tốt có khả năng cao tìm kiếm các lỗi chưa được phát hiện. Một kiểm thử thành công là kiểm thử tìm ra các lỗi mới, một kiểm thử tồi là kiểm tra mà không tìm được lỗi.

Có hai kiểu lỗi trong ứng dụng và các kiểm thử tốt sẽ xác định cả hai loại đó. Cụ thể:

- Không làm những điều cần phải làm: lỗi bỏ sót thường xuất hiện đối với ứng dụng mới được phát triển.
- Làm những điều không cần làm: lỗi của lệnh đã cư trú trước trong các ứng dụng bảo trì.

Các kiểm thử có mặt tại các mức khác nhau và được tiến hành bởi các cá nhân khác nhau trong quá trình phát triển ứng dụng. Các kiểm thử được tiến hành bởi đội ngũ dự án và kiểm thử được tiến hành bởi các cơ quan bên ngoài để chấp nhận ứng dụng.

Các kiểm thử của đội dự án được gọi là kiểm tra phát triển (Development test).

Kiểm thử phát triển bao gồm kiểm tra đơn vị, kiểm thử hệ thống con, kiểm thử tích hợp và các kiểm thử hệ thống.

Kiểm thử đơn vị (Unit test) được tiến hành cho mỗi đơn vị mã nhỏ nhất.

Kiểm thử tích hợp (Subsystem integration test) kiểm thử mặt logic và xử lý cho phù hợp của các khối, kiểm thử việc truyền tin giữa chúng.

Kiểm thử hệ thống (System test) đánh giá các đặc tả chức năng có được đáp ứng, các thao tác giao diện người có giống thiết kế không, và các công việc của ứng dụng trong môi trường thao tác mong muốn, đối với các ràng buộc.

Kiểm thử được lặp lại cho đến khi không còn lỗi, hoặc đạt được mức độ chấp nhận.

Bước đầu tiên của xử lý kiểm thử, đầu vào kiểm thử, cấu hình và mã ứng dụng được đòi hỏi để tạo các kiểm thử.

Bước thứ hai là so sánh các kết quả kiểm thử với kết quả dự tính và định lượng sự khác biệt.

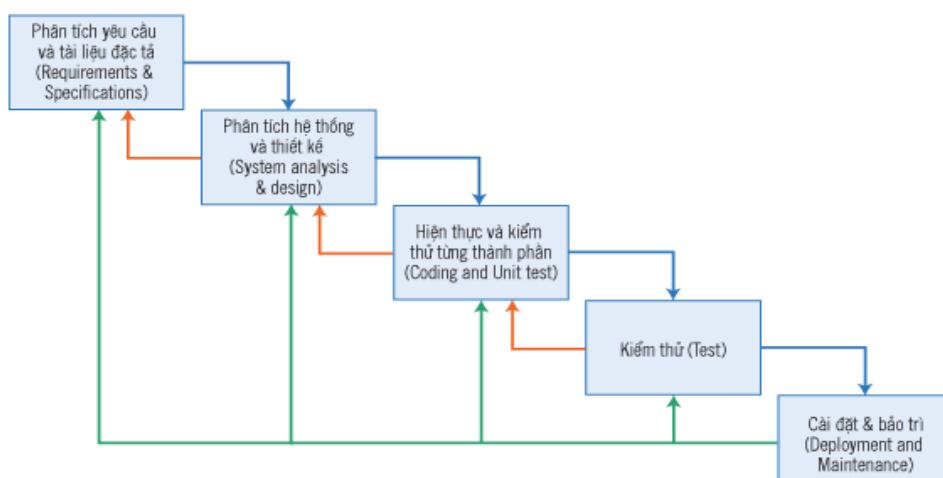
Bước tiếp theo là loại trừ các lỗi, hoặc gỡ rối mã. Khi việc mã hoá lại hoàn thành, kiểm thử lại được tiến hành.

Quá trình tiến hành kiểm thử bắt đầu từ khi thiết kế. Cộng tác viên thực hiện việc kiểm thử nên có khả năng của phân tích viên và lập trình viên để có thể hiểu các đòi hỏi của ứng dụng và kiểu cách tiến hành kiểm thử. Chương trình càng lớn và phức tạp thì càng cần người có kinh nghiệm, đôi khi cần có một đội ngũ kiểm thử. Đội ngũ kiểm thử sử dụng yêu cầu chức năng từ giai đoạn phân tích và đặc tả chương trình, đặc tả thiết kế từ giai đoạn thiết kế như là đầu vào để phát triển chiến lược kiểm thử hệ thống. Khi chiến lược đã được hoàn tất, các buổi thảo luận được tiến hành để kiểm thử lại chiến lược và thông báo tới toàn thể đội. Các nhiệm vụ tại các mức sẽ được xác định. Thời gian được ước lượng. Đội ngũ kiểm thử làm việc độc lập và song song với đội ngũ lập trình. Họ làm việc với quản trị CSDL để phát triển cơ sở dữ liệu mà có thể hỗ trợ tất cả các mức kiểm thử. Đối với kiểm thử đơn vị, đội kiểm thử kiểm thử kết quả, các chấp nhận các mô-đun và chương trình cho kiểm thử tích hợp (integration test). Đội ngũ kiểm thử tiến hành và định lượng các kiểm thử tích hợp và kiểm thử hệ thống.

III. Các mô hình qui trình phần mềm

1. Mô hình thác nước (WaterFall Model)

Đôi khi còn được gọi là mô hình tuần tự tuyến tính hay mô hình vòng đồi cỏ điền, mô hình này đề xuất một cách tiếp cận tuần tự, có hệ thống cho việc phát triển phần mềm bắt đầu từ mức hệ thống và xuyên suốt việc phân tích, thiết kế, mã hóa, kiểm thử và hỗ trợ.



Hình 5. Mô hình thác nước (Waterfall Model)

Bởi vì phần mềm bao giờ cũng là một phần của hệ thống (hay nghiệp vụ) lớn hơn nên công việc bắt đầu từ việc thiết lập yêu cầu cho mọi phần tử hệ thống và rồi cấp phát một tập con các

yêu cầu đó cho phần mềm. Quan điểm hệ thống này là điều chỉnh bản chất khi phần mềm phải tương tác với các thành phần khác như phần cứng, con người và cơ sở dữ liệu. Công nghệ và phân tích hệ thống bao gồm việc thu thập yêu cầu ở mức hệ thống với một lượng nhỏ phân tích và thiết kế ở mức định. Công nghệ thông tin bao gồm việc thu thập yêu cầu ở mức nghiệp vụ chiến lược và mức lĩnh vực nghiệp vụ.

Phân tích yêu cầu phần mềm: Phân tích yêu cầu được tập trung việc thu thập và phân tích các thông tin cần cho phần mềm, các chức năng cần phải thực hiện, hiệu năng cần có và các giao diện cho người sử dụng. Kết quả của phân tích là tư liệu về yêu cầu cho hệ thống và phần mềm (đặc tả yêu cầu) để khách hàng duyệt lại và dùng làm tài liệu cho người phát triển.

Thiết kế: Thiết kế phần mềm thực tế là một qui trình nhiều bước, tập trung vào bốn thuộc tính phân biệt của chương trình: cấu trúc dữ liệu, kiến trúc phần mềm, biểu diễn giao diện và chi tiết thủ tục (thuật toán). Qui trình thiết kế dịch các yêu cầu thành một biểu diễn của phần mềm có thể được định giá về chất lượng trước khi giai đoạn mã hoá bắt đầu. Giống như các yêu cầu, việc thiết kế phải được lập tài liệu và trở thành một phần của cấu hình phần mềm.

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

Kiểm thử: Một khi mã đã được sinh ra thì việc kiểm thử chương trình bắt đầu. Qui trình kiểm thử tập trung vào logic bên trong của phần mềm, đảm bảo rằng tất cả các câu lệnh đều được kiểm thử, và vào chức năng bên ngoài; tức là tiến hành các kiểm thử để làm lộ ra các lỗi và đảm bảo những đầu vào được định nghĩa sẽ cho kết quả thực tế thỏa mãn với kết quả mong đợi.

Vận hành và bảo trì: Phần mềm chắc chắn sẽ phải trải qua những thay đổi sau khi nó được bàn giao cho khách hàng (một ngoại lệ có thể là những phần mềm nhúng). Thay đổi sẽ xuất hiện khi lỗi xuất hiện, vì phần mềm phải thích ứng với những thay đổi trong môi trường bên ngoài (chẳng hạn như sự thay đổi do hệ điều hành mới hay thiết bị ngoại vi mới), hay bởi vì khách hàng yêu cầu cải tiến chức năng hay hiệu năng sản phẩm phần mềm. Việc bảo trì phần mềm phải áp dụng lại các bước vòng đời nói trên cho chương trình hiện tại chứ không phải chương trình mới.

Mô hình tuần tự tuyến tính là mô hình cũ nhất và được sử dụng rộng rãi nhất cho công nghệ phần mềm. Tuy nhiên, những chỉ trích về mô hình này đã làm cho những người tích cực ủng hộ mô hình này phải đặt vấn đề về tính hiệu quả của nó. Một số các vấn đề thỉnh thoảng gặp phải khi dùng mô hình này:

1. Các dự án thực tế hiếm khi tuân theo dòng chảy tuần tự mà mô hình đề nghị. Mặc dù mô hình tuyến tính có thể cho phép lặp, nhưng điều đó chỉ làm gián tiếp. Kết quả là những thay đổi có thể gây ra lộn xộn khi tổ chức tiến hành.

2. Khách hàng thường khó phát biểu mọi yêu cầu một cách tường minh. Mô hình tuần tự tuyến tính đòi hỏi điều này và thường khó thích hợp với tính dễ thay đổi tồn tại ngay từ lúc đầu của nhiều dự án.

3. Khách hàng cần phải kiên nhẫn. Phần mềm chỉ có được vào lúc cuối của thời gian dự án. Một sai lầm nào đó nếu đến khi có chương trình làm việc mới phát hiện ra, có thể sẽ là một thảm họa.

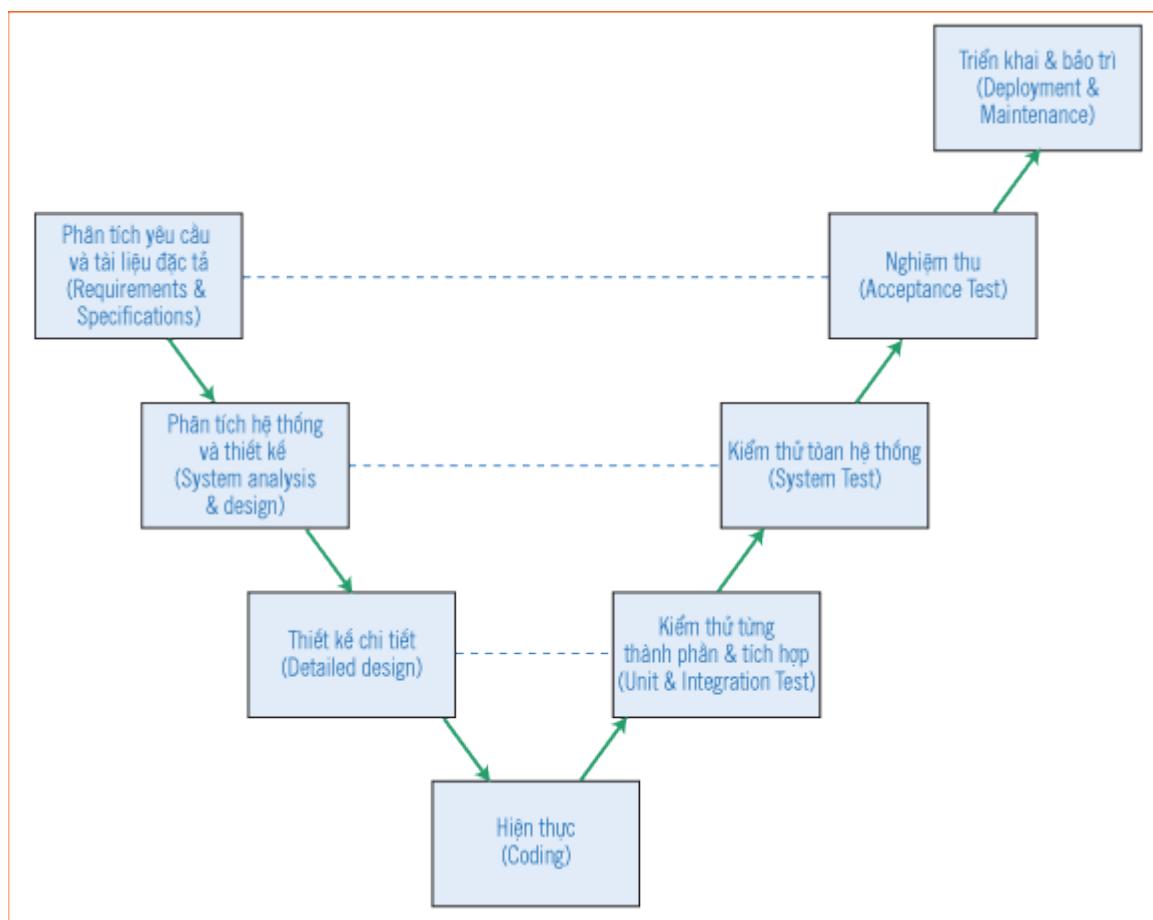
Trong một phân tích thú vị về các dự án hiện tại, Brada thấy rằng bản chất tuyến tính của vòng đời cỏ điền dẫn tới "các trạng thái nghẽn" mà trong đó một số thành viên tổ dự án phải đợi cho các thành viên khác của tổ hoàn thành các nhiệm vụ phụ thuộc. Trong thực tế, thời gian mất

cho việc chờ đợi có thể vượt quá thời gian dành cho công việc sản xuất. Trạng thái nghẽn có khuynh hướng phô biến vào thời điểm bắt đầu và cuối của qui trình tuần tự tuyến tính.

Tuy nhiên, mô hình vòng đời cổ điển có một vị trí quan trọng và xác định trong công việc về công nghệ phần mềm. Nó đưa ra một tiêu bản trong đó có thể bố trí các phương pháp cho phân tích, thiết kế, mã hóa, kiểm thử và bảo trì. Bên cạnh đó, vòng đời cổ điển vẫn còn là một mô hình thủ tục được dùng rộng rãi cho công nghệ phần mềm. Trong khi còn một số điểm yếu, nó vẫn tốt hơn đáng kể nếu so với cách tiếp cận ngẫu nhiên tới việc phát triển phần mềm.

2. Mô hình chữ V (V-Shaped Model)

Trong mô hình Waterfall, kiểm thử được thực hiện trong một giai đoạn riêng biệt. Còn với mô hình chữ V, toàn bộ qui trình được chia thành hai nhóm giai đoạn tương ứng nhau: phát triển và kiểm thử. Mỗi giai đoạn phát triển sẽ kết hợp với một giai đoạn kiểm thử tương ứng như được minh họa trong hình sau:



Hình 6. Mô hình chữ V (V-Shaped Model)

Tinh thần chủ đạo của V-model là các hoạt động kiểm thử phải được tiến hành song song (theo khả năng có thể) ngay từ đầu chu trình cùng với các hoạt động phát triển. Ví dụ, các hoạt động cho việc lập kế hoạch kiểm thử toàn hệ thống có thể được thực hiện song song với các hoạt động phân tích và thiết kế hệ thống.

Ưu điểm:

Các hoạt động kiểm thử được chú trọng và thực hiện song song với các hoạt động liên quan đến đặc tả yêu cầu và thiết kế. Hay nói cách khác, mô hình này khuyến khích các hoạt động liên quan đến kế hoạch kiểm thử được tiến hành sớm trong chu kỳ phát triển, không phải đợi đến lúc kết thúc giai đoạn hiện thực.

Nhược điểm:

Đòi hỏi tất cả yêu cầu phần mềm phải được xác định rõ ràng ngay từ đầu dự án. Nhưng đa số dự án thực tế cho thấy yêu cầu phần mềm thường ẩn chứa không nhiều thì ít những điểm không chắc chắn.

Một thực tế là các dự án hiếm khi được thực hiện đầy đủ các bước trong suốt chu kỳ dự án. Đặc biệt là giai đoạn kiểm thử khi gần đến ngày giao hàng chặng hạn, nếu có trực trặc xảy ra do yêu cầu phần mềm không rõ ràng hay thiết kế có lỗi, xu hướng là mã nguồn được sửa đổi trực tiếp mà không qua các bước bổ sung theo đúng mô hình, nên dẫn đến bản đặc tả phần mềm cũng như một số sản phẩm trung gian khác như bản thiết kế, cho dù có được cập nhật sau này cũng có thể không phản ánh đầy đủ những gì đã được sửa đổi trong mã nguồn.

Người sử dụng không có cơ hội tham gia trong suốt thời gian của các giai đoạn trung gian từ thiết kế cho đến kiểm thử. Đặc biệt với những dự án lớn, người sử dụng chỉ có thể nhận ra rằng hệ thống phần mềm không phù hợp cho nhu cầu của họ vào thời điểm cuối dự án.

Nói chung, mô hình này thường ẩn chứa nhiều rủi ro mà chỉ có thể phát hiện ở giai đoạn cuối cùng và chi phí để sửa chữa có thể rất cao.

Phạm vi ứng dụng:

Yêu cầu được định nghĩa rất rõ ràng, chi tiết và hầu như không thay đổi, thường xuất phát từ sản phẩm đã đạt mức ổn định.

Yêu cầu mới bổ sung (nếu có) cũng sớm được xác định rõ ràng, đầy đủ từ đầu dự án.

Đội ngũ thực hiện quen thuộc và hiểu rõ tất cả yêu cầu của dự án và có nhiều kinh nghiệm với các công nghệ được dùng để phát triển sản phẩm.

3. Mô hình bản mẫu (Prototyping Model)

Thông thường khách hàng đã xác định một tập các mục tiêu tổng quát cho phần mềm, nhưng còn chưa xác định chi tiết các yêu cầu đầu vào, xử lý và đầu ra. Trong các trường hợp khác, người phát triển có thể không chắc về tính hiệu quả của thuật toán, việc thích nghi hệ điều hành hay giao diện tương tác người máy cần có. Trong những trường hợp này và nhiều trường hợp khác, *mô hình bản mẫu* có thể được xem là cách tiếp cận tốt nhất.

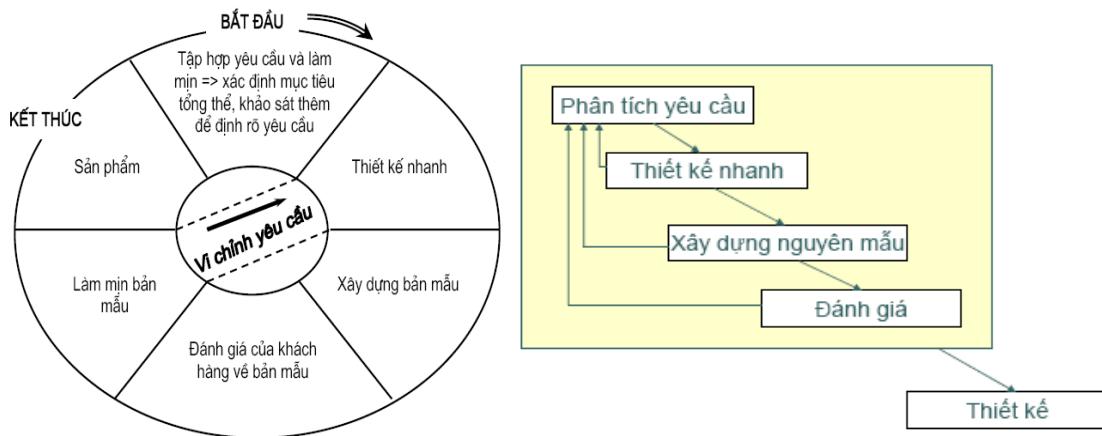


Hình 7. Áp dụng mô hình bản mẫu

Mục tiêu của việc phát triển dựa vào mẫu thử là giải quyết 2 hạn chế đầu tiên của mô hình thác nước. Ý tưởng cơ bản ở đây là thay cho việc chốt lại yêu cầu trước khi thiết kế hay code, một mẫu thử có tính sử dụng 1 lần được xây dựng để hiểu yêu cầu. Mẫu thử được xây dựng dựa vào những yêu cầu được biết hiện tại. Việc phát triển mẫu thử này dĩ nhiên cũng trải qua các giai đoạn design, coding, testing. Nhưng mỗi pha không được thực hiện một cách chu đáo. Bằng việc sử dụng mẫu thử, khách hàng sẽ có được cảm giác một hệ thống thực sự, vì khi tương tác với mẫu thử, khách hàng sẽ biết được rõ hơn những yêu cầu của hệ thống mong muốn.

Qui trình được bắt đầu bằng việc thu thập yêu cầu với sự có mặt của đại diện của cả phía phát triển lẫn khách hàng nhằm định ra mục tiêu tổng thể của hệ thống phần mềm sau này, đồng thời ghi nhận tất cả những yêu cầu có thể biết được và sơ lược những nhóm yêu cầu nào cần phải được làm rõ. Sau đó, thực hiện thiết kế nhanh tập trung chuyển tải những khía cạnh thông qua prototype để khách hàng có thể hình dung, đánh giá giúp hoàn chỉnh yêu cầu cho toàn hệ thống phần mềm. Việc này không những giúp tinh chỉnh yêu cầu, mà đồng thời giúp cho đội ngũ phát

triển thông hiểu hơn những gì cần được phát triển. Tiếp theo sau giai đoạn làm prototype này có thể là một chu trình theo mô hình waterfall hay cũng có thể là mô hình khác. Lưu ý rằng prototype thường được làm thật nhanh trong thời gian ngắn nên không được xây dựng trên cùng môi trường và công cụ phát triển của giai đoạn xây dựng phần mềm thực sự sau này. Prototype không đặt ra mục tiêu tái sử dụng cho giai đoạn phát triển thực sự sau đó.



Hình 8. Mô hình bản mẫu

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

Tóm lại, mô hình bản mẫu có những ưu, nhược điểm sau:

Ưu điểm:

Người sử dụng sớm hình dung ra chức năng và đặc điểm của hệ thống thông qua giao diện của ứng dụng.

Cải thiện sự liên lạc giữa nhà phát triển và người sử dụng.

