TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN

ThS. Trần Hải Thanh
ThS. Nguyễn Lan Oanh
ThS. Nguyễn Thu Phương

BÀI GIẢNG
ỨNG DỤNG CÔNG NGHỆ PHÁT TRIỂN PHẦN MỀM

Tài liệu lưu hành nội bộ

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN

ThS. Trần Hải Thanh
ThS. Nguyễn Lan Oanh
ThS. Nguyễn Thu Phương

BÀI GIẢNG

ỨNG DỤNG CÔNG NGHỆ PHÁT TRIỂN PHẦN MỀM

Thái Nguyên, tháng 4 năm 2024

MỤC LỤC

MỞ ĐẦU

Trong thời đại số hóa, phần mềm không chỉ là những dòng mã lệnh mà còn là những công cụ mạnh mẽ giúp chúng ta giải quyết các vấn đề phức tạp, tối ưu hóa quy trình kinh doanh và nâng cao chất lượng cuộc sống. Việc nắm vững các công nghệ và phương pháp phát triển phần mềm sẽ mở ra cho các bạn nhiều cơ hội nghề nghiệp, từ các vị trí lập trình viên, kỹ sư phần mềm cho đến quản lý dự án và chuyên gia công nghệ.

Môn học này sẽ đưa các bạn đi qua một hành trình khám phá và học hỏi về:

- Các giai đoạn phát triển phần mềm từ phân tích yêu cầu, thiết kế, lập trình, kiểm thử đến triển khai và bảo trì.
- Các phương pháp phát triển phần mềm phổ biến như mô hình thác nước, mô hình xoắn ốc, mô hình phát triển nhanh, phát triển linh hoạt (Agile),…
- Các công nghệ và công cụ hiện đại được sử dụng trong việc phát triển phần mềm.
- Vai trò và sự phối hợp của các bên liên quan trong dự án phát triển phần mềm.

Chúng ta sẽ cùng nhau thảo luận về những thách thức và cơ hội trong quá trình phát triển phần mềm, cũng như cách áp dụng những kiến thức đã học vào thực tiễn. Bên cạnh đó, các bạn sẽ có cơ hội tham gia vào các dự án thực tế, từ đó rèn luyện kỹ năng làm việc nhóm, giải quyết vấn đề và quản lý thời gian.

Hy vọng rằng, qua môn học này, các bạn sẽ không chỉ có được nền tảng kiến thức vững chắc mà còn phát triển niềm đam mê và sự sáng tạo trong lĩnh vực phát triển phần mềm.

Thái Nguyên, tháng 4 năm 2024

Các tác giả

CHƯƠNG 1: TỔNG QUAN QUY TRÌNH PHÁT TRIỂN PHẦN MỀM

Nội dung chính của chương

1.1. Giới thiệu tổng quan về quy trình phát triển phần mềm

1.2. Vòng đời phát triển dự án phần mềm.

1.3 Các loại phần mềm ứng dụng

1.4. Các bước phát triển phần mềm

Mục tiêu đạt được của chương

- Hiểu khái niệm phát triển phần mềm và tầm quan trọng của nó.
- Nắm vững các giai đoạn chính trong quá trình phát triển phần mềm.
- Biết các phương pháp phát triển phần mềm phổ biến.
- Hiểu vai trò của các bên liên quan trong dự án phát triển phần mềm

1.1. Tổng quan về quy trình phát triển phần mềm

Công nghệ thông tin (Information Technology - IT) là việc sử dụng các công nghệ hiện đại để tạo ra, lưu trữ, xử lý, bảo vệ và truyền dẫn thông tin. IT bao gồm cả phần cứng (hardware) và phần mềm (software), và đóng vai trò quan trọng trong mọi lĩnh vực của cuộc sống hiện đại, từ kinh doanh và giáo dục đến y tế và giải trí.

Các thành phần chính của công nghệ thông tin

1. Phần cứng (Hardware)

- Máy tính cá nhân (Personal Computers): Bao gồm máy tính để bàn, máy tính xách tay và thiết bị di động.
- Máy chủ (Servers): Các hệ thống máy tính mạnh mẽ quản lý và lưu trữ dữ liệu, cung cấp dịch vụ cho các máy tính khác trong mạng.
- Thiết bị mạng (Networking Devices): Router, switch, modem và các thiết bị khác giúp kết nối và quản lý mạng.
- Thiết bị lưu trữ (Storage Devices): Ổ cứng, ổ đĩa SSD, thiết bị lưu trữ mạng (NAS) và các giải pháp lưu trữ khác.

2. Phần mềm (Software)

- Hệ điều hành (Operating Systems): Windows, macOS, Linux, và các hệ điều hành khác quản lý tài nguyên phần cứng và cung cấp giao diện cho người dùng.

- Ứng dụng (Applications): Phần mềm văn phòng, trình duyệt web, ứng dụng quản lý doanh nghiệp, phần mềm thiết kế đồ họa, v.v.
- Phần mềm trung gian (Middleware): Các giải pháp giúp kết nối và tích hợp các ứng dụng và hệ thống khác nhau.

3. Mạng (Networking)

- Mạng cục bộ (LAN): Mạng nội bộ kết nối các máy tính và thiết bị trong một khu vực nhỏ.
- Mạng diện rộng (WAN): Mạng kết nối các khu vực địa lý lớn hơn, như mạng internet toàn cầu.
- Mạng không dây (Wireless Networks): Wi-Fi, Bluetooth và các công nghệ mạng không dây khác.

4. An ninh mạng (Cybersecurity)

- Bảo vệ dữ liệu (Data Protection): Các biện pháp bảo vệ dữ liệu khỏi mất mát, đánh cắp và truy cập trái phép.
- Quản lý rủi ro (Risk Management): Xác định, đánh giá và quản lý các rủi ro liên quan đến CNTT.
- Phát hiện và ứng phó sự cố (Incident Detection and Response): Phát hiện các mối đe dọa và xử lý các sự cố an ninh.

5. Điện toán đám mây (Cloud Computing)

- Dịch vụ đám mây (Cloud Services): Các dịch vụ như lưu trữ, xử lý và ứng dụng được cung cấp qua internet.
- Hạ tầng đám mây (Cloud Infrastructure): Các nền tảng cơ sở hạ tầng cho phép triển khai và quản lý dịch vụ đám mây.

Có nhiều cách phân loại các chuyên ngành học Công nghệ thông tin nhưng theo chuẩn ACM của Mỹ thì bậc học đại học Công nghệ thông tin được chia làm 5 chuyên ngành là:

1. Khoa học máy tính (Computer Science):

Khoa học máy tính tập trung vào nền tảng lý thuyết của thông tin và tính toán, cùng với các kỹ thuật thực tế để triển khai và áp dụng chúng. Các lĩnh vực chính bao gồm:

- Thuật toán và cấu trúc dữ liệu: Nghiên cứu các phương pháp tổ chức và xử lý dữ liệu một cách hiệu quả.
- Ngôn ngữ lập trình: Thiết kế, triển khai và phân tích các ngôn ngữ lập trình.
- Kiến trúc máy tính: Cấu trúc và hành vi của các hệ thống máy tính, bao gồm bộ xử lý, bộ nhớ và thiết bị I/O.
- Trí tuệ nhân tạo và học máy: Phát triển các thuật toán cho phép máy tính học hỏi và ra quyết định dựa trên dữ liệu.
- Hệ điều hành: Thiết kế và triển khai phần mềm quản lý tài nguyên phần cứng và cung cấp dịch vụ cho các chương trình máy tính.
- Đồ họa máy tính: Các kỹ thuật tạo, thao tác và hiển thị hình ảnh trực quan thông qua máy tính.

2. Mạng máy tính (Computer Network)

Mạng máy tính nghiên cứu cách thức các máy tính giao tiếp với nhau qua nhiều loại mạng khác nhau. Nó bao gồm:

- Kiến trúc và thiết kế mạng: Lập kế hoạch và cấu trúc các mạng, bao gồm LANs, WANs và Internet.
- Giao thức mạng: Thiết lập các quy tắc và quy ước cho việc trao đổi dữ liệu giữa các thiết bị mạng.
- An ninh mạng: Bảo vệ dữ liệu và tài nguyên khỏi truy cập trái phép và các mối đe dọa mạng.
- Mạng không dây: Nghiên cứu các công nghệ truyền thông không dây như Wi-Fi, Bluetooth và mạng di động.
- Internet of Things (IoT): Kết nối các thiết bị hàng ngày với internet để thu thập và trao đổi dữ liệu.
- Quản lý Mạng: Giám sát và duy trì hiệu suất và độ tin cậy của mạng.

3. Kĩ thuật phần mềm (Software Engineering)

Kĩ thuật phần mềm áp dụng các nguyên tắc kỹ thuật vào việc tạo phần mềm để đảm bảo độ tin cậy và hiệu quả. Nó bao gồm:

- Chu kỳ phát triển phần mềm (SDLC): Quá trình lập kế hoạch, tạo, kiểm thử và triển khai phần mềm.
- Phương pháp Agile: Các phương pháp tiếp cận linh hoạt và lặp đi lặp lại trong phát triển phần mềm để đáp ứng thay đổi.

- Đảm bảo chất lượng và kiểm thử: Đảm bảo phần mềm đáp ứng các tiêu chuẩn chất lượng và không có lỗi.
- Quản lý dự án: Điều phối và giám sát các dự án phần mềm để đạt được thời hạn và ngân sách.
- Phân tích yêu cầu: Thu thập và phân tích nhu cầu của người dùng để phát triển yêu cầu phần mềm rõ ràng và cụ thể.
- Thiết kế trải nghiệm người dùng (UX): Tạo phần mềm trực quan, hiệu quả và thú vị cho người dùng.

4. Hệ thống thông tin (Information System)

Hệ thống thông tin tập trung vào việc sử dụng công nghệ để quản lý và xử lý thông tin trong các tổ chức. Các lĩnh vực chính bao gồm:

- Phân tích và thiết kế hệ thống: Nghiên cứu hệ thống hiện tại của tổ chức và thiết kế các giải pháp cải tiến.
- Quản lý cơ sở dữ liệu: Tạo và quản lý cơ sở dữ liệu để lưu trữ và truy xuất dữ liệu hiệu quả.
- Trí tuệ kinh doanh: Phân tích dữ liệu để hỗ trợ quá trình ra quyết định chiến lược.
- Hệ thống doanh nghiệp: Tích hợp các quy trình và chức năng kinh doanh của tổ chức vào một hệ thống thống nhất.
- An ninh thông tin: Bảo vệ hệ thống thông tin khỏi các mối đe dọa mạng và đảm bảo tính bảo mật và toàn vẹn dữ liệu.
- Thương mại điện tử: Sử dụng hệ thống thông tin để thực hiện các giao dịch kinh doanh trực tuyến.

5. Ứng dụng CNTT (Information Technology)

Ứng dụng CNTT là về việc áp dụng thực tế của công nghệ trong các doanh nghiệp và tổ chức. Nó bao gồm:

- Hỗ trợ và dịch vụ CNTT: Cung cấp hỗ trợ kỹ thuật và duy trì hệ thống CNTT.
- Tích hợp hệ thống: Đảm bảo các hệ thống CNTT khác nhau hoạt động cùng nhau một cách liền mạch.
- An ninh mạng: Bảo vệ hệ thống, mạng và dữ liệu khỏi các cuộc tấn công mạng.
- Quản lý hạ tầng CNTT: Quản lý và tối ưu hóa tài nguyên phần cứng, phần mềm và mạng.

- Điện toán đám mây: Cung cấp các dịch vụ tính toán qua internet, bao gồm lưu trữ, xử lý và ứng dụng phần mềm.
- Chuyển đổi số: Sử dụng công nghệ số để cải tiến quy trình kinh doanh và tạo ra các mô hình kinh doanh mới.

1.2 Vòng đời phát triển dự án phần mềm

Các giai đoạn phát triển phần mềm hay vòng đời phát triển phần mềm (Software Development Life Cycle - SDLC) là một quá trình có cấu trúc bao gồm các giai đoạn rõ ràng và có trình tự để phát triển và duy trì các hệ thống phần mềm. SDLC giúp các nhóm phát triển tổ chức công việc của họ, đảm bảo rằng dự án được hoàn thành đúng hạn, trong phạm vi ngân sách, và đáp ứng các yêu cầu của khách hàng. Dưới đây là các giai đoạn chính trong vòng đời phát triển dự án phần mềm:

1. Thu thập và phân tích yêu cầu (Requirement Analysis)

- Mục tiêu: Xác định và hiểu rõ các yêu cầu chức năng và phi chức năng của phần mềm từ khách hàng hoặc người dùng cuối.
- Hoạt động chính:
  - Thu thập thông tin từ các bên liên quan.
  - Phân tích yêu cầu và ghi lại trong tài liệu yêu cầu.
  - Xác định phạm vi của dự án.
- Kết quả: Tài liệu yêu cầu phần mềm (Software Requirement Specification - SRS).

2. Thiết kế hệ thống (System Design)

- Mục tiêu: Tạo ra bản thiết kế chi tiết cho phần mềm dựa trên các yêu cầu đã xác định.
- Hoạt động chính:
  - Thiết kế kiến trúc tổng thể của hệ thống.
  - Thiết kế chi tiết các module, giao diện và cơ sở dữ liệu.
  - Xác định các công nghệ và công cụ sẽ được sử dụng.
- Kết quả: Tài liệu thiết kế hệ thống (System Design Document - SDD).

3. Lập trình (Implementation)

- Mục tiêu: Viết mã nguồn dựa trên tài liệu thiết kế.
- Hoạt động chính:
  - Phát triển các module phần mềm theo thiết kế.
  - Kiểm tra đơn vị (unit testing) để đảm bảo mỗi phần của mã hoạt động đúng.

    - Tích hợp các module lại với nhau.
- Kết quả: Mã nguồn phần mềm đã hoàn thành.

4. Kiểm thử (Testing)
- Mục tiêu: Đảm bảo rằng phần mềm hoạt động đúng như mong đợi và không có lỗi.
- Hoạt động chính:
    - Kiểm thử tích hợp (integration testing) để đảm bảo các module hoạt động cùng nhau.
    - Kiểm thử hệ thống (system testing) để kiểm tra toàn bộ hệ thống.
    - Kiểm thử chấp nhận (acceptance testing) để xác nhận rằng phần mềm đáp ứng các yêu cầu của người dùng.
- Kết quả: Báo cáo kiểm thử và phần mềm không có lỗi nghiêm trọng.

5. Triển khai (Deployment)
- Mục tiêu: Đưa phần mềm vào môi trường thực tế và cho phép người dùng cuối sử dụng.
- Hoạt động chính:
    - Cài đặt và cấu hình phần mềm trong môi trường sản xuất.
    - Đào tạo người dùng cuối và cung cấp tài liệu hướng dẫn sử dụng.
    - Thực hiện các hoạt động chuyển giao và hỗ trợ sau khi triển khai.
- Kết quả: Phần mềm hoạt động trong môi trường thực tế và được người dùng sử dụng.

6. Bảo trì (Maintenance)
- Mục tiêu: Đảm bảo phần mềm hoạt động liên tục và hiệu quả trong suốt vòng đời của nó.
- Hoạt động chính:
    - Sửa lỗi và vá bảo mật.
    - Cập nhật và cải thiện tính năng theo yêu cầu mới.
    - Giám sát và tối ưu hóa hiệu suất của phần mềm.
- Kết quả: Phần mềm được duy trì và cải tiến liên tục để đáp ứng yêu cầu thay đổi của người dùng.

1.3 Các loại phần mềm ứng dụng

Phần mềm máy tính (*Computer Software*) là một tập hợp những câu lệnh hoặc chỉ thị (*Instruction*) được viết bằng một hoặc nhiều ngôn ngữ lập trình theo một trật tự xác định, và các *dữ liệu* liên quan nhằm tự động thực hiện một số chức năng hoặc giải quyết một vấn đề cụ thể nào đó.

1) Phân loại theo chức năng thực hiện

Tuỳ theo các chức năng mà phần mềm cung cấp, chúng ta có thể phân chia phần mềm thành các loại như sau:

❖ **Phần mềm hệ thống**: Thường cung cấp các chức năng:

- Điều hành hoạt động máy tính, thiết bị và chương trình (OS, ...)
- Cung cấp các tiện ích (tổ chức tệp tin, nén, dọn ổ đĩa, ...).

❖ **Phần mềm nghiệp vụ**: Thường cung cấp các chức năng:

- Hỗ trợ các hoạt động nghiệp vụ, ví dụ: quản lý thư viện, quản lý điểm, quản lý nhân sự, …
- Có số lượng lớn, đa dạng và thường được chia làm 2 loại theo cách thức phát triển:
  - ***Sản phẩm đặt hàng***: sản xuất theo đơn đặt hàng; đáp ứng các yêu cầu đặc thù riêng của khách hàng.
  - ***Sản phẩm chung*** (sản phẩm đại trà, thương mại): được phát triển để bán rộng rãi trên thị trường; cần xây dựng đảm bảo đáp ứng các yêu cầu chung của thị trường người dùng.

Sự phân loại này ngày càng trở nên mờ nhạt, vì nhiều công ty phần mềm hiện này thường bắt đầu từ các sản phẩm chung (dòng sản phẩm) và tùy biến chúng theo các yêu cầu của khách hành riêng lẻ để trở thành sản phẩm theo đơn đặt hàng.

❖ **Phần mềm công cụ (CASE tools)**: Thường cung cấp các tính năng hỗ trợ cho các hoạt động trong SE. Ví dụ: các phần mềm ngôn ngữ lập trình; trình biên dịch; gỡ rối; hỗ trợ phân tích, thiết kế; quản lý dự án; kiểm thử; và các môi trường tích hợp phát triển ứng dụng (IDE) như Netbean, Eclipse, Visual Studio, …

2) *Phân loại theo lĩnh vực ứng dụng*

Phân loại theo lĩnh vực nghiệp vụ mà phần mềm hỗ trợ, chúng ta có thể chia phần mềm thành các loại như sau:

- **Phần mềm hệ thống:** điều khiển các thiết bị phần cứng; làm nền tảng/môi trường để vận hành các phần mềm khác. Ví dụ: các hệ điều hành, các tiện ích quản lý tệp tin, dọn ổ đĩa, …
- **Phần mềm thời gian thực:** thu thập, xử lý các dữ liệu thời gian thực. Ví dụ: phần mềm truyền phát tín hiệu vệ tinh, đo nhiệt độ, chất lượng không khí, …
- **Phần mềm nghiệp vụ:** xử lý các thông tin nghiệp vụ, hỗ trợ kinh doanh, thương mại, …
- **Phần mềm khoa học kỹ thuật:** Mô phỏng các hiện tượng vật lý, hoá học, phản ứng hạt nhân, … .Thường dùng trong các ngành khoa học, phục vụ nghiên cứu, thí nghiệm.
- **Phần mềm nhúng:** còn được xem là một hệ lập trình chuyên biệt được lắp đặt trên các thiết bị phần cứng cố định để điều khiển, xử lý các tác vụ cụ thể trên thiết bị. Ví dụ: hệ thống giao thông thông minh, giải pháp cổng IoT, máy móc, thiết bị thông minh như máy ảnh kỹ thuật số, lò vi ba, máy photocopy, máy in Laser, máy FAX, máy giặt, các bảng quảng cáo sử dụng hệ thống đèn LED….

Ngoài ba tiêu chí phân loại trên đây, phần mềm còn có thể được phân loại dựa trên các chính sách phát triển của sản phẩm như: *phần mềm tự do nguồn mở; phần mềm đóng* (được đóng gói, có bản quyền chống vi phạm, …).

Ngoài ba tiêu chí phân loại trên đây, phần mềm còn có thể được phân loại dựa trên các chính sách phát triển của sản phẩm như: *phần mềm tự do nguồn mở; phần mềm đóng* (được đóng gói, có bản quyền chống vi phạm, …).

Sự phát triển của công nghệ và nhu cầu của thị trường đã thúc đẩy nhiều xu hướng mới trong lĩnh vực phát triển phần mềm. Dưới đây là một số xu hướng nổi bật:

1. Ứng dụng di động.

Phát triển phần mềm ứng dụng di động là quá trình tạo ra các ứng dụng phần mềm hoạt động trên các thiết bị di động như điện thoại thông minh và máy tính bảng. Với sự phát triển mạnh mẽ của công nghệ di động và sự phổ biến của các thiết bị di động trên toàn cầu, lĩnh vực này đang trở thành một trong những xu hướng phát triển nhanh nhất trong ngành công nghiệp phần mềm. Điều này được thể hiện thông qua một số thống kê dưới đây:

Theo báo cáo của Statista, số lượng người dùng điện thoại thông minh trên toàn cầu đã đạt khoảng 6.6 tỷ vào năm 2023, và dự kiến sẽ tiếp tục tăng trong các năm tới.

Thị trường ứng dụng di động: Dự kiến đến năm 2024, thị trường ứng dụng di động toàn cầu sẽ đạt giá trị khoảng 935 tỷ USD, tăng trưởng từ 461 tỷ USD vào năm 2019

Số lượng ứng dụng trên App Store và Google Play: Tính đến năm 2023, có khoảng 2.87 triệu ứng dụng trên Google Play và 1.96 triệu ứng dụng trên App Store.

Thời gian sử dụng ứng dụng di động: Người dùng trung bình dành khoảng 3.7 giờ mỗi ngày để sử dụng các ứng dụng di động vào năm 2023, và con số này dự kiến sẽ tiếp tục tăng.

Doanh thu từ ứng dụng di động: Doanh thu từ các ứng dụng di động dự kiến sẽ đạt 935 tỷ USD vào năm 2024, với phần lớn đến từ các ứng dụng trò chơi, mua sắm trực tuyến, và dịch vụ đăng ký

Các nền tảng phát triển chính: iOS (Swift và Objective-C …), Android (Java, Kotlin…) , Cross-Platform Development (React Native, Flutter, Xamarin, Ionic…)

2. Big Data

Data, hay dữ liệu lớn, đề cập đến một lượng dữ liệu rất lớn và phức tạp mà các công cụ xử lý dữ liệu truyền thống không thể xử lý hiệu quả. Dữ liệu này có thể đến từ nhiều nguồn khác nhau và có các dạng khác nhau, bao gồm dữ liệu cấu trúc, phi cấu trúc và bán cấu trúc

Ứng dụng của Big Data

- Kinh doanh và Marketing: phân tích khách hàng, chiến lược tiếp thị
- Y tế: Phân tích dữ liệu y tế, dự đoán dịch bệnh bằng cách sử dụng dữ liệu lớn để dự đoán và kiểm soát sự bùng phát của các dịch bệnh
- Tài chính: Đánh giá rủi ro tài chính và phát hiện gian lận bằng cách phân tích dữ liệu giao dịch, Tối ưu hóa chiến lược đầu tư dựa trên phân tích dữ liệu thị trường và xu hướng kinh tế.
- Giáo dục: phân tích học tập bằng cách hiểu rõ hơn về hành vi học tập của sinh viên để cải thiện phương pháp giảng dạy, tối ưu quản lý các chương trình giáo dục và tài nguyên học tập
- Sản xuất và vận tải: Quản lý chuỗi cung ứng giúp cải thiện hiệu quả và giảm chi phí bằng các phân tích dữ liệu về chuỗi cung ứng và sản xuất; Dự đoán và ngăn ngừa sự cố máy móc bằng cách phân tích dữ liệu cảm biến và hoạt động.

Công nghệ và công cụ Big data:

- Hadoop: Một nền tảng mã nguồn mở cho phép xử lý và lưu trữ dữ liệu lớn trên một cụm máy chủ.
- Apache Spark: Một công cụ xử lý dữ liệu lớn có khả năng xử lý dữ liệu nhanh chóng và hiệu quả.
- NoSQL Databases: Các cơ sở dữ liệu không quan hệ như MongoDB, Cassandra, và HBase được thiết kế để xử lý dữ liệu lớn và phi cấu trúc.
- Data Visualization Tools: Các công cụ như Tableau, Power BI và QlikView giúp trực quan hóa dữ liệu lớn để dễ dàng hiểu và phân tích

3. Cloud computing (Điện toán đám mây)

Điện toán đám mây là mô hình cung cấp các dịch vụ công nghệ thông tin như máy chủ, lưu trữ, cơ sở dữ liệu, mạng lưới, phần mềm và các dịch vụ khác qua internet ("đám mây"). Thay vì sở hữu và duy trì các trung tâm dữ liệu và máy chủ vật lý, cloud computing cho phép tổ chức truy cập vào các tài nguyên theo nhu cầu từ một nhà cung cấp dịch vụ đám mây như Amazon Web Services (AWS), Microsoft Azure hoặc Google Cloud Platform (GCP)

4. Phát triển DevOps (Development and Operations)

DevOps là một phương pháp và nền tảng triển khai phần mềm nhằm cải thiện sự hợp tác giữa các nhóm phát triển phần mềm và quản lý hệ thống hoạt động (operations). Đây là một phương pháp tiếp cận thúc đẩy tích hợp liên tục, triển khai liên tục và cải tiến liên tục để đảm bảo các sản phẩm phần mềm được phát triển và triển khai nhanh chóng và hiệu quả

5. AI và Machine Learning (Trí tuệ nhân tạo và Học máy)*:*

Áp dụng AI và machine learning vào phát triển phần mềm để tạo ra các ứng dụng thông minh, nhận diện hình ảnh, xử lý ngôn ngữ tự nhiên, hệ thống gợi ý và dự đoán.

6. IoT (Internet of Things - Internet vạn vật): Phát triển ứng dụng cho các thiết bị kết nối IoT, từ các thiết bị gia đình thông minh đến các hệ thống công nghiệp và y tế

7. Blockchain và Cryptocurrency: Xây dựng các ứng dụng dựa trên blockchain như các hệ thống thanh toán, quản lý chuỗi cung ứng, và các ứng dụng phi tập trung khác

8. Cybersecurity (Bảo mật thông tin): Phát triển các giải pháp bảo mật cho ứng dụng, bao gồm bảo vệ dữ liệu, phát hiện xâm nhập, và giám sát an ninh mạng.

9. AR/VR (Augmented Reality/Virtual Reality - Thực tế mở rộng/Thực tế ảo)*:*

Xây dựng các ứng dụng thực tế mở rộng và thực tế ảo cho giáo dục, thương mại điện tử, y tế và giải trí

10. Edge Computing: Phát triển ứng dụng tính toán tại điểm xử lý (edge) để giảm độ trễ và tăng tốc độ xử lý dữ liệu trong mạng lưới IoT và các hệ thống phân tán.

11. Quantum Computing: Nghiên cứu và phát triển ứng dụng dựa trên tính toán lượng tử để giải quyết các vấn đề tính toán phức tạp mà các máy tính hiện đại không thể xử lý.

1.4. Các bước phát triển phần mềm

Để tự động hóa hoạt động xử lý, hệ thống phải trải qua một quá trình gồm nhiều bước được gọi là quá trình phát triển hệ thống. Cũng giống như nhiều tiến trình khác, phát triển hệ thống tự động cũng theo chu trình được gọi là vòng đời (Life cycle). Khái niệm vòng đời là một khái niệm rộng nó bắt đầu từ sự khởi đầu xây dựng cho đến kết thúc việc khai thác hệ thống. Nếu chúng ta chỉ chú trọng đến giai đoạn xây dựng và triển khai thì gọi là phát triển hệ thống. Vòng đời phát triển hệ thống - SDLC (Systems Development Life Cycle) là một phương pháp luận chung để phát triển nhiều loại hình hệ thống khác nhau. Tuy nhiên, các giai đoạn trong quá trình này cũng thay đổi khác nhau khoảng từ 3 cho đến 20 tùy theo qui mô và loại hình hệ thống chúng ta đang tiếp cận.

Phần sau đây sẽ giới thiệu các giai đoạn cơ bản làm nền tảng chung cho hầu hết quá trình phát triển hệ thống:

Giai đoạn khởi tạo

Hoạt động chính của giai đoạn này là khảo sát tổng quan hệ thống, vạch ra các vấn đề tồn tại trong hệ thống và các cơ hội của hệ thống, cũng như trình bày lý do tại sao hệ thống nên hoặc không nên được đầu tư phát triển tự động hóa. Một công việc quan trọng tại thời điểm này là xác định phạm vi của hệ thống đề xuất, trưởng dự án và nhóm phân tích viên ban đầu cũng lập một kế hoạch các hoạt động của nhóm trong các giai đoạn tiếp theo của dự án phát triển hệ thống. Kế hoạch này xác định thời gian và nguồn lực cần thiết. Đánh giá khả thi của dự án và nhất là phải xác định được chi phí cần phải đầu tư và lợi ít mang lại từ hệ thống. Kết quả của giai đoạn này là xác định được dự án hoặc được chấp nhận để phát triển, hoặc bị từ chối, hoặc phải định hướng lại.

Giai đoạn phân tích

Giai đoạn phân tích bao gồm các bước sau:

- Thu thập yêu cầu hệ thống: các phân tích viên làm việc với người sử dụng đề xác định tất cả những gì mà người dùng mong muốn từ hệ thống đề xuất.

- Nguyên cứu các yêu cầu và cấu trúc hoá (mô hình hoá) để dễ dàng nhận biết và loại bỏ những yếu tố dư thừa.
- Phát sinh các phương án thiết kế chọn lựa phù hợp với yêu cầu và so sánh các phương án này để xác định giải pháp nào là đáp ứng tốt nhất các yêu cầu trong một mức độ cho phép về chi phí, nhân lực, và kỹ thuật của tổ chức. Kết quả của giai đoạn này là bản mô tả về phương án được chọn.

Trong phân tích hướng đối tượng giai đoạn này quan tâm đến mức độ trừu tượng hoá đầu tiên bằng cách xác định các lớp và các đối tượng đóng vai trò quan trọng nhằm diễn đạt các yêu cầu cũng như mục tiêu hệ thống. Để hiểu rõ các yêu cầu hệ thống chúng ta cần xác định ai là người dùng và là tác nhân hệ thống. Trong phương pháp phát triển hướng đối tượng cũng như phương pháp truyền thống, các mô tả kịch bản hoạt động được sử dụng để trợ giúp các phân tích viên hiểu được yêu cầu. Tuy nhiên, các kích bản này có thể được mô tả không đầy đủ hoặc không theo một hình thức. Do đó, khái niệm use case được dùng trong giai đoạn này nhằm biểu diễn chức năng hệ thống và sự tương tác người dùng hệ thống. Các kích bản hoạt động lúc này sử dụng các mô hình động (dynamic diagram) nhằm mô tả nội dung của use case để làm rõ sự tương tác giữa các đối tượng, vai trò cũng như sự cộng tác của các đối tượng trong hoạt động của use case hệ thống. Trong giai đoạn phân tích, chỉ có các lớp tồn tại trong phạm vi hệ thống (ở thế giới thực) mới được mô hình hoá và như vậy thì kết quả mô hình hoá trong giai đoạn này sẽ phản ánh phạm vi của hệ thống, các lớp về kỹ thuật, giao diện định nghĩa phần mềm cũng không quan tâm ở giai đoạn này.

Giai đoạn thiết kế

Trong giai đoạn này kết quả của giai đoạn phân tích sẽ được chi tiết hoá để trở thành một giải pháp kỹ thuật để thực hiện. Các đối tượng và các lớp mới được xác định để bổ sung vào việc cài đặt yêu cầu và tạo ra một hạ tầng cơ sở kỹ thuật về kiến trúc. Ví dụ: các lớp mới này có thể là lớp giao diện (màn hình nhập liệu, màn hình hỏi đáp, màn hình duyệt,). Các lớp thuộc phạm vi vấn đề có từ giai đoạn phân tích sẽ được "nhúng" vào hạ tầng cơ sở kỹ thuật này, tạo ra khả năng thay đổi trong cả hai phương diện: Phạm vi vấn đề và hạ tầng cơ sở. Giai đoạn thiết kế sẽ đưa ra kết quả là bản đặc tả chi tiết cho giai đoạn xây dựng hệ thống. Về mức độ thiết kế thì có thể chia kết quả của giai đoạn này thành hai mức:

Thiết kế luận lý

Đặc tả hệ thống ở mức độ trừu tượng hóa dựa trên kết quả của giải pháp được chọn lựa từ giai đoạn phân tích. Các khái niệm và mô hình được dùng trong giai đoạn này độc lập với phần cứng, phần mềm sẽ sử dụng và sự chọn lựa cài đặt. Theo quan điểm lý thuyết, ở bước này hệ thống có thể cài đặt trên bất kỳ trên nền tảng phần cứng và hệ điều hành nào, điều này cho thấy giai đoạn này chỉ tập trung để biểu diễn khía cạnh hành vi và tính năng của hệ thống.

