NHẬP MÔN
CÔNG NGHỆ PHẦN MỀM

TÀI LIỆU

DÀNH CHO SINH VIÊN CÔNG NGHỆ THÔNG TIN

TRẦN ĐÌNH QUẾ

GIỚI THIỆU

Mục tiêu của môn Công nghệ phần mềm là cung cấp cho sinh viên những kiến thức cơ bản về tất cả mọi hoạt động liên quan đến phát triển phần mềm và kiến thức cơ bản về UML trong phát triển phần mềm. Qua môn học này sinh viên có kỹ năng sử dụng công cụ phần mềm để thực hiện các pha trong quá trình phát triển phần mềm và qua đó nâng cao năng lực làm việc nhóm và kỹ năng mềm. Sinh viên tham dự lớp và thực hành đầy đủ đặc biệt tích cực tham gia thảo luận trình bày trên lớp là yêu cầu quan trọng.

Nội dung bao gồm các kiểu hệ thống thông tin, các mô hình phát triển phần mềm, lập kế hoạch và quản lý dự án; các pha phát triển phần mềm từ xác định yêu cầu, phân tích, thiết kế đến lập trình – tích hợp; các kiến thức cơ bản về mô hình phần mềm với UML.

Mở đầu: Các đặc trưng của phần mềm**;** Các dạng phần mềm**;** Các hoạt động trong phát triển phần mềm**;** Tiến hóa trong phát triển phần mềm

Chương 2: Các pha trong phát triển phần mềm.

Các tác nhân trong quá trình phát triển phần mềm**;** Pha xác định yêu cầu**;** Pha phân tích**;** Pha thiết kế**;** Pha cài đặt và tích hợp**;** Pha bảo trì

Chương 3: Các mô hình vòng đời phần mềm

Mô hình xây - sửa; Mô hình thác nước; Mô hình bản mẫu nhanh; Mô hình tiến hoá; Mô hình RUP; Mô hình xoắn ốc; So sánh các mô hình

Chương 4: Kiểm chứng

Vấn đề chất lượng phần mềm; Kiểm chứng phần mềm; Các phương pháp kiểm chứng; Công cụ kiểm chứng

Chương 5: Lập kế hoạch và ước lượng

Vấn đề lập kế hoạch và ước lượng dự án phần mềm; Ước lượng thời gian và chi phí; Các thành phần của việc lập kế hoạch dự án phần mềm; Lập kế hoạch cho các dự án phần mềm hướng đối tượng.

Chương 6: Xác định yêu cầu

Các kỹ thuật xác định yêu cầu; Bản mẫu nhanh; Đặc tả dựa trên bản mẫu nhanh; Sử dụng lại bản mẫu; Đặc tả với bản mẫu; Kiểm thử pha yêu cầu

Chương 7: Các phương pháp phân tích truyền thống

Viết tài liệu pha đặc tả; Đặc tả phi hình thức; Các kỹ thuật đặc tả nửa hình thức; Mô hình quan hệ thực thể; Máy trạng thái hữu hạn; Các kỹ thuật đặc tả hình thức; So sánh các kỹ thuật đặc tả; Kiểm thử pha đặc tả

Chương 8: Phân tích hướng đối tượng

Tổng quan về phân tích hướng đối tượng; Mô hình use case; Mô hình lớp; Mô hình hành động; Kiểm thử pha phân tích hướng đối tượng

Chương 9: Thiết kế

Tổng quan về pha thiết kế; Thiết kế hướng hành động; Phân tích dòng dữ liệu; Thiết kế hướng đối tượng

Chương 10: Cài đặt và tích hợp

Các phương pháp cài đặt và tích hợp; Kiểm thử pha cài đặt và tích hợp; Kiểm thử sản phẩm; Kiểm thử chấp nhận

Chương 11: Bảo trì

Pha bảo trì**;** Bảo trì hệ phần mềm hướng đối tượng

TÀI LIỆU THAM KHẢO

[1] Object-Oriented and Classical Software Engineering, Stephen R. Schach, Seventh Edition, Mc Graw Hill, 2008.

[2] Giáo trình nhập môn UML, Huỳnh Văn Đức, Hoàn Thiện Ngân, NXB Lao động Xã hội, 2003.

MỤC LỤC

MỤC LỤC **3**
CHƯƠNG 1: MỞ ĐẦU **7**
1.2 CÁC KIỂU PHẦN MỀM 7
1.3 KHÍA CẠNH LỊCH SỬ 7
1.4 KHÍA CẠNH KINH TẾ 8
1.5 KHÍA CẠNH BẢO TRÌ 8
1.6 KHÍA CẠNH PHÂN TÍCH VÀ THIẾT KẾ 9
1.7 KHÍA CẠNH LẬP TRÌNH NHÓM 10
1.8 PHƯƠNG PHÁP HƯỚNG ĐỐI TƯỢNG 10
CHƯƠNG 2: CÁC PHA PHÁT TRIỂN PHẦN MỀM **13**
2.1 TIẾN TRÌNH THÀNH PHẦN 13
2.2 SQA LÀ GÌ? 14
2.3 PHA YÊU CẦU 14
2.4 PHA ĐẶC TẢ 14
2.5 PHA THIẾT KẾ 15
2.6 PHA CÀI ĐẶT 16
2.7 TÍCH HỢP 16
2.8 CẢI TIẾN TIẾN TRÌNH PHẦN MỀM 16
CHƯƠNG 3 ..... **21**
CÁC MÔ HÌNH VÒNG ĐỜI PHẦN MỀM **21**
3.1 PHÁT TRIỂN PHẦN MỀM 21
3.1.1 Theo lý thuyết phát triển phần mềm: 21
3.1.2 Thực tế phát triển phần mềm 21
3.1.3 Bài toán Winburg Mini 21
3.2 MÔ HÌNH XÂY SỬA 22
3.3 MÔ HÌNH TIẾN HÓA 22
3.4 MÔ HÌNH BẢN MẪU NHANH 23
3.5. MÔ HÌNH LẶP VÀ TĂNG 24
3.6 MÔ HÌNH UP 28
3.7 MÔ HÌNH XOẮN ỐC 33
3.8 MÔ HÌNH MÃ NGUỒN MỞ 34
CHƯƠNG 4: KIỂM THỬ **37**
4.1 VẤN ĐỀ CHẤT LƯỢNG PHẦN MỀM 37
4.1.1 Đảm bảo chất lượng phần mềm (SQA) 37
4.1.2. Độc lập trong quản lý 37
4.2 KIỂM CHỨNG PHẦN MỀM 38
4.3 CÁC PHƯƠNG PHÁP KIỂM CHỨNG 38
4.3.1. Kiểm thử không có sự thực thi 38
4.3.2 Kiểm thử có dựa trên sự thực thi 42
4.4 NHỮNG VẤN ĐỀ TRONG KIỂM THỬ 42
4.4.1 Cái gì nên được kiểm thử? 42
4.4.2 Kiểm thử và sự kiểm chứng tính chính xác 44
4.4.3 Sự kiểm chứng tính chính xác và kỹ nghệ phần mềm 45
4.4.4 Ai thực hiện kiểm thử phần mềm 46
4.4.5 Khi nào kiểm thử dừng 46

CHƯƠNG 5: LẬP KẾ HOẠCH VÀ ƯỚC LƯỢNG.....48
5.1 VẤN ĐỀ LẬP KẾ HOẠCH VÀ ƯỚC LƯỢNG DỰ ÁN PHẦN MỀM 48
5.2 ƯỚC LƯỢNG THỜI GIAN VÀ CHI PHÍ..... 49
5.2.1 Thước đo kích cỡ của sản phẩm phần mềm 49
5.2.2 Các kỹ thuật ước lượng chi phí 52
5.2.3 COCOMO trung gian 53
5.2.4 COCOMO II..... 55
5.2.5 Theo dõi ước lượng thời gian và chi phí 55
5.3 CÁC THÀNH PHẦN CỦA VIỆC LẬP KẾ HOẠCH PHÁT TRIỂN PHẦN MỀM 57
5.3.1Khung kế hoạch quản lý dự án phần mềm(SPMP) 57
5.3.2 Kế hoạch quản lý dự án phần mềm IEEE..... 57
5.3.3 Việc lập kế hoạch kiểm thử..... 58
5.3.4 Yêu cầu đào tạo 58
5.3.5 Các chuẩn tài liệu 58
5.3.6 Công cụ CASE cho việc lập kế hoạch và ước lượng..... 59
5.4 KIỂM THỬ VIỆC LẬP KẾ HOẠCH..... 59
5.5 LẬP KẾ HOẠCH CHO CÁC DỰ ÁN PHẦN MỀM HƯỚNG ĐỐI TƯỢNG 59
CHƯƠNG 6: PHA XÁC ĐỊNH YÊU CẦU.....60
6.1 XÁC ĐỊNH YÊU CẦU CỦA KHÁCH HÀNG..... 60
6.2 TỔNG QUAN VỀ LUỒNG CÔNG VIỆC XÁC ĐỊNH YÊU CẦU 60
6.2.1 Hiểu lĩnh vực ứng dụng 61
6.2.2 Mô hình nghiệp vụ..... 61
6.2.3 Các use case..... 63
6.2.4 Các yêu cầu ban đầu..... 65
6.3 PHA XÁC ĐỊNH YÊU CẦU CỔ ĐIỂN 65
6.4 BẢN MẪU NHANH 66
6.5 NHÂN TỐ CON NGƯỜI..... 66
6.6 SỬ DỤNG LẠI BẢN MẪU NHANH 67
6.7 CÁC CÔNG CỤ CASE CHO XÁC ĐỊNH YÊU CẦU 67
6.8 CÁC THƯỚC ĐO CHO XÁC ĐỊNH YÊU CẦU..... 68
6.9 NHỮNG THỬ THÁCH CHO PHA XÁC ĐỊNH YÊU CẦU..... 68
6.10 CASE STUDY CHO PHA XÁC ĐỊNH YÊU CẦU..... 68
6.10.1 Bài toán..... 68
6.10.2 Xây dựng sơ đồ use case tổng quan..... 70
6.10.3 Mô tả các use case 71
CHƯƠNG 7: CÁC PHƯƠNG PHÁP PHÂN TÍCH TRUYỀN THỐNG.....75
7.1 YÊU CẦU TÀI LIỆU ĐẶC TẢ..... 75
7.2 CÁC PHƯƠNG PHÁP ĐẶC TẢ..... 76
7.2.1 Đặc tả phi hình thức..... 76
7.2.2 Phân tích hướng cấu trúc 77
CHƯƠNG 8: PHƯƠNG PHÁP PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG 79
8.1 LUỒNG CÔNG VIỆC PHÂN TÍCH 79
8.2 VIỆC TRÍCH RÚT CÁC LỚP THỰC THỂ 79
8.3 PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG CHO BÀI TOÁN THANG MÁY 80
8.4 MÔ HÌNH HÓA CHỨC NĂNG 80
8.5 MÔ HÌNH HÓA LỚP THỰC THỂ..... 82
8.5.1 Trích rút danh từ 82

8.5.2 CRC Cards 84
8.7 KIỂM THỬ TRONG PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG 86
8.8 CÁC CÔNG CỤ CASE CHO PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG 90
8.9 CASE STUDY CHO PHA PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG 90
8.9.1. Các scenario 90
8.9.2 Trích các lớp thực thể 94
8.9.3 Viết lại các scenario 96
CHƯƠNG 9: PHA THIẾT KẾ 97
9.1 TỔNG QUAN VỀ PHA THIẾT KẾ 97
9.1.1 Dữ liệu và các hành động 97
9.1.2 Thiết kế và trừu tượng 97
9.1.3 Thiết kế 98
9.1.4 Kiểm thử trong pha thiết kế 99
9.1.5 Kỹ thuật hình thức cho thiết kế chi tiết 100
9.1.6 Kỹ thuật thiết kế hệ thống thời gian thực 100
9.1.7 Công cụ CASE cho thiết kế 101
9.1.8 Thước đo cho thiết kế 101
9.1.9 Những thách thức của pha thiết kế 102
9.2 THIẾT KẾ HƯỚNG HÀNH ĐỘNG 102
9.3 PHÂN TÍCH VA THIẾT KẾ DÒNG DỮ LIỆU 103
9.3.1 Phân tích dòng dữ liệu 103
9.3.2 Thiết kế dòng dữ liệu 108
9.4 THIẾT KẾ HƯỚNG ĐỐI TƯỢNG 108
9.5 CASE STUDY CHO PHA THIẾT KẾ 111
9.5.1 Thiết kế cơ sở dữ liệu 112
9.5.2 Thiết kế dùng bean thay cho control 114
9.5.3 Thiết kế dùng control DAO và thực thể thuần 122
9.5.4 Thiết kế theo MVC cải tiến, dùng control DAO và thực thể thuần 129
CHƯƠNG 10: PHA CÀI ĐẶT VÀ TÍCH HỢP 136
10.1 CÁC PHƯƠNG PHÁP CÀI ĐẶT VÀ TÍCH HỢP 136
10.1.1 Luồng công việc cài đặt 136
10.1.2 Tích hợp 145
10.2 KIỂM THỬ PHA CÀI ĐẶT VÀ TÍCH HỢP 149
10.2.1 Luồng công việc kiểm thử cài đặt 149
10.2.2 Kiểm thử tích hợp 162
10.4 KIỂM THỬ CHẤP NHẬN 163
10.5 CASE STUDY CHO PHA CÀI ĐẶT: VIẾT TEST CASE 163
10.5.1 Test case cho modul thêm phòng mới của khách sạn 163
10.5.2 Test case cho modul sửa thông tin phòng của khách sạn 166
10.5.3 Test case cho modul đặt phòng 168
CHƯƠNG 11: PHA BẢO TRÌ 177
11.1 PHA BẢO TRÌ SAU KHI CHUYỂN GIAO 177
11.1.1 Tại sao bảo trì sau khi chuyển giao là cần thiết 177
11.1.2 Người lập trình bảo trì sau khi chuyển giao yêu cầu những gì? 177
11.1.3 Quản lý bảo trì sau khi chuyển giao 179
11.1.4 Bảo trì sau khi chuyển giao với kỹ năng phát triển 181
11.1.5 Kỹ nghệ ngược 181

11.1.6 Công cụ CASE cho bảo trì sau khi chuyển giao..... 182
10.1.7 Thước đo của bảo trì sau khi chuyển giao..... 182
10.1.8 Những thách thức của bảo trì sau khi chuyển giao..... 182
11.2 BẢO TRÌ HỆ PHẦN MỀM HƯỚNG ĐỐI TƯỢNG..... 183
11.3 KIỂM THỬ PHA BẢO TRÌ..... 184

CHƯƠNG 1: MỞ ĐẦU

1.1 ĐẶC TRƯNG CỦA PHẦN MỀM

- Phần mềm không mòn
- Phần mềm được phát triển mà không được sản xuất theo nghĩa thông thường
- Mặc dù công nghiệp phần mềm đang hướng đến phát triển dựa trên thành phần nhưng phần lớn phần mềm phát triển dựa theo yêu cầu của khách hàng.
- Cho đến nay những đặc trưng của phần mềm vẫn còn là vấn đề tranh cãi. Chính điều này thể hiện sự chưa trưởng thành của ngành học CÔNG NGHỆ PHẦN MỀM.

1.2 CÁC KIỂU PHẦN MỀM

- **Phần mềm hệ thống:** Tập hợp các Chương trình được viết để phục vụ các chương trình khác, tương tác với phần cứng (ví dụ, biên dịch, trình soạn thảo, HĐH…)
- **Phần mềm thời gian thực:** Phần mềm điều phối/phân tích kiểm soát, đáp ứng thời gian
- **Phần mềm nghiệp vụ:** Các phần mềm tính lương, kế toán, quản lý kho…
- **Phần mềm khoa học và công nghệ.** Các ứng dụng trong thiên văn, sinh học phân tử, điều khiển tàu con thoi,…
- **Phần mềm nhúng:** Nằm trong bộ nhớ chỉ đọc, điều khiển các sản phẩm và hệ thống
- **Phần mềm máy tính cá nhân:** Xử lý văn bản, đồ họa máy tính…
- **Phần mềm trên Web**
- **Phần mềm trí tuệ nhân tạo:** Dựa trên những kỹ thuật của Trí tuệ nhân tạo như hệ chuyên gia.

Nhận xét: Hiện nay web được xem là môi trường phổ biến để xây dựng giao diện với người sử dụng cho nhiều hệ thống phần mềm trên mạng.

1.3 KHÍA CẠNH LỊCH SỬ

- Năm 1967 nhóm NATO đưa ra thuật ngữ Công nghệ phần mềm (Software Engineering).
- Năm 1968 Hội nghị Software Engineering ở Garmisch, Đức đưa ra mục đích là giải quyết "Cuộc khủng hoảng phần mềm":
  - Phần mềm hoàn thành không đúng thời hạn
  - Chi phí vượt dự toán ban đầu
  - Phần mềm còn nhiều lỗi
- Tại sao không thể sử dụng kỹ nghệ xây cất như xây dựng cầu để xây dựng các hệ điều hành?
  - Thái độ đối với việc sập cầu/hệ điều hành
  - Thông tin về CNPM thường không đầy đủ, không chắc chắn
  - Độ phức tạp

- Bảo trì

Công nghệ phần mềm không thể xem giống như các kỹ nghệ thông thường khác.

1.4 KHÍA CẠNH KINH TẾ

- CNPM và khoa học máy tính (tương tự như hóa học và kỹ nghệ hóa)
  - Khoa học có phần thực nghiệm và lý thuyết: Phần thực nghiệm của Hóa học là thí nghiệm còn của khoa học máy tính là lập trình.
  - Khoa học máy tính nghiên cứu những sách khác nhau để tạo ra phần mềm. Nhưng kỹ sư phần mềm chỉ quan tâm kỹ thuật có ý nghĩa kinh tế.
  - Ví dụ: Phương pháp mã hóa mới KTmới (lập trình hướng thành phần) nhanh hơn phương pháp đang sử dụng hiện thời KTcũ (lập trình hướng đối tượng) là 10%. Chúng ta nên sử dụng phương pháp mới hay không?
    Câu trả lời thông thường là: Tất nhiên! Câu trả lời Công nghệ phần mềm: xét hiệu quả của KTmới.

1.5 KHÍA CẠNH BẢO TRÌ

- Vòng đời phần mềm: Một loạt các pha mà phần mềm phải trải qua từ khám phá các khái niệm đến loại bỏ hoàn toàn.
- Mô hình vòng đời
  - Pha yêu cầu
  - Pha đặc tả
  - Pha thiết kế
  - Pha cài đặt
  - Pha tích hợp
  - Pha bảo trì
  - Loại bỏ
- **Bảo trì:** Mọi thay đổi đối với sản phẩm một khách hàng đã đồng ý sản phẩm thỏa mãn tài liệu đặc tả.
- Phần mềm tốt được bảo trì khác với phần mềm tồi bị loại bỏ
- Các dạng bảo trì"
  - Bảo trì sửa lỗi [17,5%]: sửa chữa lỗi nhưng đặc tả không đổi
  - Bảo trì nâng cao: sửa chữa theo thay đổi của đặc tả. Bảo trì hoàn thiện [60,5%]: thêm chức năng để cải tiến sản phẩm. Bảo trì thích nghi [18%]: thay đổi phần mềm để đáp ứng thay đổi của môi trường như quy định chính phủ, CPU, công nghệ mới…

Ví dụ 1: Tỷ lệ thuế GTGT thay đổi từ 6% đến 7%

- C++

const float *salesTax =6.0;*

- JAVA

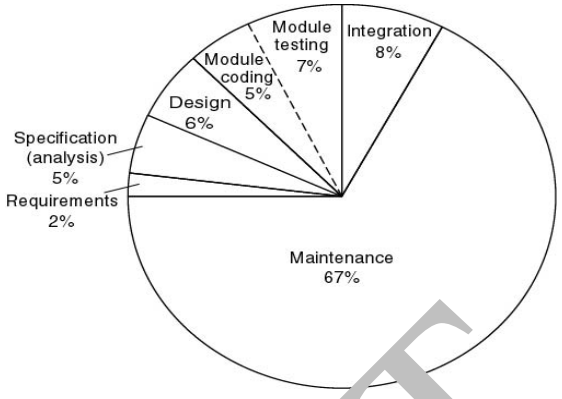
public static float *salesTax = 6.0;*

Ví dụ 2 Tổ chức y tế thay đổi ⇒ Hoạt động thay đổi

Ví dụ 3 Các hệ thời gian thực/hệ nhúng

Thế giới thực thay đổi ⇒ hệ thay đổi

- Chi phí cho các pha: Nguồn dữ liệu của các năm 1976-1981



- Hewlett-Packard (1992)
  - 60-80% nguồn nhân lực để phát triển và nghiên cứu dành cho bảo trì
  - 40-60% chi phí phần mềm dành cho bảo trì
- Nhiều tổ chức dành 80% thời gian và công sức chop ha bảo trì [Yourdon, 1996]

Kết luận: Bảo trì là pha tốn kém nhiều thời gian và chi phí

1.6 KHÍA CẠNH PHÂN TÍCH VÀ THIẾT KẾ

- 60 đến 70% lỗi của phần mềm là những lỗi do đặc tả và thiết kế
- Dữ liệu của Kelly, Sherif and Hops [1992] về chương trình không gian liên hành tinh của Nasa
  - 1,9 lỗi trên một trang đặc tả
  - 0,9 lỗi trên một trang thiết kế
  - 0,3 lỗi trên một trang chương trình nguồn
- Dữ liệu của Bhandari et al. [1994]: Lỗi trong cuối pha thiết kế của phiên bản mới của sản phẩm
  - 13% lỗi từ phiên bản trước
  - 16% lỗi trong pha đặc tả mới
  - 71% lỗi trong pha thiết kế mới

Kết luận: Xây dựng những kỹ thuật sinh ra thiết kế và đặc tả tốt hơn là một vấn đề quan trọng trong công nghệ phần mềm.

Chi phí cho việc phát hiện và sửa chữa lỗi:

1.7 KHÍA CẠNH LẬP TRÌNH NHÓM

Phần cứng rẻ, khả năng tăng, kích thước giảm

Phần mềm lớn, đắt. Nhiều phần mềm quá lớn nên một người không thể phát triển được trong thời gian có hạn.

Ví dụ: Lan và Minh viết code cho hai Mô đun p và q với mô đun p gọi q. Khi viết p Lan đã viết một hàm gọi q với 5 đối số. Minh cũng code q với 5 đối số nhưng thứ tự khác Lan.

- Compiler của C không thể phát hiện sai sót đó
- Java chỉ có thể phát hiện khi biến khác kiểu nhưng nếu cùng kiểu thì không thể phát hiện

1.8 PHƯƠNG PHÁP HƯỚNG ĐỐI TƯỢNG

Trước năm 1975, phần lớn các tổ chức phần mềm có những kỹ thuật riêng, cá nhân có cách làm việc riêng. Giữa những năm 1975-1985: Phương pháp cấu trúc đạt nhiều thành công ban đầu. Kỹ thuật cấu trúc phù hợp với chương trình 5.000 đến 50.000 dòng mã.

Hạn chế của phương pháp cấu trúc:

- Không phù hợp với phần mềm lớn (> 50.000 dòng lệnh)
- 80% chi phí và sức lực dành cho bảo trì nhưng cách tiếp cận của phương pháp cấu trúc không giải quyết được vấn đề này.

Đặc trưng của phương pháp cấu trúc:

- Hướng hành động (máy trạng thái hữu hạn, sơ đồ dòng dữ liệu); hay
- Hướng dữ liệu (sơ đồ quan hệ thực thể, phương pháp Jackson);
- Không cả hai

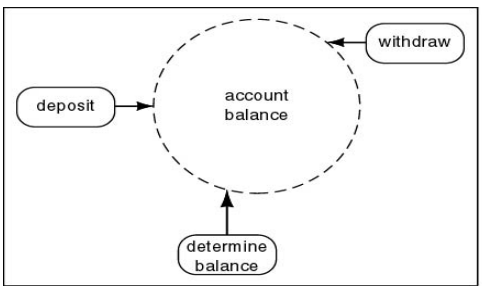
Phương pháp hướng đối tượng: Cả hai dữ liệu và hành động đều quan trọng như nhau

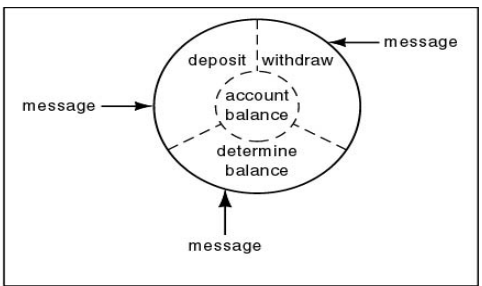
Đối tượng: Thành phần phần mềm kết hợp cả hai dữ liệu và các hành động thực hiện trên dữ liệu đó.

Ví dụ: Quản lý trương mục ngân hàng

- Dữ liệu: tiền gửi vào, rút ra
- Hành động: gửi vào, rút ra, xác định tiền gửi vào rút ra

Phương pháp lập trình hướng đối tượng khác phương pháp cấu trúc





- Ẩn dấu thông tin
- Thiết kế dựa trên trách nhiệm
- Ảnh hưởng đối với bảo trì và phát triển

- **Phương pháp hướng đối tượng:** Một đối tượng gửi message đến một đối tượng khác để yêu cầu hành động/phương pháp.
- **Từ phân tích đến thiết kế**

Structured Paradigm	**Object-OrientedParadigm**
1. Requirements phase	1. Requirements phase
2. Specification (analysis) phase	2′. Object-oriented analysis phase
3. Design phase	3′. Object-oriented design phase
4. Implementation phase	4′. Object-oriented programming phase
5. Integration phase	5. Integration phase
6. Maintenance phase	6. Maintenance phase
7. Retirement	7. Retirement

- Phương pháp cấu trúc: Tạo "cầu nối" giữa phân tích (what) và thiết kế "how"
- Phương pháp hướng đối tượng: (chuyển pha "êm dịu" nên giảm được sai sót) Các đối tượng được đưa vào ngay từ đầu của vòng đời. Đối tượng được đề xuất trong pha thiết kế, được xây dựng trong pha thiết kế, được mã hóa trong pha lập trình.
- Phân tích hệ thống: xác định điều cần phải làm WHAT
- Thiết kế xác định cách làm HOW: thiết kế kiến trúc xác định các mô đun và thiết kế chi tiết từng mô đun.

- **Lợi ích của HĐT:**
  - Phân tích hướng đối tượng: Xác định cái gì phải thực hiện và xác định các đối tượng
  - Thiết kế hướng đối tượng: Xác định cách thực hiện hành động và xây dựng các đối tượng.
- **So sánh phương pháp cấu trúc và phương pháp hướng đối tượng**

Structured Paradigm	Object-Oriented Paradigm
2. Specification (analysis) phase • Determine what the product is to do	2′. Object-oriented analysis phase • Determine what the product is to do • Extract the objects
3. Design phase • Architectural design (extract the modules) • Detailed design	3′. Object-oriented design phase • Detailed design
4. Implementation phase • Implement in appropriate programming language	4′. Object-oriented programming phase • Implement in appropriate object-oriented programminglanguage

Kết luận:

- Công nghệ phần mềm: ngành học nhằm nghiên cứu cách sản xuất ra phần mềm thỏa mãn nhu cầu khách hàng, không có lỗi, giao đúng thời hạn và trong chi phí hạn định.
- Công nghệ phần mềm xem xét các pha của vòng đời phần mềm và phân phối các khía cạnh khác nhau của tri thức con người từ kinh tế, kỹ thuật đến khoa học xã hội, tâm lý…
- Phương pháp hướng đối tượng được xem là cách tiếp cận phổ biến trong phát triểnphần mềm hiện nay.

CHƯƠNG 2: CÁC PHA PHÁT TRIỂN PHẦN MỀM

2.1 TIẾN TRÌNH THÀNH PHẦN

- Tiến trình phần mềm là "phương cách" sản xuất ra phần mềm. Tiến trình phần mềm nghiên cứu các cách kết hợp:
  - Mô hình vòng đời
  - Các công cụ CASE
  - Các cá nhân xây dựng phần mềm
  - Các công nghệ

  Tiến trình phần mềm = Khía cạnh kỹ thuật + Khía cạnh quản lý
- Các tổ chức khác nhau có những tiến trình phần mềm khác nhau.
  - Khâu viết tài liệu (quan trọng/không quan trọng)
  - Chi phí dành cho kiểm thử (1/2 chi phí/không quan trọng)
  - Tập trung khaua nghiên cứu, phát triển phần mềm. Khâu bảo trì dành cho người khác.
- Vì sao như vậy?
  - Thiếu kỹ năng về kỹ nghệ phần mềm
  - Nhiều người quản lý phần mềm thiếu kiến thức về bảo trì và phát triển phần mềm (do đó trễ hạn!)
  - Quan điểm về quản lý (giao đúng hạn hay kiểm tra kỹ trước khi giao)
  - Phụ thuộc vào cá nhân

Có pha kiểm thử không?

- KHÔNG có pha kiểm thử. Vì sao? Kiểm thử là hoạt động thực hiện trong mọi pha của sản xuất phần mềm.
- Check = test
- Testing: Thương được hiểu sau khi coding
- Kiểm tra (Varification): Thực hiện vào cuối mỗi pha
- Kiểm chứng (Validation): Thực hiện trước khi giao sản phẩm cho khách hàng

Có pha viết tài liệu không?

- KHÔNG có pha viết tài liệu
- Mọi pha phải được viết tài liệu trước khi khởi đầu một pha mới.
- Một số lý do:
  - Tài liệu bị hoãn lại thì sẽ không bao giờ hoàn thành
  - Cá nhân chịu trách nhiệm trong pha trước có thể đã chuyển sang bộ phận khác.

- Sản phẩm thường xuyên thay đổi trong khi phát triển vì thế ta cần tài liệu để ghi lại điều này. Ví dụ, thiết kế thường phải sửa đổi trong khi cài đặt. Việc sửa đổi như vậy chỉ có thể thực hiện được khi có tài liệu của nhóm thiết kế.

Kiểm thử và viết tài liệu được tiến hành trong mọi pha của tiến trình phần mềm.

2.2 SQA LÀ GÌ?

- Nhóm đảm bảo chất lượng phần mềm (SQA: Software quality assurance):
  - Có trách nhiệm đảm bảo sản phẩm được xây dựng đúng (theo đặc tả) và theo đơn đặt hàng.
  - Nhóm SQA phải đóng đúng vai trò ngay từ đầu của tiến trình và hoạt động trong mọi pha của tiến trình phần mềm.
  - SQA kiểm tra với khách hàng xem phiên bản cuối cùng thỏa mãn hoàn toàn chưa.

2.3 PHA YÊU CẦU

- Tiến trình phát triển phần mềm bắt đầu khi khách hàng tiếp xúc với công ty phần mềm và cho rằng: Phần mềm có khả năng thích hợp với khách hàng và giá thành hợp lý.
- Khám phá khái niệm:
  - Trong lần gặp đầu tiên, khách hàng phát hợ sản phẩm mà họ hình dung. Theo quan điểm của người phát triển, mô tả này không rõ ràng, không hợp lý, mâu thuẫn hay không thể xây dựng phần mềm như thế được.
  - Người phát triển xác định nhu cầu và ràng buộc của khách hàng

Kiểm thử pha yêu cầu:

- Làm bản mẫu nhanh: Bản mẫu phần mềm kết hợp nhiều chức năng của sản phẩm cuối cùng nhưng bỏ qua những khía cạnh mà khách hàng không thấy được như cập nhật file hay xử lý lỗi.
- Bản mẫu nhanh phải được kiểm tra bởi khách hàng và người sử dụng.
- Bản mẫu nhanh có thể thay đổi cho đến khi khách hàng và người sử dụng cho rằng nó có những chức năng mà họ mong muốn.

Viết tài liệu pha yêu cầu:

- Tài liệu pha yêu cầu:
  - Có bản mẫu: Bản mẫu nhanh. Bản ghi thỏa thuận với khách hàng và người sử dụng về cơ sở xây dựng và sửa đổi bản mẫu.
  - Không có bản mẫu (nhóm quyết định không xây dựng bản mẫu): Mô tả nhu cầu khách hàng. Tài liệu này phải được khách hàng, người sử dụng và nhóm phát triển kiểm tra trước khi nhóm SQA xem xét.

2.4 PHA ĐẶC TẢ

- Khi khách hàng cho rằng nhóm phát triển hiểu được yêu cầu, nhóm đặc tả viết tài liệu đặc tả để mô tả chức năng của sản phẩm (những gì sản phẩm cần có + ràng buộc).

- Đặc tả bao gồm những input của sản phẩm và output được yêu cầu
- Ví dụ: Khách hàng cần tính bảng lương thì input bao gồm mức trả cho mỗi nhân viên, thông tin từ hồ sơ cá nhân để tính thuế… và output là số lương thuế, chi bảo hiểm,…
- Yêu cầu của đặc tả
    - Không nhập nhằng: không sử dụng thuật ngữ như tiện lợi, đầy đủ chức năng, nhanh, 98%...
    - Đầy đủ: Thể hiện mọi yêu cầu của khách hàng
    - Phi mâu thuẫn: không chứa mâu thuẫn
    - Theo dõi được (Traceability): có thể lần theo phán đoán trong đặc tả trở lại phán đoán đưa ra bởi khách hàng trong pha yêu cầu. Nếu đặc tả được trình bày đúng phương pháp, có đánh chỉ số,… thì nhóm SQA sẽ ít gặp khó khăn. Nếu có bản mẫu thì phán đoán liên quan của đặc tả phải theo dõi được đến bản mẫu.
- Môt khi đặc tả được hoàn thành và đã thông qua thì hình thành kế hoạch quản lý quá trình sản xuất phần mềm (SPMP : The software product management plan).
- Yêu cầu kế hoạch:
    - SPMP cần nêu lên thời gian thực hiện, chi phí cho từng pha, gán trách nhiệm cá nhân cho từng pha, thời hạn hoàn thành cho mỗi pha.
    - Mô hình vòng đời nào sẽ sử dụng, cấu trúc tổ chức, kỹ thuật và CASE sử dụng, lịch tình, chi phí…

Kiểm thử pha đặc tả:

- Nguồn gốc chính của lỗi trong các phần mềm đã phân phối đến nay là những lỗi trong tài liệu đặc tả và những lỗi này chỉ được phát hiện khi tổ chức khách hàng sử dụng nó.
- Duyệt xét lại (Review) là cách tốt nhất để kiểm tra đặc tả. Mục đích là xác định đặc tả có đúng không. Nhóm SQA chủ trì cuộc họp với đại diện nhóm đặc tả và khách hàng.

2.5 PHA THIẾT KẾ

- Đặc tả - What
- Thiết kế - How
- Giữ lại quyết định thiết kế
    - Thời điểm kết thúc
    - Cho nhóm bảo trì
    - Thiết kế nên mở (open-ended)
- Thiết kế kiến trúc : Phân rã sản phẩm thành mô đun
- Thiết kế chi tiết: Thiết kế các mô đun: cấu trúc dữ liệu, thuật toán

Kiểm thử pha thiết kế

- Tài liệu viết sao cho dễ theo dõi
- Duyệt tài liệu

2.6 PHA CÀI ĐẶT

- Cài đặt thiết kế chi tiết thành chương trình

Kiểm thử pha cài đặt:

- Rà soát
- Các test case
  - Test không hình thức (desk checking)
  - Test hình thức (Formal testing) do nhóm SQA

2.7 TÍCH HỢP

- Kết hợp các mô đun và kiểm thử toàn bộ sản phẩm

Tài liệu pha tích hợp

- Mã nguồn có chú thích
- Các test cases

2.8 CẢI TIẾN TIẾN TRÌNH PHẦN MỀM

- Bắt đầu từ Bộ Quốc Phòng Mỹ
- Software Engineering Institue (SEI)
- Vấn đề:
  - Quản lý tiến trình phần mềm kém
- Cải tiến tiến trnfh phần mềm
  - Capability maturity model (CMM)
  - ISO 9000-Series
  - ISO/IEC 15504

CMM: Capability Maturity Model

- Không phải mô hình vòng đời
- Tập các chiến lược cải tiến tiến tình phần mềm
  - SW-CMM for software
  - P-CMM for human resource (“people”)
  - SE-CMM for systems engineering
  - IPD-CMM for integrated product development
  - SA-for software acquisition
- Các chiến lược được thống nhất thành CMMI (Capability maturity model integration)

SW-CMM

- A strategy for improving the software process
  - Put forward in 1986 by the SEI
  - Fundamental idea:
  - Improving the software process leads to
    - Improved software quality

        - Delivery on time, within budget
    - Improved management leads to
        - Improved techniques
- Five levels of "maturity" are defined
    - Organization advances stepwise from level to level

Level 1: Initial Level

- Ad hoc approach
    - Entire process is unpredictable
    - Management consists of responses to crises
- Most organizations world-wide are at level 1

Level 2: Repeatable Level

- Basic software management
    - Management decisions should be made on the basis of previous experience with similar products
    - Measurements ("metrics") are made
    - These can be used for making cost and duration predictions in the next project
    - Problems are identified, immediate corrective action is taken

Level 3: Defined Level

- The software process is fully documented
    - Managerial and technical aspects are clearly defined
    - Continual efforts are made to improve quality, productivity
    - Reviews are performed to improve software quality
    - CASE tools are applicable *now* (and not at levels 1 or 2)

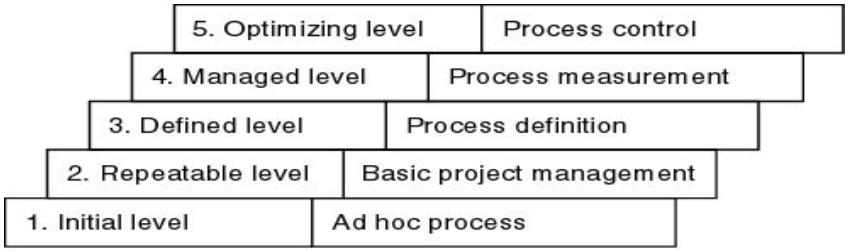
Level 4: Managed Level

- Quality and productivity goals are set for each project
    - Quality, productivity are continually monitored
    - Statistical quality controls are in place

Level 5: Optimizing level

- Continuous process improvement
    - Statistical quality and process controls
    - Feedback of knowledge from each project to the next

Kết luận:



Tiến trình chính

- Có các tiến trình chính (key Process Area) cho mỗi mức
- Mức 2:
  - Quản lý yêu cầu
  - Lập kế hoạch dự án
  - Theo dõi dự án
  - Quản lý cấu hình
  - Bảo đảm chất lượng
- So sánh
  - Mức 2: Phát hiện và sửa lỗi
  - Mức 3: Ngăn chặn lỗi

Kinh nghiệm

- Cần:
  - 3 đến 5 năm để từ mức 2 lên mức 3
  - 1.5 đến 3 năm từ mức 2 lên mức 3
- Original idea: Defense contracts would be awarded only to capable firms
- Profitability
  - Hughes Aircraft (Fullerton, CA) spent $500K (1987–90)
    - Savings: $2M per year moving from level 2 to level 3
  - Raytheon moved from level 1 in 1988 to level 3 in 1993
    - Productivity doubled
    - Return of $7.70 per dollar invested in process improvement

Other SPI Initiatives

- Other software process improvement (SPI) initiatives:
  - ISO 9000-series
  - ISO/IEC 15504

ISO 9000

- Set of five standards for industrial activities
  - ISO 9001 for quality systems
  - ISO 9000-3, guidelines to apply ISO 9001 to software
  - There is an overlap with CMM, but they are not identical
  - *Not* process improvement
  - Stress on documenting the process
  - Emphasis on measurement and metrics
  - ISO 9000 is required to do business with the E.U.
  - Also by many U.S. businesses, for example, GE
  - More and more U.S. businesses are ISO 9000 certified

ISO/IEC 15504

- Original name: Software Process Improvement Capability dEtermination (SPICE)
  - International process improvement initiative
  - Started by British Ministry of Defence (MOD)
  - Includes process improvement, software procurement
  - Extends and improves CMM, ISO 9000
  - Framework, not a method
    - CMM, ISO 9000 conform to this framework
  - Now referred to as ISO/IEC 15504
  - Or just 15504 for short

Process Improvement Data

Category	Range	Median	Number of Data Points
Years engaged in software process improvement (SPI)	1–9	3.5	24
Yearly cost of SPI per software engineer	$4[illegible]–$2004	$1375	5
Productivity gain per year	9%[illegible]7%	35%	4
Early defect detection gain per year	6%–2[illegible]	22%	3
Yearly reduction in time to market	15%–23%	19%	2
Yearly reduction in post-release defect re[illegible]ts	[illegible]0%–94%	39%	5
Business value (saving/cost of SPI)	[illegible].0–8.8:1	5.0:1	5

- SEI report on 13 org[illegible]izatio[illegible]s in the original study
- They used a variety of p[illegible]ess improvement techniques, not just SW–CMM

CMM Level	Number of Projects	[illegible]elative Decrease in Duration	Faults per MEASL Detected during Development	Relative Productivity
Level 1	3	1.0	—	—
Level 2	9	3.2	890	1.0
Level 3	5	2.7	411	0.8
Level 4	8	5.0	205	2.3
Level 5	9	7.8	126	2.8

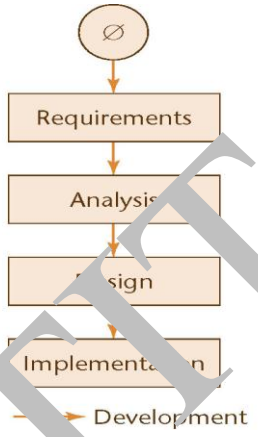
CHƯƠNG 3

CÁC MÔ HÌNH VÒNG ĐỜI PHẦN MỀM

3.1 PHÁT TRIỂN PHẦN MỀM

3.1.1 Theo lý thuyết phát triển phần mềm:

Theo lý tưởng, phần mềm được phát triển: tuyến tính và bắt đầu từ con số 0



3.1.2 Thực tế phát triển phần mềm

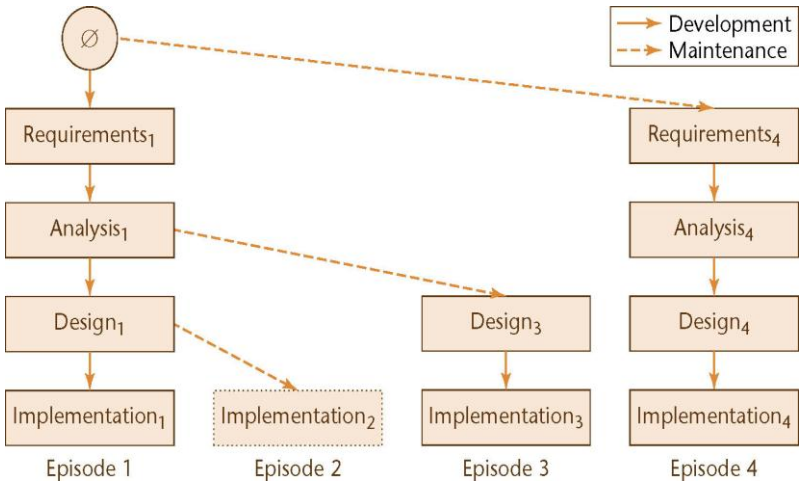
Trong thực tế, phát triển phần mềm hoàn toàn khác:

- Người phát triển có thể mắc lỗi trong quá trình phát triển phần mềm
- Yêu cầu của khách hàng thay đổi trong khi hệ thống phần mềm đang được xây dựng

3.1.3 Bài toán Winburg Mini

- **Episode 1**: Phiên bản đầu tiên được cài đặt
- **Episode 2:** Tìm ra các lỗi
  - Hệ thống phần mềm được xây dựng quá lâu bởi vì việc sử lỗi cài đặt
  - Thay đổi trong cài đặt được bắt đầu.
- **Episode 3:** Thiết kế mới được chấp nhận
  - Thuật toán nhanh hơn được sử dụng
- **Episode 4:** Các yêu cầu thay đổi
  - Yêu cầu độ chính xác cao hơn

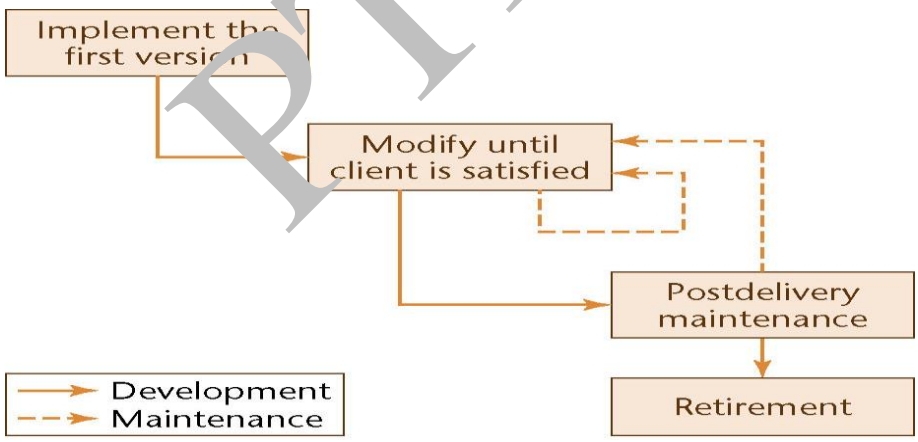
  Kết luận: Một vài năm sau, những vấn đề này lại xảy ra



Mô hình tin hóa cho bài toán Winburg Mini

3.2 MÔ HÌNH XÂY SỬA

- Không thiết kế
- Không đặc tả
  - Bảo trì là ác mộng
- Là cách dễ dàng nhất để phát triển phần mềm
- Là cách đắt nhất

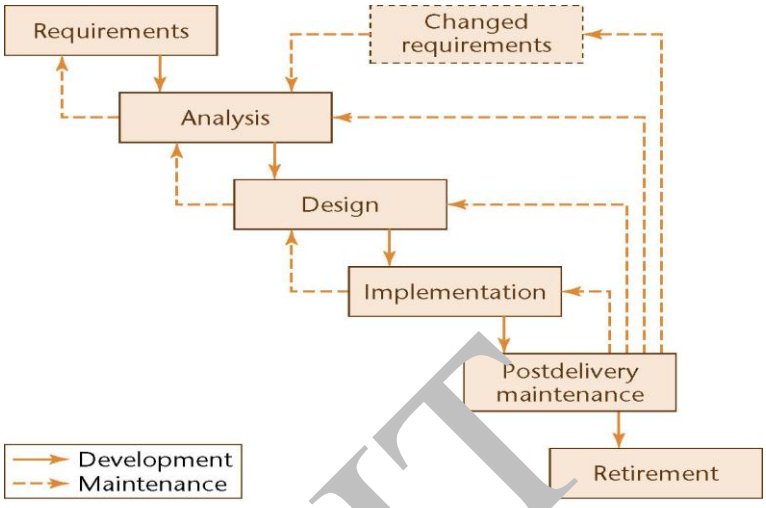


3.3 MÔ HÌNH TIẾN HÓA

- Đặc trưng bởi
  - Các vòng lặp phản hồi
  - Hướng tài liệu
  - Mô hình thác nước không biểu hiện thứ tự các sự kiện
- Thuận lợi
  - Tài liệu

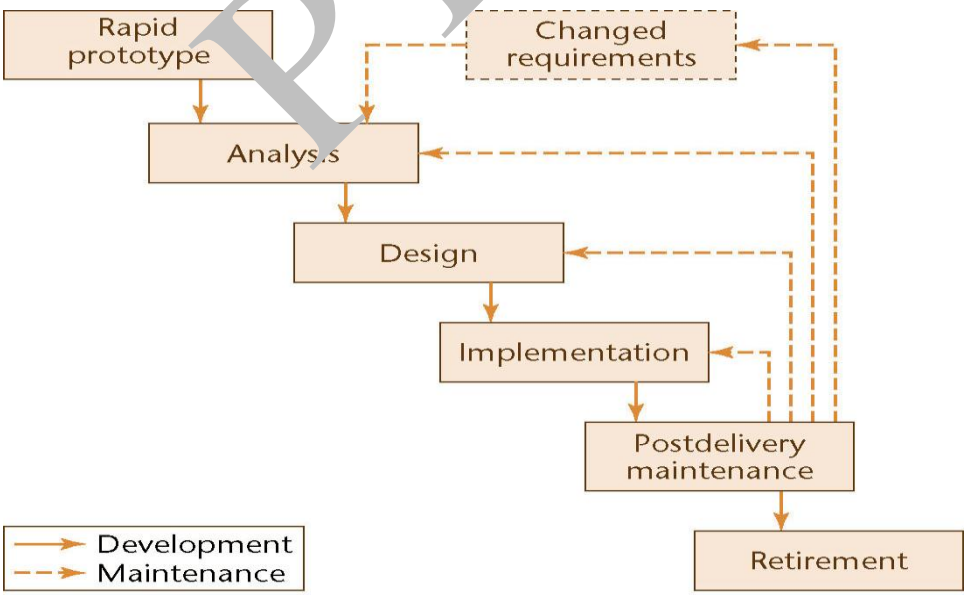
- Bảo trì dễ dàng

- Bất lợi
  - Tài liệu đặc tả
    - Joe và Jane Johnson
    - Mark Marberry

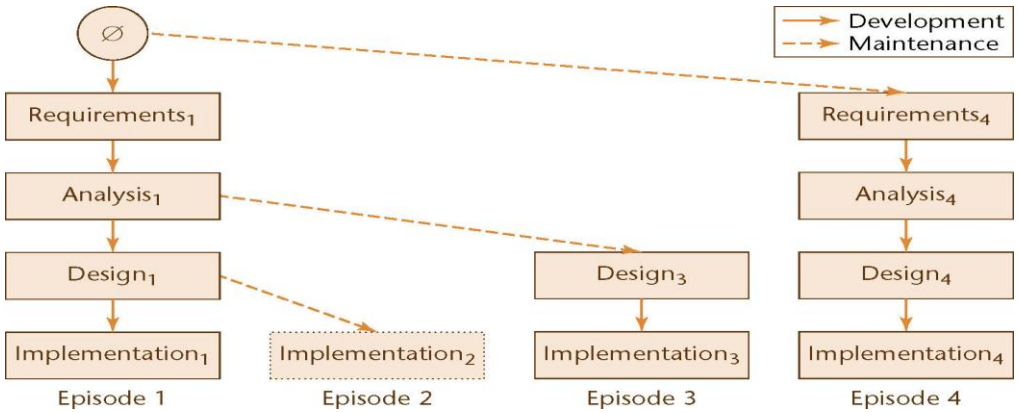


3.4 MÔ HÌNH BẢN MẪU NHANH

- Mô hình tuyến tính
- "Nhanh"



Mô hình tiến hoá



Mô hình cây tiến hóa cho bài toán Winburg Mini

- Thứ tự của các sự kiện được chỉ ra rõ ràng
- Kết thúc của mỗi Episode:
  - Có một đường cơ sở chỉ rõ tập các tài liệu phải được hoàn thiện
- Chẳng hạn:
  - Đường cơ sở ở cuối Episode 3 là:
    - Các yêu cầu$_1$, Tài liệu phân tích$_1$, Tài liệu thiết kế $_3$, cài đặt $_3$

➔ Các bài học rút ra từ bài toán Winburg Mini

- Trong thực tế, phát triển phần mềm nhiều hơn bài toán Winburg mini.
- Thay đổi luôn luôn cần thiết
  - Hệ thống phần mềm là một mô hình của thế giới thực và nó luôn luôn thay đổi
  - Chuyên gia phần mềm là con người nên có thể mắc lỗi trong quá trình phát triển phần mềm

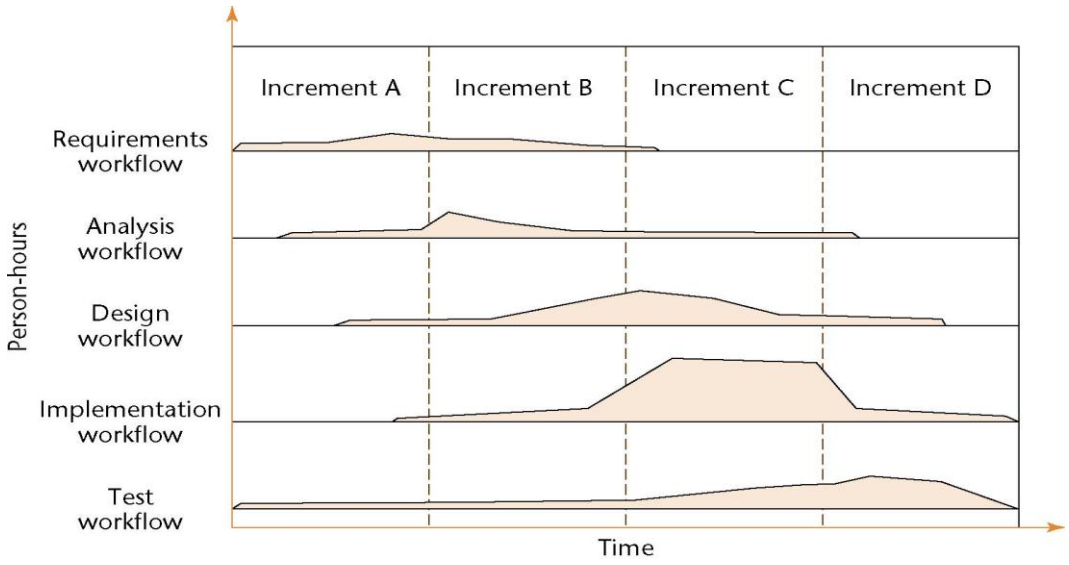
3.5. MÔ HÌNH LẶP VÀ TĂNG

- Trong thực tế, chúng ta không thể nói về "pha phân tích"
  - Mặc dù, các thao tác của pha phân tích trải rộng xuyên suốt vòng đời phần mềm.
- Tiến trình phát triển phần mềm là lặp
  - Mỗi phiên bản kế tiếp tiến gần với hệ thống đích hơn phiên bản trước đó.

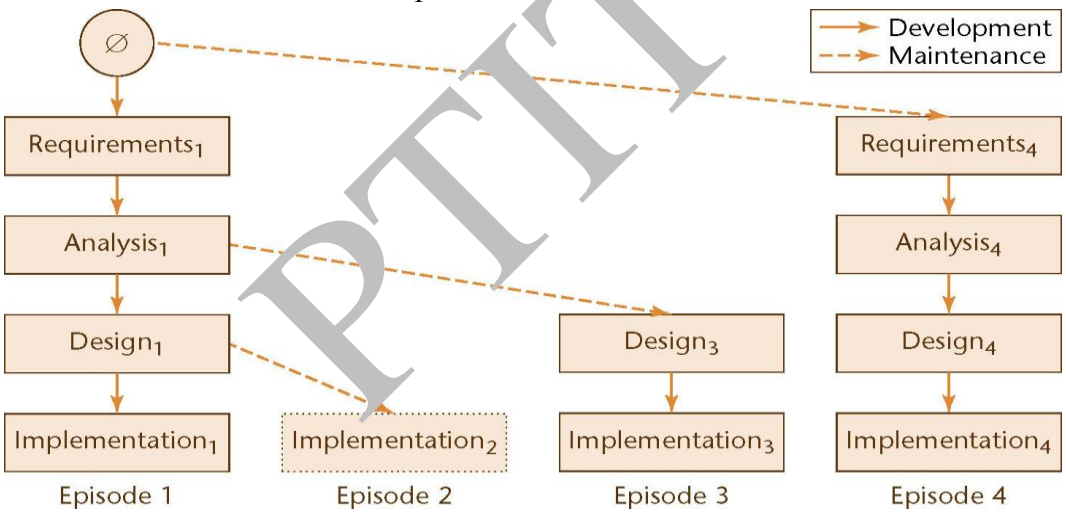
Luật Miller:

- Ở bất cứ thời điểm nào, chúng ta chỉ có thể tập trung vào khoảng 7 chunks (tương ứng 7đơn vị thông tin)
- Để xử lý lượng thông tin lớn hơn yêu cầu sử dụng bước làm mịn theo kiểu bậc thang
  - Tập trung vào các khía cạnh quan trọng nhất hiện thời
  - Các khía cạnh trì hoãn thường ít quan trọng hơn
  - Cuối cùng mọi khía cạnh đều được xử lý nhưng theo thứ tự mức độ quan trọng hiện thời

Đây là tiến trình tăng



- Lặp và tăng được sử dụng chung với các mô hình khác.
  - Không có Episode nào mà chỉ có pha "xác định yêu cầu" hoặc "pha thiết kế"
  - Có nhiều thể hiện ở mỗi pha



- Số lượng của sự gia tăng sẽ thay đổi – không phải là 4
- Và số lượng vòng lặp cũng thay đổi không phải luôn bằng ba.