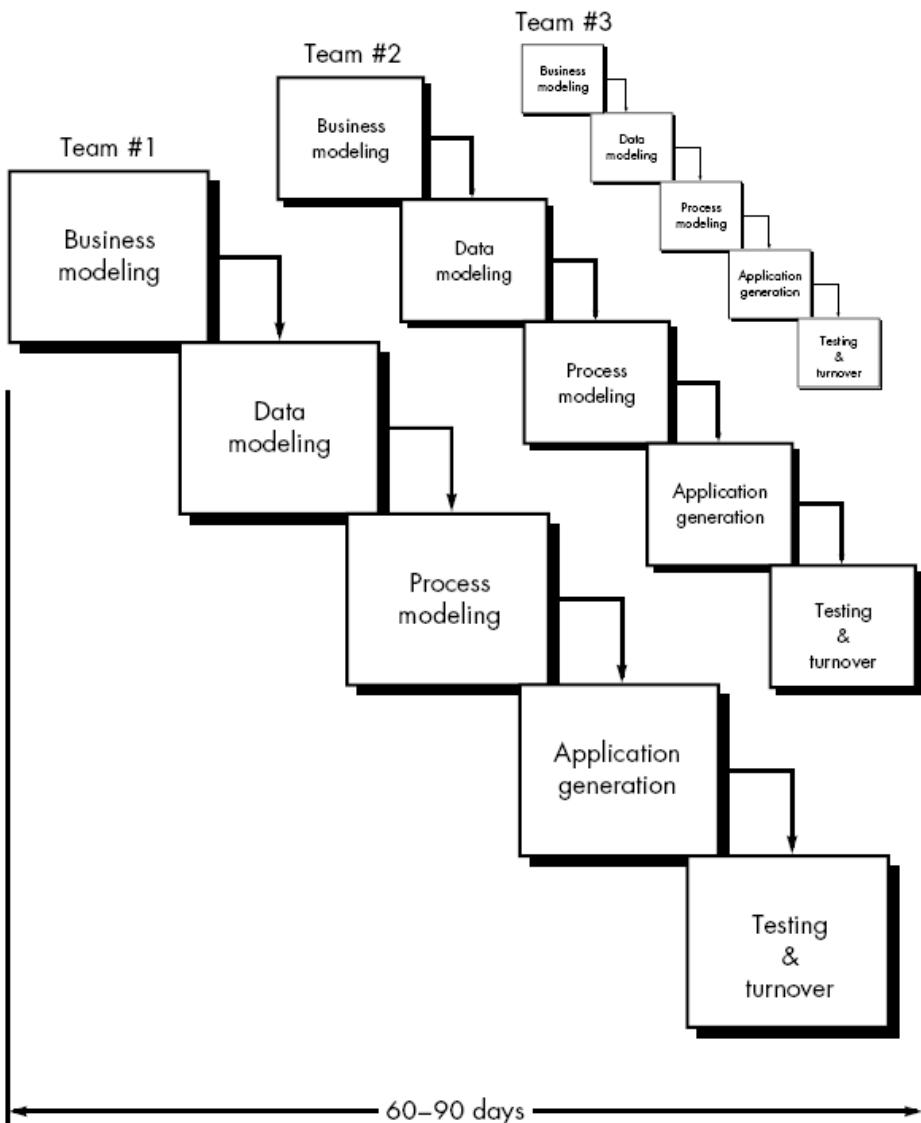
Nhược điểm:

Khi mẫu thử không chuyển tải được hết các chức năng, đặc điểm của hệ thống phần mềm thì người sử dụng có thể thất vọng và mất đi sự quan tâm đến hệ thống sẽ được phát triển.

Mẫu thử thường được làm nhanh, thậm chí vội vàng theo kiểu “làm – sửa” và có thể thiếu sự phân tích đánh giá một cách cẩn thận tất cả khía cạnh liên quan đến hệ thống cuối cùng.

Mặc dù có thể có một vài vấn đề, nhưng bản mẫu có thể là một mô hình hiệu quả cho công nghệ phần mềm. Chìa khóa là xác định các qui tắc ngay lúc bắt đầu; tức là khách hàng và nhà phát triển cả hai cần thống nhất rằng bản mẫu được xây dựng để phục vụ việc tinh chỉnh yêu cầu. Sau đó nó phải được vứt đi (ít nhất cũng một phần) và phần mềm thực tế được công nghệ hướng đến chất lượng và khả năng bảo trì.

4. Mô hình RAD (Rapid Application Development Model)



Hình 9. Mô hình phát triển ứng dụng nhanh (RAD Model)

Phát triển ứng dụng nhanh (*RAD – Rapid Application Development*) là mô hình qui trình phát triển phần mềm tăng dần mà nhấn mạnh đến vòng đời phát triển cực ngắn. Mô hình RAD là một sự thích nghi “mức cao” của mô hình tuần tự tuyến tính mà trong đó phát triển nhanh đạt được bằng cách sử dụng việc xây dựng dựa trên thành phần. Nếu các yêu cầu được hiểu tốt rồi để tạo một “hệ thống chức năng đầy đủ” trong khoảng thời gian rất ngắn (khoảng 60 đến 90 ngày) gồm các giai đoạn sau:

Mô hình hóa nghiệp vụ: Luồng thông tin giữa các chức năng business được mô hình hóa bằng cách trả lời các câu hỏi sau: Qui trình nghiệp vụ dẫn đến thông tin nào? Thông tin nào được sinh ra? Ai sinh ra? Thông tin đặt ở đâu? Ai xử lý nó?

Mô hình hóa dữ liệu: Luồng thông tin được định nghĩa như một phần của giai đoạn mô hình hóa nghiệp vụ. Các đặc tính (gọi là các thuộc tính) của mỗi đối tượng được định danh và quan hệ giữa các đối tượng được định nghĩa.

Mô hình hóa xử lý: Các đối tượng được xác định trong giai đoạn mô hình hóa dữ liệu được biến đổi để đạt được luồng thông tin cần thiết cho việc cài đặt chức năng nghiệp vụ. Mô tả các thao tác xử lý để thêm, sửa đổi, xóa hoặc xây dựng lại một đối tượng dữ liệu.

Sinh ứng dụng: RAD giả thiết sử dụng kỹ thuật thế hệ 4. Thay vì sử dụng ngôn ngữ lập trình thế hệ 3, RAD sử dụng lại các thành phần của chương trình đã tồn tại hoặc tạo ra các thành phần có thể sử dụng lại (khi cần thiết). Trong mọi trường hợp, các công cụ tự động được sử dụng để thuận tiện cho việc xây dựng phần mềm.

Kiểm thử và chuyển giao: Vì qui trình RAD nhấn mạnh việc tái sử dụng, nên nhiều thành phần chương trình đã được kiểm thử rồi. Điều này làm giảm thời gian kiểm thử. Tuy nhiên, các thành phần mới cần được kiểm thử và tất cả các giao diện cần được kiểm thử đầy đủ.

Như vậy, nếu một ứng dụng nghiệp vụ có thể được mô-đun hóa theo cách mà cho phép mỗi chức năng chính được hoàn thành dưới 3 tháng (sử dụng cách tiếp cận đã mô tả trước đây), thì có thể ứng dụng mô RAD. Mỗi chức năng chính được giải quyết bởi một nhóm RAD riêng biệt và sau đó tích hợp tạo thành một hệ thống tổng thể.

Cũng giống các mô hình qui trình khác, cách tiếp cận RAD có những nhược điểm:

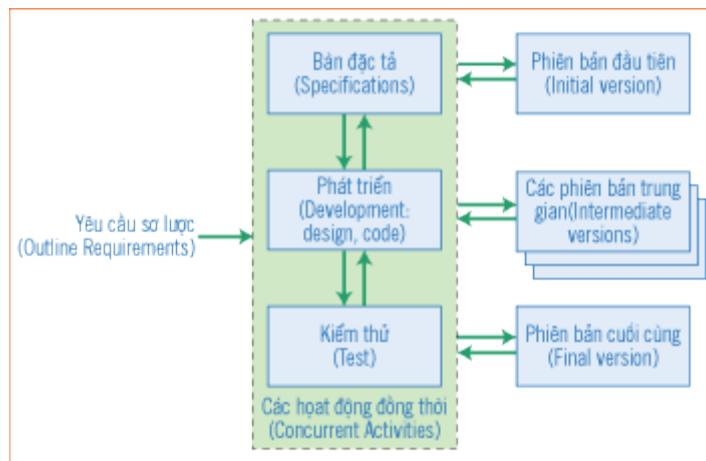
- Đối với dự án qui mô lớn, RAD yêu cầu đủ nguồn lực con người để tạo số nhóm RAD thích hợp.
- RAD yêu cầu nhà phát triển và khách hàng cam kết các hoạt động nhanh cần thiết để có được một hệ thống hoàn thành trong một khoảng thời gian được rút ngắn. Nếu thiếu sự cam kết từ các thành viên, các dự án RAD sẽ thất bại.
- Không phải tất cả các kiểu ứng dụng đều phù hợp với mô hình RAD. Nếu một hệ thống không thể mô-đun hóa một cách hợp lý, việc xây dựng các thành phần cần thiết cho RAD sẽ khó giải quyết. Nếu hiệu suất cao là vấn đề và hiệu suất đạt được thông qua việc điều chỉnh giao diện cho các thành phần của hệ thống, thì cách tiếp cận RAD có thể không làm việc được.
- RAD là không phù hợp khi các rủi ro kỹ thuật cao. Điều này xảy ra khi một ứng dụng mới yêu cầu sử dụng kỹ thuật mới hoặc khi phần mềm mới đòi hỏi mức độ cao về khả năng tương tác với các chương trình máy tính đang tồn tại.

5. Mô hình qui trình phần mềm tiến hóa

Ngày càng có nhiều công nhận rằng phần mềm, cũng giống như tất cả các hệ thống phức tạp, tiến hóa qua một chu kỳ thời gian. Các yêu cầu nghiệp vụ và sản phẩm thường thay đổi khi tiến hành phát triển, việc tạo một đường thẳng đến sản phẩm cuối là không thể; nhưng một phiên bản được giới hạn cần phải được giới thiệu để đáp ứng sự cạnh tranh hoặc áp lực kinh doanh; một bộ các yêu cầu chính của sản phẩm hoặc hệ thống được hiểu rõ, nhưng các chi tiết của các chức năng mở rộng của sản phẩm hoặc hệ thống vẫn chưa được xác định. Trong những trường hợp đó và những tình huống tương tự, các kỹ sư phần mềm cần một mô hình qui trình sao cho có được các thiết kế rõ ràng phù hợp với sản phẩm tiến hóa theo thời gian.

Mô hình tuyến tính tuần tự được thiết kế cho sự phát triển theo đường thẳng. Về bản chất, cách tiếp cận theo thác nước này giả thiết rằng một hệ thống hoàn chỉnh sẽ được giao sau khi trình tự tuyến tính hoàn thành. Mô hình nguyên mẫu được thiết kế để hỗ trợ khách hàng (hoặc người phát triển) trong việc tìm hiểu yêu cầu. Nói chung, nó không được thiết kế để cung cấp một hệ thống sản xuất. Bản chất tiến hóa của phần mềm không được xem xét trong các mô hình Công nghệ phần mềm cổ điển.

Các mô hình tiến hóa là lặp lại. Theo một khía cạnh nào đó, đặc trưng của các mô hình này là cho phép các kỹ sư phần mềm phát triển ngày càng hoàn thiện hơn các phiên bản phần mềm.



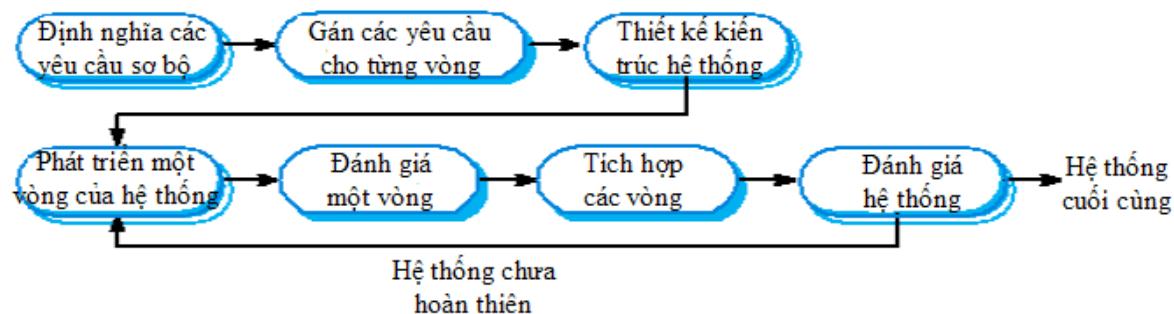
Hình 10. Mô hình tiến hóa (Evolutionary Model)

Mô hình tiến hóa cho phép khách hàng tham gia sâu hơn vào quá trình phát triển, nhờ đó sản phẩm cuối cùng thường phản ánh chính xác mong muốn của khách hàng. Tuy nhiên, mô hình này cũng có các nhược điểm là:

- Khó khăn trong việc thiết kế: Việc phát triển qua nhiều phiên bản có thể phá vỡ kiến trúc tổng thể của phần mềm.
- Khó khăn trong việc quản lí: Các nhà quản lí thích nhìn thấy sản phẩm làm ra trong từng gian đoạn để tiện cho việc quản lí tiến độ. Ngoài ra, các tài liệu mô tả cho từng phiên bản thường không được lập đầy đủ.
- Khó khăn do khách hàng gây ra: Khách hàng có thể nhầm tưởng rằng một bản mẫu có thể tốt gần như sản phẩm thật. Trong thực tế từ bản mẫu đến sản phẩm cuối cùng là một khoảng cách xa. Ngoài ra, khách hàng có xu hướng đưa thêm vào những yêu cầu không cần thiết.
- Khó khăn về địa lý: Mô hình tiến hóa đòi hỏi đội dự án phải ngồi gần khách hàng. Các dự án outsourcing khó có thể đáp ứng yêu cầu này.

Theo Ian Sommerville trong cuốn “Software Engineering”, mô hình tiến hóa là mô hình phù hợp nhất cho các dự án vừa và nhỏ (dưới 500 000 dòng code). Các dự án lớn và phức tạp nên sử dụng một mô hình kết hợp giữa mô hình thác nước và tiến hóa. Trong đó, các bản mẫu được dùng để làm rõ các yêu cầu của khách hàng. Các yêu cầu đã rõ ràng được tiếp tục phát triển theo mô hình thác nước. Các yêu cầu chưa rõ ràng có thể sử dụng mô hình khám phá và phát triển.

6. Mô hình lặp (Iteration Model)



Hình 11. Mô hình lặp (Iteration Model)

Mô hình lặp và tăng dần có lúc được hiểu là một. Tuy nhiên, ta có thể phân biệt ít nhiều sự khác biệt. Trước tiên, hai mô hình này đều có điểm giống nhau là đều dựa trên tinh thần của mô hình tiến hóa, và có thêm đặc điểm nhằm đến việc cung cấp một phần hệ thống để khách hàng có thể đưa vào sử dụng trong môi trường hoạt động sản xuất thực sự mà không cần chờ cho đến

khi toàn bộ hệ thống được hoàn thành (trong mô hình mẫu hay tiến hóa, các phiên bản mẫu hay trung gian đều không nhắm đến đưa vào vận hành thực sự cho khách hàng, trừ phiên bản cuối cùng). Để khách hàng có thể sử dụng, mỗi phiên bản đều phải được thực hiện như một qui trình đầy đủ các công việc từ phân tích yêu cầu với khả năng bổ sung hay thay đổi, thiết kế, hiện thực cho đến kiểm nghiệm và có thể xem như một qui trình (chu trình) con. Các chu trình con có thể sử dụng các mô hình khác nhau (thông thường là waterfall). Mục tiêu của phiên bản đầu tiên là phát triển phần lõi và nhóm các chức năng quan trọng. Sau mỗi phiên bản được đưa vào sử dụng, các kết quả đánh giá sẽ được phản hồi và lập kế hoạch cho chu trình con của phiên bản tiếp theo để thực hiện:

- Những thay đổi cho phiên bản trước đó nhằm đáp ứng nhu cầu khách hàng tốt hơn.
- Có thể thêm những chức năng hoặc đặc điểm bổ sung.

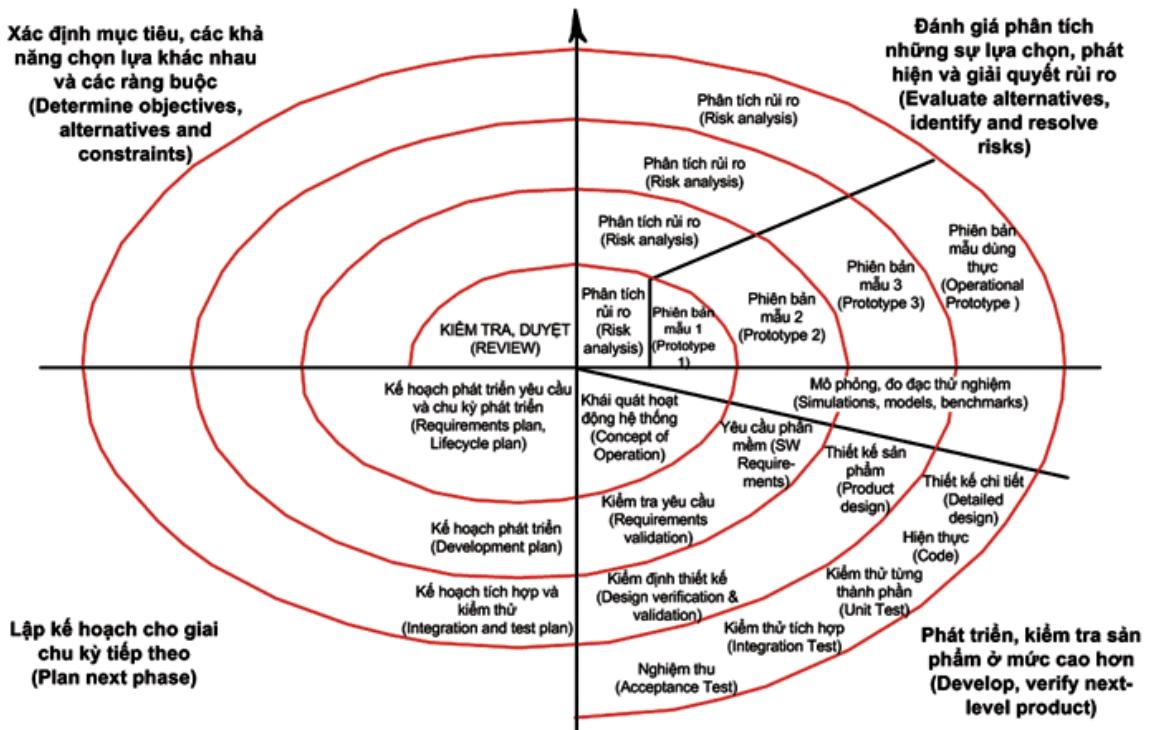
Sự khác nhau giữa hai mô hình tăng dần và lặp có thể được hiểu đơn giản như sau (so với sản phẩm được hoàn thành trong chu trình con trước):

- Mô hình tăng dần (Incremental): thêm chức năng vào sản phẩm.
- Mô hình lặp (Iterative): thay đổi sản phẩm.

7. Mô hình xoắn ốc (Spiral Model)

Mô hình xoắn ốc được Boehm đưa ra năm 1988. Mô hình này đưa thêm vào việc phân tích yếu tố rủi ro. Quá trình phát triển được chia thành nhiều bước lặp lại, mỗi bước bắt đầu bằng việc phân tích rủi ro rồi tạo bản mẫu, cải tạo và phát triển bản mẫu, duyệt lại, và cứ thế tiếp tục. Nội dung một bước gồm bốn hoạt động chính:

- Lập kế hoạch: xác định mục tiêu, các giải pháp và ràng buộc.
- Phân tích rủi ro: phân tích các phương án và xác định/giải quyết rủi ro.
- Công nghệ: phát triển sản phẩm “mức tiếp theo”.
- Đánh giá: đánh giá của khách hàng về kết quả của công nghệ.



Hình 12. Mô hình xoắn ốc (Spiral Model)

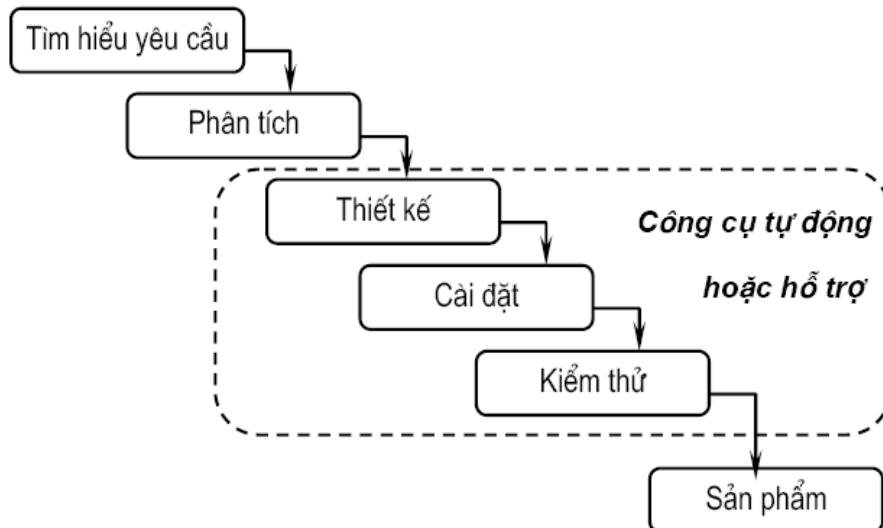
Với mỗi lần lắp xoắn ốc (bắt đầu từ tâm), các phiên bản được hoàn thiện dần. Nếu phân tích rủi ro chỉ ra rằng yêu cầu không chắc chắn thì bản mẫu có thể được sử dụng trong giai đoạn Công nghệ; các mô hình và các mô phỏng khác cũng được dùng để làm rõ hơn vấn đề và làm mịn yêu cầu. Tại một vòng xoắn ốc, phân tích rủi ro phải đi đến quyết định “tiến hành tiếp hay dừng”. Nếu rủi ro quá lớn thì có thể đình chỉ dự án. Mô hình xoắn ốc cũng có một số vấn đề như khó thuyết phục những khách hàng lớn rằng cách tiếp cận tiến hóa là kiểm soát được. Nó đòi hỏi tri thức chuyên gia đánh giá rủi ro chính xác và dựa trên tri thức chuyên gia này mà đạt được thành công. Mô hình xoắn ốc đòi hỏi năng lực quản lý cao, nếu không quản lý tốt thì rất dễ rơi vào trạng thái sửa đổi cục bộ không có kế hoạch của mô hình làm bản mẫu (thăm dò). Mô hình này còn tương đối mới và còn chưa được sử dụng rộng rãi như thác nước hoặc làm bản mẫu. Cần phải có thêm một vài năm nữa trước khi người ta có thể xác định được tính hiệu quả của mô hình này với sự chắc chắn hoàn toàn.

8. Kỹ thuật thế hệ thứ tư (4GT - 4th Generation Technology)

Thuật ngữ kỹ thuật thế hệ thứ tư bao gồm một phạm vi rộng các công cụ phần mềm có các điểm chung:

1. Cho phép người phát triển xác định một số đặc trưng của phần mềm ở mức cao.
2. Tự động sinh ra mã chương trình gốc theo nhu cầu của người phát triển.

