NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

…..........................., ngày..........tháng..........năm 2017

**Người nhận xét**

**(Ký tên và ghi rõ họ tên)**

LỜI CẢM ƠN

Với lòng biết ơn sâu sắc nhất, chúng em xin chân thành cảm ơn quý thầy cô ở khoa Công nghệ phần mềm – Đại học Công Nghệ Thông Tin đã cùng với tri thức và tâm huyết của mình truyền đạt vốn kiến thức quý báu cho chúng em trong suốt thời gian học tập tại trường. Trong học kỳ này, chúng em được tiếp cận với môn học “Phương pháp mô hình hóa” – môn học rất hữu ích đối với sinh viên khoa Công nghệ phần mềm. Và đặc biệt, quá trình thực hiện đồ án môn học đã giúp chúng em nâng cao kiến thức, trải nghiệm thực tế việc xây dựng một mô hình hệ thống đơn giản.

Chúng em xin chân thành cảm ơn thầy P.GS TS Vũ Thanh Nguyên và ThS. Nguyễn Công Hoan đã tận tâm hướng dẫn chúng em qua từng buổi học trên lớp. Nếu không có những lời hướng dẫn, chỉ bảo của thầy cô thì chúng em nghĩ quá trình hoàn thiện bài báo cáo này của chúng em sẽ gặp nhiều vấn đề và khó khăn hơn.

Do kiến thức và khả năng của chúng em còn nhiều hạn chế, do đó không tránh khỏi những thiếu sót, yếu kém. Chúng em rất mong nhận được những ý kiến đóng góp quý báu của thầy cô và các bạn học cùng lớp để kiến thức của chúng em được hoàn thiện hơn.

Sau cùng, chúng em xin kính chúc quý thầy cô ở khoa Công nghệ phần mềm và thầy Vũ Thanh Nguyên thật dồi dào sức khỏe, niềm tin để tiếp tục thực hiện sứ mệnh cao đẹp của mình là truyền đạt kiến thức cho thế hệ mai sau.

Trân trọng.

LỜI MỞ ĐẦU

Trong xu thế phát triển của thế giới với sự bùng nổ mạnh mẽ của ngành công nghệ thông tin. Nhu cầu phần mềm sẽ giúp con người nâng cao hiểu quả và chất lượng công việc. Kéo theo đó là những khó khăn của người lập trình phần mềm. Sẽ thật sự phức tạp nếu bạn xây dựng một phần mềm nào đó mà không làm theo một mô hình nhất định dù bạn giỏi đến mức nào.

Để đưa ra một mô hình tổng thể của hệ thống cần xây dựng, người lập trình cần phải thực hiện phân tích hệ thống. Mô hình này cũng có thể được sử dụng để xác định các yêu cầu người dùng đối với hệ thống và qua đó giúp chúng ta đánh giá tính khả thi của dự án. Tạo mô hình sẽ giúp cho chúng ta hiểu thấu đáo một hệ thống phức tạp trong sự tổng thể của nó. Tầm quan trọng của mô hình đã được lĩnh hội một cách thấu đáo trong hầu như tất cả ngành khoa học kỹ thuật từ nhiều thế kỷ nay. Bất kỳ ở đâu, khi muốn xây dựng một vật thể nào đó, đầu tiên người ta đã tạo nên các bản vẽ để quyết định cả ngoại hình lẫn phương thức hoạt động của nó. Mô hình nhìn chung là một cách mô tả của một vật thể nào đó. Vật đó có thể tồn tại trong một số giai đoạn nhất định, dù đó là giai đoạn thiết kế hay giai đoạn xây dựng hoặc chỉ là một kế hoạch. Nhà thiết kế cần phải tạo ra các mô hình mô tả tất các khía cạnh khác nhau của sản phẩm. Ngoài ra, một mô hình có thể được chia thành nhiều hướng nhìn, mỗi hướng nhìn trong số chúng sẽ mô tả một khía cạnh riêng biệt của sản phẩm hay hệ thống cần được xây dựng. Một mô hình cũng có thể được xây dựng trong nhiều giai đoạn và ở mỗi giai đoạn, mô hình sẽ được bổ sung thêm một số chi tiết nhất định.

Mô hình thường được mô tả trong ngôn ngữ trực quan, điều đó có nghĩa là đa phần các thông tin được thể hiện bằng các ký hiệu đồ họa và các kết nối giữa chúng, chỉ khi cần thiết một số thông tin mới được biểu diễn ở dạng văn bản, theo đúng như câu ngạn ngữ “Một bức tranh nói nhiều hơn cả ngàn từ". Tạo mô hình cho các hệ thống phần mềm trước khi thực sự xây dựng nên chúng, đã trở thành một chuẩn mực trong việc phát triển phần mềm.

Nói tóm lại, mô hình hóa một hệ thống nhằm mục đích:

- Hình dung một hệ thống thực tế theo mong muốn của khách hàng.

- Chỉ rõ cấu trúc hoặc ứng xử của hệ thống.

- Tạo một khuôn mẫu hướng dẫn người lập trình trong suốt quá trình xây dựng hệ thống.

- Ghi lại các quyết định của người lập trình để sử dụng sau này.

Trong bài báo cáo này, nhóm chúng em xin giới thiệu về ngôn ngữ mô hình hóa thống nhất (Unifield Modeling Language – UML) để hỗ trợ xây dựng mô hình cho hệ thống phần mềm và áp dụng nó để xây dựng mô hình hệ thống phần Quản lý khách sạn.

**MỤC LỤC**

[NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN 1](#_Toc502444805)

[LỜI CẢM ƠN 2](#_Toc502444806)

[LỜI MỞ ĐẦU 3](#_Toc502444807)

[**CHƯƠNG I : GIỚI THIỆU PHẦN MỀM QUẢN LÍ PHÒNG MẠCH TƯ** 6](#_Toc502444808)

[**I.1** **GIỚI THIỆU PHẦN MÈM** 6](#_Toc502444809)

[**I.2** **HỆ THỐNG CÁC YÊU CẦU PHẦN MỀM** 6](#_Toc502444810)

[**1.1. YÊU CẦU NGHIỆP VỤ** 6](#_Toc502444811)

[**1.2. YÊU CẦU TIẾN HÓA** 10](#_Toc502444812)

[**1.3. YÊU CẦU HIỆU QUẢ** 11](#_Toc502444813)

[**1.4. YÊU CẦU TIỆN DỤNG** 12](#_Toc502444814)

[**1.5. YÊU CẦU TƯƠNG THÍCH** 14](#_Toc502444815)

[**1.6. YÊU CẦU BẢO MẬT** 15](#_Toc502444816)

[**1.7. YÊU CẦU AN TOÀN** 16](#_Toc502444817)

[**1.8 YÊU CẦU CÔNG NGHỆ** 17](#_Toc502444818)

[**CHƯƠNG II : GIỚI THIỆU MÔ HÌNH PHÁT TRIỂN PHẦN MỀM LINH HOẠT ( AGILE SOFT WARE DEVELOPMENT )** 18](#_Toc502444819)

[**II.1** **ĐỊNH NGHĨA VỀ MÔ HÌNH AGILE** 18](#_Toc502444820)

[**II.2** **LỊCH SỬ HÌNH THÀNH MÔ HÌNH AGILE** 18](#_Toc502444821)

[**II.3** **TUYÊN NGÔN VÀ CÁC NGUYÊN LÍ PHÍA SAU AGILE** 19](#_Toc502444822)

[⮚ Tuyên ngôn về Agile: 19](#_Toc502444823)