Các pha cổ điển với các workflow

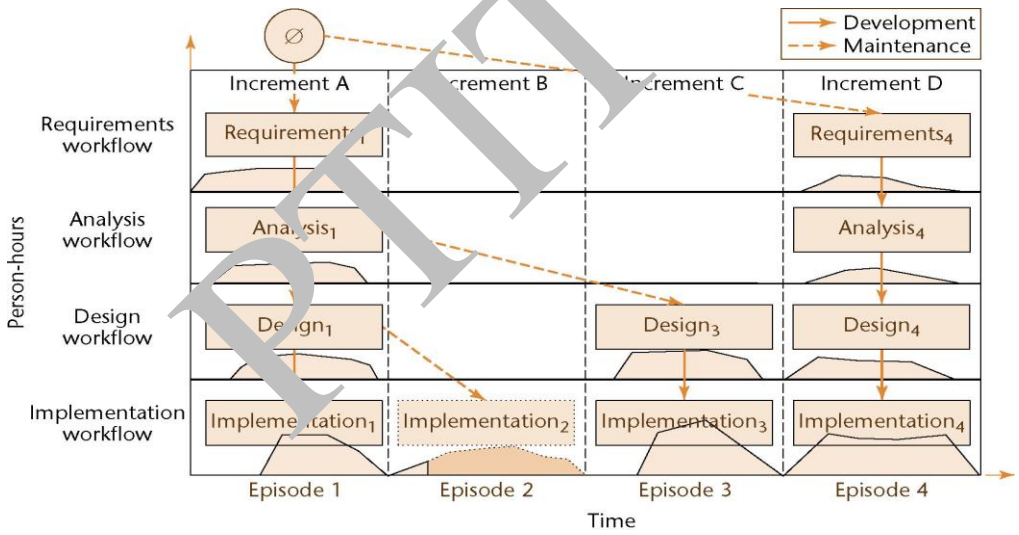
- Các pha tuần tự không có trong thế giới thực
- Mặc dù có 5 workflow (luồng công việc) chính được thực hiện ở trên toàn vòng đời phát triển phần mềm
  - Luồng công việc xác định yêu cầu - Requirements workflow
  - Luồng công việc phân tích - Analysis workflow
  - Luồng công việc thiết kế - Design workflow
  - Luồng công việc cài đặt - Implementation workflow
  - Luồng công việc kiểm thử - Test workflow

Các luồng công việc

- Cả năm luồng công việc chính được thực hiện trên toàn bộ vòng đời phần mềm
- Tuy nhiên, ở mỗi một thời điểm có một luồng công việc chiếm ưu thế.
- Chẳng hạn:
  - Ở đầu mỗi vòng đời phát triển phần mềm
    - Luồng công việc đặc tả yêu cầu chiếm ưu thế
  - Ở cuối mỗi vòng đời phát triển phần mềm
    - Luồng công việc cài đặt và kiểm thử chiếm ưu thế
- Các hoạt động lập kế hoạch và viết tài liệu được thực hiện xuyết suốt vòng đời phát triển phần mềm

Xem xét lại bài toán Winburg Mini

- Mô hình cây tiến hóa đã được thêm vào mô hình vòng đời lặp và tăng
- Luồng kiểm thử đã bị bỏ quên – mô hình cây tiến hóa giả sử rằng kiểm thử liên tục



- Đối với quá trình tăng:
  - Mỗi Episode tương ứng với một sự gia tăng
  - Không phải mỗi sự gia tăng đều bao gồm mọi luông công việc
  - Sự gia tăng B không được hoàn thiện
  - Những đường nét đứt biểu diễn sự bảo trì
    - Episodes 2, 3: Bảo trì sửa lỗi
    - Episode 4: Bảo trì hoàn thiện chức năng

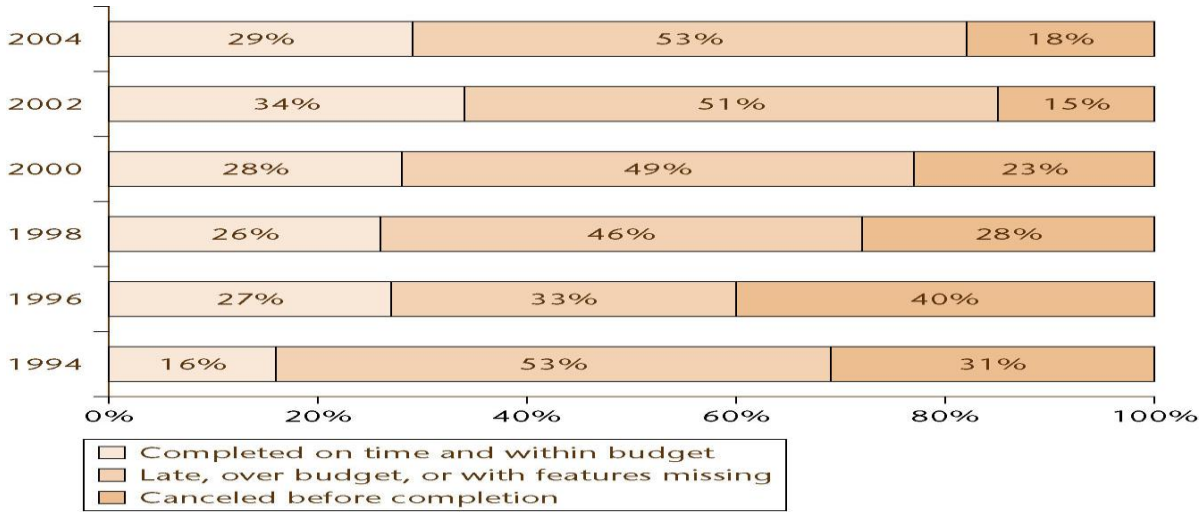
Rủi ro và những khía cạnh khác của mô hình lặp và tăng

- Chúng ta có thể xem xét toàn bộ dự án như là một tập các dự án nhỏ (quá trình tăng)
- Mỗi dự án nhỏ gồm
  - Tài liệu yêu cầu
  - Tài liệu phân tích

    - Tài liệu thiết kế
    - Tài liệu cài đặt
    - Tài liệu kiểm thử
- Tập tài liệu cuối cùng là sản phẩm phần mềm hoàn thiện
- Trong suốt dự án nhỏ:During each mini project we
    - Mở rộng các tài liệu (Sự gia tăng);
    - Kiểm tra tài liệu (luồng công việc kiểm thử); và
    - Nếu cần, thay đổi các tài liệu liên quan (vòng lặp)
- Mỗi vòng lặp có thể được xem xét như là một mô hình vòng đời thác nước nhỏ nhưng hoàn thiện
- Trong suốt mỗi vòng lặp chúng ta lựa chọn một phần của hệ thống phần mềm
- Trong mỗi phần đó cần thực hiện:
    - Pha xác định yêu cầu cổ điển
    - Pha phân tích cổ điển
    - Pha thiết kế cổ điển
    - Pha cài đặt cổ điển

Điểm mạnh của mô hình vòng đời lặp và tăng

- Có nhiều sự tối ưu cho việc kiểm tra tính đúng đắn của sản phẩm
    - Mỗi vòng lặp có tích hợp luồng công việc kiểm thử
    - Các lỗi có thể được phát hiện và sửa chữa sớm
- Sự mạnh mẽ của kiến trúc có thể được xác định sớm trong vòng đời
    - Kiến trúc – các mô đun thành phần khác nhau và cách chúng kết hợp với nhau
    - *Sự mạnh mẽ - có khả năng xử lý việc mở rộng và thay đổi mà hệ thống không bị tách thành từng mảnh*
- Chúng ta có thể giảm bớt rủi ro sớm hơn
    - Lúc nào cũng vậy rủi ro luôn liên quan tới quá trình phát triển phần mềm và bảo trì
- Chúng ta có một phiên bản hệ thống phần mềm đang làm việc
    - Khách hàng và người dùng có thể thử nghiệm với phiên bản này để xác định cái họ cần thay đổi
    - Mức độ thay đổi: các phiên bản từng phần đưa ra để làm cho sự giới thiệu sản phẩm mới tới tổ chức khách hàng dễ dàng hơn.
- Có bằng chứng thực nghiệm chứng tỏ mô hình vòng đời làm việc
- Các bản tường trình CHAOS của nhóm Standish đã chỉ ra phần trăm phần mềm thành công tăng



- Lý do của sự suy giảm của các dự án thành công năm 2004 bao gồm:
  - Nhiều dự án lớn trong năm 2004 hơn năm 2002
  - Sử dụng mô hình thác nước
  - Thiếu sự quan tâm của người dùng
  - Thiếu sự hỗ trợ từ những người điều hành cấp cao năm

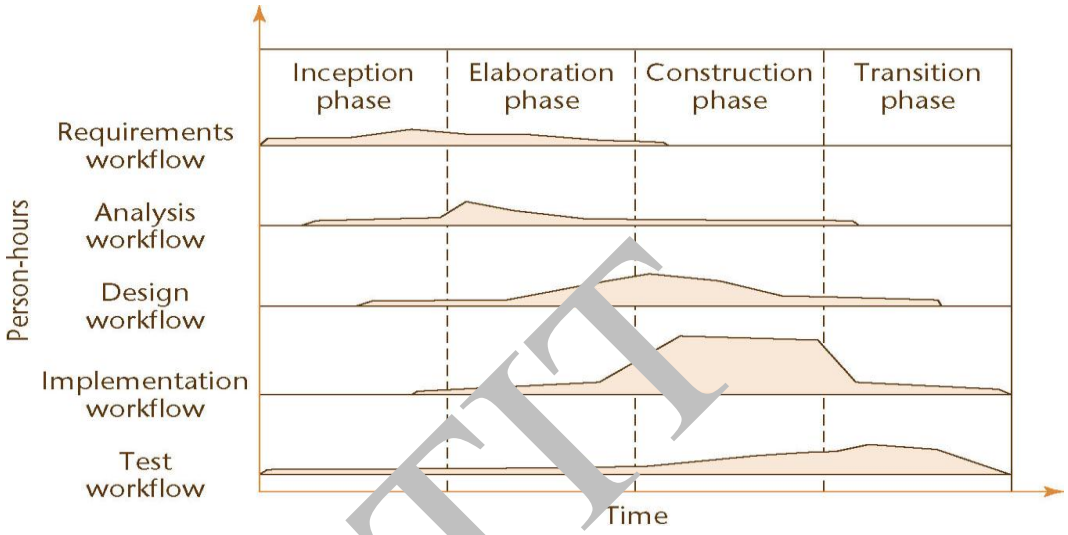
Việc quản lý lặp và tăng

- Mô hình vòng đời lặp và tăng là tập hợp các mô hình thách nước…
- … bởi vì mô hình vòng đời lặp và tăng là mô hình thác nước, được áp dụng liên tiếp
- Mỗi sự gia tăng là mộ dự án nhỏ theo mô hình vòng đời thác nước

3.6 MÔ HÌNH UP

- Cho đến gần đây, ba phương pháp luận hướng đối tượng thành công nhất:
  - Phương pháp của Booch
  - Jacobson's Objectory
  - Rumbaugh's OMT
- Năm 1999, Booch, Jacobson, và Rumbaugh đã công bố phương pháp luận phân tích và thiết kế hướng đối tượng hoàn thiện, đó là sự hợp nhất của ba phương pháp tách biệt.
  - Tên đầu tiên : *Rational Unified Process* (RUP)
  - Tiếp theo: *Unified Software Development Process* (USDP)
  - Tên được sử dụng ngày nay: *Unified Process* (for brevity)
- Quy trình hợp nhất không phải là một chuỗi các bước xây dựng hệ thống phần mềm
  - Không tồn tại phương pháp luận "một phù hợp với tất cả"
  - Có sự khác biệt lớn giữa các loại phần mềm
- Quy trình hợp nhất là một phương pháp luận có thể thích hợpThe Unified Process is an adaptable methodology
  - Nó phải được chỉnh sửa tùy theo từng phần mềm cụ thể được phát triển

- UML là đồ họa
  - Một bức tranh đáng giá ngàn từ
- Các biểu đồ UML cho phép kỹ sư phần mềm giao tiếp nhanh hơn và chính xác hơn
- Quy trình hợp nhất là một kỹ thuật mô hình hóa
  - Một mô hình là một tập các biểu đồ UML biểu diễn các khía cạnh khác nhau của sản phẩm phần mềm mà chúng ta muốn phát triển
- UML là viết tắt của ngôn ngữ mô hình hợp nhất (Unified *Modeling* Language)
  - UML là công cụ được sử dụng để mô hình hệ thống phần mềm đích.



- Bốn quá trình tăng mang tên:
  - Pha khởi đầu - Inception Phase
  - Pha khảo sát thiết kế - Elaboration phase
  - Pha xây dựng - Construction phase
  - Pha chuyển giao - Transition phase
- Các pha của tiến trình hợp nhất là tăng
- Theo lý thuyết, số lượng quá trình tăng là bất kỳ
  - Trong thực tế, quá trình phát triển thường gồm 4 quá trình tăng
- Mỗi bước thực hiện trong tiến trình hợp nhất được chia thành
  - Một trong 5 luồng công việc chính và cũng có thể
  - Một trong bốn pha
- Tại sao mỗi bước phải được xem xét hai lần?
- Luồng công việc
  - Là ngữ cảnh kỹ thuật của một bước
- Pha
  - Là ngữ cảnh nghiệp vụ của mỗi bước

Pha khởi đầu

Mục đích của pha này là xác định liệu sản phẩn phần mềm đã đề xuất có thể làm được về mặt tài chính

1. Hiểu được lĩnh vực xây dựng phần mềm
2. Xây dựng mô hình nghiệp vụ
3. Phân định phạm vi của dự án đã đề xuất
    - Tập trung vào tập con của mô hình nghiệp vụ mà đã được bao phủ bởi sản phẩm phần mềm đề xuất
4. Bắt đầu thực hiện những trường hợp nghiệp vụ ban đầu

- Các câu hỏi cần được trả lời bao gồm:
    - Có phải sản phẩm phần mềm đề xuất ước tính chi phí hiệu quả?
    - Bao lâu sẽ thu được vốn đầu tư??
    - Về mặt giải pháp, chi phí sẽ lấy từ đâu nếu công ty quyết định không phát triển sản phẩm phần mềm đã đề xuất?
    - Nếu sản phẩm phần mềm không được bán trên thị trường thì có phải việc nghiên cứu thị trường cần thiết được thực hiện không?
    - Sản phẩm phần mềm được đề xuất có thể được chuyển giao đúng thời gian không?
    - Nếu sản phẩm phần mềm được phát triển để hỗ trợ các hoạt động của tổ chức khách hàng, cái gì sẽ bị ảnh hưởng nếu sản phẩm phần mềm được chuyển giao muộn?
    - Những rủi ro nào liên quan đến việc phát triển phần mềm?
    - Những rủi ro này có thể giảm nhẹ như thế nào?
        - Có phải đội phát triển sản phẩm phần mềm đã đề xuất là những người có kinh nghiệm?
        - Có phải sản phẩm phần mềm này cần phần cứng mới?
        - Nếu như vậy, thì có phải có một rủi ro thì sản phẩm phần mềm đề xuất sẽ không được chuyển giao đúng thời gian?
        - Có phải có một cách để giảm nhẹ rủi ro, đó là bằng cách đưa ra phần cứng sao chép dự phòng từ nhà cung cấp khác??
        - Các công cụ phần mềm cần được yêu cầu (chương 5)?Are software tools (Chapter 5) needed?
        - Có phải chúng luôn sẵn có?
        - Có phải chúng có tất cả những chức năng cần thiết?

Tất cả các câu trả lời đều được yêu cầu ở cuối pha khởi đầu để trường hợp nghiệp vụ khởi đầu có thể được tạo ra

- Nắm được phiên bản đầu tiên của trường hợp nghiệp vụ là mục đích nói chung của pha khởi đầu
- Các phiên bản đầu tiên kết hợp chặt chẽ
    - Bản miêu tả phạm vi của sản phẩm phần mềm
    - Chi tiết về mặt tài chính
    - Nếu sản phẩm phần mềm được đề xuất có thể được kinh doanh thì trường hợp nghiệp vụ sẽ bao gồm:

    - Đưa ra kết hoạch thu nhập, ước lượng thị trường, ước lượng chi phí ban đầu
  - Nếu sản phẩm phần mềm được sử dụng nội bộ thì trường hợp nghiệp vụ bao gồm
    - Phân tích lợi nhuận và chi phí ban đầu

Các rủi ro:

- Có ba loại rủi ro chính:
  - Rui ro kỹ thuật
    - Xem ở slide trước đó
  - Rủi ro do xác định yêu cầu không đúng
    - Được giảm nhẹ đi bằng việc thực hiện luồng công việc đặc tả yêu cầu một cách chính xác
  - Rủi ro do thực hiện kiến trúc không đúng
    - Kiến trúc không thể đủ mạnh mẽ
- Làm giảm nhẹ ba loại rủi ro
  - Các rủi ro cần được xếp hạng để những rủi ro quan trọng được làm giảm nhẹ trước
- Công việc này kết thúc các bước của pha khởi đầu đối với luồng công việc xác định yêu cầu

Luồng công việc phân tích, thiết kế

- Một phần nhỏ luồng công việc phân tích có thể được thực hiện trong suốt pha khởi đầu
  - Thông tin cần thiết cho thiết kế kiến trúc cần được trích rút.
- Theo đó, một phần nhỏ luồng công việc của thiết kế cũng được thực hiện

Luồng công việc cài đặt

- Cài đặt nói chung dược thực hiện trong suốt pha khởi đầu
- Tuy nhiên, đôi khi proof-of- concept-prototype được xây dựng để kiểm thử tính khả thi của việc xây dựng mỗi phần của sản phẩm phần mềm

Luồng công việc kiểm thử

- Luồng công việc kiểm thử gần như bắt đầu ở giai đoạn đầu của pha khởi đầu
  - Mục đích để đảm bảo rằng các yêu cầu được xác định một cách chính xác

Lập kế hoạch ở pha khởi đầu

- Bắt đầu pha khởi đầu không có đủ thông tin để lập kế hoạch cho toàn bộ quá trình phát triển phần mềm
  - Việc lập kế hoạch duy nhất được thực hiện ở đầu dự án là việc lập kế hoạch cho chính pha khởi đầu
- Cùng với lý do đó, việc lập kế hoạch duy nhất được thực hiện ở cuối pha khởi đầu là bản kế hoạch cho pha tiếp theo (pha khảo sát tỷ mỉ)

Tài liệu của pha khởi đầu

- Những sản phẩm chuyển giao ở pha khởi đầu bao gồm::

- Phiên bản đầu tiên của mô hình lĩnh vực hoạt động
- Phiên bản đầu tiên của mô hình nghiệp vụ
- Phiên bản đầu tiên của tài liệu xác định yêu cầu
- Phiên bản sơ bộ của tài liệu phân tích
- Phiên bản sơ bộ của kiến trúc
- Danh sách ban đầu về các rủi ro
- Đưa ra các use case
- Lập kế hoạch cho pha khảo sát tỉ mỉ
- Phiên bản đầu tiên của các trường hợp nghiệp vụ

Pha khảo sát tỉ mỉ

- Mục đích của pha khảo sát tỉ mỉ là làm mịn những yêu cầu ban đầu
  - Làm mịn kiến trúc
  - Giám sát rủi ro và làm mịn độ ưu tiên của chúng
  - Làm mịn trường hợp nghiệp vụ
  - Đưa ra kế hoạch quản lý dự án
- Các hoạt động chính của pha khảo sát tỉ mỉ là làm mịn hoặc khảo sát tỉ mỉ các pha trước đó
- Những công việc của pha khảo sát tỉ mỉ bao gồm:
  - Tất cả nhưng hoàn thành luồng công việc xác định yêu cầu
  - Thực hiện một cách thực dụng luồng công việc phân tích toàn thể
  - Bắt đầu thiết kế kiến trúc
- Sản phẩm chuyển giao của pha khảo sát tỉ mỉ bao gồm:
  - Mô hình lĩnh vực hoàn thiện
  - Mô hình nghiệp vụ hoàn thiện
  - Các tài liệu xác định yêu cầu hoàn thiện
  - Các tài liệu phân tích hoàn thiện
  - Phiên bản cập nhật kiến trúc
  - Danh sách cập nhật các rủi ro
  - Kế hoạch quản lý dự dán (cho những phần còn lại của dự án)
  - Trường hợp nghiệp vụ hoàn thiện

Pha xây dựng

- Mục đích của pha này là đưa ra phiên bản chất lượng – sẵn sàng hoạt động đầu tiên của sản phẩm phần mềm
  - Đôi khi gọi đây là sự phát hành phiên bản beta
- Pha này tập trung vào những công việc
  - Cài đặt và
  - Kiểm thử
    - Kiểm thử đơn vị của các mô đun

  - Kiểm thử tích hợp của các hệ thống con
  - Kiểm thử sản phẩm của toàn hệ thống
- Sản phẩm chuyển giao của pha xây dựng bao gồm:
  - Sổ tay người dùng ban đầu và các sổ tay khác
  - Tất cả các tài liệu được tạo ra(các phiên bản phát hành beta)
  - Kiến trúc hoàn thiện
  - Danh sách rủi ro đã được cập nhật
  - Kế hoạch quản lý dự án (Cho những phần còn lại của dự án)
  - Nếu cần thiết cập nhật trường hợp nghiệp vụ

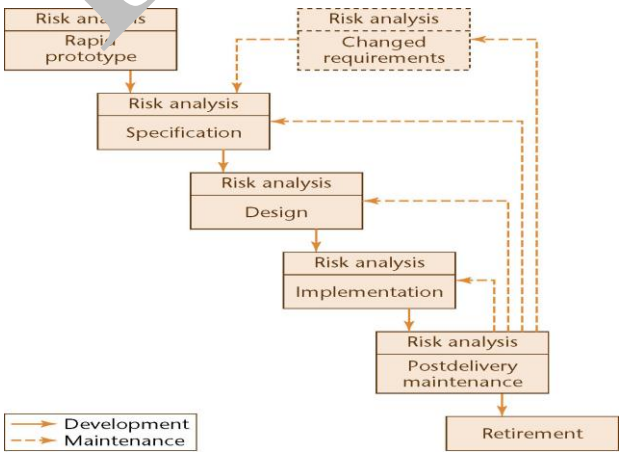
Pha chuyển tiếp

- Mục đích của pha chuyển tiếp là đảm bảo yêu cầu của khách hàng được đáp ứng
  - Lỗi trong sản phẩm phần mềm được sửa
  - Tất cả các sổ tay được hoàn thiện
  - Cố gắng tìm ra những rủi ro mà trước đó chưa được nhận dạng
- Pha này hướng tới những phản hồi từ phía mà phát hành beta được cài đặt
- Sản phẩm chuyển giao của pha chuyển tiếp bao gồm:
  - Tất cả các tài liệu được tạo ra (các phiên bản cuối cùng)
  - Các sổ tay hoàn thiện

3.7 MÔ HÌNH XOẮN ỐC

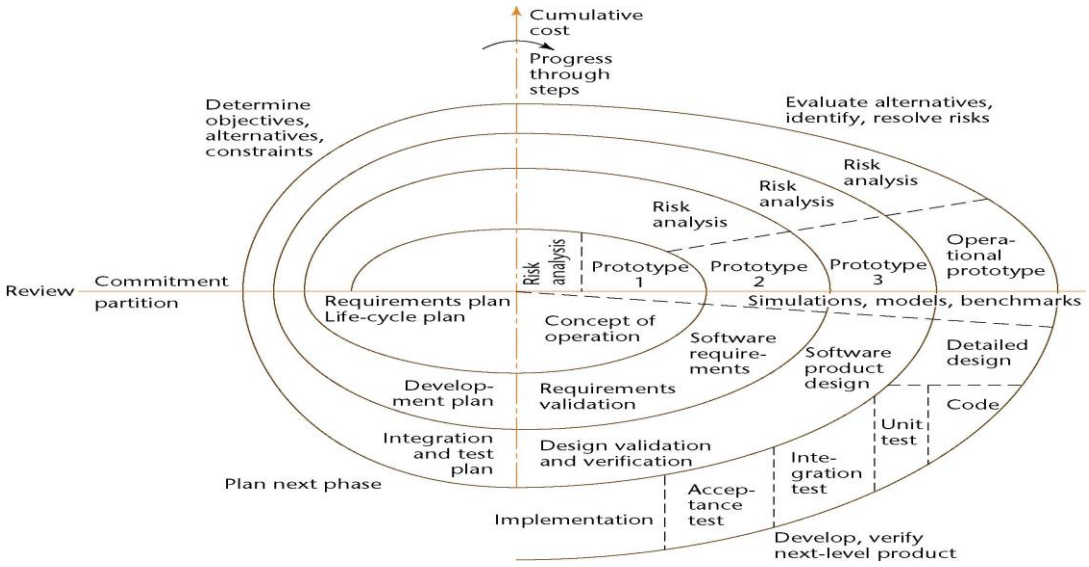
Là mô hình bản mẫu nhanh kết hợp với phân tích rủi ro đi kèm ở mỗi pha

Đặc điểm chính của mô hình xoắn ốc: Nếu các rủi ro không được làm giảm bớt, thì dự án bị kết thúc ngay lập tức.



Mô hình xoắn ốc đầy đủ:

- Ở trước mỗi pha là các giải pháp và phân tích rủi ro
- Theo sau mỗi pha là: Đánh giá và lập kế hoạch cho pha tiếp theo
- Chiều bán kính: Chi phí tích lũy cho đến hiện tại
- Chiều góc: sự đi lên theo vòng ốc



Điểm mạnh của mô hình xoắn ốc:

- Dễ dàng thay đổi mức độ kiểm thử
- Không phân biệt giữa phát triển và bảo trì

Điểm yếu của mô hình xoắn ốc:

- Chỉ phù hợp với những phần mềm lớn
- Chỉ phù hợp với những phần mềm nội bộ

3.8 MÔ HÌNH MÃ NGUỒN MỞ

- Hai pha không hình thức
- Đầu tiên, một các nhân xây dựng một phiên bản đầu tiên
  - Đưa phiên bản này lên Internet (ví dụ: SourceForge.net)
- Sau đó, nếu có sự quan tâm tới dự án này
  - Phiên bản đầu tiên sẽ được tải về một cách rộng rãi
  - Người dùng trở thành người đồng phát triển
  - Sản phẩm được mở rộng
- Điểm chính: Các cá nhân nhìn chung làm việc tình nguyện đối với dự án mã nguồn mở trong thời gian rảnh rỗi của họ
- Các hoạt động trong pha phi hình thức thứ hai
  - Báo cáo và sửa lỗi
    - Bảo trì sửa lỗi
  - Thêm các chức năng mới
    - Bảo trì hoàn thiện chức năng
    - Điều chỉnh phần mềm ở môi trường mới
    - Bảo trì thích hợp
    - Pha phi hình thức thứ hai chỉ bao gồm bảo trì sau khi đã chuyển giao sản phẩm

- Từ "người đồng phát triển" trong slide trước có thể thay thế bằng từ "đồng bảo trì"

- Mô hình vòng đời bảo trì sau khi chuyển giao sản phẩm

- Sản phẩm mã nguồn đóng được bảo trì và kiểm thử bởi nhân viên
  - Người dùng có thể đưa ra bản tường trình sự thất bại của phần mềm (failure) nhưng không bao giờ đưa ra được lỗi mã nguồn (fault) (vì mã nguồn không sẵn có)
- Phần mềm mã nguồn mở nhìn chung được bảo trì bởi người tình nguyện viên không lương
  - Người dùng được khuyến khích đưa ra bản tường trình về sự thất bại của phần mềm (failure) cũng như lỗi mã nguồn (fault)
- Nhóm chính
  - Số lượng nhỏ những người bảo trì tận tụy với sở thích, thời gian và những kỹ năng cần thiết để đưa ra những báo cáo lỗi mã nguồn đã được cố định("fixes")
  - Họ chịu trách nhiệm quản lý dự án
  - Họ có quyền cài đặt lại những lỗi đã cố định
- Nhóm ngoại vi
  - Users who choose to submit defect reports from time to time
- Các phiên bản mới của phần mềm mã nguồn đóng được phát hành thường xuyên một năm một lần
  - Sau khi đã được kiểm thử cẩn thận bởi nhóm quản lý chất lượng phần mềm
- Các phát hành của nhóm chính của một phiên bản mới của sản phẩm mã nguồn mở được đưa ra ngay khi sẵn sàng
  - Có thể là một tháng hoặc thậm chí là một ngày so với với phiên bản trước đó
  - Nhóm chính thực hiện kiểm thử tối thiểu
  - Kiểm thử mở rộng được thực hiện bởi các thành viên của nhóm ngoại vi trong suốt thời gian sử dụng phần mềm
  - "Phát hành sớm và thường xuyên"
- Phiên bản làm việc đầu tiên được đưa ra khi sử dụng:
  - Mô hình bản mẫu nhanh;
  - Mô hình xây và sửa; và

    - Mô hình mã nguồn mở
- Sau đó:
    - Mô hình bản mẫu nhanh
        - Phiên bản đầu tiên bị bỏ qua
    - Mô hình xây và sửa và mô hình vòng đời mã nguồn mở
        - Phiên bản ban đầu trở thành phiên bản đích
- Do đó, trong dự án mã nguồn mở, nhình chung không có đặc tả và không có thiết kế
- Một số dự án mã nguồn mở đã thành công như thế nào khi không có đặc tả hoặc thiết kế?
- Sản phẩm mã nguồn mở đã thu hút được một số chuyên gia phần mềm tốt nhất trên thế giới
    - Họ có thể xây dựng các chức năng hiệu quả mà không cần đặc tả hoặc thiết kế
- Tuy nhiên, cuối cùng thì sự phát triển sản phẩm mã nguồn mở cũng dừng lại khi mà nó không có sự bảo trì nữa
- Mô hình vòng đời mã nguồn mở bị hạn chế trong khả năng ứng dụng của nó
- Nó có thể thành công cực độ đối với những dự án cơ sở hạ tầng như
    - Hệ điều hành (Linux, OpenBSD, Mach, Darwin)
    - Trình duyệt Web (Firefox, Netscape)
    - Trình biên dịch (gcc)
    - Máy chủ Web (Apache)
    - Hệ thống quản lý dữ liệu (MySQL)
- Không thể phát triển mã nguồn mở một sản phẩm phần mềm mà chỉ được sử dụng trong tổ chức thương mại
    - Các thành viên của cả nhóm chính và nhóm ngoại vi đều luôn luôn là những người dùng của phần mềm đang được phát triển
- Mô hình vòng đời mã nguồn mở không không thể áp dụng được trừ khi sản phẩm đích được xem xét bởi một số lượng lớn người dùng vì sản phẩm đó có ích cho họ.
- Khoảng một nửa dự án mã nguồn mở trên Web không thu hút được các nhóm phát triển vào làm việc cho dự án
- Even where work has started, the overwhelming preponderance will never be completed
- Nhưng khi mô hình mã nguồn mở đã làm việc thì đối khi nó cũng đem lại thành công đáng kinh ngạc
    - Những sản phẩm mã nguồn mở được kể ở slide trước đã đước sử dụng bởi hàng triệu người dùng

CHƯƠNG 4: KIỂM THỬ

4.1 VẤN ĐỀ CHẤT LƯỢNG PHẦN MỀM

- Phần mềm thỏa mãn ở một mức độ nào đó các đặc tả của nó
- Mỗi chuyên gia phần mềm có trách nhiệm đảm bảo công việc của họ là chính xác
    - Chất lượng phải được xây dựng từ ban đầu

4.1.1 Đảm bảo chất lượng phần mềm (SQA)

- Các thành viên của nhóm SQA phải đảm bảo những người phát triển thực hiện công việc đạt chất lượng cao
    - Ở cuối mỗi luồng công việc
    - Khi sản phẩm được hoàn thiện
- Thêm vào đó, đảm bảo chất lượng phải được áp dụng
    - Trong mỗi tiến trình
        - **Ví dụ: các chuẩn**

4.1.2. Độc lập trong quản lý

- Phải độc lập về quản lý giữa
    - Nhóm phát triển
    - Nhóm SQA
- Mỗi nhóm không được phép vượt quá quyền hạn sang các nhóm khác
- Quản lý lâu năm hơn phải quyết định liệu
    - Chuyển giao sản phẩm đúng thời hẹn nhưng có lỗi xảy ra
    - Hoặc kiểm thử hơn nữa và chuyển giao sản phẩm muộn hơn
- Những quyết định phải xem xét đến sự quan tâm của khách hàng và tổ chức phát triển

4.2 KIỂM CHỨNG PHẦN MỀM

- Có hai kiểu : Kiểm thử không có sự thực thi và kiểm thử có dựa trên sự thực thi
- "V & V"
    - *Xác minh (Verification)*
        - Xác định nếu luồng công việc được hoàn thiện chính xác Determine if the workflow was completed correctly
    - *Thẩm định (Validation)*
        - Xác định nếu toàn bộ sản phẩm đáp ứng các yêu cầu đưa ra (Determine if the product as a whole satisfies its requirements)
    - Thuật ngữ "xác minh" cũng được sử dụng cho kiểm thử không có sự thực thi.

4.3 CÁC PHƯƠNG PHÁP KIỂM CHỨNG

4.3.1. Kiểm thử không có sự thực thi

- Nguyên lý cơ bản
    - Chúng ta không nên tự mình rà soát công việc của mình
    - Có sự hiệp lực trong nhóm

4.3.1.1 Rà soát lướt qua (Walkthrough)

- Đội rà soát lướt qua bao gồm từ 4 đến 6 thành viên
- Nó bao gồm đại diện của
    - Đội chịu trách nhiệm cho luồng công việc hiện thời
    - Đội chịu trách nhiệm cho luồng công việc tiếp thep
    - Nhóm SQA
- Rà soát lướt qua được nói trước bởi sự chuẩn bị
    - Danh sách các mục
        - Các mục không hiểu
        - Các mục mà xuất hiện ở dạng lỗi

Quản lý rà soát lướt qua:

- *Đội* rà soát lướt qua được đại diện SQA làm chủ trì

- Trong rà soát lướt qua chỉ phát hiện lỗi, không sửa
    - Việc sửa lỗi được thực hiện bởi một ủy ban và có khả năng dẫn đến chất lượng thấp
    - Chi phí sửa lỗi của ủy ban là quá cao
    - Không phải tất cả các hạng mục được đánh dấu thực tế đều sai
    - Rà soát lướt quat không nên kéo dài quá 2 tiếng
    - Cũng như không có thời gian sửa lỗi

4.3.1 .2 Kiểm tra kỹ lưỡng (Inspection)

- Quá trình kiểm tra kỹ lưỡng gồm 5 bước hình thức
    - Tổng quan
    - Chuẩn bị, với mục đích thống kê các loại lỗi
    - Kiểm tra kỹ lưỡng
    - Sửa lỗi
    - Theo dõi
- Đội kiểm tra gồm 4 thành viên
    - Người điều tiết (Moderator)
    - Một thành viên trong đội thực đang thi luồng công việc hiện hành
    - Một thành viên của đội sẽ thực hiện luồng công việc tiếp theo
    - Một thành viên của nhóm SQA
- Những vai trò đặc biệt được đảm nhiệm bởi
    - Người điều tiết (Moderator)
    - Người đọc (Reader)
    - Người ghi chép (Recorder)

Thống kê lỗi:

- Các lỗi được ghi chép lại cẩn thận
    - Ví dụ:

- Chính hoặc phụ

- Các lỗi được ghi chép theo các kiểu lỗi
  - Ví dụ các lỗi thiết kế:
    - Không phải tất cả các mục đã được bàn bạc
    - Các lý lẽ hình thức và thực tế không tương ứng với nhau
- Đối với mỗi luồng công việc xác định, chúng ta so sánh tỷ lệ lỗi hiện thời với tỷ lệ lỗi của những sản phẩm trước
- Nếu thấy số lượng lỗi không cân xứng với tài liệu
  - Thiết kế lại từ ban đầu là cách tốt nhất
- Thực hiện chuyển những thống kê lỗi tới luồng công việc tiếp theo
  - Chúng ta có thể không phát hiện được tất cả các lỗi của mỗi kiểu cụ thể trong quá trình kiểm tra kỹ lưỡng hiện tại

Thống kê quá trình kiểm tra kỹ lưỡng

- Quá trình kiểm tra kỹ lưỡng IBM đã chỉ ra:
  - 82% lỗi được phát hiện (1978)
  - 70% lỗi được phát hiện (1978)
  - 93% lỗi được phát hiện (1986)
- Chuyển hệ thống (Switching system)
  - Giảm 90% chi phí trong việc phát hiện lỗi (1986)
- JPL
  - 4 lỗi chính, 14 lỗi phụ trong 2 giờ (1990)
  - Tiết kiệm $25,000 trong mỗi quá trình kiểm tra kỹ lưỡng
  - Số lượng lỗi đã giảm theo hàm số mũ trong mỗi pha (1992)
- Cảnh báo
- Thống kê lỗi chưa bao giờ được sử dụng để đánh giá hiệu năng
  - "Killing the goose that lays the golden eggs"

Công cụ cho kiểm tra kỹ lưỡng :

- Tỷ lệ kiểm tra kỹ lưỡng (ví dụ: số trang thiết kế được kiểm tra trên 1giờ)
- Mật độ lỗi (ví dụ: số lỗi trên nghìn dòng lệnh (KLOC) được kiểm tra)
- Tỷ lệ phát hiện lỗi (ví dụ: số lỗi được phát hiện trên 1 giờ)
- Hiệu năng phát hiện lỗi (ví dụ: số lượng những lỗi chính, lỗi phụ được phát hiện trong 1 giờ)
- Có phải tăng 50% tỷ lệ phát hiện lỗi thì đồng nghĩa với
    - Chất lượng giảm? Hoặc
    - Quá trình kiểm tra kỹ lưỡng hiệu quả hơn?

4.3.1.3 So sánh giữa rà soát lướt qua và kiểm tra kỹ lưỡng

- Lướt qua
    - Hai bước, tiến trình không chính thức
        - Chuẩn bị
        - Phân tích
- Kiểm tra kỹ lưỡng
    - 5 bước, tiến trình hình thức
        - Tổng quan
        - Chuẩn bị
        - Kiểm tra kỹ lưỡng
        - Sửa lỗi
        - Theo dõi

4.3.1.4 Những điểm mạnh và điểm yếu của các loại rà soát

- Rà soát có thể rất hiệu
    - Các lỗi được phát hiện sớm ở trong quá trình phát triển phần mềm
- Rà soát ít hiệu quả nếu quá trình không đầy đủ
    - Những phần mềm phạm vi lớn bao gồm những phần nhỏ và độc lập với nhau

- Tài liệu của những luồng công việc trước phải được hoàn thiện và luôn có sẵn

4.3.2 Kiểm thử có dựa trên sự thực thi

- Các tổ chức sử dụng tới 50% ngân sách dự án để kiểm thử
  - Nhưng phần mềm được chuyển giao thường xuyên xảy ra lỗi
- Dijkstra (1972)
  - "Kiểm chương trình có thể là cách rất hiệu quả để chỉ ra các lỗi nhưng không có khả năng để chỉ ra sự vắng mặt của chúng".("Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence")

4.4 NHỮNG VẤN ĐỀ TRONG KIỂM THỬ

4.4.1 Cái gì nên được kiểm thử?

Chúng ta cần kiểm thử tính chính xác, tiện ích, tính đáng tin, tính mạnh mẽ và hiệu năng

4.4.1.1 Tính chính xác

- Một phần mềm chạy chính xác nếu nó thỏa mãn đặc tả của nó
- Tính chính xác của các đặc tả
  - Đặc tả không chính xác đối với một yêu cầu sắp xếp:

Input specification:	p : array of n integers, n > 0.
Output specification:	q : array of n integers such that $q[0] \leq q[1] \leq \cdots \leq q[n-1]$

  - Hàm trickSort đã thỏa mãn đặc tả này:

```
void trickSort (int p[ ], int q[ ])
{
    int i;
    for (i = 0; i < n; i++)
        q[i] = 0;
}
```

  - Đặc tả chính xác:

Input specification:	p : array of n integers, $n > 0$.
Output specification:	q : array of n integers such that $q[0] \leq q[1] \leq \cdots \leq q[n-1]$ The elements of array q are a permutation of the elements of array p, which are unchanged.

- Về mặt kỹ thuật, tính chính xác là không cần thiết (chẳng hạn như trình biên dịch C++) và không đầy đủ (ví dụ như : *trickSort*)

4.4.1.2 Tiện ích

- Sản phẩm phần mềm đáp ứng ở một mức nào đó yêu cầu của người dùng
  - Ví dụ:
    - Tính dễ sử dụng
    - Các chức năng hữu ích
    - Có hiệu quả trong chi phí

4.4.1.3 Tính đáng tin

- Đo tần suất và tính quan trọng của các thất bại (failure)
  - Thời gian trung bình giữa những lần chức năng không thực hiện
  - Thời gian trung bình để sửa
  - Thời gian và chi phí để sửa chữa kết quả của một thất bại trong hệ thống phần mềm

4.4.1.4 Tính mạnh mẽ

- Một chức năng của
  - Một dãy các điều kiện hoạt
  - Khả năng các kết quả không được chấp nhận với đầu vào hợp lệ
  - Ảnh hưởng của đầu vào không hợp lệ

4.4.1.5 Hiệu năng

- Những ràng buộc về thời gian và không gian được đáp ứng ở một mức độ nào đó
- Phần mềm thời gian thực được đặc trưng bởi những ràng buộc thời gian thực
- Nếu dữ liệu bị mất bởi vì hệ thống quá chậm

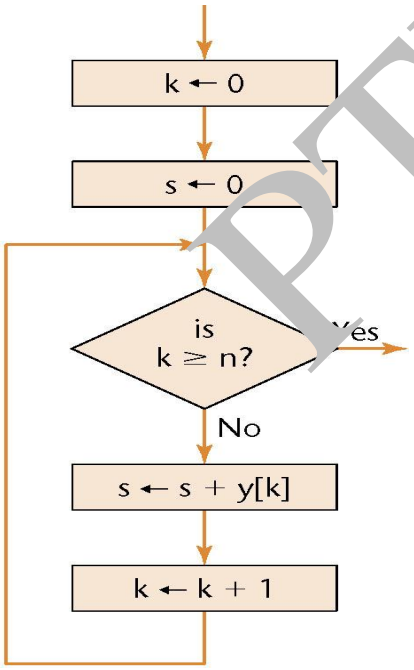
- Không có cách để phục hồi dữ liệu đó

4.4.2 Kiểm thử và sự kiểm chứng tính chính xác

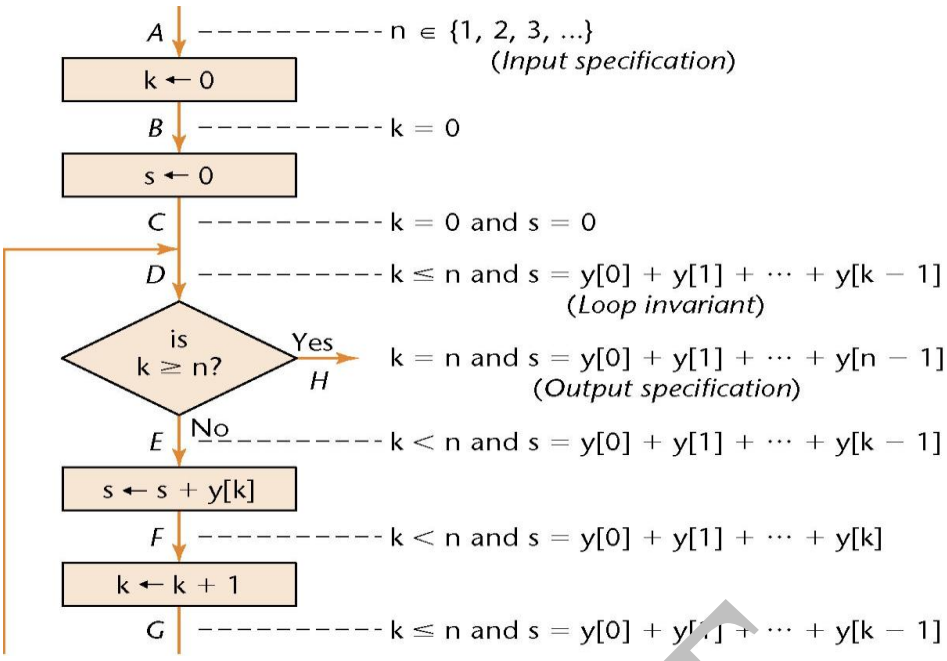
- Sự kiểm chứng tính chính xác là một phương pháp khác đối với kiểm thử có dựa trên sự thực thi
- Ví dụ về sự kiểm chứng tính chính xác
  - Đoạn code đã chứng minh sự chính xác

```
int k, s;
int y[n];
k = 0;
s = 0;
while (k < n)
{
    s = s + y[k];
    k = k + 1;
}
```

  - Biểu đồ luồng tương ứng với đoạn code trên



  - Thêm vào : đặc tả đầu vào, đặc tả đầu ra, số lượng vòng lặp và sự xác nhận



4.4.3 Sự kiểm chứng tính chính xác và kỹ nghệ phần mềm

- Ba chuyện tưởng tượng trong chứng minh tính chính xác
  - Kỹ sư phần mềm không đủ kiến thức toán học để kiểm chứng
    - Hầu hết các ngành khoa học máy tính hoặc biết hoặc có thể học toán học đã yêu cầu cho kiểm chứng
  - Việc chứng minh yêu cầu chi phí rất cao trong thực tế
    - Khả năng kinh tế được xác định từ phân tích lợi nhuận và chi phí
  - Việc chứng minh quá khó khăn
    - Nhiều sản phẩm phần mềm không bình thường đã được chứng minh thành công
    - Các công cụ giống như chứng minh định lý có thể sẽ hữu ích
- Những khó khăn khi chứng minh tính chính xác
  - Chúng ta có thể tin vào cách chứng minh định lý không?

```
void theoremProver ( )
{
    print "This product is correct";
}
```

  - Chúng ta tìm các đặc tả đầu vào, đầu ra, số lượng vòng lặp như thế nào?

- Chuyện gì sẽ xảy ra nếu các đặc tả là sai?
- Chúng ta có thể chưa bao giờ chắc chắn rằng các đặc tả hoặc hệ thống xác minh là chính xác

- Sự kiểm chứng tính chính xác là một công cụ kỹ nghệ phần mềm quan trọng, thích hợp với:
  - Khi cuộc sống con người đang lâm nguy (When human lives are at stake)
  - Khi đã được chỉ ra bởi phân tích lợi nhận – chi phí (When indicated by cost–benefit analysis)
  - Khi rủi ro của việc không phân tích quá lớn (When the risk of not proving is too great)
- Sự kiểm chứng không hình thức có thể cải thiện chất lượng của phần mềm
  - Sử dụng những câu lệnh assert

4.4.4 Ai thực hiện kiểm thử phần mềm

- Lập trình là xây dựng
- Kiểm thử là phá huỷ
  - Một kiểm thử thành công tìm ra một lỗi
- Vì thế, những người lập trình không nên kiểm thử chính tài liệu mã của họ viết
- Giải pháp:
  - Người lập trình thực hiện kiểm thử không hình thức
  - Sau đó nhóm SQA thực hiện kiểm thử một cách hệ thống
  - Người lập trình gỡ lỗi mô đun đó
- Tất cả các trường hợp kiểm thử phải
  - Được lập kế hoạch bằng tay trước đó, bao gồm đầu ra mong muốn và
  - Được giữ về sau

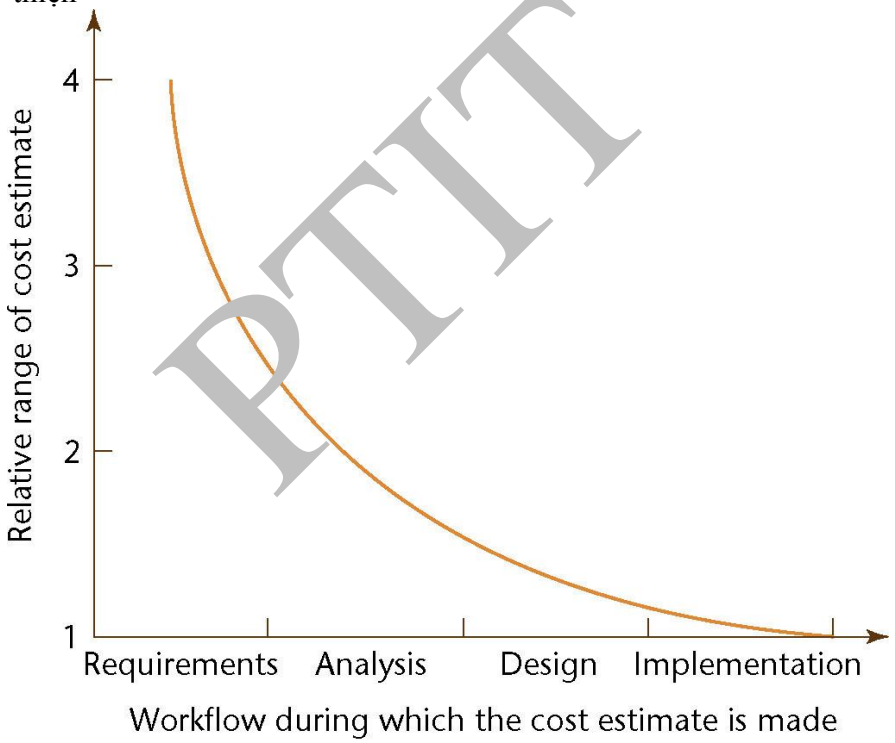
4.4.5 Khi nào kiểm thử dừng

- Chỉ khi sản phẩm phần mềm không thể thay đổi (Only when the product has been irrevocably discarded)

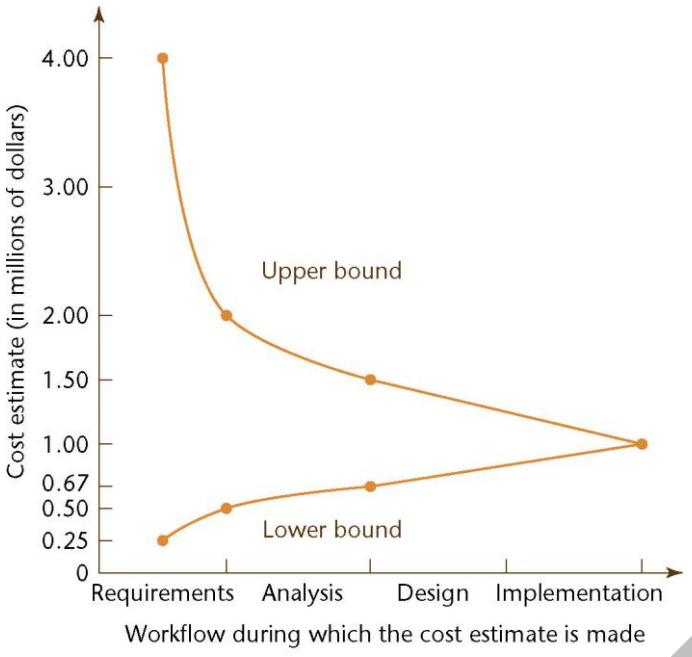
CHƯƠNG 5: LẬP KẾ HOẠCH VÀ ƯỚC LƯỢNG

5.1 VẤN ĐỀ LẬP KẾ HOẠCH VÀ ƯỚC LƯỢNG DỰ ÁN PHẦN MỀM

- Trước khi bắt đầu xây dựng phần mềm, việc lập kế hoạch cho toàn bộ quá trình phát triển một cách chi tiết là cần thiết
- Việc lập kế hoạch được thực hiện liên tục trong suốt quá trình phát triển và bảo trì sau khi chuyển giao
    - Việc lập kế hoạch ban đầu là không đủ
    - Việc lập kế hoạch phải được thực hiện trong suốt dự án
    - Việc lập kế hoạch chi tiết có thể diễn ra sớm nhất có thể sau khi các đặc tả hoàn thiện



- Độ chính xác của ước lượng tăng khi tiến trình phần mềm thực hiện được nhiều
- Ví dụ
    - Chi phí ước lượng 1 triệu đô la trong suốt luồng công việc xác định yêu cầu
        - Chi phí thực tế có thể nằm trong khoảng ($0.25M, $4M)
    - Chi phí ước lượng 1 triệu đô la ở cuối luồng công việc xác định yêu cầu
        - Chi phí thực tế có thể nằm trong khoảng ($0.5M, $2M)
    - Chi phí ước lượng của 1 triệu đô la ở cuối luồng công việc phân tích (thời gian thích hợp sớm nhất)
        - Chi phí thực có thể nằm trong khoảng ($0.67M, $1.5M)



- Mô hình là cũ (1976)
  - Kỹ thuật ước lượng đã cải tiến
  - Nhưng hình dạng của đường cong này là giống với ban đầu

5.2 ƯỚC LƯỢNG THỜI GIAN VÀ CHI PHÍ

- Ước lượng thời gian chính xác là cần thiết
- Ước lượng chi phí chính xác là cần thiết
  - Chi phí bên trong, bên ngoài
- Có quá nhiều thay đổi trong việc ước lượng chính xác chi phí hoặc thời gian

Nhân lực

- Sackman (1968) đo sự khác biệt lên đến 28-1 giữa các cặp lập trình viên(Sackman (1968) measured differences of up to 28 to 1 between pairs of programmers)
  - Ông so sánh sự kết hợp của các cặp lập trình viên êề mặt
    - Kích cỡ phần mềm
    - Thời gian thực thi sản phẩm
    - Thời gian phát triển
    - Thời gian viết mã
    - Thời gian gỡ lỗi
- Những thành viên nhân viên cốt yếu có thể từ bỏ trong suốt dự án

5.2.1 Thước đo kích cỡ của sản phẩm phần mềm

a- Số lượng dòng mã (LOC, KDSI, KLOC)

- Một thước đo khác
  - Số dòng mã nguồn (KDSI)
- Mã nguồn chỉ là một phần nhỏ của nỗ lực phát triển toàn bộ phần mềm (Source code is only a small part of the total software effort)
- Những ngôn ngữ khách nhau dẫn đến chiều dài mã lệnh khác nhau
- LOC không được dùng để xác định ngôn ngữ phi thủ tục (như LISP)

- Các đếm dòng lệnh không rõ ràng
  - Số dòng lệnh có thể thực thi?
  - Những định nghĩa dự liệu?
  - Những lời chú thích?
  - Các câu lệnh JCL?
  - Dòng lệnh đã được thay đổi/ xóa?
- Không phải mọi thứ được viết ra đều chuyển giao cho khách hàng
- Bộ sinh bản ghi, màn ảnh, GUI có thể sinh hàng nghìn dòng lệnh trong mỗi phút
- LOC chỉ được biết chính xác khi phần mềm được hoàn thiện
- Do đó, ước lượng dựa trên LOC nguy hiểm gấp đôi
  - Để bắt đầu tiến trình ước lượng, LOC trong phần mềm đã hoàn thiện phải được ước lượng
  - Sau đó, ước lượng LOC được sử dụng để ước lượng chi phí của phần mềm

b- FFP

- Được sử dụng đối với ước lượng chi phí của một phần mềm xử lý dữ liệu cỡ trung bình
- Ba thành phần cấu trúc cơ bản của pâần mềm xử lý dữ liệu
  - Các tệp
  - Các luồng
  - Các tiến trình
- Với số lượng tệp (*Fi*), số luồng (*Fl*), và số tiến trình (*Pr*)
  - Kích thước (s), chi phí (*c*) được xác định bởi

$$S = Fi + Fl + Pr$$

$$C = b \times S$$

- Hằng số *b* (hiệu năng hoặc năng suất) thay đổi từ tổ chức này đến tổ chức khác
- Tính hiệu lực và tính hữu dụng của thước đo FFP được chứng minh bằng cách sử dụng cùng một mục đích
  - Tuy nhiên, thước đo này không bao giờ được mở rộng đối với cơ sở dữ liệu

c- Điểm chức năng (Function Points)

- Dựa trên số lượng đầu vào (*Inp*), đầu ra (*Out*), các câu hỏi(*Inq*), các tệp chính (*Maf*), các giao diện (*Inf*)
- Đối với bất kỳ sản phẩm nào, số điểm chức năng được xác định bằng
  - $FP = 4 \times Inp + 5 \times Out + 4 \times Inq + 10 \times Maf + 7 \times Inf$
- Đây là một trường hợp quá đơn giản của một tiến trình 3 bước
- Bước 1. phân loại mỗi thành phần của phần mềm (*Inp*, *Out*, *Inq*, *Maf*, *Inf*) thuộc loại đơn giản, trung bình, hoặc phức tạp
  - Gán số lượng thích hợp các điểm chức năng
  - The sum gives UFP (unadjusted function points)

Component	Level of Complexity		
	Simple	Average	Complex
Input item	3	4	6
Output item	4	5	7
Inquiry	3	4	6
Master file	7	10	15
Interface	5	7	10

- Bước 2. Tính toán nhân tố độ phức tạp về mặt kỹ thuật (technical complexity factor -*TCF*)
  - Gián giá trị từ 0 (không có mặt) tới 5 (ảnh hưởng mạnh mẽ từ đầu đến cuối) đối với mỗi nhân tố thuộc 14 nhân tô như tỷ giá giao dịch, tính khả chuyển

1. Data communication
2. Distributed data processing
3. Performance criteria
4. Heavily utilized h dware
5. High transacti rates
6. Online data entry
7. End-use fficiency
8. Onl ne upda ng
9. Co plex comp tations
10. Reusab ity
11. ase of in tallation
12. Ease of operation
13. Portability
14. Maintainability

- Thêm 14 con số
  - Đưa ra mức độ ảnh hưởng tổng thể (*DI*)

$$TCF = 0.65 + 0.01 \times DI$$

- Nhân tố độ phức tạp về mặt kỹ thuật nằm trong khoảng từ 0.65 tới 1.35
- Bước 3. Số lượng điểm chức năng được tính bằng

$$FP = UFP \times TCF$$

- *Phân tích về điểm chức năng*
- Các điểm chức năng thường tốt hơn KDSI – nhưng có một vài vấn đề xảy ra

	Assembler Version	Ada Version
Source code size	70 KDSI	25 KDSI
Development costs	$1,043,000	$590,000
KDSI per person-month	0.335	0.211
Cost per source statement	$14.90	$23.60
Function points per person-month	1.65	2.92
Cost per function point	$3,023	$1,170

- Giống như FFP, bảo trì có thể được đo một cách thiếu chính xác
- Có thể thay đổi
  - Số lượng các tệp, luồng và các tiến chình
  - Số lượng đầu vào, đầu ra, câu hỏi, các tệp chính và các giao diện
- Theo lý thuyết, có thể thay đổi tất cả các dòng mã với việc thay đổi số lượng các dòng mã (In theory, it is possible to change every line of code with changing the number of lines of code)

Các điểm chức năng Mk II

- Thước đo này đã được đưa ra để tính toán UFP một cách chính xác hơn
- Chúng ta chia nhỏ phần mềm thành các giao tác thành phần, mỗi giao tác thành phần gồm đầu vào, tiến trình và đầu ra
- Các điểm chức năng Mark II được sử dụng rộng rãi trên thế giới

d- COCOMO

5.2.2 Các kỹ thuật ước lượng chi phí

a- Những đánh giá của chuyên gia nhờ sự tương tự (Expert judgment by analogy)

- Các chuyên gia so sánh sản phẩm đích với những sản phẩm đã hoàn thiện
  - Sự ước chừng có thể dẫn tới ước lượng chi phí sai
  - Các chuyên gia có thể thu thập lại những thiếu xót của các phần mềm đã hoàn thiện
  - Các chuyên gia con người có nhiều xu hướng (Human experts have biases)
  - Tuy nhiên, kết quả của sự ước lượng bởi một nhóm lớn các chuyên gia có thể chính xác
- Kỹ thuật Delphi đôi khi được sử dụng để đạt được sự đồng thuận nhất trí

b- Phương pháp dưới lên

- Chia sản phẩm phần mềm thành những thành phần nhỏ hơn
  - Các thành phần nhỏ có thể không dễ ước lượng hơn
  - Tuy nhiên, có chi phí mức tiến trình
- Khi sử dụng mô hình hướng đối tượng
  - Sự độc lập của các lớp có ở đây (The independence of the classes assists here)
  - Tuy nhiên, những tương tác giữa các lớp làm rắc rối tiến trình ước lượng

c- Các mô hình ước lượng chi phí thuật toán (Algorithmic cost estimation models)

- Một thước đo được sử dụng như đầu vào đối với mô hình để tính toán chi phí và thời gian
  - Mô hình thuật toán không thiên vị, do đó tốt hơn hẳn ý kiến chuyên gia
  - Tuy nhiên, ước lượng chỉ tốt bằng những giả định tiềm ẩn (However, estimates are only as good as the underlying assumptions)

- Ví dụ
  - SLIM Model
  - Price S Model
  - COnstructive COst MOdel (COCOMO)

5.2.3 COCOMO trung gian

- COCOMO bao gồm 3 mô hình
  - Mô hình ước lượng vĩ mô đối với toàn bộ sản phẩm phần mềm
  - COCOMO trung gian
  - Mô hình ước lượng vi mô xem xét chi tiết phần mềm
- Chúng ta nghiên cứu COCOMO trung gian
- Bước 1. Ước lượng chiều dài của phần mềm trong KDSI
- Bước 2. Ước lượng chế độ phát triển phần mềm (có tổ chức (organic), nửa tách rời (semidetached), nhúng(embedd))
- Ví dụ:
  - Phần mềm không phức tạp (chế độ organic)
    - (Công sức danh nghĩa)Nominal effort = $3.2 \times (\text{KDSI})^{1.05}$ person-months
- Bước 3. Tính toán công sức danh nghĩa
- Ví dụ:
  - Phần mềm có tổ chức (Organic product)
  - 12,000 câu lệnh được chuyển giao (= 12 KDSI) (đã ước lượng)
  - (Công sức danh nghĩa)Nominal effort = $3.2 \times (12)^{1.05}$ = 43 person-months
- Bước 4. Nhân giá trị danh nghĩa với 15 lần chi phí phát triển phần mềm (Step 4. Multiply the nominal value by 15 software development cost multipliers)

	Rating					
Cost Drivers	**Very Low**	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
Product Attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Database size		0.94	1.00	1.08	1.16	
Product complexity	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
Execution time constraint			1.00	1.11	1.30	1.66
Main storage constraint			1.00	1.06	1.21	1.56
Virtual machine volatility*		0.87	1.00	1.15	1.30	
Computer turnaround time		0.87	1.00	1.07	1.15	
Personnel Attributes						
Analyst capabilities	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Programmer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience*	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project Attributes						
Use of modern programming practices	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

*For a given software product, the underlying virtual machine is the complex of hardware and software (operating system, database management system) it calls on to accomplish its task.