Thiết kế vật lý

Chuyển đổi kết quả thiết kế luận lý sang các đặc tả trên phần cứng, phần mềm và kỹ thuật đã chọn để cài đặt hệ thống. Cụ thể là đặc tả trên hệ máy tính, hệ quản trị cơ sở dữ liệu, ngôn ngữ lập trình đã chọn,…. Kết quả của bước này là các đặc tả hệ thống vật lý sẵn sàng chuyển cho các lập trình viên hoặc những người xây dựng hệ thống khác để lập trình xây dựng hệ
thống.

Giai đoạn xây dựng

Trong giai đoạn xây dựng (giai đoạn lập trình), các lớp của giai đoạn thiết kế sẽ được biến thành những dòng mã lệnh (code) cụ thể trong một ngôn ngữ lập trình hướng đối tượng cụ thể (không nên dùng một ngôn ngữ lập trình hướng chức năng!). Phụ thuộc vào khả năng của ngôn ngữ được sử dụng, đây có thể là một công việc khó khăn hay dễ dàng. Khi tạo ra các mô hình phân tích và thiết kế trong UML, tốt nhất nên cố gắng né tránh việc ngay lập tức biến đổi các mô hình này thành các dòng mã lệnh. Trong những giai đoạn trước, mô hình được sử dụng để dễ hiểu, dễ giao tiếp và tạo nên cấu trúc của hệ thống; vì vậy, vội vàng đưa ra những kết luận về việc viết mã lệnh có thể sẽ thành một trở ngại cho việc tạo ra các mô hình chính xác và đơn giản. Giai đoạn xây dựng là một giai đoạn riêng biệt, nơi các mô hình được chuyển thành các mã lệnh.

Giai đoạn thử nghiệm

Một hệ thống phần mềm thường được thử nghiệm qua nhiều giai đoạn và với nhiều nhóm thử nghiệm khác nhau. Các nhóm sử dụng nhiều loại biểu đồ UML khác nhau làm nền tảng cho công việc của mình: Thử nghiệm đơn vị sử dụng biểu đồ lớp (class diagram) và đặc tả lớp, thử nghiệm tích hợp thường sử dụng biểu đồ thành phần (component diagram) và biểu đồ cộng tác (collaboration diagram), và giai đoạn thử nghiệm hệ thống sử dụng biểu đồ Use case (use case diagram) để đảm bảo hệ thống có phương thức hoạt động đúng như đã được định nghĩa từ ban đầu trong các biểu đồ này.

Giai đoạn cài đặt và bảo trì

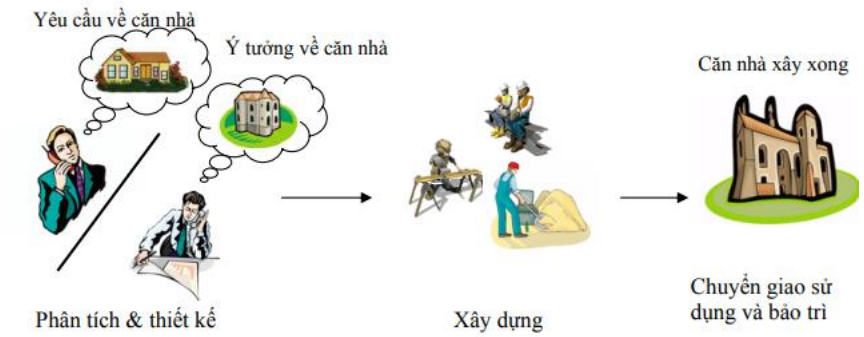
Điều chỉnh hệ thống phù hợp với nhu cầu sử dụng, các thay đổi phát sinh bao gồm: - Chức năng sử dụng chưa phù hợp tốt nhất với người sử dụng hoặc khó sử dụng

- Các điều kiện kinh doanh của doanh nghiệp thay đổi, đòi hỏi phải chỉnh sửa sao cho hệ thống vẫn hữu dụng
- Các lỗi hệ thống phát sinh do quá trình kiểm tra còn xót lại
- Nâng cấp phiên bản mới của hệ thống

Bảo trì hệ thống không nên xem như là một giai đoạn tách rời mà nên xem như là một sự lặp lại chu trình của những giai đoạn trước đòi hỏi phải được nghiên cứu đánh giá và cài đặt. Tuy nhiên, nếu một hệ thống không còn hoạt động như mong muốn do có sự thay đổi quá lớn về hoạt động, hoặc nhu cầu mới đặt ra vượt quá sự giải quyết của hệ thống hiện tại, hoặc chi phí để bảo trì là quá lớn. Lúc này yêu cầu về hệ thống mới được xác lập để thay thế hệ thống hiện tại và một qui trình lại bắt đầu.

Ví dụ về qui trình phát triển

Chúng ta có thể hình dung rằng chúng ta muốn xây dựng một căn nhà. Công việc đầu tiên là chúng ta chắc chắn là chúng ta dự tính xem chúng ta sẽ bỏ số tiền bao nhiêu để xây dựng căn nhà này, dựa trên số tiến này chúng ta tìm kiếm và phác hoạ (có thể chỉ trong ý tưởng) căn nhà này phải như thế nào? Loại căn nhà theo kiểu gì, có mấy phòng, chiều rộng và chiều dài bao nhiêu, rồi nào đến nền nhà, màu sắc, tiện nghi ?,… Rồi sau đó, chúng ta sẽ chọn một đơn vị xây dựng (trong số nhiều đơn vị mà thoả yêu cầu nhất). Tất cả các yêu trên sẽ phải trao đổi với đơn vị xây dựng này nhằm thống nhất về giá cả cũng như các điều khoản về yêu cầu xây dựng. Giai đoạn này được xem như là giai đoạn phân tích. Tiếp đó, đơn vị xây dựng sẽ thực hiện công việc thiết kế chi tiết của căn nhà, và từng đơn vị trong căn nhà (phòng, tường, trần, mái, phòng khách, phòng ăn, phòng ngủ,…). Giai đoạn này được xem là giai đoạn thiết kế. Sau đó các bản thiết kế chi tiết của căn nhà sẽ được bộ phận thi công dựa vào đó để tiến hành việc xây dựng. Giai đoạn này được xem là giai đoạn xây dựng. Căn nhà sau khi hoàn tất sẽ được chuyển giao để sử dụng, tất nhiên trong quá trình sử dụng nếu có các hư hỏng thì đơn vị



Hình 1.1. Ví dụ về quy trình phát triển phần mềm

CHƯƠNG 2: QUẢN LÝ DỰ ÁN

Nội dung chính của chương

2.1 Giới thiệu về quản lý dự án phần mềm

2.2 Kỹ năng xây dựng dự án

2.3 Case study: Lập kế hoạch dự án phần mềm

Mục tiêu cần đạt được của chương

- Hiểu được các khái niệm cơ bản về quản lý dự án phần mềm
- Nắm vững các giai đoạn chính trong quản lý dự án phần mềm
- Biết cách lập kế hoạch quản lý dự án
- Sử dụng các công cụ và kỹ thuật quản lý dự án phần mềm hiệu quả

2.1. Giới thiệu về quản lý dự án phần mềm

Quản lý dự án phần mềm là quá trình lập kế hoạch, tổ chức, điều phối và kiểm soát các hoạt động phát triển phần mềm từ đầu đến cuối. Nhiệm vụ chính của quản lý dự án phần mềm là đảm bảo dự án được triển khai thành công với các yêu cầu về chất lượng, thời gian và ngân sách đã được xác định từ trước. Công việc của quản lý dự án phần mềm bao gồm lên kế hoạch chi tiết, phân bổ tài nguyên, quản lý rủi ro, giám sát tiến độ, và tương tác chặt chẽ với các bên liên quan để đảm bảo các mục tiêu dự án được đạt được một cách hiệu quả. Công việc của quản lý dự án phần mềm bao gồm:

- Lập kế hoạch chi tiết: Xác định các bước cụ thể để hoàn thành dự án, bao gồm lịch trình công việc và phân bổ tài nguyên.
- Phân bổ tài nguyên: Bố trí và quản lý các nguồn lực như con người, vật liệu và thiết bị cần thiết cho dự án.
- Quản lý rủi ro: Đánh giá và quản lý các rủi ro có thể xảy ra trong suốt quá trình dự án để giảm thiểu tác động tiêu cực.
- Giám sát tiến độ: Theo dõi và đánh giá tiến độ của các hoạt động để đảm bảo dự án diễn ra đúng tiến độ đã đề ra.
- Tương tác chặt chẽ với các bên liên quan: Bao gồm các nhà phát triển, khách hàng, và các bên liên quan khác để đảm bảo các mục tiêu dự án được đạt một cách hiệu quả.

2.2. Kỹ năng xây dựng dự án

2.2.1. Phân tích nghiệp vụ và thu thập yêu cầu

Giai đoạn này phân tích viên và khách hàng cùng hợp tác để xác định miền ứng dụng, các dịch vụ mà hệ thống cung cấp, hiệu năng của hệ thống, các ràng buộc về hệ thống cần triển khai, ví dụ: môi trường vận hành hệ thống, về chất lượng sản phẩm, …

Ở đây, xuất hiện một khái niệm mới là Stakeholder. Stakeholder tạm dịch là các bên liên quan, hoặc những cá nhân/tổ chức liên quan đến dự án xây dựng hệ thống, hoặc những tổ chức/cá nhân sẽ cung cấp yêu cầu cho dự án, hoặc bị tác động bởi dự án. Họ có thể là những người sử dụng cuối; khách hàng; đối tác đầu tư; các đối thủ cạnh trang; các tác nhân thị trường; các kỹ sư phần mềm; người quản lý; hoặc các chuyên gia lĩnh vực; ….

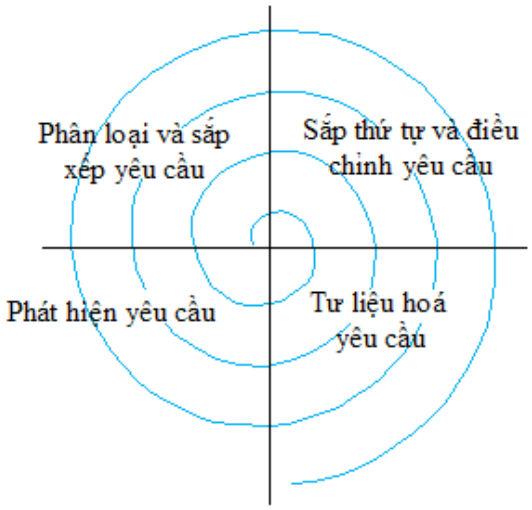
Ví dụ, trong hệ thống ATM gồm các Stakeholder sau: khách hàng của ngân hàng, đại diện của các ngân hàng khác, người quản lý ngân hàng, nhân viên ngân hàng, người quản trị cơ sở dữ liệu của ngân hàng, nhân viên quản lý bảo mật, phòng marketing, kỹ sư bảo trì phần cứng và phần mềm, người điều hành ngân hàng.

Do đó, để thu thập các yêu cầu về hệ thống cần xây dựng, trước hết phân tích viên cần xác định được các stakeholder của dự án. Khách hàng và người dùng cuối của hệ thống là hai loại Stakholder không thể thiếu trong hầu hết các dự án phần mềm. Sau đó sẽ lập kế hoạch để thu thập yêu cầu từ họ.

Trong quá trình thu thập, phát hiện và tìm hiểu yêu cầu của stakeholder, chúng ta thường gặp khó khăn như:

- Stakeholder không biết những gì mà họ thật sự mong muốn.
- Stakeholder mô tả các yêu cầu theo thuật ngữ của họ.
- Những stakeholder khác nhau có thể có các yêu cầu xung đột nhau
- Những yếu tố tổ chức và quyền lực có thể ảnh hưởng tới các yêu cầu hệ thống.
- Yêu cầu được phát biểu từ stakeholder một cách không chắc chắn, có thể thay đổi nhanh chóng trong suốt quá trình thu thập, phân tích yêu cầu.

Mô hình xoắn ốc (Hình 3.4) được khuyến cáo được sử dụng để khắc phục các vấn đề này. Hay nói cách khác, các hoạt động trong giai đoạn này nên tiến hành một cách lặp lại, mỗi lần lặp là một lần chúng ta hoàn thiện/tinh chế các kết quả của lần lặp trước đó và tiến triển bổ sung thêm các yêu cầu mới nếu phát sinh cho lần lặp hiện thời.



Hình 2.1: Quy trình thu thập và phân tích yêu cầu

Trong quy trình xoắn ốc, mỗi vòng lặp xoắn ốc cần tiến hành các công việc:

- ✓ Phát hiện yêu cầu: thu thập yêu cầu từ stakeholder; các yêu cầu miền nên làm rõ và phát hiện ở bước này.
- ✓ Phân loại, sắp xếp yêu cầu: xác định, phân loại, gom nhóm/sẵp xếp các yêu cầu thành các nhóm có liên quan và tổ chức chúng thành các nhóm gắn kết.
- ✓ Gán thứ tự ưu tiên và điều chỉnh các yêu cầu xung đột: càng nhiều stakeholder thì khả năng xung đột giữa các yêu cầu của họ càng cao. Hoạt động này nhằm đánh thứ tự ưu tiên của các yêu cầu, phát hiện và giải quyết xung đột giữa các yêu cầu.
- ✓ Tư liệu hóa yêu cầu: viết tài liệu mô tả các yêu cầu đã xác định, chuyển cho stakeholder liên quan đánh giá, hiệu chỉnh và tiếp tục vòng xoắn ốc tiếp theo nếu còn có những vấn đề phát sinh liên quan đến yêu cầu.

Các kỹ thuật/phương pháp có thể sử dụng để thu thập yêu cầu từ Stakeholder:

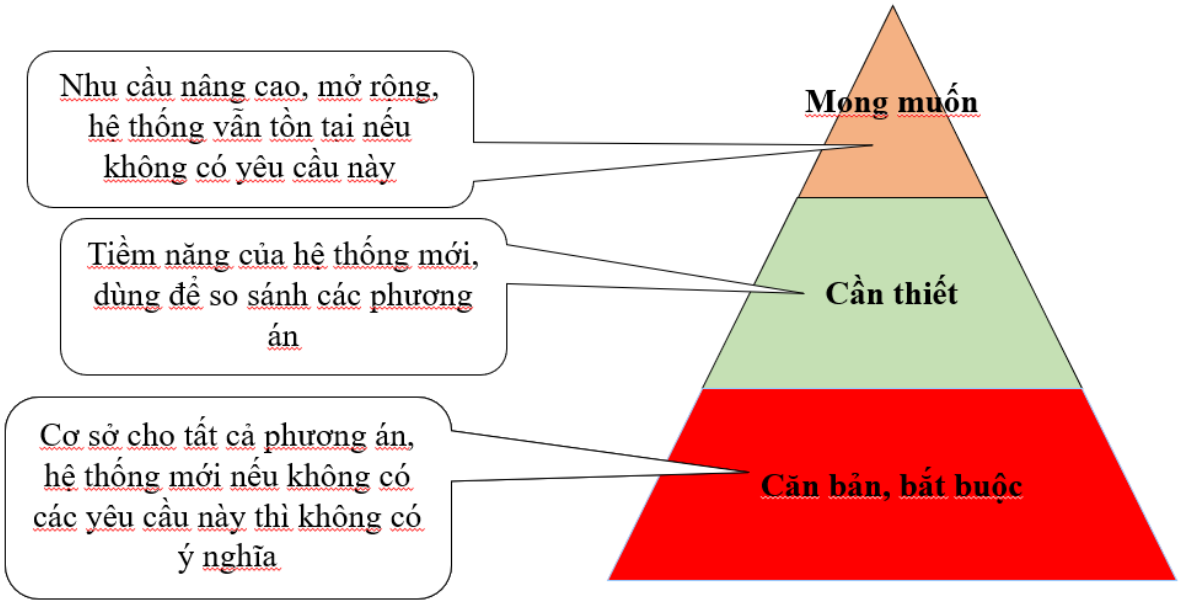
1. *Phỏng vấn - Interviews*
2. *Bảng câu hỏi thăm dò – Questionnaires*
3. *Hội thảo – Workshops*
4. *Thẻ sự kiện - Storyboarding*
5. *Phân vai - Role playing*
6. *Phiên làm việc tập trung - Brainstorming sessions*
7. *Mẫu thử - Prototyping*
8. *Trường hợp sử dụng & Kịch bản sử dụng- Use cases & scenarios.*

9. *Phân tích các tài liệu - Analysis of existing documents*
10. *Quan sát & Minh họa nhiệm vụ - Observation & Task demonstration*
11. *Phân tích các hệ thống đang tồn tại - Analysis of existing systems*
12. *Khung nhìn - Viewpoint*

Mỗi kỹ thuật thu thập yêu cầu đều có các ưu điểm và nhược điểm riêng và phù hợp với ngữ cảnh thu thập yêu cầu nhất định. Ví dụ: để thu thập các yêu cầu về giao diện chương trình, thẻ sự kiện là kỹ thuật tối ưu; để thương lượng, giải quyết các yêu cầu xung đột, phiên làm việc tập trung là lựa chọn tốt nhất; với dự án lớn, phức tạp, có nhiều đối tác cần gặp gỡ và trao đổi trực tiếp, yêu cầu mâu thuẫn, … thì hội thảo có thể là một lựa chọn cần cân nhắc để tiết kiệm thời gian…

Việc lựa chọn kỹ thuật thu thập yêu cầu phù hợp với Stakeholder sẽ mang lại nhiều lợi ích và thuận lợi cho phân tích viên hệ thống.

Sau khi có được các yêu cầu thì sẽ phân nhóm yêu cầu



Hình 2.2. Phân nhóm yêu cầu

2.2.2. Xác định mục tiêu của dự án

Để xác định mục tiêu của một ứng dụng, bạn cần xem xét các yếu tố sau:

1. Giải quyết vấn đề cụ thể: Ứng dụng cần phải đáp ứng một nhu cầu hoặc giải quyết một vấn đề cụ thể mà người dùng đang gặp phải. Điều này có thể là cung cấp thông tin, tạo điều kiện thuận lợi cho giao tiếp, quản lý công việc, giải trí, hoặc bất kỳ nhu cầu nào khác.

2. Đối tượng người dùng: Xác định rõ ràng ai sẽ là người dùng chính của ứng dụng. Điều này bao gồm độ tuổi, giới tính, sở thích, nghề nghiệp, và các đặc điểm khác.
3. Tính năng chính: Xác định những tính năng và chức năng chính mà ứng dụng sẽ cung cấp. Điều này sẽ giúp xác định cấu trúc và giao diện của ứng dụng.
4. Lợi ích cho người dùng: Ứng dụng cần mang lại giá trị cho người dùng, có thể là tiết kiệm thời gian, cải thiện hiệu suất công việc, cung cấp trải nghiệm giải trí, hoặc cung cấp thông tin hữu ích.
5. Mục tiêu kinh doanh: Nếu ứng dụng nhằm mục đích kinh doanh, xác định cách mà ứng dụng sẽ tạo ra doanh thu, như quảng cáo, mua hàng trong ứng dụng, hoặc các mô hình kinh doanh khác.

Mục tiêu phải rõ ràng và được xác định rõ, tránh đặt mục tiêu mơ hồ hoặc chung chung vì không mang lại định hướng đầy đủ.

Ví dụ: Mục tiêu của phần mềm QLSV:

- Hỗ trợ quản lý quá trình học, chương trình đào tạo và kết quả học tập
- Hỗ trợ việc báo cáo thống kê, tìm kiếm...
- Giúp sinh viên, phụ huynh có khả năng theo dõi tốt việc học tập của sinh viên

Ví dụ, nếu bạn muốn phát triển một ứng dụng học tiếng Anh, mục tiêu của ứng dụng có thể bao gồm:

- Giúp người dùng cải thiện kỹ năng tiếng Anh qua các bài học, bài tập và trò chơi.
- Cung cấp tài liệu học tập phong phú và đa dạng.
- Theo dõi tiến trình học tập của người dùng và cung cấp phản hồi.
- Tạo môi trường học tập thú vị và dễ tiếp cận.

Bằng cách xác định rõ ràng mục tiêu của ứng dụng, bạn sẽ có một hướng đi cụ thể trong quá trình phát triển và đảm bảo rằng ứng dụng sẽ đáp ứng được nhu cầu của người dùng.

2.2.3. Xác định loại ứng dụng

Để xác định loại ứng dụng trong một dự án phần mềm, bạn cần xem xét ứng dụng cần được xác định dựa trên nền tảng nào (hệ điều hành), ngỗn ngữ lập trình hỗ trợ (C#, Java…), hệ quản trị cơ sở dữ liệu, các thành phần khác liên quan và loại ứng dụng (Desktop Application, Web application, Mobile application, …)

2.2.4 Lập kế hoạch dự án

Một thực tế là những dự án CNTT có quy mô lớn thường vượt qua ngân sách trung bình được đề ra ban đầu. Có một số nhân tố ảnh hưởng đến thực tế này. Trước đây chúng ta đã xác định được là do các yêu cầu về chức năng, yêu cầu kỹ thuật được xây dựng không đầy đủ.

Nếu như quá trình lập kế hoạch không được thực hiện đúng cách thì ảnh hưởng của nó có thể sẽ là một thảm họa. Ngược lại, nếu quá trình lập kế hoạch dự án được thực hiện tốt, thì những lỗi trong yêu cầu kĩ thuật có thế được xác định sớm và dự án sẽ có được một nền tảng vững chắc. Trong bài học này chúng ta sẽ học cách đặt nền tảng cho một dự án.

Bản chất động và sự phức tạp của các dự án công nghệ thông tin đòi hỏi một người quản lý dự án xuất sắc phải thiết lập được một nền tảng vững chắc. Thiết lập được nền tảng như vậy đòi hỏi phải có sự hiểu biết vững chắc về phân tích cấu trúc chi tiết công việc (Work Breakdown Structure - WBS) và cách sử dụng các sơ đồ mạng (Network Diagram), phương pháp đường tới hạn (Critical Path Method), lịch trình dự án (Project Schedules) và ngân sách dự án. Không nắm vững được những vấn đề này thường dẫn tới việc dự án hoạt động với hiệu quả không ổn định thậm chí có thể thất bại. Ngoài ra, nếu bạn nắm vững những công cụ này, bạn sẽ dần thấy mình có một cơ sở vững chắc bất kể bạn đang làm cho dự án CNTT nào.

Kỳ vọng cho việc lập kế hoạch dự án

Lập kế hoạch là một kỹ năng mà nhiều người nghĩ rằng mình đã có kinh nghiệm, tuy nhiên rất ít người thực sự có được kỹ năng này. Xây dựng một bản kế hoạch dự án toàn diện là một trong những công việc khó khăn nhất trong suốt vòng đời của dự án.

Đặt những kỳ vọng có tính thực tế cho quy trình lập kế hoạch và thảo luận về các phương pháp truyền tải những kỳ vọng này trước khi bắt đầu công việc lập kế hoạch dự án.

Là một người quản lý dự án CNTT, bạn được đánh giá chủ yếu trên cơ sở thành công hay thất bại của các dự án mà bạn đang tham gia. Và bạn được giao thời gian hoặc nguồn lực để lập kế hoạch. Do các dự án CNTT rất phức tạp và luôn biến động nên đòi hỏi quá trình lập kế hoạch tổng thể phải thành công.

Nếu muốn dự án hoàn thành đúng thời hạn, đúng ngân sách và đúng yêu cầu kỹ thuật thì phải đảm bảo ban quản lý dự án có những kỳ vọng hợp lý về thời gian và tài nguyên để hoàn thành công việc.

2.2.4.1 Định nghĩa lập kế hoạch

Lập kế hoạch là một phương thức tiếp cận có hệ thống, cách nhìn chỉnh thể, toàn diện dự án nhằm xác định các phương pháp, tài nguyên và các công việc cần thiết để đạt được mục tiêu.

Mục tiêu của các Dự án CNTT là hoàn thành đúng thời hạn, đúng ngân sách, cung cấp hiệu quả hoạt động mong muốn về mặt kỹ thuật được chấp nhận của nhà tài trợ/khách hàng.

Quá trình lập kế hoạch xác định cần phải làm gì? Làm như thế nào? Ai là người thực hiện và thực hiện khi nào? Mục đích chính của quá trình lập kế hoạch vạch ra phương hướng, quan trọng hơn để tránh các vấn đề rắc rối, rủi ro. Lượng thời gian và chi phí dành cho quá trình lập kế hoạch cần phải tỉ lệ trực tiếp với chi phí tiềm năng từ các lỗi gặp phải do thiếu quá trình lập kế hoạch. Mục tiêu là chi đủ tài nguyên trong quá trình lập kế hoạch nhằm thúc đẩy thực hiện thành công dự án chứ không phải là chi phí quá nhiều tài nguyên đến mức không còn đủ cho quá trình thực hiện dự án. Lập kế hoạch dự án là một quá trình lặp diễn ra bất cứ khi nào có những thay đổi quan trọng trong dự án.

Ví dụ: Khi các nhà tài trợ giao cho thực hiện một dự án phát triển Web, Giám đốc dự án Phong đã rất phấn khởi nhưng anh cũng biết rằng có rất nhiều việc đang ở phía trước bởi vì anh vẫn còn chưa quen với các ứng dụng theo mô hình chủ khách 3 tầng. Anh đã tìm thấy một tài nguyên rất quý đó là một SME về Web làm cho một công ty tư vấn có mối quan hệ rất tốt với công ty anh và anh đã dùng người này vào việc xây dựng các yêu cầu kỹ thuật. Anh đã bỏ ra một số tiền trong ước tính về chi phí để có được sự giúp đỡ từ bên ngoài. Bây giờ là lúc lên lịch cho chuyên gia tư vấn giúp anh tách cấu trúc chi tiết để anh có thể định nghĩa các gói công việc, lập hồ sơ mạng, lên lịch trình dự án và ngân sách chính xác.

2.2.4.2 Kế hoạch dự án công nghệ thông tin

Cái gì tạo nên một bản kế hoạch dự án? Một bản kế hoạch dự án thực sự bao gồm những gì? Hiểu biết rõ ràng một bản kế hoạch dự án phải bao gồm những gì có thể là vô giá trong việc thiết lập kỳ vọng đối với ban quản lý dự án.

Định nghĩa:

Kế hoạch dự án CNTT (IT Project Plan) là một tài liệu dự án chính thức do người quản lý dự án, nhà tài trợ, các đối tượng liên quan đến dự án và các thành viên đội dự án xây dựng với mục đích giám sát việc thực hiện dự án. Kích thước và mức độ chi tiết của bản kế hoạch dự án phải tương ứng với loại dự án. Một bản kế hoạch dự án tối thiểu phải có những thành phần quan trọng sau:

- Bản đề xuất dự án (Project Proposal)
- Tôn chỉ dự án/ Bản công bố dự án (Project Charter)
- Tuyên bố về phạm vi (Scope Statement)
- Cấu trúc chi tiết công việc (Work Breakdown Structure)
- Ma trận trách nhiệm (Responsibility Matrix) đối với kết quả chuyển giao chính.
- Ma trận phân bố tài nguyên cho việc thống kê các kỹ năng, nguyên vật liệu và cơ sở vật chất cần thiết.
- Ước tính chi phí và thời gian cho dự án.
- Lịch trình của dự án, bao gồm cả ngày tháng cho tất cả các mốc quan trọng.
- Ngân sách của dự án, bao gồm cả chi phí cơ sở (Cost Baselines)
- Kế hoạch quản lý rủi ro.
- Kế hoạch truyền thông. (Communication Plan)

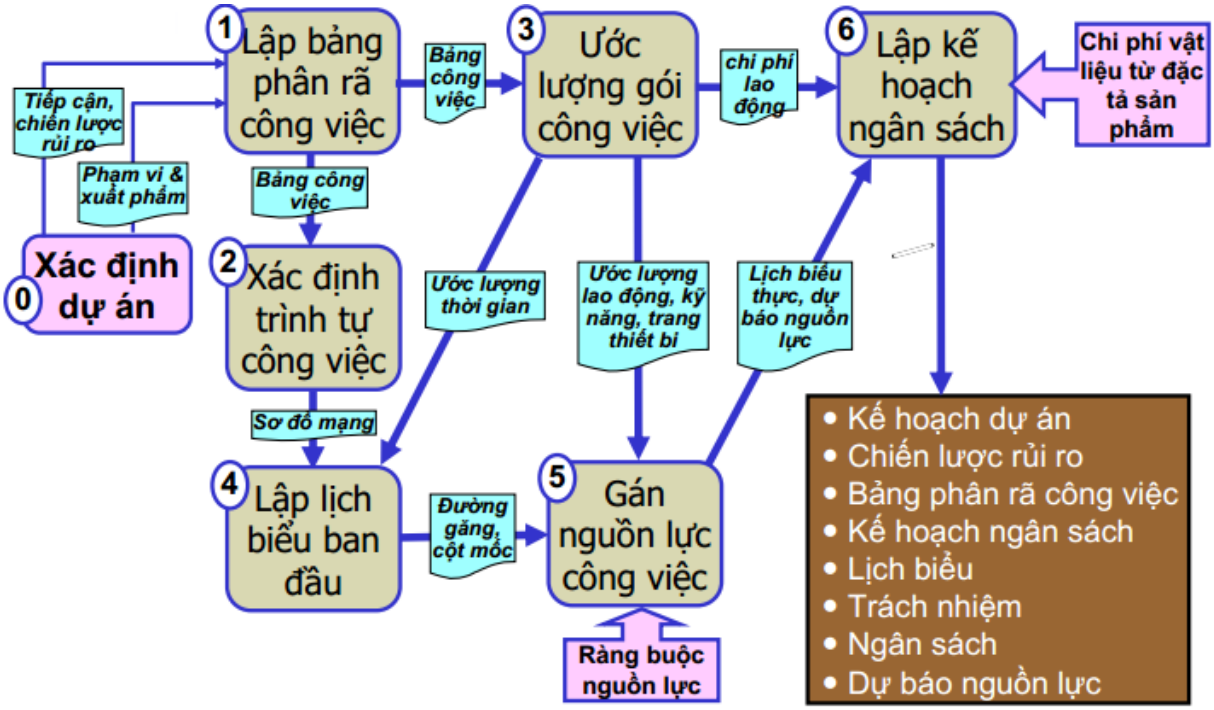
Những Dự án CNTT có quy mô lớn và độ phức tạp cao đòi hỏi phải có kế hoạch chi tiết hơn cho những lĩnh vực cụ thể. Những kế hoạch quản lý chi tiết này bao gồm:

- Kế hoạch quản lý phạm vi
- Kế hoạch quản lý lịch trình
- Kế hoạch kiểm thử hay quản lý chất lượng
- Kế hoạch quản lý nhân sự
- Kế hoạch quản lý mua sắm.
- Kế hoạch phản ứng trước rủi ro hoặc đối phó với những bất ngờ trong dự án.

Không phải mọi kế hoạch dự án đều đòi hỏi phải có những yếu tố này mới đạt tính hiệu quả. Đối với những dự án lớn và phức tạp nên có tất cả những yếu tố này.

2.2.4.3 Triển khai kế hoạch dự án

Sau khi xác định dự án sẽ đạt tới cái gì? Thì bước tiếp theo là xác định cách nó sẽ hoàn thành các mục đích và mục tiêu đó như thế nào. Cách thức để hoàn thành mục đích và mục tiêu là cần thực hiện:



Hình 2.3: Tiến trình triển khai kế hoạch dự án

Cấu trúc chi tiết công việc (WBS): WBS là bản thiết kế phân cấp các sản phẩm, sản phẩm phụ, nhiệm vụ và nhiệm vụ con cần để hoàn thành dự án. Một WBS tốt cung cấp cơ sở cho việc xây dựng các ước lượng thời gian và chi phí có ý nghĩa cũng như lịch biểu khả thi.

Ước lượng thời gian: Người quản lý dự án có thể áp dụng các ước lượng thời gian theo các nhiệm vụ và nhiệm vụ con được nhận diện trong WBS. Có một số kĩ thuật ước lượng có thể được áp dụng tùy theo mức độ tin cậy ước lượng.