[⮚ 12 nguyên lí phía sau Agile: 20](#_Toc502444824)

[**II.4** **ĐẶC TRƯNG AGILE** 27](#_Toc502444825)

[**II.5** **CÁC PHƯƠNG PHÁP CỦA AGILE** 29](#_Toc502444826)

[Một số phương pháp phổ biến: 29](#_Toc502444827)

[**II.6** **PHƯƠNG PHÁP SCRUM** 29](#_Toc502444828)

[**CHƯƠNG III : MÔ HÌNH HÓA HỆ THỐNG PHẦN MỀM** 31](#_Toc502444829)

[**III.1** **PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG (OBJECT ORIENTED ANALYSIS AND DESIGN: OOAD)** 31](#_Toc502444830)

[**III.2** **ĐỊNH NGHĨA UML** 33](#_Toc502444831)

[1. Sơ đồ lớp: 34](#_Toc502444832)

[**2.** Cách xây dựng bản vẽ Class 37](#_Toc502444833)

[**3.** Đặc tả Class 39](#_Toc502444834)

[4. Sử dụng bản vẽ Class 40](#_Toc502444835)

[ **USE CASE DIAGRAM** 40](#_Toc502444836)

[ **CLASS DIAGRAM** 46](#_Toc502444837)

[ **ACTIVITY DIAGRAM** 54](#_Toc502444838)

[ **SEQUENCE DIAGRAM** 58](#_Toc502444839)

[ **COMPONENT DIAGRAM** 60](#_Toc502444840)

[ **DEPLOYMENT DIAGRAM** 62](#_Toc502444841)

[**III.3** **ÁP DỤNG OOAD VÀO PHẦN MỀM QUẢN LÍ PHÒNG MẠCH TƯ SỬ DỤNG UML** 66](#_Toc502444842)

[**CHƯƠNG IV: QUẢN LÍ CODE VỚI MÔ HÌNH 3 LỚP** 66](#_Toc502444843)

[**IV.1** **MÔ HÌNH 3 LỚP** 66](#_Toc502444844)

[Giới thiệu mô hình 3-layer ( 3 lớp) 67](#_Toc502444845)

[Ưu điểm 68](#_Toc502444846)

[1.Presentation Layer (GUI): 68](#_Toc502444847)

[2. Bussiness Layer (BLL) : 69](#_Toc502444848)

[3. Data Layer (DAL) : 70](#_Toc502444849)

[**IV.2** **ỨNG DỤNG MÔ HÌNH 3 LỚP VÀO PHẦN MỀM** 72](#_Toc502444850)

# **CHƯƠNG I : GIỚI THIỆU PHẦN MỀM QUẢN LÍ PHÒNG MẠCH TƯ**

## **GIỚI THIỆU PHẦN MÈM**

## **HỆ THỐNG CÁC YÊU CẦU PHẦN MỀM**

### **1.1. YÊU CẦU NGHIỆP VỤ**

#### **1.1.1. Danh sách các yêu cầu**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Tên yêu cầu** | **Biểu mẫu** | **Qui định** | **Ghi chú** |
| 1 | Lập danh sách khám bệnh | BM1 | QĐ1 |  |
| 2 | Lập phiếu khám bệnh | BM2 | QĐ2 |  |
| 3 | Tra cứu bệnh nhân | BM3 |  |  |
| 4 | Lập hóa đơn thanh toán | BM4 | QĐ4 |  |
| 5 | Lập báo cáo tháng | BM5 |  |  |
| 6 | Thay đổi qui định |  | QĐ6 |  |

#### **1.1.2.Danh sách các biểu mẫu và qui định**

***1.1.2.1******. Biểu mẫu 1 và qui định 1***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BM1:** | **Danh Sách Khám Bệnh** | | | |
| Ngày khám:..................................... | | | | |
| **STT** | **Họ Tên** | **Giới Tính** | **Năm Sinh** | **Địa Chỉ** |
| 1 |  |  |  |  |
| 2 |  |  |  |  |

QĐ1: Mỗi ngày khám tối đa 40 bệnh nhân

***1.1.2.******2. Biểu mẫu 2 và qui định 2***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **BM2:** | | **Phiếu Khám Bệnh** | | | | |
| Họ tên: ................................. | | | | Ngày khám: ............................... | | |
| Triệu chứng:......................... | | | | Dự đoán loại bệnh: ................... | | |
| **STT** | **Thuốc** | | **Đơn vị** | | **Số lượng** | **Cách dùng** |
| 1 |  | |  | |  |  |
| 2 |  | |  | |  |  |

QĐ2: Có 5 loại bệnh. Có 30 loại thuốc, 2 đơn vị (viên, chai), có 4 cách dùng (1,2,3,4)

***1.1.2.3******. Biểu mẫu 3***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BM3:** | **Danh Sách Khám Bệnh** | | | |
| **STT** | **Họ Tên** | **Ngày Khám** | **Loại Bệnh** | **Triệu chứng** |
| 1 |  |  |  |  |
| 2 |  |  |  |  |

***1.1.2.4******. Biểu mẫu 4 và qui định 4***

|  |  |  |
| --- | --- | --- |
| **BM4:** | **Hóa đơn Thanh Toán** | |
| Họ và tên: ........................ | | Ngày khám: ..................... |
| Tiền khám: ...................... | | Tiền thuốc: ...................... |

QĐ4: Tiền khám 30.000. Tiền thuốc chỉ có khi bệnh nhân có đúng thuốc. Mỗi loại thuốc có đơn giá riêng.

##### ***1.1.2.5. Biểu mẫu 5***

***Biểu mẫu 5.1***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BM5.1:** | **Báo Cáo Doanh Thu Theo Ngày** | | | |
| Tháng :........................... | | | | |
| **STT** | **Ngày** | **Số Bệnh Nhân** | **Doanh Thu** | **Tỷ Lệ** |
| 1 |  |  |  |  |
| 2 |  |  |  |  |

***Biểu mẫu 5.2***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BM5.2:** | **Báo Cáo Sử Dụng Thuốc** | | | |
| Tháng: .................................... | | | | |
| **STT** | **Thuốc** | **Đơn vị tính** | **Số lượng** | **Số lần dùng** |
| 1 |  |  |  |  |
| 2 |  |  |  |  |

##### ***1.1.2.6. Qui định 6***

QĐ6: Người dùng có thể thay đổi các qui định như sau:

+ QĐ1: Thay đổi số lượng bệnh nhân tối đa trong ngày.

+ QĐ2: Thay đổi số lượng loại bệnh, thuốc, đơn vị tính, cách dùng.