- Ví dụ:

- Phần mềm xử lý giao tiếp dựa trên bộ vi xử lý cho mạng chuyển tiền điện tử với độ tin cậy, hiệu năng, lịch phát triển và các yêu cầu giao diện cao (Microprocessor-based communications processing software for electronic funds transfer network with high reliability, performance, development schedule, and interface requirements)

- Bước 1. Ước lượng chiều dài của phần mềm sản phẩm
  - 10,000 câu lệnh được chuyển giao (10 KDSI)
- Bước 2. Ước lượng chế độ phát triển
  - Chế độ phức tạp ("nhúng"-"embedded")
- Bước 3. Tính công sức danh nghĩa
  - (Công sức danh nghĩa)Nominal effort = $2.8 \times (10)^{1.20}$ = 44 person-months
- Bước 4. Nhân giá trị danh nghĩa với 15 lần chi phí phát triển phần mềm
  - Product of effort multipliers = 1.35 (Software development effort multipliers)

Cost Drivers	Situation	Rating	Effort Multiplier
Required software reliability	Serious financial consequences of software fault	High	1.15
Data base size	20,000 bytes	Low	0.94
Product complexity	Communica[illegible] processing	Very high	1.30
Execution time constraint	Will use 7[illegible]% of a[illegible]lable time	High	1.11
Main storage constraint	45K [illegible] 64K store (70	High	1.06
Virtual machine volatility	Based [illegible] commercial microp[illegible]ssor hardware	Nominal	1.00
Computer turnaround time	2 hour averag[illegible]urnaround time	Nominal	1.00
Analyst capabilities	G[illegible] senior analysts	High	0.86
Applications experience	3 yea[illegible]	Nominal	1.00
Programmer capabilit[illegible]	Good senior programmers	High	0.86
Virtual machine experienc[illegible]	6 months	Low	1.10
Programming language expe[illegible]ce	12 months	Nominal	1.00
Use of modern programming practices	Most techniques in use over 1 year	High	0.91
Use of software tools	At basic minicomputer tool level	Low	1.10
Required development schedule	9 months	Nominal	1.00

- Do đó, Ước lượng công sức cho dự án 1.35 x 44 = 59 person-months. Ước lượng công sức cho dự án (59 person-months) được sử dung là đầu vào cho công thức bổ sung đối với
  - Chi phí đô la
  - Lịch biểu phát triển
  - Phân phối pha và hoạt động
  - Chi phí máy tính
  - Chi phí bảo trì hàng năm
  - Các mục liên quan
- COCOMO trung gian đã được xác nhận với một mẫu lớn (Intermediate COCOMO has been validated with respect to a broad sample)
- Giá trị thực nằm trong khoảng 20% giá trị dự đoán và khoảng 68% thời gian
  - COCOMO trung gian là phương thức ước lượng chính xác nhất về thời gian của nó

- Vấn đề chính
  - Nếu ước lượng số lượng dòng mã của sản phẩm đích là sai, thì mọi thứ đều sai

5.2.4 COCOMO II

- 1995 đã mở rộng COCOMO 1981 với sự kết hợp
  - Hướng đối tượng
  - Mô hình vòng đời hiện đại
  - Bản mẫu nhanh
  - Ngôn ngữ thế hệ thứ tư
  - Phần mềm COTS
- COCOMO II phức tạp hơn nhiều so với phiên bản đầu
- Có 3 mô hình khác nhau
  - Mô hình kết hợp ứng dụng đối với các pha ban đầu (Application composition model for the early phases)
    - Dựa trên các điểm đặc trưng (tương tự với điểm chức năng)
  - Mô hình thiết kế sớm
    - Dựa trên các điểm chức năng
  - Mô hình hậu kiến trúc (Post-architecture model)
    - Dựa trên các điểm chức năng hoặc KDSI
- Mô hình công sức COCOMO cơ bản
  - effort = $a \times (\text{size})^b$
  - COCOMO trung gian
    - Ba giá trị đối với (a_i, b_i)
  - COCOMO II
    - *b thay đổi dựa vào giá trị của các biến xác định*
- COCOMO II hỗ trợ sử dụng lại
- COCOMO II has 17 multiplicative cost drivers (was15)
  - Seven are new
- It is too soon for results regarding
  - The accuracy of COCOMO II
  - The extent of improvement (if any) over Intermediate COCOMO

5.2.5 Theo dõi ước lượng thời gian và chi phí

- Sử dụng bất cứ phương thức ước lượng nào thì việc theo dõi cũng là quan trọng
- Công việc được làm
  - Những tài nguyên để thực hiện công việc đó
- Số tiền để chi trả cho công việc đó

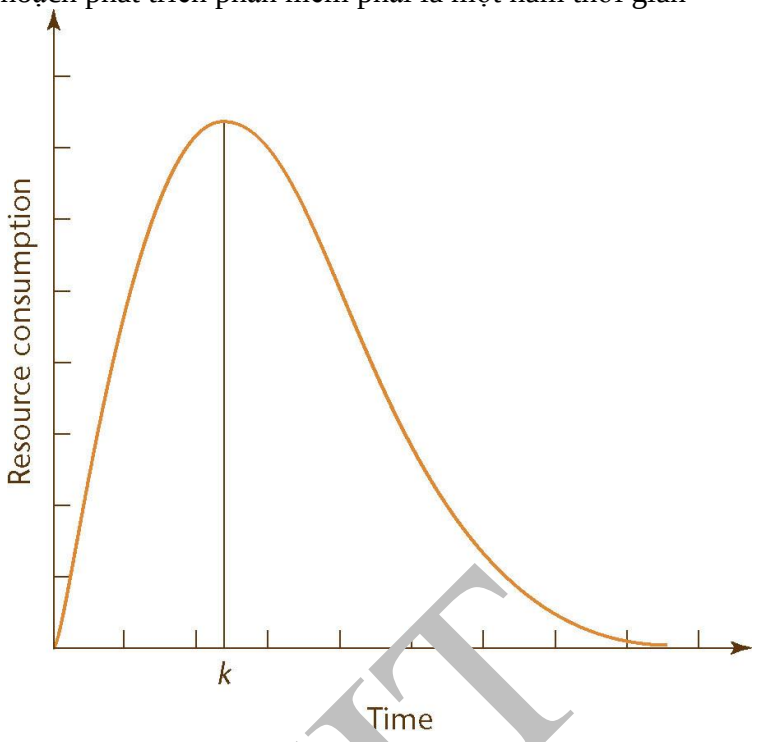
Các tài nguyên

- Các tài nguyên cần thiết cho phát triển phần mềm:
  - Con người
  - Phần cứng
  - Phần mềm hỗ trợ

Việc sử dụng các các loại tài nguyền với thời gian

- Đường cong Rayleigh miêu tả một cách chính xác việc giả định về tài nguyên

- Tòan bộ kế hoạch phát triển phần mềm phải là một hàm thời gian



Các loại công việc

- Chức năng dự án
  - Công việc được thực hiện xuyên suốt dự án
  - Ví dụ:
    - Quản lý dự án
    - Điều khiển chất lượng
- Hoạt động (activity)
  - Công việc liên quan với một pha cụ thể
  - Đơn vị chính của công việc,
  - Với ngày tháng bắt đầu và kết thúc chính xác,
  - Giả định về thời gian và
  - Các sản phẩm công việc như ngân sách, thiết kế, lập lịch, mã nguồn, hoặc sổ tay người dùng
- Nhiệm vụ (task)
  - Các hoạt động bao gồm tập các nhiệm vụ (đơn vị nhỏ nhất của công việc có thể quản lý được)

Sự hoàn thành của các sản phẩm công việc

- *Mốc quan trọng (Milestone)*: là ngày mà sản phẩm công việc được hoàn thiện
- Nó phải chuyển qua và thực hiện rà soát bởi
  - Các thành viên trong đội
  - Quản lý
  - *Khách hàng*
- Khi sản phẩm công việc được rà soát và được đồng ý, thì nó trở thành một phiên bản cơ sở

Gói công việc

- Sản phẩm công việc, cùng với

- Biên chế các yêu cầu
- Thời gian
- Tài nguyên
- Gán trách nhiệm cho các cá nhân
- Tiêu chấp nhận sản phẩm công việc
- Ngân sách chi tiết là hàm thời gian, chỉ định
  - Các chức năng dự án
  - Các hoạt động

5.3 CÁC THÀNH PHẦN CỦA VIỆC LẬP KẾ HOẠCH PHÁT TRIỂN PHẦN MỀM

5.3.1Khung kế hoạch quản lý dự án phần mềm(SPMP)

- Có 3 cách để xây dựng SPMP
- Một cách tốt nhất là chuẩn IEEE 1058.1
  - Chuẩn được chấp nhận rộng rãi
  - Nó được thiết kế để sử dụng với tất cả các loại sản phẩm phần mềm
  - Nó hỗ trợ cải thiện tiến trình
    - Many sections reflect CMM key process areas
  - Là lý tưởng đối với quy trình hợp nhất
    - Có những phần để điều khiển các yêu cầu và quản lý rủi ro
- Một trong những phần không được áp dụng trong các phần mềm cỡ nhỏ
  - Ví dụ: kế hoạch quản lý nhà thầu phụ

5.3.2 Kế hoạch quản lý dự án phần mềm IEEE

1 Overview
- 1.1 Project summary
  - 1.1.1 Purpose, scope, and objectives
  - 1.1.2 Assumptions and constraints
  - 1.1.3 Project deliverables
  - 1.1.4 Schedule and budget summary
- 1.2 Evolution of the project management plan

2 Reference materials

3 Definitions and acronyms

4 Project organization
- 4.1 External interfaces
- 4.2 Internal structure
- 4.3 Roles and responsibilities

5 Managerial process plans
- 5.1 Start-up plan
  - 5.1.1 Estimation plan
  - 5.1.2 Staffing plan
  - 5.1.3 Resource acquisition plan
  - 5.1.4 Project staff training plan
- 5.2 Work plan
  - 5.2.1 Work activities
  - 5.2.2 Schedule allocation
  - 5.2.3 Resource allocation
  - 5.2.4 Budget allocation
- 5.3 Control plan
  - 5.3.1 Requirements control plan
  - 5.3.2 Schedule control plan
  - 5.3.3 Budget control plan
  - 5.3.4 Quality control plan
  - 5.3.5 Reporting plan
  - 5.3.6 Metrics collection plan
- 5.4 Risk management plan
- 5.5 Project close-out plan

6 Technical process plans
- 6.1 Process model
- 6.2 Methods, tools, and techniques
- 6.3 Infrastructure plan
- 6.4 Product acceptance plan

7 Supporting process plans
- 7.1 Configuration management plan
- 7.2 Testing plan
- 7.3 Documentation plan
- 7.4 Quality assurance plan
- 7.5 Reviews and audits plan
- 7.6 Problem resolution plan
- 7.7 Subcontractor management plan
- 7.8 Process improvement plan

8 Additional plans

5.3.3 Việc lập kế hoạch kiểm thử

- SPMP phải chỉ ra rõ ràng những gì kiểm thử phải làm
- Việc theo dõi kiểm tra là cần thiết
- Tất cả các trường hợp kiểm thử hộp đen có thể được phác thảo ra sớm nhất có thể sau khi các đặc tả hoàn thiện

5.3.4 Yêu cầu đào tạo

- "Chúng ta không cần lo lắng về việc đào tạo đến tận khi phần mềm hoàn thiện, và sau đó chúng ta có thể đào tạo người dùng"
- Việc đào tạo nhìn chung được yêu cầu bởi các thành viên của nhóm phát triển, bắt đầu với đào tạo trong việc lập kế hoạch phần mềm
- Một phương thức phát triển phần mềm mới đòi hỏi phải đào tạo mọi thành viên trong nhóm phát triển
- Việc giới thiệu các công cụ phần cứng hoặc phần mềm của bất cứ loại nào đều đòi hỏi phải có đào tạo
- Những người lập trình có thể yêu cầu đào tạo về các hệ điều hành và/hoặc ngôn ngữ cài đặt
- Việc đào tạo chuẩn bị tài liệu có thể được yêu cầu
- Các thao tác máy tính đòi hỏi phải được đào tạo

5.3.5 Các chuẩn tài liệu

- Số lượng tài liệu được sinh ra bởi một phần mềm?
    - Phần mềm thương mại bên trong IBM (50 KDSI)
        - 28 trang tài liệu trên KDSI
    - Phần mềm thương mại cùng kích cỡ
        - 66 trang tài liệu trên KDSI
    - IMS/360 Version 2.3 (about 166 KDSI)
        - 157 trang tài liệu trên KDSI
    - [TRW] với 100 giờ sử dụng cho hoạt động viết mã, 150-200 giờ được sử dụng cho các hoạt động liên quan tới việc viết tài liệu
- Lập kế hoạch
- Điều khiển
- Tài chính
- Kỹ thuật
- Mã nguồn
- Những lời chú thích với mã nguồn

Ưu điểm của các chuẩn tài liệu

- Giảm hiểu lầm giữa các thành viên trong đội
- Trợ giúp SQA
- Chỉ những nhân viên mới phải học các chuẩn
- Các chuẩn trợ giúp những người lập trình bảo trì
- Việc chuẩn hóa là quan trọng đối với các sổ tay người dùng
- Là một phần của tiến trình lập kế hoạch
    - Các chuẩn phải được thiết lập đối với tất cả các tài liệu
- Sản phẩm là tài liệu

5.3.6 Công cụ CASE cho việc lập kế hoạch và ước lượng

- Rất cần thiết để có
  - Một bộ xử lý từ, và
  - Một bảng tính
- Hiện nay có công cụ COCOMO and COCOMO II
- Công cụ quản lý trợ giúp việc lập kế hoạch và giám sát
  - MacProject
  - Microsoft Project

5.4 KIỂM THỬ VIỆC LẬP KẾ HOẠCH

- Chúng ta phải kiểm tra tòan bộn SPMP
  - Đặc biệt chú ý tới ước lượng thời gian và chi phí

5.5 LẬP KẾ HOẠCH CHO CÁC DỰ ÁN PHẦN MỀM HƯỚNG ĐỐI TƯỢNG

- Phần mềm hướng đối tượng bao gồm các phần độc lập với nhau
- Do đó, việc lập kế hoạch phần nào dễ dàng hơn
- Việc lập kế hoạch toàn bộ nhiều hơn tổng việc lập kế hoạch của các
- Chúng ta có thể sử dụng COCOMO II
- Tuy nhiên, sử dụng lại bao gồm những sai sót trong ước lượng chi phí và thời gian
  - Sử dụng lại những thành phần sẵn có trong suốt quá trình phát triển
  - Sản phẩm của các thành phần để sử dụng lại trong tương lai
- Những công việc này theo các hướng đối nghịch nhau
- Dữ liệu mới hơn: việc lưu trữ vượt quá chi phí

CHƯƠNG 6: PHA XÁC ĐỊNH YÊU CẦU

6.1 XÁC ĐỊNH YÊU CẦU CỦA KHÁCH HÀNG

- Hiểu sai
    - Chúng ta phải xác định những gì khách hàng muốn
- "Tôi biết bạn tin rằng bạn đã hiểu những gì bạn nghĩ là tôi đã nói, nhưng tôi không chắc bạn nhận ra rằng những gì bạn nghe không phải là điều mà tôi muốn nói!" ("I know you believe you understood what you think I said, but I am not sure you realize that what you heard is not what I meant!")
- Chúng ta phải xác định những gì khách hàng cần
- Rất khó để người phân tích một hệ thống để hình dung ra một sản phẩm phần mềm và các chức năng của nó
    - Vấn đề này khó với khách hàng
- Một người phân tích hệ thống có kinh nghiệm cần làm rõ những thông tin thích hợp cho khách hàng
- Khách hàng là nguồn duy nhất của thông tin này
- Giải pháp:
    - Thu thập những thông tin ban đầu từ khách hàng
    - Sử dụng những thông tin ban đầu giống như đầu vào của quy trình hợp nhất
    - Theo sát các bước của quy trình hợp nhất để xác định các nhu cầu thực của khách hàng

6.2 TỔNG QUAN VỀ LUỒNG CÔNG VIỆC XÁC ĐỊNH YÊU CẦU

Mục đích của luồng công việc xác định yêu cầu

- Để trả lời câu hỏi:

- Sản phẩm phần mềm phải có khả năng làm được những gì?

Nội dung về luồng công việc xác định yêu cầu

- Đầu tiên, hiểu được lĩnh vực ứng dụng
  - Môi trường cụ thể mà sản phẩm phần mềm đích hoạt động
- Thứ hai, xây dựng một mô hình nghiệp vụ
  - Mô hình các tiến trình nghiệp vụ của khách hàng
- Thứ ba, sử dụng mô hình nghiệp vụ để xác định các yêu cầu khách hàng
- Lặp lại các bước trên

Các định nghĩa

- Tìm ra các yêu cầu của khách hàng
  - Thu thập các yêu cầu
  - Các phương thức bao gồm phỏng vấn và điều tra
- Làm mịn và mở rộng những yêu cầu ban đầu
  - *Phân tích yêu cầu*

6.2.1 Hiểu lĩnh vực ứng dụng

- Mỗi thành viên của dự án phát triển phải trở nên quen thuộc với lĩnh vực ứng dụng
  - Thuật ngữ chính xác là cần thiết
- Xây dựng thuật ngữ
  - Một danh sách các từ kỹ thuật được sử dụng trong lĩnh vực ứng dụng và ý nghĩa của nó

6.2.2 Mô hình nghiệp vụ

- Một mô hình nghiệp vụ là sự miêu tả các tiến trình nghiệp vụ của một tổ chức
- Mô hình nghiệp vụ đưa ra cách hiểu về toàn bộ nghiệp vụ của khách hàng
  - Tri thức này là cần thiết để đưa ra lời khuyên cho khách hàng về mặt tính toán
- Các nhà phân tích hệ thống cần thu thập một cách hiểu chi tiết về các loại tiến trình nghiệp vụ khác nhau.
  - Các kỹ thuật khác nhau được sử dung, ban đầu là phỏng vấn

6.2.2.1 Phỏng vấn

- Đội xác định yêu cầu cần gặp gỡ khách hàng và người dùng để thu thập được những thông tin liên quan
- Có hai loại câu hỏi:
    - *Câu hỏi kết thúc đóng (Close-ended* question)s*yêu cầu một câu trả lời cụ thể*
    - *Câu hỏi kết thúc mở (Open-ended* questions*) khuyến khích người được phỏng vấn nói thẳng ý kiến của mình*
- Có hai kiểu phỏng vấn
    - Trong cuộc phỏng vấn có cấu trúc, các câu hỏi đã được lập kế hoạch cụ thể từ trước và thường là những câu hỏi kết thúc đóng
    - Trong cuộc phỏng vấn không cấu trúc, các câu hỏi được đưa ra để phản ứng lại những câu trả lời đã nhận được, thường xuyên là câu hỏi với kết thúc mở
- Việc phỏng vấn là không dễ dàng
    - Một cuộc phỏng vấn mà không có cấu trúc sẽ không sinh ra thông tin liên quan
    - Người phỏng vấn phải quen thuộc với lĩnh vực ứng dụng
    - Người phỏng vấn phải sẵn sàng tiếp thu cái mới ở mọi lúc (The interviewer must remain open-minded at all times)
- Sau khi phỏng vấn, người phỏng vấn phải chuẩn bị một bài tường trình đã được viết ra
    - Nên đưa một bản sao của bản tường trình cho người được phỏng vấn

6.2.2.2 Các kỹ thuật khác

- Phỏng vấn là kỹ thuật chính
- Một bản thăm dò ý kiến rất hữu ích khi lấy ý kiến của hàng trăm người.
- Kiểm tra các định dạng nghiệp vụ mà chỉ ra cách khách hàng thực hiện những công việc nghiệp vụ (Examination of business forms shows how the client currently does business)
- Quan sát trực tiếp những người công nhân thực hiện những nhiệm vụ của họ có thể là một cách rất hữu ích
    - Máy quay là một phiên bản hiện đại của kỹ thuật này
    - Nhưng, cần rất nhiều thời gian để phân tích các băng video

- Những người công nhân có thể xem máy quay vì sự xâm phạm tùy tiện đời sống riêng tư

6.2.3 Các use case

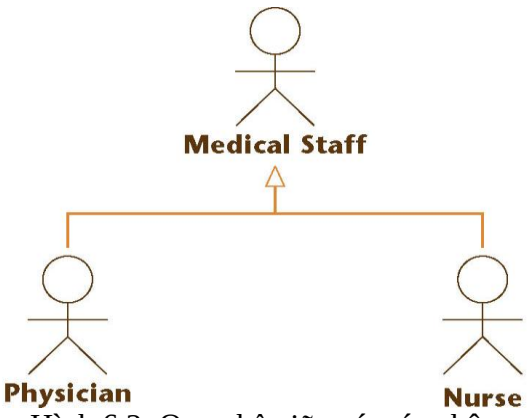
- Một use case mô hình tương tác giữa sản phẩm phần mềm với người dùng sản phẩm phần mềm đó (tác nhân - *actors*)
- Ví dụ:

Hình 6.1: Biểu diễn một use case

- Một tác nhân là một thành viên của thế giới bên ngoài sản phẩm phần mềm
- Thường rất dễ dàng nhận dạng ra tác nhân
  - Một tác nhân thường là một người dùng của hệ thống sản phẩm phần mềm
- Nhìn chung, một tác nhân đóng vai trò đối với hệ thống sản phẩm phần mềm. Vai trò này gồm:
  - Là một người dùng; hoặc
  - Là một khởi đầu; hoặc
  - Là một người nào đó đóng vài trò quan trọng trong use case
- Một người dùng của hệ thống có thể giữ nhiều hơn một vai trò
- Ví dụ: Một người khách hàng (**Customer**) của ngân hàng có thể là
  - Một người vay tiền hoặc
  - Một người cho mượn
- Ngược lại, một tác nhân có thể tham gia vào nhiều use case
- Ví dụ: một người vay tiền (**Borrower**)có thể là một tác nhân trong
  - Use case Borrow Money ;
  - Use case Pay Interest on Loan

- Use case Repay Loan Principal

- Tác nhân người vay tiền (**Borrower**)có thể đại diện cho hàng nghìn khách hàng của ngân hàng
- Một tác nhân không cần thiết phải là một con người
- Ví dụ: hệ thống thông tin thương mại điện tử phải tương tác với hệ thống thông tin công ty thẻ tín dụng
  - Hệ thống thông tin công tin thẻ tín dụng là một tác nhân từ quan điểm của hệ thống thương mại điện tử
  - Hệ thống thương mại điện tử là một tác nhân của hệ thống thông tin công ty thẻ tín dụng
- Vấn đề dễ xảy ra khi xác định các tác nhân
  - Nạp chồng tác nhân
- Ví dụ: Hệ thống phần mềm bệnh viện
  - Một use case có tác nhân **Y tá** (Nurse)
  - Một use case khác có tác nhân **Nhân viên Y khoa (Medical Staff)**
  - Tốt hơn:
    - Các tác nhân:: Bác Sỹ và Y tá (**Physician** and **Nurse)**
- Về mặt giải pháp:
  - Tác nhân **Nhân viên Y khoa (Medical Staff**) với hai sự chuyên môn hóa: Bác sỹ và Y tá (**Physician** and **Nurse)**



Hình 6.2: Quan hệ giữa các tác nhân

6.2.4 Các yêu cầu ban đầu

- Những yêu cầu ban đầu dựa trên mô hình nghiệp vụ đầu tiên
- Sau đó chúng được làm mịn
- Các yêu cầu là động – có sự thay đổi thường xuyên
    - Duy trì một danh sách những yêu cầu quan trọng, cùng với các use case của các yêu cầu đã được phê chuẩn bởi khác hàng
- Có hai loại yêu cầu
- Yêu cầu chức năng chỉ rõ hành động mà sản phẩm phần mềm phải có khả năng thực hiện
    - Thường được biểu diễn về mặt các đầu vào và đầu ra
- Các yêu phi chức năng chỉ rõ những đặc trưng của hệ thống sản phẩm phần mềm, như
    - Những ràng buộc về môi trường
    - Thời gian đáp ứng
    - Tính đáng tin
- Những yêu cầu chức năng được xử lý như là một phần của luồng công việc xác định yêu cầu và phân tích
- Một số yêu cầu phi chức năng phải phải chờ đến tận đến luồng công việc thiết kế
    - Thông tin chi tiết của những yêuc hầu phi chức năng không có sẵn cho đến tận khi luồng công việc phân tích và xác định yêu cầu hoàn thành

6.3 PHA XÁC ĐỊNH YÊU CẦU CỔ ĐIỂN

- Không phải thực hiện pha xác định yêu cầu cổ điển trong "xác định yêu cầu hướng đối tượng"
    - Luồng công việc xác định yêu cầu không phải làm gì khi sản phẩm phần mềm được xây dựng
- Tuy nhiên, phương pháp này được biểu diễn trong chương này là
    - Hướng mô hình và do đó
    - Hướng đối tượng
- Phương pháp cổ điển để xác định yêu cầu

- Làm rõ các yêu cầu
- Phân tích yêu cầu
- Xây dựng bản mẫu nhanh
- Khách hàng và người dùng trong tương lai thử nghiệm với bản mẫu nhanh

6.4 BẢN MẪU NHANH

- Xây dựng nhanh (“rapid”)
  - Sự hoàn thiện có thể được bỏ qua
- Chỉ đưa ra những chức năng chính
- Nhấn mạnh chỉ những gì mà khách hàng xem
  - Kiểm tra lỗi, cập nhật tệp có thể được bỏ qua
- Mục đích:
  - Để cung cấp tới khách hàng cách hiểu của sản phẩm phần mềm
- Bản mẫu nhanh được xây dựng để thay đổi
  - Ngôn ngữ cho bản mẫu nhanh bao gồm 4GLs và ngôn ngữ đã được thông dịch

6.5 NHÂN TỐ CON NGƯỜI

- Khách hàng và những người dùng có ý định sử dụng hệ thống phải tương tác với giao diện người dùng
- Giao diện máy tính – con người (Human-computer interface HCI)
  - Bảng chọn, không dòng lệnh (Menu, not command line)
  - “Trỏ và nhấp”(“Point and click”)
  - Cửa sổ, biểu tượng, bảng chọn kéo xuống (Windows, icons, pull-down menus)
- Nhân tố con người phải được xem xét
  - Tránh bảng đơn dài dòng
  - Cho phép mức thay đổi mức độ thành thạo của giao diện
  - Tính đồng đều của hình thức là quan trọng (Uniformity of appearance is important)

- Tâm lý tiến bộ so với cảm giác chung (Advanced psychology vs. common sense?)

- Bản mẫu nhanh của giao diện người máy của õỗi sản phẩm phần mềm là bắt buộc

6.6 SỬ DỤNG LẠI BẢN MẪU NHANH

- Việc sử dụng lại bản mẫu nhanh là về bản chất là mô hình xây và sửa
- Những thay đổi được đưa ra để xây dựng phần mềm
  - Đắt (Expensive)
- Bảo trì khó vì không có tài liệu đặc tả và tài liệu thiết kế
- Những ràng buộc về thời gian thực khó đáp ứng
- Một cách để đảm bảo rằng bảng mẫu nhanh được bỏ qua
  - Cài đặt bản mẫu nhanh bằng một ngôn ngữ khác so với ngôn ngữ đã lựa chọn cho sản phẩm đích(Implement it in a different language from that of the target product)
- Mã được sinh ra có thể được sử dụng lại
- Chúng ta có thể giữ lại một cách an toàn (các phần của bản mẫu nhanh) bản mẫu nhanh nếu
  - Điều này được chuẩn bị trước
  - Những phần của bản mẫu nhanh đã qua kiểm tra kỹ lưỡng của nhóm SQA
  - Tuy nhiên, đây không phải là một bản mẫu nhanh cổ điển ("classical" rapid prototyping)

6.7 CÁC CÔNG CỤ CASE CHO XÁC ĐỊNH YÊU CẦU

- Chúng ta cần các công cụ đồ họa cho các biểu đồ UML
  - Dễ dàng để thay đổi các biểu đồ UML
  - Tài liệu được lưu trong các công cụ và luôn có sẵn
- Các công cụ như vậy đôi khi rất khó sử dụng
- The diagrams may need considerable "tweaking"
- Nhìn chung, điểm mạnh có nhiều ảnh hưởng tốt hơn điểm yếu (Overall, the strengths outweigh the weaknesses)
- Các môi trường CASE đồ họa được mở rộng để trợ giúp UML

  - System Architect
  - Software through Pictures
- Các môi trường CASE hướng đối tượng bao gồm
  - IBM Rational Rose
  - Together
  - ArgoUML (open source)

6.8 CÁC THƯỚC ĐO CHO XÁC ĐỊNH YÊU CẦU

- Volatility and speed of convergence are measures of how rapidly the client's needs are determined
- Số lượng thay đổi được đưa ra trong suốt chuỗi các pha
- Những thay đổi được đề xướng bởi những người phát triển
  - Quá nhiều thay đổi có thể đồng nghĩa với việc quy trình không hoàn thiện
  - Nhữung thay đổi được đề xướng bởi khách hàng
  - Thay đổi san phẩm phần mềm cuối cùng

6.9 NHỮNG THỬ THÁCH CHO PHA XÁC ĐỊNH YÊU CẦU

- Nhân viên của tổ chức khách hàng thường cảm giác bị đe dọa bởi máy tính
- Những thành viên đội xác định yêu cầu phải có khả năng thương lượng
  - Yêu cầu của khách hàng có thể phải thu hẹp phạm vi
- Nhân viên chính của tổ chức khách hàng không thể có thời gian cho những cuộc thảo luận cốt yếu và sâu sắc
- Linh hoạt và khách quan là cốt yếu

6.10 CASE STUDY CHO PHA XÁC ĐỊNH YÊU CẦU

6.10.1 Bài toán

Khách hàng đến dặt hàng chúng ta xây dựng một phần mềm quản lí khách sạn với yêu cầu như sau (đây có thể coi như là một bản mô tả yêu cầu của khách hàng bằng ngôn ngữ tự nhiên):

- Phần mềm dạng ứng dụng cho máy tính cá nhân, chỉ có nhân viên lễ tân, nhân viên bán hàng, quản lí khách sạn được sử dụng
- Nhân viên lễ tân có thể tìm phòng trống theo yêu cầu trực tiếp của khách, checkin cho khách đã đặt phòng hoặc đặt phòng trực tiếp, checkout cho khách và in hóa đơn thanh toán cho khách
- Nhân viên bán hàng có thể tìm phòng trống và đặt phòng theo yêu cầu của khách.
- Quản lí có thể : thêm/sửa/xóa thông tin phòng, xem các báo cáo doanh thu theo thời gian/theo phòng/theo loại phòng, xem báo cáo tỉ lệ phòng trống theo thời gian/theo phòng/theo loại phòng, xem báo cáo khách hàng đặt nhiều theo thời gian/theo nguồn gốc khách hàng.
- Thông tin về khách sạn bao gồm : tên, địa chỉ, số sao, mô tả (bao gồm mô tả bằng text và bằng hình ảnh).
- Trong khách sạn có nhiều phòng, mỗi phòng được mô tả bằng các thông tin : tên phòng (duy nhất, để phân biệt các phòng), loại phòng, giá niêm yết, các loại dịch vụ đi kèm, mô tả phòng.
- Mỗi khách hàng, khi đến ở hoặc đặt phòng, sẽ được lưu các thông tin bao gồm số CMND (số passport nếu là người nước ngoài), loại giấy tùy thân (CMND, passport), họ tên đầy đủ, địa chỉ, số điện thoại, ghi chú về phục vụ đặc biệt như cho người khuyết tật, ăn chay...
- Mỗi phòng có thể được đặt ở bởi nhiều khách hàng khác nhau tại những thời điểm khác nhau.
- Mỗi khách hàng có thể đặt/ở nhiều phòng khác nhau tại những thời điểm khác nhau.
- Tại một thời điểm, chỉ có một khách ở trong một phòng, và xác định một giá phòng cụ thể.
- Khách hàng chỉ có thể đặt phòng nếu phòng đó còn trống trong suốt thời gian khách hàng muốn đặt.
- Khách hàng có thể thanh toán nhiều lần cho đến ngày trả phòng.
- Mỗi lần thanh toán, lễ tân sẽ in hóa đơn cho lần thanh toán đó bao gồm các thông tin : họ tên và địa chỉ khách hàng, số phòng, ngày đến, ngày đi, giá phòng, các dịch vụ đi kèm (mỗi dịch vụ bao gồm tên dịch vụ, đơn vị tính, đơn giá, tổng tiền), số tiền thanh toán.
- Khách hàng có thể hủy đặt phòng (miễn phí) nếu hủy trước ngày đến. Nếu khách hàng hủy sau ngày đặt thì khách hàng bị lưu vào danh sách đen và có thể bị từ chối đặt phòng trong các lần tiếp theo.

6.10.2 Xây dựng sơ đồ use case tổng quan

Mục đích của bước này là xây dựng một bản mô tả yêu cầu của khách hàng bằng ngôn ngữ kỹ thuật (UML).

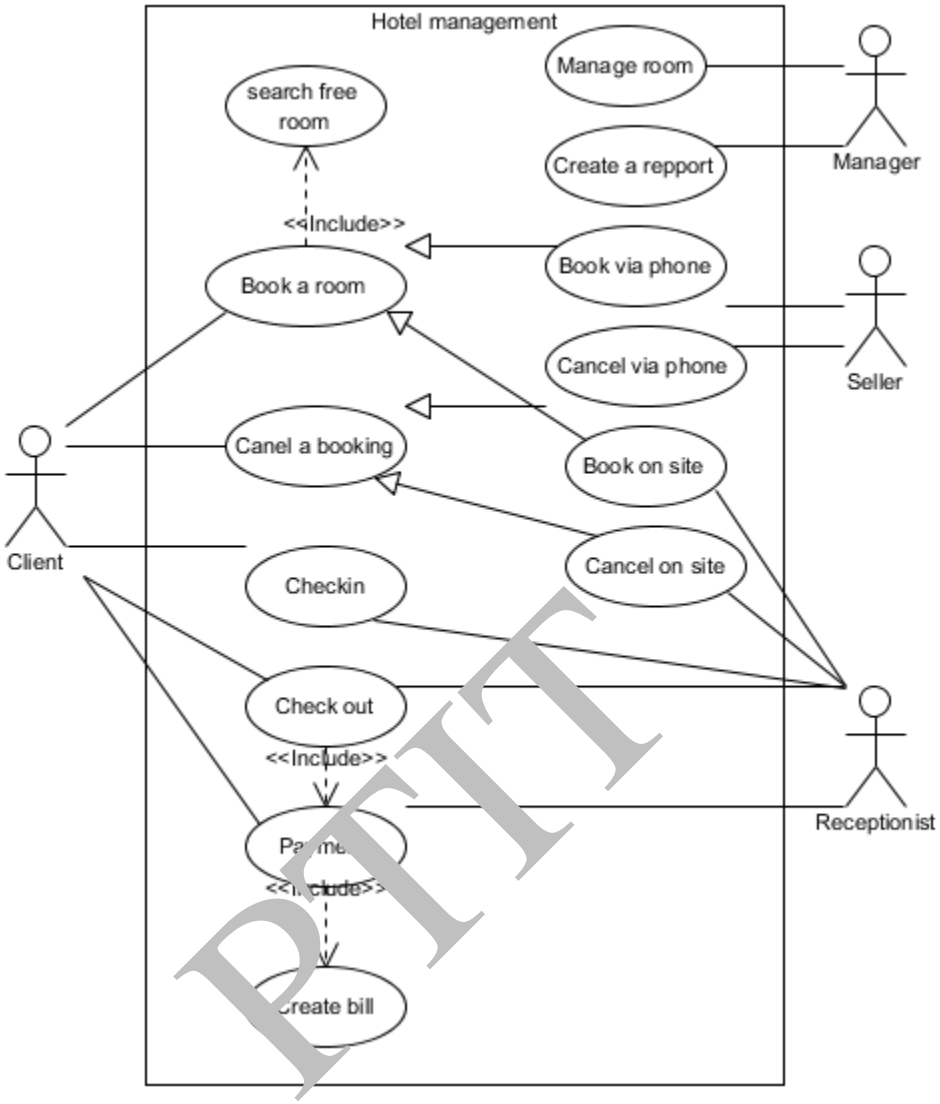
Xác định các actor có thể có của hệ thống:

- Actor là người dùng trực tiếp: người quản lí khách sạn (manager), nhân viên bán hàng (saller), nhân viên lễ tân kiêm luôn thủ quỹ để nhận thanh toán (receptionist)
- Actor là người dùng gián tiếp: Khách hàng (client), mặc dù không trực tiếp sử dụng và thao tác trên phần mềm, nhưng một số chức năng phải có mặt khách hàng mới thực hiện được như: đặt chỗ, checkin, checkout, thanh toán.

Các chức năng liên quan đến các actor:

- Người quản lí khách sạn (Manager): quản lí thông tin phòng và khách sạn (room manage), tạo và xem các loại báo cáo (create report)
- Nhân viên bán hàng (Saller): giao dịch với khách hàng (Client) qua điện thoại để đặt chỗ (Book a room) hoặc hủy đặt chỗ (Cancel a booking)
- Nhân viên tiếp tân (Receptionist): giao dịch trực tiếp với khách hàng (Client) tại quầy để đặt chỗ (Book a room), hủy đặt chỗ (Cancel a booking), nhận Checkin, Checkout và thanh toán cho khách hàng.
- Khách hàng (Client): có thể đặt phòng/hủy phòng (Book a room/Cancel a Booking) trực tiếp tại quầy với nhân viên lễ tân hoặc đặt/hủy qua điện thoại với nhân viên bán hàng. Checkin, Checkout và thanh toán tại quầy với nhân viên lễ tân.

Như vậy nhóm dự án thu được sơ đồ use case tổng quan như sau:

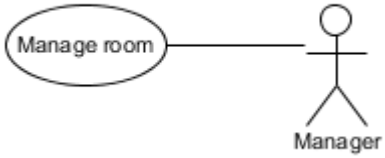


Hình 6.3: Sơ đồ use case tổng quan của hệ thống

6.10.3 Mô tả các use case

Mục đích của bước này là mô tả chi tiết các use case đã xác định được trong sơ đồ tổng quan.

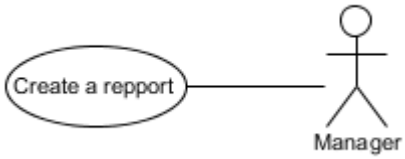
a. Use case Room manage



Hình 6.4: Use case room manage

Mô tả: Use case này cho phép người quản lí (Manager) quản lí các thông tin về phòng khách sạn như thêm, sửa, xóa...

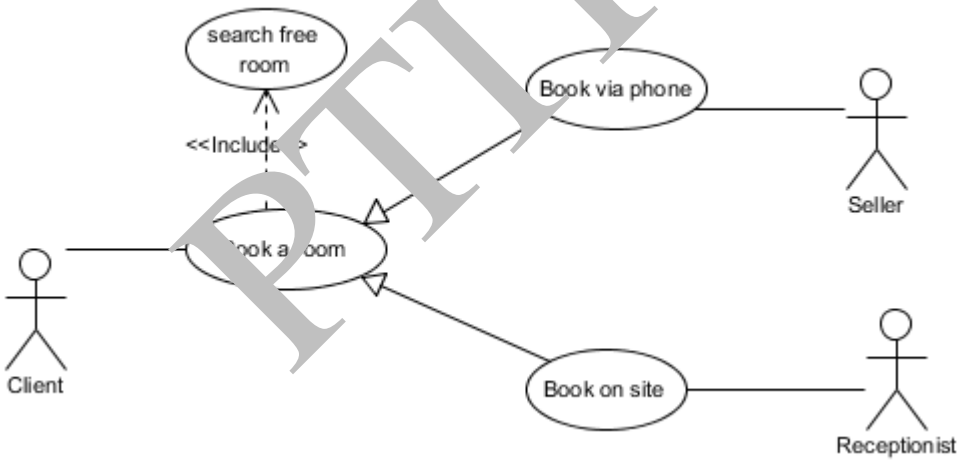
b. Use case Create report



Hình 6.5: Use case tạo báo báo

Mô tả: Use case này cho phép người quản lí (Manager) tạo và xem cáo báo cáo thống kê theo một khoảng thời gian nhất định (tuần, tháng) theo các tiêu chí khác nhau (doanh thu, tỉ lệ phòng trống,...)

c. Use case Book a room

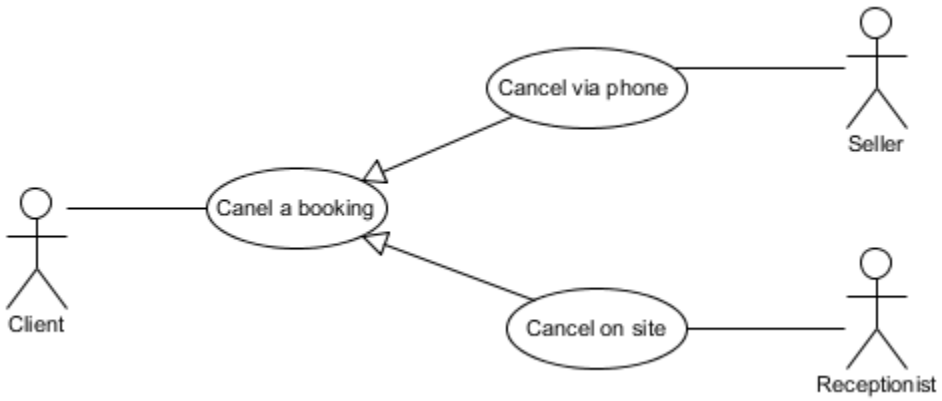


Hình 6.6: Use case đặt phòng

Mô tả: Use case này cho phép Khách hàng có thể đặt phòng. Có hai cách đặt phòng: đặt gián tiếp thông qua điện thoại với Nhân viên bán hàng (Saller, tương ứng với use case Book via phone), hoặc đặt trực tiếp tại quầy thông qua Nhân viên lễ tân (Receptionist, tương ứng với use case Book on site).

Để thực hiện được use case này, phải thực hiện use case Tìm phòng trống.

d. Use case Cancel a Booking



Hình 6.7: Use case hủy đặt phòng

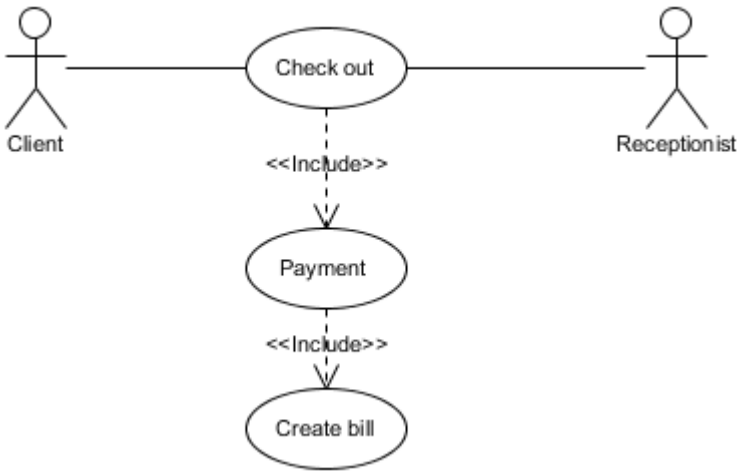
Mô tả: Use case này cho phép Khách hàng có thể hủy đặt phòng. Có hai cách hủy đặt phòng: hủy gián tiếp thông qua điện thoại với Nhân viên bán hàng (Saller, tương ứng với use case Cancel via phone), hoặc hủy trực tiếp tại quầy thông qua Nhân viên lễ tân (Receptionist, tương ứng với use case Cancel on site).

e. Use case Checkin

Hình 6.8: Use case nhận phòng

Mô tả: Use case này cho phép Nhân viên lễ tân (Receptionist) cập nhật trạng thái một khách hàng (Client) thành đã nhận phòng khi khách hàng đến nhận phòng.

f. Use case Checkout

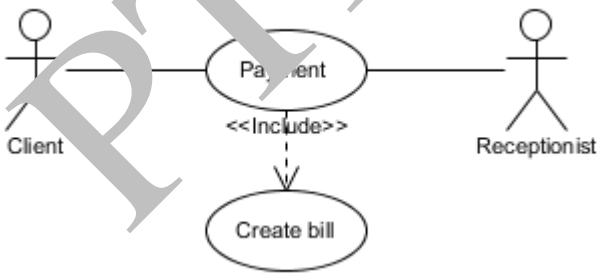


Hình 6.9: Use case trả phòng

Mô tả: Use case này cho phép Nhân viên lễ tân (Receptionist) cập nhật trạng thái một khách hàng (Client) thành đã trả phòng khi khách hàng trả phòng.

Use case này cũng đồng thời thực hiện việc thanh toán và in hóa đơn cho khách hàng.

g. Use case Payment



Hình 6.10: Use case thanh toán tiền phòng

Mô tả: Use case này cho phép Nhân viên lễ tân (Receptionist) thực hiện thanh toán và in hóa đơn cho khách hàng, khi khách hàng có thanh toán trước hoặc khi khách hàng trả phòng.

CHƯƠNG 7: CÁC PHƯƠNG PHÁP PHÂN TÍCH TRUYỀN THỐNG

7.1 YÊU CẦU TÀI LIỆU ĐẶC TẢ

Pha phân tích/đặc tả

- Kết quả của pha đặc tả thể hiện ở tài liệu. Tài liệu đặc tả là hợp đồng giữa khách hàng và người phát triển. Yêu cầu của tài liệu đặc tả:
    - Phía khách hàng: rõ ràng và dễ hiểu nên tài liệu phải có mức độ hình thức vừa đủ để khách hàngcó thể hiểu được.
    - Phía người phát triển: đầy đủ và chi tiết vì nó là nguồn thông tin để sử dụng trng phác thảo thiết kế. Vì vậy, tài liệu đặc tả phải phản ánh đúng yêu cầu khách hàng và là bản hợp đồng giữa khách hàng và nhóm phát triển nên không thẻ thiết sót, mâu thuẫn và nhập nhằng.

Tài liệu đặc tả

Tài liệu đặc tả là hợp đồng giữa khách hàng và người phát triển. Nó mô tả rõ ràng điều gì sản phẩm cần phải làm và ràng buộc đối với sản phẩm. Các ràng buộc:

- Giá thành và thời gian
- Chạy song song: chạy được cùng với phần mềm khách trong môi trường hiện thời
- Tính khả chuyển: sảnphẩm phải chạy được trên phần cứng khác có cùng hệ điều hành hay chạy được cho nhiều hệ điều hành khác nhau.
- Tính tin cậy được: ví dụ, nếu sản phẩm sử dụng để theo dõi nạn nhân thì nó phải chạy trong 24h/ngày.
- Đáp ứng nhanh: 95% câu hỏi phải trả lời trong vòng 0,25 giây. Đối với hệ thời gian thực phải là 100% (thật vô ích nếu 95% trường hợp phần mềm thông báo kịp thời <0,25 giây, cho phi công rằng tên lửa đang đến).

- Các quy tắc chấp nhận:

  - Khách hàng và người phát triển cần đưa ra các trường hợp kiểm thử.

  - Nếu sản phẩm qua được kiểm thử và sản phẩm thực sử thỏa mãn đặc tả và nhóm đã hòan thành công việc.

  - Ví dụ trường hợp kiểm thử: Dựa trên ràng buộc và khách hàng cung cấp dạng dữ liệu cần xử lý.

- Khi nhóm phát triển hiểu đầy đủ vấn đề thì sẽ tiến hành xây dựng chiến lược giải quyết.
- Chiến lược giải quyết là cách tiếp cận chung khi xây dựng sản phẩm
- Ví dụ: Sử dụng cơ sở dữ liệu online/sử dụng các tệp truyền thông
- Các bước xây dựng chiến lược:

1. Tìm các chiến lược: Không quan tâm về ràng buộc trong tài liệu đặc tả

2. Đánh giá và sửa đổi từng chiến lược: so với ràng buocọ của khách hàng (một kỹ thuật là dùng bản mẫu)

3. Xác định một hay vài chiến lược thỏa mãn ràng buộc. Lưu giữ các chiến lược loại bỏ.

4. Chọn chiến lược khả thi: Khách hàgn đưa ra quy tắc lựa chọn và nhóm phát triển đề xuất chiến lược chọn.

7.2 CÁC PHƯƠNG PHÁP ĐẶC TẢ

- Các phương pháp đặc tả được xếp thành 3 loại: Phi hình thức, nửa hình thức, hình thức.
- Đặc tả bằng ngôn ngữ tự nhiên là phi hình thức
- Các phương pháp nửa hình thức:
  - Kỹ thuật đặc tả của Gane và Sarse
  - Mô hình quan hệ thực thể
  - Kỹ thuật của Demarco, Yourdon,…
- Các phương pháp hình thức
  - Máy hữu hạn trạng thái
  - Đặc tả Z (dựa trên logic toán, lý thuyết tập hợp)
  - Mạng Petri…

7.2.1 Đặc tả phi hình thức

- Xét ví dụ trong một tài liệu đặc tả

"If sales for current month are below target sales, then report is to be printed, unless difference between target sales and actual sales is less than half of difference between target sales and actual sales in previous month, or if difference between target sales and actual sales for the current month is under 5%"

Ý nghĩa của đặc tả phi hình thức:

1. Chỉ tiêu doanh thu vào thánh Một là $ 100.000, doanh thu thực sử chỉ là $ 64.000 (kém 36%). Báo cáo được in ra.
2. Chỉ tiêu doanh thu tháng Hai là $120.000, doanh thu thực sự là $100,000 (kém 16,7%). Như vậy, khác biệt % của tháng Hai nhỏ hơn nửa của khác biệt % tháng trước. Không in báo cáo và quản lý tin rằng có cải tiến.
3. Tháng ba chỉ tiêu là $100,000 nhưng doanh thu thực sự là $98,000, chỉ 2% dưới đích (<5%) nên không in báo cáo.

- Vấn đề là gì? Đặc tả chỉ nói sự khác biệt giữa chỉ tiêu doanh thu và doanh thu thực sự mà không nói sự khác biệt %. Phán đoán cuối cùng nói là tỷ lệ %?
- Tài liệu trên không rõ ràng, không thể hiện ý muốn của khách hàng.
- Ngôn ngữ tự nhiên không phải là phương tiện tốt để viết tài liệu đặc tả.
- Tuy nhiên nhiều tổ chức vẫn sử dụng ngôn ngữ tự nhiên đặc biệt đối với các sản phẩm thương mại.
- Lý do:

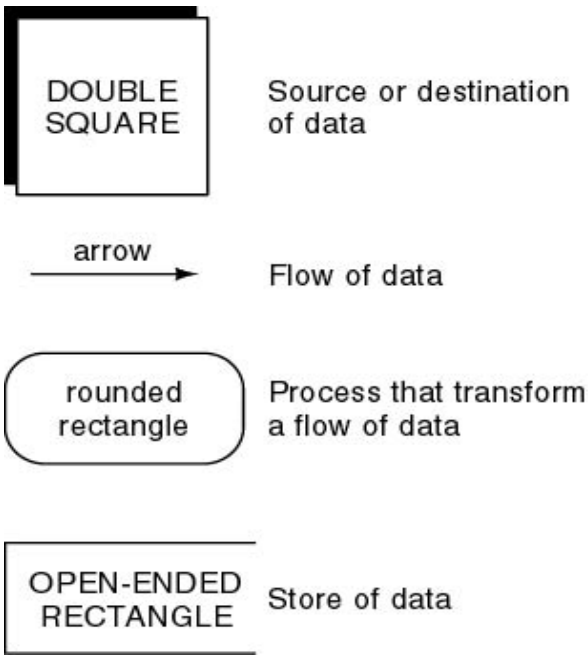
- Các chuyên gia phần mềm chưa được đào tạo cẩn thận.
- Quản lý bị áp lực bởi khách hàng
- Quản lý không sẵn lòng đầu tư vào đào tạo

7.2.2 Phân tích hướng cấu trúc

- Sử dụng biểu đồ để đặc tả phần mềm là một kỹ thuật quan trọng trong những năm 1970. Ba kỹ thuật biểu đồ quen thuộc: DeMarco (1978), Gane và Sarsen (1979) , Yourdon và Constantine (1979).
- Ba kỹ thuật đều tốt và tương đương như nhau. Trước đây nhiều công ty phần mềm ở Mỹ sử dụng chúng trong các sản phẩm thương mại.
- Các tiếp cận của Gane và Sarse hiện thời được sử dụng rộng rãi để thiết kế hướng đối tượng trong công nghiệp.