Lịch biểu: Người quản lý dự án có thể phân tích từ WBS và các ước lượng thời gian để xây dựng lịch biểu. Trước hết nhận diện mối quan hệ logic giữa các nhiệm vụ rồi áp dụng các ước lượng thời gian cho các nhiệm vụ đó, tiếp theo tính ngày tháng cho từng nhiệm vụ và lưu ý tới các ràng buộc của nó đối với dự án. Qua việc tính toán lịch biểu, người quản lý dự án nhận diện các nhiệm vụ cần cho việc hoàn thành dự án đúng hạn.

Việc xây dựng **Lịch biểu đích** và **Lịch biểu hiện thời** là tính năng quan trọng khác. Lịch biểu đích hay vạch ranh giới, là lịch biểu bạn đồng ý theo đuổi. Lịch biểu hiện thời là sự tổ hợp của lịch biểu đích và hiện trạng so với các khoản mục được chứa trong đó. Lịch biểu hiện thời cũng dự kiến khi nào các hoạt động hiện thời và tương lai sẽ bắt đầu hay kết thúc, hay cả hai nếu dự án tiếp tục như nhịp độ hiện tại của

nó. Người quản lý dự án theo đuổi bất kỳ cái gì cần thiết để so sánh lịch biểu hiện thời với lịch biểu đích. Ngẫu nhiên, phần lớn các dự án đều cho phép bạn sửa đổi lịch biểu đích bất kỳ khi nào cần thiết.

Phân bổ tài nguyên: Các dự án đều tiêu thụ tài nguyên như con người, vật tư, trang thiết bị và không gian làm việc. Người quản lý dự án phải phân bố tài nguyên cho các nhiệm vụ để hoàn thành chúng. Sau khi dùng các tài nguyên, người quản lý dự án có thể xác định liệu tài nguyên có đủ để hoàn thành việc chuyển giao sản phẩm hay không?

Tính chi phí: Sau khi tạo ra cấu trúc phân việc, xác định các ước lượng thời gian; Xây dựng lịch biểu; Và cấp phát tài nguyên, người quản lý dự án có thể tính toán chi phí để thực hiện từng nhiệm vụ và cho toàn bộ dự án. Chi phí được ước lượng cuối cùng trở thành ngân sách. Trong khi thực hiện dự án, người quản lý theo dõi hiệu năng chi phí so với ngân sách.

Kiểm soát rủi ro: Không dự án nào hoạt động độc lập, mà phải nằm trong một môi trường có tổ chức. Nên luôn có những rủi ro, mối đe dọa, ảnh hưởng tới sự thực hiện và sự thành công của dự án. Người quản lý dự án phải xác định những rủi ro này và xây dựng một kế hoạch hiện thực để giảm thiểu tác động.

2.2.2.4 Thực thi kế hoạch

Có thể nói lập kế hoạch dự án tốt là chìa khóa mở ra thành công cho dự án, thì quá trình thực hiện dự án là cơ hội để người quản lý dự án thể hiện vai trò quản lý của mình. Các dự án CNTT thường rất phức tạp, nhưng không phải vì thế mà khó kiểm soát được. Người quản lý dự án phải:

- Chủ động giám sát dự án
- Vạch ra chiến lược theo dõi sự ủng hộ của nhà tài trợ và cổ đông
- Vạch ra chiến lược cho các kênh truyền thông
- Vạch ra chiến lược cho các phương pháp và tiêu chuẩn
- Vạch ra chiến lược giám sát việc tuân thủ các quy tắc đề ra
- Xác lập một hệ thống thông tin để theo dõi
- Xác định được các dấu hiệu rủi ro

2.3 Case Study: Lập kế hoạch dự án phần mềm

a) Mục tiêu

- Phát biểu được bài toán (case study)

- Lập kế hoạch quản lý dự án

b) Yêu cầu

1. Giới thiệu thông tin nhóm thực hiện
2. Đặt vấn đề, Phát biểu được bài toán
3. Khảo sát hiện trạng
4. Xác định mục tiêu
5. Xác định loại ứng dụng
6. Lập kế hoạch dự án.

CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

Nội dung của chương:

3.1. Giới thiệu về phương pháp phân tích thiết kế hướng đối tượng

3.2 Use case và phân tích yêu cầu

3.3. Các mô hình trong phân tích thiết kế hướng đối tượng

Mục tiêu của chương:

- Biết đặc tả mục tiêu, yêu cầu của dự án
- Biết thu thập yêu cầu dựa trên các phương pháp kỹ thuật và công cụ thu thập phân loại yêu cầu
- Áp dụng được các kiến thức phân tích đặc tả yêu cầu phần mềm để giải quyết các vấn đề trong quy trình phát triển phần mềm
- Biết áp dụng để phân tích thiết kế một bài toán

3.1. Giới thiệu về phương pháp phân tích thiết kế hướng đối tượng

Vào thập niên 90, phương pháp tiếp cận phân tích thiết kế đối tượng là sự tổng hợp của phương pháp Descartes và phương pháp hệ thống. Trong khi các mô hình được đưa ra trong những thập niên trước thường đưa ra dữ liệu và xử lý theo hai hướng độc lập nhau. Khái niệm đối tượng là sự tổng hợp giữa khái niệm xử lý và khái niệm dữ liệu chung trong một cách tiếp cận, và một hệ thống là một tập hợp các đối tượng liên kết nội. Có nghĩa rằng việc xây dựng hệ thống chính là việc xác định các đối tượng đó bằng cách cố gắng ánh xạ các đối tượng của thế giới thực thành đối tượng hệ thống, thiết kế và xây dựng nó, và hệ thống hình thành chính là qua sự kết hợp của các đối tượng này. Phương pháp hướng đối tượng được xem là phương pháp phân tích thiết kế thế hệ thứ ba, các phương pháp tiêu biểu là OOD, HOOD, BON, OSA, … và sau này là OOSA, OOA, OMT, CRC, OOM, OOAD, OOSE, RUP/UML

Đặc trưng cơ bản

- *Tính bao bọc (encapsulation):* quan niệm mối quan hệ giữa đối tượng nhận và đối tượng cung cấp thông qua khái niệm hộp đen. Nghĩa là đối tượng nhận chỉ truy xuất đối tượng cung cấp qua giao diện được định nghĩa bởi đối tượng cung cấp, đối tượng nhận không được truy cập đến các đặc trưng được xem là "nội bộ" của đối tượng cung cấp.
- *Tính phân loại (classification):* gom nhóm các đối tượng có cùng cấu trúc và hành vi vào một lớp (class).

- *Tính kết hợp (aggregation):* kết hợp các đối tượng và các đối tượng cấu thành nó để mô tả cấu trúc cục bộ của đối tượng (ví dụ: toà nhà <-> phòng, xe <-> sườn xe, bánh xe,…) , hoặc sự liên kết phụ thuộc lẫn nhau giữa các đối tượng.
- *Tính thừa kế (heritage):* phân loại tổng quát hoá và chuyên biệt hoá các đối tượng, và cho phép chia sẽ các đặc trưng của một đối tượng.

Phân loại

Phương pháp lập trình hướng đối tượng được chia thành 2 hướng như sau:

- Hướng lập trình: từ lập trình đơn thể chuyển sang lập trình hướng đối tượng với lý thuyết cơ bản dựa trên việc trừu tượng hóa kiểu dữ liệu.
- Hướng hệ quản trị CSDL: phát triển thành CSDL hướng đối tượng

Có 2 cách tiếp cận riêng biệt:

- Phương pháp kỹ thuật: hướng công nghệ phần mềm như OOD, HOOD, BON, BOOCH, MECANO, OODA,…
- Phương pháp toàn cục: hướng về HTTT như OOA, OOSA, OOAD, OMT, OOM,…

Ưu điểm

- Cấu trúc hoá được các cấu trúc phức tạp và sử dụng được cấu trúc đệ qui: các phương pháp đối tượng đều sử dụng các mô hình bao gồm nhiều khái niệm để biểu diễn nhiều ngữ nghĩa khác nhau của hệ thống. Ví dụ: trong mô hình lớp của OMT có khái niệm mối kết hợp thành phần cho phép mô tả một đối tượng là một thành phần của đối tượng khác, trong khi nếu dùng mô hình ER truyền thống không có khái niệm này do đó không thể biểu diễn được quan hệ thành phần.
- Xác định được đối tượng của hệ thống qua định danh đối tượng
- Tính thừa kế được đưa ra tạo tiền đề cho việc tái sử dụng

Mô hình

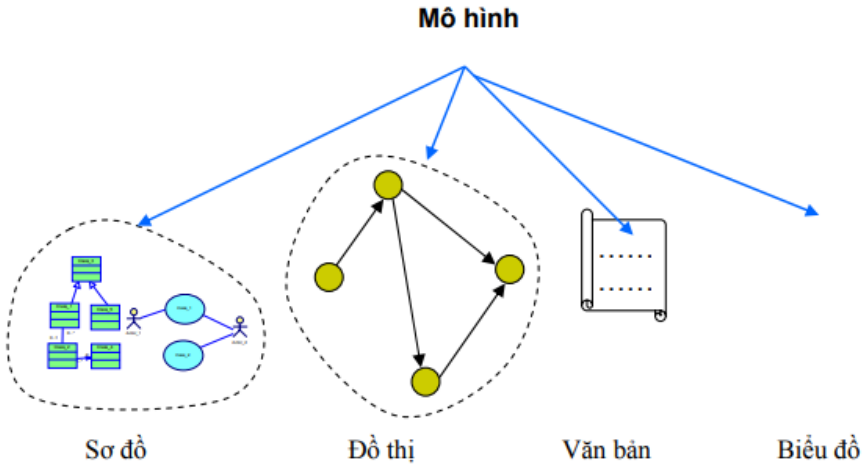
Mô hình (model) là một dạng thức trừu tượng về một hệ thống, được hình thành để hiểu hệ thống trước khi xây dựng hoặc thay đổi hệ thống đó. Theo Efraim Turban, mô hình là một dạng trình bày đơn giản hoá của thế giới thực. Bởi vì, hệ thống thực tế thì rất phức tạp và rộng lớn và khi tiếp cận hệ thống, có những chi tiết những mức độ phức tạp không cần thiết phải được mô tả và giải quyết. Mô hình cung cấp một phương tiện (các khái niệm) để quan niệm hoá vấn đề và giúp chúng ta có thể trao đổi các ý tưởng trong một hình thức cụ thể trực quan, không mơ hồ.

Các đặc điểm của một mô hình:

- Diễn đạt một mức độ trừu tượng hóa (ví dụ: mức quan niệm, mức tổ chức, mức vật lý,…)
- Tuân theo một quan điểm (quan điểm của người mô hình hoá)
- Có một hình thức biểu diễn (văn bản, đồ họa: sơ đồ, biểu đồ, đồ thị,…)

Hầu hết các kỹ thuật mô hình hóa sử dụng trong phân tích thiết kế là các ngôn ngữ đồ họa (đa số là sơ đồ - diagram), các ngôn ngữ này bao gồm một tập hợp các ký hiệu. Các ký hiệu này được dùng đi kèm theo các qui tắc của phương pháp luận giúp cho việc trao đổi các quan hệ thông tin phức tạp được rõ ràng hơn việc mô tả bằng văn bản.

Ví dụ :



Hình 3.1. Mô hình

Mô hình tĩnh và mô hình động

Mô hình tĩnh (static model): được xem như là hình ảnh về thông số hệ thống tại một thời điểm xác định. Các mô hình tĩnh được dùng để trình bày cấu trúc hoặc những khía cạnh tĩnh của hệ thống.

Mô hình động (dynamic model): được xem như là một tập hợp các hành vi, thủ tục kết hợp với nhau để mô tả hành vi của hệ thống. Các mô hình động được dùng để biểu diễn sự tương tác của các đối tượng để thực hiện công việc hệ thống.

Mục đích của mô hình hoá

Đứng trước sự gia tăng mức độ phức tạp của một hệ thống, việc trực quan hoá và mô hình hóa ngày càng trở nên chính yếu trong cách tiếp cận xem xét về một hệ thống, đặc biệt là cách tiếp cận hướng đối tượng. Việc sử dụng các ký hiệu để trình bày hoặc mô hình hóa bài toán có các mục đích sau:

- Làm sáng tỏ vấn đề: chúng ta có thể đưa ra được các lỗi hoặc các thiếu xót của hệ thống từ việc tiếp cận trực quan đồ họa hơn là các dạng trình bày khác như văn bản, đoạn mã,… Hơn nữa, việc mô hình hoá giúp chúng ta dễ dàng hiểu được hệ thống.
- Mô phỏng được hình ảnh tương tự của hệ thống: hình thức trình bày của mô hình có thể đưa ra được một hình ảnh giả lập như hoạt động thực sự của hệ thống thực tế, điều này giúp cho người tiếp cận cảm thấy thuận tiện khi làm việc với mô hình (là hình ảnh thu nhỏ của hệ thống thực tế)
- Gia tăng khả năng duy trì hệ thống: các ký hiệu trực quan có thể cải tiến khả năng duy trì hệ thống. Thay đổi các vị trí được xác định trực quan và việc xác nhận trực quan trên mô hình các thay đổi đó sẽ giảm đi các lỗi. Do đó, chúng ta có thể tạo ra các thay đổi nhanh hơn và các lỗi được kiểm soát hoặc xảy ra ít hơn.
- Làm đơn giản hóa vấn đề: mô hình hoá có thể biểu diễn hệ thống ở nhiều mức, từ mức tổng quát đến mức chi tiết, mức càng tổng quát thì ký hiệu sử dụng càng ít (do đó càng đơn giản hoá việc hiểu) và hệ thống được biểu diễn càng tổng quát.

3.2. Use case và phân tích yêu cầu

3.2.1. Biểu đồ Use case

Trong giai đoạn phân tích, kết quả của quá trình khảo sát yêu cầu phản ánh quá trình làm việc của người phát triển với người sử dụng. Các kết quả này phải nhắm đến yếu tố của người dùng. Có nghĩa là người phát triển trước tiên phải diễn đạt bức tranh của hệ thống tương lai theo cách nhìn của người sử dụng. Điều này sẽ giúp cho người dùng có thể thấy được hệ thống sẽ làm thoã mãn các yêu cầu như thế nào và đó chính là chìa khoá đầu vào cho việc phát triển hệ thống trong các giai đoạn về sau. Một công cụ giúp diễn đạt điều này chính là mô hình use case.

Jacobson và cộng sự của ông (1992) là những người tiên phong trong việc sử dụng mô hình use case để phân tích yêu cầu hệ thống. Bởi vì mô hình use case đặt trọng tâm để biểu diễn hệ thống hiện tại làm gì, hệ thống mới sẽ làm gì và môi trường của nó. Nó giúp cho người phát triển có thể hiểu rõ về yêu cầu chức năng hệ thống mà không quan tâm đến chức năng này được cài đặt như thế nào.

Để hiểu yêu cầu của hệ thống, chúng ta phải tìm ra người dùng sẽ sử dụng hệ thống như thế nào. Do đó từ một quan điểm người dùng chúng ta phát hiện các tình

huống sử dụng khác nhau của người dùng, các tình huống này được thiết lập bởi các use case, tổng hợp các use case và tác nhân cùng với quan hệ giữa chúng sẽ cho ta một mô hình use case mô tả yêu cầu của hệ thống.

Trong chương 5, quá trình mô hình hoá nghiệp vụ được áp dụng đối với các hệ thống nghiệp vụ và kết quả của nó sẽ cung cấp sơ đồ use case từ việc thống nhất các yêu cầu hệ thống phần mềm để tự động hoá hoạt động của hệ thống nghiệp vụ đó. Tuy nhiên, trong những hệ thống mà không có hoạt động nghiệp vụ (ví dụ: hệ thống nhúng), hoặc các nghiệp vụ của hệ thống không quá phức tạp hoặc không quan tâm để mô hình hoá nghiệp vụ thì việc xây dựng mô hình use case phần mềm sẽ là bước tiếp cận mô hình hoá đầu tiên về hệ thống. Một tiến trình xây dựng sơ đồ use case bao gồm các bước sau:

- Xác định tác nhân hệ thống
    - o Ai đang sử dụng hệ thống?
    - o Hoặc trong trường hợp phát triển mới thì ai sẽ sử dụng hệ thống?
- Phát triển use case
    - o Người dùng (tác nhân) đang làm gì với hệ thống?
    - o Hoặc trong trường hợp hệ thống mới thì người dùng sẽ làm gì với hệ thống? - Xây dựng sơ đồ use case
    - o Xác định mối quan hệ giữa tác nhân – use case
    - o Xác định mối quan hệ giữa các use case
- Phân chia sơ đồ use case thành các gói (package)

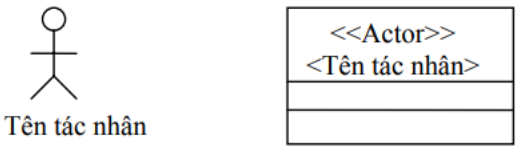
Xác định tác nhân

Tác nhân (actor)

Ý nghĩa: một tác nhân là một đối tượng bên ngoài hệ thống giao tiếp với hệ thống theo một trong những hình thức sau:

- Tương tác, trao đổi thông tin với hệ thống hoặc sử dụng chức năng hệ thống - Cung cấp đầu vào hoặc nhận các đầu ra từ hệ thống
- Không điều khiển hoạt động của hệ thống

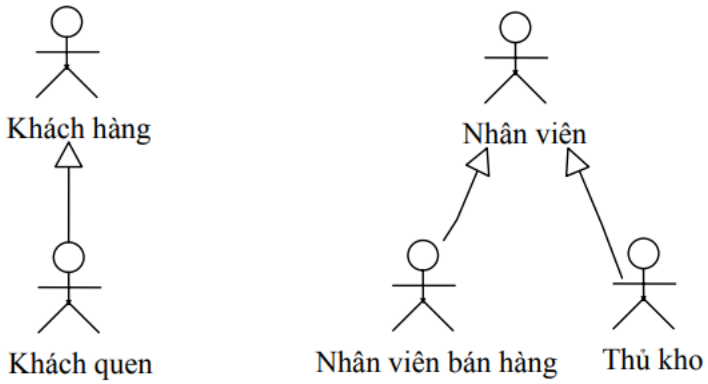
Ký hiệu



Hình 3.2. Tác nhân

Tên tác nhân: tên tác nhân là một **danh từ**

Quan hệ giữa các tác nhân

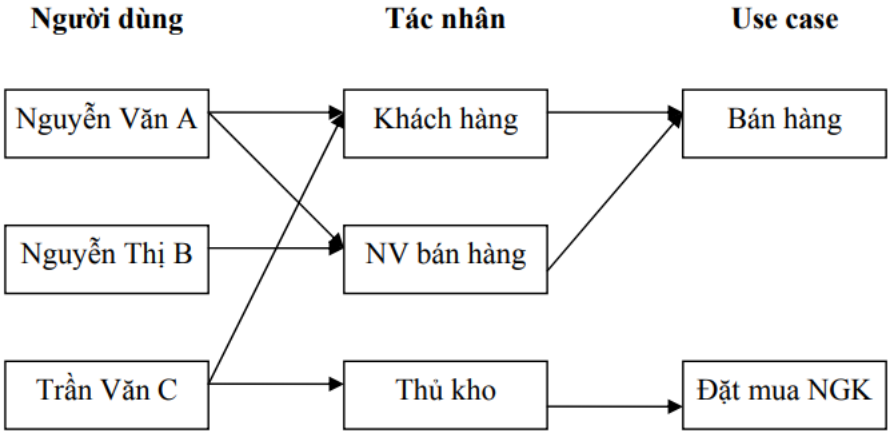


Hình 3.3. Quan hệ giữa các tác nhân

Xác định tác nhân cũng được xem có tầm quan trọng như xác định class, use case, liên kết,….

Khi xác định người sử dụng phần mềm hệ thống, chúng ta đừng quan trọng vấn đề quan sát người nào đang sử dụng hệ thống mà chúng ta nên xác định xem vai trò chịu trách nhiệm trong việc sử dụng hệ thống. Nghĩa là tác động lên hệ thống theo nghĩa cung cấp thông tin cho hệ thống hoặc nhận kết quả xử lý từ hệ thống.

Tác nhân được hiểu là một vai trò tham gia vào hệ thống không giống như một con người cụ thể hoặc một công việc. Một đối tượng có thể tham gia vào một hoặc nhiều vai trò

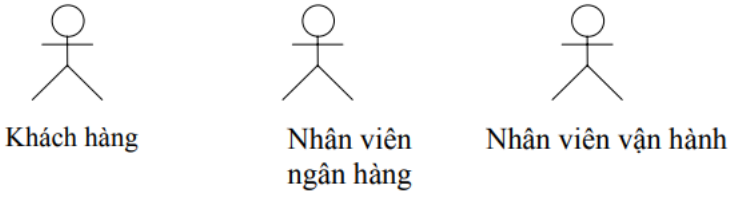


Hình 3.4. Vai trò của tác nhân

Qua quá trình khảo sát và phân tích tài liệu hệ thống, chúng ta có thể nhận ra các tác nhân thông qua các câu hỏi sau:

- Ai đang sử dụng hệ thống? Hoặc ai được tác động bởi hệ thống? Hoặc nhóm đối tượng nào cần hệ thống trợ giúp để làm công việc? (tác nhân chính)
- Ai tác động tới hệ thống? Những nhóm đối tượng nào hệ thống cần để thực hiện hoạt động của nó (hoạt động gồm chức năng chính và chức năng phụ, như là chức năng quản trị)?
- Những phần cứng hoặc hệ thống bên ngoài nào sử dụng hệ thống?

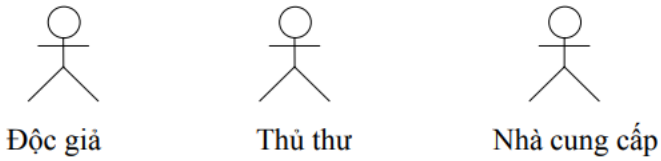
Ví dụ: trong hoạt động của máy ATM của một ngân hàng, các tác nhân được xác định là:



Hình 3.5. Ví dụ tác nhân của hệ thống ATM

Trong đó, các tác nhân Khách hàng, Nhân viên ngân hàng là các tác nhân chính (primary actor) của hệ thống ATM. Bởi vì khách hàng là mục tiêu mà hệ thống tượng tác; Nhân viên ngân hàng sử dụng hệ thống để trợ giúp công việc. Trong khi đó, Nhân viên vận hành là tác nhân phụ (secondary actor) bởi vì tác nhân này đảm nhận những chức năng phụ mà hệ thống cần có để thực hiện hoạt động của nó.

Hoặc trong một thư viện của trường đại học, các tác nhân của hệ thống phần mềm quản lý thư viên gồm:



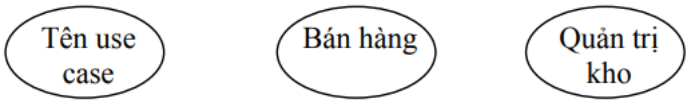
Hình 3.6. Ví dụ tác nhân của hệ thống quản lý thư viện

Xác định use case

Use case

Một Use case được xem như một chức năng hệ thống từ quan điểm người dùng, như vậy tập hợp tất cả use case biểu diễn bộ mặt của hệ thống bao gồm các chức năng cần có để cung cấp cho các đối tượng tương tác làm việc với hệ thống. Như vậy, use case dùng để mô tả yêu cầu của hệ thống mới về mặt chức năng, mỗi chức năng sẽ được biểu diễn như một hoặc nhiều use case.

Ví dụ: hệ thống cửa hàng NGK ta có một vài use case
Bán hàng, quản trị tồn kho,…



Hình 3.7: Các Use case

Xác định use case

Chúng ta bắt đầu từ tập các tác nhân đã xác định trong bước đầu tiên. Ứng với mỗi tác nhân:

- Tìm các nhiệm vụ và chức năng mà tác nhân sẽ thi hành hoặc hệ thống cần tác nhân để thi hành và mô hình hoá nó như là use case. Use case sẽ đại diện một dòng sự kiện dẫn tới một mục tiêu rõ ràng (hoặc trong một vài trường hợp, dẫn tới một vài mục tiêu riêng biệt có thể là các phương án thay thế cho tác nhân hoặc cho hệ thống so với dòng sự kiện chính)
- Đặt tên cho use case: tên use case nên đặt nhằm phản ánh một mô tả tổng quan về chức năng của use case. Tên nên diễn đạt những gì xảy ra khi một thể hiện của use case được thi hành. Một hình thức đặt tên use case phổ biến là : động từ (do) + danh từ (what).
- Mô tả use case một cách ngắn gọn bằng việc áp dụng các thuật ngữ gần gũi với người sử dụng. Điều này sẽ làm cho mô tả use case ít mơ hồ.

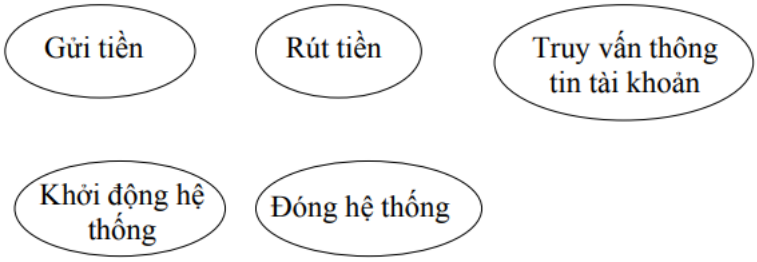
Ví dụ: trong hệ thống ATM

Tác nhân Khách hàng sẽ sử dụng hệ thống qua các chức năng:

- Gửi tiền
- Rút tiền
- Truy vấn thông tin về tài khoản

Tác nhân Nhân viên vận hành sẽ sử dụng các chức năng

- Khởi động hệ thống
- Đóng hệ thống



Hình 3.8: Các UC trong hệ thống ATM

Gửi tiền: khách hàng đăng nhập vào hệ thống và yêu cầu gửi tiền vào tài khoản. Khách hàng sẽ xác định tài khoản và số tiền gửi, hệ thống sẽ tạo một giao tác gửi tiền và lưu vào hệ thống. Các bước như sau:

- Yêu cầu xác định tài khoản
- Hệ thống hỏi số tiền gửi
- Nhập vào số tiền gửi
- Khách hàng đưa tiền vào bao thư và chuyển vào máy ATM

Rút tiền: khách hàng đăng nhập hệ thống và yêu cầu rút tiền từ tài khoản. Khách hàng xác định tài khoản và lượng tiền rút. Sau khi kiểm tra số dư tài khoản còn đủ, hệ thống sẽ tạo một giao tác rút tiền và lưu vào hệ thống. Các bước như sau:

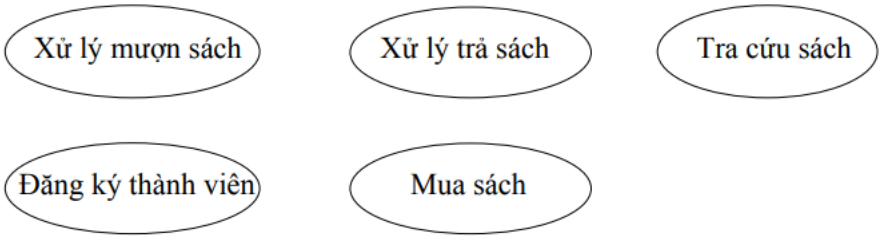
- Yêu cầu xác định tài khoản
- Yêu cầu xác định số tiền cần rút
- Nhập số tiền rút
- Kiểm tra số dư có đủ không
- Chuyển tiền ra ngoài

Truy vấn thông tin tài khoản: khách hàng đăng nhập vào hệ thống và yêu cầu xem thông tin về các giao dịch của tài khoản. Hệ thống hiển thị các thông tin về các giao tác đã tạo lên màn hình cho khách hàng.

Khởi động hệ thống: hệ thống được khởi động khi nhân viên vận hành bật công tắc của máy. Nhân viên vận hành sẽ được yêu cầu nhập vào số tiền hiện hành của máy nằm trong két đựng tiền. Sau đó, hệ thống sẽ thiết lập một kết nối tới ngân hàng và các dịch vụ của máy ATM bắt đầu vận hành.

Đóng hệ thống: hệ thống được đóng lại khi nhân viên vận hành đảm bảo rằng không có khách hàng nào đang sử dụng máy. Khi đó, nhân viên vận hành sẽ lấy các bao tiền gửi ra, bô sung lượng tiền, giấy,…

Trong hệ thống quản lý thư viện, các use case được xác định như sau:



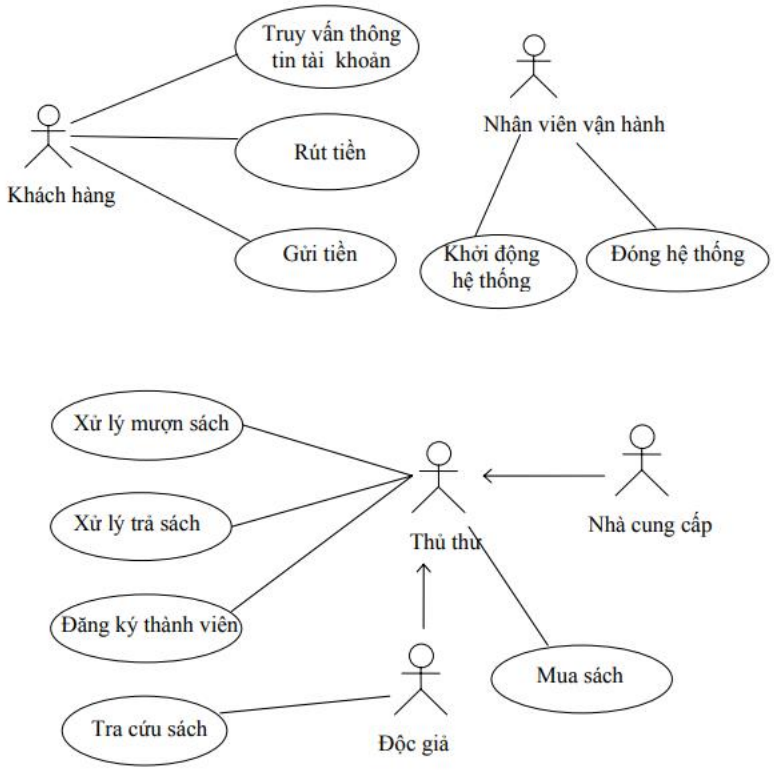
Hình 3.9: Các UC trong hệ thống quản lý thư viện

Xác định mối quan hệ

Quan hệ tác nhân – use case: quan hệ này cho biết tác nhân sẽ tương tác với use case. Một use case luôn luôn được khởi tạo bởi một tác nhân và có thể tương tác với nhiều tác nhân.

Ký hiệu

Ví dụ



Hình 3.10: Biểu đồ Use case

Mối quan hệ giữa các use case

Việc mô tả use case có thể sẽ khó hiểu nếu use case này chứa đựng nhiều dòng phụ hoặc dòng ngoại lệ chỉ xử lý cho những sự kiện trong những điều kiện đặc biệt. Để làm đơn giản mô tả này chúng ta sử dụng thêm các mối kết hợp <<extend>> và <<include>>.

Liên kết mở rộng (<<extend>>): được dùng khi chúng ta có một use case tương tự như use case khác nhưng có nhiều hơn một vài xử lý đặc biệt. Giống như liên kết tổng quát - chuyên biệt, trong đó, use case chuyên biệt là một mở rộng của use case tổng quát bằng việc đưa thêm vào các hoạt động hoặc ngữ nghĩa mới vào use case tổng quát, hoặc bỏ qua hoạt động của use case tổng quát.