Hiển nhiên là phần mềm được biểu diễn ở mức trừu tượng càng cao thì chương trình có thể được xây dựng càng nhanh hơn. Mô hình 4GT đối với Công nghệ phần mềm tập trung vào khả năng xác định phần mềm đối với một máy ở mức độ gần với ngôn ngữ tự nhiên hay dùng một ký pháp đếm lại chức năng có ý nghĩa. Hiện tại, một môi trường phát triển phần mềm hỗ trợ cho bối cảnh 4GT bao gồm một số hay tất cả các công cụ sau:



Hình 13. Kỹ thuật thế hệ thứ tư (4GT)

- Ngôn ngữ phi thủ tục để truy vấn CSDL
- Bộ sinh báo cáo
- Bộ thao tác dữ liệu
- Bộ tương tác và định nghĩa màn hình
- Bộ sinh mã chương trình
- Khả năng đồ họa mức cao
- Khả năng làm trang tính

Sinh tự động HTML và các ngôn ngữ tương tự dùng cho việc tạo trang Web sử dụng các công cụ phần mềm tiên tiến.

Mỗi một trong những công cụ này đã tồn tại, nhưng chỉ cho vài lĩnh vực ứng dụng đặc thù. Ví dụ: các tính năng macro trong các phần mềm bảng tính, cơ sở dữ liệu, khả năng tự sinh mã trong các công cụ thiết kế giao diện “kéo - thả”... Ngày nay, môi trường 4GT đã được mở rộng để thực hiện cho nhiều loại ứng dụng phần mềm.

Giống như các mô hình khác, 4GT bắt đầu với bước thu thập yêu cầu. lý tưởng nhất, các khách hàng sẽ mô tả các yêu cầu, và sẽ được dịch trực tiếp thành một bản mẫu có thể hoạt động được. Nhưng điều này không khả thi. Khách hàng có thể không chắc chắn cái gì là cần thiết, có thể mơ hồ trong việc xác định các thông tin đã biết, và có thể họ không thể hoặc không sẵn sàng để xác định các thông tin theo cách mà một công cụ 4GT có thể tiêu thụ. Vì lý do này, các đối thoại của khách hàng/nhà phát triển được mô tả cho các mô hình qui trình khác vẫn còn là một phần thiết yếu của cách tiếp cận 4GT.

Với những ứng dụng nhỏ, có thể chuyển trực tiếp từ bước thu thập yêu cầu sang cài đặt bằng cách sử dụng một ngôn ngữ thế hệ thứ 4 (4GL) hoặc mô hình bao gồm một mạng lưới các biểu tượng đồ họa. Tuy nhiên, cho các nỗ lực lớn hơn, cần thiết phát triển một chiến lược thiết kế cho hệ thống, ngay cả khi 4GL được sử dụng. Việc sử dụng 4GT mà không thiết kế (đối với các dự án lớn) sẽ gây ra những khó khăn tương tự (Kém chất lượng, bảo trì kém, nghèo nàn của khách hàng chấp nhận) đã khi phát triển phần mềm sử dụng phương pháp thông thường. Việc cài đặt bằng cách sử dụng một 4GL cho phép các nhà phát triển phần mềm trình bày các kết quả mong đợi bằng cách sinh mã tự động để tạo ra những kết quả đó. Rõ ràng, một cấu trúc dữ liệu với những thông tin có liên quan phải tồn tại và có thể được dễ dàng truy cập bởi các 4GL. Để chuyển đổi một cài đặt 4GT thành một sản phẩm, các nhà phát triển phải tiến hành kiểm tra toàn diện, phát triển các tài liệu có ý nghĩa và thực hiện tất cả các hoạt động tích hợp giải pháp khác được yêu cầu trong các mô hình công nghệ phần mềm khác. Ngoài ra, phần mềm được phát triển bởi 4GT phải được xây dựng theo một cách thức cho phép việc bảo trì được tiến hành nhanh chóng.

Giống như tất cả các mô hình Công nghệ phần mềm, các mô hình 4GT có ưu và nhược điểm.

Những người ủng hộ tuyên bố giảm đáng kể thời gian phát triển phần mềm và năng suất cải thiện rất nhiều cho những người xây dựng phần mềm.

Những người phản đối cho là các công cụ 4GT hiện tại không phải tất cả đều dễ dùng hơn các ngôn ngữ lập trình, rằng chương trình gốc do các công cụ này tạo ra là “không hiệu quả”, và rằng việc bảo trì các hệ thống phần mềm lớn được phát triển bằng cách dùng 4GT lại mở ra vấn đề mới.

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

1. Lĩnh vực ứng dụng hiện tại cho 4GT mới chỉ giới hạn vào các ứng dụng hệ thông tin nghiệp vụ, đặc biệt, việc phân tích thông tin và làm báo cáo là nhân tố chủ chốt cho các cơ sở dữ liệu lớn. Tuy nhiên, cũng đã xuất hiện các công cụ CASE mới hỗ trợ cho việc dùng 4GT để tự động sinh ra khung chương trình.

2. Đối với các ứng dụng vừa và nhỏ: thời gian cần cho việc tạo ra phần mềm được giảm đáng kể và khối lượng phân tích/thiết kế cũng được rút ngắn.

3. Đối với ứng dụng lớn: các hoạt động phân tích, thiết kế và kiểm thử chiếm phần lớn thời gian và việc loại bỏ bớt lập trình bằng cách dùng 4GT nhiều khi đem lại hiệu quả không đáng kể so với tính rườm rà, kém hiệu quả của phần mềm xây dựng bằng phương pháp này.

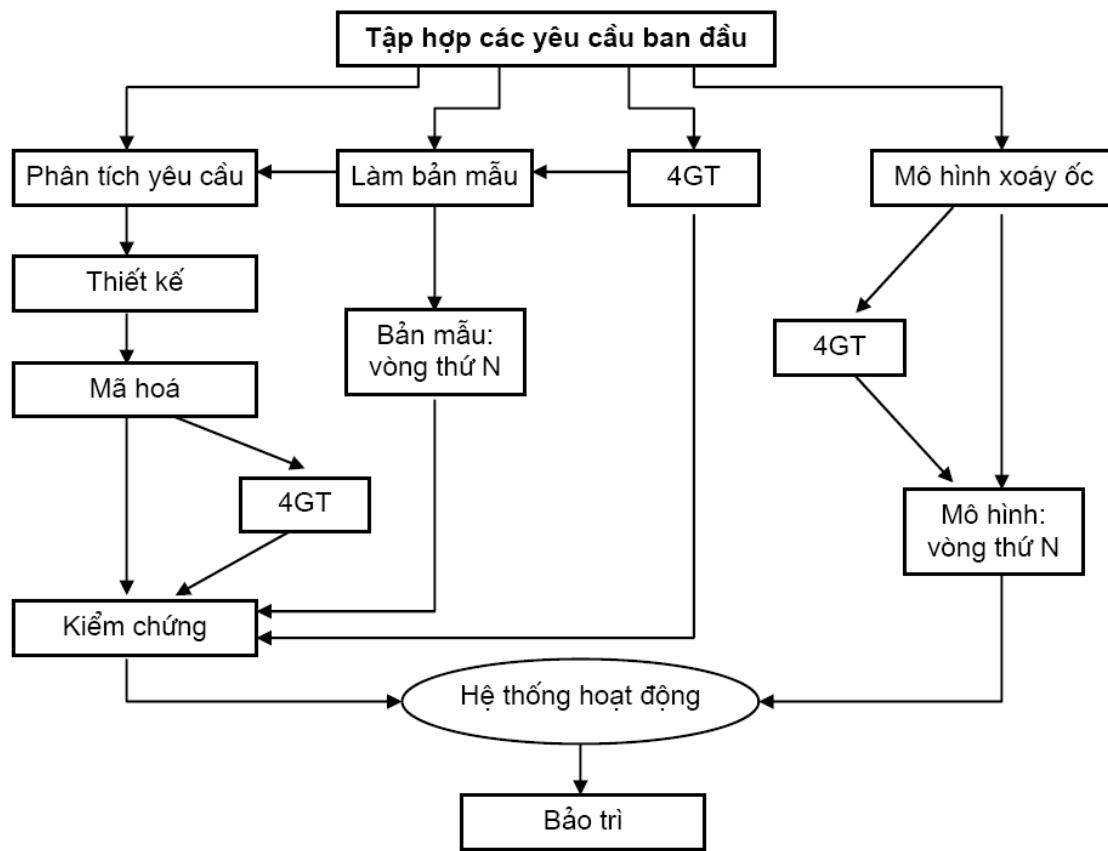
Tóm lại, 4GT đã trở thành một phần quan trọng của việc phát triển phần mềm nghiệp vụ và rất có thể sẽ được sử dụng rộng rãi trong các miền ứng dụng khác trong thời gian tới.

IV. Kết luận

Cũng như mọi ngành sản xuất khác, qui trình là một trong những yếu tố cực kỳ quan trọng đem lại sự thành công cho các nhà sản xuất phần mềm, nó giúp cho mọi thành viên trong dự án từ người cũ đến người mới, trong hay ngoài công ty đều có thể xử lý đồng bộ công việc tương ứng vị trí của mình thông qua cách thức chung của công ty, hay ít nhất ở cấp độ dự án.

Có thể nói qui trình phát triển/xây dựng phần mềm (Software Development/Engineering Process - SEP) có tính chất quyết định để tạo ra sản phẩm chất lượng tốt với chi phí thấp và năng suất cao, điều này có ý nghĩa quan trọng đối với các công ty sản xuất hay gia công phần mềm cung cấp và phát triển cùng với nền công nghiệp phần mềm đầy cạnh tranh.

Chúng ta đã xem xét các mô hình Công nghệ phần mềm như là các cách tiếp cận khác nhau tới Công nghệ phần mềm chứ không phải là các cách tiếp cận bổ sung cho nhau. Tuy nhiên, trong nhiều trường hợp chúng ta có thể và cũng nên tổ hợp các mô hình để đạt được thế mạnh của từng mô hình cho một dự án riêng lẻ. Ví dụ, mô hình xoắn ốc thực hiện điều này một cách trực tiếp, tổ hợp cả làm bản mẫu và các yếu tố của vòng đời cổ điển trong một cách tiếp cận tiến hóa tới công nghệ phần mềm. Các kỹ thuật thế hệ thứ tư có thể được dùng để cài đặt bản mẫu hay cài đặt hệ thống sản xuất trong bước mã hóa của vòng đời cổ điển. Chúng ta có thể làm bản mẫu trong bước phân tích của mô hình vòng đời cổ điển. Kết luận ở đây là chúng ta không nên bị lệ thuộc với bất cứ mô hình cụ thể nào. Tính chất và qui mô của phần mềm cần phát triển sẽ là yếu tố quyết định tới chọn mô hình. Mỗi cách tiếp cận đều có ưu điểm riêng và bằng cách tổ hợp khéo léo các cách tiếp cận thì chúng ta sẽ có một phương pháp hỗn hợp ưu việt hơn các phương pháp được dùng độc lập.



Hình 14. Tổ hợp các mô hình

Trên đây là một số các mô hình phổ biến, thực sự việc lựa chọn cụ thể mô hình nào không phải dễ dàng và trên thực tế người ta thường dùng mô hình lai, kết hợp một số mô hình với nhau sao cho phù hợp với dự án.

Việc cải tiến các mô hình phát triển phần mềm luôn là đề tài nghiên cứu hấp dẫn, với sự tham gia tích cực không những từ các nhà sản xuất phần mềm mà còn từ các viện đại học khắp thế giới. Riêng với các nhà phát triển phần mềm, họ luôn cố gắng cải tiến liên tục qui trình phát triển của mình nhằm không ngừng đổi mới, nâng cao năng suất và chất lượng sản phẩm. Tuy nhiên, một điều dễ thấy là việc lựa chọn, tùy biến mô hình phù hợp cho các dự án đã khó, nhưng việc vận hành nó vào trong quá trình phát triển sản phẩm càng khó hơn. Đó chính là thách thức cho các nhà phát triển phần mềm.

BÀI 2. PHÂN TÍCH VÀ THIẾT KẾ HƯỚNG ĐỐI TƯỢNG VÀ UML

Trong những năm gần đây phương thức lập trình hướng đối tượng đã thống lĩnh thị trường lập trình phần mềm và UML cũng đã trở thành ngôn ngữ mô hình hóa phổ biến trong sản xuất phần mềm.

I. Khái niệm

Trong công nghệ phần mềm để sản xuất được một sản phẩm phần mềm người ta chia quá trình phát triển sản phẩm ra nhiều giai đoạn như thu thập và phân tích yêu cầu, phân tích và thiết kế hệ thống, phát triển (coding), kiểm thử, triển khai và bảo trì. Trong đó, giai đoạn phân tích, thiết kế bao giờ cũng là giai đoạn khó khăn và phức tạp nhất. Giai đoạn này giúp chúng ta hiểu rõ yêu cầu đặt ra, xác định giải pháp, mô tả chi tiết giải pháp. Nó trả lời 2 câu hỏi What (phần mềm này làm cái gì?) và How (làm nó như thế nào?).

Để phân tích và thiết kế một phần mềm thì có nhiều cách làm, một trong những cách làm đó là xem hệ thống gồm những đối tượng sống trong đó và tương tác với nhau. Việc mô tả được tất cả các đối tượng và sự tương tác của chúng sẽ giúp chúng ta hiểu rõ hệ thống và cài đặt được nó. Phương thức này gọi là Phân tích thiết kế hướng đối tượng (OOAD - Object-Oriented Analysis and Design).

UML (Unified Modeling Language) là ngôn ngữ mô hình hóa hợp nhất dùng để biểu diễn hệ thống. Nói một cách đơn giản là nó dùng để tạo ra các biểu đồ nhằm mô tả thiết kế hệ thống. Các biểu đồ này được sử dụng để các nhóm thiết kế trao đổi với nhau cũng như dùng để thi công hệ thống (phát triển), thuyết phục khách hàng, các nhà đầu tư, ...

a. Tại sao lại là OOAD và UML?

OOAD cần các biểu đồ để mô tả hệ thống được thiết kế, còn UML là ngôn ngữ mô tả các biểu đồ nên cần nội dung thể hiện. Do vậy, chúng ta phân tích và thiết kế theo hướng đối tượng và sử dụng UML để biểu diễn các thiết kế đó nên chúng thường đi đôi với nhau.

b. OOAD sử dụng UML

UML sử dụng để vẽ cho nhiều lĩnh vực khác nhau như phần mềm, cơ khí, xây dựng, ... trong phạm vi môn học này chúng ta chỉ nghiên cứu cách sử dụng UML cho phân tích và thiết kế hướng đối tượng trong ngành phần mềm. OOAD sử dụng UML bao gồm các thành phần sau:

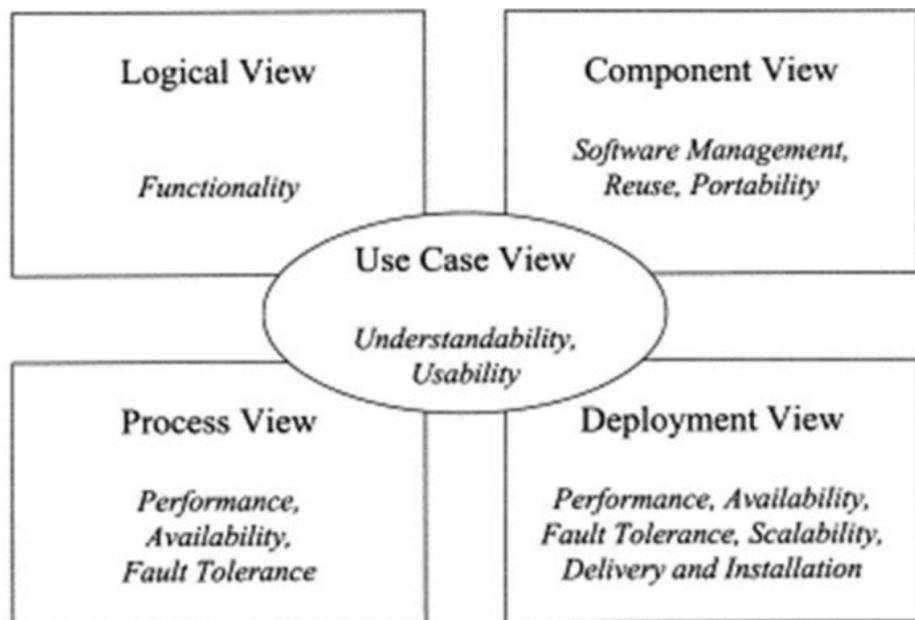
- View (góc nhìn)
- Diagram (biểu đồ)
- Notations (ký hiệu)
- Mechanisms (qui tắc, cơ chế)

Chúng ta sẽ tìm hiểu kỹ hơn các thành phần trên.

II. Thiết kế với UML

1) View (góc nhìn)

Trong phần mềm cũng như vậy, OOAD sử dụng UML có các góc nhìn sau:



Hình 15. Các View trong OOAD sử dụng UML

Trong đó:

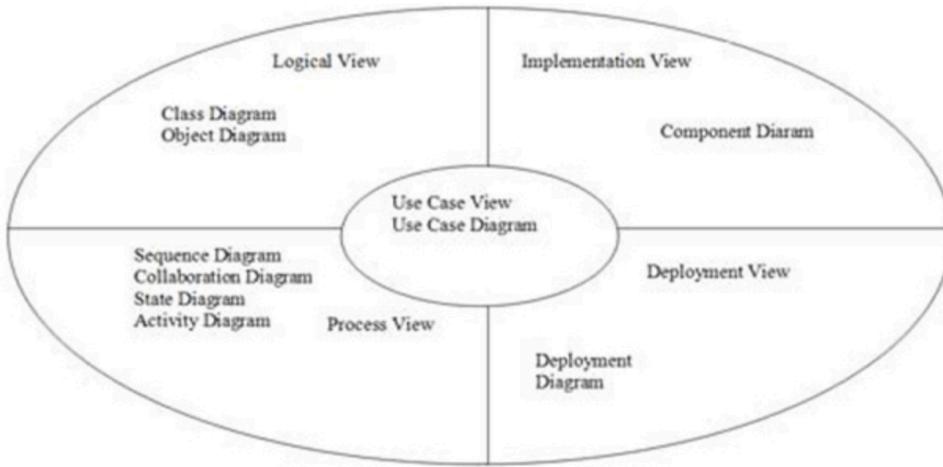
- Use Case View: cung cấp góc nhìn về các ca sử dụng giúp chúng ta hiểu hệ thống có gì? Ai dùng và dùng nó như thế nào?
- Logical View: cung cấp góc nhìn về cấu trúc hệ thống, xem nó được tổ chức như thế nào. Bên trong nó có gì?
- Process View: cung cấp góc nhìn động về hệ thống, xem các thành phần trong hệ thống tương tác với nhau như thế nào.
- Component View: Cũng là một góc nhìn về cấu trúc giúp chúng ta hiểu cách phân bổ và sử dụng lại các thành phần trong hệ thống ra sao.
- Deployment View: cung cấp góc nhìn về triển khai hệ thống, nó cũng ảnh hưởng lớn đến kiến trúc hệ thống.

Tập hợp các góc nhìn này sẽ giúp chúng ta hiểu rõ hệ thống cần phân tích, thiết kế. Trong hình vẽ chúng ta thấy góc nhìn Use Case View nằm ở giữa và chi phối tất cả các góc nhìn còn lại. Chính vì thế chúng ta thường thấy các tài liệu nói về 4 view + 1 chứ không phải 5 view nhằm nhấn mạnh vai trò của Use Case View.

2) Diagram (Biểu đồ)

Diagram các bạn có thể dịch là sơ đồ. Tuy nhiên ở đây chúng ta sử dụng từ biểu đồ cho dễ hình dung.

Các biểu đồ được dùng để thể hiện các góc nhìn của hệ thống.



Hình 16. Các biểu đồ trong OOAD sử dụng UML

Trong đó:

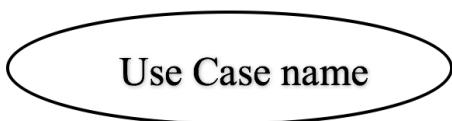
- Use Case Diagram: biểu đồ mô tả về cách sử dụng của hệ thống. Biểu đồ này sẽ giúp chúng ta biết được ai sử dụng hệ thống, hệ thống có những chức năng gì. Lập được biểu đồ này bạn sẽ hiểu được yêu cầu của hệ thống cần xây dựng.
- Class Diagram: biểu đồ này mô tả cấu trúc của hệ thống, tức hệ thống được cấu tạo từ những thành phần nào. Nó mô tả khía cạnh tĩnh của hệ thống.
- Object Diagram: tương tự như Class Diagram nhưng nó mô tả đến đối tượng thay vì lớp (Class).
- Sequence Diagram: là biểu đồ mô tả sự tương tác của các đối tượng trong hệ thống với nhau được mô tả tuần tự các bước tương tác theo thời gian.
- Collaboration Diagram: tương tự như sequence Diagram nhưng nhấn mạnh về sự tương tác thay vì tuần tự theo thời gian.
- State Diagram: biểu đồ mô tả sự thay đổi trạng thái của một đối tượng. Nó được dùng để theo dõi các đối tượng có trạng thái thay đổi nhiều trong hệ thống.
- Activity Diagram: biểu đồ mô tả các hoạt động của đối tượng, thường được sử dụng để hiểu về nghiệp vụ của hệ thống.
- Component Diagram: biểu đồ mô tả về việc bố trí các thành phần của hệ thống cũng như việc sử dụng các thành phần đó.
- Deployment Diagram: biểu đồ mô tả việc triển khai của hệ thống như việc kết nối, cài đặt, hiệu năng của hệ thống v.v...

Chúng ta sẽ bàn kỹ các biểu đồ này trong các bài tiếp theo. Vì nó chính là hạt nhân của hoạt động PT&TKHT.

Lưu ý: Ở đây chúng ta sử dụng từ hệ thống tương đương với sản phẩm phần mềm.

3) Notations (các ký hiệu)

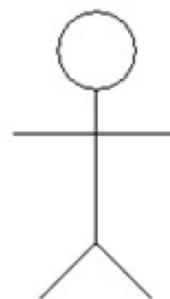
Notations là các ký hiệu để vẽ, nó như từ vựng trong ngôn ngữ tự nhiên. Bạn phải biết từ vựng thì mới ghép thành câu, thành bài được. Chúng ta sẽ tìm hiểu kỹ các notations trong từng biểu đồ sau này. Dưới đây là vài ví dụ về notation.



Hình 17. Ký hiệu về Use Case



Hình 18. Ký hiệu về Class



Hình 19. Ký hiệu về Actor

Ngoài ra, còn rất nhiều ký hiệu nữa và chúng ta sẽ tìm hiểu thêm kh xây dựng các biểu đồ.