##### ***1.1.2.7. Bảng trách nhiệm yêu cầu nghiệp vụ***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Nghiệp vụ** | **Người dùng** | **Phần Mềm** | **Ghi chú** |
| 1 | Lập danh sách khám bệnh | Cung cấp thông tin về ngày khám, thông tin chi tiết bệnh nhân: họ tên bệnh nhân, giới tính, năm sinh, địa chỉ | Kiểm tra qui định, cập nhật cơ sở dữ liệu và in ra danh sách khám bệnh | Cho phép xem, sửa, xóa các thông tin trên danh sách trước khi hoàn thành việc lập danh sách khám bệnh |
| 2 | Lập phiếu khám bệnh | Cung cấp thông tin về ngày khám, triệu chứng,dự đoán loại bệnh. Danh sách thuốc cùng với thông tin chi tiết liên quan: tên thuốc, đơn vị, số lượng, cách dùng | Kiểm tra qui định, cập nhật cơ sở dữ liệu và in ra phiếu khám bệnh | Cho phép xem, sửa, xóa các thông tin trên danh sách trước khi hoàn thành việc lập phiếu khám bệnh |
| 3 | Tra cứu bệnh nhân | Cung cấp ít nhất 1 thông tin bệnh nhân cần tra cứu | Tìm kiếm và trả ại thông tin về bệnh nhân thỏa mãn | Cho phép tìm kiếm bệnh nhân từ nhiều kiểu thông tin khác nhau: tên bệnh nhân, địa chỉ, …….) |
| 4 | Lập hóa đơn thanh toán | Cung cấp thông tin về ngày thanh toán hóa đơn, tiền khám và tiền thuốc | Kiểm tra qui định, cập nhật cơ sở dữ liệu và in ra hóa đơn thanh toán | Cho phép xem, sửa, xóa các thông tin trên hóa đơn trước khi hoàn thành việc lập hóa đơn thanh toán |
| 5 | Lập báo cáo tháng | Cung cấp thông tin về tháng năm cần lập báo cáo tháng, cách sử dụng thuốc | In ra báo cáo tháng |  |
| 6 | Thay đổi qui định | Cung cấp thông tin về qui định mới cần thay đổi | Cập nhật lại qui định mới |  |

### **1.2. YÊU CẦU TIẾN HÓA**

#### **1.2.1.Danh sách các yêu cầu tiến hóa**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Nghiệp vụ** | **Tham số cần thay đổi** | **Miền giá trị cần thay đổi** |
| 1 | Thay đổi qui định lập danh sách khám bệnh | Số lượng bệnh nhân khám tối đa trong ngày |  |
| 2 | Thay đổi qui định lập phiếu khám bệnh | Danh sách loại bệnh, danh sách loại thuốc, danh sách đơn vị, danh sách cách dùng |  |
| 3 | Thay đổi qui định hóa đơn thanh toán | Có sử dụng qui định này (QDD4)) hay không? | Có/Không |

#### **1.2.2. Bảng trách nhiệm yêu cầu tiến hóa**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Nghiệp vụ** | **Người dùng** | **Phần mềm** | **Ghi chú** |
| 1 | Thay đổi qui định lập danh sách khám bệnh | Cho biết giá trị số lượng bệnh nhân khám tối đa trong ngày | Cập nhật lại qui định mới |  |
| 2 | Thay đổi qui định lập phiếu khám bệnh | Cho biết danh sách loại bệnh, danh sách loại thuốc, danh sách đơn vị, danh sách cách dùng | Cập nhật lại qui định mới |  |
| 3 | Thay đổi qui định hóa đơn thanh toán | Cho biết có sử dụng quy định này (QĐ4) hay không? | Cập nhật lại quy định mới |  |

### **1.3. YÊU CẦU HIỆU QUẢ**

#### **1.3.1. Danh sách các yêu cầu hiệu quả**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Nghiệp vụ** | **Tốc độ xử lí** | **Dung lượng lưu trữ** | **Ghi chú** |
| 1 | Lập danh khách khám bệnh | 40 bệnh nhân/ngày |  |  |
| 2 | Lập phiếu khám bệnh | 1 bệnh nhân/phút |  |  |
| 3 | Tra cứu bệnh nhân | Dưới 10 giây |  |  |
| 4 | Lập hóa đơn thanh toán | 1 bệnh nhân/ phút |  |  |
| 5 | Lập báo cáo tháng | Dưới 10 giây |  |  |
| 6 | Thay đổi qui định | Ngay lập tức |  |  |

#### **1.3.2. Bảng trách nhiệm yêu cầu hiệu quả**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Nghiệp vụ** | **Người dùng** | **Phần mềm** | **Ghi chú** |
| 1 | Lập danh sách khám bệnh | Có sẵn thông tin chi tiết cần được nhập | Thực hiện theo đúng yêu cầu |  |
| 2 | Lập phiếu khám bệnh | Có sẵn danh sách các bệnh nhân và thông tin khám bệnh | Thực hiện theo đúng yêu cầu |  |
| 3 | Tra cứu bệnh nhân | Có sẵn thông tin về bệnh nhân cần tra cứu | Thực hiện theo đúng yêu cầu |  |
| 4 | Lập hóa đơn thanh toán | Có sẵn thông tin về bệnh nhân cần thanh toán hóa đơn | Thực hiện theo đúng yêu cầu |  |
| 5 | Lập báo cáo tháng | Có sẵn thông tin tháng, năm, thông tin sử dụng thuốc | Thực hiện theo đúng yêu cầu |  |
| 6 | Thay đổi qui định | Có sẵn các thông tin về qui định mới | Thực hiện theo đúng yêu cầu |  |

### **1.4. YÊU CẦU TIỆN DỤNG**

#### **1.4.1. Danh sách các yêu cầu tiện dụng**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Nghiệp vụ** | **Mức độ dễ học** | **Mức độ dễ sử dụng** | **Ghi chú** |
| 1 | Lập danh sách khám bệnh | 5 phút hướng dẫn | Rõ ràng, dễ dàng nhập liệu |  |
| 2 | Lập phiếu khám bệnh | 5 phút hướng dẫn | Rõ ràng, dễ dàng nhập liệu |  |
| 3 | Tra cứu bệnh nhân | 5 phút hướng dẫn | Chỉ cần biết 1 và thông tin của bệnh nhân | Có các gợi ý tìm kiếm cho người dùng |
| 4 | Lập hóa đơn thanh toán | Không cần hướng dẫn | Rõ ràng, dễ dàng nhập liệu |  |
| 5 | Lập báo cáo tháng | Không cần hướng dẫn |  |  |
| 6 | Thay đổi qui định | 5 phút hướng dẫn | Rõ ràng, dễ dàng nhập liệu |  |

#### **1.4.2. Bảng trách nhiệm yêu cầu tiện dụng**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Nghiệp vụ** | **Người dùng** | **Phần Mềm** | **Ghi chú** |
| 1 | Lập danh sách khám bệnh | Đọc hướng dẫn | Thực hiện theo đúng yêu cầu |  |
| 2 | Lập phiếu khám bệnh | Đọc hướng dẫn | Thực hiện theo đúng yêu cầu |  |
| 3 | Tra cứu bệnh nhân |  | Thực hiện theo đúng yêu cầu |  |
| 4 | Lập hóa đơn thanh toán | Đọc hướng dẫn | Thực hiện theo đúng yêu cầu |  |
| 5 | Lập báo cáo tháng |  | Thực hiện theo đúng yêu cầu |  |
| 6 | Thay đổi qui định | Đọc hướng dẫn | Thực hiện theo đúng yêu cầu |  |

### **1.5. YÊU CẦU TƯƠNG THÍCH**

#### **1.5.1. Danh sách các yêu cầu tương thích**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Nghiệp vụ** | **Đối tượng liên quan** | **Ghi chú** |
| 1 | Lập danh sách khám bệnh | Nhập tập tin từ Excel, xuất tập tin Excel | Độc lập phiên bản |
| 2 | Lập phiếu khám bệnh | In ra file pdf | Độc lập phiên bản |
| 3 | Lập hóa đơn thanh toán | Xuất tập tin Excel | Độc lập phiên bản |
| 4 | Lập báo cáo tháng | Xuất tập tin Excel | Độc lập phiên bản |