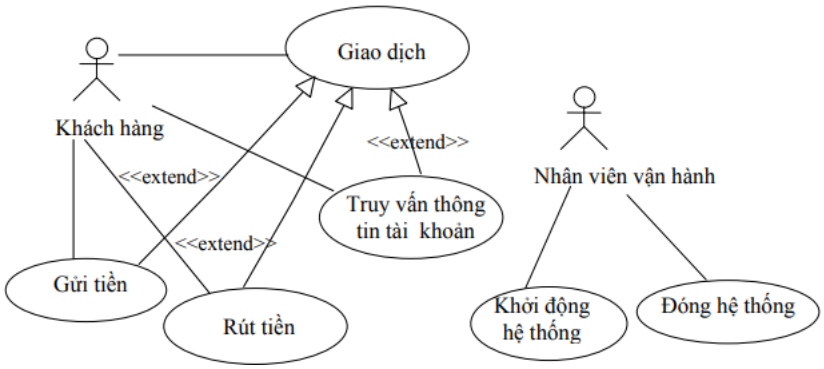
Ví dụ: giả sử Đăng nhập là một use case cơ bản. Use case này sẽ đại diện cho tất cả những gì được xem là thực hiện đăng nhập một cách xuyên suốt.Tuy nhiên, nhiều vấn đề có thể tác động đến dòng sự kiện chính. Ví dụ, mã số PIN không hợp lệ, hoặc thẻ không đọc được do bị hư,…. Do đó, chúng ta không phải luôn luôn thi hành các hoạt động thường xuyên của một use case được cho và như vậy, cần thiết tạo ra các use case mới để giải quyết những tình huống mới. Tất nhiên, chúng ta có thể đưa vào use case cơ bản các nội dung xử lý đặc biệt đó. Tuy nhiên, điều này có thể dẫn đến sự phức tạp với nhiều luận lý riêng biệt và sẽ làm giảm vai trò của dòng chính.

Để giải quyết vấn đề này chúng ta có thể sử dụng quan hệ <<extend>>. Ở đây chúng ta gom các xử lý cơ bản hoặc bình thường vào trong một use case (cơ bản). Các xử lý đặc biệt vào những use case (chuyên biệt) khác. Rồi tạo một liên kết <<extend>> giữa use case cơ bản tới các use case chuyên biệt để khai báo rằng: ngoài xử lý dòng chính (cơ bản), use case cơ bản có mở rộng đến các tình huống xử lý đặc biệt được giải quyết trong các use case chuyên biệt.

Hình 3.11: Mối quan hệ giữa các UC

Tạo một use case tổng quát có tên là Giao dịch của các use case Rút tiền, Gửi tiền và Truy vấn thông tin tài khoản. Tạo các liên kết <<extend>> từ use case Giao dịch đến các use case này. Như vậy, một rút tiền, hoặc gửi tiền, hoặc truy vấn thông tin tài khoản là một loại giao dịch mà khách hàng có thể sử dụng trên máy ATM. Có nghĩa rằng, các

xử lý trong use case Giao dịch sẽ cung cấp một dòng chung và khi khách hàng chọn một loại giao dịch đặc biệt nào đó thì use case này sẽ mở rộng việc giải quyết thông qua các use case chuyên biệt.

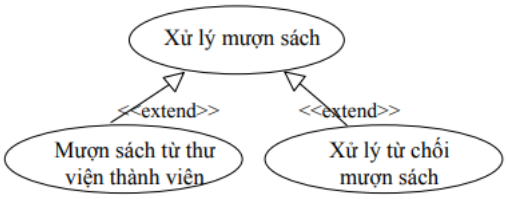


Hình 3.12: Ví dụ mối quan hệ giữa các biểu đồ UC

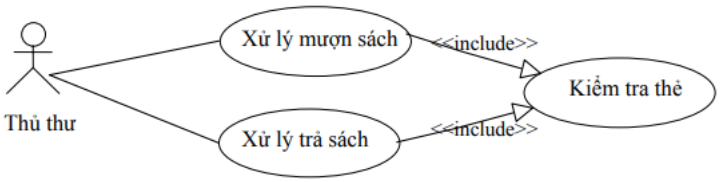
Giao dịch: khách hàng tương tác với hệ thống bắt đầu bằng việc đăng nhập hệ thống. Sau khi đăng nhập, khách hàng có thể thực hiện các giao dịch. Sau đây là các bước:

- Đưa thẻ vào máy
- Thực hiện đăng nhập
- Yêu cầu loại giao dịch
- Nhập loại giao dịch
- Thực hiện giao dịch
- Đẩy thẻ ra
- Yêu cầu lấy thẻ
- Lấy thẻ

Trong hệ thống quản lý thư viện, use case Mượn sách ngoài dòng hoạt động chính còn có các dòng phụ. Dòng phụ này sẽ được kích hoạt để giải quyết vấn đề khi một độc giả đến mượn tài liệu nhưng không có trong thư viện và thư viện sẽ mượn tài liệu đó từ những thư viện khác có liên kết. Hoặc do đọc giả không thoả các điều kiện để được mượn (mượn sách quá hạn chưa trả của lần mượn trước). Do đó, chúng ta tách dòng phụ này và use case “Mượn sách từ thư viện thành viên” và “ ”và tạo một liên kết <<extend>> từ use case này đến use case Xử lý mượn sách.

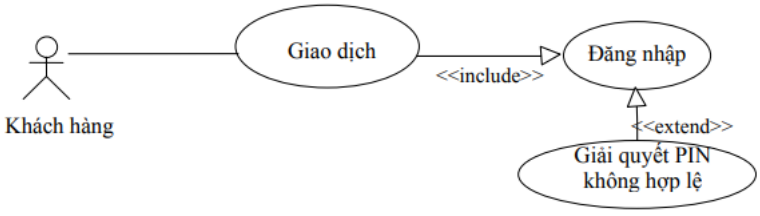


Liên kết sử dụng (<<include>>): được thành lập khi chúng ta có các use case mà tìm thấy một vài use case có những dòng hoạt động chung, và để tránh mô tả dòng hoạt động chung đó lặp lại trên những use case này, chúng ta có thể tách những dòng hoạt động chung đó ra thành một use case. Use case mới này có thể sử dụng bởi những use case khác. Quan hệ giữa những use case với use case được trích ra này gọi là quan hệ <<include>>. Quan hệ sử dụng giúp chúng ta tránh sự trùng lắp bằng cách cho phép một use case có thể được chia sẻ.



Trong ví dụ trên, use case *mượn sách* và *trả sách* đều phải thực hiện công việc kiểm tra thẻ thư viện của đọc giả, do đó chúng ta phát sinh một use case mới là *kiểm tra thẻ* bằng cách trích ra hoạt động kiểm tra thẻ thư viện từ hai use case trên và tạo một liên kết <<include>> tới use case từ hai use case đó tới use case mới. Các use case *Xử lý mượn sách* và *Xử lý trả sách* đều thừa hưởng tất cả hoạt động của use case của use case *kiểm tra thẻ*.

Trong hệ thống ATM, use case Giao dịch sẽ có mối liên kết <<include>> với use case đăng nhập.



Đăng nhập: khách hàng nhập vào mã số PIN gồm bốn ký số. Nếu mã số PIN hợp lệ, tài khoản của khách hàng sẽ sẵn sàng cho các giao dịch. Các bước như sau:

- Yêu cầu password
- Nhập password
- Kiểm tra password

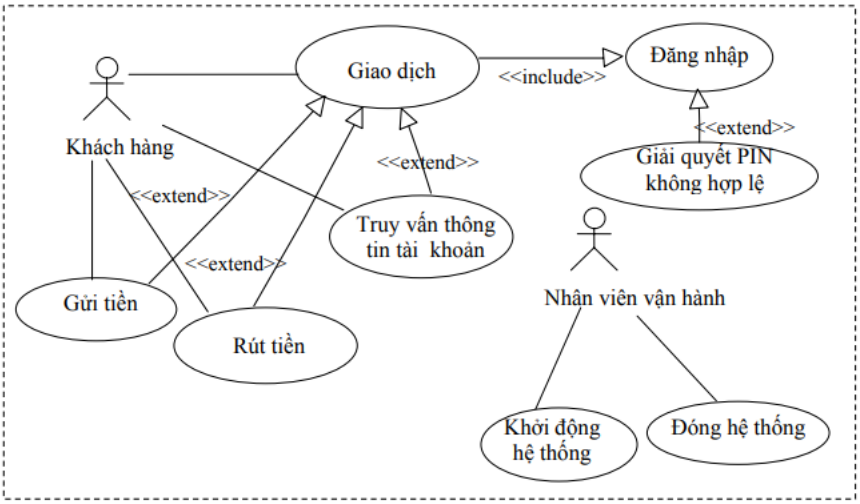
Giải quyết PIN không hợp lệ: nếu mã số PIN không hợp lệ, hệ thống sẽ hiển thị một thông báo tới khách hàng.

Sự giống nhau giữa liên kết <<extend>> và liên kết <<include>> là tất cả đều được xem như là một loại kế thừa. Khi chúng ta muốn chia sẽ một số hoạt động chung trong nhiều use case, dùng liên kết <<include>> bằng cách trích các hoạt động chia sẽ đó thành một use case mới. Khi chúng ta muốn thêm vào một ít khác biệt cho một use case để mô tả một tình huống đặc biệt trong một tình huống chung, chúng ta sẽ tạo một use case mới có liên kết <<extend>> với use case chung đó.

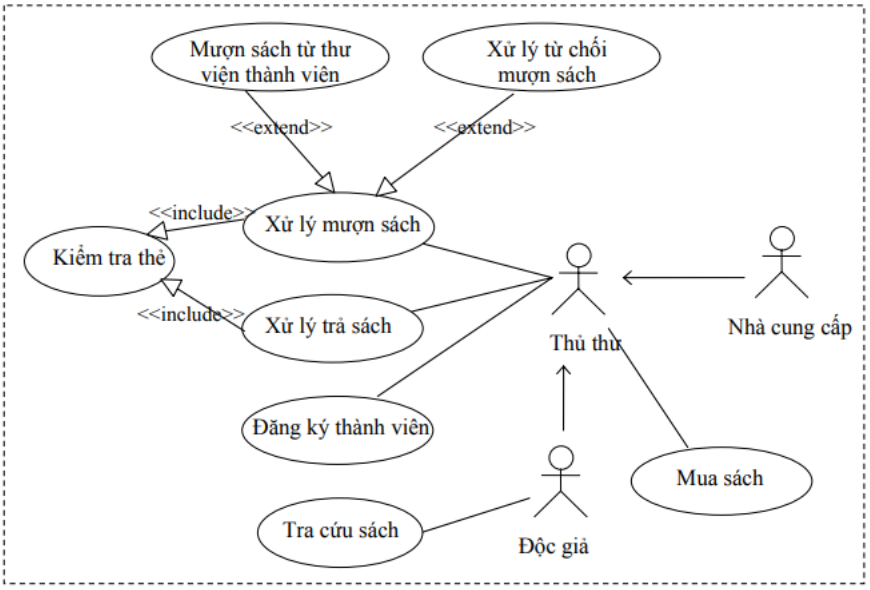
Dựa vào các liên kết được thiết lập cho các use case chúng ta phân use case thành hai loại:

Use case trừu tượng: là use case chưa hoàn hảo nghĩa là không tương tác với bất kỳ một tác nhân nào mà được sử dụng bởi một use case khác. Use case trừu tượng cũng có thể có liên kết <<extend>> hoặc liên kết <<include>> trong những mức độ khác. Ví dụ: các use case *Kiểm tra thẻ, Xử lý từ chối mượn sách,...* là các use case trừu tượng.

Use case cụ thể: là use case có tương tác trực tiếp với một tác nhân. Ví dụ: các use case *Xử lý mượn sách, Xử lý trả sách,* hoặc *Khởi động máy,Đóng máy,...*



Hình 3.13: Mô hình UC của hệ thống ATM

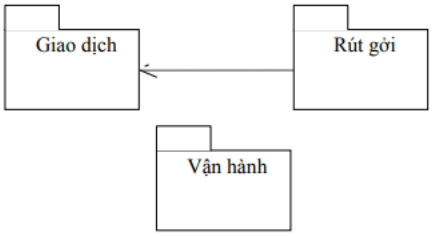


Hình 3.14: Mô hình uc của hệ thống quản lý thư viện

Phân chia các use case thành các gói (package)

Mỗi use case minh hoạ một kịch bản trong hệ thống. Khi gặp những hệ thống tương đối phức tạp thì chúng ta nên thu hẹp tiêu điểm của các kịch bản trong hệ thống bằng cách phân chia thành các gói. Mỗi gói phản ánh một phạm vi của hệ thống mà chúng ta chỉ muốn quản lý nó khi chúng ta truy cập gói đó.

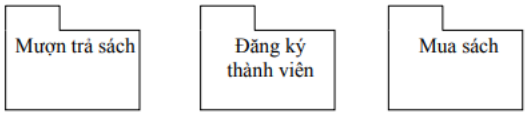
Ví dụ, có thể chia các se case của hệ thống máy ATM thành ba gói: Giao dịch, Rút gửi và Vận hành



Hình 3.15: Ví dụ về biểu đồ gói

Trong đó, gói Giao dịch gồm các use case: Giao dịch, Đăng nhập, Giải quyết PIN không hợp lệ; gói Rút gởi gồm các use case: Gởi tiền, Rút tiền, Truy vấn thông tin tài khoản; và gói Vận hành gồm các use case: Khởi động hệ thống, Đóng hệ thống

Hệ thống quản lý thư viện được chia thành ba gói như sau: Mượn trả sách, đăng ký thành viên, và Mua sách.



Trong đó, gói Mượn trả sách gồm các use case: Xử lý mượn sách, Xử lý trả sách, Kiểm tra thẻ, Mượn sách từ thư viện thành viên, Xử lý từ chối mượn sách; gói Đăng ký thành viên gồm use case: Đăng ký thành viên; gói mua sách gồm use case: Mua sách.

3.2.2. Mô hình hóa yêu cầu chức năng sử dụng use case model

Sử dụng mô hình use case của UML để thể hiện một cách trực quan các chức năng của hệ thống. Cách thực hiện:

- ***Bước 1***: Xác định actor. Actor chính là các đối tượng tương tác với hệ thống. Actor có thể là người dùng, phần cứng mở rộng, hoặc những chủ thể khác. Actor là câu trả lời của câu hỏi "Ai/đối tượng nào kích hoạt chức năng của hệ thống?"
- ***Bước 2***: Xác định các tình huống (use case). Tình huống xử lý biểu diễn hành vi tương tác của actor với hệ thống
- ***Bước 3***: Xây dựng mô hình use case
- ***Bước 4***: Đặc tả use case, các bước thực hiện use case, mô hình hóa các bước bằng sơ đồ activity (có thể tạo ra từ các bước trong đặc tả)

Từ bước 1 đến bước 3 đã được hướng dẫn ở nội dung trên.

Bước 4: Đặc tả use case

Đặc tả Use Case (Use Case Specification) là một tài liệu mô tả chi tiết về cách một hệ thống phần mềm tương tác với các tác nhân bên ngoài để đạt được một mục tiêu cụ thể. Dưới đây là một ví dụ về các thành phần cơ bản và cách thức để đặc tả một Use Case.

Tên use case: Tìm kiếm tài liệu
Actor: Độc giả (thủ thư đóng vai trò là độc giả)
Mô tả: Use case thực hiện việc tìm kiếm tài liệu theo một trong các tiêu chí: loại tài liệu, tên tài liệu, chủ đề, tên tác giả, năm xuất bản.
Precondition: Chức năng tìm kiếm tài liệu được chọn
Poscondition: Nếu tìm kiến thành công thì danh mục các tài liệu được hiển thị để độc giả có thể thực hiện các thao tác tiếp theo: đọc, tải, đăng ký đặt mượn. Ngược lại, thông báo cho độc giả biết là không tìm ra tài liệu.
Basic flow

1. Hệ thống hiển thị giao diện tìm kiếm tài liệu 2. Độc giả chọn loại tài liệu chọn loại bản in hay bản điện tử, nhập các từ khóa cho tên tài liệu, chủ đề, tên tác giả, năm xuất bản. 3. Độc giả chọn nút Tìm kiếm 4. Hệ thống sẽ thực hiện tìm tài liệu dựa trên các thông tin mà độc giả nhập. 5. Nếu có, hệ thống hiển thị danh sách các tài liệu trong Giao diện Kết quả.
Alternate flow
5.1 Nếu không tìm thấy tài liệu theo yêu cầu thì hệ thống hiển thị thông báo không có tài liệu theo yêu cầu 5.2a. Độc giả chọn lại chức năng Tìm Kiếm Tài liệu để tìm tài liệu khác, lặp lại bước 1 đến 3 5.2b. Độc giả kết thúc việc tìm kiếm tài liệu bằng cách chọn nút "Đóng".

3.3. Các mô hình trong phân tích thiết kế hướng đối tượng

Ngôn ngữ mô hình hóa thống nhất UML (Unified Modeling Language) là bộ ký hiệu với biểu đồ quen thuộc nhất để tạo ra bản thiết kế phần mềm. UML đã kết hợp những khái niệm đã có trước đó như biểu đồ của Booch, OMT của Rumbaugh, OOSE của Jacobson và biểu đồ trạng thái của Harel. Những điểm mạnh chủ yếu của UML đã được thừa nhận rộng rãi với nhiều mô hình hóa cấu trúc, quan điểm và công cụ hỗ trợ. Nó cung cấp tính nhiều chiều cho kỹ thuật mô hình với mười bốn biểu đồ để mô hình hóa các khía cạnh tĩnh và động của phần mềm. UML là một công cụ ký hiệu thiết kế và phân tích hướng đối tượng mà điển hình là cung cấp nhiều biểu đồ phân tích hữu ích và thậm chí còn tạo khung mã cho phần mềm.

UML được sử dụng rộng rãi như một kim chỉ nam cho các tài liệu thiết kế và phân tích yêu cầu phần mềm, là cơ sở để phát triển phần mềm. Thông thường, UML có thể được sử dụng để mô hình hóa miền vấn đề; mô tả các yêu cầu của người dùng; xác định các yếu tố kiến trúc quan trọng trong quá trình thiết kế phần mềm chẳng hạn như các lớp và đối tượng, mô tả hành vi và tương tác giữa các yếu tố này và tổ chức cấu trúc phần mềm, chỉ định các ràng buộc của nó, mô tả các thuộc tính cần thiết…

UML cung cấp một số biểu đồ mô hình hóa có thể được nhóm thành hai loại: cấu trúc tĩnh và hành vi (động). Các biểu đồ mô tả cấu trúc tĩnh bao gồm các biểu đồ lớp, biểu đồ thành phần, biểu đồ triển khai…Các biểu đồ mô tả cấu trúc động bao gồm: biểu đồ trình tự, biểu đồ cộng tác, biểu đồ hoạt động, …

Có rất nhiều công cụ UML IDE (môi trường phát triển tương tác); một trong số chúng là mã nguồn mở. Các công cụ UML phổ biến nhất là Rational Rose, Boland Together và Microsoft Visio. Nhiều trong số phần mềm này có khả năng ánh xạ trực tiếp từ biểu đồ UML sang khung mã hóa bằng các ngôn ngữ lập trình phổ biến như C++, C# và Java.

Các phiên bản trước đây của UML 1..* tập trung nhiều vào thiết kế chi tiết cho các lớp, thuộc tính và phương thức. Uml 2.0 đã mở rộng UML để hỗ trợ tốt hơn các cấu trúc mức cao trong kiến trúc. Việc hiểu đầy đủ ý nghĩa của các biểu đồ UML được xem là then chốt để sử dụng cho việc mô tả các quyết định thiết kế. Dưới đây là bảng tóm tắt 13 biểu đồ UML 2.0 trong các danh mục cấu trúc tĩnh và hành vi:

1. Các biểu đồ cấu trúc (tĩnh)

Biểu đồ	**Mô tả**
Lớp (class)	Tổng quan về các lớp dành cho mô hình hóa và thiết kế. Nó chỉ ra cách các lớp liên quan với nhau một cách tĩnh, nhưng không chỉ ra cách các lớp tương tác động với nhau.
Đối tượng (Object)	Các đối tượng và mối quan hệ của chúng trong thời gian chạy. Tổng quan về các trường hợp cụ thể của biểu đồ lớp tại một thời điểm cho một trường hợp cụ thể. Nó dựa trên biểu đồ lớp.
Composite structure (Cấu trúc gộp)	Mô tả cấu trúc bên trong của một thành phần bao gồm tất cả các lớp bên trong thành phần, giao diện của thành phần, v.v.
Component (thành phần)	Mô tả tất cả các thành phần trong hệ thống, mối quan hệ qua lại, tương tác của chúng và giao diện của hệ thống. Nó là một phác thảo về cấu trúc thành phần của các thành phần hoặc mô-đun.
Package (gói)	Mô tả cấu trúc và tổ chức gói. Nó bao gồm các lớp trong gói và các gói trong gói khác
Deployment (triển khai)	Mô tả phần cứng hệ thống, phần mềm và kết nối mạng cho máy tính phân tán. Nó bao gồm cấu hình máy chủ và kết nối mạng giữa các nút máy chủ trong cài đặt thế giới thực.

2. Biểu đồ hành vi (động)

Biểu đồ	Mô tả

Use case	Xuất phát từ các tình huống nghiên cứu ca sử dụng. Đây là tổng quan về các ca sử dụng, các tác nhân và các mối quan hệ giao tiếp của chúng để chứng minh cách hệ thống phản ứng với các yêu cầu từ người dùng bên ngoài. Nó được sử dụng để xác định các yêu cầu hệ thống.
Activity (hoạt động)	Phác thảo dữ liệu hoạt động và luồng điều khiển giữa các đối tượng liên quan. Hoạt động là một hành động cho một hoạt động hệ thống hoặc một quy trình nghiệp vụ, chẳng hạn như những hoạt động được nêu trong biểu đồ ca sử dụng. Nó cũng bao gồm các điểm quyết định và chủ đề của các quy trình hoạt động phức tạp. Nó mô tả cách các hoạt động được sắp xếp để đạt được chức năng cần thiết.
State-machine (máy trạng thái)	Mô tả vòng đời của các đối tượng bằng máy trạng thái hữu hạn. Biểu đồ bao gồm các trạng thái và sự chuyển đổi giữa các trạng thái. Quá trình chuyển đổi thường do các kích thích hoặc sự kiện bên ngoài gây ra. Chúng cũng có thể đại diện cho các chuyển động bên trong của đối tượng.
Sequence (trình tự)	Mô tả chuỗi thời gian của thông báo được chuyển giữa các đối tượng trong một dòng thời gian
Interaction overview (tổng quan tương tác)	Kết hợp các biểu đồ trình tự và hoạt động để cung cấp tổng quan về luồng điều khiển của hệ thống và quy trình nghiệp vụ.
Communication (giao tiếp)	Mô tả chuỗi thông báo truyền giữa các đối tượng trong hệ thống. Tương đương với biểu đồ tuần tự, ngoại trừ việc nó tập trung vào vai trò của đối tượng. Mỗi liên kết giao tiếp được liên kết với một số thứ tự trình tự cộng với các thông điệp đã chuyển.
Time sequence (trình tự thời gian)	Mô tả các thay đổi của thông báo trong trạng thái, điều kiện và sự kiện

Các biểu đồ mô tả cấu trúc bao gồm các biểu đồ lớp và đối tượng; biểu đồ thành phần, cấu trúc và gói; và các biểu đồ triển khai. Chúng tôi lần lượt thảo luận về từng vấn đề.

Biểu đồ lớp

Biểu đồ lớp cung cấp một cái nhìn tĩnh của hệ thống. Nó nắm bắt từ vựng của hệ thống được thiết kế. Đây là biểu đồ nền tảng của thiết kế hệ thống và cũng là biểu đồ UML được sử dụng thường xuyên nhất.

Biểu đồ lớp có thể được rút ra từ biểu đồ ca sử dụng hoặc từ phân tích văn bản của miền vấn đề đã cho. Biểu đồ lớp được tạo ra bởi các nhà phân tích và thiết kế hệ thống và sẽ được tinh chỉnh lặp đi lặp lại trong các giai đoạn tiếp theo trong vòng đời phát triển phần mềm.

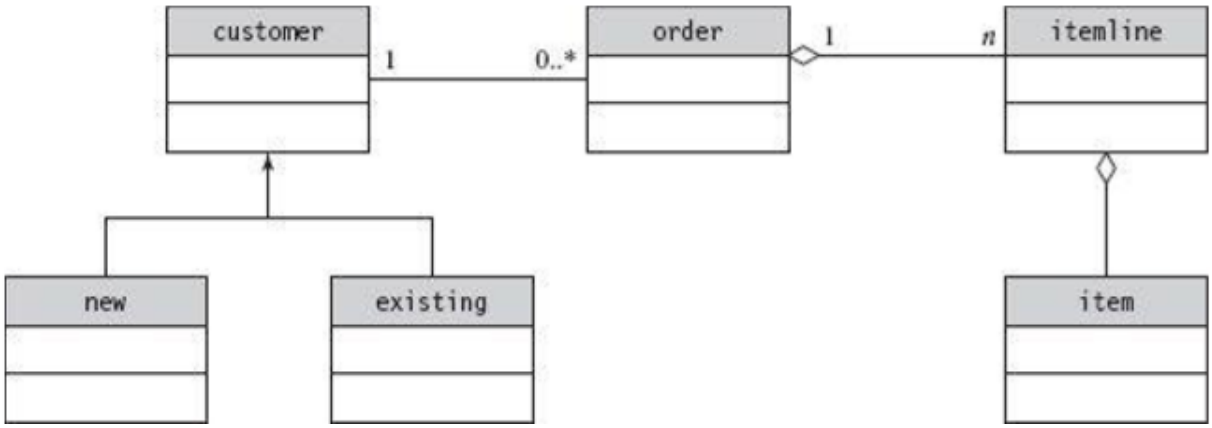
Biểu đồ lớp mô tả từng lớp riêng lẻ với kiểu, giao diện, thuộc tính và phương thức của nó. Khả năng truy cập của từng thuộc tính và hoạt động cũng có thể được chỉ định. Các truy cập phổ biến bao gồm riêng tư, công khai, được bảo vệ và mặc định.

Một phần quan trọng của biểu đồ lớp là giao diện của mỗi lớp. Một giao diện lớp cung cấp các hợp đồng hành vi mà lớp đó phải hỗ trợ.

Có ba mối quan hệ chính giữa các lớp: kế thừa, cộng gộp và kết hợp. Các mối quan hệ này có thể được biểu diễn bằng đồ thị trong một biểu đồ lớp. Đối với mỗi mối quan hệ, tính đa dạng giữa các lớp cũng có thể được ký hiệu. Các kiểu đa dạng điển hình bao gồm ánh xạ một-một, một-nhiều và nhiều-nhiều. Trong các ký hiệu đa dạng của UML, 1 là viết tắt của một trường hợp, 0 là không có trường hợp nào, 0..1 là viết tắt của 0 hoặc một trường hợp, và 1 .. * là viết tắt của ít nhất một trường hợp.

Hình 3.16 cho thấy một biểu đồ lớp cho một hệ thống xử lý đơn đặt hàng trực tuyến. Ở đây chúng tôi thấy tất cả các loại mối quan hệ giữa các lớp như kế thừa (biểu diễn bằng mũi tên tam giác rỗng), tập hợp (biểu diễn bằng mũi tên hình thoi rỗng) và kết hợp (các đường không có mũi tên). Các chỉ số đa dạng cũng được hiển thị. Nói chung, biểu đồ mô tả cấu trúc logic của hệ thống đặt hàng bao gồm sáu lớp. Lớp Customer là lớp cơ sở của New và Existing. Một Customer có thể đặt không hoặc nhiều Order. Mỗi Order bao gồm nhiều ItemLine, lần lượt chứa các item.

Biểu đồ lớp có thể được tinh chỉnh theo thời gian trong vòng đời phát triển phần mềm. Biểu đồ đối tượng và biểu đồ cấu trúc thành phần có thể được bắt nguồn trực tiếp từ một biểu đồ lớp. Các biểu đồ hành vi động khác như biểu đồ trình tự và biểu đồ giao tiếp (cộng tác) cũng dựa trên biểu đồ lớp.

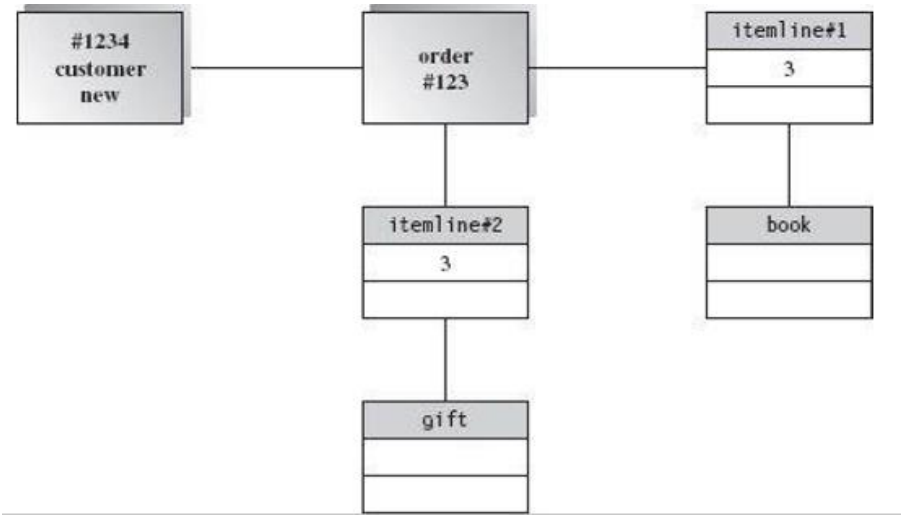


Hình 3.16: Biểu đồ lớp

Biểu đồ đối tượng

Đối tượng là thể hiện của các lớp. Biểu đồ đối tượng được sử dụng để mô tả một tập hợp con mẫu của các đối tượng trong hệ thống tại một thời điểm cụ thể. Biểu đồ này cho thấy một ảnh chụp nhanh về kết nối và tương tác của cá thể lớp. Nó có nguồn gốc từ biểu đồ lớp trước đó và là một ví dụ cụ thể về biểu đồ lớp trong thời gian chạy. Nhiều biểu đồ hành vi khác (biểu đồ tuần tự, biểu đồ giao tiếp và biểu đồ tương tác) có thể tham chiếu đến biểu đồ đối tượng.

Hình 3.17 cho thấy một biểu đồ đối tượng dựa trên biểu đồ lớp trong Hình 3.16. Mỗi hình hộp chữ nhật trong biểu đồ đại diện cho một đối tượng là một thể hiện của một số lớp. Biểu đồ cho chúng tôi biết rằng khách hàng có số nhận dạng # 1234 đã đặt hàng hai mặt hàng: book và gift.



Hình 3.17: Biểu đồ đối tượng

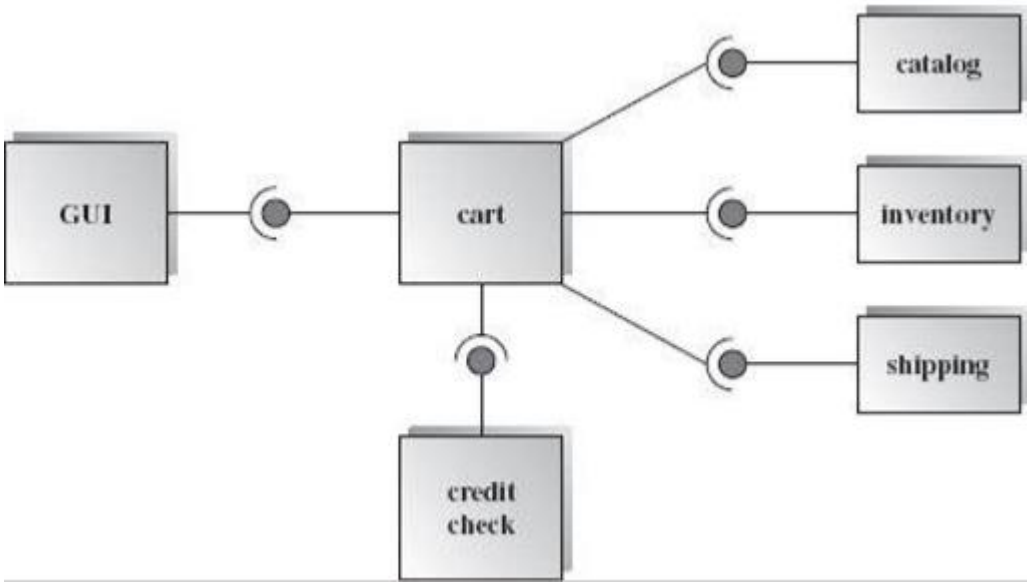
Biểu đồ cấu trúc tổng hợp

Biểu đồ cấu trúc tổng hợp được sử dụng để mô tả thành phần của các phần tử được kết nối với nhau hoặc sự cộng tác của các cá thể thời gian chạy. Có hai ký hiệu cơ bản trong một biểu đồ cấu trúc tổng hợp: cộng tác (được biểu diễn bằng cách sử dụng nhật thực nét đứt) và lớp có cấu trúc (được biểu diễn bằng hình hộp chữ nhật). Mỗi lớp cấu trúc có thể có một chú thích cho biết vai trò của nó trong sự cộng tác. Ví dụ, Hình 3.18 mô tả hai lớp tham gia vào một sự cộng tác OrderProcess. Lớp Khách hàng (Customer) đóng vai trò là "người mua" và Hệ thống xử lý đơn hàng (Order Processing System) đóng vai trò là "người bán". Lưu ý rằng OrderProcess không phải là một lớp cũng không phải là một đối tượng, nó là một sự cộng tác.