Ví dụ: Cửa hàng phần mềm Beta mua phần mềm từ các nhà cung cấp khác nhau và bán cho dân chúng. Cửa hàng có phần mềm thông thường nhưng muốn có phần mềm khác thì phải yêu cầu. Cửa hàng bán lẻ ra hàng tháng 300 sản phẩm với giá trung bình $250. Mặc dù thương vụ thành công nhưng nhiều người khuyên Beta nên tin học hóa.

- Câu hỏi: Nên tin học hóa lĩnh vực nào? Thu/chi hay bán trên mạng?
- Hiểm họa tiềm tang của nhiều cách tiếp cận thông thường "trước hết đưa ra giải pháp và sau đó xem xét vấn đề nảy sinh".
- Phương pháp của Gane và Sarse được sử dụng để phân tích yêu cầu của khách hàng theo kỹ thuật 9 bước
- Kỹ thuật 9 bước:
    1. Xây dựng DFD và phân hóa từng bước
    2. Quyết định cái gì cần tin học hóa và như thế nào
    3. Xác định chi tiết dòng dữ liệu
    4. Xác định logic của tiến trình
    5. Xác định các kho dữ liệu //nội dung và định dạng
    6. Xác định các nguồn vật lý
    7. Xác định các đặc tả Input và Output
    8. Xác định kích cỡ
    9. Xác định yêu cầu phần cứng
- Bước 1: Xây dựng Sơ đồ dòng dữ liệu: Sơ đồ dòng dữ liệu DFD chỉ ra logic của dòng dữ liệu tức là điều gì xảy ra. DFD sử dụng 4 ký hiệu cơ bản như hình sau:



DFD biểu diễn bừng hình vẽ mọi khía cạnh logic của dòng dữ liệu. Hình trên là minh hóa lần thứ nhất. Điều chủ yếu là DFD biểu diễn dòng thông tin gói hàng nào khách hàng cần không phải quan trọng.

-

CHƯƠNG 8: PHƯƠNG PHÁP PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG

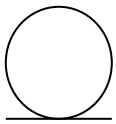
8.1 LUỒNG CÔNG VIỆC PHÂN TÍCH

Mục đích: có 2 mục đích chính

- Để hiểu sâu hơn về các yêu cầu
- Mô tả các yêu cầu theo một cách thức nhất định để tạo điều kiện thuận lợi cho việc thiết kế và cài đặt sau đó có khả năng bảo trì được.

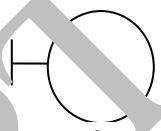
Ba kiểu lớp chính:

- Các lớp thực thể: mô hình thông tin lưu trữ lâu dài, chẳng hạn như: lớp account và lớp investment.
  Ký hiệu UML của lớp thực thể:



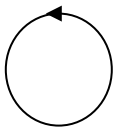
Lớp thực thể

- Các lớp biên: mô hình những tương tác giữa hệ thống phần mềm với môi trường. Lớp biên nhìn chung gắn liền với đầu vào hoặc đầu ra hoặc giao tiếp với các tác nhân. Ví dụ: lớp Investments Report và lớp Mortgages Report.
  Ký hiệu UML của lớp biên:



Lớp biên

- Các lớp điều khiển: mô hình những tính toán và những thuật toán phức tạp. Ví dụ: lớp Estimate Funds for Week.
  Ký hiệu UML của lớp điều khiển:



Lớp điều khiển

8.2 VIỆC TRÍCH RÚT CÁC LỚP THỰC THỂ

Thực hiện theo ba bước sau một cách lặp và tăng dần:

- Việc mô hình hóa chức năng (hay còn gọi là mô hình hóa Use-Case): Xác định các kết quả khác nhau được đưa ra bởi hệ thống phần mềm. Biểu diễn các thông tin đó dưới dạng các kịch bản của tất cả các Use-Case (mỗi kịch bản là một thể hiện của Use Case).
- Mô hình hóa lớp: Xác định các lớp thực thể và các thuộc tính của các lớp. Sau đó, xác định các mối quan hệ qua lại và các tương tác giữa các lớp. Biểu diễn thông tin này bằng biểu đồ lớp.

- Mô hình hóa động: Xác định các hành động được thực hiện bởi hoặc đối với mỗi lớp thực thể hoặc các lớp con. Biểu diễn thông tin này dưới dạng các biểu đồ trạng thái.

Trong thực tế, ba bước trên không được thực hiện một cách tuần tự. Khi nào có một sự thay đổi ở một biểu đồ thì tương ứng sẽ có sự sửa đổi ở hai biểu đồ kia. Do đó, ba bước của phân tích hướng đối tượng được thực hiện song song một cách có hiệu quả.

8.3 PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG CHO BÀI TOÁN THANG MÁY

Một hệ thống được cài đặt để điều khiển n thang máy trong một tòa nhà với m tầng. Việc di chuyển thang máy giữa các tầng phải tuân theo những ràng buộc sau:

- Một là, mỗi thang máy có m nút, mỗi nút tương ứng với một tầng. Các nút này sáng lên khi được bấm và khi đó thang máy sẽ di chuyển tới tầng tương ứng mới số ghi trên nút. Và khi thang máy tới tầng đó thì nút đó sẽ hết sáng và trở lại bình thường.
- Hai là, ngoại trừ tầng đầu tiên và tầng trên cùng, các tầng khác đều có hai nút, một nút để yêu cầu thang máy đi lên và một nút yêu cầu thang máy đi xuống. Những nút này sẽ sáng lên khi được bấm. Và nút đó trở lại bình thường khi thang máy tới tầng tương ứng, sau đó thang máy sẽ di chuyển theo hướng được yêu cầu sau đó.
- Ba là, nếu thang máy không nhận được yêu cầu đi lên hoặc đi xuống, thì nó vẫn ở nguyên tầng đó và đóng cửa.

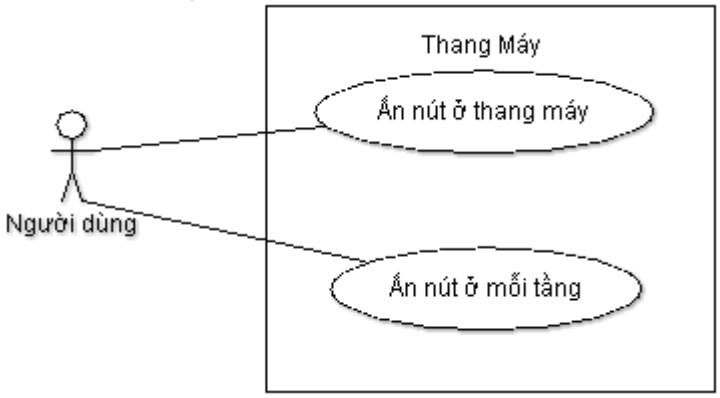
8.4 MÔ HÌNH HÓA CHỨC NĂNG

Một Use-Case miêu tả về những chức năng của hệ thống phần mềm xây dựng. Một Use-Case đưa ra một miêu tả chung về chức năng toàn thể. Còn các kịch bản là những thể hiện cụ thể của các Use-Case. Các kịch bản cần được xem xét cẩn thận để có một cái nhìn toàn diện đối với hệ thống đích được xây dựng.

Use-Case miêu tả tương tác giữa hệ thống và các tác nhân (những người dùng bên ngoài).

Đối với bài toán thang máy, chỉ có hai Use-Case là : Ấn nút ở thang máy và Ấn nút ở mỗi tầng.

Sử dụng UML để biểu diễn Use-Case cho bài toán thang máy như hình 8.1



Hình 8.1 Các use case của bài toán thang máy

Những tương tác có thể giữa người dùng và các lớp đó là: một người dùng nhấn nút tháng máy và ra lệnh cho thang máy di chuyển tới một tầng nào đó hoặc một người dùng nhấn nút để yêu cầu thang máy dừng lại ở một tầng cụ thể. Với mỗi một chức năng nói chung ta có thể đưa ra

một số lượng lớn các kịch bản khác nhau, mỗi một kịch bản biểu diễn một tập các tương tác . Hình 8.2 mô tả kịch bản chuẩn, nó bao gồm một tập các tương tác giữa người dùng và các thang máy tương ứng với cách mà thang máy được sử dụng.

Hình 8.2 được xây dựng sau khi đã quan sát tỉ mỉ những tương tác giữa người dùng với thang máy (chính xác hơn là với các nút của thang máy và các nút của các tầng). 15 sự kiện đã được đánh số miêu tả chi tiết gồm: hai tương tác giữa người dùng A và các nút của hệ thống thang máy (sự kiện 1 và sự kiện 7) và các thao tác của thang máy (sự kiện 2 đến 6 và 8 đến 15). Hai sự kiện người dùng A bước vào thang máy và người dùng A ra khỏi thang máy không được đánh số sự kiện. Những mục như vậy được xem như là những bình luận thêm, người dùng A không tương tác với các thành phần của hệ thống thang máy khi đã bước vào thang máy hoặc rời khỏi thang máy.

1. Người dùng A nhấn nút đi lên của tầng ba để yêu cầu thang máy và người dùng A muốn đi lên tầng 7.
2. Nút đi lên của tầng ba sáng lên.
3. Thang máy đến tầng 3. Trong thang máy đang có người dùng B, người dùng B đã vào thang máy từ tầng 1 và yêu cầu lên tầng 9.
4. Nút đi lên của tầng 3 trở lại trạng thái bình thường.
5. Cửa thang máy mở ra.
6. Máy bấm giờ bắt đầu. Người dùng A bước vào thang máy.
7. Người dùng A nhấn nút 7 của thang máy .
8. Nút 7 của thang máy sáng lên.
9. Cửa thang máy đóng lại sau một thời gian vượt quá thời gian quy định của máy bấm giờ.
10. Thang máy lên tới tầng 7.
11. Nút 7 của thang máy trở lại trạng thái bình thường.
12. Cửa thang máy mở và cho phép người dùng A ra khỏi thang máy.
13. Máy bấm giờ bắt đầu. Người dùng A bước ra khỏi thang máy.
14. Cửa thang máy đóng lại sau một thời gian quy định.
15. Thang máy đi tiếp tục lên tầng 9 theo yêu cầu trước đó của người B.

Hình 8.2 Kịch bản chuẩn cho bài toán thang máy

Trái lại, hình 8.3 là một kịch bản ngoại lệ. Nó miêu tả những gì xảy ra khi người dùng nhất nút đi lên ở tầng 3 nhưng thực sự muốn đi xuống tầng 1. Kịch bản này được xây dựng bởi việc quan sát các hành động của nhiều người trong thang máy.

1. Người dùng A nhấn nút đi lên của tầng ba để yêu cầu thang máy và người dùng A muốn đi xuống tầng 1.
2. Nút đi lên của tầng ba sáng lên.
3. Thang máy đến tầng 3. Trong thang máy đang có người dùng B, người dùng B đã vào thang máy từ tầng 1 và yêu cầu lên tầng 9.
4. Nút đi lên của tầng 3 trở lại trạng thái bình thường.
5. Cửa thang máy mở ra.
6. Máy bấm giờ bắt đầu. Người dùng A bước vào thang máy.
7. Người dùng A nhấn nút 1 của thang máy .
8. Nút 1 của thang máy sáng lên.
9. Cửa thang máy đóng lại sau một thời gian vượt quá thời gian quy định của máy bấm giờ.
10. Thang máy lên tới tầng 9.
11. Nút 9 của thang máy trở lại trạng thái bình thường.
12. Cửa thang máy mở và cho phép người dùng B ra khỏi thang máy.
13. Máy bấm giờ bắt đầu. Người dùng B bước ra khỏi thang máy.
14. Cửa thang máy đóng lại sau một thời gian quy định.
15. Thang máy đi tiếp tục xuống tầng 1 th[illegible] yêu cầu của người dùng A.

Hình 8.3: Kịch bản ngoại lệ của bài toán thang máy

8.5 MÔ HÌNH HÓA LỚP THỰC THỂ

Trong bước này, các lớp và các thuộc tính được trích rút và được biểu diễn bằng biểu đồ UML. Các thuộc tính của mỗi lớp thực thể được xác định ở bước này nhưng các phương thức thì không. Các phươn thức sẽ được gán cho các lớp ở pha thiết kế hướng đối tượng.

Có 3 cách thức để trích rút các lớp và các thuộc tính:

- Một là, suy luận ra các lớp từ các Use-Case và các kịch bản của các Use-Case. Phương pháp này có nhược điểm là thường có nhiều kịch bản do đó có quá nhiều lớp ứng cử để tìm ra lớp thực thể.
- Hai là, sử dụng CRC Cards (nếu chúng ta có tri thức về miền nghiệp vụ).
- Ba là, sử dụng phương pháp trích rút danh từ.

8.5.1 Trích rút danh từ

Đối với những người phát triển mà chưa có kiến thức chuyên môn thành thạo về nghiệp vụ của hệ thống xây dựng thì cách tốt nhất để trích rút ra các lớp ứng cử là sử dụng phương pháp trích rút danh từ với 2 giai đoạn như sau:

- Giai đoạn 1: Định nghĩa vấn đề một cách ngắn gọn: Miêu tả hệ thống phần mềm xây dựng một cách ngắn gọn, súc tích dưới dạng một đoạn văn duy nhất. Trong bài toán thang máy ta có đoạn văn như sau:

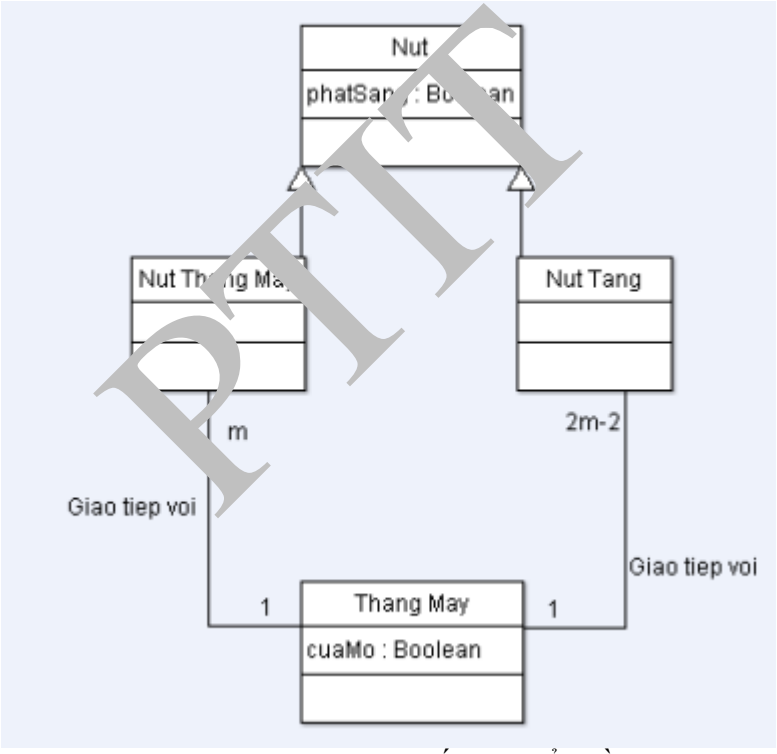
  " Các nút trong thang máy và ở mỗi tầng điều khiển sự di chuyển của n thang máy trong tòa nhà m tầng. Các nút sáng lên khi có người bấm nút đó để yêu cầu thang máy di chuyển tới một tầng nào đó; nút đó sẽ trở lại trạng thái bình thường khi yêu cầu đã được

đáp ứng. Khi thang máy không nhận được yêu cầu nào thì nó vẫn ở tầng hiện tại và cửa vẫn đóng."

- Giai đoạn 2: Xác định các danh từ: Xác định các danh từ theo chiến lược không hình thức. Trong bài toán thang máy có:
  "Các nút trong thang máy và ở mỗi tầng điều khiển sự di chuyển của n thang máy trong tòa nhà m tầng . Các nút sáng lên khi có người bấm nút đó để yêu cầu thang máy di chuyển tới một tầng nào đó; nút đó sẽ trở lại trạng thái bình thường khi yêu cầu đã được đáp ứng. Khi thang máy không nhận được yêu cầu nào thì nó vẫn ở tầng hiện tại và cửa vẫn đóng".

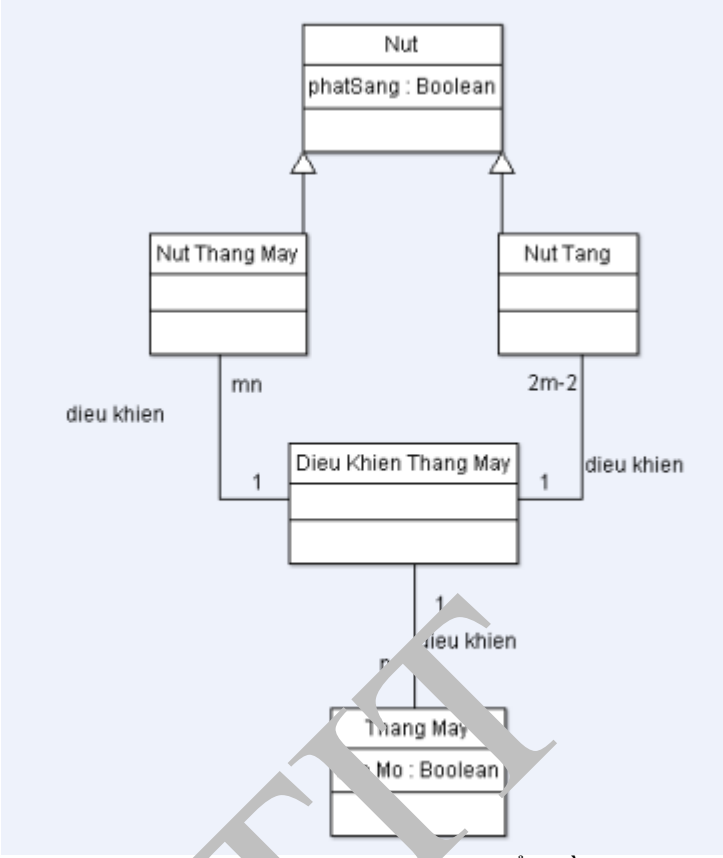
Sử dụng các danh từ trên như là các lớp ứng cử. Ta có các danh từ: Nút, thang máy, tầng , sự di chuyển, tòa nhà, yêu cầu, cửa. Trong đó, các danh từ: tầng, tòa nhà, cửa nằm bên ngoài biên của bài toán nên bị loại trừ. Còn danh từ sự di chuyển là danh từ trừu tượng nên bị loại trừ (chúng có thể trở thành các thuộc tính). Do đó, các lớp ứng cử là: thang máy và lớp nút và các lớp con: lớp nút thang máy và lớp nút tầng.

Biểu đồ lớp kết quả được biểu diễn bằng UML như hình 8.4



Hình 8.4 Bước lặp thứ nhất của biểu đồ lớp

Vấn đề xảy ra ở đây là trong thực tế các nút của thang máy không giao tiếp trực tiếp với thang máy. Khi đó thường yêu cầu có một vài loại điều khiển thang máy, và quyết định thang máy nào sẽ đáp ứng yêu cầu cụ thể. Tuy nhiên, trong phát biểu bài toán không đề cập đến lớp điều khiển, vì thế không có lớp điều khiển trong suốt quá trình trích rút danh từ. Mặt khác, kỹ thuật trích danh từ mà chúng ta sử dụng ở đây để tìm ra các lớp ứng cử được xem như điểm khởi đầu nhưng cũng không nên dựa hoàn toàn vào đó. Chúng ta cần thêm lớp điều khiển thang máy vào biểu đồ lớp hình 8.4. Đồng thời cũng thêm mối quan hệ 1-n giữa lớp điều khiển thang máy và lớp thang máy. Yếu tố này sẽ tạo điều kiện cho việc thiết kế và cài đặt dễ dàng hơn



Hình 8.5 Bước lặp thứ hai của biểu đồ lớp

8.5.2 CRC Cards

Trong nhiều năm về trước, CRC (Class-responsibility-collaboration) Card được sử dụng trong suốt pha phân tích [Wirfs-Brock, Wilkerson và Wiener, 1990]. Đối với mỗi lớp, đội phát triển phầm mềm điền vào thẻ biểu diễn tên của lớp (name of **C**lass), các chức năng của lớp (**R**esponsibility) và danh sách các lớp khác có liên quan đến lớp đó để cùng nhau thực hiện các chức năng của lớp đó (**C**ollaboration). Hiện nay, CRC card được thực hiện một cách tự động bằng cách sử dụng thành phần công cụ CASE.

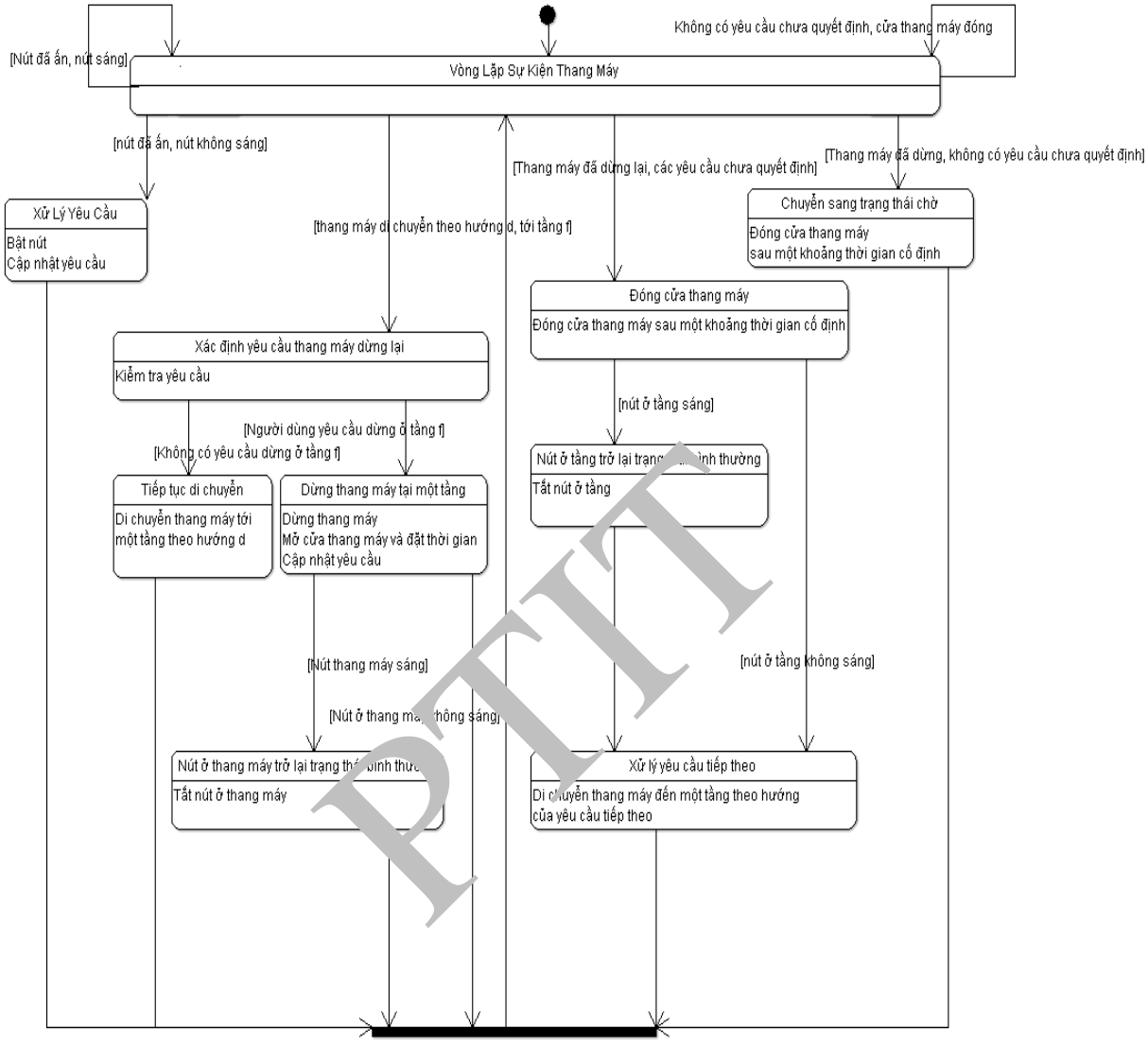
Điểm mạnh của CRC card là khi CRC card được thực hiện bởi các thành viên trong đội thì tương tác qua lại giữa các thành viên trong đội dễ dàng phát hiện ra những thiếu sót hoặc những mục không đúng trong CRC card.

Điểm yếu của CRC card là nếu chúng ta sử dụng CRC card để nhận dạng các lớp thực thể, thì yêu cầu có sự thành thạo về chuyên môn nghiệp vụ.

8.6 MÔ HÌNH HÓA ĐỘNG

Mục đích của việc mô hình hóa động là đưa ra biểu đồ tuần tự. Biểu đồ tuần tự mô tả hệ thống phần mềm cuối cùng gần giống với máy hữu hạn trạng thái đối với mỗi lớp. Hình 8.6 biểu diễn biểu đồ trạng thái của lớp Elevator Contronller

Khi biểu diễn biểu đồ trạng thái bằng UML gồm các 3 phần chính trạng thái, sự kiện và vị từ được phân bố trên toàn biểu đồ trang thái. Ví dụ trạng thái **Chuyển sang trạng thái chờ** trong hình 8.6 được bắt đầu nếu trạng thái hiện tại là **Vòng lặp sự kiện thang máy** và vị từ **[*Thang máy đã dừng, không có yêu cầu nào chưa quyết định*]** là đúng. Khi trạng thái **Chuyển sang trạng thái chờ** đã bắt đầu thì hành động ***Đóng cửa thang máy sau một khoảng thời gian cố định*** được thực hiện.



Hình 8.6 Biểu diễn biểu đồ trạng thái của lớp Elevator Controller

Để thấy sự tương đương giữa biểu đồ trạng thái trong hình 8.6 và STD của hình 11.14 và 11.16, phải xem xét trong các loại kịch bản khác nhau. Chẳng hạn, xem xét phần đầu tiên của kịch bản của hình 8.2. Đầu tiên, người dùng A nhấn nút đi lên ở tầng 3. Nếu nút ở tầng ba không sáng lên thì sau đó phải xử lý như hình 11.15 và trạng thái **Xử lý yêu cầu** của hình 8.6 xử lý tình huốn nút được bật sáng lên. Trong trường hợp của biểu đồ trạng thái thì trạng thái tiếp theo sẽ là **Vòng lặp sự kiện thang máy.** Sau đó, thang máy tiến gần lại tầng 3. Trước tiên ta xem xét cách tiếp cận STD, trong hình 11.16, thang máy bắt đầu với trạng thái S(U, 3) có nghĩa là đi lên và dừng ở tầng 3 (giả định đơn giản hóa ở đây là chỉ có duy nhất một thang máy nên đối số e trong hình 11.16 được bỏ ở đây.) Từ hình 11.15, khi mà thang máy đến tầng 3 thì nút đi lên ở tầng 3 bị

tắt. (Theo hình 11.16) hiện thời cửa thang máy đóng và thang máy bắt đầu di chuyển lên tầng 4. Quay trở lại hình 8.6 chúng ta xem xét chuyện gì sẽ xảy ra khi mà thang máy lên tới gần tầng 3. Bởi vì thang máy di chuyển nên trạng thái tiếp theo sẽ là **Xác định yêu cầu thang máy dừng lại**. Các yêu cầu được kiểm tra và bởi vì, người dùng A đã yêu cầu thang máy dừng ở đó, nên trạng thái tiếp theo là **Dừng thang máy tại một tầng.** Thang máy dừng ở tầng 3 và cửa thang máy mở. Nút thang máy ở tầng 3 không được nhấn nên trạng thái tiếp theo là **Vòng lặp sự kiện thang máy.** Người dùng A bước vào và nhấn nút thang máy lên tầng 7. Do đó, trạng thái tiếp theo lại là **Xử lý yêu cầu**, tiếp đó là trạng thái **Vòng lặp sự kiện thang máy.** Thang máy dừng lại và hai yêu cầu đang bị treo, vì thế trạng thái tiếp theo là **Đóng cửa thang máy** cửa đóng lại sau khoảng thời gian cố định. Nút ở tầng 3 được nhấn ở tầng 3 bởi người dùng A, vì thế trạng thái tiếp theo là **Nút ở tầng trở lại trạng thái bình thường** và nút đi lên ở tầng 3 trở về trạng thái bình thường. Trạng thái tiếp theo là **Xử lý yêu cầu tiếp theo** và thang máy bắt đầu di chuyển lên tầng 4. Rõ ràng những thành phần liên quan của biểu đồ tương ứng là tương đương với kịch bản này.

Trong thực tế, biểu đồ trạng thái được xây dựng từ việc mô hình hóa các sự kiện của kịch bản cụ thể. Chẳng hạn, xem xét sự kiện đầu tiên của kịch bản trong hình 8.2, người dùng A nhấn nút đi lên ở tầng ba. Sự kiện này được tổng quan hóa thành là một nút bất kỳ được nhấn. Có hai khả năng có thể xảy ra hoặc là nút sẵn sàng bật sáng (trong trường hợp không có gì xảy ra) hoặc là nút không được bật sáng (trong trường hợp hành động phải nắm bắt được để xử lý yêu cầu của người dùng). Để mô hình sự kiện này, trạng thái **Vòng lặp sự kiện thang máy** được đưa ra trong hình 8.6. Trường hợp nút sẵn sàng phát sáng được mô hình bởi vòng lặp không làm gì với vị từ ***[nút đã ấn, nút sáng]*** ở góc trái phía trên của hình 8.6. Trong trường hợp khác, nút không phát sáng, được mô hình bởi mũi tên được gán nhãn là vị từ ***[nút đã ấn, nút không sáng]*** dẫn tới trạng thái **Xử lý yêu cầu.** Từ sự kiện 2 của kịch bản rõ ràng hành động ***Bật nút*** là cần thiết trong mỗi trạng thái **Xử lý yêu cầu**. Hơn nữa, mục đích của hành động của người dùng là việc nhấn một nút bất kỳ để yêu cầu thang máy, vì thế hành động ***Cập nhật yêu cầu*** cũng phải được thực thi ở trạng thái **Xử lý yêu cầu.**

Bây giờ chúng ta sẽ xem xét sự kiện 3 của kịch bản, thang máy tới tầng 3. Sự kiện này được mô hình hóa là thang máy bất kỳ đang di chuyển giữa các tầng. Sự di của thang máy được mô hình bằng vị từ [***Thang máy đi chuyển theo hướng d, tới tầng f***] và trạng thái tiếp theo là **Xác định yêu cầu thang máy dừng lại.** Và ở đây lại có hai khả năng xảy ra hoặc là có một yêu cầu dừng tại tầng f hoặc là không có yêu cầu nào như vậy. Trong trường hợp yêu cầu dừng tại từng f tương ứng với vị từ [***người dùng đã yêu cầu dừng thang máy ở tầng f***] và trạng thái tiếp theo là **Dừng thang máy tại một tầng** và có các hành động tương ứng với kịch bản ở hình 8.2 là ***Dừng thang máy (***từ sự kiện 3), ***Mở cửa và bắt đầu đếm giờ*** (từ sự kiện 5, 6) và ***Cập nhật yêu cầu*** . Cuối cùng, mô hình hóa sự kiện 4 trong kịch bản đó là nút thang may sẽ trở lại trạng thái bình thường nếu nó sáng lên và được mô hình hóa bằng trạng thái **Nút thang máy trở lại trạng thái bình thường (**là trạng thái cuối cùng bên trái của hình 8.6**),** cùng với 2 vị từ ở phía trên của hộp trạng thái này.

Mô hình hóa sự kiện 9 của kịch bản trong hình 8.2 sinh ra trạng thái **Đóng cửa thang máy;** mô hình hóa sự kiện 10 tương ứng là trạng thái **Xử lý yêu cầu tiếp theo.** Tuy nhiên, rất cần thiết có trạng thái **Chuyển sang trạng thái chờ** và vị từ [***Không có yêu cầu chưa quyết định, cửa thang máy đóng***] được suy luận ra từ việc mô hình hóa một sự kiện của kịch bản khác, đó là kịch bản người dùng ra khỏi thang máy và không còn nút nào sáng.

8.7 KIỂM THỬ TRONG PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG

Sau khi 3 mô hình của tiến trình phân tích hướng đối tượng được xem là hoàn thành thì bước tiếp theo là kiểm tra lại pha phân tích hướng đối tượng. CRC Cards là kỹ thuật kiểm thử tốt.

Các CRC cards đối với mỗi lớp **Nút, Nút Thang Máy , Nút Tầng, Thang Máy** và **Điều Khiển Thang Máy** được điền đầy đủ các thông tin. Ví dụ hình 8.7 được suy luận từ biểu đồ lớp 8.5 và biểu đồ trạng thái của hình 8.6. **Trách nhiệm** của lớp **Điều Khiển Thang Máy** chứa một loạt các hành động trong biểu đồ trạng thái của **Điều Khiển Thang Máy (**hình 8.6**). Các lớp cộng tác** của lớp **Điều Khiển Thang Máy** được xác định bằng cách xem xét biểu đồ lớp hình 8.5 và chú ý rằng các lớp **Nút Thang Máy**, **Nút Tầng** và **Thang Máy** tương tác với lớp **Điều Khiển Thang Máy.**

Lớp Điều Khiển Thang Máy
Trách nhiệm 1. Bật nút thang máy 2. Tắt nút thang máy 3. Bật nút ở tầng 4. Tắt nút ở tầng 5. Di chuyển thang máy lên một tầng nào đó 6. Di chuyển thang máy xuống một tầng nào đó 7. Mở cửa thang máy và bắt đầu đếm giờ 8. Đóng cửa thang máy sau một khoảng thời gian nhất định 9. Kiểm tra yêu cầu 10. Cập nhật yêu cầu
Cộng tác 1. **Lớp Nút Thang Máy** 2. **Lớp Nút Tầng** 3. **Lớp Thang Máy**

Hình 8.7 Vòng lặp thứ nhất cho CRC Card đối với lớp Elevator Controller

CRC Card này nổi bật với hai vấn đề chính trong vòng lặp đầu tiên của phân tích hướng đối tượng. Trước tiên hãy xem xét trách nhiệm 1.*Bật nút thang máy.* Yêu cầu này nói chung nằm ngoài phạm vi của mô hình hướng đối tượng. Từ quan điểm của thiết kế hướng trách nhiệm (1.6), chính các đối tượng của lớp **Nút Thang Máy** chịu trách nhiệm bật và tắt. Từ quan điểm ẩn dấu thông tin (7.6) **Điều Khiển Thang Máy** cần biết về khả năng của **Nút Thang Máy** là có thể bật một nút. Khi đó trách nhiệm đó phải được sửa lại là : gửi một thông điệp tới nút **Nút Thang Máy** để bật một nút. Tương tự ta cần phải thay đổi đối với các trách nhiệm 2 đến 6 trong hình 8.7. Sự sửa đổi này được phản ánh trong hình 8.8, vòng lặp thứ 2 của CRC Card đối với lớp **Điều Khiển Thang Máy.**

Vấn đề thứ hai là một lớp đã bị bỏ qua. Xem xét trách nhiệm 7.*Mở cửa thang máy và bắt đầu đếm giờ.* Khái niệm chính ở đây chính là trạng thái. Các thuộc tính của một lớp đôi khi được gọi là biến trạng thái. Trong hầu hết pha cài đặt hướng đối tượng, trạng thái của hệ thống phần mềm được định nghĩa bởi giá trị của các thuộc tính của các loại đối tượng thành phần khác nhau. Biểu đồ trạng thái có nhiều đặc trưng chung so với máy hữu hạn trạng thái. Vì thế khái niệm trạng thái đóng vai trò quan trọng trong mô hình hướng đối tượng. Khái niệm này được sử dụng để xác định liệu một thành phần có nên được mô hình như một lớp không? Nếu thành phần có trạng thái sẽ bị thay đổi trong suốt quá trình thực thi của sự cài đặt thì nó có thể được mô hình như một lớp. Rõ ràng, cửa của thanh máy có trạng thái (đóng và mở) và do đó nên mô hình **Cửa Thang Máy** là một lớp.

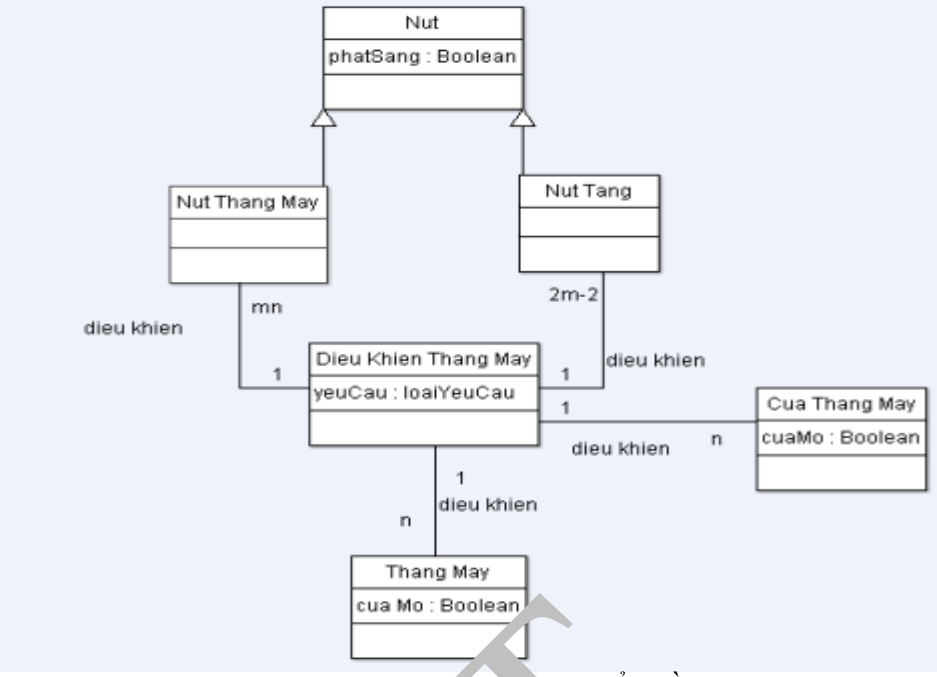
Có một lý do khác lý giải tại sao **Cửa Thang Máy** nên là một lớp. Mô hình hướng đối tượng cho phép trạng thái được ẩn dấu bên trong đối tượng và do đó trạng thái được bảo vệ khỏi những sự thay đổi không cho phép. Nếu có một đối tượng **Cửa Thang Máy**, thì có duy nhất một cách để đóng hoặc mở cửa của thanh máy bằng cách gửi một thông điệp tới đối tượng **Cửa Thang Máy.**

Việc đưa thêm lớp **Cửa Thang Máy** có nghĩa là trách nhiệm 7 và 8 trong hình 8.7 được thay đổi tương tự như trách nhiệm 1 cho đến 6. Có một thông điệp gửi tới lớp **Cửa Thang Máy** để tự đối tượng đó đóng và mở. Nhưng có thêm sự phức tạp, trách nhiệm 7 là *Mở cửa thang máy và bắt đầu đếm giờ* phải được phân tách thành 2 trách nhiệm riêng biệt. Một trách nhiệm tương ứng là gửi thông điệp tới **Cửa Thang Máy** để mở cửa thang máy. Vì máy bấm giờ là một phần của **Điều Khiển Thang Máy** đo đó việc bắt đầu tính giờ là trách nhiệm của **Điều Khiển Thang Máy**. Vòng lặp thứ hai của CRC Card đối với lớp **Điều Khiển Thang Máy** được biểu diễn như hình 8.8. Trong hình 8.8 cũng thêm vào hai trách nhiệm là *Kiểm tra yêu cầu* và *Cập nhật yêu cầu* đối với lớp **Điều Khiển Thang Máy.**

Lớp Điều Khiển Thang Máy
Trách nhiệm 1. Gửi thông điệp tới **Lớp Nút Thang Máy** để bật nút 2. Gửi thông điệp tới **Lớp Nút Thang Máy** để tắt nút 3. Gửi thông điệp tới **Lớp Nút Tầng** để bật nút 4. Gửi thông điệp tới **Lớp Nút Tầng** để tắt nút 5. Gửi thông điệp tới **Lớp Thang Máy** để di chuyển lên một tầng nào đó. 6. Gửi thông điệp tới **Lớp Thang Máy** để di chuyển xuống một tầng nào đó. 7. Gửi thông điệp tới **Lớp Cửa Thang Máy** để mở cửa 8. Bắt đầu đếm giờ 9. Gửi thông điệp tới **Lớp Cửa Thang Máy** để đóng cửa sau một khoảng thời gian nhất định. 10. Kiểm tra yêu cầu 11. Cập nhật yêu cầu
Cộng tác 1. **Lớp Nút Thang Máy** (lớp con) 2. **Lớp Nút Tầng** (lớp con) 3. **Lớp Cửa Thang Máy** 4. **Lớp Thang Máy**

Hình 8.8 Vòng lặp thứ 2 của CRC Card của lớp Elevator Controller

Biểu đồ lớp được sửa lại như hình 8.9. Sau khi chỉnh sửa biểu đồ lớp, thì biểu đồ Use-Case và biểu đồ trạng thái cũng phải xem xét lại, nếu cần thiết sẽ làm mịn hơn nữa. Biểu đồ Use case vẫn đầy đủ. Tuy nhiên, các hành động trong biểu đồ trạng thái hình 8.6 phải được chỉnh sửa để phản ánh đầy đủ những trách nhiệm trong 8.8 (vòng lặp thứ hai của CRC Card). Tập các biểu đồ trạng thái phải được mở rộng vì có thêm một lớp mới. Kịch bản cần được cập nhật để thể hiện sử thay đổi; Hình 8.10 biểu diễn vòng lặp thứ hai của kịch bản hình 8.2. Mặc dù sau tất cả những thay đổi đã được đưa ra và kiểm tra nhưng trong suotó pha thiết kế hướng đối tượng vẫn phải quay trở lại phân tích hướng đối tượng và xem xét lại một hoặc nhiều biểu đồ.



Hình 8.9 Vòng lặp mứ[illegible] của biểu đồ lớp

1. Người dùng A nhấn nút đi lên của tầng [illegible] để yêu cầu thang máy và người dùng A muốn đi lên tầng 7.
2. Nút ở tầng 3 thông báo cho bộ điều khiển tha[illegible] máy là nút ở tầng đã được nhấn vào.
3. Điều khiển thang máy gửi một thông điệp tới nút đi lên ở tầng 3 để nút đó tự bật sáng.
4. Điều khiển thang máy gửi một loạt các thông điệp tới thang máy để thang máy tự di chuyển lên tầng 3. Trong [illegible]ang máy [illegible]ện có người dùng B đã vào thang máy từ tầng một và yêu cầu lê[illegible]ng 9.
5. Điều khiển thang máy [illegible] một thông điệp tới cửa thang máy để mở cửa.
6. Thang máy bắt đầu tính [illegible]ời gian. Người dùng A bước vào thang máy.
7. Người dùng A nhấn nút thang máy lên tầng 7.
8. Nút thang máy thông báo tới điều khiển thang máy rằng là nút thang máy đã được nhấn vào.
9. Bộ điều khiển thang máy gửi một thông điệp tới nút số 7 của thang máy để nút đó được bật sáng.
10. Bộ điều khiển thang máy gửi một thông điệp tới cửa thang máy để đóng thang máy sau một khoảng thời gian cố định.
11. Bộ điều khiển gửi một thông điệp tới nút đi lên ở tầng 3 để nó trở về trạng thái bình thường.
12. Bộ điều khiển thang máy gửi một loạt thông điệp tới thang máy để nó di chuyển lên tới tầng 7.
13. Bộ điều khiển thang máy gửi một thông điệp tới nút 7 ở thang máy để nó trở về trạng thái bình thường (không sáng).
14. Bộ điều khiển thang máy gửi một thông điệp tới cửa thang máy yêu cầu mở cửa để cho phép người dùng A bước ra khỏi thang máy.
15. Bộ điều khiển thang máy bắt đầu đặt thời gian. Người dùng A bước ra khỏi thang máy.
16. Bộ điều khiển thang máy gửi một thông điệp tới cửa thang máy để đóng cửa sau một thời gian cố định.
17. Bộ điều khiển thang máy gửi một loạt thông điệp tới thang máy để nó di chuyển lên tầng 9.

8.8 CÁC CÔNG CỤ CASE CHO PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG

Các biểu đồ đóng vai trò quan trọng trong phân tích hướng đối tượng. Các biểu đồ thường xuyên thay đổi do đó cần có các công cụ vẽbiểu đồ.

Các công cụ hỗ trợ cho UML:

- Các công cụ mang tính thương mại như: IBM Rational Rose và Together.
- Các công cụ Open-source: AgroUML

8.9 CASE STUDY CHO PHA PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG

8.9.1. Các scenario

Mục đích của bước này là viết các kịch bản (scenario) cho các use case đã xác định được trong pha lấy yêu cầu.

a. Scenario cho Use case Room manage

Mô tả: Use case này cho phép người quản lí (Manager) quản lí các thông tin về phòng khách sạn như thêm, sửa, xóa...

Thêm phòng mới:

1. Người quản lí A click vào chức năng thêm phòng trong menu quản lí chính. Quản lí A muốn thêm thông tin một phòng mới của khách sạn.
2. Hệ thống hiển thị giao diện thêm phòng mới, yêu cầu nhập các thông tin: mã (tên) phòng, kiểu phòng, giá hiển thị, và mô tả phòng
3. Quản lí A nhập đầy đủ các thông tin về phòng mới và clik vào nút Thêm phòng
4. Hệ thống lưu thông tin phòng mới vào CSDL và thông báo thêm thành công

Sửa thông tin một phòng đã tồn tại:

1. Người quản lí A click vào chức năng sửa phòng trong menu quản lí chính. Quản lí A muốn sửa thông tin một phòng 305 của khách sạn.
2. Hệ thống hiển thị giao diện tìm phòng để sửa, yêu cầu nhập các thông tin: mã (tên) phòng
3. Quản lí A nhập tên phòng 305 và clik vào nút Tìm phòng
4. Hệ thống tìm kiếm thông tin phòng 305 từ CSDL và hiển thị lên các ô có thể sửa được các thuộc tính của phòng 305:: mã (tên) phòng, kiểu phòng, giá hiển thị, và mô tả phòng
5. Quản lí A sửa lại các thông tin về phòng 305 và clik vào nút Thêm phòng
6. Hệ thống lưu thông tin phòng mới vào CSDL và thông báo thêm thành công

b. Scenario cho Use case Create report

Mô tả: Use case này cho phép người quản lí (Manager) tạo và xem cáo báo cáo thống kê theo một khoảng thời gian nhất định (tuần, tháng) theo các tiêu chí khác nhau (doanh thu, tỉ lệ phòng trống,...)

Xem báo cáo doanh thu theo thời gian:

1. Người quản lí A click vào chức năng xem báo cáo trong menu quản lí chính. Quản lí A muốn báo cáo doanh thu theo thời gian của khách sạn.
2. Hệ thống hiển thị giao diện yêu cầu nhập khoảng thời gian: ngày bắt đầu, ngày kết thúc thống kê. Nút chọn kiểu báo cáo theo doanh thu hay theo tỉ lệ phòng kín chỗ
3. Quản lí A nhập ngày bắt đầu là 01/01/2013, ngày kết thúc là 31/12/2013, chọn kiểu báo cáo theo doanh thu và clik vào nút Thống kê
4. Hệ thống tìm kiếm từ CSDL và hiển thị lên thông tin thống kê doanh thu cho từng phòng, theo thứ tự giảm dần của tổng doanh thu theo phòng trong khoảng thời gian 01/01/2013-31/12/2013. Hàng dưới cùng ghi tổng doanh thu của tất cả các phòng
5. Quản lí A xem thông tin thống kê và clik vào nút Quay lại menu chính

c. Scenario cho Use case Book a room

Mô tả: Use case này cho phép Khách hàng có thể đặt phòng. Có hai cách đặt phòng: đặt gián tiếp thông qua điện thoại với Nhân viên bán hàng (Saller, tương ứng với use case Book via phone), hoặc đặt trực tiếp tại quầy thông qua Nhân viên lễ tân (Receptionist, tương ứng với use case Book on site).

Để thực hiện được use case này, phải thực hiện use case Tìm phòng trống.

Đặt phòng qua điện thoại:

1. Nhân viên bán hàng A click vào chức năng đặt phòng trong menu quản lí đặt phòng. Nhân viên bán hàng A muốn đặt phòng cho khách hàng B, người đang gọi điện thoại yêu cầu đặt phòng.
2. Hệ thống hiển thị giao diện yêu cầu nhập khoảng thời gian muốn đặt của khách hàng: ngày bắt đầu, ngày kết thúc.
3. Nhân viên A hỏi lại khách hàng B ngày bắt đầu và kết thúc theo mong muốn
4. Khách hàng B trả lời ngày mong muốn cho nhân viên A qua điện thoại
5. Nhân viên A nhập ngày bắt đầu, ngày kết thúc vào các ô tìm kiếm và click vào nút tìm phòng trống.
6. Hệ thống tìm kiếm từ CSDL và hiển thị lên thông tin các phòng trống trong khoảng thời gian đã nhập. Mỗi hàng một phòng với đầy đủ thông tin: tên (mã) phòng, kiểu phòng, giá hiển thị, mô tả.
7. Nhân viên A thông báo các phòng có thể đặt cho khách hàng B và yêu cầu khách hàng B chọn 1 phòng trong số đó.
8. Khách hàng B chọn 1 phòng và thông báo lại cho nhân viên A, qua điện thoại
9. Nhân viên A click vào phòng tương ứng trên giao diện, và chọn đặt phòng
10. Hệ thống hiện lên giao diện yêu cầu nhập thông tin khách hàng: tên, ngày sinh, số CMT (passport), kiểu giấy tờ tùy thân, địa chỉ.
11. Nhân viên A hỏi lại khách hàng B các thông tin trên và điền lần lượt vào các ô. Sau đó click vào submit.
12. Hệ thống lưu thông tin đặt phòng vào CSDL và thông báo đặt phòng thành công với các thông tin: tên khách hàng, số id của khách hàng, địa chỉ khách hàng, tên phòng, ngày đến, ngày đi, giá phòng theo đêm, tổng giá dự kiến.
13. Nhân viên A xác nhận và thông báo lại cho khách hàng B

d. Scenario cho Use case Cancel a Booking

Mô tả: Use case này cho phép Khách hàng có thể hủy đặt phòng. Có hai cách hủy đặt phòng: hủy gián tiếp thông qua điện thoại với Nhân viên bán hàng (Saller, tương ứng với use case Cancel via phone), hoặc hủy trực tiếp tại quầy thông qua Nhân viên lễ tân (Receptionist, tương ứng với use case Cancel on site).

Hủy đặt phòng qua điện thoại:

1. Nhân viên bán hàng A click vào chức năng hủy đặt phòng trong menu quản lí đặt phòng. Nhân viên bán hàng A muốn hủy đặt phòng cho khách hàng B, người đang gọi điện thoại yêu cầu hủy đặt phòng.
2. Hệ thống hiển thị giao diện yêu cầu tên hoặc số id của khách hàng: 1 ô nhập id, 1 ô nhập tên và một nút tìm kiếm lịch sử đặt phòng của khác hàng
3. Nhân viên A hỏi lại khách hàng B id và tên
4. Khách hàng B trả lời tên và số id của mình cho nhân viên A qua điện thoại
5. Nhân viên A nhập id của khách hàng B ô tìm kiếm và click vào nút tìm kiếm.
6. Hệ thống tìm kiếm từ CSDL và hiển thị lên thông tin các lần đặt phòng của khách hàng B. Mỗi hàng một lần đặt với đầy đủ thông tin: tên (mã) phòng, kiểu phòng, giá đặt, mô tả, ngày bắt đầu, ngày kết thúc.
7. Nhân viên A yêu cầu khách hàng B và xác nhận thông tin phòng đặt và ngày đặt.
8. Khách hàng B thông báo lại cho nhân viên A qua điện thoại
9. Nhân viên A click vào lần đặt phòng tương ứng trên giao diện, và chọn hủy đặt phòng
10. Hệ thống cập nhật thông tin hủy đặt phòng vào CSDL và thông báo hủy đặt phòng thành công.
11. Nhân viên A xác nhận và thông báo lại cho khách hàng B

e. Scenario cho Use case Checkin

Mô tả: Use case này cho phép Nhân viên lễ tân (Receptionist) cập nhật trạng thái một khách hàng (Client) thành đã nhận phòng khi khách hàng đến nhận phòng.

Khách hàng checkin tại quầy:

1. Nhân viên lễ tân A click vào chức năng nhận phòng trong menu quản lí khách nhận phòng. Nhân viên lễ tân A muốn cập nhật trạng thái nhận phòng cho khách hàng B.
2. Hệ thống hiển thị giao diện yêu cầu tên hoặc số id của khách hàng: 1 ô nhập id, 1 ô nhập tên và một nút tìm kiếm lịch sử đặt phòng của khác hàng
3. Nhân viên A hỏi lại khách hàng B id và tên
4. Khách hàng B trả lời tên và số id của mình cho nhân viên A
5. Nhân viên A nhập id của khách hàng B ô tìm kiếm và click vào nút tìm kiếm.
6. Hệ thống tìm kiếm từ CSDL và hiển thị lên thông tin các lần đặt phòng của khách hàng B. Mỗi hàng một lần đặt với đầy đủ thông tin: tên (mã) phòng, kiểu phòng, giá đặt, mô tả, ngày bắt đầu, ngày kết thúc.
7. Nhân viên A yêu cầu khách hàng B và xác nhận thông tin phòng đặt: ngày đến và đi.
8. Khách hàng B xác nhận ngày đến ngày đi cho nhân viên A
9. Nhân viên A click vào lần đặt phòng tương ứng trên giao diện, và chọn nhận phòng
10. Hệ thống cập nhật thông tin nhận phòng vào CSDL và thông báo nhận phòng thành công.
11. Nhân viên A xác nhận và trao chìa khóa phòng cho khách hàng B

f. Scenario cho Use case Checkout

Mô tả: Use case này cho phép Nhân viên lễ tân (Receptionist) cập nhật trạng thái một khách hàng (Client) thành đã trả phòng khi khách hàng trả phòng.

Use case này cũng đồng thời thực hiện việc thanh toán và in hóa đơn cho khách hàng.