#### **1.5.2. Bảng trách nhiệm yêu cầu tương thích**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Nghiệp vụ** | **Người dùng** | **Phần Mềm** | **Ghi chú** |
| 1 | Lập danh sách khám bệnh | Cài đặt phần mềm Excel | Thực hiện đúng yêu cầu |  |
| 2 | Lập phiếu khám bệnh | Cài đặt phần mềm Excel | Thực hiện đúng yêu cầu |  |
| 3 | Lập hóa đơn thanh toán | Cài đặt phần mềm Excel | Thực hiện đúng yêu cầu |  |
| 4 | Lập báo cáo tháng | Cài đặt phần mềm Excel | Thực hiện đúng yêu cầu |  |

### **1.6. YÊU CẦU BẢO MẬT**

#### **1.6.1. Danh sách các yêu cầu bảo mật**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STT** | **Nghiệp vụ** | **Quản trị hệ thống** | **Bác sỹ** | **Nhân viên** | **Người dùng khác** |
| 1 | Phân quyền | X |  |  |  |
| 2 | Lập danh sách khám bệnh |  |  | X |  |
| 3 | Lập phiếu khám bệnh |  | X | X |  |
| 4 | Tra cứu bệnh nhân |  | X | X | X |
| 5 | Lập hóa đơn thanh toán |  |  | X |  |
| 6 | Lập báo cáo tháng |  |  | X |  |
| 7 | Thay đổi qui định | X |  |  |  |

#### **1.6.2. Bảng trách nhiệm yêu cầu bảo mật**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Nghiệp vụ** | **Người dùng** | **Phần Mềm** | **Ghi chú** |
| 1 | Quản trị hệ thống | Cho biết thông tin người dùng mới và quyền hạn | Ghi nhận và thực hiện |  |
| 2 | Bác sỹ | Cung cấp tên và mật khẩu | Ghi nhận và thực hiện |  |
| 3 | Nhân viên | Cung cấp tên và mật khẩu | Ghi nhận và thực hiện |  |
| 4 | Người dùng khác |  | Ghi nhận và thực hiện |  |

### **1.7. YÊU CẦU AN TOÀN**

#### **1.7.1. Danh sách các yêu cầu an toàn**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Nghiệp vụ** | **Đối tượng** | **Ghi chú** |
| 1 | Cập nhật thông tin bệnh nhân | Bệnh nhân hiện có |  |
| 2 | Xóa thông tin 1 bệnh nhân | Bệnh nhân hiện có |  |
| 3 | Phục hồi thông tin 1 bệnh nhân | Bệnh nhân hiện có |  |
| 4 | Cập nhật thông tin loại thuốc | Loại thuốc hiện có |  |
| 5 | Xóa thông tin 1 loại thuốc | Loại thuốc hiện có |  |
| 6 | Phục hồi thông tin 1 loại thuốc | Loại thuốc hiện có |  |

#### **1.7.2. Bảng trách nhiệm yêu cầu an toàn**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Nghiệp vụ** | **Người dùng** | **Phần Mềm** | **Ghi chú** |
| 1 | Cập nhật thông tin bệnh nhân | Cho biết bệnh nhân cần cập nhật và những thông tin mớicần cập nhật | Thực hiện theo đúng yêu cầu |  |
| 2 | Xóa thông tin 1 bệnh nhân | Cho biết bệnh nhân cần xóa thông tin | Thực hiện theo đúng yêu cầu |  |
| 3 | Phục hồi thông tin 1 bệnh nhân | Cho biết bệnh nhân và thông tin cần phục hồi | Thực hiện theo đúng yêu cầu |  |
| 4 | Cập nhật thông tin loại thuốc | Cho biết loại thuốc và thông tin mới cần cập nhật | Thực hiện theo đúng yêu cầu |  |
| 5 | Xóa thông tin 1 loại thuốc | Cho biết loại thuốc cần xóa | Thực hiện theo đúng yêu cầu |  |
| 6 | Phục hồi thông tin 1 loại thuốc | Cho biết loại thuốc cần phục hồi | Thực hiện theo đúng yêu cầu |  |

### **1.8 YÊU CẦU CÔNG NGHỆ**

#### **1.8.1 Danh sách các yêu cầu công nghệ**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Yêu cầu** | **Mô tả chi tiết** | **Ghi chú** |
| 1 | Dễ sửa lỗi | Xác định lỗi trung bình trong 15 phút | Khi sửa lỗi một chức năng thì chức năng khác không bị ảnh hưởng |
| 2 | Dễ bảo trì | Thêm chức năng mới nhanh | Không ảnh hưởng đến chức năng đã có |
| 3 | Tái sử dụng | Xây dựng chương trình quản lý thư viện trong thời gian ngắn | Cùng với các yêu cầu |
| 4 | Dễ mang chuyển | Đổi sang hệ quản trị cơ sở dữ liệu mới trong thời gian ngắn | Cùng với các yêu cầu |

# **CHƯƠNG II : GIỚI THIỆU MÔ HÌNH PHÁT TRIỂN PHẦN MỀM LINH HOẠT ( AGILE SOFT WARE DEVELOPMENT )**

## **ĐỊNH NGHĨA VỀ MÔ HÌNH AGILE**

**Agile** là một tập hợp các nguyên lý dành cho phát triển phần mềm, trong đó khuyến khích việc lập kế hoạch thích ứng, phát triển tăng dần, chuyển giao sớm, và cải tiến liên tục. Agile cũng chủ trương thích ứng nhanh chóng với các thay đổi

Agile không định nghĩa một phương pháp cụ thể để đạt được những điều này, nhưng lại có nhiều phương pháp phát triển phần mềm khác nhau thỏa mãn và hướng theo các tiêu chí đó.

Mục đích của các phương pháp Agile là giúp các doanh nghiệp đạt được sự linh hoạt (Agility), từ đó nâng cao sức cạnh tranh và phát triển bền vững. Các phương pháp Agile đã thay đổi diện mạo thế giới không chỉ trong Phát triển phần mềm mà còn đang thể hiện giá trị trong các lĩnh vực khác như Marketting (Agile Marketting), giáo dục (EduScrum, Lean Edu, v.v.), thiết kế (Lean UX, Design Thinking), khởi nghiệp (Lean Startup) và Phần cứng.

## **LỊCH SỬ HÌNH THÀNH MÔ HÌNH AGILE**

Trong các dự án, đặc biệt là các dự án phần mềm chúng ta sẽ gặp rất nhiều khó khăn trong việc thu thập đầy đủ và chính xác các yêu cầu của sản phẩm để lập kế hoạch tốt ngay từ đầu. Có quá nhiều vấn đề gây ảnh hưởng đến việc phát triển phần mềm. Trong khi đó có quá nhiều vấn đề mà chúng ta không lường trước được. Những vấn đề này có thể đến từ những yếu tố như kinh doanh, kỹ thuật, con người…

Tại thời điểm này giai đoạn trước những năm 90 của thế kỷ 20, trên thế giới xảy ra cuộc khủng hoảng về phương pháp phát triển phần mềm. Lí do của việc này đó là phương pháp truyền thống ngày càng bộc lộ nhiều nhược điểm và tỉ thệ các dự án bị thất bại quá cao. Có rất nhiều các cá nhân và công ty riêng lẻ đã tự tìm tòi và phát triển những phương pháp khác nhau để thích ứng với tình hình mới, ở đó những yếu tố môi trường kinh doanh và công nghệ thay đổi nhanh chóng, khiến cho phương pháp phát triển truyền thống không còn phù hợp. Những phương pháp riêng lẻ này phần nào giải quyết được một số vấn đề, nhưng lại nảy sinh những vấn đề khác về sự chia sẻ, cộng tác, các kỹ thuật, công cụ, sự mở rộng, hướng phát triển,…

Do đó, Tháng 2 năm 2001, 17 lập trình viên là đại diện cho những phương pháp phát triển mới này đã gặp nhau tại Utah. Họ đã đi đến thống nhất về quan điểm chung giữa các phương pháp và cho ra đời một tài liệu được gọi là: [Tuyên ngôn Phát triển Phần mềm Linh hoạt](http://hocvienagile.com/agipedia/tuyen-ngon-agile/) kèm với 12 nguyên lý phía sau. Đây chính là thời điểm mà thuật ngữ Agile được sử dụng hiện nay ra đời, mặc dù các phương pháp riêng lẻ thì đã có trước đó.