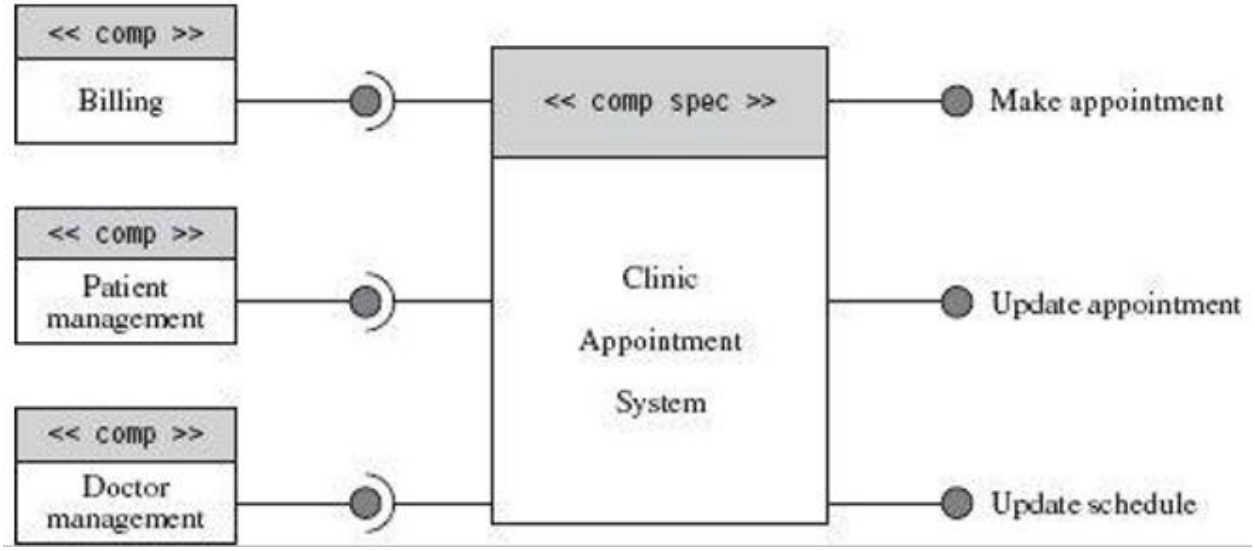
Hình 3.18: Cấu trúc tổng hợp

Biểu đồ thành phần

Một thành phần không phải là một lớp cũng không phải là một đối tượng. Thành phần là một khối xây dựng có thể triển khai, tái sử dụng được sử dụng trong thiết kế và phát triển phần mềm. Ví dụ: một thành phần JavaBean được triển khai trong một tệp jar, một thành phần EJB được triển khai trong một tệp. ear và thành phần NET được triển khai trong một tệp .dll. Mỗi thành phần có một giao diện để hiển thị các dịch vụ của nó và ẩn các triển khai của nó. Giao diện là hợp đồng giữa một thành phần có thể tái sử dụng và các máy khách của nó.



Hình 3.19: Biểu đồ thành phần



Hình 3.20: Một ví dụ khác của biểu đồ thành phần

UML 2.0 đã giới thiệu một ký hiệu mới cho các thành phần và kết nối của chúng. Hình dạng kẹo mút của một thành phần đại diện cho một giao diện được triển khai. Hình chiếc cốc đại diện cho giao diện bắt buộc và giao diện bắt buộc phải được cung cấp bởi một số thành phần khác. Trong biểu đồ thành phần, một số thành phần có thể tồn tại và có sẵn trong hệ thống hoặc bên ngoài. Các thành phần khác được thiết kế và phát triển bởi những người làm việc trong dự án.

Hình 3.19 cho thấy một biểu đồ thành phần cho một ứng dụng giỏ hàng. Thành phần giỏ hàng cung cấp các dịch vụ cho giao diện GUI front-end như ASP, JSP hoặc các

trang web PHP. Bản thân thành phần giỏ hàng có thể cần các dịch vụ từ các thành phần khác: danh mục, hàng tồn kho, vận chuyển và kiểm tra tín dụng.

Biểu đồ thành phần trong Hình 3.20 cho thấy bốn thành phần: thành phần phòng khám (*Clinic Appointment System*), thành phần thanh toán (*Billing*), thành phần bệnh nhân (*Patient management*) và thành phần bác sĩ (*Doctor management*). Thành phần phòng khám cung cấp các dịch vụ sau (dưới dạng giao diện): đặt lịch hẹn theo bệnh nhân (*make appointment*), cập nhật lịch hẹn theo bệnh nhân (*update appointment*), cập nhật lịch khám theo lịch của bác sĩ (*update schedule)*. Thành phần phòng khám cũng cần các dịch vụ từ thành phần thanh toán, thành phần bệnh nhân và thành phần bác sĩ.

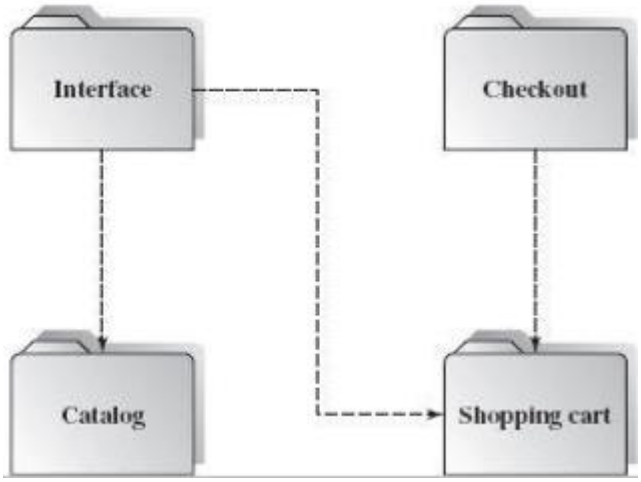
Biểu đồ gói

Một gói được đại diện bởi một thư mục theo thẻ cho biết vị trí của tất cả các lớp và gói con được bao gồm. Các gói đóng vai trò tương tự như một thư mục để nhóm các tệp trong hệ thống tệp; chúng cho phép tổ chức tất cả các lớp có liên quan chặt chẽ trong một "vùng chứa - container".

Ví dụ, các không gian tên trong. NET và các gói trong Java cung cấp các cấu trúc được định hình tốt cho khả năng truy cập lớp và các mối tương quan của lớp. Chúng tôi có thể tổ chức các lớp liên quan đến chức năng trong cùng một gói để các lớp này có thể truy cập lẫn nhau trong khả năng truy cập hoặc khả năng hiển thị mặc định. Chúng tôi cũng có thể tổ chức các gói liên quan trong cùng một gói mẹ để xây dựng một cấu trúc phân cấp gói và lớp giống như vậy. Thư viện lớp NET và API Java. Một lý do khác để sử dụng tổ chức gói là chia sẻ không gian tên; theo cách này, tất cả các lớp trong cùng một gói có một tên duy nhất nhưng chúng có thể có cùng tên trong các gói khác nhau (không gian tên).

Biểu đồ gói cho thấy mối quan hệ phụ thuộc giữa các gói trong đó sự thay đổi của một gói có thể dẫn đến những thay đổi trong các gói khác. Biểu đồ gói cũng có thể chỉ định nội dung của một gói, tức là các lớp cấu thành một gói và các mối quan hệ của chúng. Việc sử dụng biểu đồ gói để biểu diễn cấu trúc hệ thống có thể giúp giảm độ phức tạp của sự phụ thuộc và đơn giản hóa mối quan hệ giữa các nhóm lớp.

Hình 3.21 cho thấy một biểu đồ gói đơn giản trong đó gói thanh toán, chứa tất cả các lớp liên quan đến thanh toán, phụ thuộc vào các lớp được nhóm trong gói giỏ hàng. Tương tự với gói giao diện người dùng có tất cả các lớp trình bày GUI để hiển thị danh mục và giỏ hàng. Biểu đồ gói cũng mô tả mối quan hệ phụ thuộc của các đơn vị gói. Biểu đồ này thường được sử dụng để thiết kế kiến trúc phần mềm dựa trên thành phần.



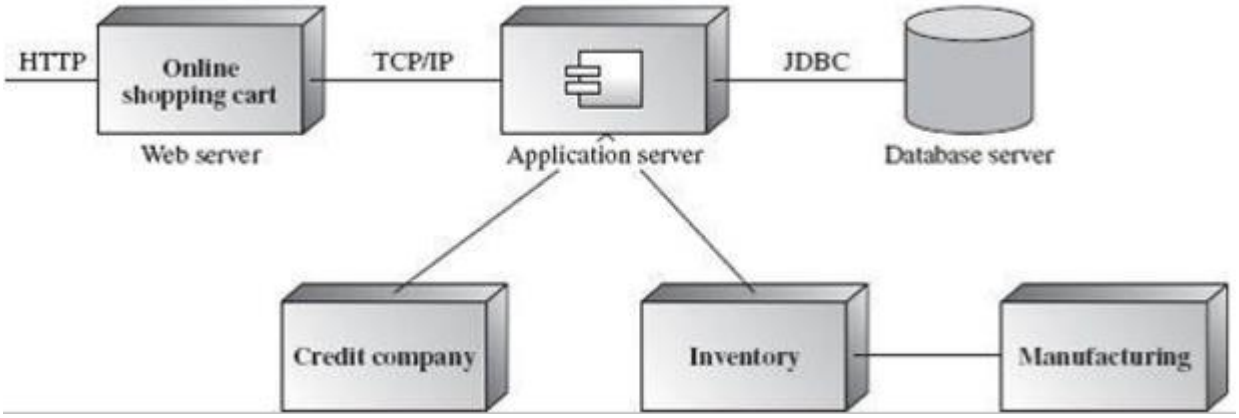
Hình 3.21: Biểu đồ gói

Biểu đồ triển khai

Biểu đồ triển khai mô tả cấu hình vật lý của hệ thống phần mềm được triển khai trên các nút máy chủ phần cứng và mạng giữa các nút (được định nghĩa là giao thức). Biểu đồ này được tạo ra trong giai đoạn sau của vòng đời phát triển phần mềm. Tất cả các thành phần trong hệ thống phải được triển khai trên các máy chủ để cung cấp dịch vụ thông qua giao thức mạng. Các biểu đồ thành phần là cơ sở cho các biểu đồ triển khai.

UML sử dụng một biểu tượng hình khối để đại diện cho một nút tài nguyên máy tính; tài nguyên đó có thể là một thiết bị phần cứng hoặc một hệ thống con phần mềm được triển khai. Ví dụ, máy chủ dữ liệu, máy chủ web và máy chủ ứng dụng có thể là các nút và được mô tả bằng các hình khối trong một biểu đồ triển khai. Liên kết giữa các nút là kết nối mạng được mô tả bởi giao thức mạng. Biểu đồ triển khai được sử dụng rộng rãi để mô hình hóa và thiết kế các hệ thống phần mềm phân tán.

Hình 3.22 cho thấy một biểu đồ triển khai trong đó một giỏ hàng được triển khai trong một máy chủ web, thành phần giao dịch kinh doanh được triển khai trong một máy chủ ứng dụng riêng biệt và cơ sở dữ liệu có sẵn trong một máy chủ dữ liệu. Các dịch vụ khác có sẵn từ ba thành phần được triển khai bởi các nhà cung cấp dịch vụ tương ứng.



Hình 3.22: Biểu đồ triển khai

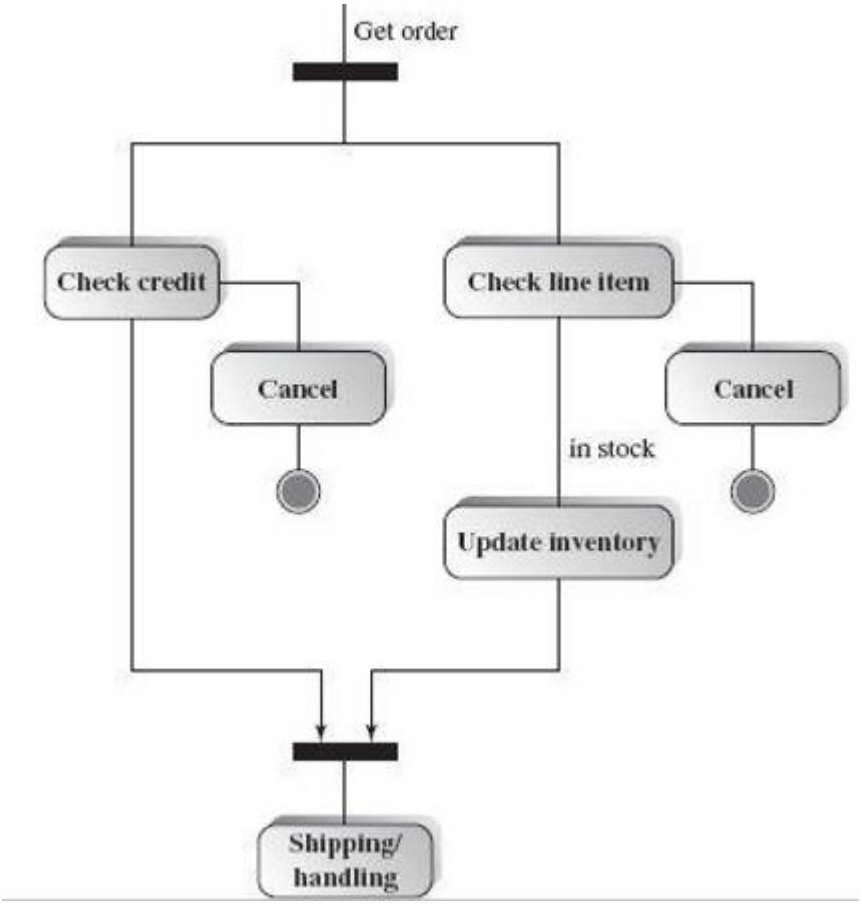
Biểu đồ hoạt động

Biểu đồ hoạt động được sử dụng để mô tả các quy trình nghiệp vụ phức tạp. Biểu đồ này thường liên quan đến quy trình làm việc phức tạp, ra quyết định, thực hiện đồng thời, xử lý ngoại lệ, kết thúc quy trình, v.v. Biểu đồ hoạt động là biểu đồ hướng quy trình làm việc mô tả các bước trong một quy trình duy nhất. Một biểu đồ hoạt động tương ứng với một quy trình nghiệp vụ. Có nhiều hoạt động trong một quy trình nghiệp vụ và biểu đồ này khám phá sự phụ thuộc của chúng trong quy trình.

Biểu đồ hoạt động UML sử dụng một hình chữ nhật tròn để biểu diễn một hoạt động. Mỗi biểu đồ hoạt động có một điểm bắt đầu và một hoặc nhiều điểm kết thúc. Một thời nhỏ đại diện cho một điểm quyết định trong biểu đồ. Biểu đồ hoạt động hỗ trợ xử lý song song bằng cách sử dụng một cặp thanh ngang màu đen để chỉ ra các hành động nối/rẽ nhánh tương ứng trong các lộ trình như vậy. Biểu đồ hoạt động UML cũng hỗ trợ giao tiếp giữa hai luồng đồng thời bằng cách gửi tín hiệu từ đường dẫn này sang đường dẫn khác. (Đây được gọi là một sự kiện và được ghi nhận là một cặp đa giác lồi.)

Biểu đồ hoạt động cung cấp một cái nhìn động chi tiết về một nhiệm vụ hoặc quy trình cụ thể trong một hệ thống để nhà phát triển phần mềm có thể dễ dàng nhận ra các yêu cầu thực hiện. Biểu đồ này là cơ sở cho một biểu đồ truyền thông và các biểu đồ tương tác động khác.

Hình 3.23cho thấy một nhóm các hoạt động trong hệ thống xử lý đơn đặt hàng. Thanh màu đen đầu tiên chia (fork) hai hoạt động đồng thời có thể được thực hiện song song. Hoạt động vận chuyển/xếp dỡ (*shipping handling*) sẽ không bắt đầu cho đến khi cả hai hoàn thành và kết hợp với nhau (được biểu thị bằng thanh thứ hai ở dưới cùng).



Hình 3.23: Biểu đồ hoạt động

Biểu đồ máy trạng thái

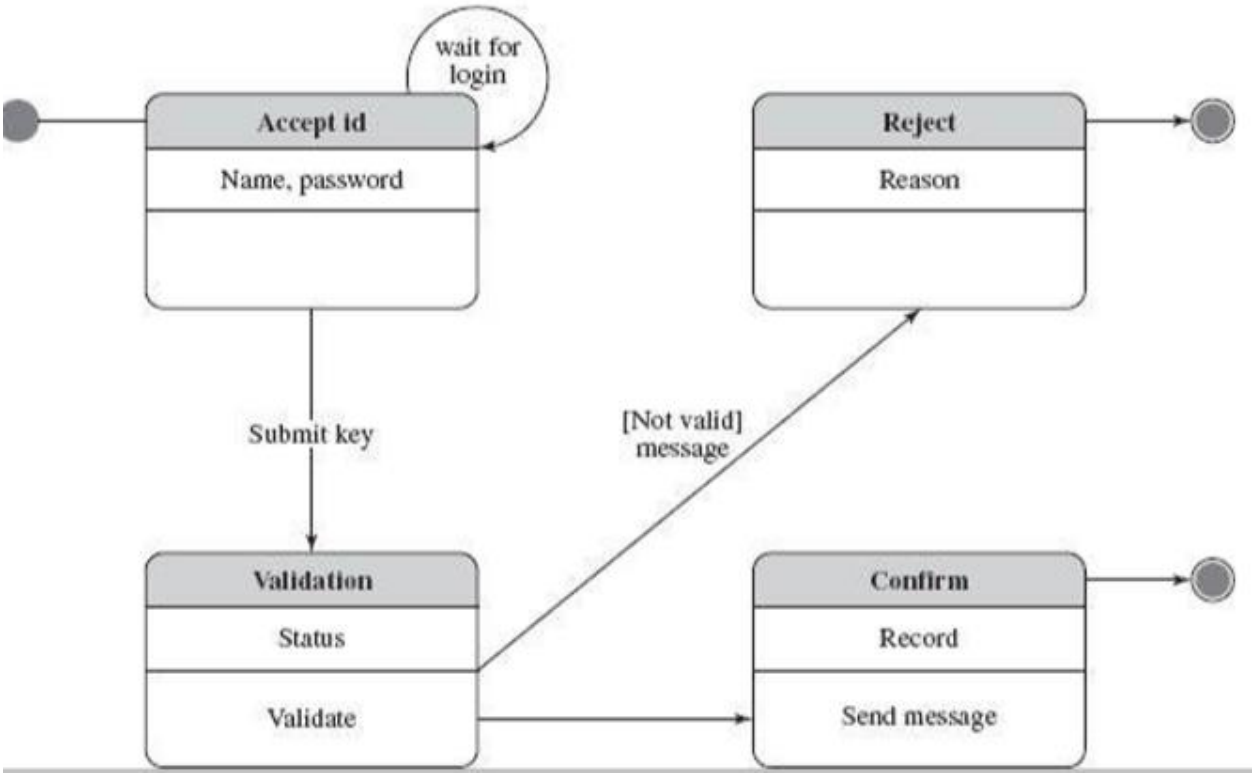
Biểu đồ trạng thái, được gọi là biểu đồ trạng thái trong UML 1.x, được sử dụng rộng rãi cho các hệ thống nhúng và phân tích và thiết kế phần mềm thiết bị. Nó là một biểu đồ hướng sự kiện, trong đó các phần tử của hệ thống thay đổi trạng thái của chúng để đáp ứng với các kích thích bên ngoài hoặc bên trong (chẳng hạn như các sự kiện thời gian hoặc các sự kiện hệ thống khác). Những biểu đồ này là lý tưởng để xác định hành vi bên trong của các đối tượng.

Trong biểu đồ máy trạng thái, trạng thái là một hình chữ nhật tròn với ba phần nhỏ: tên trạng thái, các biến trạng thái và các hoạt động của trạng thái. Trạng thái là một tình huống trong đó một đối tượng đáp ứng các điều kiện, thực hiện các hành động và chờ đợi một sự kiện mới. Khi một sự kiện mới diễn ra ở trạng thái hiện tại, máy sẽ thực hiện các hành động được chỉ định và sau đó sẽ chuyển sang trạng thái mới (trạng thái tiếp theo).

Một trạng thái hỗn hợp phức tạp có thể có một biểu đồ trạng thái cấp dưới. Các trạng thái con ở trạng thái hỗn hợp có thể được chuyển đổi từ trạng thái này sang trạng thái tiếp theo, tuần tự hoặc đồng thời.

Mỗi biểu đồ máy trạng thái có một điểm bắt đầu trong một vòng tròn màu đen đặc và có một hoặc nhiều điểm cuối, điểm cuối được biểu thị bằng các hình tròn. Các liên kết chuyển tiếp giữa các trạng thái là các đường liền nét với các đầu mũi tên để chỉ hướng. Biểu đồ trạng thái giúp nhà phát triển phần mềm hiểu cách hệ thống phản hồi và xử lý các sự kiện bên ngoài và điều kiện kích hoạt sự kiện tương ứng.

Hình 3.24 cho thấy một biểu đồ trạng thái máy mô tả một quá trình đăng nhập. Ban đầu, máy trạng thái thực hiện một vòng lặp bận để chờ đăng nhập của người dùng, và sau đó cặp tên người dùng / mật khẩu được xác minh. Nếu cặp khớp với hồ sơ hệ thống, đăng nhập được xác nhận; nếu không đăng nhập bị từ chối.



Hình 3.24: Biểu đồ máy trạng thái

Biểu đồ trình tự

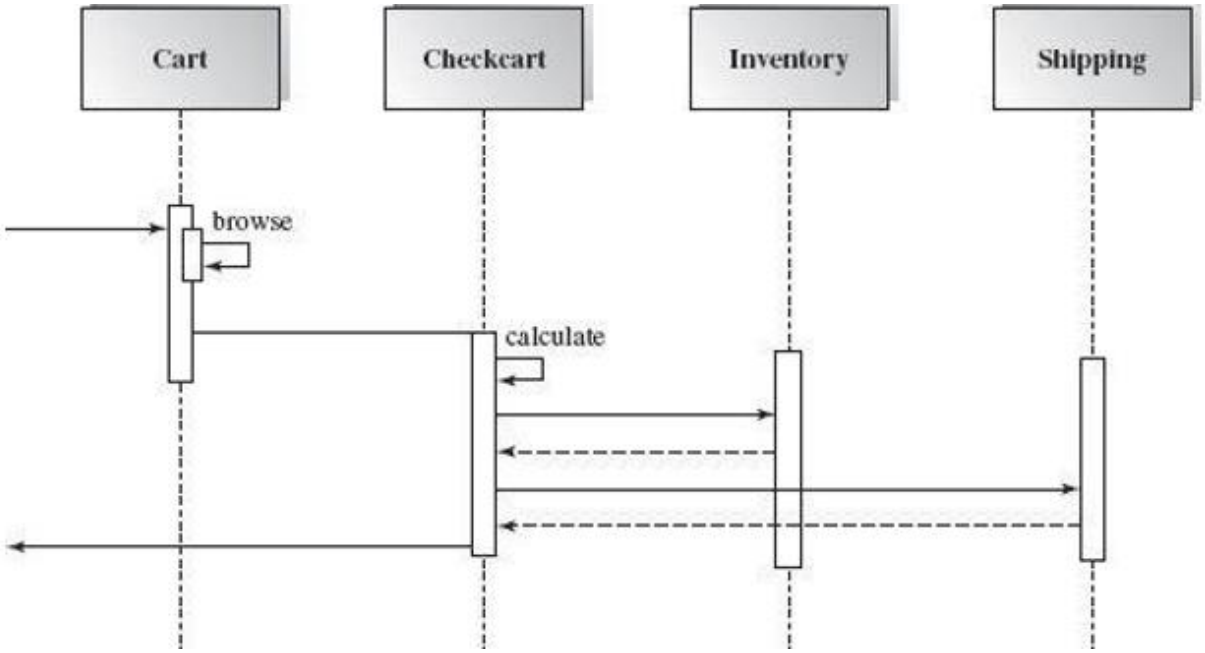
Biểu đồ trình tự là một trong những biểu đồ UML quan trọng nhất và được sử dụng rộng rãi để phân tích và thiết kế hệ thống phần mềm. Nó là một biểu đồ tương tác

định hướng thời gian hiển thị trình tự thời gian của thông điệp giữa các đối tượng. Thông thường, một biểu đồ trình tự tương ứng với một ca sử dụng.

Mỗi đối tượng tham gia trong biểu đồ này có một mốc thời gian dọc cho vòng đời của nó. Time tiến lên cùng với dòng thời gian đi xuống. Mỗi dòng thời gian dọc có một số ô chữ nhật hẹp (gọi là kích hoạt) đại diện cho trạng thái kích hoạt đối tượng mà nó nhận hoặc gửi thông điệp. Mỗi hộp kích hoạt cũng có thể có một liên kết được định hướng tự đệ quy được trỏ về chính nó, chỉ ra rằng đối tượng chuyển thông điệp cho chính nó. Một kích hoạt cũng có thể phân nhánh hoặc phân nhánh nhiều đường sống riêng biệt cho các điều kiện kịch bản nếu lựa chọn; cuối cùng tất cả các dòng được chia sẻ kết hợp với nhau.

Truyền thông điệp giữa các đối tượng được biểu diễn bằng một liên kết mũi tên ngang từ nguồn đến đích. Một dòng thông báo chuyển đơn giản, được biểu diễn bằng một đường liền nét có đầu mũi tên, chuyển quyền điều khiển từ đối tượng này sang đối tượng khác. Một đối tượng có thể gửi một thông điệp đồng bộ đến một đối tượng khác bằng một dòng có đầu mũi tên đầy đủ. Một thông điệp đồng bộ có nghĩa là người gửi phải đợi phản hồi từ đối tượng đích trước khi nó có thể tiến lên trong dòng thời gian. Một đối tượng cũng có thể gửi một thông điệp không đồng bộ đến một đối tượng khác, được biểu thị bằng một dòng có nửa đầu mũi tên. Người gửi một thông điệp không đồng bộ có thể tiếp tục công việc của mình theo dòng thời gian mà không cần đợi thông điệp trả lại từ đối tượng đích.

Hình 2.25 cho thấy một biểu đồ trình tự đơn giản cho mua sắm trực tuyến. Chuỗi trao đổi thông điệp bắt đầu từ đối tượng giỏ hàng (Cart). Sau khi duyệt thông điệp, đối tượng giỏ hàng (Cart) sẽ gửi thông điệp đến đối tượng kiểm tra giỏ hàng (Checkcart) để thanh toán. Checkcart thực hiện một phép tính (calculate - tức là thông qua cách tự gửi thông điệp) và sau đó gửi thông điệp đến kho (Inventory). Thông báo từ checkcart đến kho là một thông báo đồng bộ, vì checkcart phải đợi thông báo phản hồi (được biểu diễn bằng một đường mũi tên chấm chấm). Sau đó checkcart liên hệ vận chuyển (shipping) và cuối cùng gửi lại thông điệp cho tác nhân đã khởi tạo ca sử dụng.

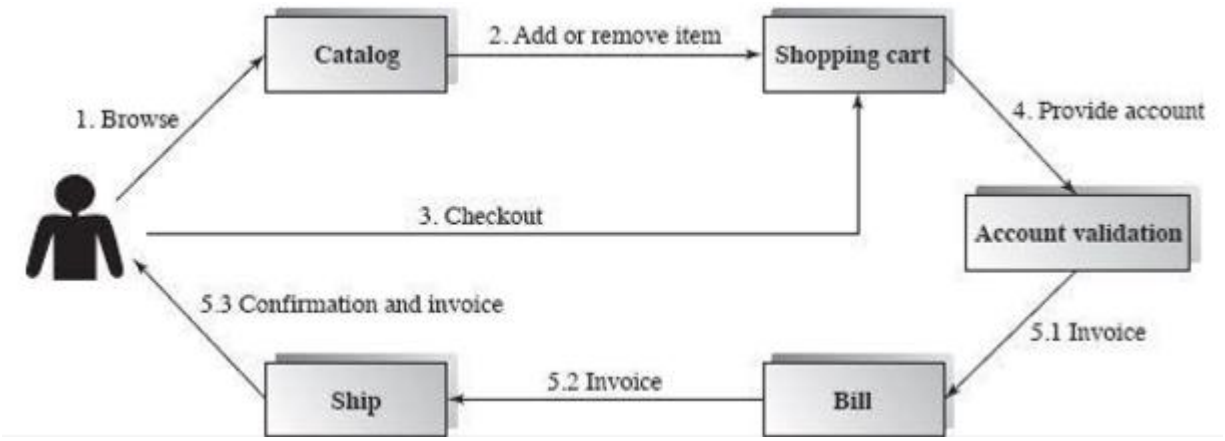


Hình 3.25: Biểu đồ trình tự

Biểu đồ giao tiếp hoặc cộng tác

Biểu đồ truyền thông UML, được gọi là biểu đồ cộng tác trong UML 1. x, là một biểu đồ hướng thông điệp mô tả tất cả các trình tự truyền thông điệp, điều khiển luồng, điều phối đối tượng, v.v., giữa các đối tượng tham gia vào các trường hợp sử dụng nhất định. Nó tóm tắt cách các đối tượng trong hệ thống nhận và gửi thông điệp. Nó là một phần mở rộng của biểu đồ đối tượng tĩnh trong đó các liên kết giữa các đối tượng thể hiện các mối quan hệ liên kết. Phía trên các liên kết trong một biểu đồ liên lạc là các thông điệp được đánh số, cho biết thứ tự mà chúng được gửi hoặc nhận. Các thông báo yêu cầu người nhận thực hiện một thao tác với các đối số được chỉ định. Mọi biểu đồ giao tiếp đều tương đương với một biểu đồ tuần tự, tức là một biểu đồ giao tiếp có thể được chuyển đổi thành một biểu đồ trình tự tương đương và ngược lại. Hai loại biểu đồ này lần lượt cung cấp một cái nhìn hướng thông điệp và hướng thời gian

Hình 3.26 cho thấy một ví dụ về biểu đồ truyền thông. Nó tương đương với biểu đồ trình tự trong Hình 3.25 ngoại trừ các tên thông báo được đưa ra.



Hình 3.26: Biểu đồ cộng tác

CHƯƠNG 4: LẬP TRÌNH VÀ XÂY DỰNG PHẦN MỀM

Nội dung của chương:

4.1. Tổng quan về cài đặt phần mềm

4.2 Các phương pháp luận lập trình

4.3. Sử dụng các công cụ và môi trường phát triển

Mục tiêu của chương:

- Biết các phương thức lập trình mệnh lệnh và khai báo, ưu nhược điểm của phương thức lập trình mệnh lệnh và khai báo
- Hiểu một số phương pháp lập trình, từ cổ điển đến hiện đại, từ đó có thể lựa chọn phương pháp phù hợp cho 1 số bài toán cụ thể
- Vận dụng 1 số cộng cụ để tổ chức, quản lý và chia sẻ mã nguồn

4.1. Tổng quan về cài đặt phần mềm

Kỹ nghệ phần mềm bao gồm tất cả các hoạt động liên quan đến phát triển phần mềm từ các yêu cầu ban đầu của hệ thống cho đến bảo trì và quả nlý hệ thống được triển khai. Tất nhiên, một giai đoạn quan trọng của quá trình này là cài đặt hệ thống, nơi ban tạo ra các bản thực thi của phần mềm. Việc thực hiện này có thể liên quan đến việc sử dụng các ngôn ngữ lập trình để tạo ra hệ thống.