4) Mechanisms (Rules)

Mechanisms là các qui tắc để lập nên biểu đồ, mỗi biểu đồ có qui tắc riêng và bạn phải nắm được để tạo nên các biểu đồ thiết kế đúng. Các qui tắc này chúng ta sẽ bàn kỹ trong các bài về các biểu đồ.

III. Kết luận

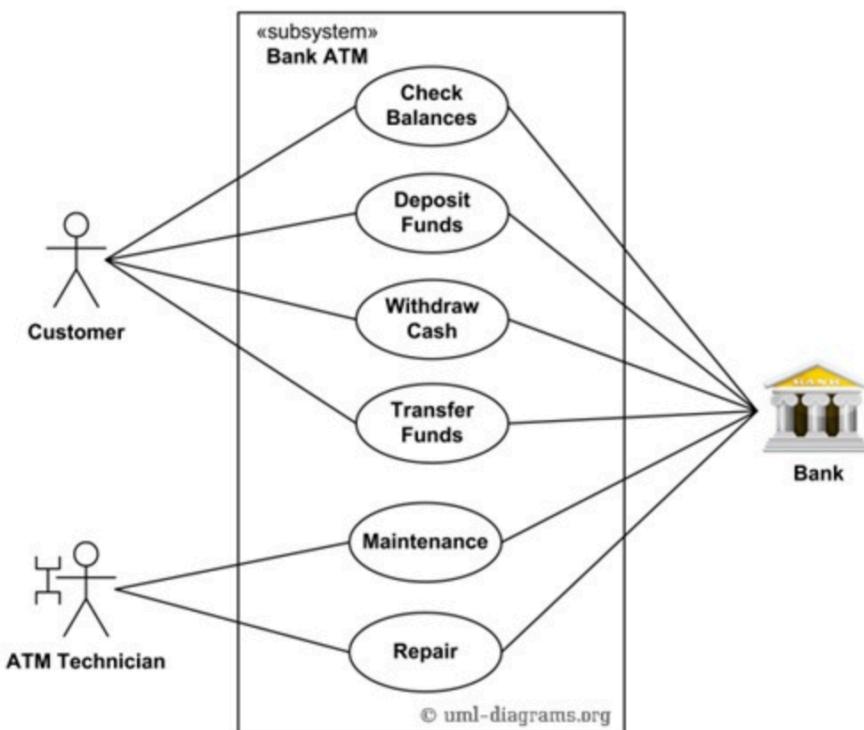
Nguyên tắc phân tích, thiết kế một hệ thống phần mềm cũng không khác việc xây dựng một cái nhà trong xây dựng. Bạn hãy nhớ cách tiếp cận này để dễ hiểu hơn trong việc phân tích và thiết kế hệ thống. Hãy giữ mọi thứ cho thật đơn giản để dễ hiểu và dễ áp dụng.

BÀI 3. BIỂU ĐỒ USE CASE

Trong bài trước chúng ta đã biết vai trò của biểu đồ Use Case là rất quan trọng, nó giúp chúng ta hiểu yêu cầu, kiến trúc chức năng của hệ thống và chi phối tất cả các biểu đồ còn lại. Trong bài này chúng ta sẽ tìm hiểu về các thành phần cấu tạo nên biểu đồ này, cách xây dựng và sử dụng nó.

I. Các thành phần trong biểu đồ Use Case

Đầu tiên, chúng ta xem một ví dụ về Use Case Diagram.



Hình 20. Biểu đồ Use Case về ứng dụng ATM

Nhìn biểu đồ này chúng ta thấy có hai người dùng là Customer và ATM Technician và một đối tượng sử dụng hệ thống là Bank. Bên cạnh đó nó mô tả các chức năng của hệ thống và người dùng nào dùng chức năng gì. Điều này giúp chúng ta hình dung được là chúng ta sẽ xây dựng hệ thống với những chức năng gì? Cho ai dùng.

Bây giờ chúng ta sẽ tìm hiểu kỹ hơn về các thành phần của biểu đồ.

1. Actor (tác nhân)

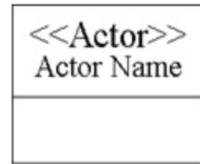
Actor được dùng để chỉ người sử dụng hoặc một đối tượng nào đó bên ngoài tương tác với hệ thống chúng ta đang xem xét. Lưu ý, chúng ta hay bỏ quên đối tượng tương tác với hệ thống, ví dụ như Bank ở trên.

Actor được biểu diễn như sau:



Actor Name

Hoặc:



2. Use Case

Use Case là chức năng mà các Actor sẽ sử dụng. Nó được ký hiệu như sau:

Use Case name

Với việc xác định các chức năng mà Actor sử dụng bạn sẽ xác định được các Use Case cần có trong hệ thống.

3. Relationship (Quan hệ)

Relationship hay còn gọi là connector được sử dụng để kết nối giữa các đối tượng với nhau tạo nên biểu đồ Use Case. Có các kiểu quan hệ cơ bản sau:

- Association
- Generalization
- Include
- Extend

Chúng ta sẽ lần lượt tìm hiểu về các kiểu quan hệ dưới đây.

+ Quan hệ Association:



Association

Association thường được dùng để mô tả mối quan hệ giữa Actor và Use Case và giữa các Use Case với nhau.



+ Quan hệ Generalization



Generalization

Generalization được sử dụng để thể hiện quan hệ thừa kế giữa các Actor hoặc giữa các Use Case với nhau.

Ví dụ Actor User thừa kế toàn bộ quyền của Actor Guest:



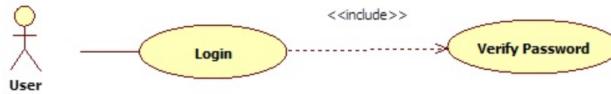
+ Quan hệ Include



Include

Include là quan hệ giữa các Use Case với nhau, nó mô tả việc một Use Case lớn được chia ra thành các Use Case nhỏ để dễ cài đặt (mô-đun hóa) hoặc thể hiện sự dùng lại.

Ví dụ về quan hệ Include giữa các Use Case:



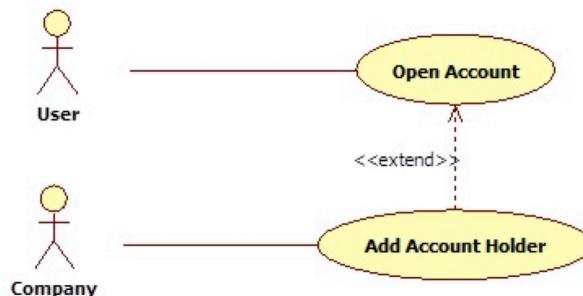
Chúng ta thấy Use Case “Verify Password” có thể gộp chung vào Use Case Login nhưng ở đây chúng ta tách ra để cho các Use Case khác sử dụng hoặc để mô-đun hóa cho dễ hiểu, dễ cài đặt.

+ Quan hệ Extend



Extend dùng để mô tả quan hệ giữa 2 Use Case. Quan hệ Extend được sử dụng khi có một Use Case được tạo ra để **bổ sung** chức năng cho một Use Case có sẵn và được sử dụng trong một điều kiện nhất định nào đó.

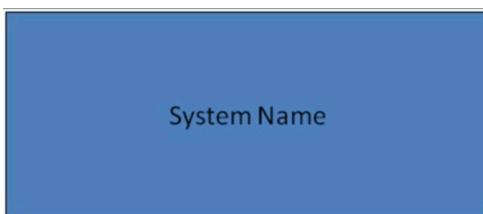
Ví dụ về quan hệ Extend giữa các Use Case:



Trong ví dụ trên “Open Account” là Use Case cơ sở để cho khách hàng mở tài khoản. Tuy nhiên, có thêm một điều kiện là nếu khách hàng là công ty thì có thể thêm người sở hữu lên tài khoản này. Add Account Holder là chức năng mở rộng của Use Case “Open Account” cho trường hợp cụ thể nếu Actor là Công ty nên quan hệ của nó là quan hệ Extend.

4. System Boundary

System Boundary được sử dụng để xác định phạm vi của hệ thống mà chúng ta đang thiết kế. Các đối tượng nằm ngoài hệ thống này có tương tác với hệ thống được xem là các Actor.



System Boundary sẽ giúp chúng ta dễ hiểu hơn khi chia hệ thống lớn thành các hệ thống con để phân tích, thiết kế.

II. Các bước xây dựng Use Case Diagram

Chúng ta đã nắm được các ký hiệu của biểu đồ Use Case, bây giờ là lúc chúng ta tìm cách lắp chúng lại để tạo nên biểu đồ hoàn chỉnh. Thực hiện các bước sau để xây dựng một biểu đồ Use Case:

+ Bước 1: Tìm các Actor

Trả lời các câu hỏi sau để xác định Actor cho hệ thống:

- Ai sử dụng hệ thống này?
- Hệ thống nào tương tác với hệ thống này?

Xem xét ví dụ về ATM ở trên chúng ta thấy:

- Ai sử dụng hệ thống? -> Customer, ATM Technician
- Hệ thống nào tương tác với hệ thống này? -> Bank

Như vậy có 03 Actor: Customer, ATM Technician và Bank

+ Bước 2: Tìm các Use Case

Trả lời câu hỏi các Actor sử dụng chức năng gì trong hệ thống? chúng ta sẽ xác định được các Use Case cần thiết cho hệ thống.

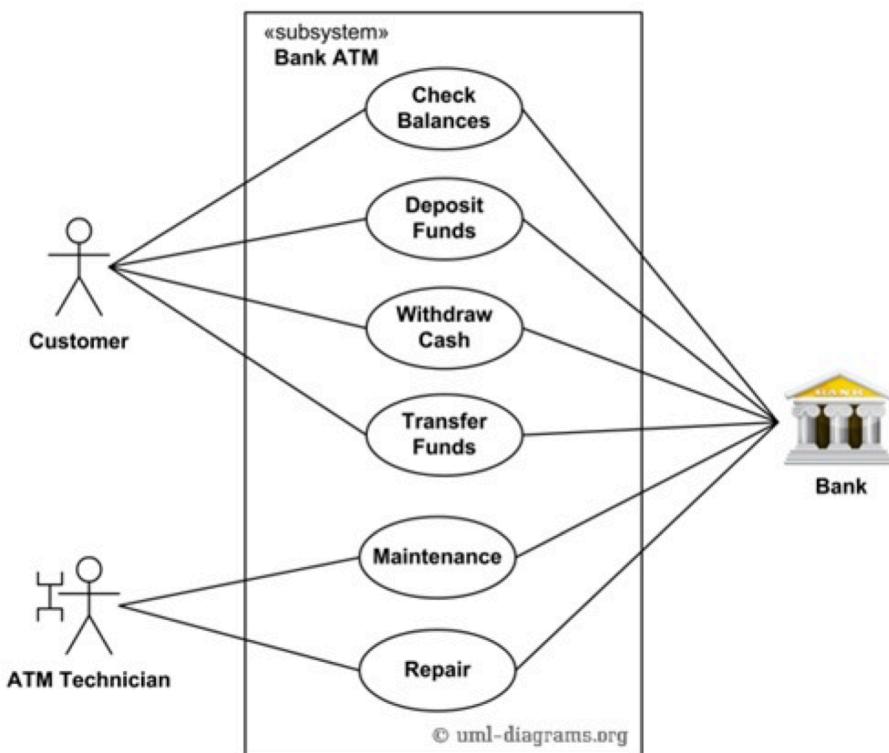
Xem xét ví dụ ở trên ta thấy:

- Customer sử dụng các chức năng: Check Balance, Deposit, Withdraw và Transfer
- ATM technician sử dụng: Maintenance và Repair
- Bank tương tác với tất cả các chức năng trên.

Tóm lại, chúng ta phải xây dựng hệ thống có các chức năng: Check Balance, Deposit, Withdraw, Transfer, Maintenance và Repair để đáp ứng được cho người sử dụng và các hệ thống tương tác.

+ Bước 3: Xác định các quan hệ

Phân tích và xác định các quan hệ giữa các Actor và Use Case, giữa các Actor với nhau, giữa các Use Case với nhau sau đó nối lại chúng lại chúng ta sẽ được biểu đồ Use Case.



III. ĐẶC TẢ USE CASE

Nhìn vào biểu đồ trên chúng ta nhận biết hệ thống cần những chức năng gì và ai sử dụng. Tuy nhiên, chúng ta chưa biết được chúng vận hành ra sao? Sử dụng chúng như thế nào? Để hiểu rõ hơn hệ thống chúng ta cần đặc tả các Use Case.

Có 2 cách để đặc tả Use Case.

1. Cách 1: Viết đặc tả cho các Use Case

Chúng ta có thể viết đặc tả Use Case theo mẫu sau:

- Tên Use Case //Account Details
 - Mã số Use Case //UCSEC35
 - Mô tả tóm tắt// Hiển thị thông tin chi tiết của Account
 - Các bước thực hiện // Liệt kê các bước thực hiện
- Điều kiện thoát // Khi người dùng kích nút Close
 - Yêu cầu đặc biệt// Ghi rõ nếu có
 - Yêu cầu trước khi thực hiện// Phải đăng nhập
 - Điều kiện sau khi thực hiện // Ghi rõ những điều kiện nếu có sau khi thực hiện Use Case này

2. Cách 2: Sử dụng các biểu đồ để đặc tả

Chúng ta có thể dùng các biểu đồ như Activity Diagram, Sequence Diagram để đặc tả Use case. Các biểu đồ này chúng ta sẽ bàn ở những bài tiếp theo.

IV. Sử dụng UseCase Diagram

Như chúng ta đã biết Use Case Diagram có một vai trò đặc biệt quan trọng trong quá trình phân tích, thiết kế và phát triển hệ thống. Dưới đây chúng tôi liệt kê một số ứng dụng tiêu biểu của Use Case Diagram.

- Phân tích và hiểu hệ thống
- Thiết kế hệ thống
- Làm cơ sở cho việc phát triển, kiểm tra các biểu đồ như Class Diagram, Activity Diagram, Sequence Diagram, Component Diagram
- Làm cơ sở để giao tiếp với khách hàng, các nhà đầu tư
- Giúp cho việc kiểm thử chức năng, kiểm thử chấp nhận

V. Kết luận

Đến đây, chúng ta đã tìm hiểu được biểu đồ đầu tiên và rất quan trọng (Use Case Diagram), các bạn cần tiếp tục thực hành để nắm rõ hơn về biểu đồ này cũng như cách xây dựng và sử dụng chúng trong quá trình phát triển sản phẩm phần mềm.

Để giúp các bạn hiểu rõ hơn về biểu đồ Use Case trong bài tiếp theo chúng ta sẽ thực hiện qua từng bước bài thực hành xây dựng Use Case Diagram.

THỰC HÀNH XÂY DỰNG BIỂU ĐỒ USE CASE

Chúng ta đã hiểu được các thành phần cấu tạo, cách xây dựng và sử dụng của biểu đồ Use Case ở bài trước. Trong phần này, chúng ta sẽ tiến hành thực tập để xây dựng biểu đồ này cho một hệ thống cụ thể nhằm giúp bạn nắm rõ hơn về biểu đồ này.

Case Study – Yêu cầu của bài thực hành

Yêu cầu xây dựng một hệ thống thương mại điện tử (E-Commerce) như sau:

Công ty Hi-Tech là một công ty chuyên kinh doanh về các thiết bị điện tử và công nghệ thông tin trong nhiều năm nay và đã có một lượng khách hàng nhất định.

Để mở rộng hoạt động kinh doanh của mình, công ty mong muốn xây dựng một hệ thống thương mại điện tử nhằm mở rộng phạm vi kinh doanh trên mạng Internet.

Hệ thống mới phải đảm bảo cho khách hàng viếng thăm Website dễ dàng lựa chọn các sản phẩm, xem các khuyến mãi cũng như mua hàng. Việc thanh toán có thể được thực hiện qua mạng hoặc thanh toán trực tiếp tại cửa hàng. Khách hàng có thể nhận hàng tại cửa hàng hoặc sử dụng dịch vụ chuyển hàng có phí của công ty.

Ngoài ra, hệ thống cũng cần có phân hệ để đảm bảo cho công ty quản lý các hoạt động kinh doanh như số lượng hàng có trong kho, quản lý đơn đặt hàng, tình trạng giao hàng, thanh toán v.v...

*Thông tin chi tiết các chức năng các bạn có thể tham khảo thêm tại: <http://www.bkc.vn>
Bạn hãy giúp công ty Hi-Tech xây dựng hệ thống trên.*

Case Study này sẽ được sử dụng xuyên suốt các bài sau để giúp các bạn dễ hiểu và dễ thực tập.

Các bước xây dựng biểu đồ Use Case:

Bước 1: Thu thập kiến thức liên quan đến hệ thống sẽ xây dựng

Trước hết, để phân tích hệ thống trên bạn phải có kiến thức về hệ thống thương mại điện tử, chúng ta có thể tìm hiểu thông qua các nguồn sau:

- Xem các trang Web bán hàng qua mạng như amazon, lazada.vn, bkc.vn v.v..
- Xem các hệ thống mẫu về thương mại điện tử nguồn mở như Magento, OpenCart, Spree Commerce v.v...
- Đọc sách, báo về eCommerce
- Hỏi những người chuyên về lĩnh vực này (hỏi chuyên gia)

Lưu ý: Bạn không thể thiết kế tốt được nếu bạn không có kiến thức về lĩnh vực của sản phẩm mà bạn sẽ xây dựng.

Bước 2: Xác định các Actor

Bạn hãy trả lời cho câu hỏi “Ai sử dụng hệ thống này?”

Xem xét Website chúng ta nhận thấy:

- Những người muốn mua hàng vào website để xem thông tin. Những người này là Khách hàng tiềm năng (Guest).
- Những người đã đặt hàng vào kiểm tra đơn hàng, thanh toán v.v.. gọi là Khách hàng (Customer).

Về phía đơn vị bán hàng, có những người sau đây tham gia vào hệ thống:

- Người quản lý bán hàng: quyết định nhập hàng, giá bán, quản lý tồn kho, doanh thu, chính sách khuyến mãi.
- Người bán hàng: Tư vấn cho khách hàng, theo dõi đơn hàng, thu tiền, theo dõi chuyển hàng cho khách.
- Quản lý kho: xuất, nhập hàng, quản lý tồn kho
- Quản trị hệ thống: Tạo người dùng, Phân quyền, Tạo cửa hàng

Tiếp theo chúng ta trả lời câu hỏi “Hệ thống nào tương tác với hệ thống này?”

Giả sử ở đây, chúng ta sử dụng dịch vụ của ngân hàng để thanh toán trực tuyến và gọi nó là “Cổng thanh toán” thì ta có thêm một Actor tương tác với hệ thống.

Như vậy, chúng ta đã có các Actor của hệ thống gồm: Khách hàng tiềm năng, khách hàng, Người bán hàng, Quản lý Kho, Quản trị hệ thống, Cổng thanh toán

Bạn cần khảo sát và phân tích thêm cũng như hỏi trực tiếp khách hàng để xác định đầy đủ các Actor cho hệ thống.

Bước 3: Xác định Use Case

Bạn cần trả lời câu hỏi “Actor sử dụng chức năng gì trên hệ thống?”.

Trước tiên, xem xét với Actor “Khách hàng tiềm năng” trên trang bkc.vn để xem họ sử dụng chức năng nào?

- Xem trang chủ
- Xem các sản phẩm theo:
 - Theo chủng loại
 - Nhà sản xuất
 - Tìm kiếm theo văn bản gõ vào
- Xem chi tiết sản phẩm được chọn
- Xem khuyến mãi
- Xem so sánh
- Mua hàng
- Quản lý giỏ hàng
- Chat với người bán hàng
- Đăng ký tài khoản để trở thành khách hàng

Tiếp theo, xem xét Actor “Khách hàng” và nhận thấy họ sử dụng chức năng:

- Đăng nhập
- Xem đơn hàng
- Thanh toán

Tiếp theo, xem xét Actor “Người bán hàng” và họ có thể sử dụng các chức năng:

- Đăng nhập
- Chat với khách hàng
- Theo dõi đơn hàng
- Thu tiền
- Theo dõi chuyển hàng

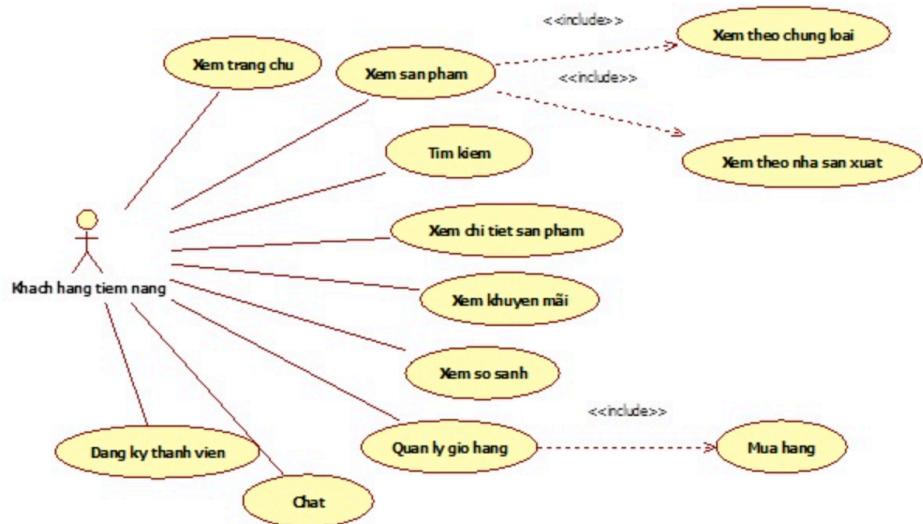
Tương tự như vậy bạn xác định chức năng cho các Actor còn lại.

Bước 4: Vẽ biểu đồ Use Case

Trước hết chúng ta xem xét và phân tích các chức năng của “Khách hàng tiềm năng” chúng ta nhận thấy.

- Chức năng xem sản phẩm có 2 cách là chọn loại sản phẩm, nhà sản xuất để xem và gõ vào ô tìm kiếm. Nên chúng ta tách ra làm 2 là Xem sản phẩm và Tìm kiếm.
- Chức năng mua hàng, thực chất là thêm vào giỏ hàng nên có thể xem là chức năng con của quản lý giỏ hàng.