Khách hàng trả phòng tại quầy:

1. Nhân viên lễ tân A click vào chức năng trả phòng trong menu quản lí khách nhận phòng. Nhân viên lễ tân A muốn làm thủ tục trả phòng cho khách hàng B.
2. Hệ thống hiển thị giao diện yêu cầu tên hoặc số id của khách hàng hoặc số phòng của khách: 1 ô nhập id, 1 ô nhập tên và một nút tìm kiếm lịch sử đặt phòng của khác hàng
3. Nhân viên A hỏi lại khách hàng B số hiệu phòng muốn trả
4. Khách hàng B trả lời số hiệu phòng cho nhân viên A là 305
5. Nhân viên A nhập số hiệu phòng của khách hàng B ô tìm kiếm và click vào nút tìm kiếm.
6. Hệ thống tìm kiếm từ CSDL và hiển thị lên thông tin các phòng đã nhận của khách hàng B. Mỗi hàng một phòng với đầy đủ thông tin: tên (mã) phòng, kiểu phòng, giá đặt, mô tả, ngày bắt đầu, ngày kết thúc.
7. Nhân viên A click vào phòng 305 trên giao diện và chọn trả phòng
8. Hệ thống cập nhật thông tin nhận phòng vào CSDL và hiển thị thông tin hóa đơn cần thanh toán bao gồm các thông tin: mã hóa đơn, ngày tạo, tên và địa chỉ khách hàng, số hiệu phòng, kiểu phòng, đơn giá. Dòng tiếp theo ghi tổng số tiền của hóa đơn, một dòng ghi số tổng số tiền đã thanh toán trước đó, dòng cuối cùng tổng số tiền còn lại khách hàng phải thanh toán. Bên dưới có thêm một nút nhấn "bổ sung các dịch vụ"
9. Nhân viên A click vào bổ sung các dịch vụ
10. Hệ thống hiển thị một cửa sổ cho phép nhân viên A nhập thông tin các dịch vụ mà khách hàng B đã sử dụng trong khoảng thời gian ở khách sạn. Mỗi dòng có các thông tin: tên dịch vụ, đơn vị tính, đơn giá, thành tiền.
11. Nhân viên A nhập đầy đủ thông tin các dịch vụ mà khách hàng B đã sử dụng và click vào nút xác nhận.
12. Hệ thống quay lại giao diện hiển thị hóa đơn, có bổ sung thêm phần các dịch vụ đã dùng. Tổng số tiền phải trả cũng được cập nhật theo.
13. Nhân viên A thông báo số tiền phải trả cho khách hàng B
14. Khách hàng B thanh toán cho nhân viên A số tiền yêu cầu.
15. Nhân viên A click vào nút xác nhận thanh toán
16. Hệ thống lưu thông tin thanh toán vào CSDL và thông báo trả phòng thành công.
17. Nhân viên A thông báo lại cho khách hàng B kết thúc thành công giao dịch.

g. Scenario cho Use case Payment

Mô tả: Use case này cho phép Nhân viên lễ tân (Receptionist) thực hiện thanh toán và in hóa đơn cho khách hàng.

Khách hàng thanh toán trước một số tiền đặt phòng:

1. Nhân viên lễ tân A click vào chức năng thanh toán trong menu quản lí khách nhận phòng. Nhân viên lễ tân A muốn làm thủ tục thanh toán đặt cọc cho khách hàng B.
2. Hệ thống hiển thị giao diện yêu cầu tên hoặc số id của khách hàng: 1 ô nhập id, 1 ô nhập tên và một nút tìm kiếm lịch sử đặt phòng của khác hàng
3. Nhân viên A hỏi lại khách hàng B id và tên
4. Khách hàng B trả lời tên và số id của mình cho nhân viên A
5. Nhân viên A nhập id của khách hàng B ô tìm kiếm và click vào nút tìm kiếm.
6. Hệ thống tìm kiếm từ CSDL và hiển thị lên thông tin các lần đặt phòng của khách hàng B. Mỗi hàng một lần đặt với đầy đủ thông tin: tên (mã) phòng, kiểu phòng, giá đặt, mô tả, ngày bắt đầu, ngày kết thúc.
7. Nhân viên A yêu cầu khách hàng B và xác nhận thông tin phòng đặt: ngày đến và đi.
8. Khách hàng B xác nhận ngày đến ngày đi cho nhân viên A
9. Nhân viên A click vào lần đặt phòng tương ứng trên giao diện, và chọn thanh toán
10. Hệ thống hiển thị thông tin hóa đơn cần thanh toán bao gồm các thông tin: mã hóa đơn, ngày tạo, tên và địa chỉ khách hàng, số hiệu phòng, kiểu phòng, đơn giá. Dòng tiếp theo ghi tổng số tiền của hóa đơn, một dòng ghi số tổng số tiền đã thanh toán trước đó. Dòng cuối cùng yêu cầu nhập vào số tiền khách hàng muốn thanh toán.
11. Nhân viên A nhập vào số tiền đã nhận của khách hàng B
12. Hệ thống hiển thị lại thông tin hóa đơn một lần nữa: trong đó cập nhật lại số tiền đã thanh toán và số tiền còn lại cần thanh toán.
13. Nhân viên A thông báo lại cho khách hàng số tiền còn lại cần trả và in hóa đơn cho lần thanh toán này cho khách hàng B.

8.9.2 Trích các lớp thực thể

Mô tả hệ thống trong một đoạn văn như sau:

Hệ thống quản lí thông tin về phòng của khách sạn, thông tin về khách hàng đặt phòng. Hệ thống cho phép người quản lí có thể quản lí thông tin về phòng và khách sạn, cho phép khách hàng đặt phòng qua điện thoại thông qua nhân viên bán hàng, hoặc đặt phòng trực tiếp tại quầy thông qua nhân viên lễ tân. Hệ thống cũng cho phép nhân viên lễ tân thực hiện các hoạt động như nhận phòng, trả phòng, thanh toán khi có yêu cầu từ khách hàng. Mỗi khi thanh toán, hóa đơn sẽ được in ra cho khách hàng, bao gồm cả phí các dịch vụ mà khách hàng đã sử dụng khi nghỉ tại khách sạn.

Như vậy, ta có các danh từ và các phân tích như sau:

- Hệ thống: danh từ chung chung --> loại.
- Thông tin: danh từ chung chung --> loại.
- Phòng: là đối tượng xử lí của hệ thống --> là 1 lớp thực thể: Room
- Khách sạn: là đối tượng xử lí của hệ thống --> là 1 lớp thực thể: Hotel
- Khách hàng: là đối tượng xử lí của hệ thống --> là 1 lớp thực thể: Client

- Người quản lí: không phải là đối tượng xử lí trực tiếp của hệ thống, nhưng cũng bị quản lí cùng với nhân viên lễ tân và nhân viên bán hàng theo kiểu người dùng trực tiếp của phần mềm --> đề xuất là 1 lớp thực thể chung: User
- Điện thoại: không thuộc phạm vi xử lí của phần mềm --> loại
- Nhân viên bán hàng: không phải là đối tượng xử lí trực tiếp của hệ thống, nhưng cũng bị quản lí cùng với người quản lí và nhân viên lễ tân theo kiểu người dùng trực tiếp của phần mềm --> đề xuất là 1 lớp thực thể chung: User
- Quầy: không thuộc phạm vi xử lí của phần mềm --> loại
- Nhân viên lễ tân: không phải là đối tượng xử lí trực tiếp của hệ thống, nhưng cũng bị quản lí cùng với người quản lí và nhân viên bán hàng theo kiểu người dùng trực tiếp của phần mềm --> đề xuất là 1 lớp thực thể chung: User
- Hóa đơn: là đối tượng xử lí của hệ thống --> là 1 lớp thực thể: Bill
- Dịch vụ: là đối tượng xử lí của hệ thống --> là 1 lớp thực thể: Service

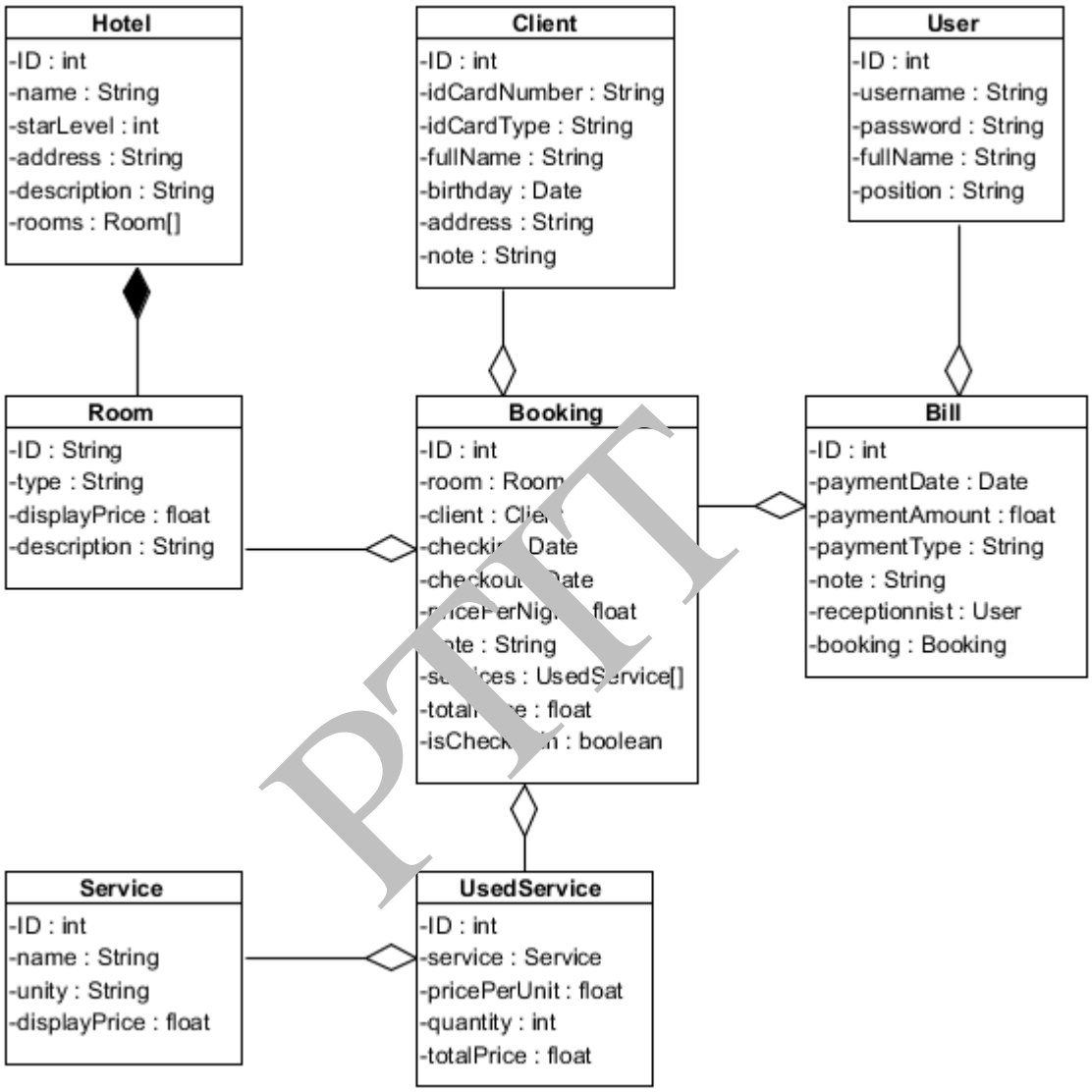
Vậy chúng ta thu được các lớp thực thể ban đầu là Room, Hotel, Client, User, Bill, và Service.

Quan hệ giữa các lớp thực thể được xác định như sau:

- Một Hotel có nhiều Room, một Room chỉ thuộc vào một Hotel. Vậy quan hệ giữa Hotel và Room là 1-n.
- Một Client có thể đặt nhiều Room, một Room có thể bị đặt trước bởi nhiều Client ở nhiều thời điểm khác nhau: quan hệ giữa Client và Room là n-n. Do đó có thể bổ sung một lớp thực thể liên kết giữa hai đối tượng này là Booking: Một Client và một Room sẽ xác định duy nhất một Booking tại một thời điểm nhất định. Liên kết này xác định thêm các thông tin: ngày đến, ngày đi, giá thực.
- Một Booking có thể được thanh toán nhiều lần khác nhau. Do đó có nhiều hóa đơn khác nhau. Vậy quan hệ giữa Booking và Bill là 1-n.
- Một nhân viên lễ tân có thể lập nhiều hóa đơn khác nhau cho các Booking khác nhau. Do đó quan hệ giữa User và Bill cũng là 1-n.
- Một Service có thể dùng bởi nhiều Client khác nhau, tại nhiều lần Booking khác nhau. Nhưng một Service chỉ được thanh toán 1 lần (có thể với nhiều đơn vị) tại một lần Booking. Một Booking có thể dùng nhiều Service khác nhau: Quan hệ giữa Booking và Service là 1-n. Tuy nhiên một Service sẽ thanh toán với giá khác nhau tại các lần Booking

khác nhau. Do đó đề xuất bổ sung lớp UsedService làm cầu nối 1-n giữa lớp Booking và Service.

Như vậy, ta thu được sơ đồ các lớp thực thể của hệ thống như Hình 8.10.



Hình 8.10 : Sơ đồ lớp thực thể của hệ thống

8.9.3 Viết lại các scenario

CHƯƠNG 9: PHA THIẾT KẾ

9.1 TỔNG QUAN VỀ PHA THIẾT KẾ

9.1.1 Dữ liệu và các hành động

- Hai khía cạnh của một sản phẩm phần mềm
    - Những hành động thực hiện trên dữ liệu
    - Dữ liệu mà các hành động thao tác trên dữ liệu đó
- Hai cách cơ bản của thiết kế hệ thống phần mềm
    - Thiết kế hướng hành động
    - Thiết kế hướng dữ liệu
- Cách thứ ba
    - Các phương pháp lai
    - Chẳng hạn, thiết kế hướng đối tượng

9.1.2 Thiết kế và trừu tượng

- Các hoạt động thiết kế cấp cao
    - Thiết kế kiến trúc
    - Thiết kế chi tiết
    - Kiểm thử thiết kế
- Thiết kế kiến trúc
    - Đầu vào: Những đặc tả
    - Đầu ra: Sự phân nhỏ thành các mô đun
- Thiết kế chi tiết
    - Mỗi mô đun được thiết kế
        - Các thuật toán đặc trưng, các cấu trúc dữ liệu

9.1.3 Thiết kế

- Tổng kết luồng công việc thiết kế:
    - Các tài liệu luồng công việc thiết kế được lặp và tích hợp đến khi người lập trình có thể sử dụng được chúng
- Các quyết định bao gồm:
    - Ngôn ngữ lập trình
    - Sử dụng lại
    - Tính khả chuyển
- Ý tưởng của việc phân tách luồng công việc lớp thành những luồng công việc nhỏ độc lập (các gói) được thực hiện ở luồng công việc phân tích
- Mục tiêu là chia nhỏ luồng công việc cài đặt thành những phần có thể quản lý được
    - *Các hệ thống con*
- Tại sao phần mềm được chia nhỏ thành các hệ thống con:
    - Dễ dàng để cài đặt một số hệ thống con hơn là một hệ thống lớn
    - Nếu các hệ thống con độc lập với nhau thì chúng có thể được cài đặt bởi các đội lập trình khác nhau cùng một lúc
        - Khi đó toàn bộ sản phẩm phần mềm được chuyển giao sớm
- Kiến trúc của sản phẩm phần mềm bao gồm:
    - Các thành phần khác nhau
    - Cách chúng ăn khớp với nhau
    - Phân chia các thành phần vào các hệ thống con
- Công việc của thiết kế kiến trúc được chuyên môn hóa
    - Nó được thực hiện bởi các kiến trúc sư phần mềm
- Kiến trúc sư (architect)cần có sự cân bằng về:
    - Mỗi sản phẩm phần mềm phải đáp ứng các yêu cầu chức năng của chúng (các use case)

- Mỗi sản phẩm phần mềm cũng phải đáp ứng các yêu cầu phi chức năng, bao gồm:
  - Khả chuyển, đáng tin, mạnh mẽ, bảo trì và bảo mật
- Mỗi sản phẩm phần mềm phải thực hiện tất cả những yêu cầu này trong ràng buộc chi phí và thời gian

- Kiến trúc sư phải giúp khác hàng bằng cách sắp xếp những cân bằng này.
- Thường không thể đáp ứng tất cả các yêu cầu chức năng và phi chức năng trong ràng buộc về chi phí và thời gian
  - Có một vài sự sắp xếp được thực hiện
- Khách hàng phải
  - Giảm bớt một số yêu cầu;
  - Tăng chi phíe; và /hoặc
  - Thay đổi thời gian chuyển giao
- Kiến trúc của sản phẩm phần mềm là quan trọng
  - Luồng công việc xác định yêu cầu có thể được sửa lại (fix) trong suốt luồng phân tích
  - Luồng công việc phân tích có thể được sửa lại trong suốt luồng công việc thiết kế
  - Luồng công việc thiết kế có thể được sửa lại trong suốt luồng công viec cài đặt
- Nhưng không có cách để phục hồi từ kiến trúc gần tốt nhất
  - Kiến trúc phải được thiết kế lại ngay lập tức

9.1.4 Kiểm thử trong pha thiết kế

- Rà soát thiết kế phải được thực hiện
  - Thiết kế phải phản ánh chính xác đặc tả
  - Chính thiết kế phải chính xác
- Kiểm tra kỹ lưỡng hướng giao tác (Transaction-driven inspections)
  - Cần thiết cho các phần mềm hướng giao tác

- Tuy nhiên, những kiểm tra hướng giao tác là chưa đủ nên những kiểm tra hướng đặc tả cũng được yêu cầu

9.1.5 Kỹ thuật hình thức cho thiết kế chi tiết

- Việc cài đặt một phần mềm hoàn thiện và sau đó chứng minh nó là chính xác là rất khó
- Tuy nhiên, sử dụng kỹ thuật hình thứuc trong suốt thiết kế chi tiết có thể giúp:
  - Việc chứng minh tính chính xác có thể được áp dụng đối với các phần mô đun
  - Thiết kế có ít lỗi hơn nếu nó được phát triển song song với sự kiểm chứng tính chính xác
  - Nếu cùng một người lập trình thực hiện cả thiết kế chi tiết và cài đặt
    - Người lập trình sẽ có một quan điểm tích cực đối với thiết kế chi tiết
    - Chính điều này dẫn đến ít lỗi

9.1.6 Kỹ thuật thiết kế hệ thống thời gian thực

- Những kho khăn của hệ thống thời gian thực
  - Đầu vào lấy từ thế giới thực
    - Phần mềm không có điều khiển bộ định thời của các đầu vào (Software has no control over the timing of the inputs)
  - Được cài đặt thường xuyên trên phần mềm phân tán
    - Communications implications
    - Những vấn đề định thời (Timing issues)
  - Những vấn đề của đồng bộ
    - Race conditions
    - Bế tắc (Deadlock - deadly embrace)
- Những khó khăn chính trong thiết kế hệ thống thời gian thực
  - Xác định liệu những ràng buộc về thời gian có được đáp ứng bởi thiết kế không?
- Hầu hết các phương thức thiết kế thời gian thực là những sự mở rộng của các phương thức phi thời gian thực (Most real-time design methods are extensions of non-real-time methods to real-time)

- Chúng ta đã hạn chế những thử nghiệm trong cách sử dụng bất cứ phương thực thời gian thực nào. (We have limited experience in the use of any real-time methods)
- The state-of-the-art is not where we would like it to be

9.1.7 Công cụ CASE cho thiết kế

- Rất quan trọng để kiểm tra tài liệu thiết kế hợp nhất với mọi khía cạnh của phân tích
    - Để xử lý tài liệu phân tích và thiết kế chúng ta cần một công cụ upperCASE
- Công cụ upperCASE
    - Được xây dựng xung quanh từ điển dữ liệu(Are built around a data dictionary)
    - They incorporate a consistency checker, and
    - Màn ảnh và các bản tường trình (Screen and report generators)
    - Công cụ quản lý đôi khi bao gồm
        - Ước lượng
        - Lập kế hoạch
- Ví dụ của các công cụ cho thiết kế hướng đối tượng
    - Công cụ thương mại
        - Software through Pictures
        - IBM Rational Rose
        - Together
    - Công cụ mã nguồn mở
        - ArgoUML

9.1.8 Thước đo cho thiết kế

- Thước đo chất lượng thiết kế
    - Kết dính (Cohesion)
    - Sự kết nối (Coupling)
    - Thống kê lỗi

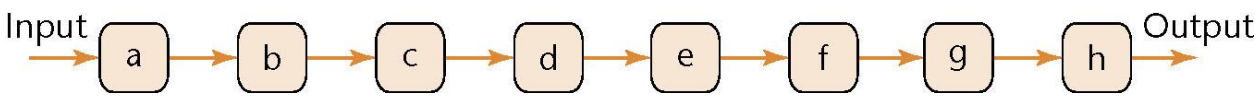
- Cyclomatic complexity is problematic
    - Độ phức tạp dữ liệu được bỏ qua
    - Nó không được sử dụng nhiều với mô hình hướng đối tượng
- Các thước đo được đề xuất cho mô hình hướng đối tượng
    - Chúng đã chứng tỏ được tính hữu ích trong cả lý thuyết và thử nghiệm.(They have been challenged on both theoretical and experimental grounds)

9.1.9 Những thách thức của pha thiết kế

- Đội thiết kế không nên làm quá nhiều
    - Thiết kế chi tiết không nên là những người viết mã
- Đội thiết kế không nên làm quá ít
    - Đủ để cho đội thiết kế để đưa ra một thiết kế chi tiết hoàn thiện
- Chúng ta cần "phát triển " những người thiết kế giỏi
- Những người thiết kế giỏi tiềm năng phải
    - Được nhận định
    - Đã được đào tạo một cách hình thức,
    - Đang học nghề để trở thành một người thiết kế giỏi, và
    - Được phép giao tiếp với những người thiết kế khác
- Phải có một hướng đi nghề nghiệp cụ thể cho những người thiết kế này, với một chế độ thưởng thích hợp

9.2 THIẾT KẾ HƯỚNG HÀNH ĐỘNG

- Phân tích luồng dữ liệu
    - Sử dụng phân tích luồng dữ liệu với hầu hết các phương pháp đặc tả (ở đây là phân tích hệ thống theo hướng cấu trúc)
- Điểm chính: Chúng ta đã chi tiết hóa các thông tin hành động từ DFD

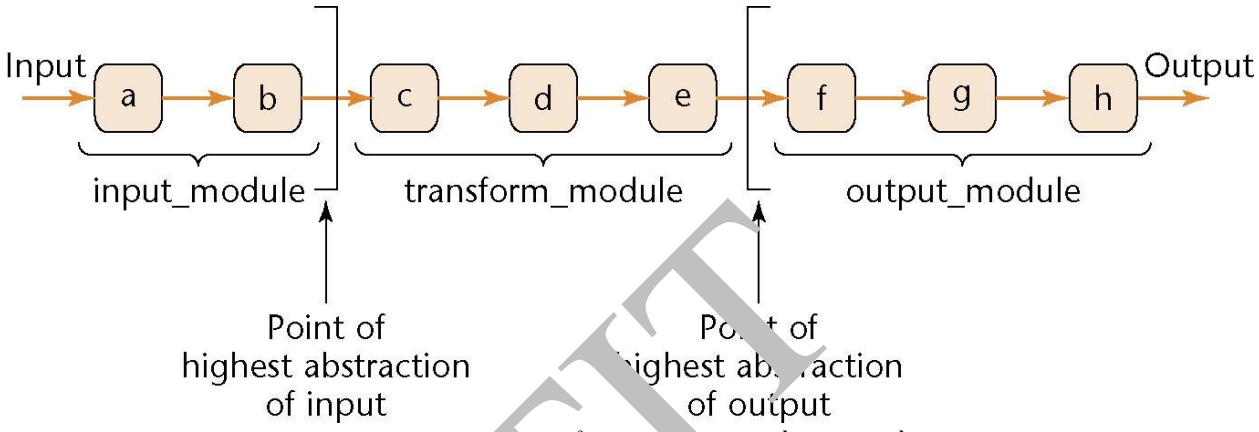


Hình 9.1: Luồng dữ liệu

9.3 PHÂN TÍCH VA THIẾT KẾ DÒNG DỮ LIỆU

9.3.1 Phân tích dòng dữ liệu

- Mỗi sản phẩm phần mềm biến đổi từ đầu vào thành đầu ra
- Xác định
  - "Điểm trừu tượng nhất của đầu vào"
  - "Điểm trừu tượng nhất của đầu ra"

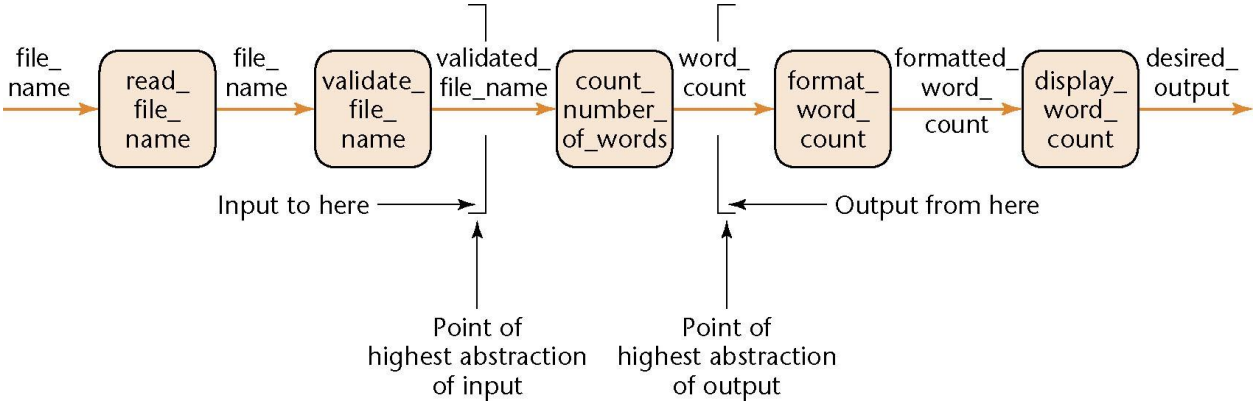


Hình 9.2: Tìm các điểm trừu tượng đầu vào, đầu ra

- Phân chia phần mềm thành các mô đun
- Lặp lại các bước cho đến khi mỗi mô đun có độ dính kết cao (cohesion)
  - Những chỉnh sửa phụ có thể cần thiết để giảm bớt độ kết nối (coupling)

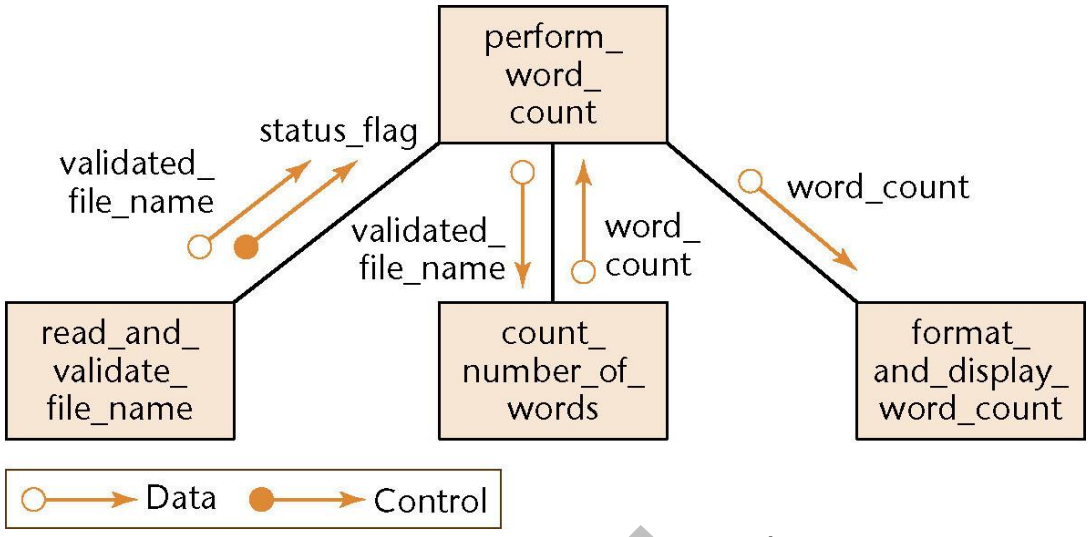
9.3.1.1 Bài toán đếm từ

- Ví dụ: Thiết kế một hệ thống phần mềm với đầu vào là một tên tệp, và kết quả trả về là số lượng từ trong file đó(giống như UNIX *wc*)



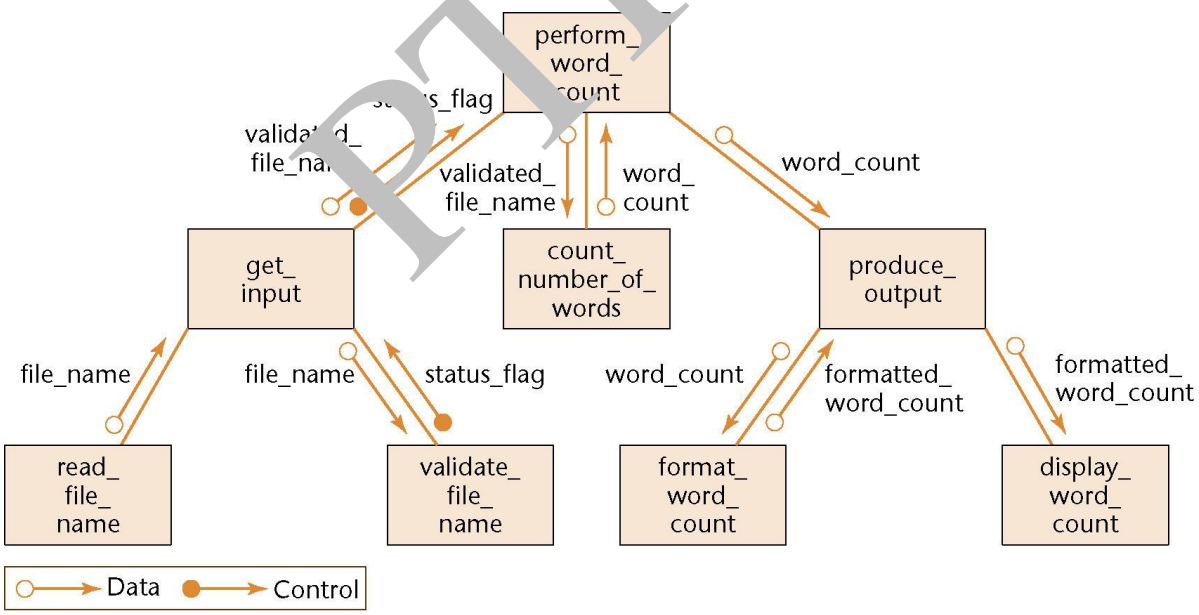
Hình 9.3: Xác định điểm trừu tượng vào/ra cho bài toán đếm từ

- Quá trình làm mịn thứ nhất



Hình 9.4: Quá trình làm mịn thứ nhất

- Làm mịn hai mô đun ở mức dính kết giao tiếp (communicational cohesion)
- Bước làm mịn thứ hai



Hình 9.5: Quá trình làm mịn thứ hai

- Tất cả 8 mô đun đều ở mức dính kết chức năng (functional cohesion)
- Thiết kế kiến trúc đã hoàn thiện, vì thế thực hiện thiết kế chi tiết.
- Hai định dạng để biểu diễn thiết kế chi tiết:

- Tabular

Module name	**read_file_name**
Module type	Function
Return type	**string**
Input arguments	None
Output arguments	None
Error messages	None
Files accessed	None
Files changed	None
Modules called	None
Narrative	The product is invoked by the user by means of the command string **word_count <file_name>** Using an operating system call, this module accesses the contents of the command string input by the user, extracts **<file_name>**, and returns it as the value of the module.

Module name	**validate_file_name**
Module type	Function
Return type	**Boolean**
Input arguments	**file_name : string**
Output arguments	None
Error messages	None
Files accessed	None
Files changed	None
Modules called	None
Narrative	This module makes an operating system call to determine whether file **file_name** exists. The module returns **true** if the file exists and **false** otherwise.

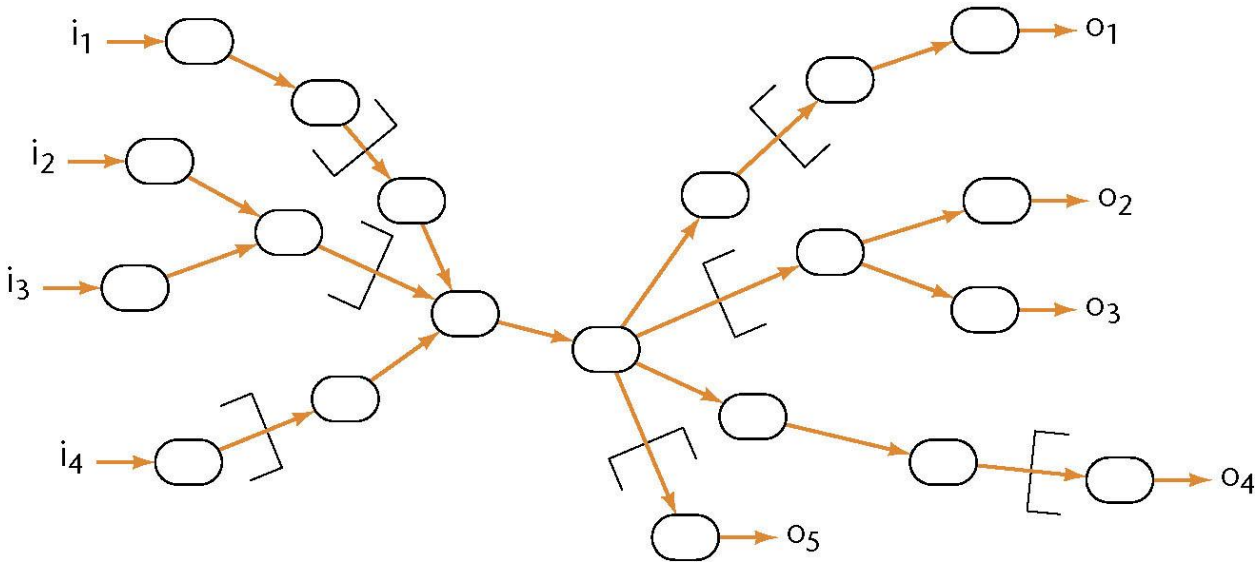
Module name	**produce_output**
Module type	Function
Return type	**void**
Input arguments	**word_count : integer**
Output arguments	None
Error messages	None
Files accessed	None
Files changed	None
Modules called	**format_word_count** arguments: **word_count : integer** **formatted_word_count : string** **display_word_count** arguments: **formatted_word_count : string**
Narrative	This module takes the integer **word_count** passed to it by the calling module and calls **format_word_count** to have that integer formatted according to the specifications. Then it calls **display_word_count** to have the line printed.

- Mã giả (PDL — program design language – ngôn ngữ thiết kế chương trình)

```
void perform_word_count()
{
  String                validate_file_name;
  int                    word_count;

  if (get_input (validate_file_name) is false)
     print "error 1: file does not exist";
  else
  {
    set word_count equal to count_number_of_words (validate_file_name);
    if (word_count is equal to –1)
       print "error 2: file is not a text file";
    else
      produce_output (word_count);
  }
}

Boolean get_input (String validate_file_name)
{
  String                file_name;

  file_name = read_file_name ();
  if (validate_file_name (file_name) is true)
  {
     set validate_file_name equal to file_name
     return true;
  }
  else
    return false;
}

void display_word_count (String formatted_word_count)
{
  print formatted_word_count, to justified;
}

String format_word_count (int word_count);
{
  return "File contains" word_count "words";
}
```


9.3.1.2 Mở rộng phân tích luồng dữ liệu

- Sản phẩm phần mềm trong thực tế, có
    - Nhiều hơn một luồng đầu vào
    - Nhiều hơn một luồng đầu ra
- Tìm điểm trừu tượng nhất của mỗi luồng



Hình 9.6: mở rộng luồng phân tích dữ liệu

- Tiếp tục thực hiện cho đến khi mỗi mô đun có độ dính kết cao
    - Điều chỉnh kết nối (coupling) nếu cần thiết

9.3.2 Thiết kế dòng dữ liệu

- Nguyên lý cơ bản
    - Cấu trúc của một phần mềm phải phù hợp với cấu trúc của dữ liệu của nó
- Có ba phương pháp rất giống nhau
    - Michael Jackson [1975], Warnier [1976], Orr [1981]
    - Thiết kế hướng dữ liệu
    - Chưa từng phổ biến như thiết kế hướng hành động
    - Với sự ra đời của OOD, thiết kế hướng dữ liệu đã bị lỗi thời

9.4 THIẾT KẾ HƯỚNG ĐỐI TƯỢNG

- Mục đích
    - Thiết kế phần mềm dưới dạng các lớp được trích rút trong suốt phân tích hướng đối tượng (OOA)
- Nếu chúng ta đang sử dụng một ngôn ngữ mà không có tính kế thừa như C, Ada83…
    - Sử dụng thiết kế kiểu dữ liệu trừu tượng

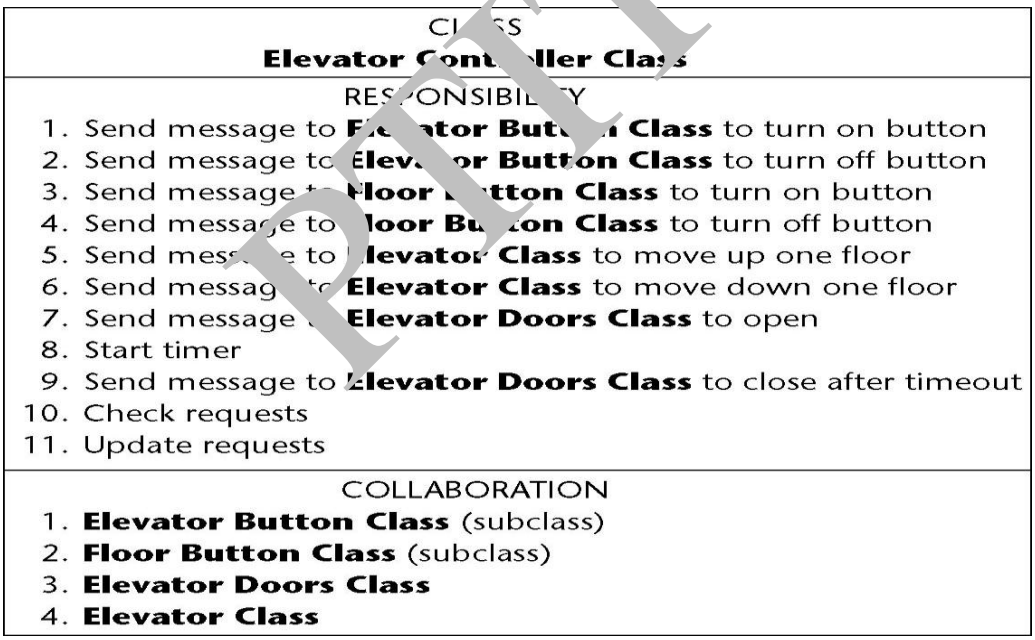
- Nếu chúng ta đang sử dung ngôn ngữ không có khai báo kiểu như FOTRAN, COBAL…
    - Sử dụng đóng gói dữ liệu

OOD gồm 2 bước:
Bước 1. Hoàn thiện biểu đồ lớp

- Xác định định dạng của các thuộc tính : Định dạng của các thuộc tính có thể được rút ra từ tài liệu phân tích .Ví dụ: Ngày theo định dạng của Mỹ (mm/mm/yyyy) và định danh của Châu Âu (dd/mm/yyyy). Trong cả hai trường hợp đều yêu cầu 10 ký tự.Định dạng có thể được thêm vào trong suốt quá trình phân tích. Để cực tiểu hóa việc làm lại, không bao giờ thêm một mục vào biểu đồ UML cho đến lúc cần thiết

- Gán mỗi phương thức cho 1lớp hoặc một client đã gửi thông điệp tới đối tượng của lớp đó. Với 3 nguyên lý: Nguyên lý A: Ẩn giấu thông tin. Nguyên lý B: Nếu một phương thức được gọi bởi nhiều client của một đối tượng thì sẽ gán phương thức đó cho đối tượng chứ không phải các client. Nguyên lý C: thiết kế hướng trách nhiệm

- Xem xét vòng lặp thứ hai của thẻ CRC của điều khiển thang máy

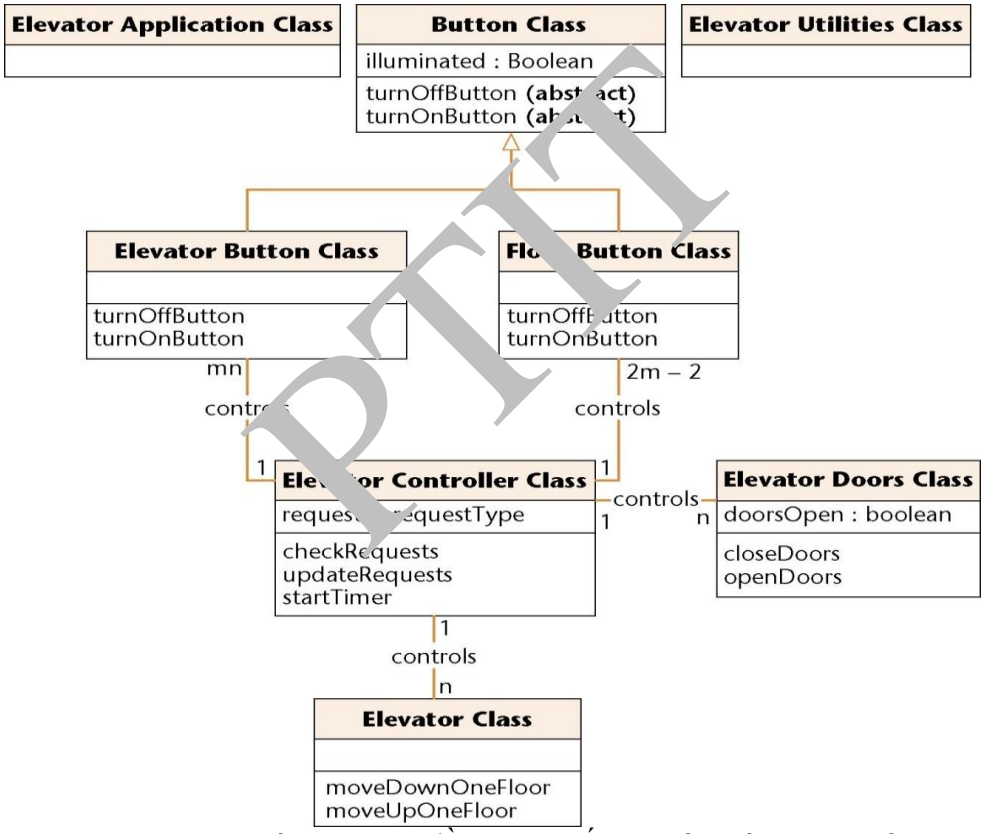


Hình 9.7: Xác định trách nhiệm từ thẻ CRC

- Trách nhiệm sau được gán trách nhiệm cho lớp điều khiển thang máy
    - 8. Bắt đầu đặt thời gian
    - 10. Kiểm tra yêu cầu
    - 11. Cập nhật yêu cầu
- Bởi vì chúng được thực hiện bởi lớp điều khiển thang máy

- Tám trách nhiệm còn lại có cùng một dạng:
  - "Gửi một thông điệp tới một lớp khác để yêu cầu nó làm một cái gì đó"
- Những trách nhiệm này được gán cho các lớp khác
  - Thiết kế hướng trách nhiệm
  - Xem xét độ an toàn
- Các phương thức *mở cửa*, *đóng cửa* được gán cho lớp **Cửa thang máy**
- Các phương thức *mở nút*, *tắt nút* được gán cho lớp **Nút Tầng** và **lớp Thang Máy**

- Biểu đồ lớp chi tiết cho bài toán thang máy:



Hình 9.8: Sơ đồ lớp chi tiết cho bài toán thang máy

Bước 2. Thực hiện thiết kế chi tiết

- Thiết kế chi tiết của *vòng lặp sự kiện thang máy* được đưa ra từ biểu đồ trạng thái

```
void elevatorEventLoop (void)
{
  while (TRUE)
  {
    if (a button has been pressed)
      if (button is not on)
      {
        updateRequests;
        button::turnOnButton;
      }
    else if (elevator is moving up)
    {
      if (there is no request to stop at floor f)
        elevator::moveUpOneFloor;
      else
      {
        stop elevator by not sending a message to move;
        elevatorDoors::openDoors;
        startTimer;
        if (elevatorButton is on)
          elevatorButton::turnOffButton;
        updateRequests;
      }
    }
    else if (elevator is moving down)
      [similar to up case]
    else if (elevator is stopped and request is pending)
    {
      elevatorDoors::closeDoors;
      determine direction of next request;
      if (appropriate floorButton is on)
        floorButton::turnOffButton;
      elevator::moveUp/DownOneFloor;
    }
    else if (elevator is at rest and r... (request... pending))
      elevatorDoors::closeDoors;
    else
      there are no requests, elevator is stoppe... elevatorDoors closed, so do nothing;
  }
}
```


9.5 CASE STUDY CHO PHA THIẾT KẾ

Mục này tiếp tục tiến hành thiết kế hệ thống cho ứng dụng quản lí đặt phòng khách sạn đã được trình bày trong case study của các pha lấy yêu cầu và pha phân tích. Nội dung bao gồm:

- Thiết kế cơ sở dữ liệu
- Thiết kế hệ thống: chúng ta sẽ học một số cách thiết kế khác nhau cho cùng một ứng dụng: thiết kế không dựa vào lớp điều khiển, thiết kế có dựa vào lớp điều khiển, và thiết kế theo mô hình MVC cải tiến. Để minh họa cho các hướng thiết kế này, bài giảng sẽ trình bày phần thiết kế cho 3 modul: thêm một phòng khách sạn, sửa thông tin một phòng khách sạn, và khách hàng đặt phòng.

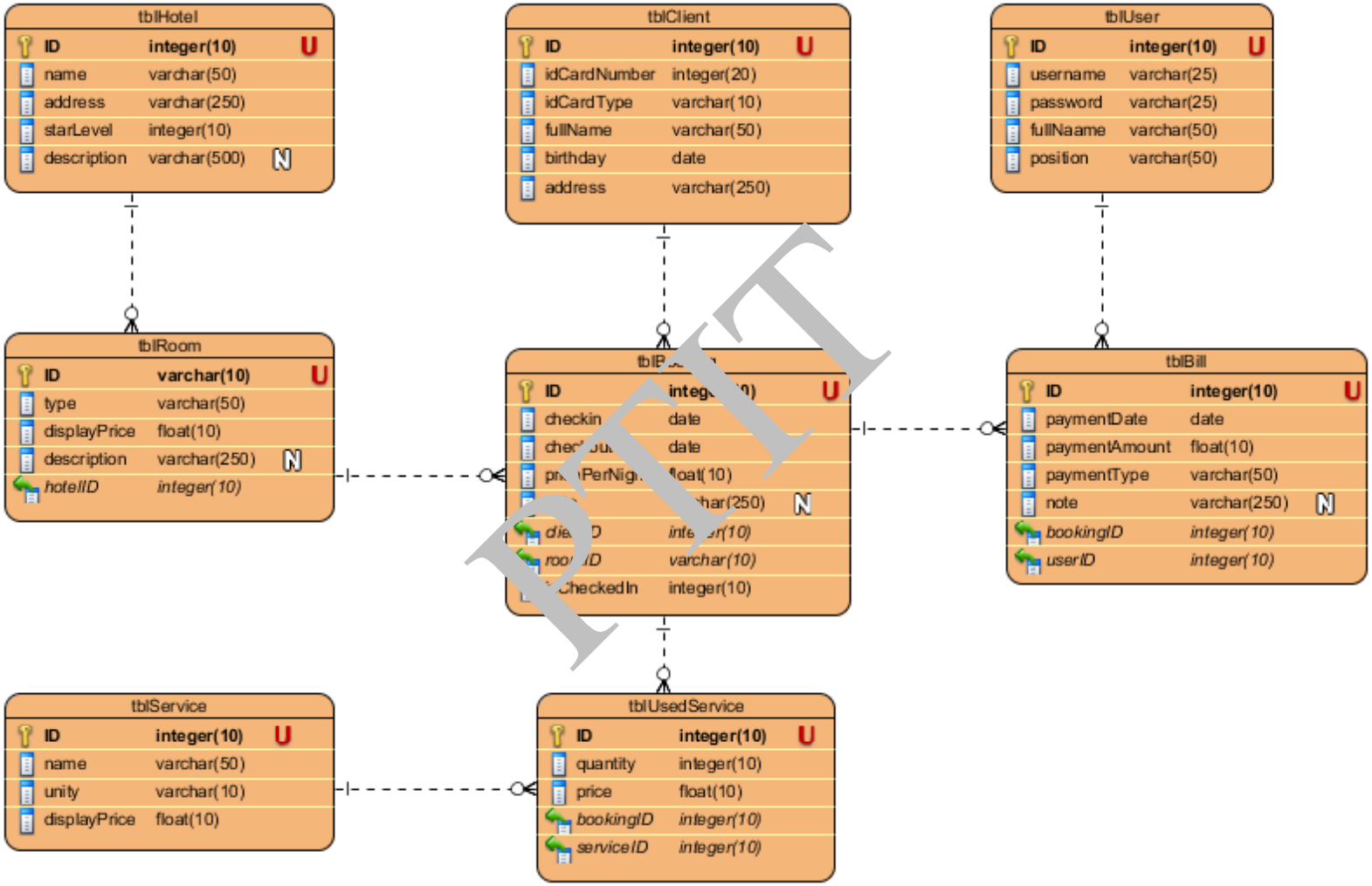
9.5.1 Thiết kế cơ sở dữ liệu

Dựa vào sơ đồ lớp thực thể đã trích được trong pha phân tích (Hình 8.10), chúng ta có thể đề xuất các bảng dữ liệu như sau:

- tblHotel: lưu các thông tin về khách sạn, bao gồm: id, tên, địa chỉ, hạng sao, và mô tả về khách sạn.
- tblRoom: lưu các thông tin về phòng khách sạn, bao gồm: id (cũng là số hiệu phòng), kiểu phòng, giá hiển thị, và mô tả chung về phòng.
- tblClient: lưu thông tin các khách hàng đặt phòng, bao gồm: id, số thẻ căn cước, kiểu thẻ căn cước, họ tên đầy đủ, ngày sinh, và địa chỉ.
- tblBooking: lưu thông tin về từng lần đặt chỗ của khách hàng, bao gồm: ngày đến, ngày đi, giá đặt, trạng thái đã checkin hay chưa, và ghi chú bổ sung theo yêu cầu của khách hàng để khách sạn lưu ý.
- tblUser: lưu thông tin về người dùng của phần mềm: id, tên đăng nhập, mật khẩu, và vị trí công việc.
- tblService: lưu thông tin các dịch vụ kèm theo phòng và có tính phí cho khách hàng, mỗi dịch vụ được lưu bởi: id, tên dịch vụ, đơn vị tính, đơn giá hiển thị.
- tblUsedService: lưu thông tin các dịch vụ đã được sử dụng bởi khách đặt phòng, bao gồm: id, số lượng theo đơn vị tính, đơn giá thực trả.
- tblBill: lưu thông tin hóa đơn mỗi lần thanh toán của khác hàng, bao gồm: id, ngày trả, số tiền thanh toán cho lần trả đó, hình thức thanh toán.

Quan hệ giữa các bảng được mô tả như Hình 9.9:

- Bảng tblHotel quan hệ 1-n với bảng tblRoom
- Bảng tblRoom và bảng tblClient đều quan hệ 1-n với bảng tblBooking
- Bảng tblService và bảng tblBooking đều quan hệ 1-n với bảng tblUsedService
- Bảng tblUser và bảng tblBooking đều quan hệ 1-n với bảng tblBill.



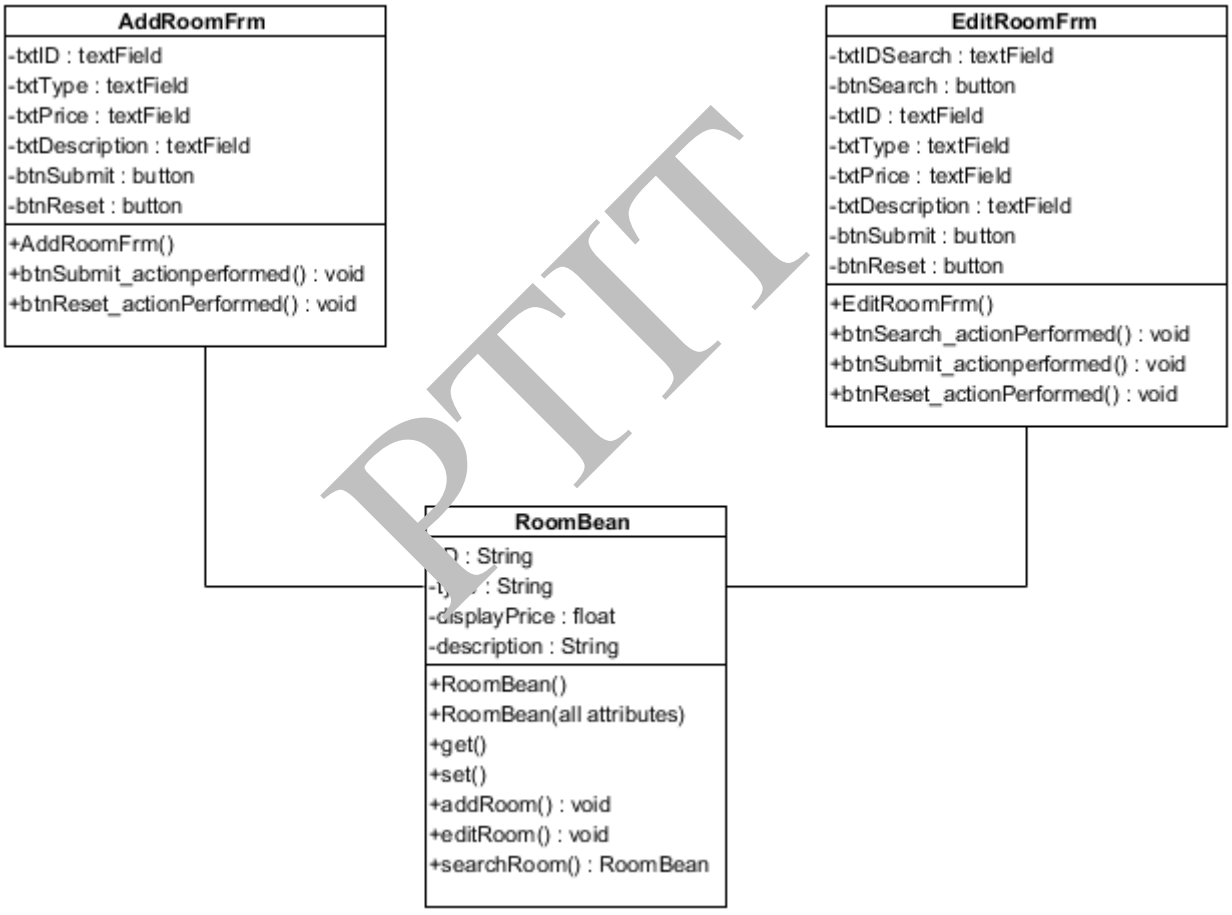
Hình 9.9: Sơ đồ thiết kế CSDL cho hệ thống

9.5.2 Thiết kế dùng bean thay cho control

Tư tưởng chủ đạo của phương pháp này là đóng gói thông tin và hành động của các lớp thực thể thành một lớp chung, gọi là lớp bean (bản chất không còn là lớp thực thể nữa, mà đã bao gồm các phương thức của lớp điều khiển). Do đó, hệ thống sẽ không còn cần đến lớp điều khiển. Khi có sự kiện trên form, lớp giao diện sẽ gọi hàm actionPerformed(), hàm này sẽ gọi phương thức tương ứng của lớp bean để truy cập CSDL.

a. Thiết kế cho chức năng thêm/sửa phòng

+ Sơ đồ lớp chi tiết cho modul



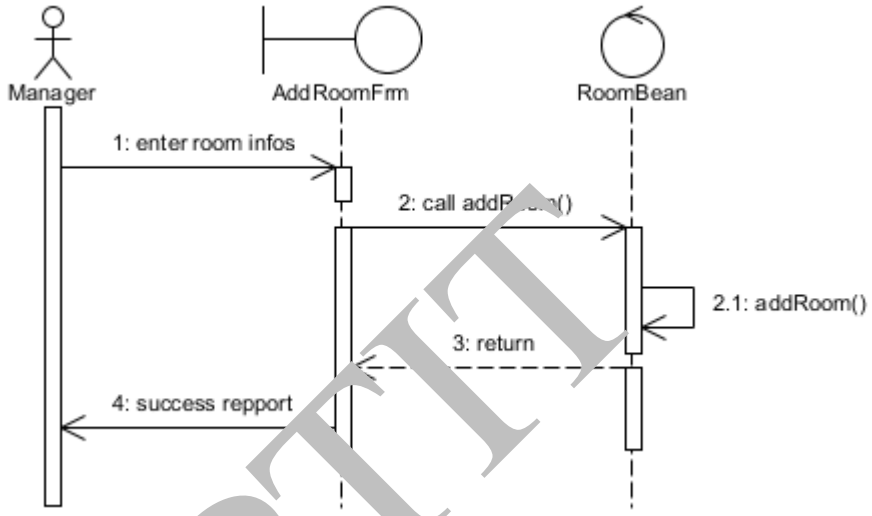
Hình 9.10: Sơ đồ lớp thiết kế kiểu bean cho modul thêm/sửa phòng

Vì thiết kế theo kiểu dùng bean nên lớp RoomBean vừa chứa các thuộc tính của phòng, vừa chứa các phương thức thêm, sửa, tìm kiếm phòng từ CSDL. Trong các lớp giao diện thì các phương thức xử lí sự kiện actionPerformed() sẽ gọi trực tiếp các phương thức của lớp bean để thự hiện.