## **TUYÊN NGÔN VÀ CÁC NGUYÊN LÍ PHÍA SAU AGILE**

### ⮚ Tuyên ngôn về Agile:

1. “Cá nhân và sự tương tác hơn là quy trình và công cụ”
2. “Phần mềm chạy tốt hơn là tài liệu đầy đủ”
3. “Cộng tác với khách hàng hơn là đàm phán hợp đồng”
4. “Phản hồi với sự thay đổi hơn là bám theo kế hoạch”

#### Giải thích:

##### Cá nhân và sự tương tác hơn là quy trình và công cụ

Ý tưởng là đặt trọng tâm vào con người và sự tương hỗ giữa những thành viên trong nhóm. Cơ bản là nếu dự án có những thành viên có năng lực, chịu làm việc cùng nhau thì sẽ mang đến thành công cho dự án. Nếu dự án của bạn có quy trình làm việc tốt, được hỗ trợ những công cụ tốt nhất nhưng những thành viên không “cùng nhìn về một hướng” thì khả năng dự án thất bại là rất lớn. Nói điều này không có nghĩa là phủ nhận tầm quan trọng của quy trình và công cụ nhưng trong Agile nó được đặt sau yếu tố con người.

Quy trình là à các thủ tục cần thiết để phát triển dự án như thiết kế, sau đó đến coding, rồi test. Hay để xuất bản một chức năng nào đó cần phải có sự đồng ý của bộ phận đảm bảo chất lượng …. Quy trình này do mỗi công ty quy định và bắt  buộc các nhân viên khi tham gia vào dự án phải tuân thủ.  
Công cụ là phần mềm được sử dụng trong dự án như : Phần mềm quản lý công việc, phần mềm quản lý bug, phần mềm quản lý source code, phần mềm quản lý thời gian làm việc, phần mềm quản lý mail …. Có rất nhiều công cụ được sử dụng để một tổ chức vận hành đúng theo mục tiêu của công ty đặt ra.

##### Phần mềm chạy tốt hơn là tài liệu đầy đủ

Trong một số quy trình phát triển phần mềm, việc tạo ra và cập nhật các tài liệu về sản phẩm là bắt buộc. Nhóm Dev không thể hoặc không đồng ý tiến hành công việc nếu không có tài liệu đặc tả về yêu cầu, thiết kế hệ thống. Nhóm Test thì yêu cầu tài liệu về sản phẩm để có thể viết trường hợp kiểm thử và kiểm thử được. Nhóm QA đòi tất cả các tài liệu phải được viết trước khi sản phẩm được giao cho khách hàng nếu không thì không đủ điều kiện, chuẩn để giao sản phẩm cho khách hàng. Thực ra đứng với góc độ khách hàng thì khách hàng chỉ quan tâm đến sản phẩm có hoạt động được và tốt hay không. Trong khi việc tạo và cập nhật tài liệu mất nhiều thời gian và được cho là buồn tẻ. Vậy tại sao mình phải tập trung quá nhiều cho việc không cần thiết mà không dành thời gian đó để trao đổi để hiểu thêm về công việc phải làm. Mọi người đừng hiểu lầm là làm Agile là không viết tài liệu. Ý tưởng là chỉ viết những gì mà mọi người cần đọc.

Nhóm Dev là những lập trình viên có nhiệm vụ viết mã code cho sản phẩm.  
Nhóm Test là người thực hiện kiểm tra (test) các sản phẩm do các bạn Dev viết ra.  
Nhóm QA là người chịu trách nhiệm đảm bảo chất lượng sản phẩm (Quality Assurance).

##### Cộng tác với khách hàng hơn là đàm phán hợp đồng

“Khách hàng là thượng đế” hay “khách hàng luôn luôn đúng”. Tuy nhiên thì khách hàng có đủ loại khách hàng. Có khách hàng am hiểu về công nghệ, có người không. Có người suy nghĩ nhất quán có người thay đổi xoành xoạch, có người lạnh lùng có người cười nói suốt ngày, v.v và cách duy nhất để có thể làm việc tốt là phải cộng tác với khách hàng để hiểu được khách hàng muốn gì và cần gì để có thể tư vấn và điều chỉnh thay vì chỉ dựa vào những điều đã quy định trong hợp đồng.

Ví dụ về cộng tác là: Trao đổi và thảo luận với khách hàng về sự cần thiết có hay không của một chức năng trong sản phẩm, từ đó quyết định là có nên làm hay không.  
Ví dụ về đàm phán là: Nêu lý do đây là điều khoản đã được kí kết trong hợp đồng nên cần phải làm, mặc dù việc làm đó không mang lại giá trị cho người dùng.

##### Phản hồi với sự thay đổi hơn là bám theo kế hoạch

Có một điểm chung mà mình thấy trong hầu hết những dự án mình đã trải qua đó là không có dự án nào không có sự thay đổi điều chỉnh khi thực thi. Sự thay đổi đó có thể là thay đổi về yêu cầu, thay đổi công nghệ, thay đổi nhân sự, thay đổi deadline, thay đổi phương thức làm việc, v.v mặc dù kế hoạch đã được định ra rõ ràng từ đầu. Agile không khuyến khích cho sự thay đổi nhưng khuyến

Có một điều thú vị là đa số trong chúng ta đều cơ bản đồng ý với 4 tuyên ngôn của Agile. Nhiều người hiểu tầm quan trọng của “cá nhân” hay “cá nhân là tài sản quý giá nhất công ty” nhưng sẵn sàng thay đổi nhân lực để tương thích với quy trình/công cụ hiện có. Nhiều người hiểu “khách hàng là thượng đế” và “phải thích nghi với sự thay đổi” nhưng sẵn sàng tuyên bố “Dẹp, không làm nữa” vì khách hàng thay đổi yêu cầu liên tục.

Câu hỏi thường thấy là: Khách hàng thay đổi thoải mái, mô hình đáp ứng tối ưu…Nhưng sự thay đổi đó có được thảo luận giữa hai bên hay là do khách hàng ép? Khách hàng có chịu phí phát sinh cho các thay đổi thêm vào sản phẩm hay không? Đội dự án có dũng cảm và có quyền thảo luận với khách hàng về sự thay đổi hay không?

⮚ 12 nguyên lí phía sau Agile:

#### 1. Thoả mãn yêu cầu của khách hàng thông qua việc giao sản phẩm sớm và liên tục

“Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.”

Chúng ta hãy break nguyên tắc 1 này thành 3 phần và cùng bàn về chúng đó là: Khách hàng, giao sản phẩm sớm, có giá trị (valuable)

##### Khách hàng

Thoả mãn yêu cầu của khách hàng là ưu tiên cao nhất trong quản lý dự án agile. Việc nhận định khách hàng thuộc đối tượng nào, ở trong hay ngoài công ty không quá cần thiết. Điều quan trọng hơn ở đây là chúng ta cần biết dự án cần làm gì cho khách hàng và làm thế nào để tương tác một cách tốt nhất với khách hàng, thoả mãn với những gì mà họ cần. Điều đó có nghĩa là việc quản lý dự án agile không chỉ đơn thuần là hoàn thành list các việc được yêu cầu mà không quan tâm đến những gì họ cần, hay có ảnh hưởng gì đến khách hàng hay không.