Đặt lại tên cho gọn và xác định các mối quan hệ của chúng, chúng ta có thể vẽ Use Case Diagram cho Actor này như sau:



Hình 21. Biểu đồ Use Case cho Actor “Khách hàng tiềm năng”

Tiếp theo, chúng xem xét các chức năng cho Actor “Khách hàng” và nhận thấy chức năng “Thanh toán” thường thực hiện cho từng đơn hàng cụ thể nên có thể nó là chức năng con của “Quản lý đơn hàng”. Ngoài ra, các chức năng Actor này sử dụng không giao với Actor “Khách hàng tiềm năng” nên nó được biểu diễn như sau:



Hình 22. Biểu đồ Use Case cho Actor “Khách hàng”

Tiếp tục xem xét Actor “Người bán hàng” chúng ta nhận thấy:

- Chức năng “Thu tiền” thực tế là thanh toán trực tiếp tại quầy cho từng đơn hàng và chức năng “Theo dõi chuyển hàng” được thực hiện trên từng đơn hàng nên nó có thể là chức năng con của “Quản lý đơn hàng”.
- Chức năng “Quản lý đơn hàng” ở đây quản lý cho nhiều khách hàng nên sẽ khác với chức năng “Quản lý đơn hàng” của Actor “Khách hàng” nên để phân biệt chúng ta sửa chức năng “Quản lý đơn hàng” của Actor “Khách hàng” thành “Quản lý đơn hàng cá nhân”
- Chức năng “Đăng nhập” có thể dùng chung với Actor “Khách hàng”, chức năng Chat dùng chung với Actor “Khách hàng tiềm năng”

Vẽ chúng chung với nhau chúng ta được biểu đồ như sau:



Figure 23. Biểu đồ Use Case

Các bạn hãy tiếp tục hoàn tất các chức năng cho các Actor còn lại để có một biểu đồ hoàn chỉnh về Use Case cho hệ thống.

Như vậy, chúng ta đã thực hành việc xây dựng biểu đồ Use Case cho hệ thống eCommerce. Trong bài tiếp theo chúng ta sẽ tìm hiểu về Class Diagram, một biểu đồ cơ sở rất quan trọng nữa trong việc phân tích và thiết kế hệ thống.

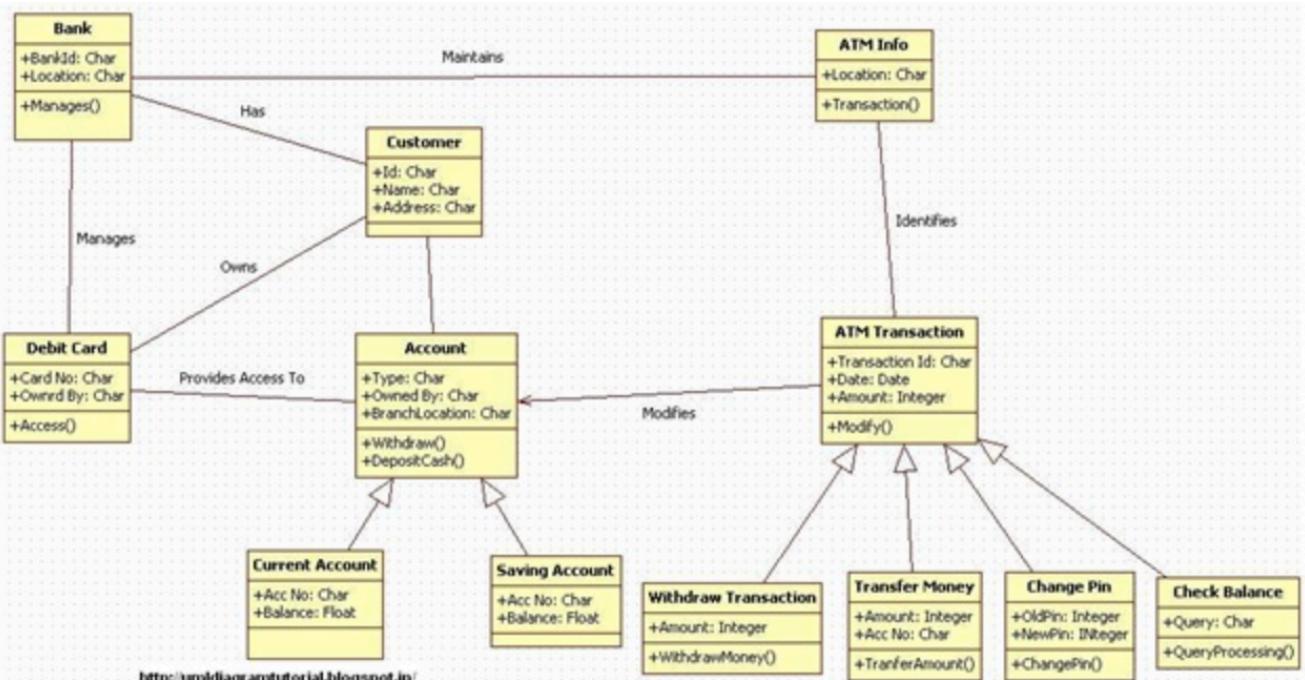
BÀI 4. BIỂU ĐỒ LỚP (CLASS DIAGRAM)

Class Diagram là một trong những biểu đồ quan trọng nhất của thiết kế phần mềm, nó cho thấy cấu trúc và quan hệ giữa các thành phần tạo nên phần mềm. Trong quá trình xây dựng Class Diagram chúng ta sẽ phải quyết định rất nhiều yếu tố về thiết kế nên nó là biểu đồ khó xây dựng nhất. Biểu đồ này sẽ cho thấy cấu trúc tĩnh của phần mềm.

Trong bài này, chúng ta sẽ tìm hiểu các thành phần tạo nên biểu đồ, cách xây dựng và sử dụng class diagram để giúp các bạn hiểu và áp dụng biểu đồ này trong thiết kế. Ở đây, mặc định các bạn đã có kiến thức về lập trình hướng đối tượng và không nhắc lại các khái niệm trong lập trình hướng đối tượng.

I. Các thành phần trong biểu đồ Class

Trước tiên, chúng ta xem một biểu đồ Class.



Hình 24. Ví dụ về biểu đồ lớp của ATM

Ví dụ trên là Class Diagram của ứng dụng ATM. Tiếp theo chúng ta sẽ bàn kỹ về các thành phần của biểu đồ này và lấy ứng dụng về ATM ở trên để minh họa.

1. Classes (Các lớp)

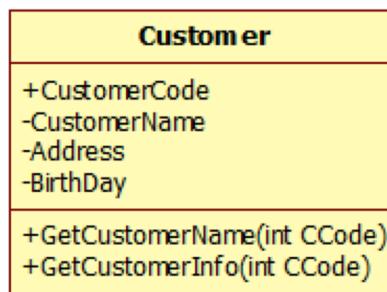
Class là thành phần chính của biểu đồ Class Diagram. Class mô tả về một nhóm đối tượng có cùng tính chất, hành động trong hệ thống. Ví dụ mô tả về khách hàng chúng ta dùng lớp “Customer”. Class được mô tả gồm tên Class, thuộc tính và phương thức.

Class Name
Attributes
Methods

Hình 25. Ký hiệu một Class

Trong đó,

- Class Name: là tên của lớp.
- Attributes (thuộc tính): mô tả tính chất của các đối tượng. Ví dụ như khách hàng có Mã khách hàng, Tên khách hàng, Địa chỉ, Ngày sinh v.v...
- Method (Phương thức): chỉ các hành động mà đối tượng này có thể thực hiện trong hệ thống. Nó thể hiện hành vi của các đối tượng do lớp này tạo ra.



Hình 26. Ví dụ về một Class "Customer"

Một số loại Class đặc biệt như Abstract Class (lớp không tạo ra đối tượng), Interface (lớp khai báo mà không cài đặt) v.v.. chúng ta xem thêm các tài liệu về lập trình hướng đối tượng để hiểu rõ hơn các vấn đề này.

2. Relationship (Quan hệ)

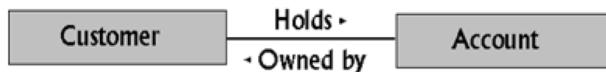
Relationship thể hiện mối quan hệ giữa các Class với nhau. Trong UML 2.0 có các quan hệ thường sử dụng như sau:

- Association
- Aggregation
- Composition
- Generalization

Chúng ta sẽ lần lượt tìm hiểu về chúng.

+ Association

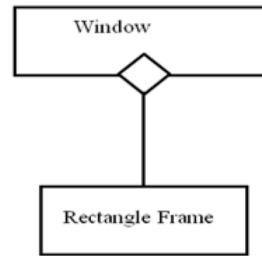
Association là quan hệ giữa hai lớp với nhau, thể hiện chúng có liên quan với nhau. Association thể hiện qua các quan hệ như “has: có”, “Own: sở hữu” v.v...



Ví dụ quan hệ trên thể hiện Khách hàng nắm giữ Tài khoản và Tài khoản được sở hữu bởi Khách hàng.

+ Aggregation

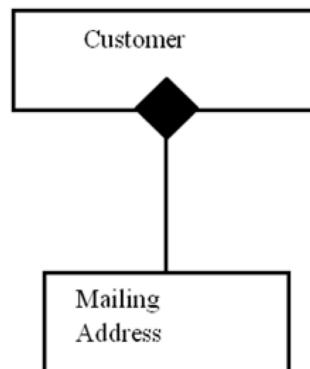
Aggregation là một loại của quan hệ Association nhưng mạnh hơn. Nó có thể cùng thời gian sống (cùng sinh ra hoặc cùng chết đi)



Ví dụ quan hệ trên thể hiện lớp Window (cửa sổ) được lắp trên Khung cửa hình chữ nhật. Nó có thể cùng sinh ra cùng lúc.

+ Composition

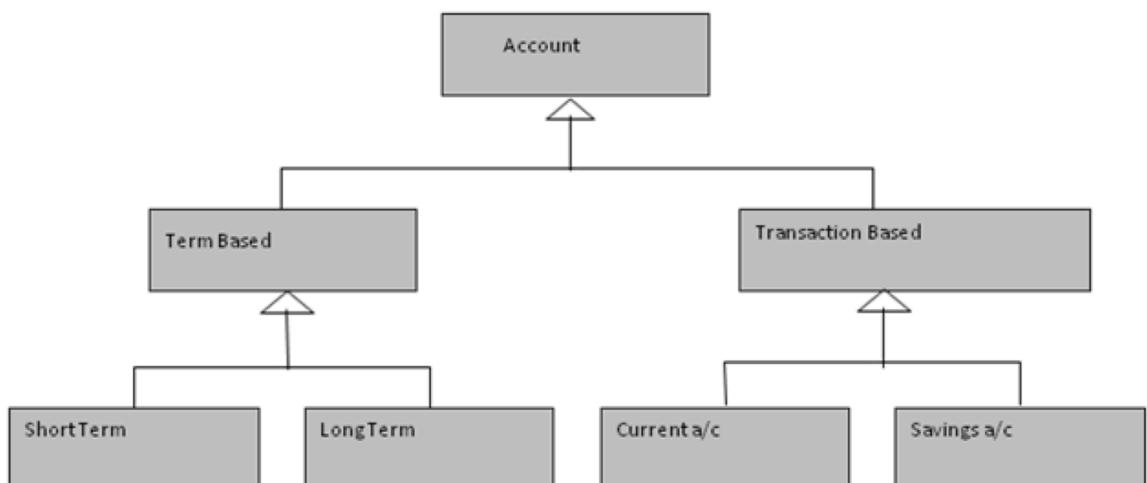
Composition là một loại mạnh hơn của Aggregation thể hiện quan hệ class này là một phần của class kia nên dẫn đến cùng tạo ra hoặc cùng chết đi.



Ví dụ trên class Mailing Address là một phần của class Customer nên chỉ khi nào có đối tượng Customer thì mới phát sinh đối tượng Mailing Address.

+Generalization

Generalization là quan hệ thừa kế được sử dụng rộng rãi trong lập trình hướng đối tượng.



Các lớp ở cuối cùng như Short Term, Long Term, Current a/c, Savings a/c gọi là các lớp cụ thể (concrete Class). Chúng có thể tạo ra đối tượng và các đối tượng này thừa kế toàn bộ các thuộc tính, phương thức của các lớp trên.

Các lớp trên như Account, Term Based, Transaction Based là những lớp trừu tượng (Abstract Class), những lớp này không tạo ra đối tượng.

Ngoài ra, còn một số quan hệ khác như dependence, realization nhưng ít được sử dụng nên chúng ta không bàn ở đây.

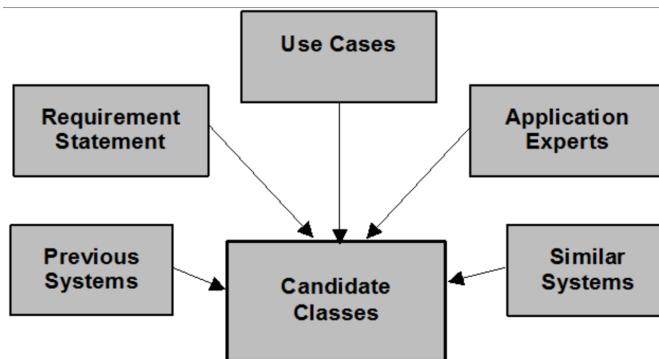
II. Cách xây dựng biểu đồ Class

Class Diagram là biểu đồ khó xây dựng nhất so với các biểu đồ khác trong OOAD và UML. Bạn phải hiểu được hệ thống một cách rõ ràng và có kinh nghiệm về lập trình hướng đối tượng mới có thể xây dựng thành công biểu đồ này.

Thực hiện theo các bước sau đây để xây dựng Class Diagram.

Bước 1: Tìm các Classes dự kiến

Entity Classes(các lớp thực thể) là các thực thể có thật và hoạt động trong hệ thống, bạn dựa vào các nguồn sau để xác định chúng.



Hình 27. Các nguồn thông tin có thể tìm Class dự kiến

- Requirement statement: Các yêu cầu. Chúng ta phân tích các danh từ trong các yêu cầu để tìm ra các thực thể.
- Use Cases: Phân tích các Use Case sẽ cung cấp thêm các Classes dự kiến.
- Previous và Similar System: có thể sẽ cung cấp thêm cho bạn các lớp dự kiến.
- Application Experts: các chuyên gia ứng dụng cũng có thể giúp bạn.

Xem xét, ví dụ ATM ở trên chúng ta có thể thấy các đối tượng là Entity Class như sau:

- Customers: khách hàng giao dịch là một thực thể có thật và quản lý trong hệ thống.
- Accounts: Tài khoản của khách hàng cũng là một đối tượng thực tế.
- ATM Cards: Thẻ dùng để truy cập ATM cũng được quản lý trong hệ thống.
- ATM Transactions: Các giao dịch được lưu giữ lại, nó cũng là một đối tượng có thật.
- Banks: Thông tin ngân hàng bạn đang giao dịch, nếu có nhiều nhà Bank tham gia vào hệ thống bạn phải quản lý nó. Lúc đó Bank trở thành đối tượng bạn phải quản lý.
- ATM: Thông tin ATM bạn sẽ giao dịch. Nó cũng được quản lý tương tự như Banks.

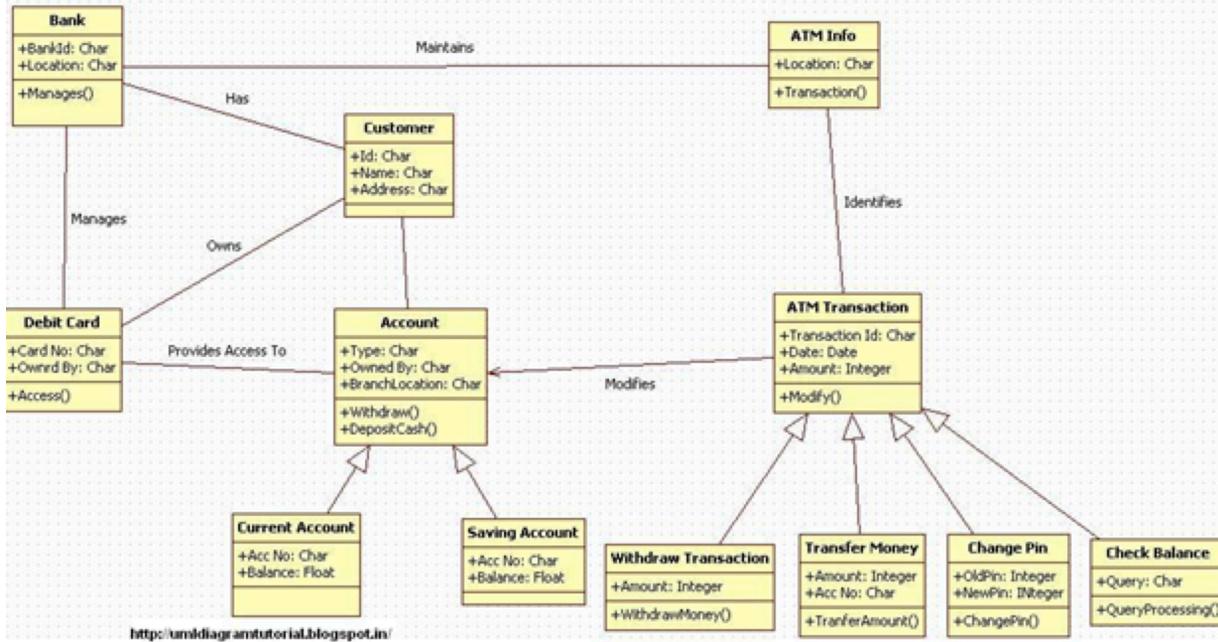
Lưu ý: Chỉ các thực thể bên trong hệ thống được xem xét, các thực thể bên ngoài hệ thống không được xem xét. Ví dụ Customers là những người khách hàng được quản lý trong hệ thống chứ không phải người dùng máy ATM bên ngoài. Bạn phải lưu ý điều này để phân biệt Class và Actor.

Bước 2: Tìm các thuộc tính và phương thức cho lớp

- Tìm thuộc tính: phân tích thông tin từ các form mẫu có sẵn, bạn sẽ tìm ra thuộc tính cho các đối tượng của lớp. Ví dụ các thuộc tính của lớp Customer sẽ thể hiện trên Form đăng ký thông tin khách hàng.
- Tìm phương thức: phương thức là các hoạt động mà các đối tượng của lớp này có thể thực hiện. Chúng ta sẽ bổ sung phương thức đầy đủ cho các lớp khi phân tích Sequence Diagram sau này.

Bước 3: Xây dựng các quan hệ giữa các lớp và phát hiện các lớp phát sinh

- Phân tích các quan hệ giữa các lớp và định nghĩa các lớp phát sinh do các quan hệ sinh ra. Chúng ta phân tích các thực thể ở trên và nhận thấy.
 - Lớp Accounts có thể chia thành nhiều loại tài khoản như Current Accounts và Saving Accounts và có quan hệ thừa kế với nhau.
 - Lớp ATM Transactions cũng có thể chia thành nhiều loại giao dịch như Deposit, Withdraw, Transfer... và chúng cũng có quan hệ thừa kế với nhau.
- Tách chúng ta và vẽ chúng lên biểu đồ chúng ta sẽ có Class Diagram cho hệ thống ATM như sau:



Hình 28. Ví dụ về Class Diagram cho hệ thống ATM

4. ĐẶC TẢ CLASS

Nhìn vào Class Diagram chúng ta có thể thấy cấu trúc của hệ thống gồm những lớp nào nhưng để cài đặt chúng, chúng ta phải đặc tả chi tiết hơn nữa. Trong đó, cần mô tả:

- Các thuộc tính: Tên, kiểu dữ liệu, kích thước
- Các phương thức:
 - Tên
 - Mô tả
 - Tham số đầu vào: Tên, kiểu dữ liệu, kích thước
 - Kết quả đầu ra: Tên, kiểu dữ liệu, kích thước
 - Luồng xử lý
 - Điều kiện bắt đầu
 - Điều kiện kết thúc

Tuy nhiên, việc này cũng mất khá nhiều thời gian. Nếu phát triển theo mô hình Agile thì bạn không phải làm việc này mà các thành viên phát triển phải nắm điều này để cài đặt.

III. SỬ DỤNG BIỂU ĐỒ CLASS

Có thể tóm tắt một số ứng dụng của biểu đồ Class Diagram như sau:

- Hiểu cấu trúc của hệ thống
- Thiết kế hệ thống
- Sử dụng để phân tích chi tiết các chức năng (Sequence Diagram, State Diagram v.v...)

- Sử dụng để cài đặt (coding)

IV. Kết luận

Như vậy, chúng ta đã tìm hiểu xong về Class Diagram, các bạn cần thực hành nhiều để hiểu về biểu đồ quan trọng này.

Để giúp các bạn nắm rõ hơn về Class Diagram, trong bài tiếp theo chúng ta sẽ thực hành xây dựng Class Diagram cho hệ thống eCommerce đã mô tả trong Case Study ở bài sau.

THỰC HÀNH XÂY DỰNG BIỂU ĐỒ LỚP

Chúng ta đã nắm được khái niệm, các thành phần cũng như cách xây dựng Class Diagram thông qua ví dụ về xây dựng phần mềm cho ATM. Bài này, chúng ta sẽ bàn kỹ hơn về cách xây dựng Class Diagram cho ứng dụng eCommerce với hy vọng giúp các bạn nắm rõ hơn cách xây dựng biểu đồ này.

Hãy xem lại các yêu cầu và Use Case Diagram của hệ thống eCommerce chúng ta đã bàn ở bài số 3 của chuyên mục này.

1. Xây dựng Class Diagram cho hệ thống eCommerce

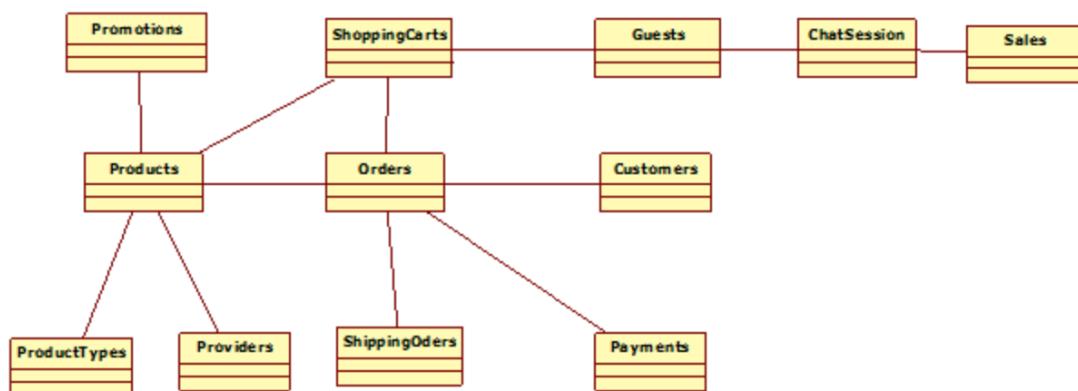
Bước 1: Tìm các Classes dự kiến

Nghiên cứu kỹ các yêu cầu, Use Case và nghiên cứu kỹ các hệ thống tương tự để xác định các lớp dự kiến thông qua việc xác định các đối tượng có trong hệ thống.