+ Viết lại scenario cho chức năng nhập mới thông tin phòng

1. Người quản lí nhập thông tin một phòng mới vào AddRoomFrm và click vào nút Submit trên form
2. Lớp AddRoomFrm gọi phương thức addRoom() của lớp RoomBean
3. Lớp Roombean thực hiện phương thức addRoom() rồi trả về cho lớp AddRoomFrm
4. Lớp AddRoomFrm hiện thông báo thêm phòng thành công cho người quản lí.

+ Sơ đồ tuần tự cho chức năng nhập mới thông tin phòng



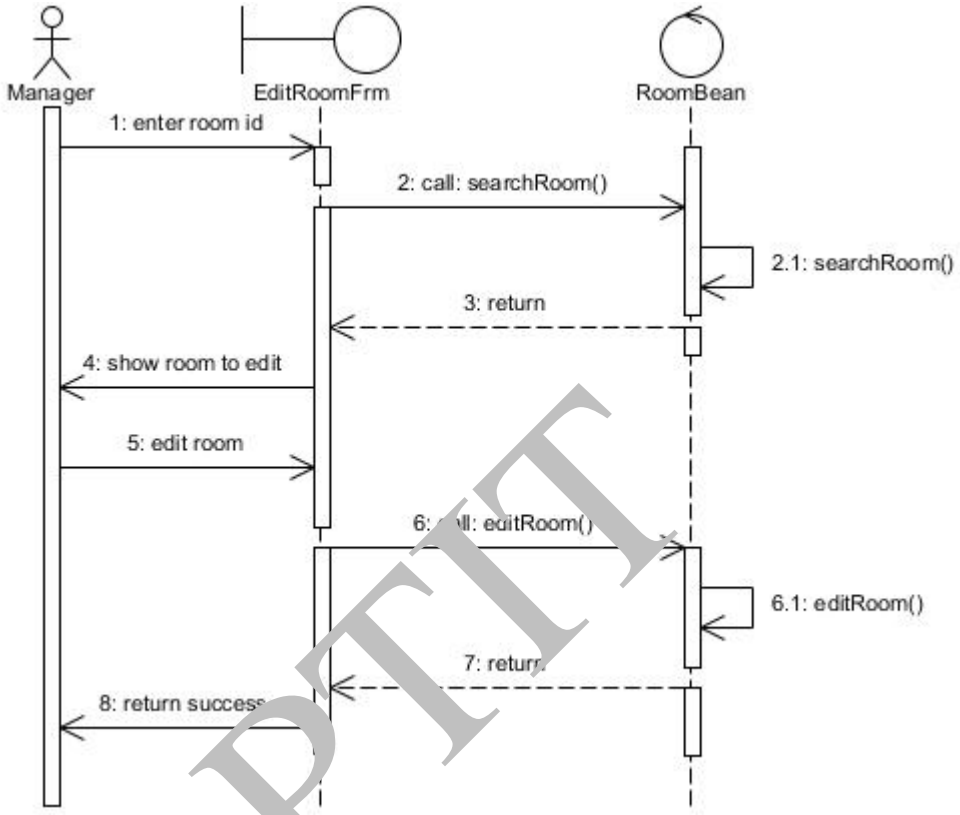
Hình 9.11: Sơ đồ tuần tự cho chức năng thêm phòng, dùng bean

+ Viết lại scenario cho chức năng cập nhật thông tin phòng

1. Người quản lí nhập tên phòng muốn thay đổi vào EditRoomFrm và click vào nút Search trên form
2. Lớp EditRoomFrm gọi phương thức searchRoom(ID) của lớp RoomBean
3. Lớp Roombean thực hiện phương thức searchRoom(ID) rồi trả về cho lớp EditRoomFrm
4. Lớp EditRoomFrm hiện thông tin phòng lên các ô thuộc tính tương ứng trên form cho người quản lí sửa đổi.
5. Người quản lí sửa đổi các thuộc tính cần thiết trên form và click vào nút Submit
6. Lớp EditRoomFrm gọi phương thức btnSubmit_actionPerformed() để thực hiện.
7. Phương thức btnSubmit_actionPerformed() gọi phương thức editRoom() của lớp RoomBean

8. Lớp Roombean thực hiện phương thức editRoom() rồi trả về cho lớp EditRoomFrm
9. Lớp EditRoomFrm hiện thông báo sửa đổi thành công cho người quản lí.

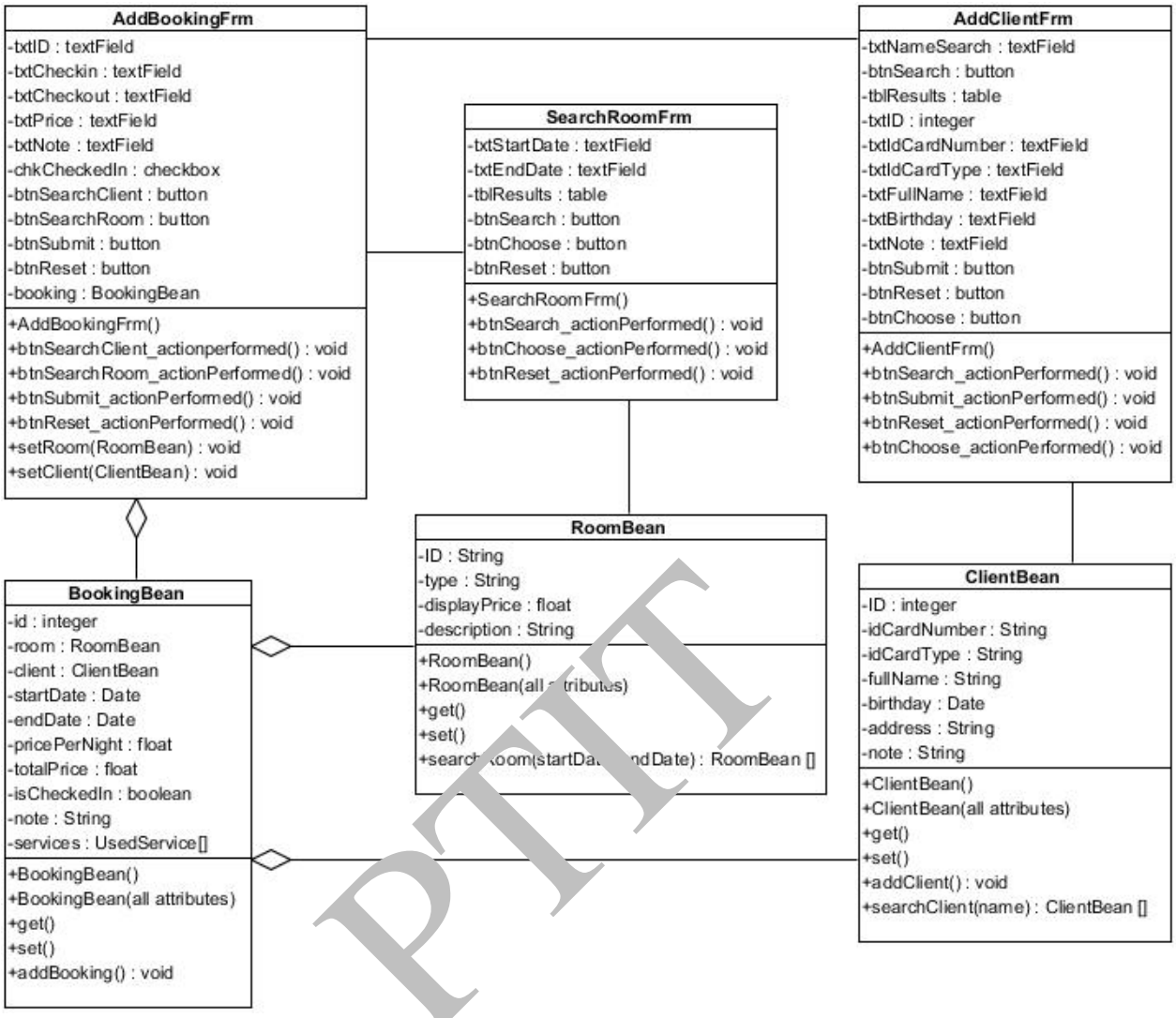
+ Sơ đồ tuần tự cho chức năng cập nhật thông tin phòng



Hình 9.12: Sơ đồ tuần tự cho chức năng sửa thông tin phòng, dùng bean

b. Thiết kế cho chức năng đặt phòng

+ Sơ đồ lớp chi tiết cho modul



Hình 9.13: Sơ đồ lớp cho chức năng thêm/sửa phòng, thiết kế dùng bean

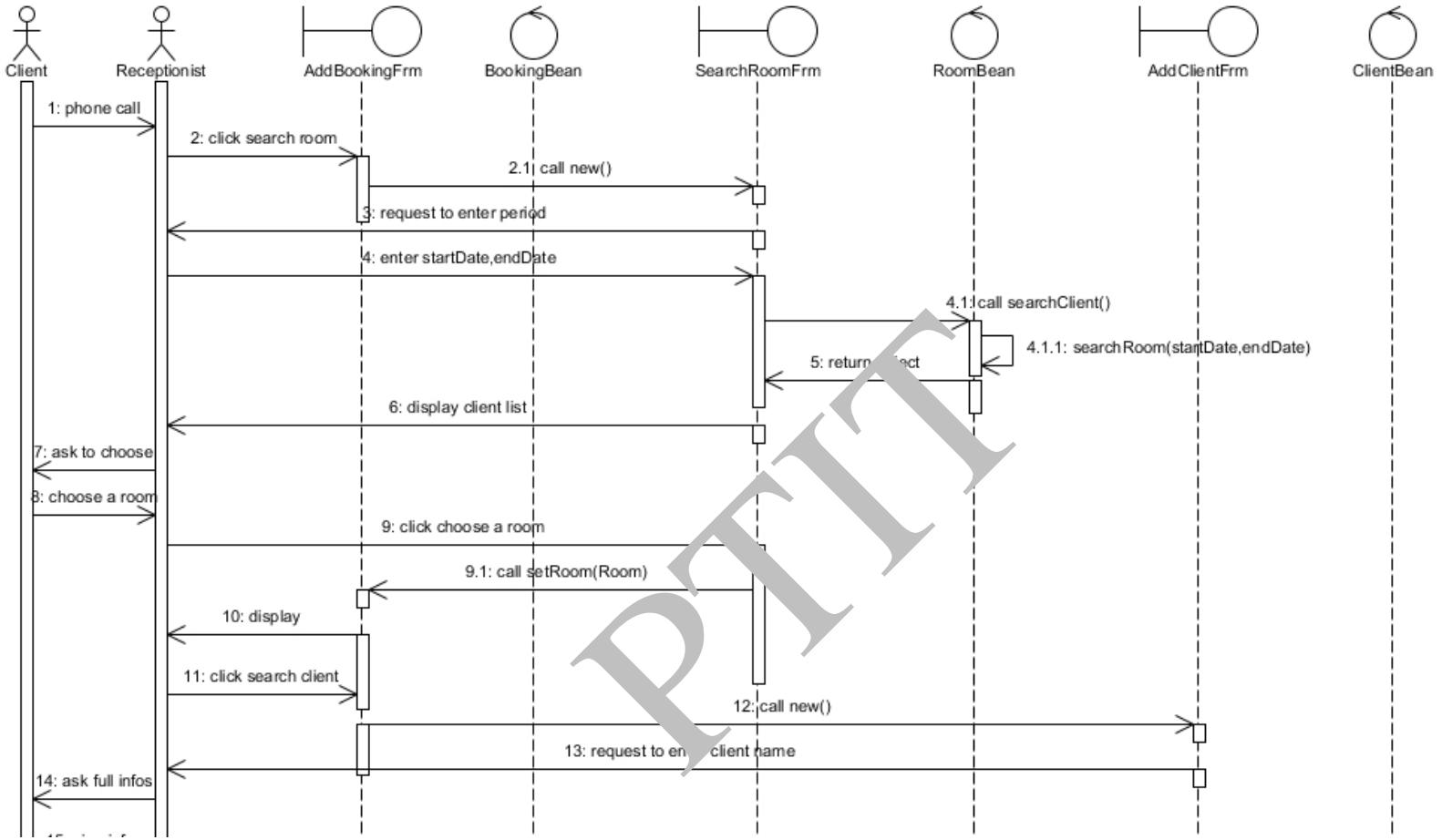
+ Viết lại scenario

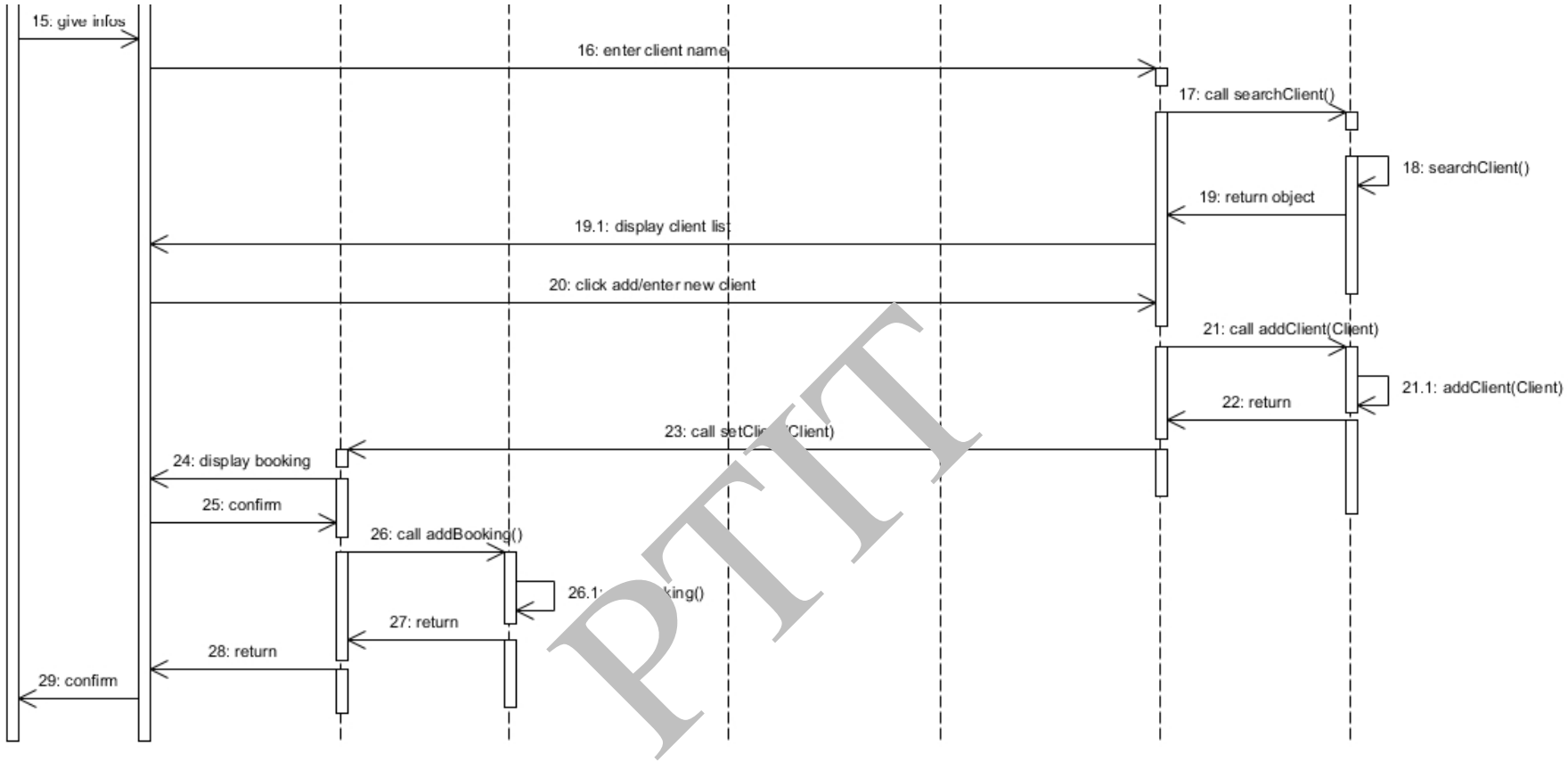
1. Khách hàng gọi điện cho nhân viên bán hàng yêu cầu đặt phòng.
2. Nhân viên chọn chức năng tìm kiếm phòng trống trên form AddBookingFrm.
3. Lớp AddBookingFrm gọi form SearchRoomFrm hiển thị.
4. Lớp form SearchRoomFrm hiển thị yêu cầu nhân viên nhập ngày checkin, ngày checkout để tìm kiếm.
5. Nhân viên nhập ngày checkin, checkout theo yêu cầu khách hàng và click vào nút tìm kiếm.
6. Lớp SearchRoomFrm gọi phương thức searchRoom() của lớp RoomBean.

7. Lớp RoomBean thực hiện hàm searchRoom() và trả kết quả lại cho lớp SearchRoomFrm.
8. Lớp SearchRoomFrm hiển thị danh sách các phòng trống lên cho nhân viên.
9. Nhân viên thông báo các phòng có thể đặt cho khách hàng.
10. Khách hàng chọn 1 phòng theo yêu cầu và trả lời cho nhân viên.
11. Nhân viên click chọn phòng tương ứng trên giao diện SearchRoomFrm.
12. Lớp SearchRoomFrm gọi hàm setRoom() của lớp AddBookingFrm và tự đóng form lại.
13. Lớp AddBookingFrm cập nhật thông tin phòng đã chọn và yêu cầu nhân viên chọn nhập thông tin khách hàng.
14. Nhân viên click vào nút tìm kiếm khách hàng trên form AddBookingFrm.
15. Lớp AddBookingFrm gọi lớp form AddClientFrm hiển thị.
16. Lớp AddClientFrm hiển thị yêu cầu nhân viên nhập tên khách hàng để tìm kiếm.
17. Nhân viên hỏi lại khách hàng thông tin cá nhân đầy đủ.
18. Khách hàng cung cấp đầy đủ thông tin cá nhân cho nhân viên.
19. Nhân viên gõ tên khách hàng vào và click nút tìm kiếm trên form AddClientFrm.
20. Lớp AddClientFrm gọi hàm searchClient() của lớp ClientBean.
21. Lớp ClientBean thực hiện hàm searchClient() và trả kết quả lại cho lớp AddClientFrm.
22. Lớp AddClientFrm hiện kết quả các khách hàng có tên đã nhập lên cho nhân viên lựa chọn.
23. Vì khác hàng chưa có trong danh sách đã đặt chỗ nên nhân viên chọn nhập thông tin khách hàng mới và click vào nút Submit trên form AddClientFrm.
24. Lớp AddClientFrm gọi phương thức addClient() của lớp ClientBean.
25. Lớp ClientBean thực hiện hàm addClient() và trả về cho lớp AddClientFrm.
26. Lớp AddClientFrm gọi phương thức setClient() của lớp AddBookingFrm và tự đóng form của mình.
27. Lớp AddBookingFrm cập nhật thông tin khách hàng lên form và yêu cầu nhân viên hoàn thành các thông tin đặt chỗ còn lại.
28. Nhân viên nhập các thông tin đặt chỗ còn lại và click vào Submit.

29. Lớp AddBookingFrm gọi phương thức addBooking() của lớp BookingBean.
30. Lớp BookingBean thực hiện hàm addBooking() và trả về cho lớp AddBookingFrm.
31. Lớp AddBookingFrm thông báo đặt chỗ thành công cho nhân viên.
32. Nhân viên xác nhận lại cho khách hàng thông tin đặt chỗ thành công và kết thúc giao dịch.

+ Sơ đồ tuần tự


Client
Receptionist
AddBookingFrm
BookingBean
SearchRoomFrm
RoomBean
AddClientFrm
ClientBean
1: phone call
2: click search room
2.1: call new()
3: request to enter period
4: enter startDate,endDate
4.1: call searchClient()
4.1.1: searchRoom(startDate,endDate)
6: display client list
7: ask to choose
8: choose a room
9: click choose a room
9.1: call setRoom(Room)
10: display
11: click search client
12: call new()
14: ask full infos



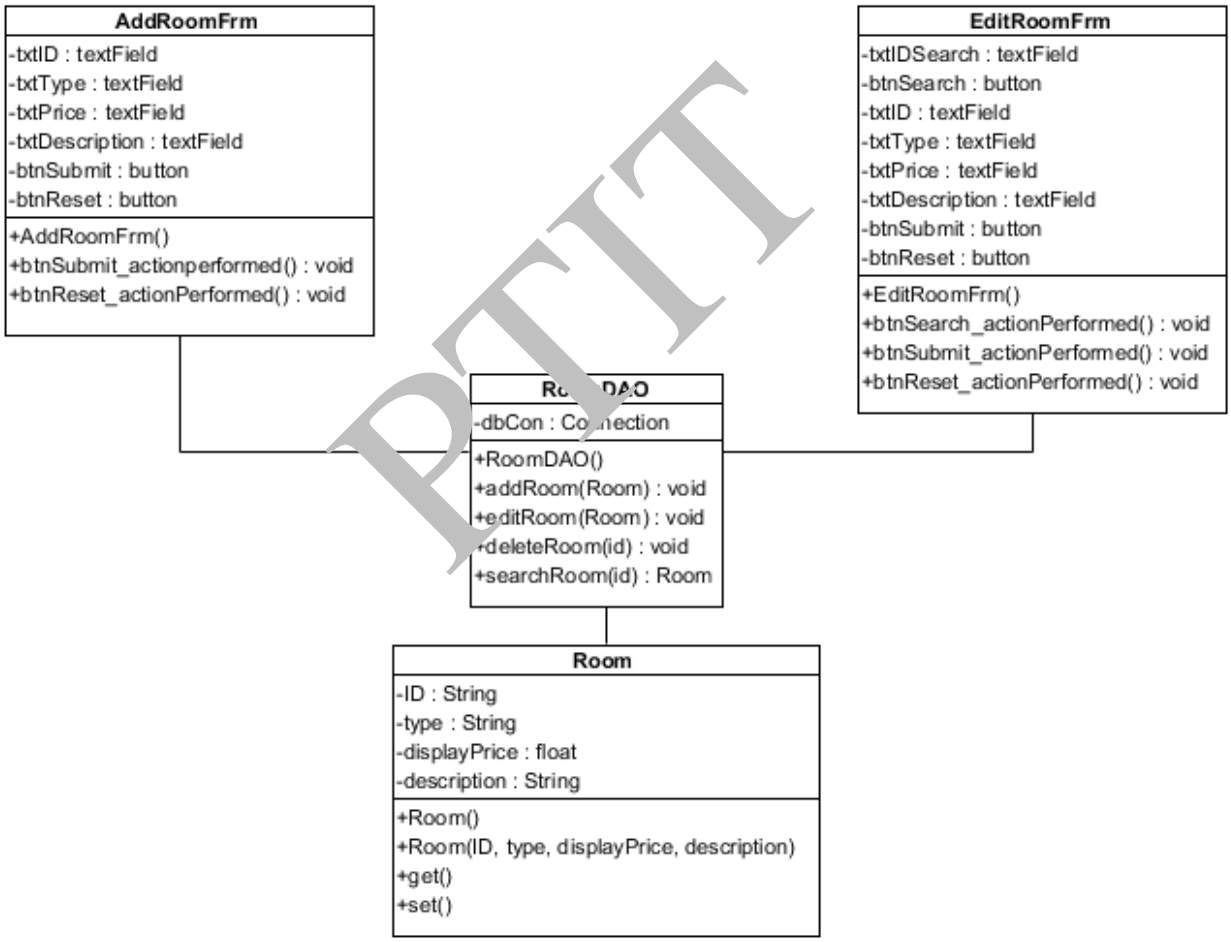
Hình 9.14: Sơ đồ tuần tự chức năng đặt phòng, thiết kế theo cách dùng bean

9.5.3 Thiết kế dùng control DAO và thực thể thuần

Tư tưởng chủ đạo của phương pháp này là tách bạch thông tin và hành động của các lớp thực thể thành 2 lớp riêng biệt: lớp chỉ chứa thông tin được gọi là các lớp thực thể thuần, lớp chỉ chứa hành động (phương thức) được gọi là lớp điều khiển truy cập dữ liệu DAO (Data Access Object). Khi có sự kiện trên form, lớp giao diện sẽ gọi hàm actionPerformed(), hàm này sẽ tạo ra một đối tượng lớp thực thể thuần để truyền vào khi gọi phương thức tương ứng của lớp điều khiển để truy cập CSDL.

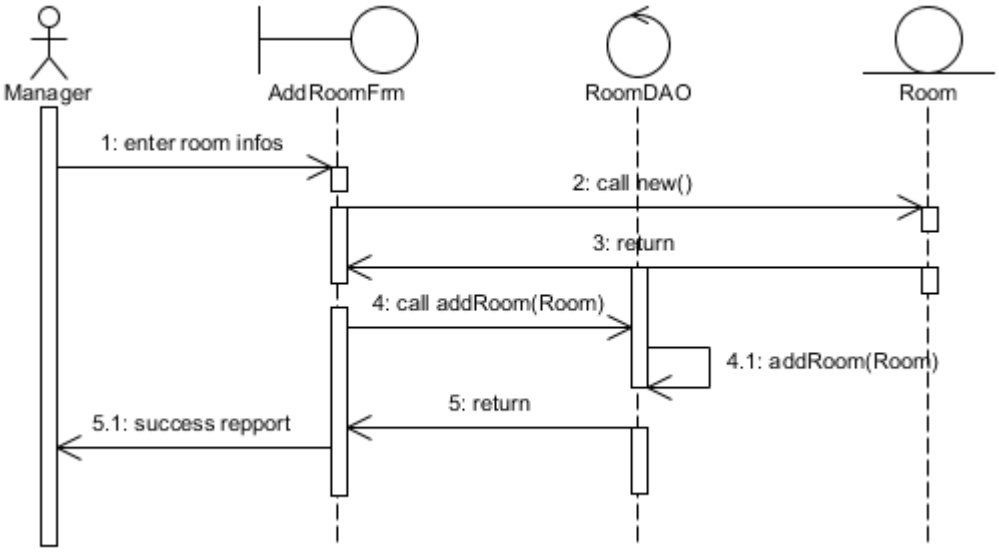
a. Thiết kế cho chức năng thêm/sửa phòng

+ Sơ đồ lớp chi tiết cho modul

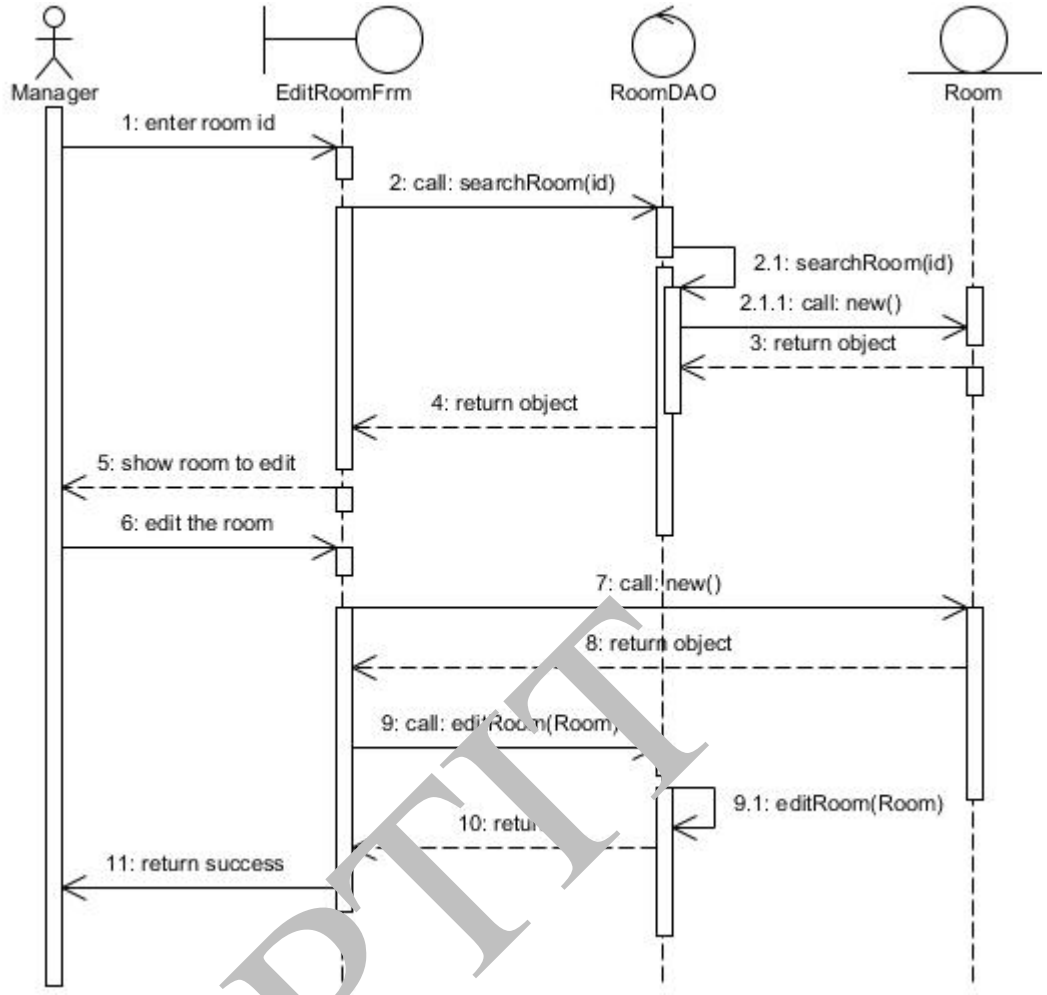


Hình 9.15: Sơ đồ lớp cho modul thêm/sửa thông tin phòng, thiết kế dùng DAO và thực thể thuần

+ Sơ đồ tuần tự



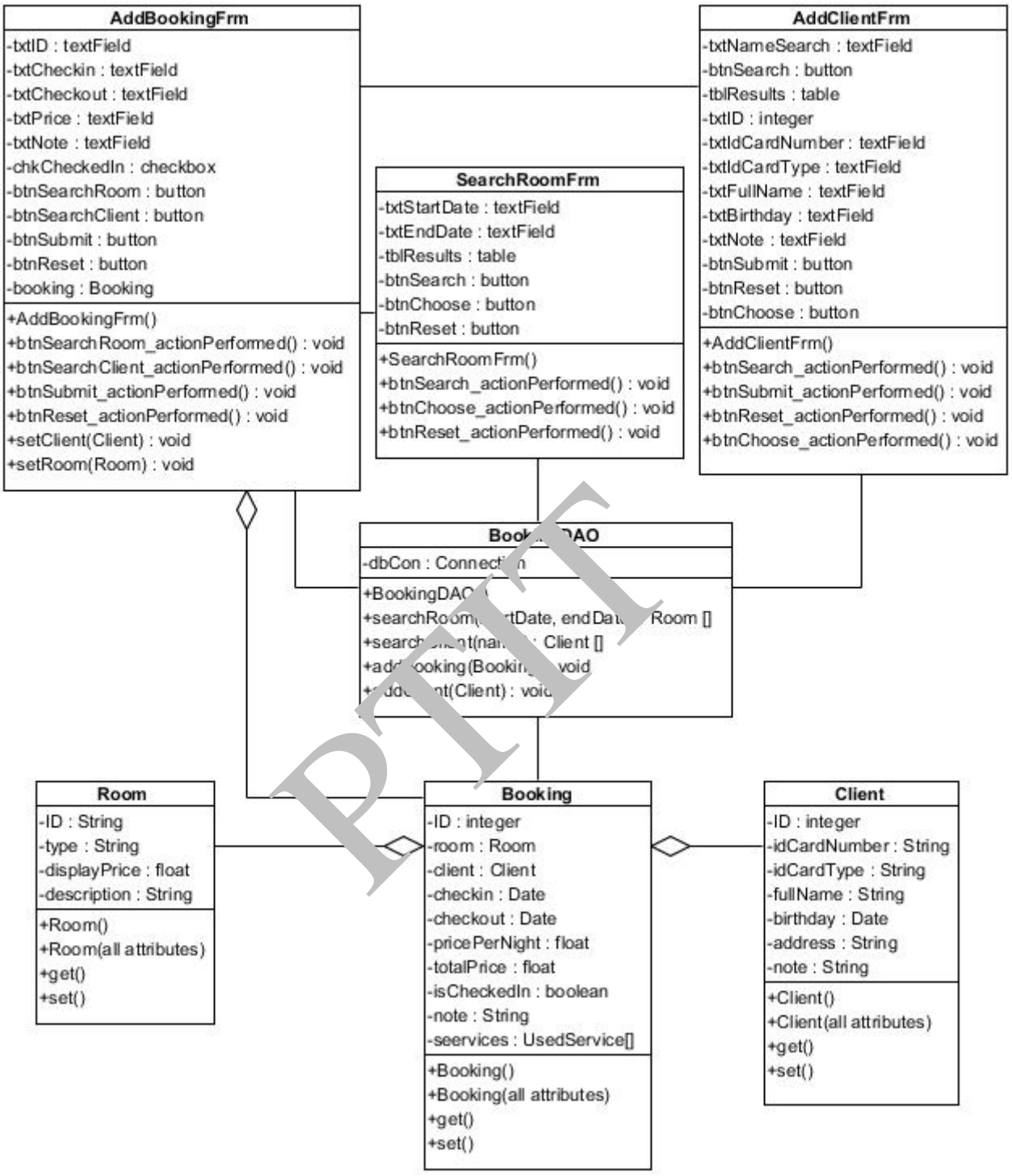
Hình 9.16: Sơ đồ tuần tự cho chức năng thêm thông tin phòng, thiết kế dùng DAO và thực thể thuần



Hình 9.17: Sơ đồ tuần tự cho chức năng sửa thông tin phòng, thiết kế dùng DAO và thực thể thuần

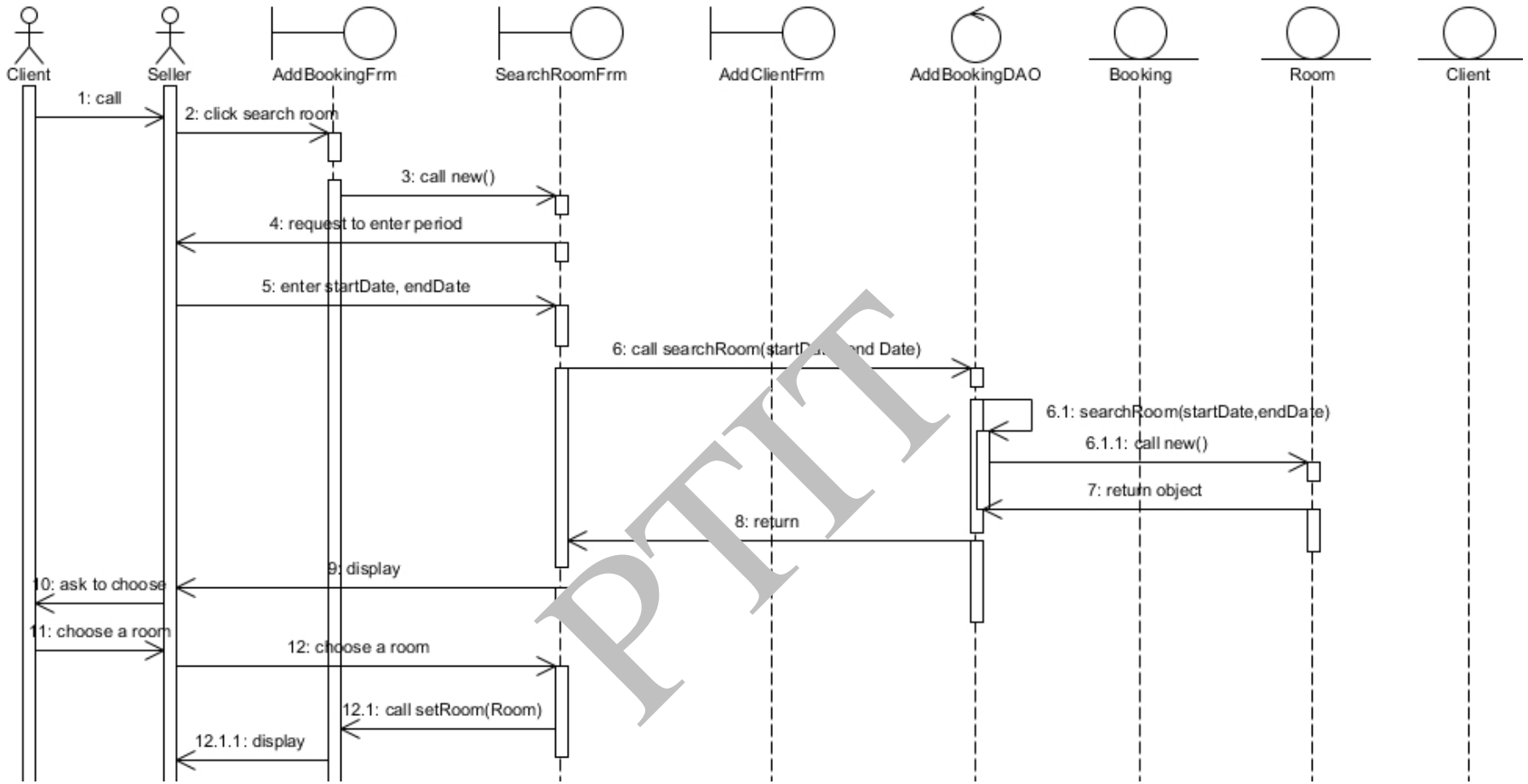
b. Thiết kế cho chức năng đặt phòng

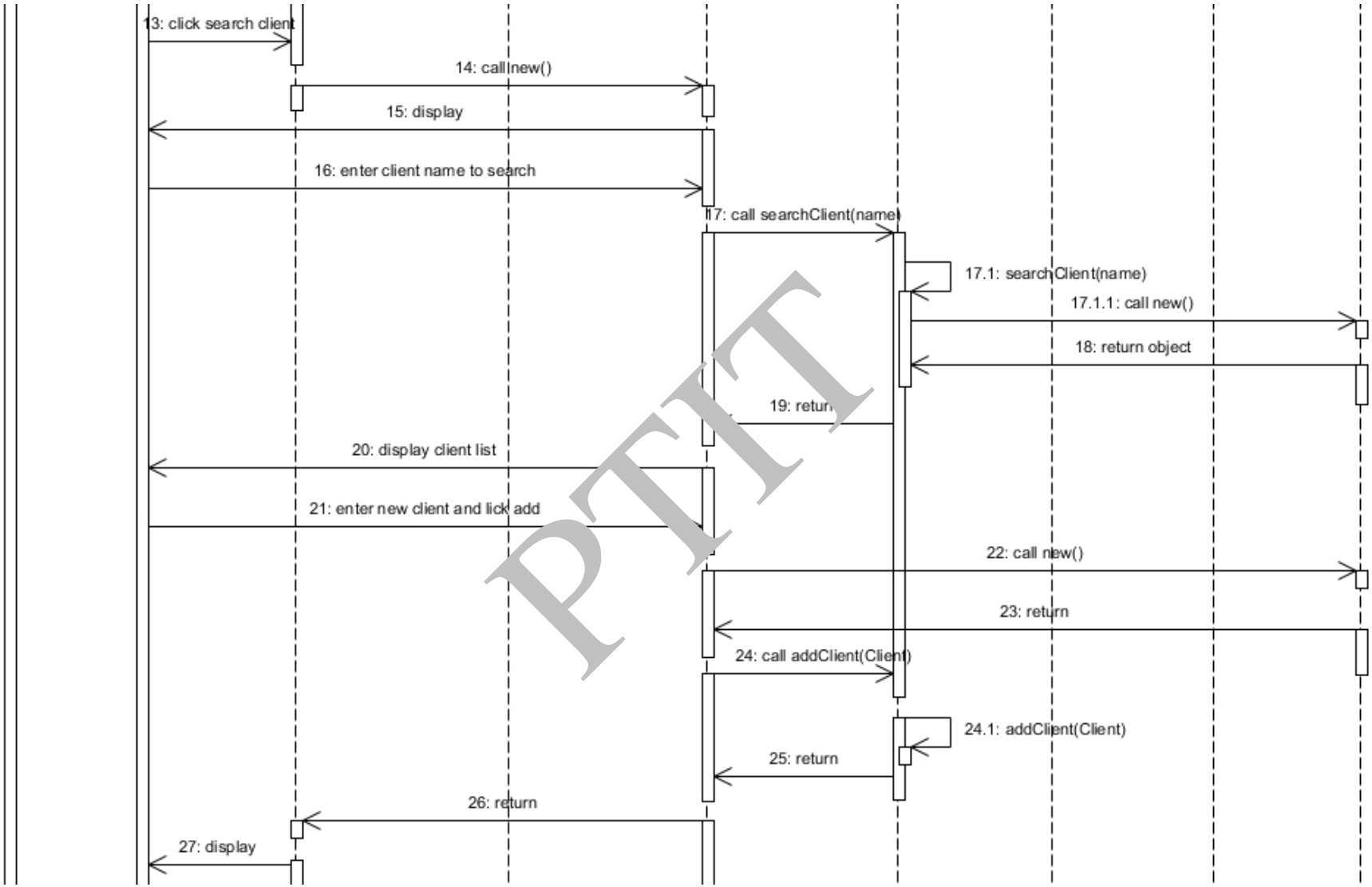
+ Sơ đồ lớp chi tiết cho modul

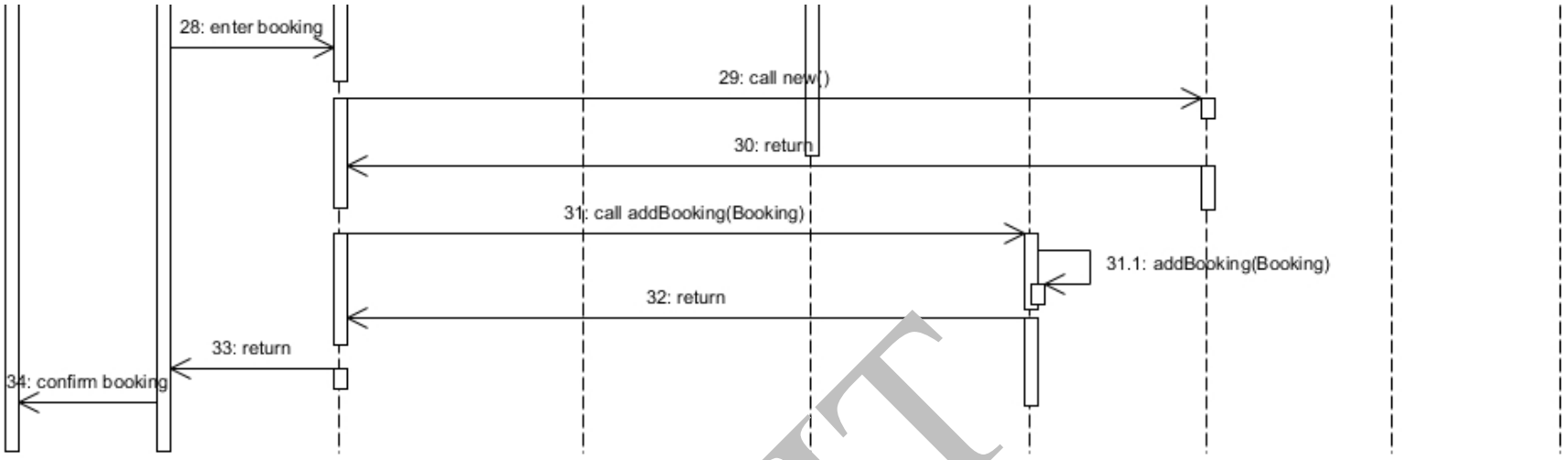


Hình 9.18: Sơ đồ lớp chức năng đặt phòng, thiết kế dùng DAO và thực thể thuần

+ Sơ đồ tuần tự


Client
Seller
AddBookingFrm
SearchRoomFrm
AddClientFrm
AddBookingDAO
Booking
Room
Client
1: call
2: click search room
3: call new()
4: request to enter period
5: enter startDate, endDate
6: call searchRoom(startDate,endDate)
6.1: searchRoom(startDate,endDate)
6.1.1: call new()
7: return object
8: return
9: display
10: ask to choose
11: choose a room
12: choose a room
12.1: call setRoom(Room)
12.1.1: display


13: click search client
14: callInew()
15: display
16: enter client name to search
17: call searchClient(name)
17.1: searchClient(name)
17.1.1: call new()
18: return object
19: retur
20: display client list
21: enter new client and lick add
22: call new()
23: return
24: call addClient(Client)
24.1: addClient(Client)
25: return
26: return
27: display



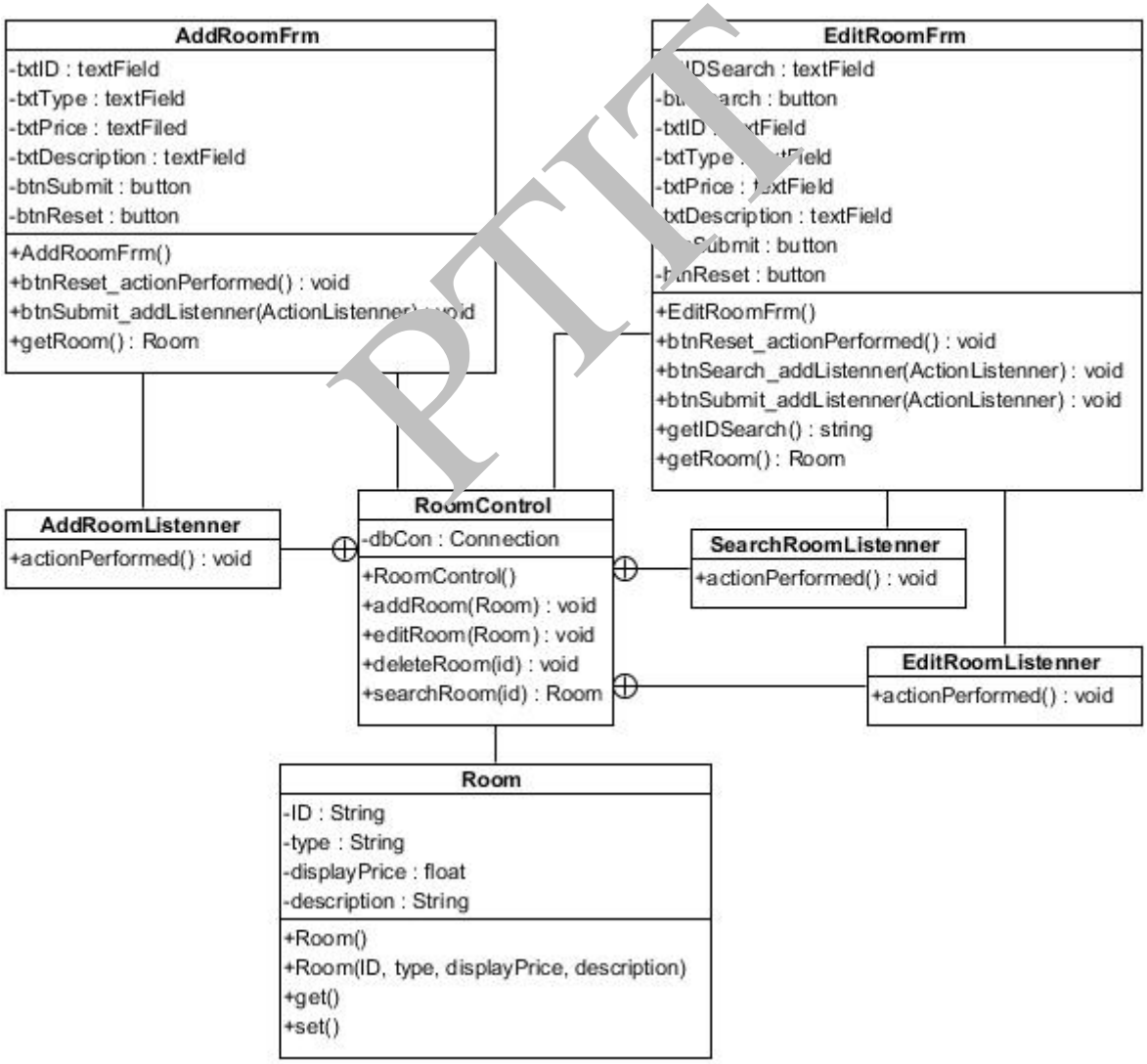
Hình 9.19: Sơ đồ tuần tự chức năng đặt phòng, thiết kế dùng DAO và thực thể thuần

9.5.4 Thiết kế theo MVC cải tiến, dùng control DAO và thực thể thuần

Tư tưởng của phương pháp này tương tự như phương pháp dùng control DAO và thực thể thuần. Nhưng trong mô hình dùng control DAO và thực thể thuần, lớp view gọi lớp control trong hàm actionPerformed() để xử lí các sự kiện. Trong mmo hình có tuân thủ MVC cải tiến thì lớp view không có quyền gọi contrrol, mà chỉ có control mới có quyền gọi và điều khiển các hàm của view. Do đó, lớp điều khiển sẽ cần đến các lớp nội tại để xử lí sự kiện thay cho lớp view: Khi có sự kiện trên form, lớp giao diện sẽ không gọi hàm actionPerformed() mà truyền sự kiện này cho lớp control xử lí, lớp control sẽ gọi hàm actionPerformed() của lớp nội tại của nó để xử lí, trong hàm này sẽ gọi các phương thức truy nhập CSDL của lớp control.