Trong nội dung chương này sẽ đề cập tới các nguyên tắc lập trình, và một số kinh nghiệm trong lập trình, tập trung vào các vấn đề về làm thế nào để có thể lập trình tốt cho các hệ thống mà không đi vào cụ thể một ngôn ngữ lập trình nào cả

4.2. Các phương pháp luận lập trình

Phương pháp luận lập trình được biểu diễn bởi mô hình lập trình. Một mô hình lập trình thể hiện một quan điểm/một cách tiếp cận trong lập trình. Mỗi mô hình hỗ trợ một tập các khái niệm, các kỹ thuật và các nguyên tắc thiết kế. Các mô hình lập trình khác nhau có những mức độ diễn đạt khác nhau, các kỹ thuật lập trình khác nhau và các cách lập luận khác nhau về chúng.

Mọi mô hình lập trình đều có vị thế của nó, được thiết kế và sử dụng trong hoàn cảnh riêng. Một nguyên tắc quan trọng ta cần nhận biết đó là: Cần kết hợp giữa các mô hình để giải quyết triệt để một hệ thống ứng dụng phức tạp.

Các mô hình lập trình được liệt kê trong bảng sau:

Phương thức lập trình

Mệnh lệnh	1	Lập trình hướng thiết bị - Gear Oriented Programming
	2	Lập trình hướng công tắc- Switch Oriented Programming
	3	Lập trình hướng thủ tục – Procedural/structured Programming
	4	Lập trình hướng đối tượng – Object Oriented Programming
	5	Lập trình hướng lát cắt – Aspect Oriented Programming
	6	Lập trình hướng cấu phần – Component Oriented Programming
	7	Lập trình hướng dịch vụ - Service Oriented Programming
	8	Điện toán đám mây – Clound Computing
Khai báo	1	Lập trình logic – Logic Programming
	2	Lập trình hàm – Functional Programming
	3	Lập trình CSDL – Database Programming

+ Với phương thức lập trình mệnh lệnh:

- Người lập trình phải tìm cách diễn đạt được thuật toán, Chỉ ra cách thức làm thế nào để giải quyết bài toán đã cho.
- *Ưu điểm:* Hiệu quả trong lập trình, vì người lập trình có thể tác động trực tiếp vào phần cứng
- *Hạn chế*: Chương trình không có khả năng suy đoán, không có trí tuệ.

+ Với phương thức lập trình khai báo:

- Người lập trình xây dựng cơ sở tri thức, các yêu cầu tính toán, truy vấn dựa trên các khai báo để giải quyết bài toán
- *Ưu điểm*: Chương trình có khả năng suy diễn
- *Hạn chế*: Khó cài đặt và vận hành hơn so với chương trình mệnh lệnh.

4.2.1 Lập trình tuyến tính

Lập trình tuyến tính là một phương pháp, kỹ thuật lập trình truyền thống. Trong lập trình tuyến tính, toàn bộ chương trình chỉ là một đơn thể duy nhất, các lệnh được thực hiện tuần tự theo thứ tự xuất hiện trong chương trình

Đặc trưng:

- Đơn giản: chương trình được tiến hành theo lối tuần tự, không phức tạp

- Đơn luồng: chỉ có một luồng công việc duy nhất, và các công việc được thực hiện tuần tự trong luồng đó

Ưu nhược điểm:

- Ưu điểm: do tính đơn giản, lập trình tuyến tính được ứng dụng cho các chương trình đơn giản và có ưu điểm dễ hiểu
- Nhược điểm: với các ứng dụng phức tạp, người ta không thể dùng lập trình tuyến tính để giải quyết.

Trong ngôn ngữ C, lập trình theo kiểu tuyến tính sẽ chỉ có một hàm main.

Ví dụ: viết ct nhập họ tên sv, đlt, đth và tính đtb của sv.

```
#include <stdio.h>
void main()
{
        char hoten[30];
        float dlt,dth,dtb;
        printf("Nhap ho ten:"); gets(hoten);
        printf("Nhap dlt:"); scanf("%f",&dlt);
        printf("Nhap dth:"); scanf("%f",&dth);
        dtb=(dlt+dth)/2;
        printf("\nHo ten: %s",hoten);
        printf("\nDlt:%.2f",dlt);
        printf("\nDth:%.2f",dth);
        printf("\nDtb:%.2f",dtb);
}
```

4.2.2 Lập trình có cấu trúc

Lập trình cấu trúc là nguyên lý chủ đạo trong công nghệ phần mềm. Theo nguyên lý này ta sử dụng rộng rãi khái niệm trừu tượng hóa nhằm mục đích phân rã bài toán thành những bài toán nhỏ hơn để dễ dàng triển khai và đảm bảo tính đúng đắn của chương trình.

- Lập trình hướng cấu trúc hay còn gọi là lập trình hướng thủ tục (Procedure Oriented Programming -- POP): là một kỹ thuật lập trình truyền thống, trong đó chương trình được chia thành các hàm (chương trình con)

- Mỗi chương trình còn có thể được chia ra nhiều chương trình con khác để đơn giản hóa công việc của chúng (quá trình làm mịn)

Ví dụ chương trình nhập và hiển thị thông tin người dùng sẽ chia thành hai chương trình con là chương trình nhập và xuất, nếu việc nhập thông tin phức tạp thì chương trình nhập thông tin có thể chia ra nhiều chương trình con khác nhau...

Ví dụ: Triển khai chương trình phân số đã đề cập ở phần 2.3.1

Phương pháp này ra đời cùng với sự ra đời của các ngôn ngữ lập trình bậc cao như: Pascal, C, Basic, ... hỗ trợ cho phương pháp này. Tính cấu trúc của phương pháp và của ngôn ngữ thể hiện ở cấu trúc điều khiển, cấu trúc dữ liệu và cấu trúc của chương trình...

a. Cấu trúc lệnh, lệnh có cấu trúc

(1) Cấu trúc lệnh (Cấu trúc điều khiển)

- Mỗi module chương trình máy tính về bản chất là một bản mã hóa thuật toán. Đến lượt mình, thuật toán được coi là một dãy hữu hạn các thao tác trên các đối tượng nhằm thu được một số kết quả sau một số hữu hạn bước
- Các thao tác trong một chương trình cụ thể được điều khiển bởi các lệnh hay các cấu trúc điều khiển, còn các đối tượng chịu sự tác động của thao tác thì được mô tả và được kiến trúc thông qua các Cấu trúc dữ liệu.

(2) Lệnh cấu trúc

Lệnh có cấu trúc là các cấu trúc điều khiển cho phép chứa các cấu trúc điều khiển khác bên trong nó. Khi tìm hiểu một cấu trúc điều khiển cần xác định rõ vị trí được phép đặt một cấu trúc điều khiển khác trong nó.

b. Cấu trúc dữ liệu (Data Structures)

Chúng ta có thể đặt ra câu hỏi: Vì sao trong tin học lại xuất hiện CTDL & các cấu trúc điều khiển?

Tin học được xây dựng trên nền tảng của khái niệm toán học về thuật toán. Đến lượt mình, thuật toán quan tâm đến thao tác và các đối tượng. Có thể nêu vắn tắt những vấn đề nghiên cứu của lý thuyết thuật toán như sau:

1. Các thuật toán (máy giải, máy trừu tượng) xử lý những đối tượng nào (CTDL)?
2. Cách xử lý ra sao? (Các thao tác & các cấu trúc điều khiển thao tác)
3. Độ phức tạp của biểu diễn dữ liệu & thao tác là bao nhiêu (tốn bao nhiêu miền nhớ, miền nháp và kết quả trung gian, thời gian xử lý).

Như vậy:

Cấu trúc điều khiển được nẩy sinh từ nhu cầu diễn đạt thuật toán. Người ta có thể đặt ra nhiều cấu trúc điều khiển, tuy nhiên những người sáng tạo ra những ngôn ngữ lập trình thường chọn 3 cấu trúc điều khiển trong sáng, đơn giản là cấu trúc lặp, tuần tự và rẽ nhánh.

Cấu trúc dữ liệu đa dạng hơn cấu trúc điều khiển vì chúng nẩy sinh do mô phỏng các đối tượng thực tế. Ví dụ cây, đồ thị, tập hợp, ...

Tin học tiếp thu một số CTDL "kinh điển" của toán học như: Cây, đồ thị, ma trận...và cũng sáng tạo ra các CTDL hết sức độc đáo như: Ngăn xếp, hàng đợi, danh sách, tệp tin, ... Tệp tin và ngăn xếp là hai cấu trúc không thể thiếu được khi tổ chức chương trình dịch và thực hiện chương trình đã dịch. Khi giải một bài toán cụ thể, khéo léo chọn cấu trúc dữ liệu sẽ giúp ta diễn đạt thuật toán một cách thuận lợi. Đó chính là nội dung của công thức nổi tiếng do Wirth – tác giả của ngôn ngữ lập trình Pascal đề xuất:

Program = Algorithms + Data Structures

Ý nghĩa chính của công thức này thể hiện ở chỗ: Thuật toán thực hiện các thao tác trên CTDL. CTDL và thuật toán phải tương thích với nhau

4.2.3 Lập trình hướng đối tượng

Lập trình hướng đối tượng (object-oriented programming - OPP) hay chi tiết hơn là Lập trình định hướng đối tượng, là phương pháp lập trình lấy đối tượng làm nền tảng để xây dựng giải thuật, xây dựng chương trình. OPP là một phương pháp mới trên bước đường tiến hóa của việc lập trình, nhằm làm cho chương trình trở nên linh hoạt, tin cậy và dễ phát triển; là kĩ thuật lập trình hỗ trợ công nghệ đối tượng. OOP được xem là giúp tăng năng suất, đơn giản hóa độ phức tạp khi bảo trì cũng như mở rộng phần mềm bằng cách cho phép lập trình viên tập trung vào các đối tượng phần mềm ở bậc cao hơn. Ngoài ra, nhiều người còn cho rằng OOP dễ tiếp thu hơn cho những người mới học về lập trình hơn là các phương pháp trước đó. Một cách giản lược, đây là khái niệm và là một nỗ lực nhằm giảm nhẹ các thao tác viết mã cho người lập trình, cho phép họ tạo ra các ứng dụng mà các yếu tố bên ngoài có thể tương tác với các chương trình đó giống như là tương tác với các đối tượng vật lý. Những đối tượng trong một ngôn ngữ OOP là các kết hợp giữa mã và dữ liệu mà chúng được nhìn nhận như là một đơn vị duy nhất. Mỗi đối tượng có một tên riêng biệt và tất cả các tham chiếu đến đối tượng đó được tiến hành qua tên của nó. Như vậy, mỗi đối tượng có khả năng nhận vào các thông báo, xử lý dữ liệu (bên

trong của nó), và gửi ra hay trả lời đến các đối tượng khác hay đến môi trường.

- Nguyên tắc chủ yếu để làm chủ độ phức tạp của chương trình là triển khai chương trình theo mức và nguyên tắc này được sử dụng rất rộng rãi
- Khi mô tả các kiểu dữ liệu phù hợp với các đối tượng của thế giới thực, người ta rút ra 3 nhận xét hết sức quan trọng sau đây:

Nhận xét 1

Không thể tách rời CTDL với các thao tác trên cấu trúc ấy. Dữ liệu được phát sinh để chịu sự biến đổi, để được xử lý, nghĩa là chúng là đối tượng của thao tác. Khi mô tả một đối tượng là liệt kê các tính chất của đối tượng. Các tính chất này được phân làm 2 loại:

Nhóm 1: Khuôn dạng, sự tổ chức, kiến trúc của đối tượng, tên, kích thước, xuất xứ, ...

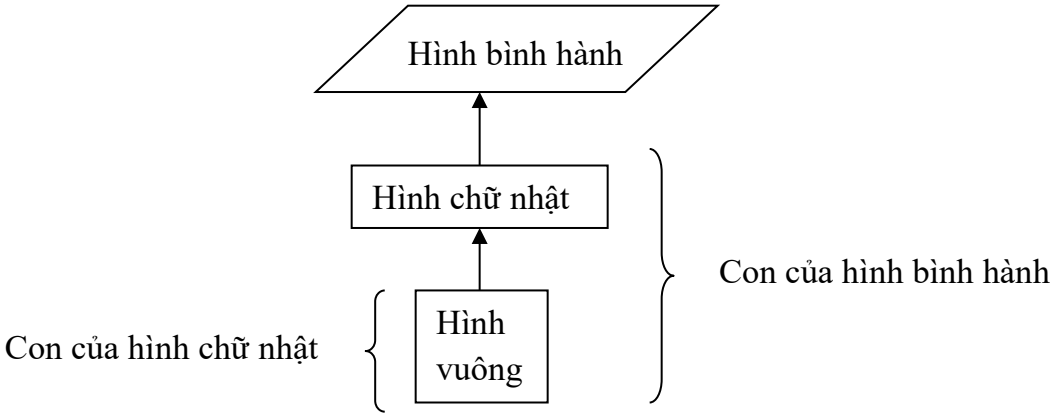
Nhóm 2: Sự vận động của đối tượng và các tương tác với các đối tượng khác.

Nhận xét 2

Nhiều kiểu dữ liệu thực ra chỉ là sự kế thừa và mở rộng từ các kiểu dữ liệu trước đó. Ví dụ:

- Kiểu hình chữ nhật (HCN) $\overset{\Delta}{=}$ là hình bình hành có một góc vuông
- Kiểu hình vuông (HV) $\overset{\Delta}{=}$ Là hình chữ nhật có 2 cạnh liên tiếp = nhau
- ...

=> HCN và HV là những kiểu con của hình bình hành. Trong đó, hình vuông lại là kiểu con của hình chữ nhật.



Trật tự phân cấp cha – con cho các kiểu, ngoài thuận lợi tiết kiệm cho việc mô tả, ta còn thấy một ưu thế tuyệt với đó là: Hình ảnh phân cấp làm giảm đáng kể tính phức tạp của vấn đề so với những mô tả lộn xộn, bình đẳng.

Nhận xét 3:

Sự xuất hiện và mất đi của một đối tượng cần được mô tả một cách tường minh. Sẽ rất khó khăn cho việc quản lý các đối tượng nếu ta không biết trước được sự tồn tại của chúng

=> 3 nhận xét trên đã được thể hiện trong phương pháp lập trình mới – Lập trình hướng đối tượng.

Phương pháp hướng đối tượng là sự kế tục và phát triển tự nhiên của phương pháp lập trình có cấu trúc. Sự mở rộng được thể hiện như sau:

(1) Mô tả cấu trúc và các thao tác cho các phần tử của một kiểu phải được thực hiện tường minh vào lúc mô tả kiểu

(2) Các đối tượng con (lớp con) của một đối tượng (lớp cha) cho trước được mô tả bằng cách chỉ rõ cha của nó và các thao tác, các thuộc tính riêng của nó

(3) Mỗi đối tượng cần được tạo lập và hủy bỏ một cách tường minh. Việc truy cập tới các đối tượng & các thành phần của chúng được thực hiện giống như truy cập đến các biến bản ghi và các trường của nó.

4.2.4 Lập trình hướng cấu phần

Năm 1975 Freed Brooks, một nhà quản lý dự án IBM, viết cuốn The Mythical Man-month. Brooks viết một chương với tiêu đề là "No Silver Bullet" giải thích rằng các hệ thống phần mềm là phức tạp. Ông dự đoán sẽ không có kỹ thuật nào là duy nhất – no silver bullet – mà nó có thể cải thiện năng suất theo danh sách yêu cầu cho mọi hệ thống. Brooks trình bày 2 phương pháp khả thi giúp giảm mức độ phức tạp của phần mềm đó là "Buy before Build" và "Reuse before Buy". Các khái niệm mấu chốt được đưa ra nhằm giúp giảm được chi phí trong công nghệ phát triển phần mềm.

IBM tiên phong mở ra lối nghiên cứu về Mô hình đối tượng hệ thống ngay đầu những năm 1990. Một số các đóng góp được ứng dụng trong phần mềm cấu phần đó là OLE và COM. Mô hình cấu phần phần mềm vẫn tiếp tục thu được những thành quả đáng kể.

Theo Paul Allen thì hiện nay có đến hơn 70% hệ thống phần mềm mới được phát triển dựa trên cơ sở cấu phần. Các cấu phần thường được phát triển theo hướng đối tượng

và được viết bằng các ngôn ngữ khác nhau, chạy trên các môi trường khác nhau, có thể phân tán khắp nơi và người phát triển phần mềm mới không được cung cấp các mã nguồn.

Thực tế cho thấy phương pháp Phát triển phần mềm theo cấu phần đã làm giảm chi phí của dự án phát triển phần mềm. So với công nghệ truyền thống chuẩn, công nghệ phần mềm trên cơ sở cấu phần quan tâm đến cách xây dựng phần mềm nhiều hơn. Thông qua việc sử dụng lại các cấu phần, vòng đời phát triển phần mềm được rút ngắn lại, đồng thời tăng tính mềm dẻo khi sử dụng và bảo trì phần mềm. Hơn nữa, phát triển phần mềm có khả năng làm tăng chất lượng phần mềm

Lập trình hướng cấu phần là kiểu lập trình có xu hướng chia hệ thống phần mềm thành những thành phần giữ các chức năng khác nhau (mỗi thành phần này được gọi là một bộ phận hợp thành) mà khi kết hợp lại ta thu được một hệ thống phần mềm hoàn chỉnh.

Với COP, chương trình được xây dựng bằng cách lắp ráp các thành phần phần mềm có thể sử dụng lại, các khối tự chứa mã máy (hay còn gọi là các khối thành phần thực hiện). Các thành phần này gồm các thành phần giao diện, các kết nối. COP nẩy sinh xuất phát từ thực tế rằng mọi thứ có cấu trúc đều được tạo nên từ các thành phần khác. Điển hình như trong nền công nghiệp tự động, các hệ thống được cấu tạo từ các thành phần. Ví dụ, để phát triển một chiếc ô tô là rất phức tạp.

Ô tô = {Các thành phần được ghép nối};

Các thành phần cấu tạo nên ô tô là thuộc nhiều loại khác nhau, kích cỡ khác nhau, chức năng khác nhau, được sản xuất bởi các nhà sản xuất khác nhau. Các thành phần này giới hạn từ các ốc vít rất nhỏ đến các hệ thống con phức tạp hơn như các động cơ, các bộ truyền phát nhanh, ...

Trong công nghiệp phần mềm. Sản phẩm vẫn làm bằng tay là chủ yếu. Điều đó dẫn đến tính năng của sản phẩm thấp, chất lượng không đảm bảo, khó tái sử dụng, ...

Trong công nghệ phần cứng. Mọi sản phẩm cũng được tạo ra dựa trên phương pháp hướng cấu phần. Thực tế cho thấy nền công nghiệp này phát triển rất nhanh, thu được nhiều lợi nhuận. Đây chính là lý do tại sao COP lại quan trọng.

COP sử dụng nhiều khái niệm của OOP nhưng hai phương pháp này là độc lập nhau. COP phát triển phần mềm bằng cách lắp ráp các thành phần trong khi OOP nhấn mạnh đến các lớp và các đối tượng. COP nhấn mạnh giao diện và kết cấu, trong khi OOP nhấn mạnh về cài đặt viết mã. COP không cần biết bất cứ kiến thức nào về cách thức

một thành phần cài đặt giao diện của chúng, nó xem thành phần như một hộp đen (không bị ảnh hưởng bởi sự thay đổi trong cài đặt của giao diện thành phần), chỉ quan tâm đến đầu vào, đầu ra, chức năng nhiệm vụ của hộp đen đó. Ví dụ: Ốc vít dùng để làm gì, làm thế nào để sử dụng nó mà không cần biết nó được làm như thế nào, sử dụng công cụ gì. COP lắp ráp các thành phần thông qua giao diện của các thành phần này.

Có một số lý do vì sao COP là quan trọng. COP cung cấp một mức độ trừu tượng cao hơn. Có một số lượng ngày càng lớn của các thư viện cấu phần tái sử dụng mà hỗ trợ cho việc phát triển ứng dụng cho lĩnh vực khác nhau.

Có ba mục tiêu chính của COP: chinh phục sự phức tạp, quản lý thay đổi, và tái sử dụng.

4.2.5 Lập trình Cơ sở dữ liệu

CSDL là tập hợp các dữ liệu có cấu trúc và liên quan với nhau được lưu trữ trên máy tính, được nhiều người sử dụng và được tổ chức theo mô hình.

Ví dụ :

Hồ sơ sinh viên là một ví dụ về CSDL

- Là các thông tin có ý nghĩa.
- Là tập hợp các thông tin có cấu trúc.
- Các thông tin này có liên quan nhau và có thể hệ thống được.

Trong khái niệm này chúng ta cần nhấn mạnh CSDL là tập hợp thông tin có tính chất hệ thống, không phải là các thông tin rời rạc, không có liên quan với nhau. Các thông tin này phải có cấu trúc và tập hợp các thông tin này phải có khả năng đáp ứng nhu cầu khai thác nhiều người sử dụng một cách đồng thời. Đó cũng chính là đặc trưng của CSDL.

** Ưu điểm của CSDL*

Giảm sự trùng lặp thông tin xuống mức thấp nhất và do đó đảm bảo tính *nhất quán và toàn vẹn dữ liệu* (Cấu trúc của cơ sở dữ liệu được định nghĩa một lần. Phần định nghĩa cấu trúc này gọi là meta-data, và được Catalog của Hệ quản trị CSDL lưu trữ).

Đảm bảo sự độc lập giữa dữ liệu và chương trình ứng dụng (Insulation between programs and data): Cho phép thay đổi cấu trúc, dữ liệu trong CSDL mà không cần thay đổi chương trình ứng dụng.

Trừu tượng hóa CSDL(Data Abstraction): Mô hình dữ liệu được sử dụng để làm ẩn lưu trữ bật lý chi tiết của dữ liệu, chỉ biểu diễn cho người sử dụng mức khái niệm của CSDL.

Nhiều khung nhìn (multi-view) cho các đối tượng người dùng khác nhau: Đảm bảo dữ liệu có thể được truy xuất theo nhiều cách khác nhau. Vì yêu cầu của mỗi đối tượng sử dụng CSDL là khác nhau nên tạo ra nhiều khung nhìn vào dữ liệu cần thiết

Đa người dùng (multi-user): Khả năng chia sử thông tin cho nhiều ngời sử dụng và nhiều ứng dụng khác nhau.

** Vấn đề cần giải quyết*

Để đạt được các ưu điểm trên, CSDL đặt ra những bấn để cần giải quyết, đó là

Tính chủ quyền của dữ liệu: Do tính chia sẻ của CSDL nên chủ quyền của CSDL dễ bị xâm phạm.

Tính bảo mật và quyền khai thác thông tin của người sử dụng: Do có nhiều người được phép khai thác CSDL nên cần thiết phải có một cơ chế bảo mật và phân quyền hạn khai thác CSDL.

Tranh chấp dữ liệu: Nhiều người được phép cùng truy cập vào CSDL với những mục đích khác nhau: Xem, thêm, xóa hoặc sửa dữ liệu. Cần phải có cơ chế ưu tiên truy cập dữ liệu hoặc giải quyết tình trạng xung đột trong quá trình khai thác cạnh tranh. Cơ chế ưu tiên có thể được thực hiện bằng việc cấp quyền (hay mức độ) ưu tiên cho từng người khai thác.

Đảm bảo dữ liệu khi có sự cố: Việc quản lý dữ liệu tập trung có thể làm tăng nguy cơ mất mát hoặc sai lệch thông tin khi có sự cố mất điện đột ngột hoặc đĩa lưu trữ bị hỏng… Một số hệ điều hành mạng có cung cấp dịch vụ sao lưu đĩa cứng (có chế sử dụng đĩa cứng dự phòng- RAID), tự động kiểm tra và khắc phục lỗi khi có sự cố. Tuy nhiên, bên cạnh dịch vụ của hệ điều hành, để đảm bảo an toàn cho CSDL, nhất thiết phải có một cơ chế khôi phục dữ liệu khi có sự cố xảy ra.

Trong những năm gần đây, làm việc với "cơ sở dữ liệu" đã trở nên quen thuộc không chỉ riêng với những người làm Tin học mà còn đối với cả những người làm trong nhiều lĩnh vực khác như Thống kê, kinh tế, quản lý doanh nghiệp,….

Các ứng dụng của Tin học vào công tác quản lý ngày càng nhiều hơn và càng đa dạng hơn. Có thể nói hầu hết các lĩnh vực kinh tế, xã hội, giáo dục, y tế v.v... đều đã ứng dụng các thành tựu mới của Tin học vào phục vụ công tác chuyên môn của mình. Chính vì lẽ đó mà ngày càng nhiều người quan tâm đến lĩnh vực thiết kế và xây dựng các CSDL

4.2.6 Lập trình mã nguồn mở

Phần mềm nguồn mở đầu tiên được công bố năm 1977 là hệ điều hành Unix BSD có thu phí tượng trưng. Phong trào phần mềm nguồn mở thực sự phát triển từ khi công bố hệ điều hành Linux đầu tiên tổ hợp phần lõi của Linus Torvalds và hệ điều hành Unix GNU của Richard Stallman vào năm 1991. Như vậy lịch sử phần mềm nguồn mở có từ rất lâu nhưng mới trở thành bình dân chỉ chưa đầy 20 năm.

Cho đến nay, phần mềm nguồn mở đã và đang trở thành một phong trào phát triển cực kỳ mạnh mẽ, có ứng dụng thực tế trong các lĩnh vực:
Hệ điều hành các máy chủ, từ cỡ lớn nhất (vd: mainframe System z của IBM chạy hệ điều hành SUSE Linux Enterprise Server) cho đến những loại máy chủ thông dụng nhất (máy chủ web Apache, máy chủ email Qmail,...).

Các loại ứng dụng từ chuyên biệt như quản trị doanh nghiệp ERP (OpenBravo), cơ sở dữ liệu cỡ lớn Oracle, đến phổ thông như bộ phần mềm văn phòng (OpenOffice) v.v..

Một vài hàng tin ngắn dưới đây cho thấy quy mô ứng dụng phần mềm nguồn mở trong thực tế:

- Quốc hội Pháp đang bắt đầu chuyển từ Windows sang phần mềm nguồn mở. Vào tháng 6/2007 tới, các nghị sỹ sẽ làm việc trên 1154 máy tính chạy Ubuntu Linux.
- Ba công ty Brazin đang tiến hành khai triển các máy tính chạy Linux cho chương trình "Máy tính dành cho mọi người" của chính phủ Brazin. Dự kiến hàng tháng sẽ giao 10.000 máy và 50.000 máy đã được giao. Công ty không nói rõ tổng số máy sẽ giao là bao nhiêu.
- Tại triển lãm Giải pháp Linux Paris 30-1-2007, hãng chế tạo ôtô lớn thứ hai châu Âu Peugeot Citroen đã ký với công ty phần mềm Novell hợp đồng khai triển 20.000 bộ Novell SUSE Linux cho máy tính cá nhân và 2500 bộ phần mềm SUSE Linux Enterprise dành cho máy chủ.
- Hãng tin Bloomberg báo cáo rằng Linux đã chính thức thắng trên 14.000 máy tính của chính quyền bang Munich, CHLB Đức, sau một quá trình xem xét dài trong đó Microsoft đã giảm giá và đích thân Tổng Giám đốc Microsoft Steve Balmer đi vận động.
- HSBC, một ngân hàng lớn của Anh có 125 triệu khách hàng toàn cầu, 9.500 văn phòng với 284.000 nhân viên tại 76 nước, đã quyết định chuẩn hóa hệ điều hành

theo một hệ SUSE Linux. Ngoài hạ tầng Windows, HSBC có khoảng vài nghìn máy chủ Linux.

- Hà Lan thống nhất dùng phần mềm nguồn mở: Chính phủ Hà lan đã đặt ra thời hạn cuối cùng là tháng 4/2008, tất cả các cơ quan chính phủ phải bắt đầu sử dụng phần mềm nguồn mở. Những đơn vị nào dùng phần mềm bản quyền phải có luận chứng đệ trình.

Như vậy có thể thấy rằng phần mềm nguồn mở đang được sử dụng trong một phạm vi rất rộng (từ hành chính, sản xuất, ngân hàng đến đại chúng) tại các quốc gia tiến tiến nhất. Không có lý gì tại Việt Nam với niềm tự hào về trí thông minh dân tộc mà lại không thể ứng dụng được.

Tại Việt Nam, phần mềm nguồn mở được coi là một hướng chiến lược trong phát triển tin học quốc gia. Tóm tắt một vài sự kiện gần đây:

- Ngày 2/3/2004 Thủ tướng Chính phủ ban hành Quyết định số 235/2004/QĐ-TTg phê duyệt Dự án tổng thể "Ứng dụng và phát triển phần mềm nguồn mở ở Việt Nam giai đoạn 2004- 2008
- QUYẾT ĐỊNH của Thủ tướng Chính phủ số 169/2006/QĐ-TTg ngày 17 tháng 7 năm 2006 “Quy định về việc đầu tư, mua sắm các sản phẩm công nghệ thông tin của các cơ quan, tổ chức sử dụng nguồn vốn ngân sách nhà nước” nêu rõ “ Ưu tiên đầu tư, mua sắm sử dụng các sản phẩm phần mềm mã nguồn mở...”.
- Từ năm 2008, hơn 20.000 máy tính của các cơ quan Đảng chuyển sang dùng, hệ điều hành máy chủ và máy trạm là Linux, bộ phần mềm văn phòng mã nguồn mở OpenOffice.
- ICTnews đưa tin: Tất cả máy tính để bàn của ngành giáo dục cả nước sẽ chuyển sang sử dụng phần mềm văn phòng nguồn mở OpenOffice vào năm 2008, coi đó là một chỉ tiêu thi đua... Ngoài OpenOffice, Bộ Giáo dục và Đào tạo cũng yêu cầu các Sở Giáo dục và Đào tạo, các trường đại học và cao đẳng sử dụng bộ gõ chữ Việt mã nguồn mở Unikey và trình duyệt nguồn mở FireFox trong công tác giảng dạy.
- Bộ Thông tin và Truyền thông đã thành lập Hội đồng tư vấn đánh giá OpenOffice và dự kiến sẽ có hướng dẫn việc dùng Open Office trong các cơ quan nhà nước.

4.3. Sử dụng công cụ và môi trường phát triển

Trong quá trình phát triển phần mềm theo nhóm hay theo từng cá nhân, thường xuyên sẽ gặp phải nhiều vấn đề như:

- Làm thế nào để quản lý được các phiên bản của quá trình quản lý phần mềm?
- Làm thế nào để quản lý source code chung cho cả nhóm?

Để giải quyết vấn đề đó, có thể sử dụng các công cụ quản lý Sourcecode, trong số đó có một số công cụ được sử dụng phổ biến như GitHub, Subverssion (SVN),…

4.3.1 GitHub

GitHub là một dịch vụ cung cấp kho lưu trữ mã nguồn Git dựa trên nền web cho các dự án phát triển phần mềm. GitHub cung cấp cả phiên bản trả tiền lẫn miễn phí cho các tài khoản. Các dự án mã nguồn mở sẽ được cung cấp kho lưu trữ miễn phí. Tính đến tháng 4 năm 2016, GitHub có hơn 14 triệu người sử dụng với hơn 35 triệu kho mã nguồn, làm cho nó trở thành máy chủ chứa mã nguồn lớn trên thế giới. Github đã trở thành một yếu tố có sức ảnh hưởng trong cộng đồng phát triển mã nguồn mở. Thậm chí nhiều nhà phát triển đã bắt đầu xem nó là một sự thay thế cho sơ yếu lý lịch và một số nhà tuyển dụng yêu cầu các ứng viên cung cấp một liên kết đến tài khoản Github để đánh giá ứng viên

Vào ngày 4 tháng 6 năm 2018, Microsoft đã thông báo việc đạt được thỏa thuận mua lại GitHub với giá 7.5 tỷ Đô la Mỹ. Ngày chính thức chuyển nhượng quyền sở hữu không được công bố. Sự phát triển của nền tảng GitHub bắt đầu vào ngày 19 tháng 10 năm 2007. Trang web được đưa ra vào tháng 4 năm 2008 do Tom Preston-Werner, Chris Wanstrath, và PJ Hyett thực hiện sau khi nó đã được hoàn thành một vài tháng trước đó, xem như giai đoạn beta.