Xem xét Use Case Diagram của hệ thống:

- Phân tích Use Case “Xem sản phẩm” chúng ta xác định thực thể sản phẩm (Products). Sản phẩm được phân loại theo chủng loại (Product Types) và Nhà sản xuất (Providers) nên đây có thể là 2 lớp có quan hệ với class Products.
- Xem xét Use Case “Xem khuyến mãi” xác định Class Chương trình khuyến mãi (Promotions)
- Use Case “Quản lý giỏ hàng” -> Class giỏ hàng (Shopping Carts)
- Use Case Chat -> Class Chat session. Những người dùng tham gia Chat là Sales và Guest có thể là hai class dự kiến.
- Use Case “Đăng ký thành viên” -> Khách hàng (Customers)
- Use Case “Quản lý đơn hàng” -> Class đơn hàng (Orders), class thu tiền (Payments) và Quản lý chuyển hàng (Shipping Orders) có thể là 2 lớp có liên quan với Class Orders.

Tạm thời vẽ và xác định quan hệ sơ bộ chúng ta có biểu đồ Class dự kiến như sau:



Hình 29. Class Diagram sơ bộ khi phân tích các Use Case

Biểu đồ sơ bộ này giúp chúng ta có cái nhìn cơ bản về cấu trúc hệ thống để tiếp tục phân tích. Tất nhiên, bạn cần phân tích tất cả các Use Case còn lại và tìm hiểu thêm về hệ thống để bổ sung đầy đủ Class dự kiến cho hệ thống.

Bước 2: Xác định thuộc tính và quan hệ cho các lớp

Chúng ta bổ sung các thuộc tính cho các lớp và phân tích quan hệ của chúng.

- Products: xem xét tài liệu mô tả sản phẩm của hệ thống chúng ta có thể thấy Class Products cần những thuộc tính sau: Tên sản phẩm, mô tả, cấu hình, Giá bán, khuyến mãi, bảo hành (xem mô tả chi tiết sản phẩm trên Website)... Trong đó, thuộc tính giá thay đổi theo thời gian nên chúng ta nên tách ra thành lớp riêng là Giá (Prices). Tương tự thuộc tính khuyến mãi cũng được tách ra thành lớp Promotions.
- Prices: có các thuộc tính là Mã sản phẩm, Giá, ngày bắt đầu, ngày hết hạn.
- Promotions: tương tự như giá nó cần có lớp riêng với các thuộc tính là Mã sản phẩm, Mô tả khuyến mãi, Giá trị khuyến mãi, Ngày bắt đầu, Ngày hết hạn.
- ProductTypes: chứa loại sản phẩm
- Providers: chứa tên nhà sản xuất
- ShoppingCarts: chứa các thông tin như: cartID, ngày, mã sản phẩm, số lượng, đơn giá. Chúng ta nhận thấy nếu để nguyên lớp này khi tạo đối tượng chúng sẽ lặp thông tin cartID và ngày mua nên tách chúng ra thành ShoppingCarts với các thuộc tính CartID, ngày và CartDetails với các thuộc tính ProductID, số lượng, đơn giá.
- Tương tự chúng ta có class Orders với OrderID, ngày, customerID và class Orderdetails với ProductID, số lượng, đơn giá.
- Payments: chứa các thông tin như PaymentID, OrderID, ngày trả, số tiền, hình thức thanh toán.
- Shippings: có thể chứa ShippingID, OrderID, Ngày chuyển, ngày đến, số tiền, phương thức vận chuyển.
- Customers: CustomerID, Họ và tên, địa chỉ, điện thoại, ngày đăng ký v.v...
- Guests: có thể chứa sessionID để xác định thông tin khi chat
- Sales: có thể gộp với lớp người dùng (Users) chứa UserID, Name
- ChatSessions: ChatsessionID, tên người bán hàng, mã khách, mã tin nhắn, nội dung tin nhắn, ngày.

Nhập đầy đủ thuộc tính và vẽ chúng ra, chúng ta có biểu đồ như sau:

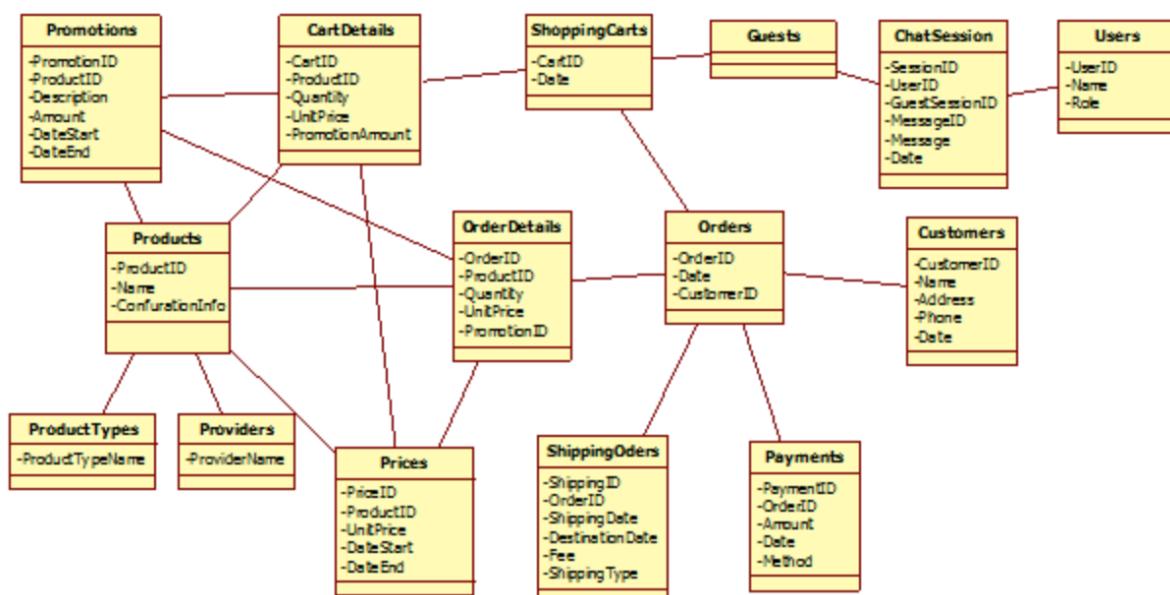
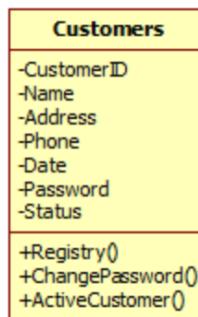


Figure 30. Class Diagram sau khi thêm thuộc tính và tách các quan hệ

Bước 3: Bổ sung phương thức cho các lớp

Phương thức là các hành động mà đối tượng sinh ra từ lớp đó có thể thực hiện trong hệ thống. Ví dụ các đối tượng của lớp Customers có thể đăng ký mới, có thể thay đổi mật khẩu (password), kích hoạt người dùng (Active),...

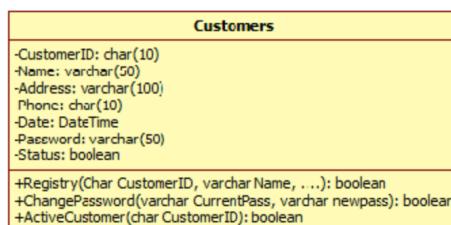


Hình 31. Ví dụ về các thuộc tính và phương thức

Có một vấn đề ở đây là chúng ta rất khó xác định chính xác các phương thức của một lớp. Nếu thiếu bạn sẽ không thể cài đặt đủ yêu cầu chức năng, nếu thừa bạn tốn công cài đặt vô ích mà không dùng đến. Bạn có thể đổi chiểu khi phân tích và thiết kế tất cả các Use Case của hệ thống. Do vậy, chúng ta có thể bổ sung dần các phương thức cho các lớp khi sử dụng Activity Diagram và Sequence Diagram để phân tích Use Case. Các biểu đồ này sẽ bàn ở các bài tiếp theo.

Khi đã có được Class Diagram, bạn cần thiết kế chi tiết các lớp bằng cách đặc tả các thuộc tính và phương thức của nó.

- Đặc tả thuộc tính: chúng ta xác định kiểu dữ liệu và kích thước.
- Đặc tả phương thức: chúng ta xác định dữ liệu đầu vào, dữ liệu đầu ra.



Hình 32. Thiết kế chi tiết các thuộc tính và phương thức cho lớp

Việc sử dụng các kiểu dữ liệu và mô tả các phương thức của một lớp chúng ta đã học kỹ trong lập trình hướng đối tượng nên chúng ta không bàn ở đây. Hoàn tất các bước trên cho toàn bộ các Use Case chúng ta sẽ có biểu đồ Class hoàn chỉnh.

2. Kết luận

Bài này đã hướng dẫn các bước để xây dựng một biểu đồ Class cho hệ thống eCommerce. Tiếp tục thực hiện các bước còn lại để có được biểu đồ Class hoàn chỉnh.

Trong bài tiếp theo chúng ta sẽ phân tích hoạt động của hệ thống thông qua các biểu đồ của mô hình động (Dynamic Model) như Activity Diagram, Sequence Diagram, ...

BÀI 5. BIỂU ĐỒ HOẠT ĐỘNG (ACTIVITY DIAGRAM)

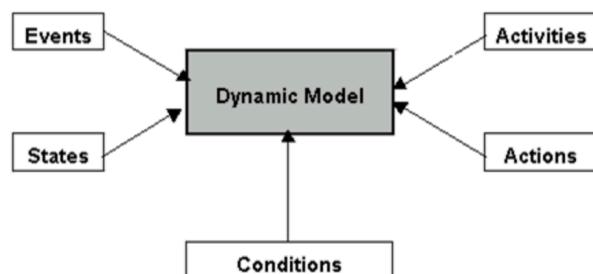
Chúng ta đã tìm hiểu kiến trúc hệ thống qua Use Case Diagram, cấu trúc hệ thống thông qua Class Diagram. Bài này, chúng ta sẽ đi vào phân tích khía cạnh hoạt động trong hệ thống. Theo UML 2.0 thì hệ thống có thể được mô tả theo 2 mô hình tĩnh (Static Model) và mô hình động (Dynamic Model).

Static Model: mô tả cấu trúc của hệ thống bao gồm các biểu đồ Class Diagram, Object Diagram, Component Diagram và Deployment Diagram.

Dynamic Model: mô tả các hoạt động bên trong hệ thống bao gồm các biểu đồ Activity Diagram, State Diagram, Sequence Diagram, Collaboration Diagram.

Trong bài này chúng ta chỉ bàn về hai biểu đồ của mô hình động được sử dụng thường xuyên trong thiết kế hệ thống phần mềm là Activity Diagram và Sequence Diagram. Các biểu đồ khác các bạn tự tìm hiểu trong tài liệu tham khảo.

I. Các thành phần cơ bản của Dynamic Model



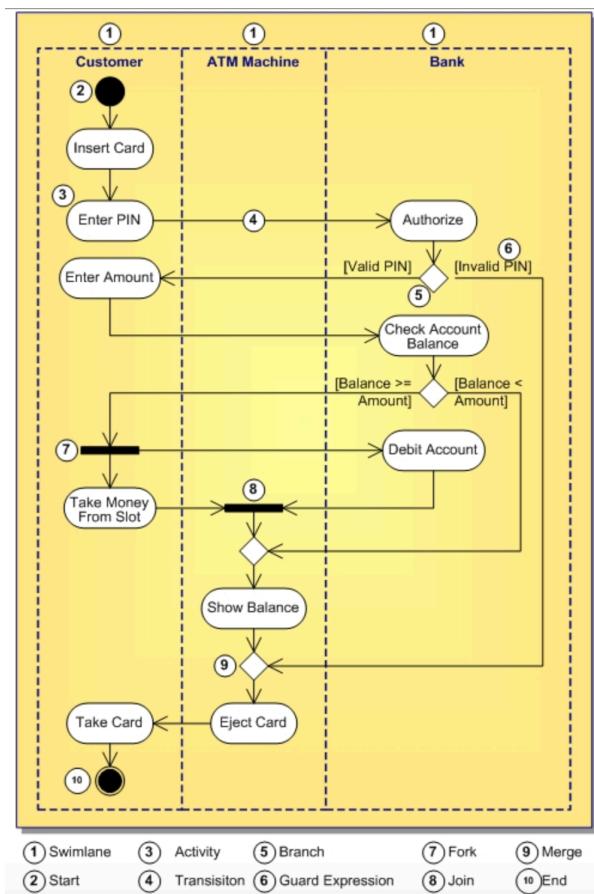
Hình 33. Các thành phần của mô hình động

- Event: là sự kiện, mô tả một hoạt động bên ngoài tác động vào đối tượng và được đối tượng nhận biết và có phản ứng lại.
- Activity: mô tả một hoạt động trong hệ thống. Hoạt động có thể do một hoặc nhiều đối tượng thực hiện.
- State: là trạng thái của một đối tượng trong hệ thống, được mô tả bằng giá trị của một hoặc nhiều thuộc tính.
- Action: chỉ hành động của đối tượng.
- Condition: mô tả một điều kiện.

II. Activity Diagram

Activity Diagram là biểu đồ tập trung vào mô tả các hoạt động, luồng xử lý bên trong hệ thống. Nó có thể được sử dụng để mô tả các qui trình nghiệp vụ trong hệ thống, các luồng của một chức năng hoặc các hoạt động của một đối tượng.

Chúng ta xem một ví dụ Activity Diagram về hoạt động rút tiền từ ATM như sau:

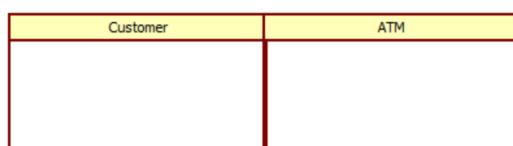


Hình 34. Ví dụ về Activity Diagram của hoạt động rút tiền từ ATM

Chúng ta thấy chúng có các ký hiệu sau:

1) Swimlane

Swimlane được dùng để xác định đối tượng nào tham gia hoạt động nào trong một qui trình. Ví dụ ở trên Customer thì Insert Card còn ATM Machine thì Show Balance.



Hình 35. Ký hiệu về Swimlane

2) Nút Start, End

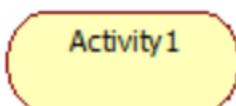
Start thể hiện điểm bắt đầu qui trình, End thể hiện điểm kết thúc qui trình.



Ký hiệu nút Start và End

3) Activity

Activity mô tả một hoạt động trong hệ thống. Các hoạt động này do các đối tượng thực hiện.



Hình 36. Ký hiệu về Activity

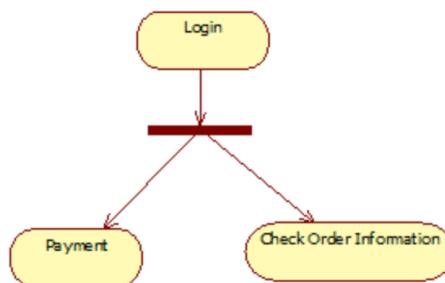
4) Branch

Branch thể hiện rẽ nhánh trong mệnh đề điều kiện.



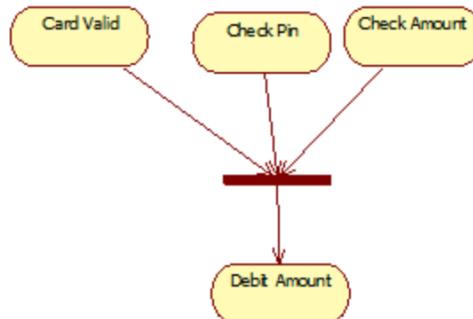
5) Fork

Fork thể hiện cho trường hợp thực hiện xong một hoạt động rồi sẽ rẽ nhánh thực hiện nhiều hoạt động tiếp theo.



6) Join

Cùng ký hiệu với Fork nhưng thể hiện trường hợp phải thực hiện hai hay nhiều hành động trước rồi mới thực hiện hành động tiếp theo.



III. Cách xây dựng Activity Diagram

Thực hiện các bước sau đây để xây dựng biểu đồ Activity Diagram.

Bước 1: Xác định các nghiệp vụ cần mô tả

Xem xét biểu đồ Use Case để xác định nghiệp vụ nào bạn cần mô tả.

Bước 2: Xác định trạng thái đầu tiên và trạng thái kết thúc

Bước 3: Xác định các hoạt động tiếp theo

Xuất phát từ điểm bắt đầu, phân tích để xác định các hoạt động tiếp theo cho đến khi gặp điểm kết thúc để hoàn tất biểu đồ này.

Bạn có thể hỏi chuyên gia, học hệ thống tương tự, hỏi khách hàng để nắm rõ về qui trình của hệ thống.

IV. Sử dụng biểu đồ Activity Diagram

- Phân tích nghiệp vụ để hiểu rõ hệ thống
- Phân tích Use Case
- Cung cấp thông tin để thiết kế biểu đồ Sequence Diagram

V. Kết luận

Như vậy, chúng ta đã tìm hiểu được Activity Diagram, một biểu đồ quan trọng mô tả hoạt động của hệ thống. Chúng ta sẽ tiếp tục bàn về thực hành xây dựng biểu đồ này cho một ứng dụng cụ thể trong bài tiếp theo.

THỰC HÀNH XÂY DỰNG ACTIVITY DIAGRAM

Trong bài trước chúng ta đã nắm được khái niệm, các thành phần, cách xây dựng và ứng dụng của Activity Diagram. Nay giờ, chúng ta áp dụng nó để phân tích nghiệp vụ cho hệ thống eCommerce mà chúng ta đã xem xét trong bài số 3 của chuyên mục này.

Xây dựng Activity Diagram cho hệ thống eCommerce:

Bước 1: Xác định các nghiệp vụ cần phân tích.

Trước tiên, chúng ta xem xét các Use Case. Về nguyên tắc bạn phải phân tích và mô tả tất cả các nghiệp vụ của hệ thống để làm rõ hệ thống.

Xem xét biểu đồ Use Case Diagram chúng ta đã vẽ ở bài 3, chúng ta có thể thấy các Use Case sau cần làm rõ:

- Xem sản phẩm theo chủng loại
- Thêm sản phẩm theo nhà cung cấp
- Thêm giỏ hàng
- Chat
- Quản lý đơn hàng
- Thanh toán
- Theo dõi chuyển hàng
- Đăng nhập

Tiếp theo, chúng ta bắt đầu phân tích và vẽ cho chức năng xem sản phẩm theo chủng loại.

Bước 2: Xác định các bước thực hiện và đối tượng liên quan

Để thực hiện chức năng xem sản phẩm theo chủng loại hệ thống sẽ thực hiện như sau:

- Điều kiện ban đầu: ở trang chủ
- Điều kiện kết thúc: hiển thị xong trang sản phẩm

Các bước như sau:

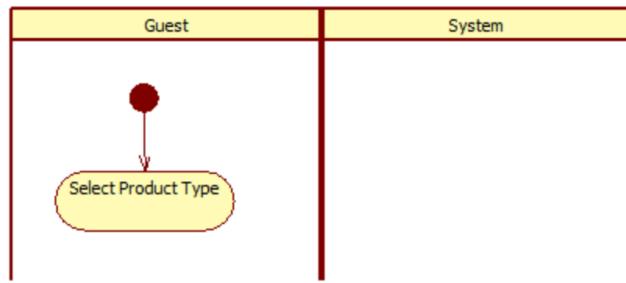
- Người dùng chọn loại sản phẩm.
- Hệ thống sẽ lọc lấy loại sản phẩm tương ứng, sau đó lấy giá, lấy khuyến mãi cho tất cả các sản phẩm đã được chọn và hiển thị lên màn hình.
- Người dùng xem sản phẩm.

Bước 3: Thực hiện biểu đồ

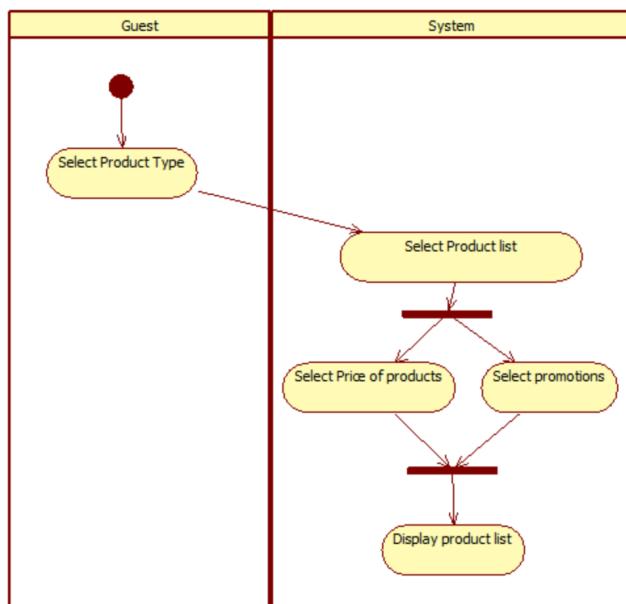
Chúng ta thấy có 2 đối tượng tham gia vào giao dịch này là Người dùng và Hệ thống. Chúng ta nên dùng Swimlane để thể hiện 2 đối tượng trên.



- Xác định trạng thái đầu tiên.
- Hành động tiếp theo là Guest chọn loại sản phẩm

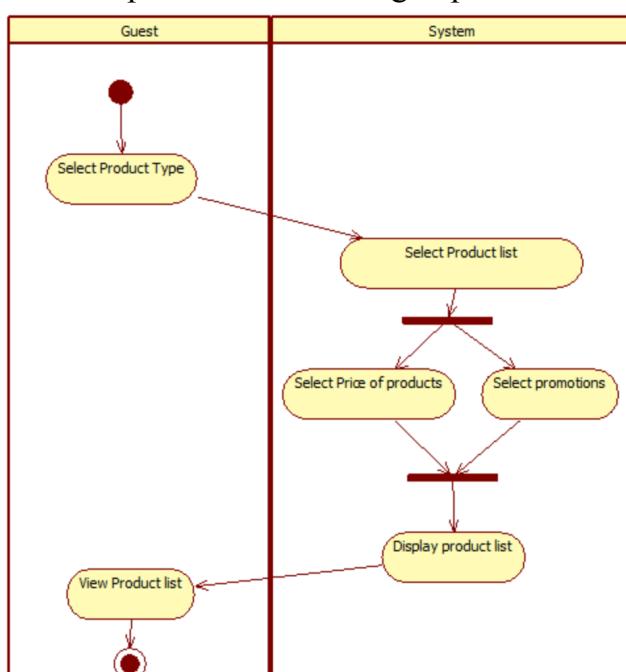


Hệ thống sẽ lấy danh sách sản phẩm tương ứng với loại đó, sau đó lấy giá, lấy khuyến mãi của chúng và hiển thị ra màn hình. Hành động lấy giá và khuyến mãi của sản phẩm có thể làm song song nên chúng ta dùng Fork và Join để thể hiện.