##### Giao hàng sớm (early .... delivery)

Việc giao sản phẩm sớm là điều cần thiết. Đôi khi bạn muốn giữ lại vì phần đó chưa hoàn thành, muốn giành thêm thời gian để cải thiện nó và không muốn nhận những lời phê bình từ người khác. Tuy nhiên bạn cũng sẽ không muốn tốn thêm thời gian để làm những những thứ mà khách hàng không muốn đâu. Mà để khách hàng có thể confirm tiến độ công việc đang đúng như yêu cầu, có những feedback sớm thì con đường tốt nhất vẫn là việc chúng ta nên giao sản phẩm sớm cho khách hàng.

##### Có giá trị (valuable)

Dù chúng ta có đang quản lý cái gì đi nữa thì chúng cũng nên quy về cái cuối cùng đó là **có giá trị**. Nếu như khách hàng, end users không công nhận giá trị đó thì có lẽ nó không có giá trị gì với họ ngay từ lúc bắt đầu phát triển. Hoặc là nó đã không được phát triển theo đúng hướng để có thể tạo ra những giá trị, trải nghiệm cho khách hàng.

Đây là một nguyên tắc khá là **nặng nề** nhỉ, chắc vì thế mà nó là nguyên tắc đầu tiên, và nó yêu cầu **"ưu tiên cao nhất"**

#### 2. Chào đón việc thay đổi yêu cầu, thậm chí là những thay đổi yêu cầu muộn

“Welcome changing requirements, even late in development.Agile processes harness changes to the customer’s competitive advantage.”

Nguyên tắc số 2 là một sự thay đổi lớn cho những người đã quản lý hay làm việc với phương thức quản lý truyền thống.  
Nội dung của nó đã rất rõ ràng đó là chúng ta chào đón việc thay đổi yêu cầu từ khách hàng, kể cả khi mà phần lớn công việc đã được làm xong đi chăng nữa. Và chúng ta nên chấp nhận thay đổi với thái đội vui vẻ thay vì kêu ca phàn nàn. Điều này chưa từng nghe ở các phương thức quản lý ở project khác, nhưng nó là một phần cuộc sống của Agile ^^

Khách hàng đã nhận được sản phẩm sớm, thường xuyên. Do đó mà họ có thể nhìn thấy được tiến triển cũng như những vấn đề trong dự án. Vì vậy mà những thay đổi của họ đa phần là đều có ý nghĩa. Thay vì kêu ca phàn nàn chúng ta hãy tìm hiểu, bàn luận với khách hàng để xử lý những yêu cầu thay đổi một cách trơn tru nhất ^^

#### 3. Giao phần mềm chạy được cho khách hàng một cách thường xuyên (giao hàng tuần hơn là hàng tháng)

“Deliver working software frequently, from a couple of weeks toa couple of months, with a preference to the shorter timescale.”

Nguyên tắc số 3 focus vào sự cần thiết của tốc độ giao hàng. Khoảng thời gian giao sản phẩm cho khách hàng được quy định tuỳ theo độ lớn của project. Tuy nhiên,khoảng thời gian ngắn hơn vẫn luôn được ưa thích hơn cả. Nó đặc biệt quan trọng khi **chúng ta và khách hàng cần confirm để có những quyết định hay lựa chọn trong sản phẩm**. Khi đó việc thường xuyên kiểm tra sản phẩm là chìa khoá để dự án được phát triển tốt nhất. Càng gửi sớm chúng ta càng có cái để thảo luận với khách hàng và đi đến những quyết định cho dự án.

#### 4. Nhà kinh doanh và người phát triển phải cùng nhau làm việc hàng ngay trong suốt dự án

“Business people and developers must work together daily throughout the project.”

Nguyên tắc này được xây dựng dựa trên những nguyên tắc ở phía trên. Để nhận được những feedback sớm từ khách hàng không gì tốt hơn đó là team và khách hàng (ở đây là nhà kinh doanh) phải cùng nhau làm việc hằng ngày. Việc phải làm việc với nhau hằng ngày và nhận nhiều feedback có thể làm ta mệt mỏi một chút nhưng nó quan trọng vô cùng. Feedback tốt nhất cần được thực hiện bằng hội thoại mặt đối mặt (face-to-face- nguyên tắc số 6) hơn là những phương thức ngắt quãng khác như email, chat ...  
Để thực hiện được tốt việc đó, chúng ta cần thống nhất với khách hàng và đề ra một khoảng thời gian ngắn hàng ngày (timebox) để làm việc với nhau, và đặt mục tiêu (goal) trong cuộc hội thoại đó.  
Đối với công ty chúng ta với đặc thù out source phần mềm, khách hàng không thể ngồi làm việc trực tiếp với team. Khi đó hãy dùng các tool ad-hoc để nói chuyện hàng ngày như: skype, chatwork ...

#### 5. Các dự án được xây dựng xung quanh những cá nhân có động lực. Cung cấp cho họ môi trường và sự hỗ trợ cần thiết, và tin tưởng họ để hoàn thành công việc

“Build projects around motivated individuals. Give them theenvironment and support they need, and trust them to get the jobdone.”

Việc sử dụng người thông minh và năng động có hiệu quả hơn bất cứ công cụ hay quy trình nào khác. Khi bạn đã chọn con đường quản lý dự án bằng Agile, tức là bạn đang quản lý những con người hiểu biết mà mọi thứ đều được lấy ra từ bên trong họ ^^

Những kỹ sư thời nay không muốn bị quản lý một cách chi tiết bằng những thứ giấy tờ công việc thừa thãi, hay những bản kế hoạch quá chi tiết của dự án, hay là những chỉ thị ép buộc từ trên xuống. Hãy tạo ra khoảng không trong dự án để các cá thể có thể phát triển, suy nghĩ, nâng cao tay nghề. Tin tưởng và tự tin vào các thành viên trong team tạo nên sức mạnh. Hãy support họ những gì họ cần và tin tưởng họ sẽ thành công. Support ở đây có thể là tool và resource, hay kiến thức dự phòng ... Nó không chỉ là việc bạn tin tưởng vào team mà hãy làm thế nào để tất cả các thành viên trong team đều có được sự tự tin, niền tin khi mà họ đã có được đầy đủ nguồn lực, kiến thức cần thiết để hoàn thành dự án ^^

#### 6. Trao đổi trực tiếp mặt đối mặt là phương pháp hiệu quả nhất để truyền đạt thông tin

“The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.”

Nguyên tắc số 6 ở đây chính là nói về **nghệ thuật giao tiếp** ^^  
Mặc dù chúng ta đã trải qua biết bao nhiêu năm với sự phát triển của khoa học kỹ thuật, nhưng việc giao tiếp của chúng ta vẫn sử dụng một dạng cơ bản. Vâng, sau tất cả thì việc **giao tiếp mặt đối mặt** vẫn là cách chủ yếu để chúng ta giao tiếp hàng nghìn năm nay. Đó là **cách tốt nhất** để ta giao tiếp, và cũng là phương thức mà chúng ta **luyện tập nhiều nhất**.