Dự án trên Github có thể được truy cập và thao tác sử dụng một giao diện dòng lệnh và làm việc với tất cả các lệnh Git tiêu chuẩn. Github cũng cho phép người dùng đăng ký và không đăng ký để duyệt kho công cộng trên trang web. Github cũng tạo ra nhiều client và plugin cho máy tính để bàn. Trang web cung cấp các chức năng mạng xã hội như feed, theo dõi, wiki (sử dụng phần mềm Gollum Wiki) và đồ thị mạng xã hội để hiển thị cách các nhà phát triển làm việc trên kho lưu trữ.

Một người sử dụng phải tạo ra một tài khoản cá nhân để đóng góp nội dung lên Github, nhưng các kho mã nguồn công cộng có thể được duyệt và tải về với bất cứ ai.

Với một người dùng đã đăng ký tài khoản, họ có thể thảo luận, quản lý, tạo ra các kho, đóng góp cho kho của người dùng khác, và xem xét thay đổi mã.

GitHub cũng có một dịch vụ khác: một trang web kiểu pastebin gọi là Gist [2], dùng để lưu trữ các đoạn mã; trong khi Github sẽ được cho lưu trữ các dự án lớn hơn. Một dịch vụ lưu trữ khác được gọi là Speaker Deck.

Các phần mềm chạy GitHub được viết bằng Ruby on Rails và Erlang bởi GitHub, Inc, phát triển Chris Wanstrath, PJ Hyett, và Tom Preston-Werner.

4.3.2 Subversion

Hệ thống Subversion (viết tắt SVN) là một hệ thống quản lý phần tài nguyên (code, hình ảnh, video…) của một dự án. Hệ thống có khả năng cập nhật, so sánh và kết hợp tài nguyên mới vào tài nguyên cũ được giới thiệu vào năm 2000 bởi công ty CollabNet (http://subversion.tigris.org). Đây là hệ thống hỗ trợ làm việc theo nhóm rất hiệu quả. SVN hoạt động theo phương thwucs Client/Server, code project sẽ được lưu trữ trên server (SVN hosting, Google code).

Phần mềm:

- Cho client: TortoiseSVN, Download:http://tortoisesvn.net/
- Cho server: VisualSVN – Server Download: http://tortoisesvn.net/downloads.html

Các client có thể thao tác, cập nhật trực tiếp trên đó, mọi thay đổi của từng client sẽ được lưu lại.

SVN Subversion giúp giải quyết các vấn đề:

- Khi một nhóm làm việc trên cùng một project, việc nhiều người cùng chỉnh sửa nội dung của một file là điều không thể tránh khỏi. SVN Subversion cung cấp các chức năng để có thể thực hiện việc này một cách đơn giản và an toàn.
- SVN Subversion được thiết kế với mục đích thay thế hệ thống quản lý phiên bản Concurrent Versioning System (CVS) đã cũ và có nhiều nhược điểm. Subversion có thể được sử dụng để quản lý bất cứ hệ thống phiên bản nào.
- SVN Subversion là hệ thống quản lý source code tập trung (Centralized).
- SVN Subversion là hệ thống quản lý phiên bản mạnh mẽ, hữu dụng, và linh hoạt.
- SVN Subversion quản lý tập tin và thư mục theo thời gian.
- SVN Subversion giống như một hệ thống file server mà các client có thể download và upload file một cách bình thường.

- Điểm đặt biệt của SVN Subversion là nó lưu lại tất cả những gì thay đổi trên hệ thống file: file nào đã bị thay đổi lúc nào, thay đổi như thế nào, và ai đã thay đổi nó.

- SVN Subversion cũng cho phép recover lại những version cũ một cách chính xác. Các chức năng này giúp cho việc làm việc nhóm trở nên hiệu quả và an toàn hơn rất nhiều.

- Thông thường, client và server kết nối thông qua mạng LAN hoặc Internet. Client và server có thể cùng chạy trên một máy nếu SVN Subversion có nhiệm vụ theo vết lịch sử của dự án do các nhà phát triển phần mềm phát triển trong nội bộ.

- SVN Subversion hỗ trợ khá nhiều giao thức để kết nối giữa client và server. Ví dụ bạn có thể dùng các giao thức của ứng dụng web như http:// hoặc https://, hay các giao thức của svn như svn:// hoặc svn+ssh://, hoặc nếu phần mềm client và server cài chung trên 1 máy thì có thể dùng file://.

Việc cho phép server hỗ trợ giao thức nào phụ thuộc vào lúc cấu hình.

CHƯƠNG 5: KIỂM THỬ PHẦN MỀM

Nội dung của chương:

5.1. Các mức kiểm thử phần mềm

5.2 Quy trình kiểm thử phần mềm

5.3. Kiểm thử tự động và công cụ hỗ trợ

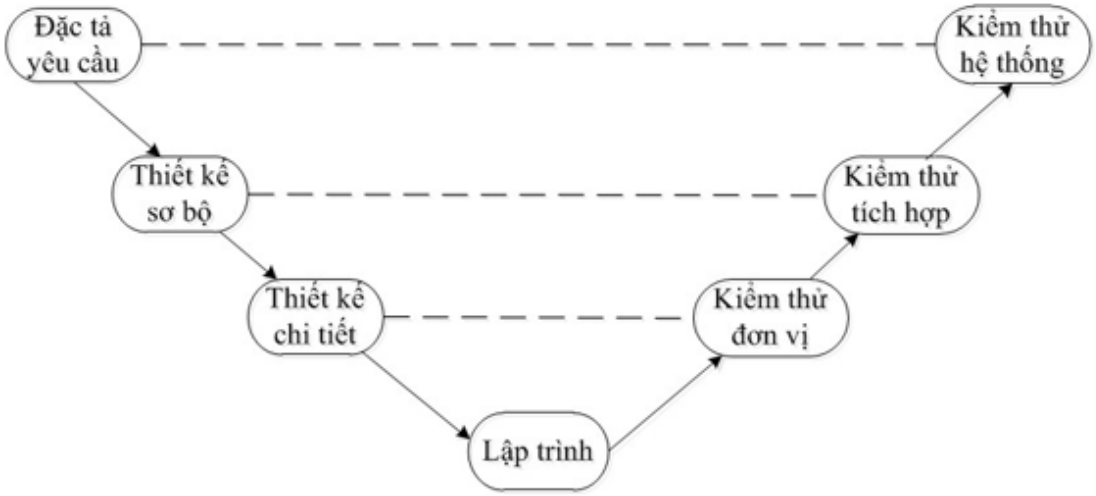
Mục tiêu của chương:

- Phân biệt các mức kiểm thử phần mềm
- Hiểu được từng giai đoạn trong quy trình kiểm thử phần mềm
- Biết cách sử dụng 1 số công cụ kiểm thử tự động, thực thi các testcase sử dụng công cụ kiểm thử tự động.

5.1. Các mức kiểm thử phần mềm

Một trong các khái niệm then chốt của việc kiểm thử là các mức của việc kiểm thử. Các mức của việc kiểm thử phản ánh mức độ trừu tượng được thấy trong mô hình thác nước của vòng đời của việc phát triển phần mềm.

Dù có một số nhược điểm, mô hình này vẫn rất hữu ích cho việc kiểm thử, là phương tiện để xác định các mức kiểm thử khác nhau và làm sáng tỏ mục đích của mỗi mức. Một dạng của mô hình thác nước được trình bày trong hình 1.6. Dạng này nhấn mạnh sự tương ứng của việc kiểm thử với các mức thiết kế. Lưu ý rằng theo các thuật ngữ của việc kiểm thử hàm, ba mức của định nghĩa (đặc tả, thiết kế sơ bộ và thiết kế chi tiết) tương ứng trực tiếp với ba mức của việc kiểm thử là kiểm thử đơn vị, kiểm thử tích hợp và kiểm thử hệ thống. Các mức của kiểm thử cũng làm nảy sinh vấn đề về thứ tự kiểm thử: dưới lên, trên xuống hoặc các khả năng khác



Hình 5.1: Các mức trừu tượng và mức kiểm thử trong mô hình thác nước

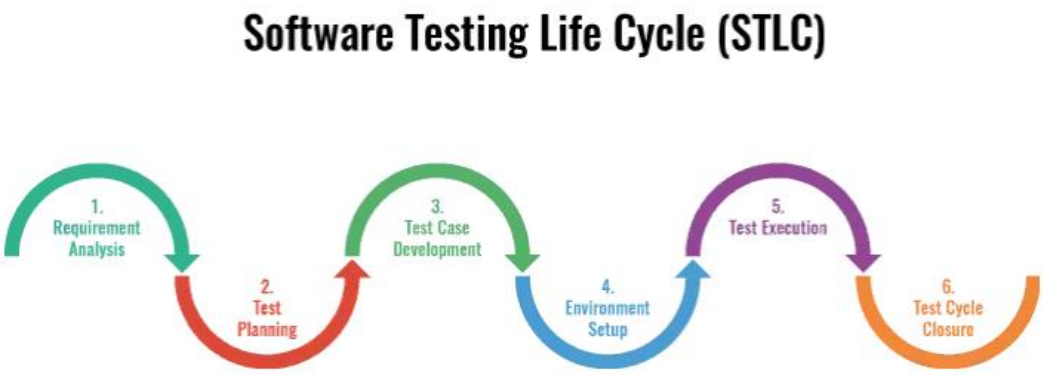
Các mức kiểm thử có thể được mô tả sơ bộ như sau:

- Kiểm thử đơn vị: Kiểm thử đơn vị là việc kiểm thử các đơn vị chương trình một cách cô lập. Thế nào là một đơn vị chương trình? Câu trả lời phụ thuộc vào ngữ cảnh công việc. Một đơn vị chương trình là một đoạn mã nguồn như hàm hoặc phương thức của một lớp, có thể được gọi từ ngoài, và cũng có thể gọi đến các đơn vị chương trình khác. Đơn vị cũng còn được coi là một đơn thể để kết hợp. Đơn vị chương trình cần được kiểm thử riêng biệt để phát hiện lỗi trong nội tại và khắc phục trước khi được tích hợp với các đơn vị khác.
- Kiểm thử tích hợp: Mức kế tiếp với kiểm thử đơn vị là kiểm thử tích hợp. Sau khi các đơn vị chương trình để cấu thành hệ thống đã được kiểm thử, chúng cần được kết nối với nhau để tạo thành hệ thống đầy đủ và có thể làm việc. Công việc này không hề đơn giản và có thể có những lỗi về giao diện giữa các đơn vị, và cần phải kiểm thử để pháthiện những lỗi này. Công đoạn này gồm hai giai đoạn: giai đoạn kiểm thử tích hợp và giai đoạn kiểm thử hệ thống. Kiểm thử tích hợp nhằm đảm bảo hệ thống làm việc ổn định trong môi trường thí nghiệm để sẵn sàng cho việc đưa vào môi trường thực sự bằng cách đặt các đơn vị với nhau theo phương pháp tăng dần.
- Kiểm thử hệ thống: Kiểm thử mức này được áp dụng khi đã có một hệ thống đầy đủ sau khi tất cả các thành phần đã được tích hợp. Mục đích của kiểm thử hệ thống là để đảm bảo rằng việc cài đặt tuân thủ đầy đủ các yêu cầu được đặc tả của người dùng. Công việc này tốn nhiều công sức, vì có nhiều khía cạnh về yêu cầu người dùng cần được kiểm thử.

Kiểm thử chấp nhận: Khi nhóm kiểm thử hệ thống đã thỏa mãn với một sản phẩm, sản phẩm đó đã sẵn sàng để đưa vào sử dụng. Khi đó hệ thống cần trải qua giai đoạn kiểm thử chấp nhận. Kiểm thử chấp nhận được thực thi bởi chính các khách hàng nhằm đảm bảo rằng sản phẩm phần mềm làm việc đúng như họ mong đợi. Có hai loại kiểm thử chấp nhận: kiểm thử chấp nhận người dùng, được tiến hành bởi người dùng, và kiểm thử chấp nhận doanh nghiệp, được tiến hành bởi nhà sản xuất ra sản phẩm phần mềm.

5.2. Quy trình kiểm thử phần mềm

Quy trình kiểm thử phần mềm xác định các giai đoạn/ pha trong kiểm thử phần mềm. Tuy nhiên, không có STLC tiêu chuẩn cố định nào trên thế giới, nhưng về cơ bản quy trình kiểm thử bao gồm những giai đoạn sau:



Hình 5.2 Quy trình kiểm thử phần mềm

1. Requirenment analysis - Phân tích yêu cầu

2. Test planning - Lập kế hoạch kiểm thử

3. Test case development - Thiết kế kịch bản kiểm thử

4. Test environment set up - Thiết lập môi trường kiểm thử

5. Test execution - Thực hiện kiểm thử

6. Test cycle closure – Kết thúc chu kỳ kiểm thử

Các giai đoạn kiểm thử được thực hiện một cách tuần tự. Mỗi giai đoạn sẽ có những mục tiêu khác nhau, đầu vào và kết quả đầu ra khác nhau nhưng mục đích cuối cùng vẫn là đảm bảo chất lượng sản phẩm phần mềm tốt nhất. Sau đây, chúng ta sẽ tìm hiểu chi tiết thông tin về các hoạt động, ai là người thực hiện, đầu vào, đầu ra của từng giai đoạn trong quy trình kiểm thử phần mềm.

5.2.1. Requirenment analysis - Phân tích yêu cầu

Đầu vào

Đầu vào của giai đoạn phân tích yêu cầu bao gồm các tài liệu như: tài liệu đặc tả yêu cầu, tài liệu thiết kế hệ thống, tài liệu khách hàng yêu cầu về các tiêu chí chấp nhận của sản phẩm, bản prototype của khách hàng yêu cầu (nếu có), ...

Hoạt động

- Phân tích yêu cầu là giai đoạn đầu tiên trong quy trình kiểm thử phần mềm.

- QA team sẽ thực hiện đọc hiểu, nghiên cứu và phân tích cụ thể các yêu cầu trong tài liệu đặc tả của dự án hoặc tài liệu khách hàng. Qua hoạt động này, QA team sẽ nắm bắt được các yêu cầu mà dự án đưa ra bao gồm yêu cầu kiểm thử chức năng/ phi chức năng nào.
- Ngoài ra, trong quá trình phân tích, nghiên cứu tài liệu, nếu có câu hỏi phát sinh hay đề xuất giải quyết, QA team sẽ đưa ra câu hỏi với các bên liên quan như BA (Business Analysis), PM (Project Manager), team leader, khách hàng để hiểu chính xác hơn về yêu cầu của sản phẩm. Những câu hỏi này sẽ được lưu trữ vào file Q&A (Question and Answer). Các câu hỏi nên được đưa ra dưới dạng Yes/No question hoặc các lựa chọn để tiết kiệm thời gian trả lời cũng như hỗ trợ đưa ra những gợi ý hay để xây dựng sản phẩm ngay từ đầu. Như vậy, đương nhiên là chúng ta không nên nêu ra những câu hỏi dạng là gì, như thế nào, tại sao... Những câu hỏi như thế thường mất thời gian để giải thích và cũng khó có thể giải thích một cách chi tiết nhất có thể. Hơn nữa, đối với khách hàng không có sự hiểu biết về lĩnh vực phần mềm mà họ yêu cầu thì càng không thể trả lời những câu hỏi mang tính chuyên môn cao. Chính chúng ta sẽ là người hỗ trợ và đưa ra giải pháp thích hợp cho khách hàng lựa chọn.

Đầu ra

Đầu ra của giai đoạn phân tích yêu cầu bao gồm tài liệu chứa các câu hỏi và câu trả lời liên quan đến nghiệp vụ của hệ thống, tài liệu báo cáo tính khả thi, phân tích rủi ro của việc kiểm thử phần mềm.

5.2.2 Test planning - Lập kế hoạch kiểm thử

Đầu vào

Đầu vào của giai đoạn lập kế hoạch kiểm thử là các tài liệu đặc tả đã được cập nhật thông qua các câu hỏi và trả lời được đưa ra trong giai đoạn phân tích yêu cầu, tài liệu báo cáo tính khả thi, phân tích rủi ro của việc kiểm thử phần mềm.

Hoạt động

Dựa vào các tài liệu được cung cấp và cập nhật mới nhất, thông thường, test manager hoặc test leader sẽ là người lập kế hoạch kiểm thử cho cả QA team. Lập kế hoạch kiểm thử nhằm xác định một số yếu tố quan trọng sau:

- **Xác định phạm vi (Scope) dự án:** Dự án thực hiện trong thời gian bao lâu? Bao gồm những công việc gì cho từng khoảng thời gian xác định? Từ đó đưa ra lịch trình thực hiện cho từng công việc nhỏ sao cho phù hợp với toàn bộ đội dự án.

- **Xác định phương pháp tiếp cận**: Nói về cách tiếp cận để kiểm thử cho một đối tượng nào đó, thì phải dựa vào nhiều thứ, ví dụ như: Thời gian cho phép test có phù hợp với con số ước lượng, nhiều hay ít, yêu cầu chất lượng từ phía khách hàng thế nào? Cao, thấp hay khắc khe hay sao cũng được? Công nghệ / kỹ thuật sử dụng để phát triển ứng dụng này là gì? Lĩnh vực của hệ thống/sản phẩm đang được test (domain) là gì?...Từ đó, test manager có thể đưa ra những phương pháp và kế hoạch phù hợp nhất cho cả quá trình thực hiện dự án sao cho đúng với các tiêu chí chấp nhận của sản phẩm và kịp tiến độ với các mốc thời gian bàn giao, phát hành.
- **Xác định các nguồn lực**

  Con người: Bao nhiêu người tham gia dự án, ai sẽ test phần nào, bao nhiêu tester tham gia? Tester và nhóm phát triển có kinh nghiệm về lĩnh vực này không?

  Thiết bị: số lượng server, version, máy tính, mobile để thực hiện test là bao nhiêu.
- **Lên kế hoạch thiết kế công việc test:** Bản kế hoạch kiểm thử sẽ bao gồm các nội dung:

  Liệt kê các chức năng cần kiểm thử.

  Để thực hiện test chức năng này thì cần làm những công việc gì, trong thời gian bao lâu, cái nào thực hiện trước, cái nào thực hiện sau, ai là người thực hiện.

  Xác định điều kiện bắt đầu: xác định những điều kiện tối thiểu để bắt đầu hoạt động kiểm thử cho từng chức năng.

  Xác định điều kiện kết thúc: khi có những điều kiện nào thì sẽ kết thúc việc kiểm thử.

  Đầu ra

  Đầu ra của giai đoạn lập kế hoạch bao gồm các tài liệu như test plan, test estimation, test schedule.

5.2.3. Test case development - Thiết kế kịch bản kiểm thử

Đầu vào

Đầu vào của giai đoạn thiết kế kịch bản kiểm thử là test plan, test estimation, test schedule, các tài liệu đặc tả đã được cập nhật.

Hoạt động

- **Review tài liệu:** Đầu tiên, các kiểm thử viên cần review lại tất cả các tài liệu để xác định công việc cần làm, các công việc có khác gì so với dự án trước khách hàng đưa cho, chức năng nào cần test, chức năng nào không cần test lại nữa. Từ đó, vừa có thể tiết kiệm thời gian mà vẫn đưa ra được một kịch bản kiểm thử đầy đủ và hiệu quả.

- **Viết test case/ check list:** Sau đó, tester bắt tay vào việc viết test case chi tiết dựa vào kế hoạch đã đưa ra và vận dụng các kỹ thuật thiết kế kịch bản kiểm thử. Test case cần bao phủ được tất cả các trường hợp kiểm thử có thể xảy ra cũng như đáp ứng đầy đủ các tiêu chí của sản phẩm. Đồng thời tester cũng cần đánh giá mức độ ưu tiên cho từng test case.
- **Chuẩn bị dữ liệu kiểm thử:** Cùng với việc tạo ra các test case chi tiết, đội kiểm thử cũng cần chuẩn bị trước các dữ liệu kiểm thử cho các trường hợp cần thiết như test data, test script.
- **Review test case/ check list:** Sau khi hoàn thành, các thành viên trong đội kiểm thử hoặc test leader cũng cần review lại test case đã tạo để có thể bổ sung, hỗ trợ lẫn nhau nhằm tránh những sai sót trong thiết kế test case và rủi ro về sau.

Đầu ra

Sau khi hoàn thành thiết kế kịch bản kiểm thử, đội kiểm thử sẽ có các tài liệu bao gồm: test design, test case, check list, test data, test automation script.

5.2.4. Test environment set up - Thiết lập môi trường kiểm thử

Đầu vào

Đầu vào của giai đoạn cài đặt môi trường kiểm thử là test plan, smoke test case, test data.

Hoạt động

- Việc cài đặt môi trường kiểm thử là giai đoạn cũng rất quan trọng trong vòng đời phát triển phần mềm. Môi trường kiểm thử sẽ được quyết định dựa trên những yêu cầu của khách hàng, hay đặc thù của sản phẩm ví dụ như server/ client/ network, ...
- Tester cần chuẩn bị một vài test case để kiểm tra xem môi trường cài đặt đã sẵn sàng cho việc kiểm thử hay chưa. Đây chính là việc thực thi các smoke test case.

Đầu ra

Đầu ra của giai đoạn này là môi trường đã được cài đặt đúng theo yêu cầu, sẵn sàng cho việc kiểm thử và kết quả của smoke test case.

5.2.5. Test execution - Thực hiện kiểm thử

Đầu vào

Tài liệu đầu vào của giai đoạn này là test plan, test design, test case, check list, test data, test automation script.

Hoạt động

- Thực hiện các test case như thiết kế và mức độ ưu tiên đã đưa ra trên môi trường đã được cài đặt.
- So sánh với kết quả mong đợi sau báo cáo các bug xảy ra lên tool quản lý lỗi và theo dõi trạng thái của lỗi đến khi được sửa thành công.
- Thực hiện re-test để verify các bug đã được fix và regression test khi có sự thay đổi liên quan.
- Trong quá trình thực hiện kiểm thử, kiểm thử viên cũng có thể hỗ trợ, đề xuất cho cả đội dự án để có giải pháp hợp lý và kết hợp công việc hiệu quả.
- Đo và phân tích tiến độ: kiểm thử viên cũng cần kiểm soát chặt chẽ tiến độ công việc của mình bằng cách so sánh tiến độ thực tế với kế hoạch, nếu chậm cần phải điều chỉnh sao cho kịp tiến độ dự án, nếu nhanh cũng cần điều chỉnh vì có thể test lead lên kế hoạch chưa sát với thực tế dự án. Từ đó có thể sửa chữa test plan cần điều chỉnh để phù hợp với tiến độ dự án đưa ra.
- Report thường xuyên cho PM và khách hàng về tình hình thực hiện dự án: Cung cấp thông tin trong quá trình kiểm thử đã làm được những chức năng nào, còn chức năng nào, hoàn thành được bao nhiêu phần trăm công việc, báo cáo các trường hợp phát sinh sớm, tránh ảnh hưởng tiến độ công việc của cả ngày.

Đầu ra

Đầu ra của giai đoạn này là test results (kết quả kiểm thử), defect reports (danh sách các lỗi tìm được).

5.2.6. Test cycle closure – Kết thúc chu kỳ kiểm thử

Đầu vào

Đầu vào của giai đoạn đóng chu trình kiểm thử là bao gồm tất cả những tài liệu liên quan đã được tổng hợp, ghi chép và hoàn thiện đầy đủ trong suốt quy trình kiểm thử của dự án: tài liệu phân tích đặc tả yêu cầu, test plan, test results, defect reports, tài liệu Q&A, ...

Hoạt động

- Đây là giai đoạn cuối cùng trong quy trình kiểm thử phần mềm.
- Ở giai đoạn này, QA team thực hiện tổng kết, báo cáo kết quả về việc thực thi test case, bao nhiêu case pass/ fail, bao nhiêu case đã được fix, mức độ nghiêm trọng của lỗi, bao nhiêu lỗi cao/ thấp, lỗi còn nhiều ở chức năng nào, dev nào nhiều lỗi. Chức năng nào đã hoàn thành test/ chưa hoàn thành test/ trễ tiến độ bàn giao.

- Đánh giá các tiêu chí hoàn thành như phạm vi kiểm tra, chất lượng, chi phí, thời gian, mục tiêu kinh doanh quan trọng.
- Ngoài ra, giai đoạn này cũng thảo luận tất cả những điểm tốt, điểm chưa tốt và rút ra bài học kinh nghiệm cho những dự án sau, giúp cải thiện quy trình kiểm thử.

Đầu ra

Đầu ra của giai đoạn này bao gồm các tài liệu: Test report, Test results (final)

5.3. Kiểm thử tự động và công cụ hỗ trợ

Kiểm thử tự động là quá trình thực hiện một cách tự động các bước trong một kịch bản kiểm thử. Kiểm thử tự động bằng một công cụ nhằm rút ngắn thời gian kiểm thử. Mục đích của kiểm thử tự động là giảm thiểu thời gian, công sức và kinh phí, tăng độ tin cậy, tăng tính hiệu quả và giảm sự nhàm chán cho người kiểm thử trong quá trình kiểm thử sản phẩm phần mềm. Kiểm thử tự động sẽ được sử dụng khi dự án không đủ tài nguyên (thời gian, nhân lực và chi phí), phải thực hiện kiểm thử hồi quy khi sản phẩm được sửa đổi hoặc nâng cấp và cần kiểm thử lại các tính năng đã thực hiện tốt trước đó, kiểm tra khả năng vận hành của sản phẩm trong các môi trường đặc biệt (đo tốc độ xử lý trung bình ứng với mỗi yêu cầu, xác định khả năng chịu tải tối đa, xác định cấu hình tối thiểu để thực thi hệ thống, kiểm tra các cơ chế an ninh và an toàn, ...).

Kiểm thử tự động đang được quan tâm như là một giải pháp hiệu quả và duy nhất nhằm cải thiện tính chính xác và hiệu quả cũng như giảm kinh phí và rút ngắn thời gian trong quá trình kiểm thử các sản phẩm phần mềm. Đã có nhiều công cụ được phát triển hỗ trợ các mục đích trên. Tùy thuộc vào yêu cầu kiểm thử của từng sản phẩm, các công ty sẽ lựa chọn các công cụ phù hợp. Tuy nhiên, rất khó để tìm được một bộ công cụ đáp ứng tất cả các yêu cầu kiểm thử. Trong nhiều trường hợp, các công ty cần chủ động mở rộng và phát triển thêm các công cụ phục vụ các mục đích cụ thể.

5.3.1. JUnit

Công cụ kiểm thử cho các đơn vị chương trình viết bằng Java, JUnit, cung cấp một cơ sở hạ tầng chuẩn cho việc thiết lập các bộ kiểm thử. Một khi bộ kiểm thử được thiết lập, nó có thể tự động chạy mỗi khi mã thay đổi. JUnit khuyến khích các nhà phát triển viết các kịch bản kiểm thử, chèn các mã kiểm thử vào mã nguồn Java và thực hiện chúng để phát hiện các lỗi bên trong đơn vị chương trình. Khác với các

công cụ khác, JUnit không hỗ trợ cơ chế sinh các ca kiểm thử. Hiện nay, JUnit đã được tích hợp trong Eclipse và hỗ trợ rất đắc lực cho quá trình kiểm thử.

Tính năng:

- Hỗ trợ kiểm thử tự động các đơn vị nhỏ nhất của mã nguồn.
- Tích hợp dễ dàng với các công cụ CI/CD.
- Cung cấp các annotation và assert để viết các test case dễ dàng.

Ưu điểm: Đơn giản, dễ sử dụng, và là tiêu chuẩn cho kiểm thử đơn vị trong Java.

5.3.2. Apache Jmeter

Apache Jmeter được dùng để kiểm thử khả năng chịu tải và kiểm thử hiệu năng cho các ứng dụng Web và một số ứng dụng khác. Công cụ này hỗ trợ kiểm thử hiệu năng của các mã nguồn được viết bằng các ngôn ngữ khác nhau như PHP, Java, ASP.NET, . . . Apache JMeter mô phỏng khả năng chịu tải của các máy chủ trên máy sử dụng để kiểm thử hệ thống. Công cụ này hỗ trợ giao diện đồ họa giúp phân tích tốt hiệu suất khi kiểm thử đồng thời nhiều ca kiểm thử. Ngoài ra, Apache JMeter còn hỗ trợ thêm nhiều tiện ích khác.

Tính năng:

- Hỗ trợ kiểm thử hiệu suất, tải, và stress cho các ứng dụng web.
- Cung cấp giao diện người dùng để thiết lập và chạy các kịch bản kiểm thử.
- Hỗ trợ nhiều giao thức khác nhau như HTTP, FTP, JDBC.

Ưu điểm: Mạnh mẽ, linh hoạt, và có khả năng mở rộng.

5.3.4. TestNG

TestNG là một khung kiểm thử mạnh mẽ và linh hoạt được thiết kế đặc biệt cho ngôn ngữ lập trình Java. Được phát triển bởi Cedric Beust, TestNG được tạo ra nhằm khắc phục những hạn chế của JUnit và cung cấp nhiều tính năng bổ sung để hỗ trợ việc kiểm thử phần mềm một cách toàn diện và hiệu quả hơn. Tên "TestNG" bắt nguồn từ "Test Next Generation," phản ánh mục tiêu của công cụ này là cải tiến và mở rộng khả năng kiểm thử.

Các tính năng chính của TestNG

- **Kiểm thử đơn vị (Unit Testing)**: Giúp kiểm tra các đơn vị nhỏ nhất của mã nguồn.

- **Kiểm thử tích hợp (Integration Testing)**: Đảm bảo các thành phần khác nhau của hệ thống hoạt động tốt khi tích hợp với nhau.
- **Kiểm thử chức năng (Functional Testing)**: Kiểm thử các chức năng cụ thể của ứng dụng.
- **Kiểm thử hệ thống (System Testing)**: Đảm bảo hệ thống hoạt động đúng như mong đợi khi hoạt động đầy đủ.

TestNG là một công cụ kiểm thử mạnh mẽ và linh hoạt, cung cấp nhiều tính năng tiên tiến để hỗ trợ kiểm thử phần mềm hiệu quả. Với sự hỗ trợ đa dạng các loại kiểm thử, khả năng cấu hình linh hoạt, và tích hợp tốt với các công cụ CI/CD, TestNG đã trở thành một lựa chọn phổ biến cho các nhà phát triển và kiểm thử viên trong việc đảm bảo chất lượng phần mềm.

5.3.4. Selenium

Selenium là một bộ công cụ kiểm thử phần mềm mã nguồn mở, được thiết kế để kiểm thử các ứng dụng web. Selenium cho phép các nhà phát triển và kiểm thử viên tự động hóa các tác vụ trình duyệt, giúp thực hiện các kiểm thử chức năng, hồi quy và kiểm thử giao diện người dùng trên nhiều trình duyệt khác nhau một cách hiệu quả và tiết kiệm thời gian.

Các thành phần chính của Selenium

1. **Selenium WebDriver:** Selenium WebDriver là một giao diện lập trình ứng dụng (API) cung cấp các phương thức để điều khiển và tương tác trực tiếp với trình duyệt. WebDriver hỗ trợ nhiều ngôn ngữ lập trình như Java, C#, Python, Ruby, JavaScript, và Kotlin. Selenium WebDriver tương tác trực tiếp với trình duyệt, hỗ trợ nhiều ngôn ngữ lập trình và trình duyệt.
2. **Selenium IDE (Integrated Development Environment):** Selenium IDE là một tiện ích mở rộng của trình duyệt Firefox và Chrome, cho phép ghi lại và phát lại các tác vụ trình duyệt. Nó là công cụ lý tưởng cho việc tạo các kiểm thử nhanh chóng và dễ dàng. Selenium IDE dễ sử dụng, không yêu cầu kiến thức lập trình, phù hợp cho việc tạo các kiểm thử đơn giản.
3. **Selenium Grid:** Selenium Grid cho phép chạy các kiểm thử song song trên nhiều máy và nhiều trình duyệt khác nhau. Điều này giúp tiết kiệm thời gian kiểm thử

và tăng cường hiệu suất. Selenium IDE hỗ trợ kiểm thử song song, dễ dàng mở rộng và tích hợp với các công cụ CI/CD.