Hình 37. Hệ thống tập hợp danh sách sản phẩm và thông tin liên quan để hiển thị lên Browser

Người dùng xem danh sách sản phẩm và kết thúc nghiệp vụ của Use Case này.



Tương tự, như vậy bạn hãy hoàn tất Activity Diagram cho các nghiệp vụ còn lại cho hệ thống.

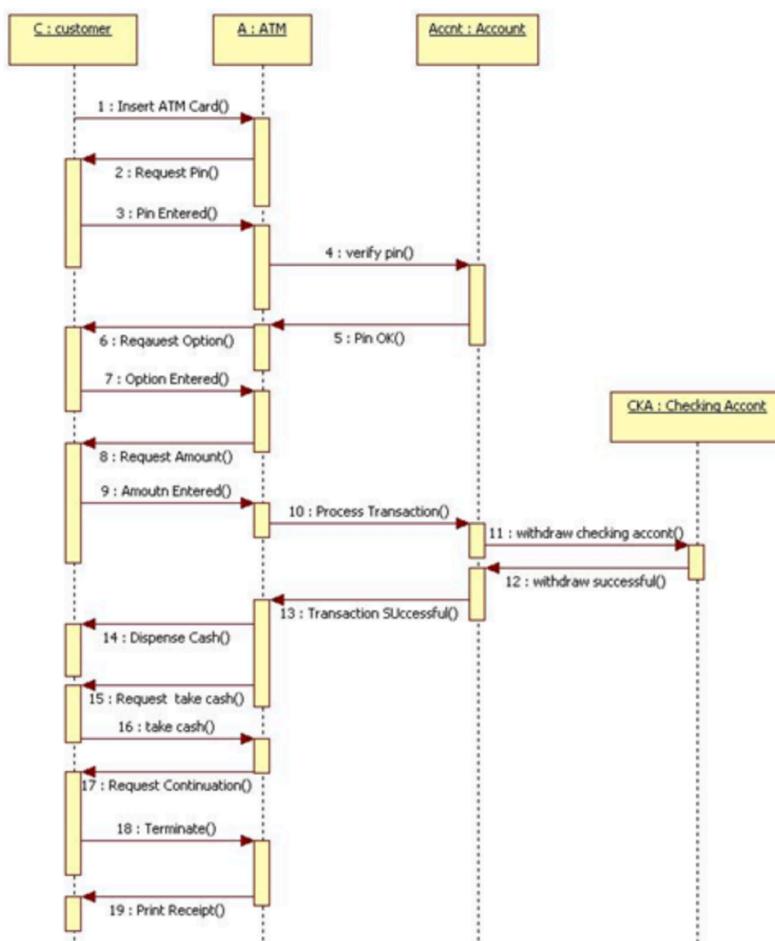
BÀI 6. BIỂU ĐỒ TUẦN TỤ (SEQUENCE DIAGRAM)

Trong bài trước chúng ta đã biết cách sử dụng Activity Diagram để phân tích nghiệp vụ của hệ thống. Trong bài này chúng ta sẽ sử dụng Sequence để thiết kế chi tiết chức năng cho hệ thống.

I. Khái niệm

Sequence Diagram là biểu đồ mô tả sự tương tác của các đối tượng để tạo nên các chức năng của hệ thống. Biểu đồ này mô tả sự tương tác theo thời gian nên rất phù hợp với việc sử dụng để thiết kế và cài đặt chức năng cho hệ thống phần mềm.

Chúng ta hãy xem một ví dụ Sequence Diagram.

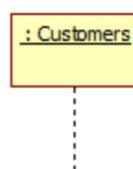


Hình 38. Ví dụ Sequence Diagram cho hoạt động rút tiền ở ATM

II. Các thành phần của Sequence Diagram

1. Objects

Object mô tả một đối tượng trong hệ thống. Để phân biệt với Class, Object có dấu “:” phía trước tên của nó.



Đường gạch chấm bên dưới đối tượng thể hiện thời gian sống của đối tượng.

2. Stimulus (message)

Stimulus thể hiện thông điệp từ một đối tượng này tương tác với một đối tượng khác.

Ký hiệu về Stimulus trong biểu đồ Sequence Diagram như sau:



3. Axes

Trục tọa độ, trục ngang thể hiện các đối tượng, trục đứng thể hiện thời gian.

Chúng ta, dễ dàng nhận thấy các đối tượng tương tác với nhau theo tuần tự các bước để hình thành nên chức năng của hệ thống.

III. Xây dựng Sequence Diagram

Để xây dựng Sequence Diagram chúng ta thực hiện các bước sau:

Bước 1: Xác định chức năng cần thiết kế. Bạn dựa vào Use Case Diagram để xác định xem chức năng nào cần thiết kế.

Bước 2: Dựa vào Activity Diagram để xác định các bước thực hiện theo nghiệp vụ.

Bước 3: Đối chiếu với Class Diagram để xác định lớp trong hệ thống tham gia vào nghiệp vụ.

Bước 4: Vẽ Sequence Diagram

Bước 5: Cập nhật lại biểu đồ Class Diagram

IV. Ứng dụng Sequence Diagram

- Thiết kế các chức năng
- Kiểm chứng và bổ sung method cho các Class
- Sử dụng trong việc coding các chức năng

V. Kết luận

Chúng ta vừa tìm hiểu xong biểu đồ Sequence Diagram, biểu đồ này giúp thiết kế các chức năng cho hệ thống cũng như kiểm chứng các biểu đồ trước đây như Class Diagram, Activity Diagram v.v...

Chúng ta sẽ tiếp tục bàn về thực hành xây dựng biểu đồ này cho ứng dụng ecommerce trong bài tiếp theo.

THỰC HÀNH XÂY DỰNG SEQUENCE DIAGRAM

Trong bài trước chúng ta đã tìm hiểu về Sequence Diagram, các thành phần, cách xây dựng và ứng dụng của nó. Trong bài này, chúng ta sẽ bàn về cách ứng dụng sequence diagram để thiết kế cho hệ thống eCommerce mà chúng ta đã bàn ở bài 3 của chuyên mục này.

1. Xây dựng Sequence Diagram

Bước 1: Xác định các Use Case cần thiết kế

Tương tự như Activity Diagram, chúng ta cũng cần xác định các Use Case mà chúng ta cần sử dụng sequence Diagram để thiết kế chi tiết.

Xem xét biểu đồ Use Case Diagram chúng ta đã vẽ ở bài 3, chúng ta có thể thấy các Use Case sau cần thiết kế:

- Xem sản phẩm theo chủng loại
- Thêm sản phẩm theo nhà cung cấp
- Thêm giỏ hàng
- Chat
- Quản lý đơn hàng
- Thanh toán
- Theo dõi chuyển hàng
- Đăng nhập

Tiếp theo, chúng ta sẽ thiết kế cho chức năng “Xem sản phẩm theo chủng loại”.

Bước 2: Xem Activity Diagram cho Use Case này chúng ta xác định các bước sau:

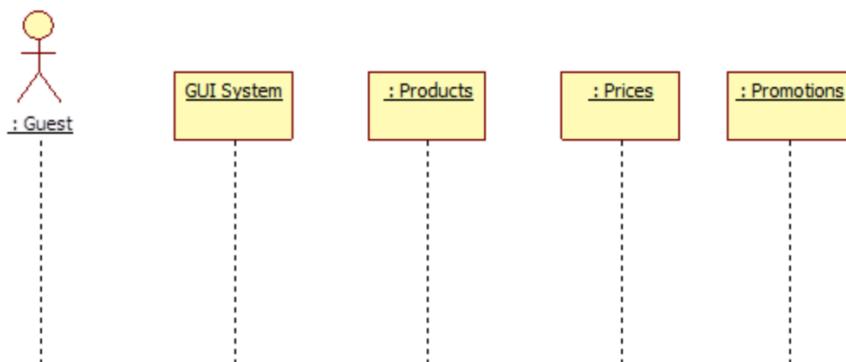
- Người dùng chọn loại sản phẩm
- Hệ thống sẽ lọc lấy loại sản phẩm tương ứng, sau đó lấy giá, lấy khuyến mãi và hiển thị lên màn hình.
- Người dùng xem sản phẩm

Bước 3: Đối chiếu với Class Diagram chúng ta xác định các đối tượng thực hiện như sau:

- Người dùng: chọn loại sản phẩm qua giao diện
- Giao diện: sẽ lấy danh sách sản phẩm tương ứng từ Products
- Giao diện: lấy giá của từng sản phẩm từ Class Prices và Promotion Amount từ lớp Promotions
- Giao diện: tổng hợp danh sách và hiển thị
- Người dùng: Xem sản phẩm

Bước 4: Vẽ sequence Diagram

- Xác định các lớp tham gia vào hệ thống gồm: người dùng (Guest), Giao diện (GUI System), Sản phẩm (Products), Giá (Prices), Khuyến mãi (Promotions). Trong đó GUI System để sử dụng chung cho giao diện, bạn có thể sử dụng cụ thể trang Web nào nếu bạn đã có Mockup (thiết kế chi tiết của giao diện).



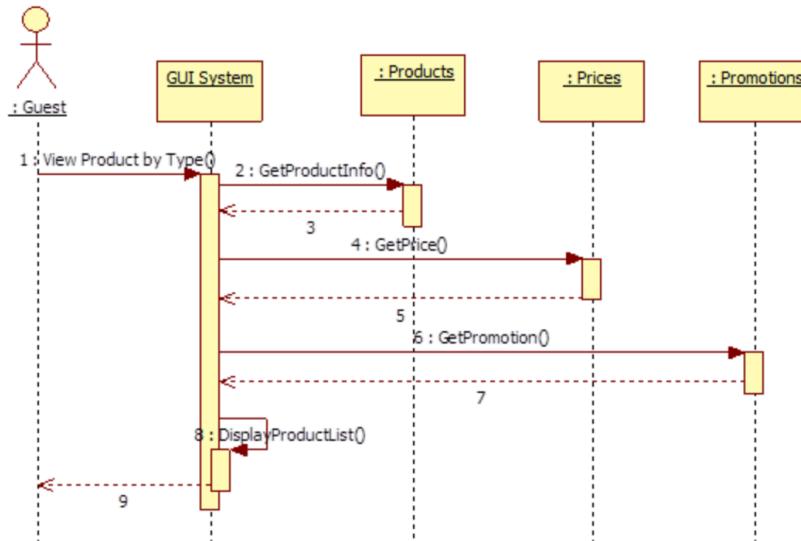
Hình 39. Xác định các đối tượng tham gia vào biểu đồ

Các bước thực hiện của Use Case này như sau:

- Guest gửi yêu cầu xem sản phẩm lên giao diện kèm theo chủng loại

- GUI system: gửi yêu cầu lấy danh sách các sản phẩm tương ứng với chủng loại cho lớp sản phẩm và nhận lại danh sách.
- GUI system: gửi yêu cầu lấy Giá cho từng sản phẩm từ Prices
- GUI system: gửi yêu cầu lấy khuyến mãi cho từng sản phẩm từ Promotions và nhận lại kết quả
- GUI system: ghép lại danh sách và hiển thị lên browser và trả về cho Guest

Thể hiện lên biểu đồ như sau:



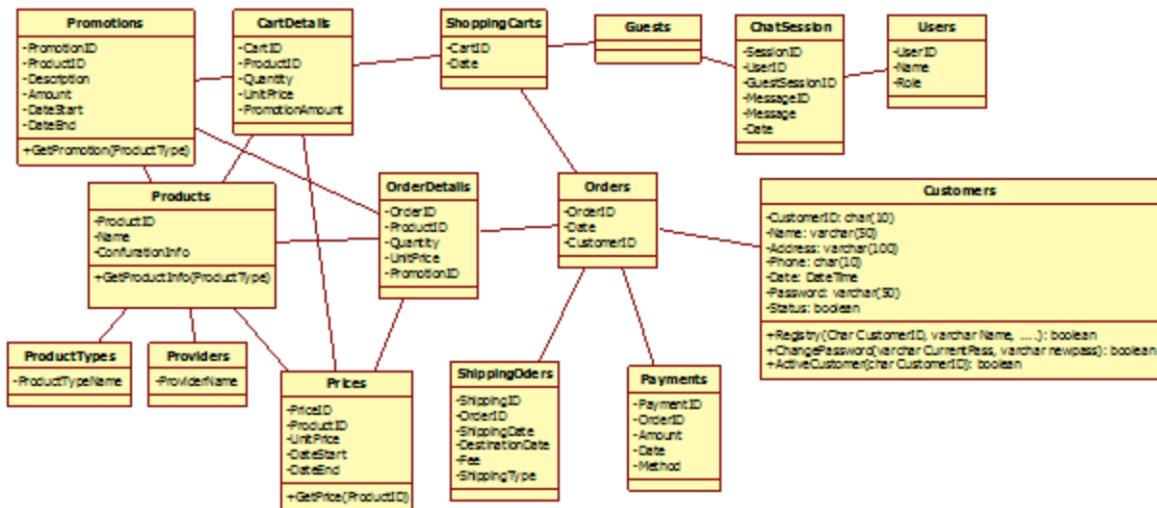
Hình 40. Biểu đồ Sequence Diagram cho chức năng Xem sản phẩm theo loại

Bước 5: Kiểm tra và cập nhật biểu đồ Class Diagram

Chúng ta nhận thấy để thực hiện được biểu đồ trên chúng ta cần bổ sung các phương thức cho các lớp như sau:

- Products class: bổ sung phương thức GetProductInfo(Product Type): trả về thông tin sản phẩm có loại được truyền vào. Việc này các đối tượng của lớp Products hoàn toàn làm được vì họ đã có thuộc tính ProductType nên họ có thể trả về được thông tin này.
- Prices: bổ sung phương thức GetPrice(ProductID): UnitPrice. Sau khi lấy được ProductID từ Products, GUI gọi phương thức này để lấy giá của sản phẩm từ lớp giá. Các đối tượng từ lớp Prices hoàn toàn đáp ứng điều này.
- Promotions: tương tự bổ sung phương thức GetPromotion(ProductID).
- GUI System(View Product Page): bổ sung phương thức DisplayProductList(List of product)để hiển thị danh sách lên sản phẩm. Ngoài ra, bạn cần có thêm một phương thức ViewProductbyType(ProductType) để mô tả chính hoạt động này khi người dùng kích chọn.

Như vậy, chúng ta thấy các phương thức trên đều thực hiện được trên các đối tượng của các lớp nên thiết kế trên là khả thi. Bổ sung các phương thức trên vào các Class tương ứng chúng ta có biểu đồ Class Diagram như sau:



Hình 41. Class Diagram sau khi đã bổ sung các phương thức mới

Ngoài ra, bạn có thể bổ sung các lớp giao diện vào Class Diagram để hoàn chỉnh thiết kế cho hệ thống.

Hoàn tất sequence diagram cho tất cả các Use Case chúng ta sẽ hoàn thành việc thiết kế, đồng thời cũng hoàn tất biểu đồ Class Diagram.

2. Kết luận

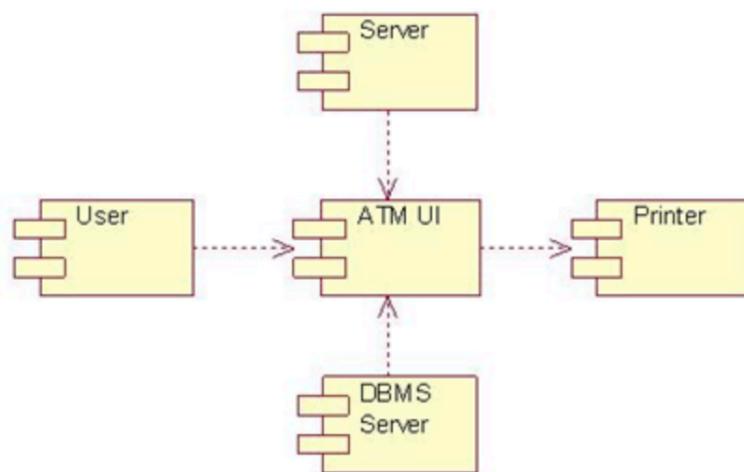
Biểu đồ Sequence Diagram có vai trò quan trọng trong việc thiết kế hệ thống. Đồng thời giúp chúng ta kiểm tra lại quá trình phân tích, thiết kế trước đây cũng như hoàn thành biểu đồ Class Diagram. Việc sử dụng thành thạo biểu đồ này giúp các bạn rất nhiều trong việc phân tích và thiết kế phần mềm.

Trong bài tiếp theo chúng ta sẽ bàn về Component Diagram và Deployment Diagram, những biểu đồ cuối cùng cho việc phân tích và thiết kế hướng đối tượng sử dụng UML.

BÀI 7. BIẾU ĐỒ THÀNH PHẦN (COMPONENT DIAGRAM)

Khi thiết kế các hệ thống phức tạp chúng ta nên chia chúng ra thành nhiều hệ thống con (subsystem) để dễ thiết kế. Mỗi hệ thống con sau khi xây dựng có thể đóng gói thành một thành phần phần mềm được triển khai độc lập. Biểu đồ Component Diagram sẽ giúp chúng ta thể hiện cách chia hệ thống ra nhiều thành phần và quan hệ của chúng.

Component Diagram là biểu đồ cho biết cấu trúc của hệ thống theo thành phần phần mềm. Chúng ta xem một ví dụ về Component Diagram như sau:



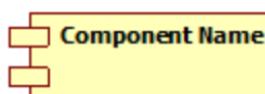
Hình 42. Ví dụ về Component Diagram cho ứng dụng máy ATM

Ví dụ trên cho thấy hệ thống phần mềm ATM chia ra thành 5 thành phần là ATM UI, Server, User, DBMS Server và Printer. Trong đó ATM UI sử dụng chức năng của các thành phần còn lại để vận hành hệ thống.

I. Các thành phần của Component Diagram

1. Component

Component là một thành phần phần mềm được đóng gói độc lập, nó có thể được triển khai độc lập trên hệ thống và có khả năng tương tác với các thành phần khác khi thực hiện các chức năng của hệ thống.



Hình 43. Ký hiệu về Component

2. Component Dependence

Component Dependence thể hiện quan hệ giữa các thành phần với nhau. Các thành phần phần mềm luôn cần sử dụng một số chức năng ở các thành phần khác trong hệ thống nên quan hệ Dependence được sử dụng thường xuyên.



II. Xây dựng Component Diagram

Để xây dựng biểu đồ Component chúng ta thực hiện các bước sau:

Bước 1: Chia hệ thống thành những SubSystem

Bước 2: Xác định các thành phần và vẽ

THỰC HÀNH XÂY DỰNG COMPOANANT DIAGRAM

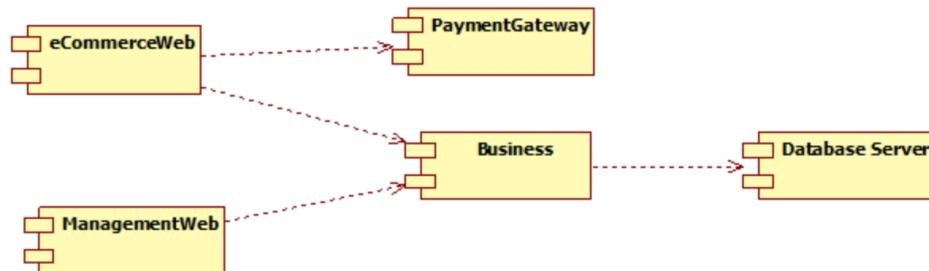
Xem xét ví dụ về hệ thống eCommerce chúng ta đã khảo sát ở bài 3 và thực hiện theo các bước sau:

Bước 1: Chia hệ thống thành các SubSystem như sau:

- Chia phần Website phục vụ cho đối tượng bên ngoài công ty là Guest và Customer ra một gói riêng để dễ triển khai và bảo mật. Thành phần này gọi là EcommerceWeb.
- Phần Website phục vụ cho đối tượng bên trong công ty được chia thành một gói gọi là ManagementWeb .
- Phần Business được sử dụng để tương tác CSDL và xử lý các nghiệp vụ.
- Phần PaymentGateway để xử lý thanh toán trực tuyến.
- Phần Database Server cũng được tách ra một gói riêng.

Lưu ý, việc phân chia này tùy thuộc vào nhu cầu tổ chức phát triển và triển khai của hệ thống. Bạn cần có kinh nghiệm về kiến trúc hệ thống khi tham gia thiết kế biểu đồ này.

Bước 2: Xác định quan hệ và vẽ ta được biểu đồ Component Diagram



Hình 44. Component Diagram cho hệ thống eCommerce

Việc chia ra các gói sẽ giúp chúng ta thuận tiện trong quá trình thiết kế, phát triển và triển khai. Bạn có thể triển khai mỗi thành phần trên một Server riêng để tăng năng lực cho hệ thống.

III. Ứng dụng của Component Diagram

Component được sử dụng vào các công việc sau:

- Thể hiện cấu trúc của hệ thống
- Cung cấp đầu vào cho biểu đồ Deployment
- Hỗ trợ cho việc thiết kế kiến trúc phần mềm

IV. Kết luận

Component Diagram là một biểu đồ đơn giản nhưng có ảnh hưởng lớn đến việc thiết kế chi tiết của hệ thống phần mềm và cách thức tổ chức phát triển sản phẩm. Do vậy, các bạn nên dành thời gian cho biểu đồ này để đảm bảo hiểu và dùng được nó sẽ có lợi về sau.

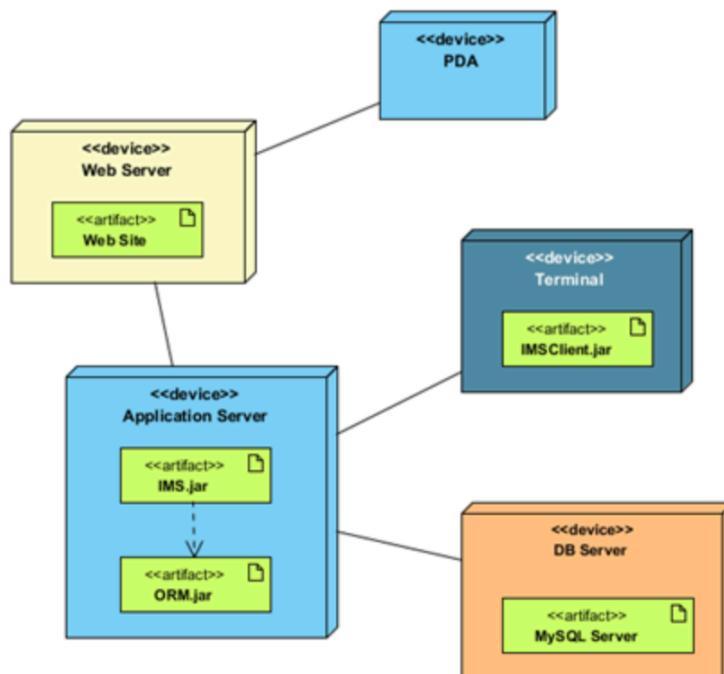
BÀI 8. BIẾU ĐỒ TRIỂN KHAI (DEPLOYMENT DIAGRAM)

I. Khái niệm

Deployment Diagram là biểu đồ 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.