Như đã nói ở các nguyên tắc trước, thời gian là **giới hạn**. Phương thức Agile được tạo nên nhờ 2 đặc điểm chính đó là nhanh và linh động do đó việc mặt đối mặt để truyền đạt thông tin là cách nhanh nhất. Việc giao tiếp bằng các viết sẽ tốn thời gian để chúng ta viết, đương nhiên rồi ^-^, và đôi khi chúng ta sẽ bị hiểu sai đi hoặc thiếu thông tin trong quá trình phiên dịch.

Team và khách hàng ngày nay càng trở nên phân tán (chia cách) về mặt vị trí nên việc mặt đối mặt trở nên khó khăn hơn. Tuy nhiên bạn hãy dùng các công cụ hỗ trợ để thực hiện nó nhé (truyền hình hội nghị, video call)

#### 7. Phần mềm chạy được là thước đo chính của tiến độ

“Working software is the primary measure of progress.”

Kể cả bạn đang quản lý dự án non-software thì cũng đừng bỏ qua nguyên tắc này bời vì nó nói đến software. Working ở đây chính là nói đến kết quả, đến việc mình tạo được những thứ mà có thể **đo lường** được hay nó hoàn thành được một vấn đề nào đó. Việc có các phần được hoàn thành không chỉ giúp mọi người có thể nhìn thấy được tiến độ mà còn là chất xúc tác cho những nguyên tắc khác, đặc biệt là việc sẽ có được những feedback hay hơn nữa là sự chấp nhận từ khách hàng cho những tính năng đó.  
Việc test những Working Model giúp team có được sự thống nhất trong giải pháp. Và nó dễ dàng hơn rất nhiều để đi đến những sự chấp thuận hay không chấp thuận, sự lựa chọn và thay đổi.  
Một điều chúng ta cần lưu ý trong nguyên tắc này đó là: Khi ta cần giành nhiều thời gian hơn để tạo ra working software thì khi đó ta cần dùng ít thời gian hơn cho việc viết tài liệu hay các bản design phức tạp. Chúng quan trọng tuy nhiên với đặc thù về sự thay đổi liên tục của agile project không cho phép chúng ta dùng quá nhiều thời gian để làm tài liệu, thứ mà có thể thay đổi ngay trong tương lai gần

#### 8. Agile Project khuyến khích phát triển bền vững, Khách hàng, team và người dùng nên duy trì được nhip độ phát triển một cách liên tục

“Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.”

Khi các dự án bắt đầu, chúng ta thường cảm thấy rất sung sức và tràn đầy năng nượng, và khao khát thể hiện thật tốt. Đặc biệt thường thấy ở những dự án mà cần độ cấp bách về tiến độ. Khi đó chúng ta sẽ đẩy áp lực làm việc mạnh lên để có được tiến độ dự án chạy một cách nhanh chóng. Nó có thể tạo ra những kết quả tốt ngắn hạn, nhưng sẽ phá hỏng kế hoạch dài hạn. Rõ ràng là không thực tế nếu kỳ vọng team sẽ làm việc với hiệu suất cao trong thời gian dài. Làm việc quá sức có thể tạo ra nhiều sai lầm và lỗi hơn. Khi con người làm việc quá tải và có ít thời gian để nghỉ ngơi, tập luyện thể dục hay hồi phục sức khoẻ, môi trường làm việc sẽ trở nên xấu đi, thậm chí là độc hại. Khi đó từng cá nhân sẽ trở nên kém kiên nhẫn và xung đột lẫn nhau, chúng ta sẽ khó có thể quản lý và giải quyết được các vấn đề. Do đó, một môi trường làm việc cân bằng giữa công việc và cuộc sống cần được lưu ý.  
Cần nhận thấy rằng Agile projects không chỉ cho những khoảng ngắn hạn mà còn cần thiết để thành công trong dài hạn. Nên việc thiết lập và duy trì nhịp độ phát triển chính là những gì nguyên tắc này muốn nói đến.

#### 9. Liên tục quan tâm đến kĩ thuật và thiết kế để cải tiến sự linh hoạt

“Continuous attention to technical excellence and good design enhances agility.”

Nguyên tắc số 9 nhắc nhở tất cả chúng ta rằng chúng ta cần xây dựng các dự án dựa trên những kiến thức về kỹ thuật. Điều này rất dễ được nhận thấy và được chú ý trong dự án phát triển phần mềm.

Việc có được những design , technique tốt và thường xuyên cải tiến chúng là việc làm mang tính quyết định. Nếu team chúng ta không thể nghĩ về bức tranh tổng thể lớn của dự án và nhận thấy những điều tốt nhất, con đường hiệu quả nhất để giải quyết các vấn đề hay design một chiến lược phát triển dự án thì bạn đã vô tình thiết lập cho bạn để làm rất nhiều những việc không cần thiết sau đó. Và có thể kết thúc với sự lãng phí rất lớn, thứ mà ta có thể tránh được.

Các dự án Agile hầu hết đều chưa được làm trước đó bao giờ nên chắc chắn là không thể chỉ có một con đường đúng đắn. Do đó chúng ta cần thử nghiệm, lựa chọn và refactor chúng. Refactor giống như là việc dọn dẹp nhà cửa, là cách để chúng ta làm mọi thứ trở nên tốt hơn, đơn giản và chắc chắn hơn. Nó có lẽ không phải là phần đẹp đẽ lôi cuốn trong coding nhưng dùng thời gian cho việc đó thực sự quan trọng. Khi làm việc trong môi trường agile, nếu chỉ nhìn về phía trước chúng ta sẽ đánh mất tầm nhìn, tổng quan về mọi thứ dự án. Cần dọn dẹp và tối ưu hoá, kể cả những thứ không thể nhìn thấy được bởi khách hàng.

#### 10. Sự đơn giản là cần thiết - nghệ thuật tối đa hoá lượng công việc chưa hoàn thành

“Simplicity — the art of maximizing the amount of work not done— is essential.”  
Ta có thể giải thích nguyên tắc này cũng theo một cách đơn giản đó là: Sau tất cả thì mục đính chính đó là việc **hoàn thành nhiều hơn trong thời gian ngắn hơn.**

Có một câu nói mà chúng ta thường được trích dẫn từ Leonardo da Vinci là "Sự đơn giản cũng là sự tinh tế tột cùng" (Simplicity is the ultimate sophistication.) hay là nguyên tắc KISS: “Keep It Simple, Silly” . Việc phát triển phần mềm hay những hệ thống bất kỳ đều nên đơn giản và trực giác.  
Lấy ví dụ đó là: bạn không nhất thiết phải mô tả một con bug dài lê thê chỉ để theo đúng định dạng của hệ thống bug trong khi bạn đã báo và trao đổi với Dev về con bug đó và họ đã sửa nó. Tuy nhiên hãy cẩn thận. Không phải bạn bỏ đi là bạn linh hoạt và tối ưu hóa công việc. Vấn đề là bạn bỏ đi những thứ không mang lại giá trị.

#### 11. Những bản kiến trúc, yêu cầu, thiết kế tốt nhất được tạo ra từ những team tự

“The best architectures, requirements, and designs emerge from self organizing teams.”