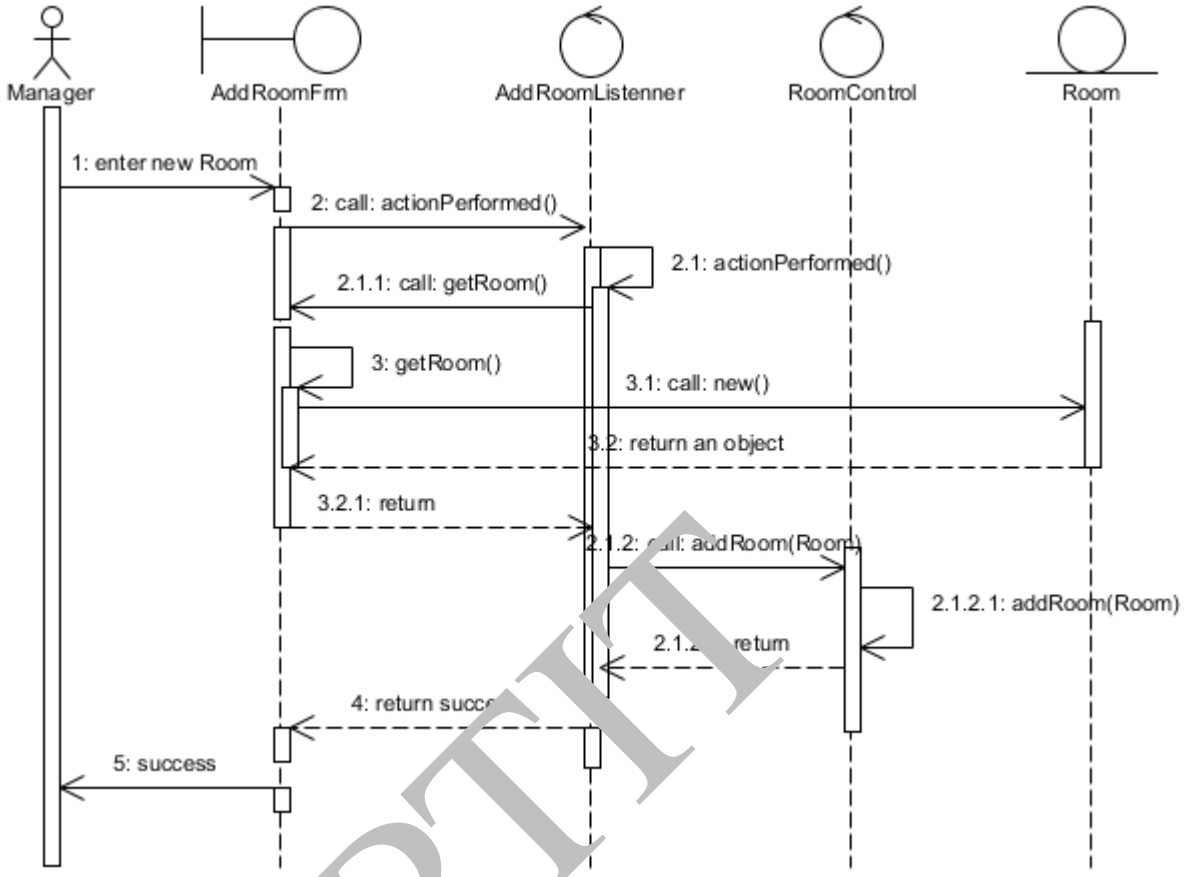
a. Thiết kế cho chức năng thêm/sửa phòng

+ Sơ đồ lớp chi tiết cho modul

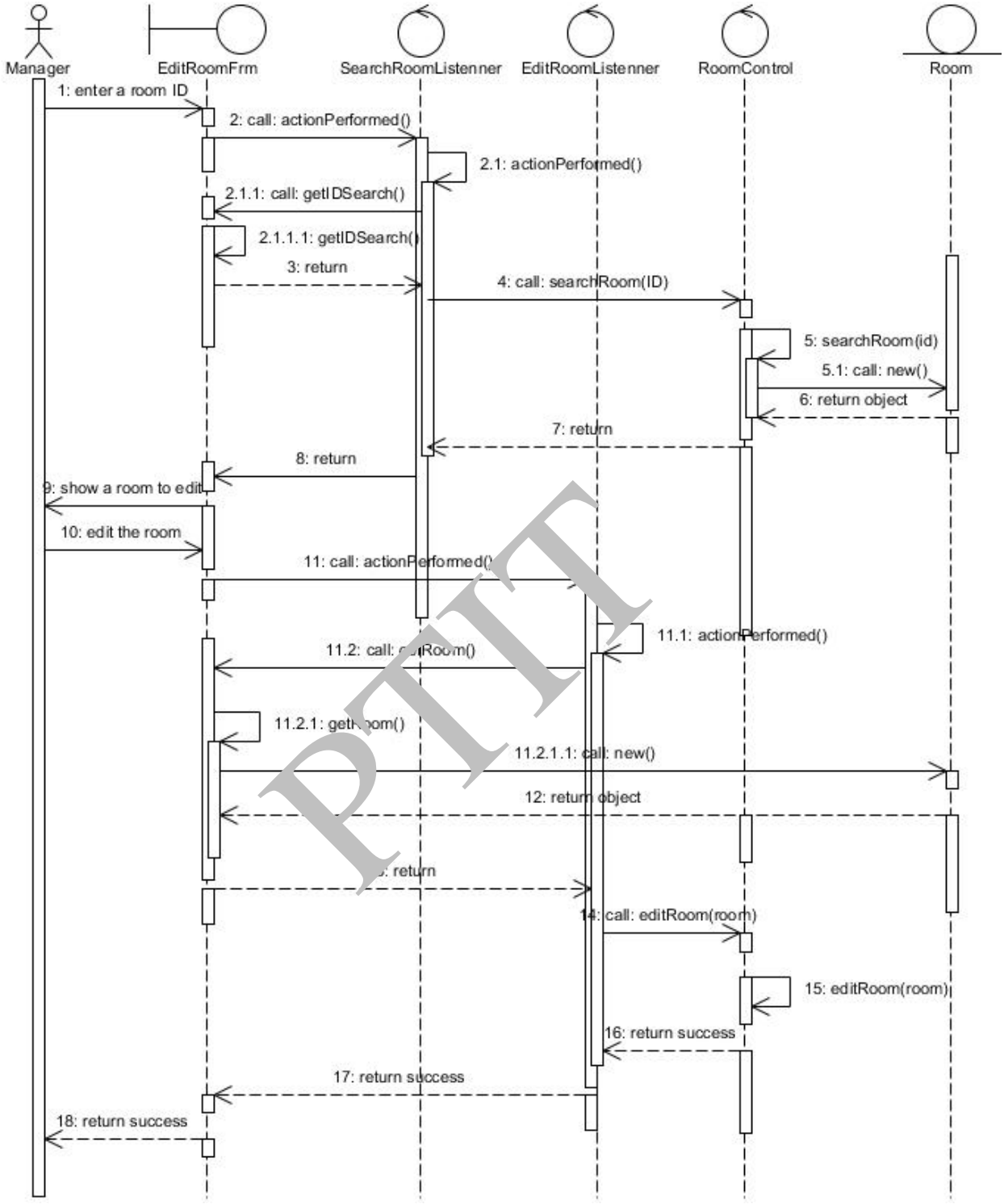


Hình 9.20: Sơ đồ lớp chức năng thêm/sửa thông tin phòng, thiết kế theo mô hình MVC cải tiến

+ Sơ đồ tuần tự



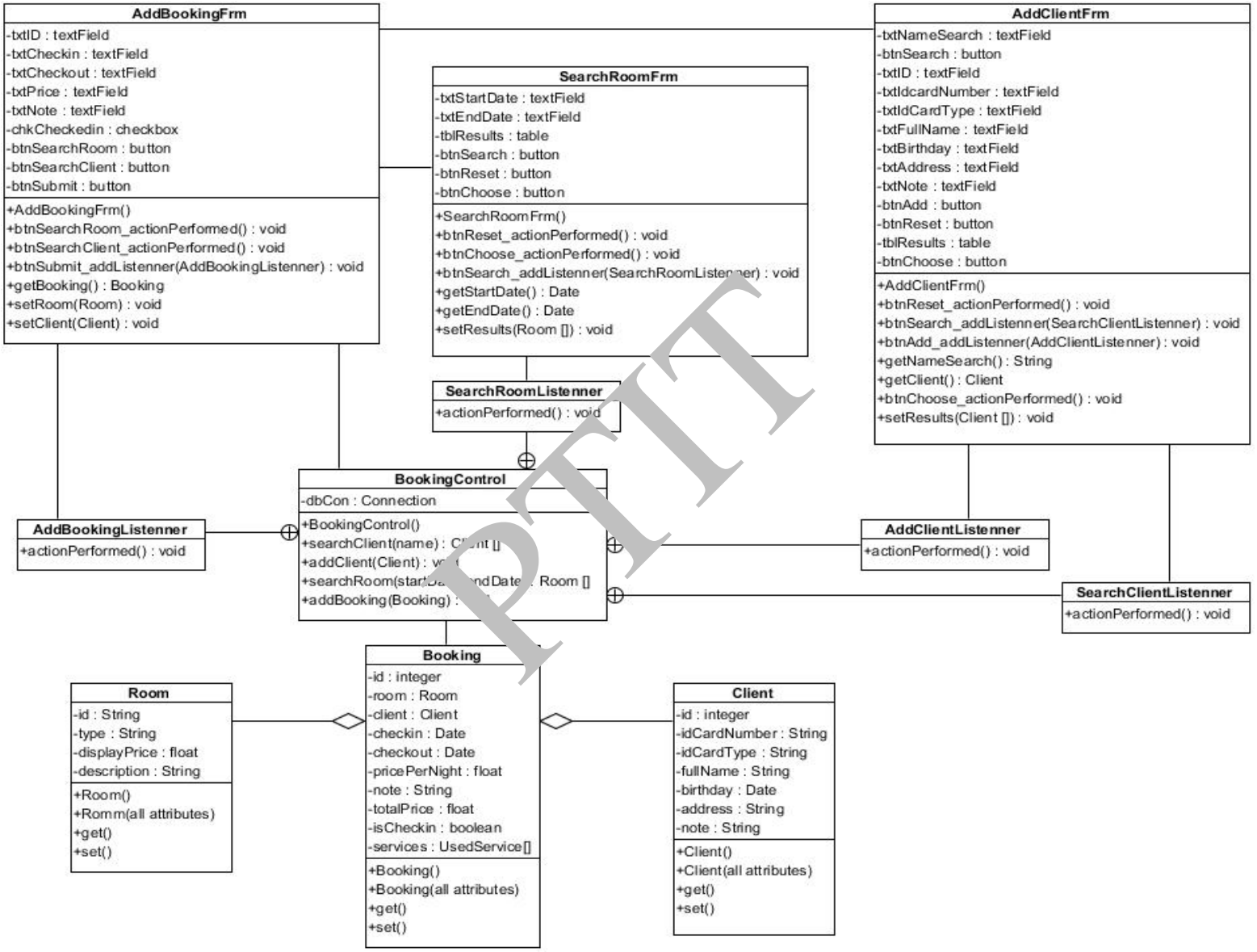
Hình 9.21: Sơ đồ tuần tự chức năng thêm thông tin phòng, thiết kế theo mô hình MVC cải tiến



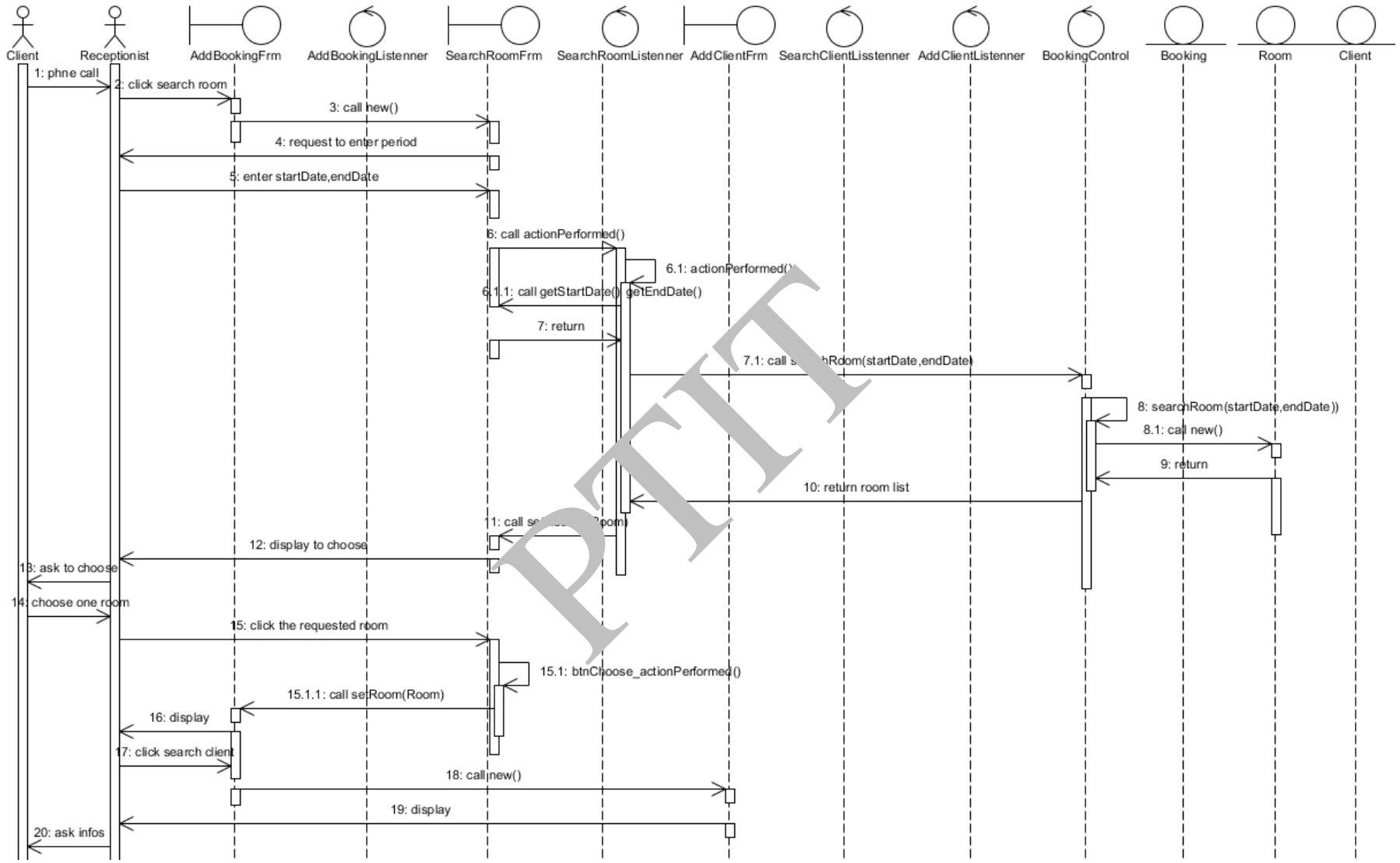
Hình 9.22: Sơ đồ tuần tự chức năng sửa thông tin phòng, thiết kế theo mô hình MVC cải tiến

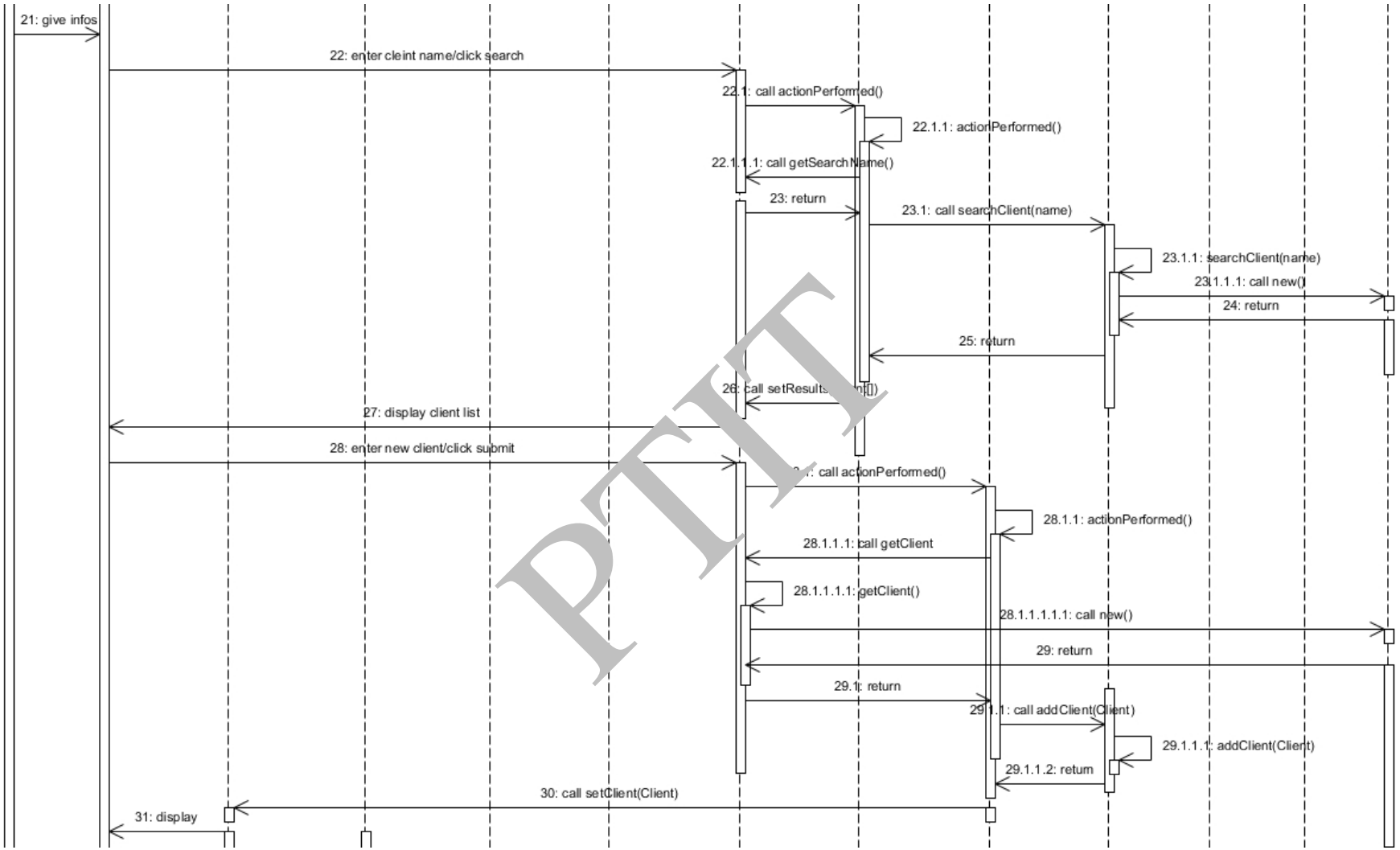
b. Thiết kế cho chức năng đặt phòng

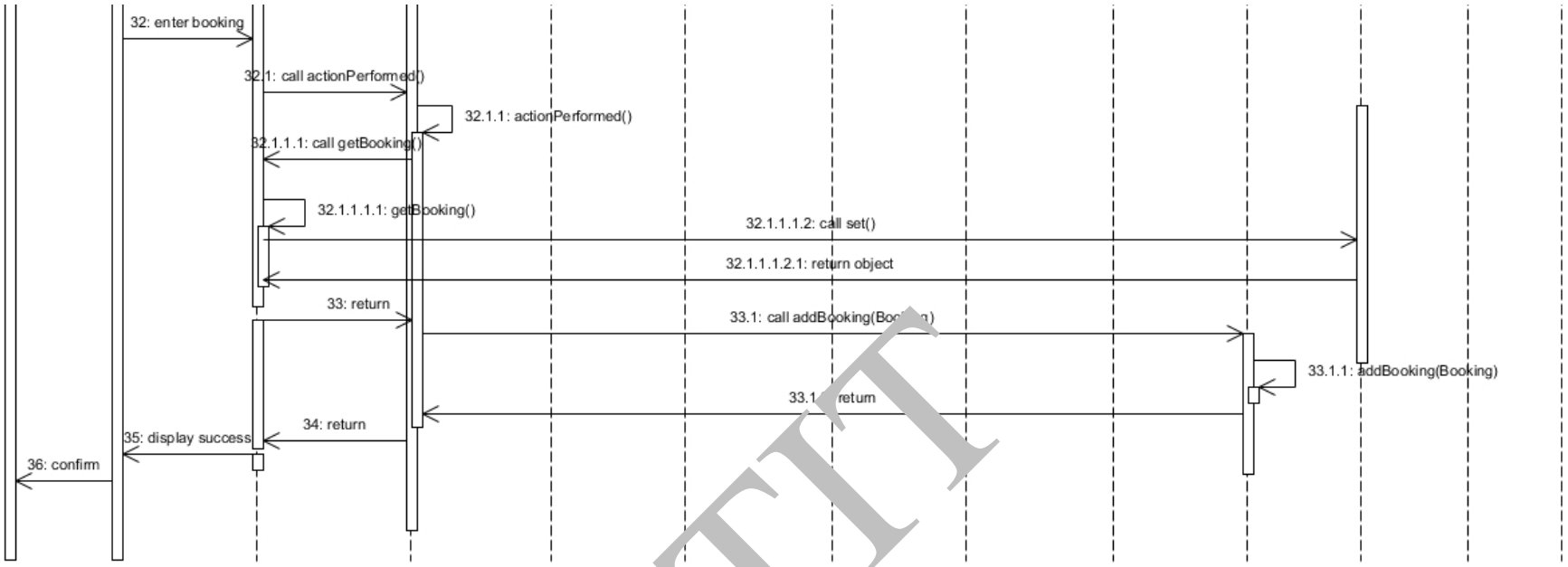
+ Sơ đồ lớp chi tiết cho modul



Hình 9.23: Sơ đồ lớp cho chức năng đặt phòng, thiết kế theo mô hình MVC cải tiến


Client
Receptionist
AddBookingFrm
AddBookingListenner
SearchRoomFrm
SearchRoomListenner
AddClientFrm
SearchClientLisstenner
AddClientListenner
BookingControl
Booking
Room
Client
1: phne call
2: click search room
3: call new()
4: request to enter period
5: enter startDate,endDate
6: call actionPerformed()
6.1: actionPerformed()
6.1.1: call getStartDate() getEndDate()
7: return
7.1: call searchRoom(startDate,endDate)
8: searchRoom(startDate,endDate))
8.1: call new()
9: return
10: return room list
11: call
12: display to choose
13: ask to choose
14: choose one room
15: click the requested room
15.1: btnChoose_actionPerformed()
15.1.1: call setRoom(Room)
16: display
17: click search client
18: call new()
19: display
20: ask infos


21: give infos
22: enter cleint name/click search
22.1: call actionPerformed()
22.1.1: actionPerformed()
22.1.1.1: call getSearchName()
23: return
23.1: call searchClient(name)
23.1.1: searchClient(name)
23.1.1.1: call new()
24: return
25: return
26: call setResults
27: display client list
28: enter new client/click submit
28.1: call actionPerformed()
28.1.1: actionPerformed()
28.1.1.1: call getClient
28.1.1.1.1: getClient()
28.1.1.1.1.1: call new()
29: return
29.1: return
29.1.1: call addClient(Client)
29.1.1.1: addClient(Client)
29.1.1.2: return
30: call setClient(Client)
31: display



Hình 9.24: Sơ đồ tuần tự cho chức năng đặt phòng, thiết kế theo mô hình MVC cải tiến

CHƯƠNG 10: PHA CÀI ĐẶT VÀ TÍCH HỢP

10.1 CÁC PHƯƠNG PHÁP CÀI ĐẶT VÀ TÍCH HỢP

10.1.1 Luồng công việc cài đặt

- Mục đích của luồng công việc cài đặt là cài đặt phần mềm đích
- Phần mềm lớn được chia thành các hệ thống con
    - Được cài đặt song song bởi các đội viết mã
- Các hệ thống con bao gồm các thành phần và các mô đun mã
- Một khi người lập trình đã cài đặt một mô đun, người ấy thực hiện kiểm thử đơn vị mô đun đó
- Sau đó mô đun được chuyển qua nhóm SQA để kiểm thử mức cao hơn
    - Kiểm thử này là một phần của luồng công việc kiểm thử

10.1.1.1 Chọn ngôn ngữ lập trình

- Ngôn ngữ thường được chỉ rõ trong hợp đồng
- Nhưng điều gì sẽ xảy ra khi hợp đồng chỉ ra rằng:
    - Sản phẩm phần mềm được cài đặt bằng ngôn ngữ lập trình "phù hợp nhất"
- Ngôn ngữ nào nên được chọn?
- Ví dụ:
    - Tổ chức QQQ đã viết bằng ngôn ngữ COBOL suốt 25 năm qua
    - Trên 200 nhân viên phần mềm, tất cả để là chuyên gia về COBOL
    - Ngôn ngữ lập trình nào là phù hợp nhất?
- Hiển nhiên COBOL
- Chuyện gì xảy ra khi ngôn ngữ lập trình mới (như C++) được giới thiệu:
    - Những chuyên gia C++ phải thuê
    - Những chuyên gia COBOL vốn có phải đào tạo lại
    - Những sản phẩm trong tương lại được viết bằng C++
    - Những sản phẩm phần mềm COBOL sẵn có phải được bảo trì
    - Có hai kiểu người lập trình khác nhau
        - Những người bảo trì COBOL (bị coi nhẹ)
        - Những người lập trình C++ (được trả nhiều tiền hơn)
    - Yêu cầu phần mềm và phần cứng đắt tiền để chạy ngôn ngữ lập trình
    - Hàng trăm chuyên gia COBOL bị bỏ phí
- Kết luận duy nhất là:
    - COBOL là ngôn ngữ lập trình phù hợp nhất
- Và ngôn ngữ phù hợp nhất cho dự án mới nhất có thể là C++
    - COBOL phù hợp với những ứng dụng chỉ xử lý dữ liệu

- Cách chọn ngôn ngữ lập trình
  - Phân tích lợi nhuận – chi phí
  - Tính toán chi phí và lợi nhật của tất cả các ngôn ngữ liên quan
- Ngôn ngữ hướng đối tượng nào thích hợp nhất?
  - C++ giống C (C++ is (unfortunately) C-like)
  - Do đó, mỗi chương trình C cổ điển tự động là chương trình C++
  - Java được yêu cầu đối với mô hình hướng đối tượng
  - Việc đào tạo trong mô hình hướng đối tượng là cần thiết trước khi áp dụng bất cứ ngôn ngữ hướng đối tượng nào
- Còn việc lựa chọn ngôn ngữ thế hệ thức tư? (Fourth generation language -4GL)?

10.1.1.2 Ngôn ngữ thế hệ thứ tư

- Ngôn ngữ thế hệ thứ nhất
  - Ngôn ngữ máy
- Ngôn ngữ thế hệ thứ hai
  - Hợp ngữ
- Ngôn ngữ thế hệ thứ ba
  - Ngôn ngữ bậc cao (COBOL, FORTRAN, C++, Java)
- Ngôn ngữ thế hệ thứ tư (4GLs)
  - Một câu lệnh ngôn ngữ thế hệ thứ ba tương đương với 5 đến 10 câu lệnh hợp ngữ
  - Mỗi câu lệnh ngôn ngữ thứ tư tương đương với 30 hoặc 50 câu lệnh hợp ngữ
- Hy vọng rằng ngôn ngữ thứ tư sẽ:
  - Tăng nhanh tốc độ xây dựng ứng dụng
  - Kết quả của các ứng dụng là dễ dàng xây dựng và nhanh chóng thay đổi
    - Giảm chi phí bảo trì
  - Đơn giản trong việc gỡ lỗi
  - Tạo ngôn ngữ thân thiện người dùng
- Có thể thực hiện được nếu ngôn ngữ thứ tư là ngôn ngữ bậc cao, thân thiện với người dùng
- Những đóng góp vào thị trường:
  - Không có một ngôn ngữt thế hệ thứ 4 nào chiếm ưu thế trong thị trường phần mềm
  - Có hàng trăm 4GL
  - Hàng tá nhóm người dùng cỡ lớn
  - Oracle, DB2, và PowerBuilder cực kỳ phổ biến
- Lý do
  - Không có một 4GL có đủ những đặc trưng cần thiết
- Kết luận
  - Đặc biệt quan tâm đến việc lựa chọn 4GL thích hợp

Tăng hiệu năng với 4GL

- Bức tranh không phải toàn màu hồng
- Playtex used ADF, obtained an 80 to 1 productivity increase over COBOL
    - However, Playtex then used COBOL for later applications
- 4GL productivity increases of 10 to 1 over COBOL have been reported
    - Tuy nhiên, có quá nhiều bản tường trình của những thử nghiệm tồi

Những thử nghiệm thực tế với 4GL

- Nhiều ngôn ngữ thế hệ thứ tư được hỗ trợ bởi môi trường CASE mạnh mẽ
    - Đây là một vấn đề đối với những tổ chức ở mức CMM 1 hoặc 2
    - Một vài thất bại của ngôn ngữ thế thế thứ 4 được ghi lại là do môi trường CASE vật lý
- Quan điểm của 43 tổ chức đối với 4GLs
    - Việc sử dụng 4GL đã giảm sự thất vọng của người dùng
    - Đáp ứng nhanh hơn từ bộ phận DP (Quicker response from DP department)
    - 4GLs are slow and inefficient, on average
    - Nhìn chung, 28 tổ chức sử dụng 4GL trong 3 năm thấy lợi nhuận thu được vượt quá chi phí bỏ ra

Những nguy hiểm với ngôn ngữ thứ tư

- End-user programming
    - Những người lập trình được đào tạo để nghi ngờ các đầu ra của máy tính (Programmers are taught to mistrust computer output)
    - Những dùng cuối được dạy để tin tưởng vào đầu ra của máy tính (End users are taught to believe computer output)
    - Người dùng cuối đang cập nhật cơ sở dữ liệu có thể đặc biệt nguy hiểm (An end-user updating a database can be particularly dangerous)
- Những nguy hiểm tiềm năng đối với quản lý (Potential pitfalls for management)
    - Trường hợp giới thiệu sớm môi trường CASE (Premature introduction of a CASE environment)
    - Đào tạo không đủ đối với độ phát triển (Providing insufficient training for the development team)
    - Chọn 4GL sai

10.1.1.3 Lập trình tốt trong thực tế (Good Programming Practice)

- Sử dụng tên biến nhất quán và có ý nghĩa
    - "Có ý nghĩa" để những người lập trình bảo trì trong tương lai
    - "Nhất quán" để trợ giúp cho những người bảo trì trong tương lai
    - Tài liệu mã bao gồm tên các biến như freqAverage, frequencyMaximum, minFr, frqncyTotl
    - Người lập trình bảo trì phải biết nếu freq, frequency, fr, frqncy đều liên quan đến cùng một thứ

        - Nếu sử dụng từ đồng nhất, tốt nhất là frequency, có thể freq hoặc frqncy, không thể fr
        - Nếu không sử dụng một từ khác (như: rate) cho một số lượng khác
    - Chúng ta có thể sử dụng frequencyAverage, frequencyMaximum, frequencyMinimum, frequencyTotal
    - Chúng ta cũng có thể sử dụng averageFrequency, maximumFrequency, minimumFrequency, totalFrequency
    - Nhưng bốn tên phải xuất phát cùng một tập
- Vấn đề của Self-Documenting Code
    - Self-documenting code cực kỳ hiếm
    - Vấn đề chính: tài liệu mã có thể được hiểu một cách dễ dàng và không nhập nhằng bởi:
        - Đội SQA
        - Những người lập trình bảo trì
        - Những người khác đọc mã
    - Ví dụ:
        - Tài liệu mã bao gồm biến xCoordinateOfPositionOfRobotArm
        - Biến này được viết tắt là xCoord
        - Biến này rất tốt, bởi vì toàn bộ mô đun xử lý sự di chuyển của cánh tay rô bốt
        - Nhưng người lập trình bảo trì có biết điều này không?
    - Những lời giải thích mở đầu tối thiểu cho một tài liệu viết mã artifact

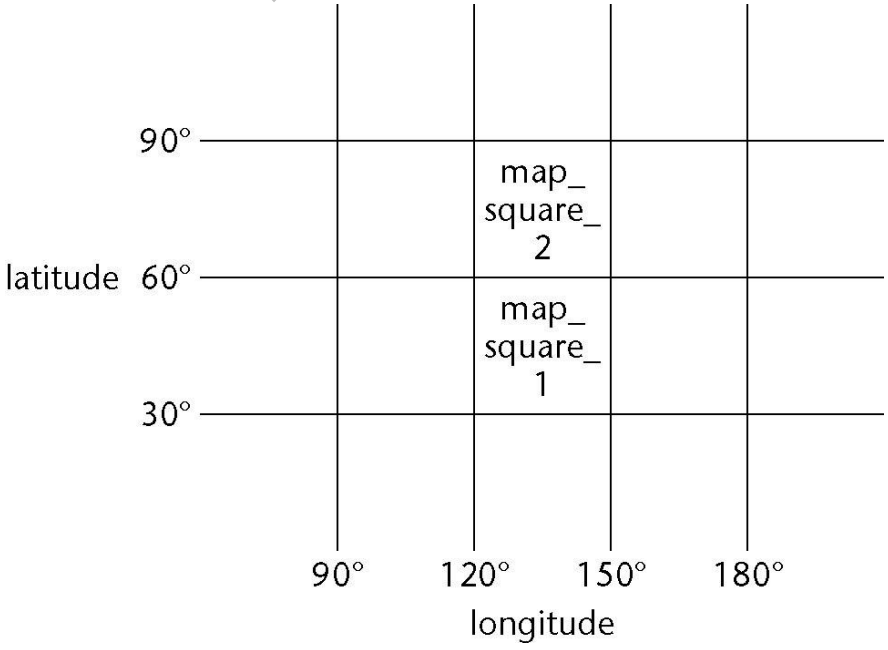
> The name of the code artifact
> A brief description of what the code artifact does
> The programmer's name
> The date the code artifact was coded
> The date the code artifact was approved
> The name of the person who approved the code artifact
> The arguments of the code artifact
> A list of the name of each variable of the code artifact, preferably in alphabetical order, and a brief description of its use
> The names of any files accessed by this code artifact
> The names of any files changed by this code artifact
> Input–output, if any
> Error-handling capabilities
> The name of the file containing test data (to be used later for regression testing)
> A list of each modification made to the code artifact, the date the modification was made, and who approved the modification
> Any known faults

  - Những đề nghị (Suggestion): Những lời giải thích là cần thiết khi mã được viết theo cách không rõ ràng hoặc cách sử dụng một khía cạnh tinh tế nào đó của ngôn ngữ
  - Lời giải thích vô nghĩa (Nonsense): Viết lại mã theo cách rõ ràng hơn, chúng ta không bao giờ thúc đẩy/bỏ qua việc lập trình tồi. Tuy nhiên, những lời giải thích có thể trợ giúp những người lập trình bảo trì trong tương lai
  -

- Việc sử dụng các tham số
  - Gần như không có những hằng số thực
  - Một giải pháp:
    - Sử dụng câu lệnh **const** (C++), hoặc
    - Sử dụng câu lệnh **public static final** (Java)
  - Một cách tốt hơn:
    - Đọc những giá trị "hằng số" từ tệp tham số

- Việc bố trí mã để tăng khả năng có thể đọc được
  - Sử dụng thụt đầu dòng
  - Tốt hơn, sử dụng (Better, use a pretty-printer)
  - Sử dụng nhiều dòng trống
    - Để tách những khối lệnh lớn (To break up big blocks of code)

- Câu lệnh if lồng
  - Ví dụ: Một bản đồ bao gồm hai hình vuông. Viết mã để xác định liệu một điểm nằm trên bề mặt trái đất nằm trong map_square_1 hoặc map_square_2, hoặc không nằm trong hình vuông nào



- Giải pháp 1: Được định dạng tồi

```
if (latitude > 30 && longitude > 120) {if (latitude <= 60 && longitude <= 150)
mapSquareNo = 1; else if (latitude <= 90 && longitude <= 150) mapSquareNo = 2
else print "Not on the map";} else print "Not on the map";
```

- Định dạng tốt, được cấu trúc tồi

```
if (latitude > 30 && longitude > 120)
{
  if (latitude <= 60 && longitude <= 150)
    mapSquareNo = 1;
  else
    if (latitude <= 90 && longitude <= 150)
      mapSquareNo = 2;
    else
      print "Not on the map";
}
else
  print "Not on the map";
```

- Cấu trúc lồng được chấp nhận

```
if (longitude > 120 && longitude <= 150 && latitude > 30 && latitude <= 60)
  mapSquareNo = 1;
else
  if (longitude > 120 && longitude <= 150 && latitude > 60 && latitude <= 90)
    mapSquareNo = 2;
  else
    print "Not on the map";
```

- Sự kết hợp của câu lệnh **if-if** và **if-else-if** thường rất khó đọc
- Đơn giản: Sự kết hợp **if-if**

if <*condition1*>
if <*condition2*>

tương đương với điều kiện đơn

if <*condition1*> && <*condition2*>

- Quy luật ngón tay cái (Rule of thumb) : **Câu lệnh if** được lồng lớn hơn ba lần nên tránh vì đó là cách lập trình tồi

10.1.1.4 Những chuẩn lập trình

- Standards can be both a blessing and a curse
- Những mô đun có độ kết dính ngẫu nhiên (coincidental cohesion) sinh ra từ các luật giống như:
  - "Mỗi mô đun sẽ bao gồm 35 và 50 câu lệnh có thể thực thi được"
- Tốt hơn
  - "Những người lập trình nên hỏi ý kiến những người quản lý của họ trước khi xây dựng một mô đun với ít hơn 35 hoặc nhiều hơn 50 câu lệnh có thể thực thi được"

Nhận xét:

- Chưa từng có một chuẩn nào được chấp nhận phổ biến

- Các chuẩn đã áp đặt từ bên trên sẽ được bỏ qua
- Các chuẩn có thể kiểm tra bằng máy
- Mục đích của chuẩn là để bảo trì dễ dàng
  - Nếu các chuẩn làm cho việc phát triển gặp khó khăn, thì chúng phải được chỉnh sửa
  - Những chuẩn giới hạn quá mức phản tác dụng (Overly restrictive standards are counterproductive)
  - Chất lượng phần mềm có áp dụng các chuẩn (The quality of software suffers)

Ví dụ của chuẩn lập trình tốt

- "Việc xếp lồng vào nhau các câu lệnh **if** skhông nên vượt quá 3 lần, ngoại trừ được phê chuẩn trước từ đội trưởng"
- "Các mô đun gồm từ 35 đến 50 câu lệnh, ngoại trừ có sự phê chuẩn từ trước của đội trưởng"
- "Sử dụng câu lệnh **goto** nên tránh. Tuy nhiên, với sự phê chuẩn từ trước của đội *trưởng, lệnh* ***goto*** *có thể được sử dụng để xử lý lỗi"*

10.1.1.5 Sử dụng lại mã

- Sử dụng lại mã là một dạng sử dụng lại phổ biến nhất
- Tuy nhiên, tài liệu của tất cả các hoạt động công việc có thể được sử dụng lại

10.1.1.6 Công cụ CASE cho cài đặt

- Công cụ CASE sử dụng cho tích hợp gồm
  - Công cụ điều khiển phiên bản, công cụ điều khiển cấu hình, và công cụ xây dựng
  - Ví dụ:
    - *rcs, sccs,* PCVS, SourceSafe
- Công cụ điều khiển cấu hình
  - Mang tính thươngmại
    - PCVS, SourceSafe
  - Mã nguồn mở
    - CVS

Các công cụ CASE cho tiến tình phần mềm hoàn thiện

- Một tổ chức lớn cần một môi trường
- Một tổ chức với cỡ trung bình có thể quản lý sử dụng workbench (A medium-sized organization can probably manage with a workbench)
- Một tổ chức nhỏ có thể quản lý mà chỉ sử dụng các công cụ

Môi trường phát triển đã tích hợp

- Ý nghĩa của từ "tích hợp"
  - Tích hợp giao diện người dùng
  - Tương tự "nhìn và cảm nhận"
  - Thành công nhất trên hệ điều hành Macintosh
- Cũng có các kiểu tích hợp khác

- Tích hợp công cụ
    - Tất cả các công cụ giao tiếp sử dụng cùng một định dạng
    - Ví dụ:
        - Unix Programmer's Workbench
- Tích hợp tiến trình
    - Môi trường hỗ trợ một tiến trình riêng biệt
    - Tập con: Môi trường dựa trên kỹ thuật
        - Trước đây: "môi trường dựa trên phương thức"
        - Hỗ trợ một kỹ thuật riêng biệt hơn là một tiến trình hoàn thiện
        - Môi trường của các kỹ thuật là:Phân tích hệ thống trúc (Structured systems analysis) và Petri nets

- Môi trường dựa trên kỹ thuật
    - Đồ họa hỗ trợ cho phân tích, thiết kế
    - Từ điển dữ liệu
    - Việc vài kiểm tra tính nhất quán
    - Hỗ trợ quản lý
    - Hỗ trợ và hình thức hóa tiến trình bằng tay
    - Ví dụ:
        - Analyst/Designer
        - Software through Pictures
        - IBM Rational Rose
        - Rhapsody (for Statecharts)
    - Thuận lợi:
        - Người dùng ép buộc phải sử dụng một phương pháp cụ thể, chính xác
    - Bất lợi:
        - Người dùng bị ép buộc sử dụng một phương thức cụ thể, vì thế phương thức đó phải là một phần của tiến trình phần mềm của tổ chức đó (The user is forced to use one specific method, so that the method must be part of the software process of that organization)

Môi trường của các ứng dụng doanh nghiệp

- Nhấn mạnh tính dễ dàng khi sử dụng bao gồm
    - Bộ sinh giao diện người dùng thân thiện
    - Chuẩn màn hình cho đầu vào và đầu ra, và
    - Bộ sinh mã
        - Thiết kế chi tiết là mức thấp nhất của trừu tượng
        - Thiết kế chi tiết là đầu vào của bộ sinh mã
- Việc sử dụng ngôn ngữ lập trình này làm tăng hiệu năng

- Ví dụ: Oracle Development Suite
- PCTE — Portable common tool environment
    - *Không phải là một môi trường*
    - Là một cơ sở hạ tầng để trợ giúp công cụ CASE (tương tự với cách hệ điều hành cung cấp các dịch vụ cho các sản phẩm phần mềm người dùng)
    - Được chấp nhận bởi ECMA (European Computer Manufacturers Association)
- Ví dụ sự cài đặt:
    - IBM, Emeraude

Những vấn đề xảy ra với môi trường

- Không có môi trường lý tưởng cho tất cả các tổ chức
    - Mỗi môi trường có điểm mạnh điểm yếu
- Cảnh báo 1
    - Chọn môi trường sai có thể tồi hơn khi không có môi trường
    - Ép buộc kỹ thuật sai là phản tác dụng
- Cảnh báo 2
    - Những môi trường Shun CASE dưới mức 3 CMM
    - Không thể tự động hóa một tiến trình không tồn tại
    - Tuy nhiên, công cụ CASE hoặc workbench CASE là rất tốt
- Năm thước đo cơ bản cùng với
    - Thước đo độ phức tạp
- Thống kê lỗi là quan trọng
    - Số lượng trường hợp kiểm thử
    - Phần trăm trường hợp kiểm thử sinh ra lỗi
    - Tổng số lượng lỗi
- Dữ liệu lỗi được hợp nhất với danh sách kiểm tra (checklist) đối với quá trình kiểm tra kỹ lưỡng mã

10.1.1.7 Thước đo của luồng công việc cài đặt

- Năm thước đo cơ bản cùng với
    - Thước đo độ phức tạp
- Thống kê lỗi là quan trọng
    - Số lượng trường hợp kiểm thử
    - Phần trăm trường hợp kiểm thử sinh ra lỗi
    - Tổng số lượng lỗi
- Dữ liệu lỗi được hợp nhất với danh sách kiểm tra (checklist) đối với quá trình kiểm tra kỹ lưỡng mã

10.1.1.8 Những thách thức của luồng công việc cài đặt

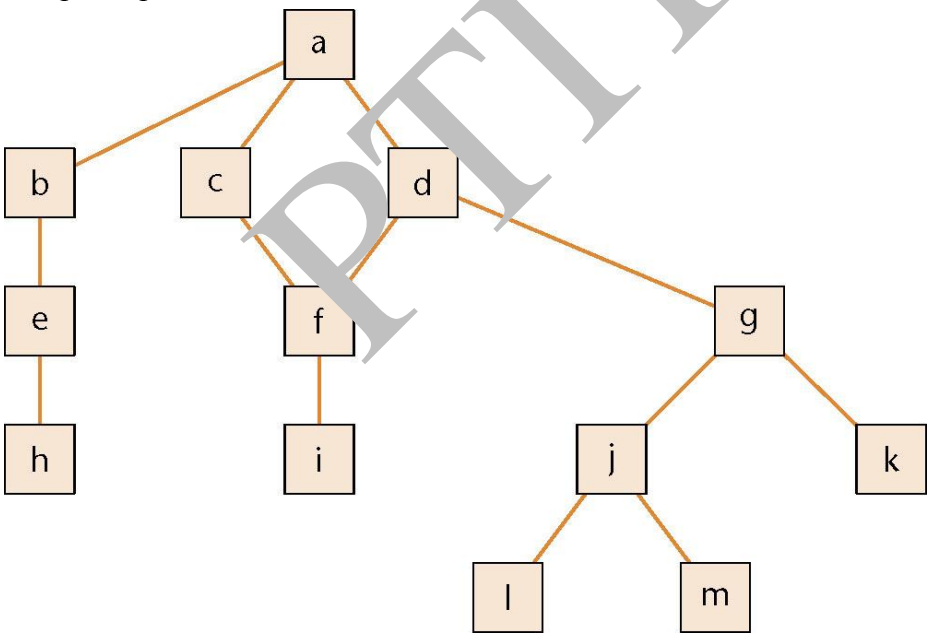
- Những vấn đề quản lý có ý nghĩa lớn ở đây
    - Các công cụ CASE thích hợp

- Lập kế hoạch kiểm thử
- Truyền đạt những thay đổi tới tất cả nhân viên
- Quyết định khi nào dừng kiểm thử

- Sử dụng lại mã cần được đưa vào phần mềm từ lúc băt đầu
  - Sử dụng lại phải là yêu cầu của khách hàng
  - Kế hoạch quản lý dự án phần mềm phải hợp nhất với việc sử dụng lại
- Cài đặt dễ hiểu về mặt kỹ thuật
  - Những thử thách trong việc quản lý

10.1.2 Tích hợp

- Cho đến tận bây giờ phương pháp phổ biến là:
  - Tích hợp theo sau tích hợp
- Đây là phương pháp tồi
- Tốt hơn:
  - Kết hợp giữa cài đặt và tích
- Sản phẩm phần mềm với 13 mô đun



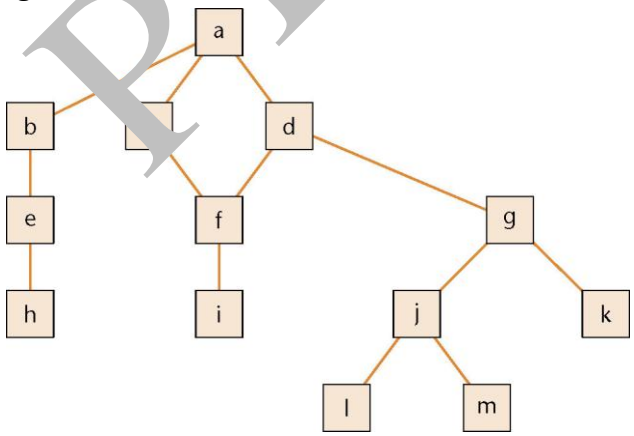
- Cài đặt sau đó tích hợp
  - Viết mã và kiểm thử tài liệu viết mã là tách biệt
  - Liên kết 13 tài liệu với nhau, kiểm thử toàn bộ phần mềm
- Driver và stub
  - Để kiểm thử tài liệu a, các tài liệu b,c,d phải trở thành những stub
    - Một tài liệu trống hoặc
    - In ra một thông báo ("Procedure radarCalc called"), hoặc
    - Trả lại một phần giá trị từ các trường hợp kiểm thửu đã được lập kế hoạch

- Để kiểm thử tài liệu h thì yêu cầu một bộ điều khiển (driver) mà sẽ gọi mô đun h
  - Một lần, hoặc
  - Một vài lần, hoặc
  - Nhiều lần, mỗi lần kiểm tra giá trị được trả lại
- Để kiểm thử tài liệu d yêu cầu một bộ điều khiển và 2 stubs

- Vấn đề 1
  - Stubs và drivers phải được viết sau đó phải được kiểm thử đơn vị hoàn thiện mới được sử dụng
- Vấn đề 2
  - Thiếu sự cô lập lỗi
  - Mội lỗi có thể nằm ở bất cứ chỗ nào của 13 mô đun (artifact)được tạo ra hoặc 13 giao diện (interface)
  - Với một phần mềm lớn, có 103 mô đun (artifact) và 108 giao diện (interface), thì phải có 211 chỗ lỗi có thể xảy ra
- Giải pháp cho cả hai vấn đề
  - Kết hợp giữa kiểm thử đơn vị và tích hợ

10.1.2.1 Tích hợp trên xuống

- Nếu mô đun mAbove gửi một thông điệp với mô đun mBelow, thì mAbove phải được cài đạt và tích hợp trước mBelow
- Thứ tự từ trên xuống có thể là
  - a,b,c,d,e,f,g,h,i,j,k,l,m



- Một thứ tự khác từ trên xuống

  a
  [a] b,e,h
  [a] c,d,f,i
  [a,d] g,j,k,l,m
- Thuận lợi 1: Cô lập lỗi
  - Trường hợp kiểm thử thành công trước đó bị lỗi khi mô đun mNew được thêm vào những cái đã được kiểm thử cho đến lúc này

        - Lỗi phải nằm trong mô đun mNew hoặc giao diện giữa mNew và phần còn lại của phần mềm
- Thuận lợi 2: Stubs không bị lãng phí
    - Mỗi stub được sử dụng vào mô đun hoàn thiện tương ứng ở mỗi bước thích hợp
- Thuận lợi 3: Những thiếu sót thiết kế chính được đưa ra từ sớm
- Các mô đun lô gic bao gồm luồng điều khiển đưa ra quyết định (Logic artifacts include the decision-making flow of control)
    - Trong ví dụ, các mô đun a,b,c,d,g,j
- Các mô đun hoạt động thực hiện những thao tác thực sự của phần mềm
    - Trong ví dụ, các mô đun e,f,h,i,k,l,m
- Các mô đun lô gic được xây dựng trước các mô đun hoạt động
- Vấn đề 1
    - Các mô đun có thể sử dụng lại không được kiểm thử một cách thích đáng
    - Các mô đun mức thấp hơn (mức hành động) không được kiểm thử thường xuyên
    - Vấn đề càng trở nên trầm trọng nếu sản phẩm phần mềm thiết kế tốt (The situation is aggravated if the product is well designed)
- Defensive programming (fault shielding)
    - Ví dụ:
        - if (x >= 0)
        - y = computeSquareRoot (x, errorFlag);
    - computeSquareRoot không bao giờ kiểm thử với x < 0
    - Điều này ngăn cản để sử dụng lại

10.1.2.2 Tích hợp dưới lên

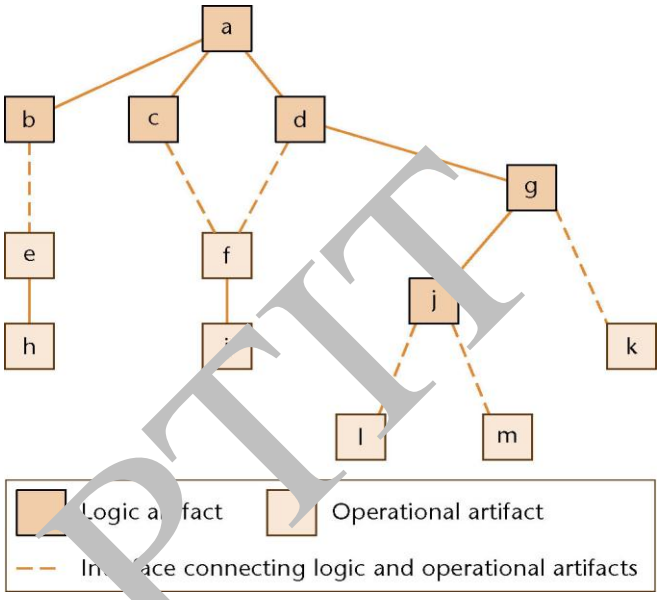
- Nếu mô đun mAbove gọi mô đun mBelow, thì mBelow được cài đặt và tích hợp trước mAbove
- Thứ tự tích hợp từ dưới lên có thể là : l,m,h,i,j,k,e,f,g,b,c,d,a
- Một thứ tự tích hợp từ dưới lên khác có thể là

  h,e,b
  i,f,c,d
  l,m,j,k,g [d]
  a [b,c,d]
- Thuận lợi 1
    - Các mô đun hoạt động được kiểm thử kỹ lưỡng
- Thuận lợi 2
    - Các mô đun hoạt động được kiểm thử với các bộ điều khiển (driver), không có tấm chắn lỗi, các mô đun được lập trình một cách ….(Operational artifacts are tested with drivers, not by fault shielding, defensively programmed artifacts)
    - Thuận lợi

- Cô lập lỗi
- Khó khăn 1
  - Các lỗi thiết kế được phát hiện muộn
- Giải pháp
  - Kết hợp chiến lược tích hợp dưới lên và trên xuống để tận dụng điểm mạnh của cả hai chiến lược và cực tiểu điểm yếu của chúng

10.1.2.3 Tích hợp Sanwich

- Các mô đun lô gic được tích hợp trên xuống (Logic artifact)
- Các mô đun hoạt động tích hợp dưới lên (Operational artifacts)
- Cuối cùng, các giao diện của hai nhóm mô đun trên được kiểm thử



- Thuận lợi 1
  - Các lỗi thiết kế chính được tìm thấy sớm
- Thuận lợi 2
  - Các mô đun hoạt động được kiểm thử kỹ lưỡng
  - Chúng có thể được sử dụng lại một cách tin tưởng
- Thuận lợi 3
  - Luôn luôn có sự cô lập lỗi

Tổng kết:

Approach	Strengths	Weaknesses
Implementation then integration (Section 14.6)	—	No fault isolation Major design faults show up late Potentially reusable code artifacts are not adequately tested
Top-down integration (Section 14.6.1)	Fault isolation Major design faults show up early	Potentially reusable code artifacts are not adequately tested
Bottom-up integration (Section 14.6.2)	Fault isolation Potentially reusable code artifacts are adequately tested	Major design faults show up late
Sandwich integration (Section 14.6.3)	Fault isolation Major design faults show up early Potentially reusable code artifacts are adequately tested	—

10.1.2.4 Tích hợp các phần mềm hướng đối tượng

- Cài đặt và tích hợp hướng đối tượng
  - Hầu hết các phần mềm đều cài đặt và tích hợp sandwich
  - Các đối tượng được tích hợp dưới lên
  - Các mô đun khác được tích hợp trên xuống

10.1.2.5 Quản lý tích hợp

- Ví dụ:
  - Tài liệu thiết kế được sử dụng bởi người lập trình P1 (người đã viết mã đối tượng o1) chỉ ra đối tượng o1 gửi thông điệp tới đối tượng o2 với bốn tham số
  - Tài liệu thiết kế được sử dụng bởi người lập trình P2 (người đã viết mã đối tượng o2) chỉ ra rõ ràng rằng chỉ 3 đối số được truyền tới đối tượng o2
- Giải pháp:
  - Tiến trình tích hợp phải được quản lý bởi nhóm SQA

10.2 KIỂM THỬ PHA CÀI ĐẶT VÀ TÍCH HỢP

10.2.1 Luồng công việc kiểm thử cài đặt

- Kiểm thử đơn vị
  - Kiểm thử đơn vị không hình thức được thực hiện bởi người lập trình
  - Kiểm thử đơn vị một cách cẩn thận có phương pháp bởi nhóm SQA

- Có hai loại kiểm thử đơn vị có phương pháp
    - Kiểm thử không có sự thực thi
    - Kiểm thử dựa trên sự thực thi
- Lựa chọn trường hợp kiểm thử
    - Cách tồi nhất – kiểm thử ngẫu nhiên
        - Không có thời gian để kiểm thử tất cả nhưng phần trăm nhỏ nhất của các trường hợp có thể kiểm thử được trên tổng số là 10% hoặc hơn
    - Chúng ta cần có một cách có hệ thống để xây dựng các trường hợp kiểm thử

10.2.1.1 Kiểm thử với các đặc tả so với kiểm thử với mã

- *Kiểm thử với các đặc tả(Test to specifications)* (cũng được gọi là kiểm thử hướng đầu vào/đầu ra, kiểm thử hướng dữ liệu, hoặc kiểm thử chức năng hoặc kiểm thử hộp đen)
    - Lờ qua mã – sử dụng các đặc tả để lựa chọn các trường hợp kiểm thử
- *Kiểm thử với mã(Test to code) (cũng được gọi là kiểm thử hướng đường dẫn, cấu trúc, hướng lô gic, kiểm thử hộp kính)*
    - Lờ đi các đặc tả - sử dụng mã để lựa chọn trường hợp kiểm thử

Tính khả thi của việc kiểm thử với đặc tả

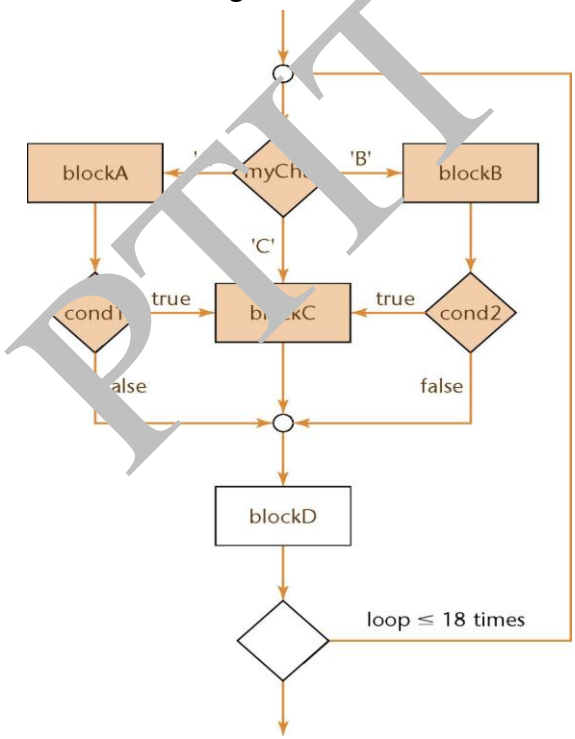
- Ví dụ:
    - Các đặc tả đối với một phần mềm xử lý dữ liệu gồm 5 loại nhiệm vụ và 7 (types of discount)
    - 35 trường hợp kiểm thử
- We cannot say that commission and discount are computed in two entirely separate artifacts — the structure is irrelevant
- Mục đích của các đặc tả bao gồm 20 nhân tố, mỗi nhân tố, mỗi nhân tố đảm nhiệm 4 giá trị
    - Có 4^{20} hoặc 1.1×10^{12} trường hợp kiểm thử
    - Nếu mỗi nhân tố sử dụng mất 30 giây để thực thi thì việc chạy tất cả các trường hợp kiểm thử mất nhiều hơn 1triệu năm
- Sự tăng nhanh tổ hợp làm cho kiểm thử sử dụng các đặc tả không thể thực hiện được

Tính khả thi của kiểm thử với mã

- Mỗi đường dẫn tiến trình xuyên suốt mỗi mô đun phải được thử hiện ít nhất một lần
    - Sự tăng nhanh tổ hợp (Combinatorial explosion)
- Ví dụ mã:

```
read (kmax)                          // kmax is an integer between 1 and 18
for (k = 0; k < kmax; k++) do
{
   read (myChar)                     // myChar is the character A, B, or C
   switch (myChar)
   {
     case 'A':
       blockA;
       if (cond1) blockC;
       break;
     case 'B':
       blockB;
       if (cond2) blockC;
       break;
     case 'C':
       blockC;
       break;
   }
   blockD;
}
```

- Biểu đồ luồng có hơn 10^{12} đường dẫn khác nhau



- Kiểm thử với mã không đáng tin cậy

```
if ((x + y + z)/3 == x)
  print "x, y, z are equal in value";
else
  print "x, y, z are unequal";
```

Test case 1: x = 1, y = 2, z = 3
Test case 2: x = y = z = 2

- Chúng ta có thể thi hành từng đường dẫn mà không phát hiện ra hết lỗi
- Mỗi đường dẫn có thể được kiểm thử chỉ khi nó đưa ra
- Khi người lập trình bỏ sót kiểm thử với giá trị d =0 trong mã thì họ không ý thức được mức độ nguy hiểm có thể xảy ra

```
if (d == 0)
  zeroDivisionRoutine ();
else
  x = n/d
```

```
x = n/d;
```

(b)

- Tiêu chuẩn ("thực hiện một đường dẫn") là không đáng tin
    - Products exist for which some data exercising a given path detect a fault, and other data exercising the same path do

10.2.1.2 Kỹ thuật kiểm thử đơn vị hộp đen

- Cả kiểm thử với các đặc tả và kiểm thử với mã đều không có tính khả thi
- Nghệ thuật kiểm thử:
    - Lựa chọn một tập nhỏ, có thể quản lý được của các trường hợp kiểm thử để
    - Cực đại những cơ hội phát hiện lỗi, trong khi
    - Cực tiểu những cơ hội lãng phí trường hợp kiểm thử
- Mỗi trường hợp kiểm thử phải phát hiện ra một lỗi không phát hiện được trước đó
- Chúng ta cần một phương thức làm nổi bật nên nhiều lỗi nhất có thể
    - Đầu tiên thực hiện các trường hợp kiểm thử hộp đen (Kiểm thử với các đặc tả)
    - Sau đó là các phương thức kiểm thử hộp kính (kiểm thử với mã)

a. Kiểm thử tương đương và phân tích các giá trị biên

- Ví dụ

- Các đặc tả cho DBMS chỉ ra rằng sản phẩm phần mềm phải xử lý một số lượng bản ghi nằm trong khoảng từ 1 đến 16,383 ($2^{14} - 1$)
- Nếu hệ thống có thể xử lý trong 34 bản ghi và 14,870 bản ghi, thì nó có thể làm việc tốt với 8,252 bản ghi

- Nếu hệ thống làm việc với bất cứ trường hợp kiểm thử nào nằm trong khoảng (1…16,383)If the , thì nó có thể làm việc tốt với bất cứ trường hợp kiểm thử nào thuộc khoảng đó
  - Dãy (1..16,383) tạo thành lớp tương đương
- Kiểm thử tương đương
  - Bất cứ thành viên nào của lớp tương tương cũng là một trường hợp kiểm thử tốt bằng các thành viên khác của lớp tương đương
  - Dãy (1..16,383) xác định ra ba lớp tương đương:
    - Lớp tương đương 1: Ít hơn 1 bản ghi
    - Lớp tương đương 2: Giữa 1 và 16,383 bản ghi
    - Lớp tương đương thứ 3: Nhiều hơn 16,383 bản ghi

- Phân tích các giá trị biên

  Lựa chọn trường hợp kiểm thử chỉ dựa trên biên của các lớp tương đương
  - Điều này làm tăng nhanh xác suất phát hiện lỗi

  Ví dụ cơ sở dữ liệu
  - Trường hợp kiểm thử 1: [illegible] bản ghi, thành viên của lớp tương đương 1 và kề sát với giá trị biên
  - Trường hợp kiểm thử 2: 1 bản ghi, giá trị biên
  - Trường hợp kiểm thử 3: 2 bản ghi, kề sát với giá trị biên
  - Trường hợp kiểm thử 4: 723 bản ghi là thành viên của lớp tương đương 2
  - Trường hợp kiểm thử 5: 16,382 giá trị bản ghi, kề sát với giá trị biên
  - Trường hợp kiểm thử 6: 16,383 giá trị bản ghi, giá trị biên
  - Trường hợp kiểm thử 7: 16,384 giá trị bản ghi, là thành viên thuộc lớp tương đương 3 và kề sát với giá trị biên

- Kiểm thử tương đương của các đặc tả đầu ra
  - Chúng ta cũng cần thực hiện kiểm thử tương đương các đặc tả đầu ra
  - Ví dụ:
    - Năm 2006, In 2006, the minimum Social Security (OASDI) deduction from any one paycheck was $0.00, and the maximum was $5,840.40
    - Test cases must include input data that should result in deductions of exactly $0.00 and exactly $5,840.40
    - Also, test data that might result in deductions of less than $0.00 or more than $5,840.40

- Chiến lược tổng quan
    - Các lớp tương đương sử dụng chung phân tích giá trị biên để kiểm thử cả đặc tả đầu vào và đặc tả đầu ra
        - Phương pháp này sinh ra một tập nhỏ dữ liệu kiểm thử với khả năng phát hiện ra một số lượng lớn lỗi

b. Kiểm thử chức năng

- Một kiểu khác của kiểm thử hộp đen đối với phần mềm cổ điển
    - Dự liệu kiểm thử dựa trên những chức năng của các mô đun
    - Mỗi mục của chức năng hoặc chức năng được xác định
    - Dữ liệu kiểm thử được nghĩ ra để kiểm thử chức năng ở mức thấp hơn một các tách biệt
- Sau đó, các chức năng mức cao đã kết hợp với các chức năng ở mức thấp được kiểm thử
- Tuy nhiên, trong thực tế
    - Không phải các chức năng mức cao hơn luôn luôn được xây dựng tách biệt khỏi những chức năng mức thấp hơn bằng cách sử dụng cấu trúc của lập trình cấu trúc(Higher-level functions are not always directly constructed out of lower-level functions using the constructs of structured programming)
    - Thay vì đó, chức năng mức thấp hơn thường được đan xen vào
- Cũng như thế, các biên về mặt chức năng không phải luôn luôn trùng khớp với biên của các mô đun mã
    - Sực phân biệt giữa kiểm thử đơn vị và kiểm thử tích hợp trở thành lu mờ
    - Vấn đề này cũng có thể nảy sinh trong mô hình hướng đối tượng khi thông điệp được truyền giữa các đối tượng
- Mối quan hệ qua lại ngẫu nhiên giữa các mô đun mã có thể dẫn đến kết quả tiêu cực trong quản lý
    - Các mốc quan trọng và thời hạn cuối cùng có thể trở nên không rõ ràng
    - Sau đó rất khó để xác định trạng thái của dự án

10.2.1.3 Kỹ thuật kiểm thử đơn vị hộp kính

- Phủ dòng lệnh (statement coverage)
- Phủ nhánh(Branch coverage)
- Phủ đường dẫn (Path coverage)
- Chuỗi mã tuyến tính
- Phủ đường dẫn sử dụng tất cả các định nghĩa (All-definition-use path coverage)

a. Kiểm thử cấu trúc: phủ dòng lệnh, nhánh và đường dẫn

Phủ dòng lệnh

- Thực hiện tập trường hợp kiểm thử trong đó mỗi dòng lệnh được thực hiện ít nhất một lần

- Công cụ CASE cần được sử dụng để kiểm tra

- Nhược điểm
  - Câu lệnh rẽ nhánh
  - Cả hai câu lệnh đều được thực thi mà không chỉ ra lỗi

```
if (s > 1 && t == 0)
  x = 9;
```

Test case: s = 2, t = 0.