Các tính năng chính của Selenium

1. **Hỗ trợ nhiều trình duyệt và nền tảng:** Selenium hỗ trợ các trình duyệt phổ biến như Chrome, Firefox, Safari, Edge, và Internet Explorer. Ngoài ra, nó cũng hỗ trợ các hệ điều hành khác nhau như Windows, macOS, và Linux.
2. **Hỗ trợ đa ngôn ngữ lập trình:** Selenium cung cấp API cho nhiều ngôn ngữ lập trình, giúp các nhà phát triển và kiểm thử viên có thể viết các kịch bản kiểm thử bằng ngôn ngữ mà họ quen thuộc như Java, C#, Python, Ruby, JavaScript.
3. **Khả năng tương tác với các thành phần trên trang web:** Selenium cho phép tương tác với các thành phần trên trang web như nhập liệu vào các trường văn bản, nhấp chuột, chọn tùy chọn trong các danh sách, và thực hiện các hành động phức tạp khác.
4. **Kiểm thử không đồng bộ:** Selenium hỗ trợ kiểm thử các ứng dụng web không đồng bộ, giúp xử lý các trường hợp tải trang hoặc xử lý AJAX một cách hiệu quả.

Lợi ích của Selenium

1. **Tiết kiệm thời gian và chi phí**: Tự động hóa kiểm thử giúp giảm thiểu thời gian và chi phí so với kiểm thử thủ công.
2. **Độ tin cậy cao**: Kiểm thử tự động đảm bảo các kiểm thử được thực hiện một cách nhất quán và không bị ảnh hưởng bởi lỗi con người.
3. **Tích hợp dễ dàng**: Selenium dễ dàng tích hợp với các công cụ quản lý kiểm thử, CI/CD và các framework kiểm thử khác

Selenium là một công cụ mạnh mẽ và linh hoạt cho việc kiểm thử tự động các ứng dụng web. Với sự hỗ trợ đa nền tảng, đa ngôn ngữ và khả năng tương tác trực tiếp với trình duyệt, Selenium đã trở thành lựa chọn phổ biến và được tin cậy bởi nhiều tổ chức và nhà phát triển trên toàn thế giới.

5.3.5. Katalon Studio

Katalon Studio là một công cụ kiểm thử tự động toàn diện, hỗ trợ kiểm thử các ứng dụng web, ứng dụng di động, API và máy tính để bàn. Được phát triển bởi Katalon LLC, Katalon Studio cung cấp một nền tảng dễ sử dụng với nhiều tính năng mạnh mẽ, giúp cải thiện hiệu suất và hiệu quả của quy trình kiểm thử phần mềm.

Các tính năng chính của Katalon Studio

1. **Kiểm thử web:** Katalon Studio cung cấp các công cụ để kiểm thử tự động các ứng dụng web, hỗ trợ nhiều trình duyệt khác nhau như Chrome, Firefox, Edge, và Safari; Cung cấp các tính năng như ghi và phát lại các hành động trên trình duyệt, kiểm thử các yếu tố trên trang web, và tích hợp với các framework kiểm thử phổ biến.
2. **Kiểm thử di động:** Hỗ trợ kiểm thử các ứng dụng di động trên cả nền tảng iOS và Android; Cung cấp khả năng ghi và phát lại các hành động trên thiết bị di động thực tế hoặc các trình giả lập/giả lập.
3. **Kiểm thử API:** Hỗ trợ kiểm thử API RESTful và SOAP; Cung cấp các tính năng để tạo, gửi và xác minh các yêu cầu API, cũng như tích hợp với các dịch vụ kiểm thử API khác như Postman.
4. **Kiểm thử máy tính để bàn:** Katalon Studio cũng hỗ trợ kiểm thử các ứng dụng máy tính để bàn trên nền tảng Windows, giúp kiểm thử các ứng dụng doanh nghiệp truyền thống.
5. **Tích hợp CI/CD:** Dễ dàng tích hợp với các công cụ CI/CD như Jenkins, Bamboo, và Azure DevOps, giúp tự động hóa quá trình kiểm thử và triển khai phần mềm.
6. **Báo cáo và phân tích:** Cung cấp các báo cáo kiểm thử chi tiết và khả năng phân tích dữ liệu kiểm thử để giúp hiểu rõ hơn về kết quả kiểm thử và hiệu suất của ứng dụng.
7. **Hỗ trợ nhiều ngôn ngữ lập trình:** Mặc dù Katalon Studio sử dụng ngôn ngữ Groovy làm ngôn ngữ mặc định, nó cũng hỗ trợ các ngôn ngữ lập trình khác như Java, giúp linh hoạt hơn trong việc viết và duy trì các kịch bản kiểm thử.

Lợi ích của Katalon Studio

1. **Dễ sử dụng**: Với giao diện người dùng thân thiện và khả năng ghi và phát lại các kiểm thử, Katalon Studio rất dễ sử dụng, ngay cả với những người mới bắt đầu kiểm thử tự động.
2. **Tích hợp toàn diện**: Katalon Studio tích hợp tốt với nhiều công cụ và dịch vụ khác, bao gồm các công cụ quản lý kiểm thử, CI/CD, và các dịch vụ kiểm thử API.

3. **Miễn phí và có phiên bản thương mại**: Katalon Studio có phiên bản miễn phí với đầy đủ các tính năng cần thiết cho kiểm thử tự động, cũng như phiên bản thương mại với các tính năng bổ sung và hỗ trợ kỹ thuật.
4. **Cộng đồng mạnh mẽ**: Với một cộng đồng người dùng và nhà phát triển rộng lớn, Katalon Studio cung cấp nhiều tài liệu, hướng dẫn và hỗ trợ trực tuyến.

Katalon Studio là một công cụ kiểm thử tự động mạnh mẽ và dễ sử dụng, cung cấp nhiều tính năng để hỗ trợ kiểm thử các ứng dụng web, di động, API và máy tính để bàn. Với sự tích hợp toàn diện và khả năng tùy chỉnh linh hoạt, Katalon Studio đã trở thành một lựa chọn phổ biến cho các nhà phát triển và kiểm thử viên trên toàn thế giới, giúp nâng cao chất lượng phần mềm và tối ưu hóa quy trình kiểm thử

CHƯƠNG 6: TRIỂN KHAI VÀ BẢO TRÌ PHẦN MỀM

Nội dung của chương:

6.1 Tổng quan về triển khai và bảo trì phần mềm

6.2 Triển khai phần mềm

6.3 Bảo trì phần mềm

6.4 Công cụ, kỹ thuật trợ giúp

Mục tiêu cần đạt được của chương

Áp dụng được các kiến thức vận hành & bảo trì phần mềm trong triển khai các giai đoạn của vòng đời phần mềm

6.1 Giới thiệu về triển khai & bảo trì phần mềm

- **Triển khai Phần mềm**

Triển khai phần mềm còn được gọi là triển khai IT. Đây là quá trình lắp/cài đặt thêm các tính năng mới hoặc tích hợp phần mềm mới vào môi trường vận hành của người dùng, nơi các ứng dụng và dịch vụ khác của họ đang vận hành. Để tránh lỗi và các vấn đề xung đột môi trường, việc lập kế hoạch triển khai cần phải được chú trọng. Hình 6.1 là ví dụ minh hoạ việc triển khai không được lập kế hoạch.

Hình 6.1: Minh hoạ kết quả triển khai không được lập kế hoạch

Bên cạnh công việc chuyển giao, cài đặt và thiết lập thông tin cấu hình sản phẩm để đảm bảo hệ thống vận hành thông suốt trên môi trường người dùng; triển khai phần mềm còn bao gồm các hoạt động như:

- Biên dịch và đóng gói phần mềm để sẵn sàng cho hoạt động kiểm thử chấp thuận phần mềm.
- Xây dựng các tài liệu hướng dẫn người dùng.
- Thông báo phát hành/cập nhật sản phẩm.
- Huấn luyện người dùng sử dụng sản phẩm.
- Giám sát, theo dõi quá trình vận hành sản phẩm.
- Định kỳ bảo trì/bảo dưỡng sản phẩm.

Để đánh giá việc triển khai phần mềm thành công, chúng ta có thể dựa vào 4 tiêu chí cơ bản sau:

- Phản hồi tích cực từ phía người dùng cuối về sản phẩm/tính năng mới
- Số lượng khách hàng sử dụng sản phẩm ngày càng tăng
- Giảm chi cho việc hỗ trợ khách hàng sử dụng sản phẩm, bảo trì sản phẩm
- Tăng lợi nhuận từ việc sử dụng sản phẩm, cập nhật tính năng mới cho sản phẩm

❖ **Bảo trì phần mềm**

Theo IEEE (1993), bảo trì phần mềm được định nghĩa như sau: *"Bảo trì là việc sửa đổi phần mềm sau khi phát hành nhằm chỉnh sửa các lỗi phát sinh, cải thiện hiệu năng, các thuộc tính của phần mềm hoặc làm cho phần mềm thích ứng với môi trường vận hành mới".*

Bảo trì phần mềm đóng một vai trò quan trọng trong việc giữ cho phần mềm chạy trơn tru. Ngoài ra, vì đặc tính thay đổi của phần mềm, bảo trì là hoạt động không thể thiếu nhằm duy trì thời gian sống và tính hữu ích của sản phẩm. Bảo trì có mối quan hệ chặt chẽ với hoạt động triển khai phần mềm & nên tiến hành một cách định kỳ nhằm:

- Giảm thiểu thời gian phần mềm hỏng hóc dẫn tới ngưng hoạt động, giảm thiểu chi phí bảo trì.
- Duy trì tính an toàn, an ninh và độ tin cậy của sản phẩm bằng cách khắc phục các lỗ hổng bị xâm nhập khi phần mềm được vận hành trong thời gian dài.
- Loại bỏ những chức năng lỗi thời của sản phẩm, cải thiện hiệu xuất của chương trình.

Lưu ý: Khi quyết định thực hiện bảo trì phần mềm, điều quan trọng là phải xem xét những thay đổi nào có thể xảy ra và ảnh hưởng đến người dùng như thế nào, phản ứng tiềm năng của họ ra sao và các giải pháp để giải quyết vấn đề đó.

Bảo trì có thể được phân loại thành bốn loại:

1) **Bảo trì sửa lỗi (Corrective Maintenance)**: sửa chữa các lỗi, sai sót và khiếm khuyết trong phần mềm; thường xuất hiện dưới dạng các bản cập nhật nhanh, nhỏ & thường xuyên.
2) **Bảo trì thích ứng (Adaptive Maintenance)**: tập trung vào những thay đổi liên quan đến cơ sở hạ tầng phần mềm (hệ điều hành, phần cứng, và nền tảng mới) để giữ cho chương trình tương thích với chúng.
3) **Bảo trì nâng cấp/hoàn thiện (Perfective Maintenance)**: là loại bảo trì chiếm phần lớn kinh phí, thường gấp nhiều lần so với các loại hình bảo trì khác (chiếm ~ 65% chi phí bảo trì), đặc biệt với các phần mềm có tuổi thọ cao. Bảo trì nâng tập trung vào các thay đổi liên quan đến việc nâng cấp, hiệu chỉnh các tính năng & khả năng sử dụng của phần mềm bằng cách tinh chỉnh, xóa, thêm các tính năng mới.
4) **Bảo trì phòng ngừa (Preventative Maintenance)**: trọng tâm của loại bảo trì là ngăn chặn sự xuống cấp của phần mềm khi nó bị thay đổi trong một thời gian dài. Bảo trì phòng ngừa giúp phần mềm trở nên ổn định, dễ hiểu, dễ bảo trì hơn, và duy trì năng lực của nó trong việc chịu đựng các thay đổi theo thời gian. Các hoạt động bảo trì có thể bao gồm: tối ưu hóa mã; cập nhật tài liệu (nếu cần); tái cấu trúc, …Theo một báo cáo được công bố trên transcendent , bất kỳ doanh nghiệp nào cũng có thể tiết kiệm 12-18% bằng cách đầu tư vào bảo trì phòng ngừa thay vì bảo trì theo phản ứng của người dùng.

6.2 Quy trình triển khai phần mềm

Quy trình triển khai phần mềm thường trải qua sáu bước (xem Hình 6.2) như sau:

Bước 1: Lập kế hoạch triển khai

Bước 2: Xác định mục tiêu rõ ràng và thiết kế triển khai

Bước 3: Đào tạo nhóm triển khai phần mềm, đào tạo người dùng sử dụng/cấu hình sản phẩm

Bước 4: Kiểm thử triển khai sau khi khởi chạy hệ thống trên môi trường mô phỏng và lập báo cáo

Bước 5: Bắt đầu triển khai

Bước 6: Người dùng trải nghiệm hệ thống: Nếu khách hàng & người dùng cuối không hưởng lợi được lợi ích gì từ hệ thống/các thay đổi mới, việc triển khai phần mềm xem như là thất bại.

Hình 6.2: Các bước triển khai phần mềm

Một số công cụ hỗ trợ triển khai phần mềm:

- UserGuiding: hướng dẫn người dùng trải nghiệm lần đầu tiên với phần mềm của bạn, giúp họ làm quen bằng cách yêu cầu họ cung cấp các đầu vào trong suốt quá trình vận hành các chức năng của hệ thống và theo dõi các hoạt động vận hành của người dùng
- Flourish giúp việc thu thập và phân tích phản hồi từ người dùng cuối về sản phẩm. Theo dõi việc sử dụng phần mềm của họ để phát hiện sớm các vấn đề về vận hành.
- Optimizely là một công cụ phân tích và thử nghiệm A / B. Cho phép kiểm tra các chức năng của sản phẩm có hoạt động hay không mà không cần phải đợi đến toàn bộ hệ thống khởi chạy; giúp phân tích chức năng, xem cách thức người dùng tương tác với chức năng đó và thực hiện các thay đổi nếu cần. Bằng cách này, nhóm phát triển có thể tự tin vào việc triển khai các chức năng của họ.

6.3 Quy trình bảo trì phần mềm

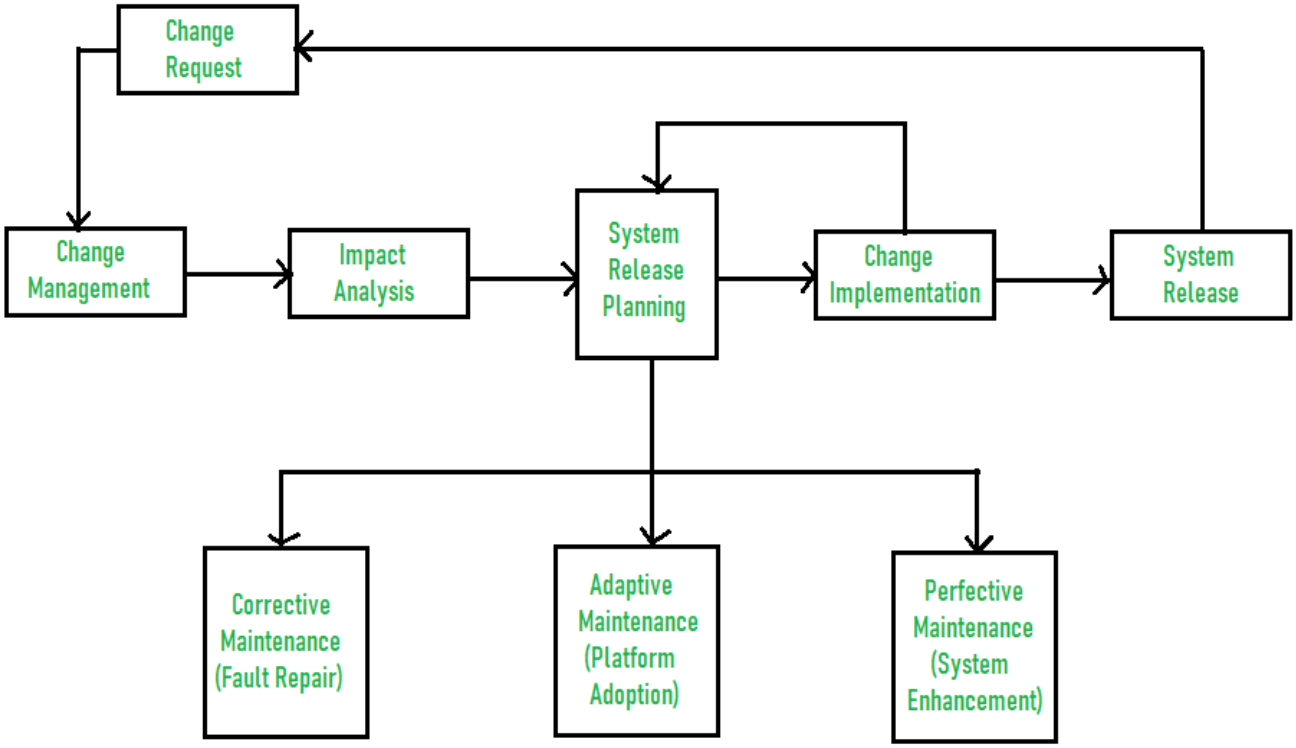
Hoạt động bảo trì phần mềm thường được kích hoạt khi có các yêu cầu thay đổi/phản ứng từ phía người dùng sản phẩm, hoặc định kỳ bảo trì dựa trên các kết quả giám sát sự vận hành của phần mềm của người dùng.

Quy trình bảo trì thường được tiến hành theo các bước:

1. Quản lý các yêu cầu thay đổi (Change Request);
2. Phân tích ảnh hương (Inpact Analysis): Phân tích những tác động của sự thay đổi đến những thành phần của hệ thống.

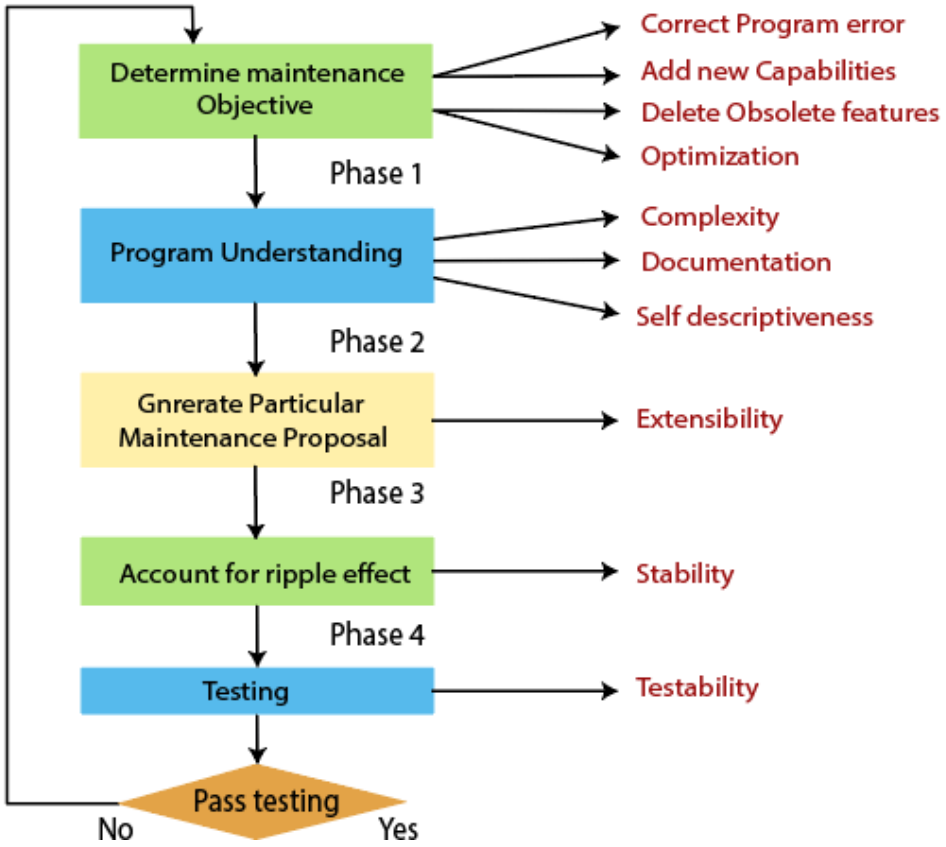
3. Lập kế hoạch phát hành hệ thống (System Release Planning): xác định các mục tiêu bảo trì, xác định loại hình bảo trì (sửa lỗi? thích ứng? hoàn thiện?); hiểu hệ thống; xây dựng giải pháp triển khai; xử lý các hiệu ứng lề; & kế hoạch kiểm thử hệ thống sau khi bảo trì. Quy trình lập kế hoạch được tiến hành theo các bước như Hình 6.4.
4. Triển khai các thay đổi (Change Implementation): thực hiện các thay đổi hệ thống theo kế hoạch đã lập
5. Phát hành hệ thống (System Release): đóng gói, phát hành, triển khai hệ thống/các tính năng được cập nhật đến môi trường vận hành của người dùng.

Chu trình lại tiếp tục cho lần bảo trì tiếp theo (chi tiết, xem Hình 6.3).



Hình 6.3: Quy trình bảo trì phần mềm

Quy trình lập kế hoạch phát hành/bảo trì hệ thống:

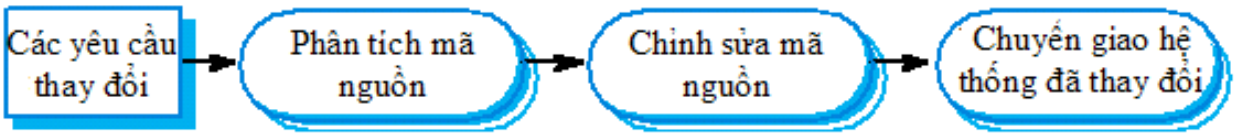


Hình 6.4: Quy trình lập kế hoạch bảo trì phần mềm

Đối với những yêu cầu thay đổi khẩn cấp (hotfixs), quy trình bảo trì khẩn cấp sẽ được áp dụng mà không cần phải trải qua đầy đủ các pha như quy trình bảo trì thông thường. Một số tình huống cần bảo trì khẩn cấp:

- Có lỗi hệ thống nghiêm trọng xảy ra và cần phải sửa chữa ngay.
- Các thay đổi về môi trường gây ra tác động làm phần mềm vận hành không mong đợi.
- Các thay đổi nghiệp vụ yêu cầu phải có đáp ứng nhanh.

Quy trình bảo trì khẩn cấp được chỉ ra trong Hình 6.5 Tuy nhiên, chúng ta nên hạn chế áp dụng quy trình này vì nó có thể dẫn đến phần mềm khó theo dõi, khó hiểu và khó bảo trì trong tương lai.



Hình 6.5: Quy trình cài đặt thay đổi khẩn cấp

Bảo trì phần mềm giữ một vai trò quan trọng trong việc duy trì thời gian sống & tính hữu ích của các phần mềm. DevOps là một giải pháp tối ưu giám sát vận hành; lập kế hoạch bảo trì phần mềm; phát hành và tích hợp ứng dụng liên tục (CI/CD); tăng tính tự động, tốc độ phát hành dịch vụ; đảm bảo chất lượng và sự hài lòng của khách hàng đối với chất lượng dịch vụ bảo trì.

Có nhiều chiến lược bảo trì có thể được áp dụng. Trong các chiến lược bảo trì này thì Tái kỹ nghệ (Re-Engineering) là một trong các chiến lược hiệu quả trong việc bảo trì các hệ thống di sản. Tái kỹ nghệ hệ thống là tiến trình bảo trì hệ thống bởi tiến trình gồm 2 giai đoạn: kỹ nghệ ngược, sau đó là kỹ nghệ tiến nhằm tái sử dụng lại các thành phần của hệ thống di sản và tạo ra hệ thống mới có kiến trúc tốt hơn, chất lượng cao hơn, dễ bảo trì hơn, … với các chức năng không đổi.

6.4 Công cụ, kỹ thuật trợ giúp

Bảo trì phần mềm có thể xem như bao gồm tất cả các hoạt động được tiến hành sau khi phát hành, triển khai phần mềm đến môi trường vận hành của người dùng nhằm duy trì tuổi thọ và tính hữu ích của phần mềm

Các công cụ hỗ trợ bảo trì khá phong phú, ví dụ: hỗ trợ hiểu mã trước khi bảo trì như: gỡ rối code (code debuggers), duyệt code (code browsers); hỗ trợ dịch ngược mã (decompilers); phân tích mã nguồn (code analysis); ghi lại nhật ký vận hành phần mềm (logging) & báo lỗi tự động (ví dụ: Bugzilla để theo dõi lỗi)); kiểm thử hồi quy; và quản lý cấu hình phần mềm. Mục này sẽ tập trung giới thiệu một số công cụ, kỹ thuật hỗ trợ bảo trì gồm:

- Các công cụ kỹ nghệ ngược;
- Chuỗi công cụ hỗ trợ DevOps Framework

6.4.1 Tái kỹ nghệ, kỹ nghệ ngược, kỹ nghệ tiến

Tái kỹ nghệ phần mềm (Software Re-Engineering) là tiến trình cải tiến khả năng bảo trì của sản phẩm bằng cách chuyển dịch nó sang dạng biểu diễn mới trong khi vẫn đảm bảo tính nguyên vẹn của các chức năng phần mềm.

Theo định nghĩa kỹ thuật, tái kỹ nghệ được xem là tiến trình kiểm tra và biến đổi hệ thống để hoàn nguyên nó sang một dạng biểu diễn mới. Tiến trình này nhấn mạnh đến sự kết hợp của các quy trình con như:

- Kỹ nghệ đảo ngược (Reverse Engineering);

- Tái cấu trúc (Restructing);
- Tái tự liệu (Redocumentation);
- Kỹ nghệ tiến (Forward Engineering); và
- Tái nền tảng vận hành (Retargeting): chuyển dịch phần mềm sang nền tảng hệ thống mới.

Một số công cụ hỗ trợ kỹ nghệ ngược gồm:

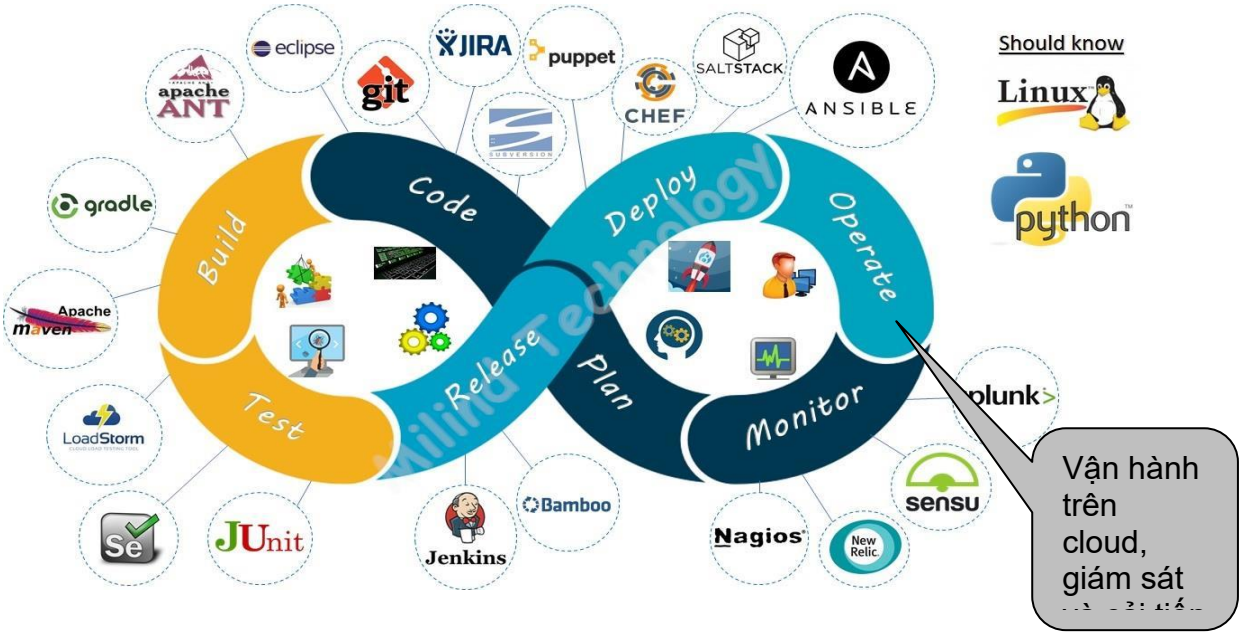
✓ Từ mã nhị phân (bộ cài đặt phần mềm, gói phần mềm sau khi biên dịch và đóng gói) dịch ngược lại mã nguồn dự án:
  - *Dotpeek (.NET), Hex-Rays (C));*
  - *JD-Eclipse (Java); JDCore service (Java)*

✓ Từ mã nguồn sinh ngược lại biểu đồ lớp đối tượng (áp dụng cho Java code)
  - *ObjectAid UML Explorer (plugin vào Eclipse IDE để sử dụng);*
  - *EasyUML (cho NetBeans)*

✓ Từ cơ sở dữ liệu quan hệ sinh ngược các lớp thực thể (Entity classes, Java):
  - *JPA/Hibernate Framework*

6.4.2 DevOps Framework – Chuỗi công cụ trợ giúp

Với DevOps, phần mềm thường được phát triển dưới dạng các dịch vụ. DevOps áp dụng các cách tiếp cận thực tế tốt nhất nhằm tự động hoá mọi giai đoạn trong quy trình (xem Hình 6.6).



Hình 6.6: Khung DevOps

Chuỗi các công cụ hỗ trợ DevOps gồm:

- Tools quản lý mã nguồn, xét duyệt mã nguồn tự động: Git, GitHub, Bitbucket, …
- Tools quản lý build (quản lý phụ thuộc, biên dịch và đóng gói): Maven, Ant, Make, MsBuild, …
- Tools kiểm thử tự động: Junit, Selenium, Cucumner, ..
- Tools quản lý kho cấu hình: Nexus, Artifactory, ..,
- Tools tích hợp liên tục (CI): Jenkins, Bamboo, TeamCity, …
- Tools giám sát cấu hình: Chef, Puppet, Ansible, …
- Tools quản lý phát hành: Visusal Studio, StackStorm, ..,
- Dịch vụ kho chứa hosting/đám mây: AWS, Azure, Docker, …
- Tools quản lý triển khai: Rapid Deploy, Code Deploy, …
- Tools hỗ trợ làm việc cộng tác: Jira, Team Foundation, Slack, …
- Tools giám sát vận hành: Kibana, Nagios, …
- Tools theo dõi, loggging và báo lỗi tự động: Splunk, Logstash, …

TÀI LIỆU THAM KHẢO

- Giáo trình chính

[1] Ian Sommerville (2015), *Software Engineering,* 9th Edition, Addison – Wesley.

[2] Bộ môn Công nghệ Phần mềm, Khoa Công nghệ thông tin, Đại học Công nghệ Thông tin và Truyền thông, Đại học Thái Nguyên (2022), *Bài giảng Nhập môn công nghệ phần mềm (Lưu hành nội bộ)*.

- Tài liệu tham khảo

[3] Ivan Marsic (2012), *Software Engineering,* Rutgers University, New Brunswick, New Jersey.

[4] Rajib Mall (2014), *Fundamentals of Software Engineering, Fourth Edition,* PHI Learning Private Limi ted, Delhi.

[5] Eric J. Braude and Michael E. Bernstein (2016), *Software Engineering - Modern Approaches, Second Edition,* Waveland Press, Inc.

[6] Len Bass, Paul Clements, Rick Kaman (2015) *Software Architecture in Practice* (3rd), Addison - Wesley.

[7] https://en.wikipedia.org/wiki/History_of_software_engineering

[8] http://catless.ncl.ac.uk/Risks/).

[9]https://www.techsignin.com/3-xu-huong-dinh-hinh-cong-nghe-phan-mem-nam-2020/

[10] https://en.wikipedia.org/wiki/IEEE_Standards_Association

[11] https://shecancode.io/blog/my-software-engineer-roadmap-by-joana-carneiro

[12] https://itviec.com

[13] https://github.com/MunGell/awesome-for-beginners

[14] https://personal.utdallas.edu/~chung/RE/IEEE830-1993.pdf

[15]https://sceweb.uhcl.edu/helm/RUP_course_example/courseregistrationproject/indexcourse.htm

[16]https://sceweb.uhcl.edu/helm/RUP_course_example/courseregistrationproject/indexcourse.htm

[17] VADAPALLI, Sricharan. *DevOps: continuous delivery, integration, and deployment with DevOps: dive into the core DevOps strategies*. Packt Publishing Ltd, 2018.

[18] Võ Đình Hiếu (2016), *Giáo trình kiến trúc hướng dịch vụ,* NXB Đại học Quốc gia Hà Nội

[19] http://www.softwaretestinghelp.com/what-is-stlc-v-model/

[20] Capstone Project Document - SPCS GROUP (2018), Sharing platform for cleaning services, FPT University.