Deployment Diagram thể hiện rõ kiến trúc triển khai nên nó sẽ ảnh hưởng đến sự thiết kế, phát triển, hiệu năng, khả năng mở rộng của hệ thống v.v...

Chúng ta xem một ví dụ về deployment diagram như sau:



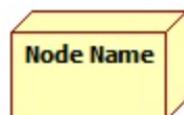
Hình 45. Ví dụ về Deployment Diagram

Biểu đồ trên mô tả hệ thống được triển khai trên 03 Server khác nhau gồm Webserver, Application Server, DB server và 02 thiết bị truy cập đầu cuối.

II. Các thành phần của Deployment Diagram

1. 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.



2. Relationship

Deployment Diagram sử dụng quan hệ Association và Dependence để thể hiện mối quan hệ giữa các node với nhau.

Ký hiệu về Association:



Ký hiệu về Dependence:

Việc xác định quan hệ giữa các thành phần và kết nối chúng lại với nhau chúng ta sẽ có biểu đồ Deployment Diagram.

III. Xây dựng Deployment Diagram

Thực hiện các bước sau đây để xây dựng biểu đồ Deployment Diagram.

Bước 1: Xác định các thành phần tham gia vào việc triển khai hệ thống

Việc này liên quan đến kiến trúc hệ thống, hiệu năng, khả năng mở rộng và cả vấn đề tài chính và hạ tầng của hệ thống nên bạn cần có kinh nghiệm về kiến trúc hệ thống để làm được việc này.

Bước 2: Xác định các thành phần để triển khai lên các Node

Khi đã có phân cứng, bước tiếp theo chúng ta xác định những component liên quan để triển khai trên mỗi node.

Bước 3: Xác định các quan hệ và hoàn tất biểu đồ

Xác định các mối quan hệ giữa các thành phần với nhau và nối chúng lại để hoàn tất biểu đồ.

THỰC HÀNH XÂY DỰNG DEPLOYMENT DIAGRAM

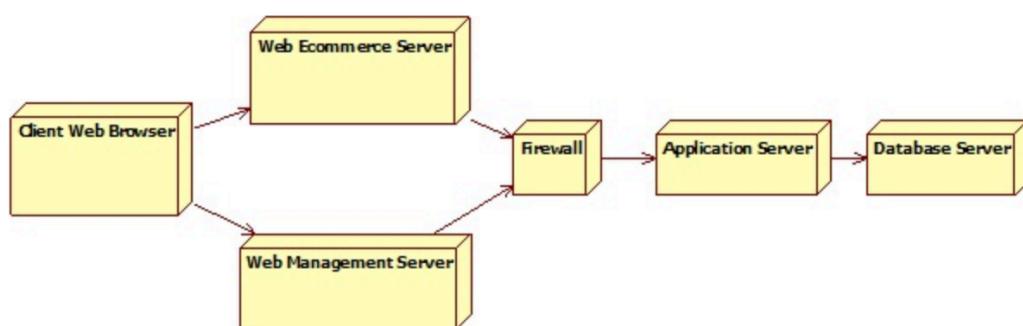
Xem xét hệ thống eCommerce mà chúng ta đã bàn ở bài 3 và tiến hành xây dựng biểu đồ Deployment Diagram cho hệ thống này. Thực hiện các bước sau đây:

Bước 1: Xác định các Node và bố trí các thành phần lên node

- Để tăng cường an ninh và sức chịu đựng cho hệ thống chúng ta bố trí phần cho người dùng bên ngoài công ty (Guest và Customer) ra một Server riêng gọi là Web eCommerce Server.
- Website chứa phần tương tác với nhân viên công ty đặt lên một Node riêng gọi là Web Management Server.
- Phần Business được đưa ra một Server ứng dụng gọi là Application Server.
- Database được đặt lên một Server gọi là Database Server.
- Phần PaymentGateWay có thể được đặt trên Web eCommerce Server.
- Bổ sung thêm các thiết bị bảo mật và hạ tầng.

Bước 2: Xác định quan hệ giữa các thành phần và hoàn tất biểu đồ

Xem xét các thành phần gọi với nhau để hoàn tất chức năng, chúng ta sẽ xác định các quan hệ của chúng. Biểu diễn lên biểu đồ chúng ta sẽ có Deployment Diagram như sau:



Hình 46. Biểu đồ deployment diagram cho hệ thống eCommerce

Nếu theo mô hình này, hệ thống cần 4 Server và 01 thiết bị mạng là Firewall để triển khai nhằm đảm bảo an ninh, sức chịu đựng khi có đông người sử dụng và khả năng mở rộng hệ thống. Tuy nhiên, nếu hệ thống của bạn ít người sử dụng và cần tiết kiệm chi phí, bạn có thể triển khai trên một hoặc hai Server chứ không cần thiết phải triển khai trên nhiều Server như mô hình trên. Ngoài ra, bạn cũng có thể hiệu chỉnh lại để biểu đồ trên để trở nên hấp dẫn và chi tiết hơn như hình vẽ trên.

4. Ứng dụng của Deployment Diagram

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.

5. Kết luận

Cũng như Component Diagram, Deployment Diagram là một biểu đồ khá đơn giản và dễ xây dựng nhưng có ảnh hưởng rất lớn đến quá trình phát triển, triển khai và kinh phí xây dựng dự án. Do vậy, các bạn nên dành thời gian cho biểu đồ này để tránh những khó khăn trong quá trình phát triển các sản phẩm phần mềm.

Chúng ta đã tìm hiểu và thực tập xây dựng các biểu đồ cần thiết trong quá trình thiết kế một hệ thống phần mềm. Trong bài tiếp theo chúng ta sẽ bàn về việc sử dụng công cụ để vẽ các biểu đồ UML nhằm tăng năng suất thiết kế.

BÀI 9. CÁC CÔNG CỤ ĐỂ XÂY DỰNG CÁC BIỂU ĐỒ UML

Trong các bài trước, chúng ta đã tìm hiểu các biểu đồ quan trọng và thường xuyên sử dụng trong quá trình phát triển phần mềm. Trong đó, có đề cập đến việc biểu diễn biểu đồ, việc này bạn có thể vẽ trên giấy hoặc sử dụng các công cụ đơn giản như Microsoft Word, Paint, ... Tuy nhiên, nếu bạn dùng một công cụ biểu diễn chuyên nghiệp sẽ giúp bạn tiết kiệm được nhiều thời gian cũng như thoải mái hơn trong việc thiết kế.

Trong bài này, chúng ta sẽ bàn về một số công cụ có thể dùng để biểu diễn và quản lý các biểu đồ UML một cách hiệu quả.

I. Giới thiệu

Có rất nhiều công cụ được sử dụng để vẽ các biểu đồ UML rất chuyên nghiệp như Rational Rose, Enterprise Architect, Microsoft Visio, ... và rất nhiều các công cụ phần mềm nguồn mở miễn phí có thể sử dụng tốt.

Các công cụ có cách sử dụng khá giống nhau và ký hiệu của các bạn vẽ trên UML cũng đã thống nhất nên việc nắm bắt một công cụ khi chuyển sang làm việc với một công cụ khác không quá khó khăn.

Trong bài này, xin giới thiệu với các bạn công cụ Start UML, một phần mềm nguồn mở, miễn phí, có đầy đủ chức năng và có thể sử dụng tốt trên môi trường Windows.

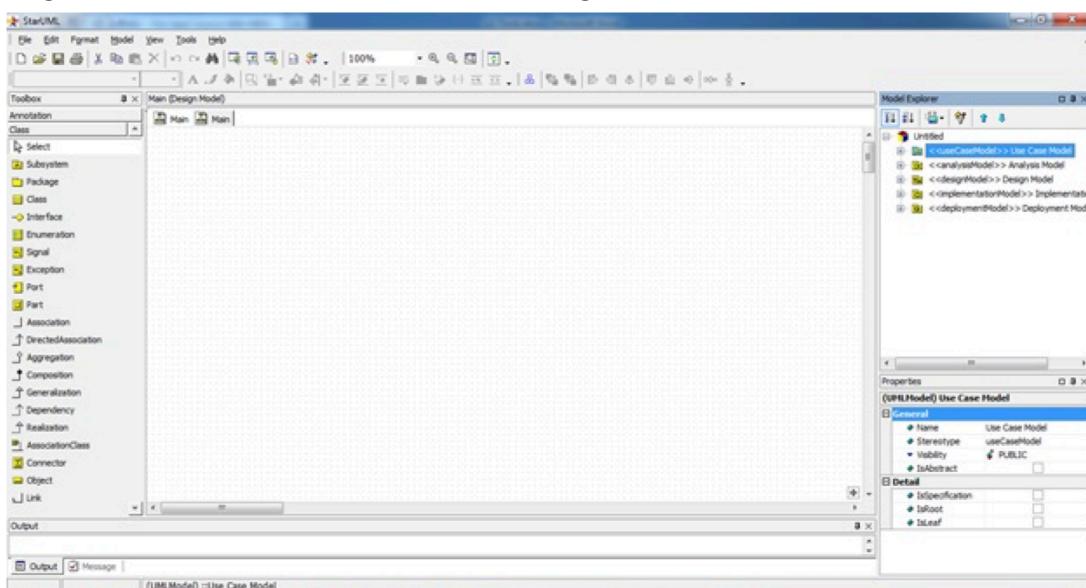
II. Giới thiệu về Start UML

1. Cài đặt

Bạn có thể download bộ cài đặt của phần mềm Start UML tại <http://staruml.sourceforge.net/en/>. Sau khi download và tiến hành các bước cài đặt chúng ta nhanh chóng có được công cụ này trên máy tính.

2. Các Model

Khởi động Start UML vào màn hình chính chúng ta có được các model như sau:



Hình 47. Cửa sổ giao diện của Start UML

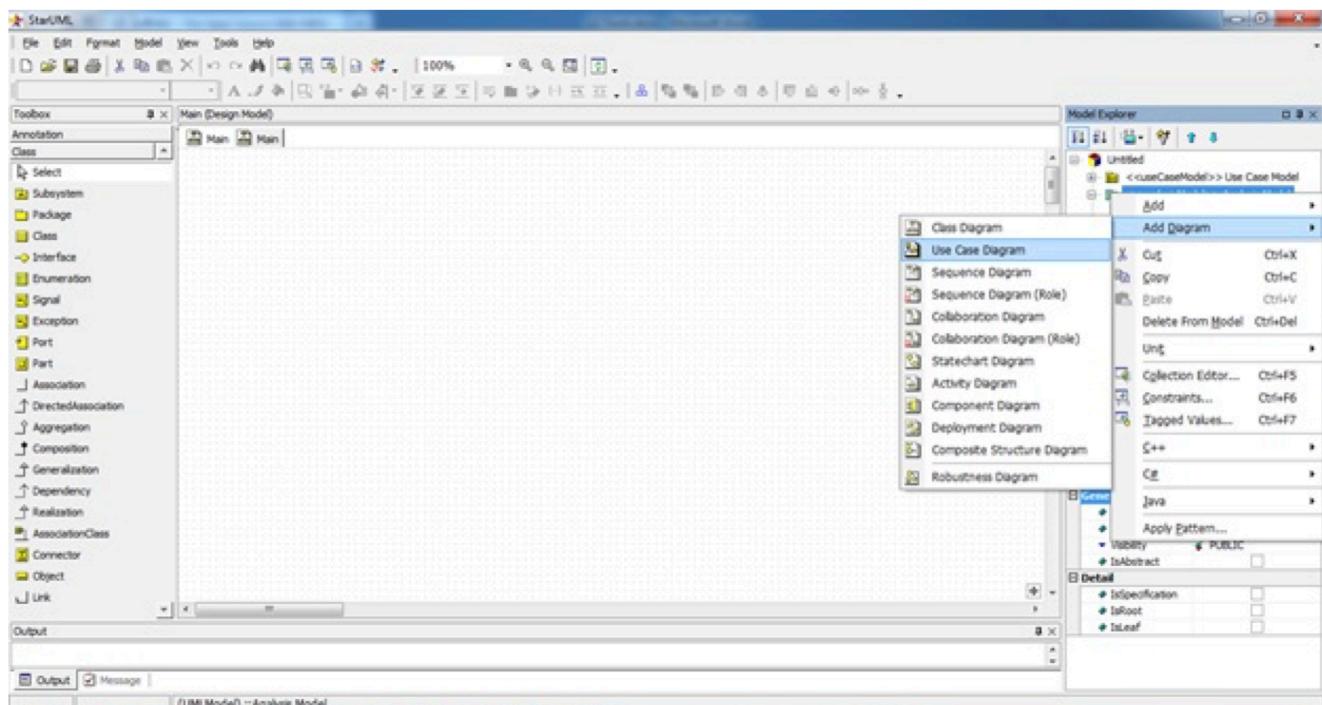
Nhìn cửa sổ Model Explorer bên phải chúng ta nhận thấy có 5 model.

- Use Case Model: chứa các biểu đồ phân tích Use Case
- Analysis Model: chứa các biểu đồ phân tích
- Design Model: chứa các biểu đồ thiết kế
- Implementation Model: chứa các biểu đồ cài đặt
- Deployment Model: chứa các biểu đồ triển khai

Tùy theo nhu cầu phân tích, thiết kế chúng ta xác định sẽ sử dụng model nào để thể hiện.

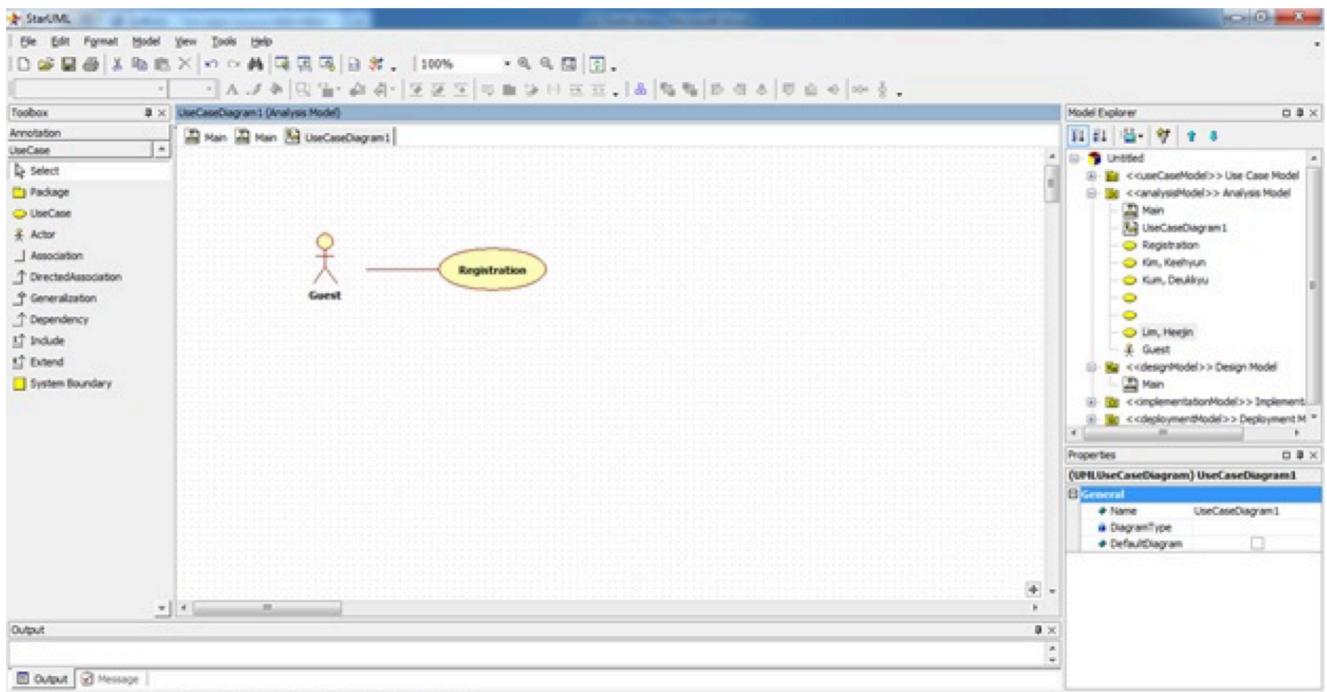
3. Cách tạo các Diagram

Để tạo các biểu đồ, chúng ta chỉ cần chọn model mà bạn muốn sử dụng, kích phải chuột, chọn add diagram và chọn biểu đồ cần xây dựng.



Hình 48. Cách tạo ra một biểu đồ

Sau khi chọn biểu đồ, cửa sổ bên trái sẽ hiển thị thanh công cụ chứa các ký hiệu tương ứng của biểu đồ để bạn có thể vẽ được các biểu đồ một cách dễ dàng.



Hình 49. Vẽ biểu đồ Use case

Việc xây dựng các biểu đồ chúng ta đã bàn kỹ trong các bài trước, bạn xem lại các bài trước và biếu diễi lại các biếu đồ này lên Start UML. Các bạn có thể đọc thêm tài liệu hướng dẫn sử dụng công cụ này trong phần help của phần mềm hoặc xem chi tiết tại <http://staruml.sourceforge.net/docs/user-guide%28en%29/toc.html>.

III. Kết luận

Như vậy, chúng ta đã nghiên cứu qua tất cả các biếu đồ UML được sử dụng phổ biến trong OOAD. Đến đây, bạn đã có đủ kiến thức và kỹ năng để phân tích và thiết kế một phần mềm. Bây giờ bạn hãy cố gắng thực hành phân tích và thiết kế các hệ thống phần mềm để có thêm kinh nghiệm.

Các kiến thức này các bạn có thể dùng để phân tích và thiết kế một phần mềm mới hoặc dùng để mô tả nghiên cứu một phần mềm hoặc framework có sẵn nhằm phục vụ cho việc hiệu chỉnh phần mềm cho phù hợp với nhu cầu của khách hàng.

TÀI LIỆU THAM KHẢO

- [1] Alan Dennis, *Systems Analysis and Design*, 5th Edition, Published by Wiley, 2012
- [2] Langer Arthur, *Analysis and Design of Information Systems*, e-Book, ISBN 978-1-84628-655-1, 2008
- [3] Alan Dennis (Indiana University), Barbara Haley Wixom (University of Virginia), David Tegarden (Virginia Tech), *Systems Analysis and Design with UML Version 2.0*, Second Edition, John Wiley & Sons, Inc., 2005

MỤC LỤC

BÀI 1. QUY TRÌNH PHÁT TRIỂN PHẦN MỀM	3
I. Công nghệ phần mềm (SE – Software Engineering)	3
II. Qui trình phần mềm	4
1. Giới thiệu.....	4
2. Các giai đoạn của qui trình phần mềm	4
III. Các mô hình qui trình phần mềm	11
1. Mô hình thác nước (WaterFall Model)	11
2. Mô hình chữ V (V-Shaped Model)	13
3. Mô hình bản mẫu (Prototyping Model).....	14
4. Mô hình RAD (Rapid Application Development Model).....	16
5. Mô hình qui trình phần mềm tiến hóa	17
6. Mô hình lặp (Interation Model).....	18
7. Mô hình xoắn ốc (Spiral Model)	19
8. Kỹ thuật thế hệ thứ tư (4GT - 4 th Generation Technology).....	20
IV. Kết luận.....	22
BÀI 2. PHÂN TÍCH VÀ THIẾT KẾ HƯỚNG ĐỐI TƯỢNG VÀ UML	24
I. Khái niệm.....	24
II. Thiết kế với UML.....	24
III. Kết luận.....	27
BÀI 3. BIỂU ĐỒ USE CASE.....	28
I. Các thành phần trong biểu đồ Use Case.....	28
1. Actor (tác nhân).....	28
2. Use Case	29
3. Relationship (Quan hệ).....	29
4. System Boundary	30
II. Các bước xây dựng Use Case Diagram	30
III. Đặc tả Use Case	31
1. Cách 1: Viết đặc tả cho các Use Case	32
2. Cách 2: Sử dụng các biểu đồ để đặc tả	32
IV. Sử dụng UseCase Diagram	32
V. Kết luận.....	32
THỰC HÀNH XÂY DỰNG BIỂU ĐỒ USE CASE	32
BÀI 4. BIỂU ĐỒ LỚP (CLASS DIAGRAM).....	37
I. Các thành phần trong biểu đồ Class	37
1. Classes (Các lớp).....	37
2. Relationship (Quan hệ).....	38
II. Cách xây dựng biểu đồ Class	40

III. Sử dụng biểu đồ Class.....	41
IV. Kết luận.....	42
THỰC HÀNH XÂY DỰNG BIỂU ĐỒ LỚP.....	42
BÀI 5. BIỂU ĐỒ HOẠT ĐỘNG (ACTIVITY DIAGRAM)	45
I. Các thành phần cơ bản của Dynamic Model.....	45
II. Activity Diagram	45
III. Cách xây dựng Activity Diagram	47
IV. Sử dụng biểu đồ Activity Diagram	47
V. Kết luận.....	48
THỰC HÀNH XÂY DỰNG ACTIVITY DIAGRAM.....	48
BÀI 6. BIỂU ĐỒ TUẦN TỰ (SEQUENCE DIAGRAM)	51
I. Khái niệm.....	51
II. Các thành phần của Sequence Diagram	51
1. Objects.....	51
2. Stimulus (message)	52
3. Axes.....	52
III. Xây dựng Sequence Diagram.....	52
IV. Ứng dụng Sequence Diagram	52
V. Kết luận.....	52
THỰC HÀNH XÂY DỰNG SEQUENCE DIAGARM	52
BÀI 7. BIỂU ĐỒ THÀNH PHẦN (COMPONENT DIAGRAM).....	56
I. Các thành phần của Component Diagram	56
1. Component	56
2. Component Dependence	56
II. Xây dựng Component Diagram.....	56
THỰC HÀNH XÂY DỰNG COMPONANT DIAGRAM.....	57
III. Ứng dụng của Component Diagram	57
IV. Kết luận.....	57
BÀI 8. BIỂU ĐỒ TRIỂN KHAI (DEPLOYMENT DIAGRAM)	58
I. Khái niệm.....	58
II. Các thành phần của Deployment Diagram	58
1. Node	58
2. Relationship.....	58
III. Xây dựng Deployment Diagram	59

THỰC HÀNH XÂY DỰNG DEPLOYMENT DIAGRAM	59
BÀI 9. CÁC CÔNG CỤ ĐỂ XÂY DỰNG CÁC BIỂU ĐỒ UML.....	61
I. Giới thiệu	61
II. Giới thiệu về Start UML.....	61
1. Cài đặt	61
2. Các Model	61
3. Cách tạo các Diagram	62
III. Kết luận.....	63
TÀI LIỆU THAM KHẢO.....	64
MỤC LỤC	65