Phủ nhánh

- Việc thực hiện một tập các trường hợp kiểm thử trong đó mỗi nhánh được thực hiện ít nhất một lần (cũng như tất cả các câu lệnh)
  - Điều này giải quyết vấn đề ở phủ dòng lệnh
  - Công cụ CASE là cần thiết

Phủ đường dẫn

- Thực hiện tập các trường hợp kiểm thử trong đó mỗi đường dẫn được thực hiện ít nhất một lần (cũng như tất cả các câu lệnh)
- Vấn đề:
  - Số lượng của các đường dẫn có thể rất lớn
  - Chúng ta muốn một điều kiện ít thực hiện ít đường dẫn hơn nhưng lại chỉ ra được nhiều lỗi hơn phủ nhánh

Chuỗi mã tuyến tính

- Xác định ra tập các điểm mà từ các điểm đó luồng điều khiển có thể nhảy đến một vị trí nào đó, cộng các đầu mục và thoát khỏi các điểm (Identify the set of points L from which control flow may jump, plus entry and exit points)
- Hạn chế các trường hợp kiểm thử so với phủ đường dẫn bằng cách bắt đầu và kết thúc với các thành phần của L
- Phương pháp này phát hiện ra nhiều lỗi mà không phải kiểm thử mọi đường dẫn
- *Phủ đường dẫn sử dụng tất cả các định nghĩa*
- Mỗi biến cố của biến zz được gán nhãn hoặc là:
  - Định nghĩa của một biến (The *definition* of a variable)
    zz = 1 or read (zz)
  - Hoặc sự sử dụng của một biến (or the *use* of variable)
    y = zz + 3 or if (zz < 9) errorB ()
  - Xác định tất cả các đường dẫn từ sự định nghĩa của một biến tới sự sử dụng của định nghĩa đó
  - Điều này có thể được thực hiện bằng một công cụ tự động
- Mỗi trường hợp kiểm thử được thiết lập cho mỗi đường dẫn như vậy

- Bất lợi:
  - Với d nhánh thì có trên 2^d số lượng đường dẫn
- (Upper bound on number of paths is 2^d, where *d* is the number of branches)
- Trong thực tế:
  - Số lượng đường dẫn thực tế tương ứng với d
- Do đó đây là kỹ thuật lựa chọn trường hợp kiểm thử thực tế
- Không thể kiểm thử một câu lệnh cụ thể
  - Chúng ta có thể có một đường dẫn không khả thi ("mã chết") trong mô đun
- Thường đây là dấu hiệu của lỗi

```
if (k < 2)
{
   if (k > 3)              [should be k > −3]
         ↑
      x = x * k;
}
```

(a)

```
for (j = 0; j < 0; j++)   [should be j < 10]
      ↑
   total = total + value[j];
```

(b)

b. Các thước đo độ phức tạp

- Là một phương pháp đảm bảo chất lượng để kiểm thử hộp kính
- Mô đun m1 phức tạp hơn mô đun m2
  - Về mặt trực quan, m1 có khả năng sinh ra nhiều lỗi hơn mô đun m2
- Nếu độ phức tạp vượt quá mức độ cho phép thì nên thiết kế lại và sau đó viết mã lại thì mô đun viết mã
  - Rẻ hơn và nhanh hơn việc cố gắn sửa lỗi mô đun viết mã có thể xảy ra lỗi

Số lượng dòng mã

- Thước đo đơn giản nhất của độ phức tạp
  - Giả định cơ bản: có một xác suất một dòng mã chứa lỗi là p
- Ví dụ
  - Người kiểm thử tin tưởng mỗi dòng mã có 2% khả năng sinh ra lỗi.
  - Nếu tài liệu mã kiểm thử có 200 dòng lệnh thì có thể chứa 2 lỗi
- Thực vậy, số lượng lỗi liên quan tới kích cỡ của toàn bộ sản phẩm

Các thước đo khác để đo độ phức tạp

- Cyclomatic complexity *M* (McCabe)
  - Số lượng các điểm quyết định (các nhánh) trong mô đun mã

  - Dễ dàng tính toán
  - Thước đo tốt của các lỗi (xem slide sau)
- Trong 1 thử nghiệm, các mô đun mã với $M > 10$ đã thống chỉ ra nhiều lỗi hơn về mặt thống kê

Vấn đề với thước đo độ phức tạp

- Thước đo độ phức tạp, đặc biệt là cyclomatic complexity, đã trải qua những thử thách trong
  - Lý thuyết
  - Thử nghiệm và
  - Có liên quan nhiều với LOC
- Về bản chất, chúng ta đang đo số lượng dòng mã, không phải độ phức tạp
- Rà soát mã sẽ phát hiện lỗi nhanh và kỹ lưỡng
  - Giảm tới 95% chi phí bảo trì

10.2.1.4 So sánh các kỹ thuật kiểm thử đơn vị

- So sánh các thử nghiệm
  - Kiểm thử hộp đen
  - Kiểm thử hộp kính
  - Các kiểu rà soát
  - [Myers, 1978] 59 lập trình viên có kinh nghiệm tốt
  - Cả ba phương pháp đều có hiệu quả bằng nhau trong việc phát hiện lỗi
  - Rà soát kỹ lưỡng đã có hiệu năng về chi phí thấp (Code inspections were less cost-effective)
- [Hwang, 1981]
  - Cả ba phương pháp đều có hiệu quả bằng nhau
- [Basili and Selby, 1987] 42 sinh viên tiến tiên tiến trong 2 nhóm, 32 lập trình viên chuyên nghiệp
  - Những sinh viên tiên tiến ở nhóm 1
  - Không có sự khác nhau đáng kể trong ba phương pháp kiểm thử
  - Những sinh viên tiên tiến ở nhóm 2
  - Rà soát mã và kiểm thử hộp đen tốt bằng nhau
  - Cả hai việc trên làm tốt hơn kiểm thử hộp kính
- Những người lập trình chuyên nghiệp
  - Rà soát mã phát hiện được nhiều lỗi hơn
  - Rà soát mã có tốc độ phát hiện lỗi nhanh hơn
- Kết luận
  - Rà soát kỹ lưỡng mã ít nhất cũng fát hiện được số lượng lỗi bằng với kiểm thử hộp đen và hộp kính

10.2.1.5 Cleanroom

- Là một phương pháp tiếp cận khác đối với phát triển phần mềm

- Hợp nhất
    - Mô hình tiến trình tăng
    - Các kỹ thuật hình thức
    - Các kiểu rà soát
- Prototype automated documentation system for the U.S. Naval Underwater Systems Center
- 1820 dòng của FoxBASE
    - 18 lỗi được phát hiện bởi "xác minh chức năng"
    - Kiểm tra không hình thức được sử dụng
    - 19 lỗi được phát hiện bằng rà soát lướt qua trước khi biên dịch
    - Không có lỗi biên dịch
    - Không có sự thất bại ở thời gian thực thi

Sự khác nhau trong việc tính toán tỷ lệ lỗi kiểm thử:

- Các mô hình thông thường:
    - Đếm số lỗi sau khi kiểm thử không hình thức được hoàn thành (Khi SQA bắt đầu)
- Cleanroom
    - Đếm số lỗi sau khi rà soát kỹ lưỡng được hoàn thành (khi biên dịch được bắt đầu)

Báo cáo về 17 sản phẩm phần mềm Cleanroom

- Hệ điều hành
    - 350,000 LOC
    - Phát triển trong 18 tháng
    - Bởi một đội 70 người
    - Tỷ lệ lỗi kiểm thử chỉ là 1.0 lỗi trên KLOC
- Các loại phần mềm khác nhau với tổng số 1triệu LOC
    - Tỷ lệ lỗi kiểm thử trung bình có đánh trọng số: 2.3 lỗi trên KLOC
- "[R]emarkable quality achievement"

10.2.1.6 Những vấn đề có khả năng xảy ra khi kiểm thử các đối tượng

- Phải kiểm tra kỹ lưỡng các lớp và các đối tượng
- Có thể chạy các trường hợp kiểm thử đối với các đối tượng (nhưng không đối với các lớp)
- Một mô đun cô điển điển hình :
    - Gồm khoảng 50 câu lệnh có thể thực thi
    - Sinh ra các tham số đầu vào, kiểm tra các tham óố đầu ra
- Các đối tượng điển hình:
    - Gồm khoảng 30 phương thức, mỗi phướng thức chỉ với 2 hoặc 3 câu lệnh
    - Một phương thức thường không trả lại một giá trị tới người gọi – thay vào đó thì nó thay đổi trạng thái
    - Nó không thể kiểm tra trạng thái bởi vì sự ẩn giấu thông tin
    - Ví dụ: phương thức determineBalance —cần biết trước accountBalance
- Chúng ta cần thêm vào các phương thức để trả lại các giá trị của các biến trạng thái

- Đó là một phần của kế hoạch kiểm thử
- Biên dịch điều kiện có thể phải được sử dụng (Conditional compilation may have to be used)

- Các phương thức đã kế thừa có thể vẫn phải được kiểm thử (xem ví dụ sau đây)

Ví dụ cài đặt java của một cây phân cấp

```
class RootedTreeClass
{
  ...
  void displayNodeContents (Node a);
  void printRoutine (Node b);
//
// method displayNodeContents uses method printRoutine
//
  ...
}

class BinaryTreeClass extends RootedTreeClass
{
  ...
  void displayNodeContents (Node a);
//
// method displayNodeContents defined in this class uses
// method printRoutine inherited from ClassRootedTree
//
  ...
}

class BalancedBinaryTreeClass extends BinaryTreeClass
{
  ...
  void printRoutine (Node b);
//
// method displayNodeContents (inherited from BinaryTreeClass) uses this
// local version of printRoutine within class BalancedBinaryTreeClass
//
  ...
}
```

Nửa trên Khi displayNodeContents được gọi trong**BinaryTreeClass**, nó sử dụng uses **RootedTreeClass**.printRoutine

```
class RootedTreeClass
{
  ...
  void displayNodeContents (Node a);
  void printRoutine (Node b);
//
// method displayNodeContents uses method printRoutine
//
  ...
}

class BinaryTreeClass extends RootedTreeClass
{
  ...
  void displayNodeContents (Node a);
//
// method displayNodeContents defined in this class uses
// method printRoutine inherited from ClassRootedTree
//
  ...
}
```

Nửa dưới khi **displayNodeContents** được gọi trong **BalancedBinaryTreeClass**, nó sử dụng **BalancedBinaryTreeClass**.printRoutine

```
class BinaryTreeClass extends RootedTreeClass
{
  ...
  void displayNodeContents (Node a);
//
// method displayNodeContents defined in this class uses
// method printRoutine inherited from ClassRootedTree
//
  ...
}

class BalancedBinaryTreeClass extends BinaryTreeClass
{
  ...
  void printRoutine (Node b);
//
// method displayNodeContents (inherited from BinaryTreeClass) uses this
// local version of printRoutine within class BalancedBinaryTreeClass
//
  ...
}
```

- Tin xấu
  - **BinaryTreeClass**.displayNodeContents phải được kiểm thử lại từ ban đầu khi sử dụng **BalancedBinaryTreeClass**
  - Nó gọi một phiên bản khác của printRoutine
- Tin xấu hơn
  - Với những lý do lý thuyết, cần sử dụng toàn bộ những trường hợp kiểm thử khác nhau để kiểm thử

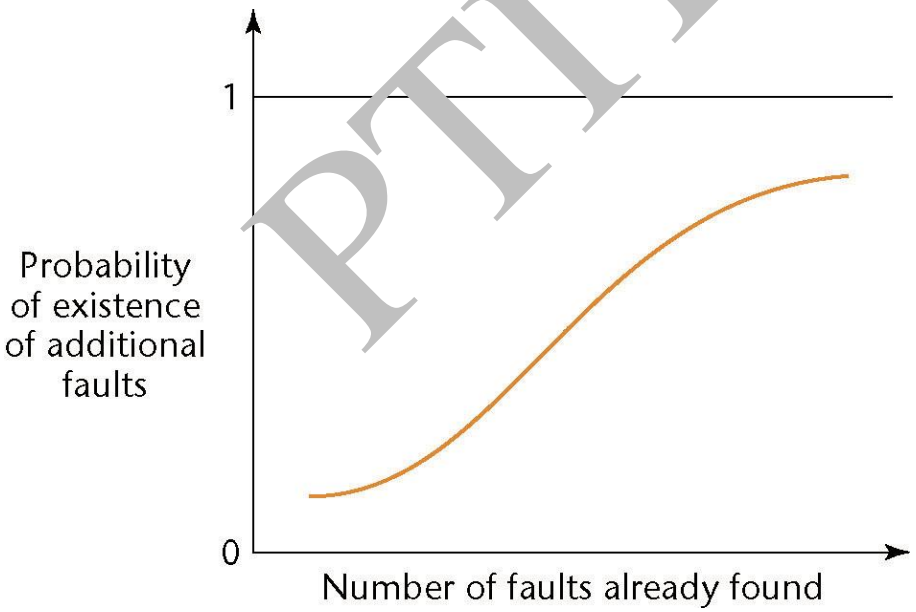
- Làm cho các biến trạng thái có thể thấy được
  - Vấn đề nhỏ
- Việc kiểm thử lại trước khi sử dụng lại
  - Chỉ xuất hiện khi các phương thức tương tác
  - Chúng ta có thể xác định khi nào cần kiểm thử lại
- Đây không phải là những lý do để từ bỏ mô hình hướng đối tượng

10.2.1.7 Các khía cạnh quản lý của kiểm thử đơn vị

- Cần biết khi nào dừng kiểm thử
- Các kỹ thuật khác nhau có thể được sử dụng
  - Phân tích chi phí – lợi nhuận
  - Phân tích rủi ro
  - Các kỹ thuật thống kê

10.2.1.8 Khi nào viết lại hơn là gỡ lỗi

- Khi mô đun mã có quá nhiều lỗi
  - Thiết kế lại và viết mã lại hơn



- Rủi ro và chi phí của các lỗi thêm nữa sẽ rất lớn
- Với mỗi mô đun, việc quản lý phải xác định trước số lượng lỗi lớn nhất được cho phép trong suốt quá trình kiểm thử
- Nếu con số này đạt tới thì
  - Bỏ qua
  - Thiết kế lại
  - Viết mã lại
- Số lượng lỗi lớn nhất được cho phép sau khi chuyển giao là 0

Sự phân bố của lỗi trong mô đun là không đồng nhất

- [Myers, 1979]
  - 47% lỗi trong OS/370 chỉ nằm ở 4% các mô đun
- [Endres, 1975]
  - 512 lỗi nằm trong 202 mô đun của DOS/VS (phát hành lần thứ 28)
  - 112 mô đun chỉ có 1 lỗi
  - Có các mô đun chỉ có 14, 15, 19 và 28 lỗi
  - Mô đun với 28 lỗi là mô đun lớn nhất trong phần mềm, với hơn 3000 dòng lệnh của ngôn ngữ hợp ngữ macro DOS (The latter three were the largest modules in the product, with over 3000 lines of DOS macro assembler language)
  - Mô đun với 14 lỗi là mô đun tương đối nhỏ và rất bất định
  - Giải pháp để ra là bỏ qua, thiết kế lại, viết mã lại

10.2.2 Kiểm thử tích hợp

- Là việc kiểm thử khi có một mô đun mã mới được thêm vào nhóm các mô đun đã được kiểm thử
- Vấn đề đặc biệt có thể nảy sinh khi kiểm thử giao diện người dùng

Kiểm thử tích hợp giao diện người dùng

- Các trường hợp kiểm thử GUI bao gồm
  - Những cú nhấp chuột, và
  - Nhấn phím
- Những kiểu trường hợp kiểm thử này không thể được lưu giữ theo cách thông thường
  - Yêu cầu các công cụ CASE đặc biệt
- Ví dụ:
  - QAPartner
  - XRunner

10.3 KIỂM THỬ SẢN PHẨM

- Kiểm thử sản phẩm cho phần mềm COTS
  - Kiểm thử Alpha, beta
- Kiểm thử sản phẩm đối với phần mềm tùy chỉnh
  - Nhóm SQA phải đảm bảo rằng phần mềm chuyển qua kiểm thử chấp nhận
  - Thất bại kiểm thử chấp nhận gây ra tác động xấu đến tổ chức phát triển

Kiểm thử sản phẩm đối với phần mềm tùy chỉnh

- Đội SQA phải cố gắng thực hiện kiểm thử gần giống với kiểm thử chấp nhận
  - Thực hiện các trương hợp kiểm thử hộp đen đối với toàn sản phẩm phần mềm
  - Tính mạnh mẽ của toàn bộ sản phẩm
    - *Stress testing* (under peak load)
    - *Volume testing* (e.g., can it handle large input files?)

    - Tất cả những ràng buộc phải được kiểm tra
    - Tất cả tài liệu phải được
        - Kiểm tra tính chính xác
        - Kiểm tra sự tương thích với chuẩn
        - Xác minh lại với phiên bản hiện thời của phần mềm
- Sản phẩm phần mềm (tài liệu và mã) được chuyển giao cho tổ chức khách hàng để thực hiện kiểm thử chấp nhận

10.4 KIỂM THỬ CHẤP NHẬN

- Khách hàng xác định liệu phần mềm thoả mãn những đặc tả của nó
- Kiểm thử chấp nhận được thực hiện bởi
    - Tổ chức khách hàng, hoặc
    - Đội SQA cùng với sự có mặt của đại diện của khách hàng, hoặc
    - Đội SQA độc lập được khách hàng thuê
- Bốn thành phần chính của kiểm thử chấp nhận là
    - Tính chính xác
    - Tính mạnh mẽ
    - Hiệu năng
    - Tài liệu
- Những thành phần này được kiểm thử cẩn thận bởi người phát triển trong suốt quá trình kiểm thử sản phẩm
- Sự khác nhau giữa kiểm thử sản phẩm và kiểm thử chấp nhận là:
    - Kiểm thử chấp nhận được thực hiện trên dữ liệu thực
    - Kiểm thử sản phẩm được thực hiện trên dữ liệu thử nghiệm, là những cái được định nghĩa không có thực

10.5 CASE STUDY CHO PHA CÀI ĐẶT: VIẾT TEST CASE

Nội dung phần này sẽ trình bày quá trình kiểm thử cho các modul đã trình bày trong pha thiết kế của phần mềm quản lí đặt phòng khách sạn:

- Modul thêm phòng mới của khách sạn
- Modul sửa thông tin một phòng khách sạn
- Modul đặt phòng khách sạn

10.5.1 Test case cho modul thêm phòng mới của khách sạn

a. Lập kế hoạch test

Có hai trường hợp phải test cho modul này:

- Thêm một phòng chưa có trong CSDL

- Thêm một phòng đã có trong CSDL

b. Các test case

+ Test case 1: thêm một phòng chưa có trong CSDL

CSDL trước khi test:

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	900000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

Các bước thực hiện	**Kết quả mong đợi**
Click vào chức năng thêm phòng trên giao diện quản lí phòng	Giao diện thêm phòng hiện ra
Nhập phòng mới: - id = 8 - hotel id = 4 - name = 103 - type = double - display price = 1 200 000 - description = và click vào nút submit	Thông báo hiện ra: Thêm phòng thành công !
click vào nút OK của thông báo	Quay trở về giao diện trang chủ quản lí thông tin phòng

CSDL sau khi test:

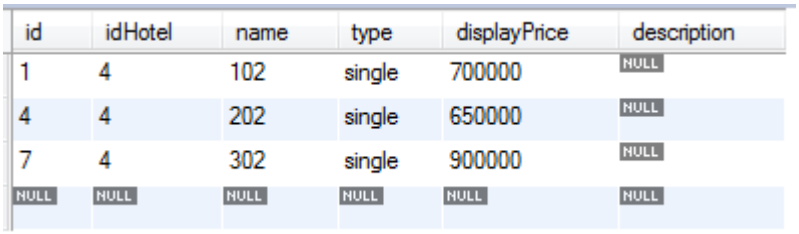
id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	900000	NULL
8	4	103	double	1200000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

+ Test case 2: thêm một phòng đã có trong CSDL

CSDL trước khi test:

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	900000	NULL
NULL	NULL	NULL	NULL	NULL	

Các bước thực hiện	**Kết quả mong đợi**
Click vào chức năng thêm phòng trên giao diện quản lí phòng	Giao diện thêm phòng hiện ra
Nhập phòng mới: - id = 7 - hotel id = 4 - name = 302 - type = single - display price = 900 000 - description = và click vào nút Submit	Thông báo hiện ra: phòng đã tồn tại !

click vào nút OK của thông báo	quay trở lại giao diện thêm phòng cho người dùng sửa lại các thông tin để tránh bị trùng
CSDL sau khi test: 	

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	900000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

10.5.2 Test case cho modul sửa thông tin phòng của khách sạn

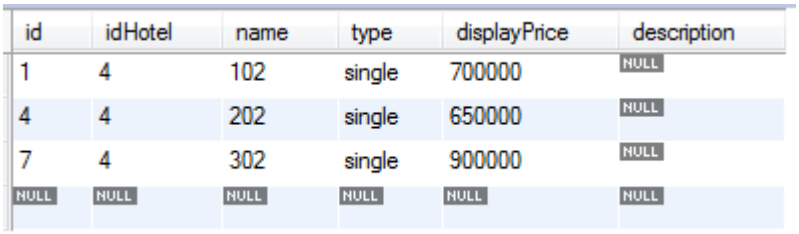
a. Lập kế hoạch test

Có hai trường hợp phải test cho modul này:

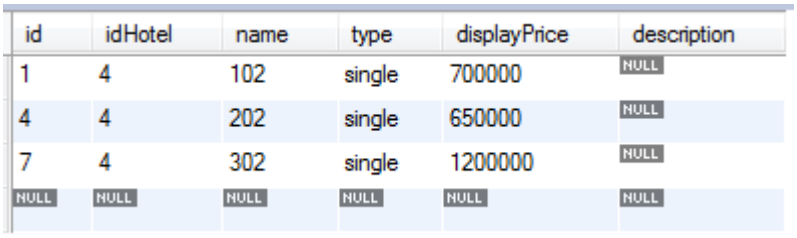
- Sửa một phòng đã có trong CSDL
- Sửa một phòng chưa có trong CSDL

b. Các test case

+ Test case 1: sửa một phòng đã có trong CSDL

CSDL trước khi test:	
	
Các bước thực hiện	**Kết quả mong đợi**
Click vào chức năng sửa thông tin phòng từ giao diện quản lí phòng	Giao diện tìm kiếm phòng theo mã hiện ra

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	900000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

Nhập id = 7 và click vào nút Search	Giao diện hiện lên thông tin phòng có mã là 7 ở dạng edit được: - id = 7 (không sửa được) - hotel id = 4 (không sửa được) - name = 302 - type = single - display price = 900 000 - description = và nút Submit
Sửa thông tin giá phòng: - id = 7 (không sửa được) - hotel id = 4 (không sửa được) - name = 302 - type = single - display price = 1 200 000 - description = và click vào nút Submit	Thông báo hiện lên: sửa phòng thành công!
Click vào nút OK của thông báo	Quay về giao diện chính của quản lí thông tin phòng
CSDL sau khi test: 	

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	1200000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

+ Test case 2: sửa một phòng chưa có trong CSDL

CSDL trước khi test:

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	900000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

Các bước thực hiện	**Kết quả mong đợi**
Click vào chức năng sửa thông tin phòng trong menu quản lí thông tin phòng	Giao diện tìm kiếm phòng theo mã hiện ra
Gõ id = 8 và click vào nút tìm kiếm	Thông báo hiện ra: không tồn tại phòng
click vào nút OK của thông báo	Quay trở lại Giao diện tìm kiếm phòng theo mã

CSDL sau khi test:

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	900000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

10.5.3 Test case cho modul đặt phòng

a. Lập kế hoạch test

Có hai trường hợp phải test cho modul này:

- Đặt phòng: có phòng trống và chưa có khách hàng trong CSDL
- Đặt phòng: có phòng trống và đã có khách hàng trong CSDL
- Đặt phòng: không có phòng trống

b. Các test case

+ Test case 1: có phòng trống và chưa có khách hàng trong CSDL

CSDL trước khi test:

Bảng tblRoom:

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	1200000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

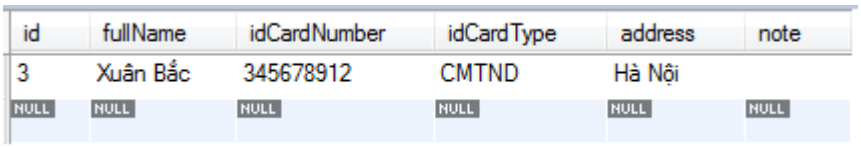
Bảng tblClient:

id	fullName	idCardNumber	idCardType	address	note
1	Xuân Hinh	123456789	CMTND	Nam Định	
2	Minh Vượng	234567891	CMTND	Hà Nội	
3	Xuân Bắc	345678912	CMTND	Hà N	
NULL	NULL	NULL	NULL	NU	NULL

Bảng tblBooking:

id	idRoom	idUser	startDate	e Date	price
1	1	1	2013-08-25	2013- 28	750000
2	4	2	13-08-2	2013-09-	800000
3	7	3	20 08-2	2013-09-15	600000
NULL	NULL	NULL	NULL	NULL	NULL

Các bước thực hiện	**Kết quả mong đợi**
Chọn chức năng đặt phòng từ menu chính	Giao diện tìm phòng trống hiện ra
Nhập : - Ngày bắt đầu = 04 - 09 - 2013 - Ngày kết thúc = 08 – 09 – 2013 Và click vào nút Search	Kết quả hiện ra danh sách gồm 2 phòng trống: id | idHotel | name | type | displayPrice | description 1 | 4 | 102 | single | 700000 | NULL 4 | 4 | 202 | single | 650000 | NULL

Click chọn phòng 202	Quay trở về giao diện đặt phòng (cập nhật thông tin phòng chọn và ngày bắt đầu, ngày kết thúc)
Click chọn tìm kiếm khách hàng	Giao diện tìm kiếm khác hàng theo tên hiện ra
Nhập vào: - Name = Bắc Và click vào nút Search	kết quả hiện ra thông tin khách hàng:  id | fullName | idCardNumber | idCardType | address | note 3 | Xuân Bắc | 345678912 | CMTND | Hà Nội |
Click vào nút Thêm khách hàng	Giao diện thêm khách hàng mới hiện ra
Nhập: - id = 4 - name = Bắc Đẩu - id card = 2233445566 - id type = CMTND - address = Đà Nẵng và click Submit	Thông báo: Thêm khách hàng thành công!
Clock vào nút OK của thông báo	Quay trở lại giao diện đặt phòng (cập nhật thông tin khách hàng mới vào form)
Click nút Submit	Thông báo: đặt phòng thành công!
Click nút OK của thông báo	Quay trở về trang chủ của nhân viên bán hàng

CSDL sau khi test:

Bảng tblRoom:

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	1200000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

Bảng tblClient:

id	fullName	idCardNumber	idCardType	address	note
1	Xuân Hinh	123456789	CMTND	Nam Định	
2	Minh Vượng	234567891	CMTND	Hà Nội	
3	Xuân Bắc	345678912	CMTND	Hà N	
4	Bắc Đẩu	2233445566	CMTND	Đ Nẵng	NULL
NULL	NULL	NULL	NULL	NULL	

Bảng tblBooking:

id	idRoom	idUser	startDate	end te	price
1	1	1	201	2013-08	750000
2	4	2	13-08-2	2013-09-02	800000
3	7	3	2013 26	2013-09-15	600000
4	4	4	2013-09-	2013-09-08	650000
NULL	NULL	NULL	NULL	NULL	NULL

+ Test case 2: có phòng trống và đã có khách hàng trong CSDL

CSDL trước khi test:

Bảng tblRoom:

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	1200000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

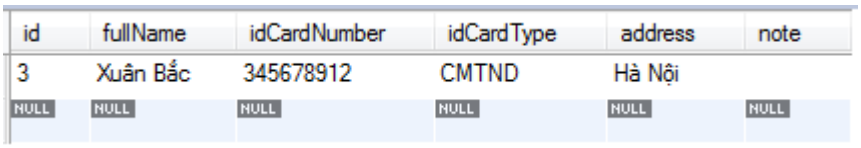
Bảng tblClient:

id	fullName	idCardNumber	idCardType	address	note
1	Xuân Hinh	123456789	CMTND	Nam Định	
2	Minh Vượng	234567891	CMTND	Hà Nội	
3	Xuân Bắc	345678912	CMTND	Hà N	
NULL	NULL	NULL	NULL	NUL	NULL

Bảng tblBooking:

id	idRoom	idUser	startDate	e Date	price
1	1	1	2013-08-25	2013- 28	750000
2	4	2	13-08-2	2013-09-	800000
3	7	3	20 08-2	2013-09-15	600000
NULL	NULL	NULL	NULL	NULL	NULL

Các bước thực hiện	**Kết quả mong đợi**
Chọn chức năng đặt phòng từ menu chính	Giao diện tìm phòng trống hiện ra
Nhập : - Ngày bắt đầu = 04 - 09 - 2013 - Ngày kết thúc = 08 – 09 – 2013 Và click vào nút Search	Kết quả hiện ra danh sách gồm 2 phòng trống: id | idHotel | name | type | displayPrice | description 1 | 4 | 102 | single | 700000 | NULL 4 | 4 | 202 | single | 650000 | NULL

Click chọn phòng 202	Quay trở về giao diện đặt phòng (cập nhật thông tin phòng chọn và ngày bắt đầu, ngày kết thúc)
Click chọn tìm kiếm khách hàng	Giao diện tìm kiếm khác hàng theo tên hiện ra
Nhập vào: - Name = Bắc Và click vào nút Search	kết quả hiện ra thông tin khách hàng:  id | fullName | idCardNumber | idCardType | address | note 3 | Xuân Bắc | 345678912 | CMTND | Hà Nội |
Click chọn khách hàng "Xuân Bắc"	Quay trở về giao diện đặt phòng phòng (cập nhật thông tin khách hàng mới vào form)
Click nút Submit	Thông báo: đặt phòng thành công!
Click nút OK của thông báo	Quay trở về trang chủ của nhân viên bán hàng

CSDL sau khi test:

Bảng tblRoom:

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	1200000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

Bảng tblClient:

id	fullName	idCardNumber	idCardType	address	note
1	Xuân Hinh	123456789	CMTND	Nam Định	
2	Minh Vượng	234567891	CMTND	Hà Nội	
3	Xuân Bắc	345678912	CMTND	Hà N	
NULL	NULL	NULL	NULL	NULL	NULL

Bảng tblBooking:

id	idRoom	idUser	startDate	endDate	price
1	1	1	2013-[illegible]	2013-[illegible]	750000
2	4	2	[illegible]13-08-2[illegible]	2013-09-[illegible]	800000
3	7	3	20[illegible]	2013-09-15	600000
4	4	3	2013-0[illegible]	2013-09-08	650000
NULL	NULL	NULL	NULL	NULL	NULL

+ Test case 3: không có phòng trống

CSDL trước khi test:

Bảng tblRoom:

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	1200000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

Bảng tblClient:

id	fullName	idCardNumber	idCardType	address	note
1	Xuân Hinh	123456789	CMTND	Nam Định	
2	Minh Vượng	234567891	CMTND	Hà Nội	
3	Xuân Bắc	345678912	CMTND	Hà N	
NULL	NULL	NULL	NULL	NU	NULL

Bảng tblBooking:

id	idRoom	idUser	startDate	e Date	price
1	1	1	2013-08-25	2013- 28	750000
2	4	2	2013-08-2	2013-09-	800000
3	7	3	20 08-	2013-09-15	600000
NULL	NULL	NULL	NULL	NULL	NULL

Các bước thực hiện	**Kết quả mong đợi**
Chọn chức năng đặt phòng từ menu chính	Giao diện tìm phòng trống hiện ra
Nhập : - Ngày bắt đầu = 24 - 08 - 2013 - Ngày kết thúc = 28 – 08 – 2013 Và click vào nút Search	Thông báo hiện ra: không còn phòng trống trong khoảng thời gian đã cho!
Click vào nút OK của thông báo	Quay trở về giao diện tìm kiếm phòng trống

CSDL sau khi test:

Bảng tblRoom:

id	idHotel	name	type	displayPrice	description
1	4	102	single	700000	NULL
4	4	202	single	650000	NULL
7	4	302	single	1200000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

Bảng tblClient:

id	fullName	idCardNumber	idCardType	address	note
1	Xuân Hinh	123456789	CMTND	Nam Định	
2	Minh Vượng	234567891	CMTND	Hà Nội	
3	Xuân Bắc	345678912	CMTND	Hà N	
NULL	NULL	NULL	NULL	NU	NULL

Bảng tblBooking:

id	idRoom	idUser	startDate	e Date	price
1	1	1	2013-08-25	2013- 28	750000
2	4	2	2013-08-2	2013-09-	800000
3	7	3	20 08-3	2013-09-15	600000
NULL	NULL	NULL	NULL	NULL	NULL

CHƯƠNG 11: PHA BẢO TRÌ

11.1 PHA BẢO TRÌ SAU KHI CHUYỂN GIAO

Bất cứ thay đổi nào ở bất cứ thành phần nào của phần mềm (bao gồm tài liệu) sau khi phần mềm đó đã trải qua kiểm thử chấp nhận

11.1.1 Tại sao bảo trì sau khi chuyển giao là cần thiết

- Bảo trì sửa lỗi
  - Để sửa lỗi còn sót lại
    - Phân tích, thiết kế, cài đặt, tài liệu hoặc bất cứ các loại lỗi khác
- Bảo trì hoàn thiện (Perfective maintenance)
  - Các yêu cầu khách hàng thay đổi để cải thiện hiệu năng phần mềm
    - Thêm các chức năng
    - Làm cho phần mềm chạy nhanh hơn
    - Cải thiện việc bảo trì
- Bảo trì thích hợp
  - Đáp ứng những thay đổi với môi trường mà phần mềm hoạt động
    - Phần mềm được chuyển sang trình biên dịch mới, hệ điều hành hoặc/và phần cứng mới
    - Thay đổi mã thuế
    - Mã ZIP 9 số

11.1.2 Người lập trình bảo trì sau khi chuyển giao yêu cầu những gì?

- Ít nhất 67 % tổng số chi phí của phần mềm được dồn lại trong suốt quá trình bảo trì sau khi chuyển giao
- Bảo trì là một nguồn thu nhập chính
- Tuy nhiên, ngày nay nhiều tổ chức chỉ định việc bảo trì cho
  - Những người bắt đầu không bị giám sát và (Unsupervised beginners)
  - Ít người lập trình thành thạo
- Bảo trì sau khi chuyển giao là một trong những khía cạnh khó của sản phẩm phần mềm bởi vì
  - Bảo trì sau khi chuyển giao kết hợp các khía cạnh của các luồng công việc khác
- Cho rằng một bản ghi khuyết điểm được chuyển giao cho người lập trình bảo trì
  - Nhớ là "khuyết điểm" là một thuật ngữ chung của lỗi, thất bại
- Nguyên nhân là gì?
  - Không có cái gì sai
  - Sổ tay người dùng có thể bị sai, khổng phải ở mã lệnh
  - Tuy nhiên, thường có một lỗi nằm trong mã lệnh

a- Bảo trì sửa lỗi

- Công cụ nào mà người lập trình bảo trì phải dùng để tìm ra lỗi?
  - Bản ghi khuyết điểm được đưa ra bởi người dùng
  - Mã nguồn
  - Thuường không còn gì khác
- Do đó người lập trình bảo trì phải có kỹ năng gỡ lỗi xuất sắc
  - Lỗi có thể nằm ở bất cứ chỗ nào trong phần mềm
  - Nguyên nhân đầu tiên của lỗi có thể nằm ở đặc tả không tồn tại hoặc tài liệu thiết kế

- Giả sử rằng người lập trình bảo trì đã định vị được lỗi
- Vấn đề:
  - Cách cố định lỗi mà không đưa ra lỗi hồi quy
- Cách cực tiểu lỗi hồi quy
  - Tham khảo tài liệu chi tiết của toàn bộ sản phẩm
  - Tham khảo tài liệu chi tiết của mỗi mô đun riêng lẻ
- Cái gì thường xuyên xảy ra
  - Không có tài liệu nào, hoặc
  - Tài liệu không hoàn thiện, hoặc
  - Tài liệu bị lỗi
- Người lập trình phải xem xét lại mã nguồn để tránh đưa ra lỗi hồi quy
- Người lập trình thay đổi mã nguồn
- Kiểm thử để thấy phần chỉnh sửa làm việc một cách chính xác
  - Đặc biệt, việc sử dụng những trường hợp kiểm thử cấu trúc
- *Người lập trình phải*
  - Kiểm tra đối với các lỗi hồi quy
    - Sử dụng dữ liệu kiểm thử đã lưu trữ
  - Thêm các trường hợp kiểm thử đã xây dựng một cách đặc biệt vào dữ liệu kiểm thử đã lưu trữ để cho việc kiểm tra hồi quy trong tương lai
  - Viết tài liệu tất cả các thay đổi
- Những kỹ năng chính được yêu cầu cho bảo trì sửa lỗi
  - Kỹ năng chuẩn đoán giỏi
  - Kỹ nưng kiểm thử giỏi
  - Kỹ năng viết tài liệu giỏi

b- Bảo trì hoàn thiện và bảo trì thích ứng

- Người lập trình bảo trì phải đi xuyên suốt các luồng công việc
  - Xác định các yêu cầu
  - Viết các đặc tả
  - Thiết kế
  - Cài đặt và tích hợp
- Việc sử dụng các phần mềm có sẵn từ ban đầu
- Khi người lập trình đã phát triển
  - Các đặc tả được đưa ra bởi những chuyên gia phân tích
  - Thiết kế được đưa ra bởi các chuyên gia thiết kế
  - Mã nguồn được viết bởi các chuyên viên lập trình
  - Nhưng những người lập trình bảo trì phải là chuyên gia ở cả ba lĩnh vực trên và cả lĩnh vực kiểm thử và viết tài liệu

Kết luận

- Không có khuôn mẫu cho bảo trì
  - Có phải đó là một công việc cho những người bắt đầu không bị giám sát hoặc
  - Bảo trì nên được thực hiện bởi chuyên gia máy tính không có kỹ năng

c- Phần thưởng của bảo trì

- Bảo trì là một công việc bạc bẽo theo mọi cách
  - Người bảo trì thương lượng với những người dùng không hài lòng về phần mềm
  - Nếu người dùng vui, thì phần mềm sẽ không cần bảo trì

- Vấn đề của người dùng thường bắt nguồn từ những cá nhân đã phát triển sản phẩm phần mềm, không phải người bảo trì
- Bản thân mã lệnh có thể được viết rất tồi
- Bảo trì sau khi chuyển giao bị nhiều người phát triển phần mềm xem thường
- Trừ khi dịch vụ bảo trì tốt được đưa ra thì khách hàng sẽ thực hiện những giao dịch phát triển trong tương lai ở một nơi khác
- Bảo trì sau khi chuyển giao là một khía cạnh thử thách nhất của phần mềm và bạc bẽo nhất

- Những người quản lý phải chỉ định công việc bảo trì cho những người lập trình giỏi nhất và
- Trả lương cho họ phù hợp

11.1.3 Quản lý bảo trì sau khi chuyển giao

- Các vấn đề khác nhau về mặt quản lý bảo trì sau khi chuyển giao cần được xem xét
- Trước tiên, người lập trình bảo trì nên xem xét tệp bản ghi khuyết điểm
- Nó bao gồm
  - Tất cả các lỗi đã được ghi lại mà chưa sửa và
  - Những đề nghị về các công việc sẽ thực hiện về những khuyết điểm đó
- Trong một thế giới lý tưởng
  - Sửa tất cả mọi lỗi ngay lập tức
  - Sau đó chúng ta công bố phiên bản phần mềm mới ở mọi vị trí
- Trong thế giới thực
  - Phân bố các bản ghi lỗi ở tất cả các vị trí
  - Không có nhân viên để bảo trì ngay lập tức
  - Thực hiện nhiều thay đổi ở cùng một lúc sẽ rẻ hơn, đặc biệt nếu có nhiều vị trí cài đặt

a- Các bản ghi khuyết điểm

- Cần một cơ chế đối với việc thay đội sản phẩm phần mềm
- Nếu sản phẩm phần mềm xuất hiện một chức năng không đúng, thì người dùng đưa ra một bản ghi khuyết điểm
  - Bản ghi đó phải đủ thông tin để cho phép người lập trình bảo trì tái tạo lại vấn đề
- Theo lý tưởng, mỗi khuyết điểm nên được cố định ngay lập tức
  - Trong thực tế, tốt nhất chúng ta có thể làm là nghiên cứu sơ bộ ngay lập tức
- Nếu khuyết điểm đã được ghi lại trước đó: Đưa thông tin trong tệp bản ghi khuyết điểm tới người dùng
- Nếu nó là một khuyết điểm mới:
  - Người lập trình bảo trì nên cố gắng tìm
    - Nguyên nhân,
    - Cách để sửa khuyết điểm đó, và
    - Cách làm việc xung quanh vấn đề đó
  - Khuyết điểm mới được ghi lại vào tệp tường trình phát hiện lỗi, cùng với tài liệu
    - Dánh sách (Listings)
    - Thiết kế (Designs)
    - Sổ tay (Manuals)

- Tệp bản ghi khuyết điểm cũng nên chứa những yêu cầu cảu khách hàng để bảo trì thích hợp và hoàn thiện chức năng
  - Nội dung của tệp phải được định độ ưu tiên bởi khách hàng
  - Những chỉnh sửa tiếp theo là một trong những nội dung có độ ưu tiên cao nhất trong tệp
- Các bản sao của bản ghi khuyết điểm phải lưu hành tới tất cả
  - Bao gồm: ước lượng khi nào khuyết điểm được sửa
- Nếu cùng thất bại xảy ra ở một vị trí khác, người dùng có thể xác định
  - Khả năng làm việc xung quanh khuyết điểm
  - Thời gian mà khuyết điểm được sửa

b- Cho phép thay đổi phần mềm

- Bảo trì sửa lỗi
  - Chỉ định một người lập trình bảo trì xác định lỗi và nguyên nhân của lỗi, sau đó sửa lỗi đó
  - Kiểm thử sửa chữa, kiểm thử toàn bộ phần mềm (kiểm thử hồi quy)
  - Cập nhật tài liệu để phản ánh những thay đổi đã thực hiện
  - Cập nhật những lời giải thích ban đầu để phản ánh
    - Những gì đã thay đổi,
    - Tại sao nó được thay đổi,
    - Ai thực hiện thay đổi, và
    - Khi nào
- Bảo trì thích hợp và hoàn thiện
  - Giống với bảo trì sửa lỗi, ngoại trừ không có bản ghi khuyết điểm
  - Thay vì có thay đổi trong yêu cầu
- Điều gì sẽ xảy ra nếu người lập trình không kiểm thử những lỗi đã được sửa một cách thích đáng?
  - Trước khi phần mềm được phân phối, thì phần mềm phải được kiểm thử bởi nhóm SQA
- Bảo trì sau khi chuyển giao là cực kỳ khó
- Kiểm thử là khó và tiêu tốn thời gian
  - Được thực hiện bởi nhóm SQA
- Kỹ thuật phiên bản cơ sở và các bản sao riêng phải được sử dụng
- Người lập trình thực hiện các thay đổi đối với các bản sao chép riêng của các mô đun mã và kiểm thử chúng
- Người lập trình đóng băng phiên bản trước đó, và đưa ra phiên bản chỉnh sửa cho nhóm SQA để kiểm thử
- SQA thực hiện kiểm thử trên phiên bản cơ sở của tất cả các mô đun mã

c- Bảo đảm việc bảo trì

- Bảo trì không là sự cố gắng một lần (Maintenance is not a one-time effort)
- Phải lập kế hoạch cho bảo trì xuyên suốt toàn bộ vòng đời phần mềm
  - Luồng công việc thiết kế - sử dụng kỹ thuật ẩn dấu thông tin
  - Luồng cài đặt – lựa chọn đặt tên có ý nghĩa để thuận tiện cho những người lập trình trong tương lai

- Tài liệu phải được hoàn thiện và chính xác và phản ánh đúng phiên bản hiện thời của mỗi mô đun mã
- Trong suốt quá trình bảo trì sau khi chuyển giao, bảo trì không phải giàn
  - Luôn luôn biết rõ việc bảo trì trong tương lai là không thể tránh được

d- Vấn đề của bảo trì lặp

- Việc thay đổi những yêu cầu phần mềm gây nhiều khó khăn cho đội phát triển
- Việc thay đổi thường xuyên thường gây bất lợi cho việc bảo trì phần mềm

Thay đổi bài toán đích

- The problem is exacerbated during postdelivery maintenance
- Càng nhiều thay đổi
  - Thì sản phẩm phần mềm càng khác xa so với thiết kế ban đầu
  - Việc thay đổi trong tương lai càng trở nên khó hơn
  - Tài liệu trở nên kém tin cậy hơn bình thường
  - Các file kiểm thử hồi quy không được cập nhật
  - Viết lại toàn bộ có thể cần thiết đối với bảo trì trong tương lai
- Giải pháp hiển nhiên
  - Đóng băng các đặc tả khi chúng được ký đến tận khi chuyển giao sản phẩm phần mềm
  - Sau mỗi yêu cầu của bảo trì hoàn thiện, đóng băng các đặc tả trong 3 tháng hoặc 1 năm
- Trong thực tế
  - Khách hàng có thể đưa ra những thay đổi ngay ngày hôm sau
  - Nếu bằng lòng với giá cả thì khách hàng có thể đưa ra những thay đổi hàng ngày
- "Ai trả tiền thì người ấy có tiền"("He who pays the piper calls the tune")

11.1.4 Bảo trì sau khi chuyển giao với kỹ năng phát triển

- Những kỹ năng cần thiết cho bảo trì gồm
  - Khả năng xác định nguyên nhân gây ra lỗi của một phần mềm lớn
    - Cũng cần thiết trong suốt quá trình tích hợp và kiểm thử sản phẩm
  - Khả năng thực hiện chức năng có hiệu quả mà không cần có tài liệu chính xác
    - Tài liệu hiếm khi được hoàn thiện đến tận khi chuyển giao
  - Kỹ năng phân tích, thiết kế, cài đặt và kiểm thử
    - Tất cả bốn hoạt động được thực thi trong suốt quá trình phát triển
- Những kỹ năng cần thiết cho bảo trì sau khi chuyển giao giống với những kỹ năng của tất cả các luồng công việc khác
- Điểm chính
  - Người lập trình bảo trì không phải chỉ có kỹ năng rộng ở mọi lĩnh vực mà những kỹ năng đó phải ở trình độ cao
  - Sự chuyên môn hóa không thể có ở những người lập trình bảo trì

11.1.5 Kỹ nghệ ngược

- Khi nào tài liệu duy nhất đối với bảo trì sau khi chuyển giao là mã nguồn thì
  - Bắt đầu với mã
  - Tái tạo lại thiết kế
  - Tái tạo lại các đặc tả (cực kỳ khó)
  - Công cụ CASE có thể trở giúp ((flowcharters, các mục đích trực quan khác)

- Kỹ nghệ lại
  - Kỹ nghệ ngược, được sinh ra bởi kỹ nghệ tiên tiến (Reverse engineering, followed by forward engineering)
  - Mức độ thấp hơn tới cao hơn tới thấp hơn của trừu tượng (Lower to higher to lower levels of abstraction)
- Xây dựng lại
  - Việc cải thiện sản phẩm phần mềm mà không có thay đổi chức năng phần mềm
  - Ví dụ:
    - Cải thiện việc bảo trì
    - Xây dựng lại(XP, agile processes)
- Điều gì sẽ xảy ra nếu chúng ta chỉ có mã thực thi?
  - Xem xét phần mềm như hộp đen
  - Suy luận những đặc tả từ hành vi của phần mềm hiện thời

11.1.6 Công cụ CASE cho bảo trì sau khi chuyển giao

- Công cụ điều khiển cấu hình là cần thiết
  - Công cụ thương mại
    - CCC
  - Công cụ mã nguồn mở
    - *cvs*
- Các công cụ kỹ nghệ lại
  - Các công cụ thương mại
    - IBM Rational [illegible], Together
  - Các công cụ mã nguồn mở
    - Doxygen
- Các công cụ theo dõi – phát hiện
  - Công cụ thương mại
    - IBM Rational ClearQuest
  - Công cụ mã nguồn mở
    - Bugzilla

10.1.7 Thước đo của bảo trì sau khi chuyển giao

- Về bản chất, các hoạt động của bảo trì sau khi chuyển giao là những hoạt động của quá trình phát triển
  - Các thước đo đối với luồng công việc phát triển
- Các thước đo bản ghi khuyết điểm (Defect report metrics)
  - Sự phân loại khuyết điểm
  - Trạng thái khuyết điểm (Defect status)

10.1.8 Những thách thức của bảo trì sau khi chuyển giao

- Chương này miêu tả rất nhiều thách thức
- Thách thức khó nhất cần giải quyết
  - Bảo trì khó hơn phát triển, nhưng
  - Những người phát triển có xu hướng nhìn xuống (tend to look down) những người bảo trì và
  - Thường xuyên được trả tiền lương nhiều hơn

11.2 BẢO TRÌ HỆ PHẦN MỀM HƯỚNG ĐỐI TƯỢNG

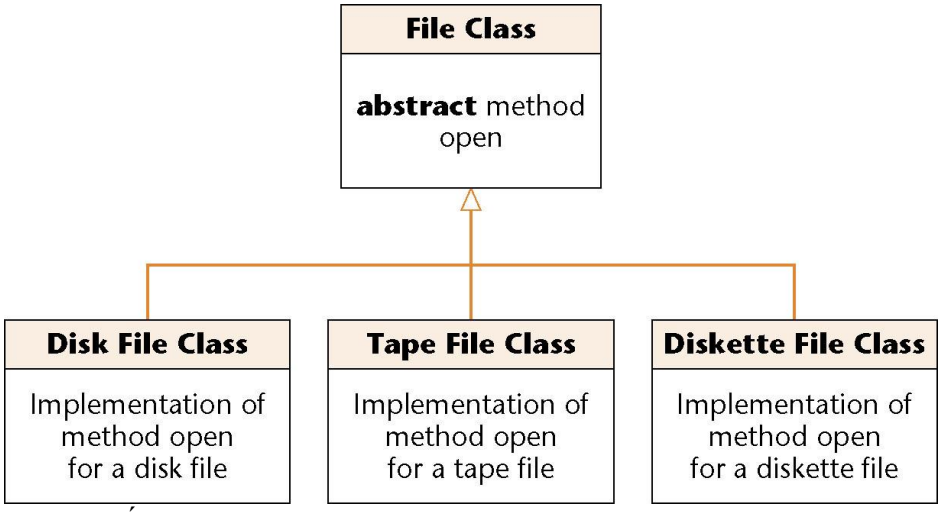
- Bề ngoài, mô hình hướng đối tượng khuyến kích việc bảo trì theo bốn cách
  - Phần mềm bao gồm các đơn vị độc lập
  - Đóng gói (độc lập về mặt khái niệm)
  - Ẩn giấu thông tin (độc lập về mặt vật lý)
  - Truyền tham số là các giao tiếp duy nhất (Message-passing is the sole communication)
- Thực thế hơi khác (The reality is somewhat different)
- Ba cản trở

Một là: Cây phân cấp kế thừa có thể lớn

```
class UndirectedTreeClass
{
  ...
  void displayNode (Node a);
  ...
}// class UndirectedTreeClass

class DirectedTreeClass : public UndirectedTreeClass
{
  ...
}// class DirectedTreeClass

class RootedTreeClass : public DirectedTreeClass
{
  ...
  void displayNode (Node a);
  ...
}// class RootedTreeClass

class BinaryTreeClass : public RootedTreeClass
{
  ...
}// class BinaryTreeClass

class BalancedBinaryTreeClass : public BinaryTreeClass
{
  Node        hhh;
  displayNode (hhh);
}// class BalancedBinaryTreeClass
```

- Để hình dung ra những gì displayNode làm trong **BalancedBinaryTreeClass**, chúng ta phải kiểm tra tỉ mỉ cây hoàn thiện
  - Cây hoàn thiện có thể trải rộng toàn bộ phần mềm
  - Khác xa "những đơn vị độc lập " (A far cry from "independent units")
  - Giải pháp
  - Công cụ CASE có thể dàn mỏng cây kế thừa (A CASE tool can flatten the inheritance tree)

Hai là:Hậu quả của liên kết động và đa hình



- Hệ thống không hoạt động khi gọi myFile.open ()
- Phiên bản nào của open có chứa lỗi?
  - Công cụ CASE không thể trợ giúp (công cụ tĩnh)
  - Chúng ta phải theo dõi (kiểm tra)
- Liên kết động và đa hình có thể có
  - Ảnh hưởng tích cực tới đội phát triển, nhưng
  - Ảnh hưởng tiêu cực đối với bảo trì

Ba là: Hậu quả của kế thừa

- Tạo một lớp mới qua kế thừa
- Lớp con mới
  - Không ảnh hưởng tới lớp cha, và
  - Không ảnh hưởng tới bất cứ lớp con
  - Chỉnh sửa lớp con mới này
  - Một lần nữa không ảnh hưởng
  - Chỉnh sửa lớp cha
  - Tất cả các lớp con kế thừa đều bị ảnh hưởng
  - "Fragile base class problem"
- Kế thừa có
  - Ảnh hưởng tích đối với người phát triển, nhưng
  - Ảnh hưởng tiêu cực đối với bảo trì

11.3 KIỂM THỬ PHA BẢO TRÌ

- Người bảo trì xem xét phần mềm như một tập các thành phần có liên quan lỏng lẻo
  - Chúng không liên quan đến sự phát triển phần mềm
- Kiểm thử hồi quy là cần thiết
  - Lưu giữ các trường hợp kiểm thử và đầu ra của chúng, chỉnh sửa nếu cần thiết