Đây là điểm khác biệt hoàn toàn với team trong quản lý dự án truyền thống. Trong quản lý dự án Agile, chúng ta làm việc với những member hiểu biết với trình độ và kinh nghiệm của họ, họ không cần chúng ta phải thúc ép hay chỉ dẫn nhỏ nhặt trong công việc. Những người làm việc hiểu biết hiện đại ngày nay, đặc biệt nếu là những người kinh nghiệm trong công việc sẽ đạt hiệu quả công việc tốt hơn nếu chúng ta tin tưởng trao cho họ quyền quyết định, sáng tạo. Các cá nhân vẫn cần người hướng dẫn, cần leader, nhưng lãnh đạo tốt nhất ở trong môi trường này đó là việc dẫn dắt bằng các ví dụ hay sự hướng dẫn trong quan hệ bình đẳng. Mà chúng ta có thể gọi là "lãnh đạo đầy tớ" (servent leadership)

Việc cho phép team tự tổ chức, cho phép họ làm việc gần gũi với nhau hơn, họ sẽ hiểu được giá trị của việc làm việc gắn kết chặt chẽ và học hỏi từ nhau. Một team gắn kết xuyên suốt dự án sẽ tạo ra những bản thiết kế và xử lý chúng một cách dễ dàng hơn. Nếu có lỗi xảy ra, team tự quản lý cũng dễ nhận biết nó và tìm giải pháp khác phục nhanh hơn.

Dĩ nhiên việc tự tổ chức không phải là một việc dễ dàng là việc tập hợp các thành viên và tuyên bố tự tổ chức. Đó là một quá trình phải được hướng dẫn, đào tạo để có thể tự tổ chức, và việc "tự tổ chức" đến đâu còn tuỳ thuộc vào nhiều yếu tố như năng lực, nhận thức của thành viên, khả năng của người hướng dẫn Agile, sự tin tưởng của ban lãnh đạo

“At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.”

Đây là nguyên tắc cuối cùng nên có lẽ nó nhắc nhở chúng ta cần phải thường xuyên nhìn lại và nghĩ về những giừ chúng ta đã làm tốt và chúng ta có thể làm gì để tốt hơn.

Tôi nghĩ rằng đây là điều mà chúng ta thường quên nhất. Chúng ta làm việc trong dự án, hoàn thành và chuyển sang dự án khác mà quên đánh giá mọi thứ đã đi như thế nào. Chúng ta có thể học được rất nhiều thứ khi chúng ta đánh giá những sai lầm và những thành công của chúng ta. Đó là một phần của cuộc sống và là một phần của dự án. Bạn rút ra được gì từ những lỗi, những sai lầm đó sẽ làm cho bạn, team của bạn, tổ chức của bạn mạnh hơn, tốt hơn và phù hợp hơn.

Để nhìn về phía trước, chúng ta cần nhìn lại. Nếu chúng ta không giành thời gian định kỳ để kiểm tra và đánh giá những gì đã làm, thì chúng ta sẽ không học được bài học gì cả. Tuy nhiên thì \*\*những bài học chúng ta rút ra được cũng cần được chia sẻ để mọi người cùng biết và đúc rút kinh nghiệm.

*tổ chức*

#### 12. Thường xuyên nhìn nhận đánh giá làm sao để hiệu quả hơn, từ đó thay đổi và thích ứng

“At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.”

Đây là nguyên tắc cuối cùng nên có lẽ nó nhắc nhở chúng ta cần phải thường xuyên nhìn lại và nghĩ về những giừ chúng ta đã làm tốt và chúng ta có thể làm gì để tốt hơn.

Tôi nghĩ rằng đây là điều mà chúng ta thường quên nhất. Chúng ta làm việc trong dự án, hoàn thành và chuyển sang dự án khác mà quên đánh giá mọi thứ đã đi như thế nào. Chúng ta có thể học được rất nhiều thứ khi chúng ta đánh giá những sai lầm và những thành công của chúng ta. Đó là một phần của cuộc sống và là một phần của dự án. Bạn rút ra được gì từ những lỗi, những sai lầm đó sẽ làm cho bạn, team của bạn, tổ chức của bạn mạnh hơn, tốt hơn và phù hợp hơn.

Để nhìn về phía trước, chúng ta cần nhìn lại. Nếu chúng ta không giành thời gian định kỳ để kiểm tra và đánh giá những gì đã làm, thì chúng ta sẽ không học được bài học gì cả. Tuy nhiên thì \*\*những bài học chúng ta rút ra được cũng cần được chia sẻ để mọi người cùng biết và đúc rút kinh nghiệm.

## **ĐẶC TRƯNG AGILE**

##### Tính lặp (Iterative)

Dự án sẽ được thực hiện trong các phân đoạn lặp đi lặp lại. Các phân đoạn (được gọi là Iteration hoặc Sprint) này thường có khung thời gian ngắn (từ 1 – 4 tuần). Trong mỗi phân đoạn này, nhóm phát triển thực hiện đầy đủ các công việc cần thiết như lập kế hoạch, phân tích yêu cầu, thiết kế, triển khai, kiểm thử (với các mức độ khác nhau) để cho ra các phần nhỏ của sản phẩm.

Các phương pháp Agile thường phân rã mục tiêu thành các phần nhỏ với quá trình lập kế hoạch đơn giản và gọn nhẹ nhất có thể, và không thực hiện việc lập kế hoạch dài hạn.

##### Tính tiệm tiến (Incremental) và tiến hóa (Evolutionary)

Cuối các phân đoạn, nhóm phát triển thường cho ra các phần nhỏ của sản phẩm cuối cùng. Các phần nhỏ này thường là đầy đủ, có khả năng chạy tốt, được kiểm thử cẩn thận và có thể sử dụng ngay (gọi là potentially shippable product increment of functionality). Theo thời gian, phân đoạn này tiếp nối phân đoạn kia, các phần chạy được này sẽ được tích lũy, lớn dần lên cho tới khi toàn bộ yêu cầu của khách hàng được thỏa mãn.

##### Tính thích ứng (hay thích nghi – adaptive)

Do các phân đoạn chỉ kéo dài trong một khoảng thời gian ngắn, và việc lập kế hoạch cũng được điều chỉnh liên tục, nên các thay đổi trong quá trình phát triển (yêu cầu thay đổi, thay đổi công nghệ, thay đổi định hướng về mục tiêu v.v.) đều có thể được đáp ứng theo cách thích hợp. Theo đó, các quy trình Agile thường thích ứng rất tốt với các thay đổi.

##### Nhóm tự tổ chức và liên chức năng

Cấu trúc nhóm Agile thường là liên chức năng (cross-functionality) và tự tổ chức (self-organizing). Theo đó, các nhóm này tự thực hiện lấy việc phân công công việc mà không dựa trên các mô tả cứng về chức danh (title) hay làm việc dựa trên một sự phân cấp rõ ràng trong tổ chức.

Nhóm tự tổ chức có nghĩa là nó đã đủ các kĩ năng (competency) cần thiết cho việc phát triển phần mềm, do vậy nó có thể được trao quyền để tự ra quyết định, tự quản lí và tổ chức lấy công việc của chính mình để đạt được hiệu quả cao nhất.

##### Quản lý tiến trình thực nghiệm (Empirical Process Control)

Các nhóm Agile ra các quyết định dựa trên các dữ liệu thực tiễn thay vì tính toán lý thuyết hay các tiền giả định (prescription). Nói cách khác, Agile rút ngắn vòng đời phản hồi (short feedback life cycle) để dễ dàng thích nghi và gia tăng tính linh hoạt. Theo thời gian, các chiến lược này sẽ tiến gần đến trạng thái tối ưu, nhờ đó nhóm có thể kiểm soát được tiến trình, và nâng cao năng suất lao động.

##### Giao tiếp trực diện (face-to-face communication)

Về yêu cầu của khách hàng, Agile khuyến khích nhóm phát triển trực tiếp nói chuyện với khách hàng để hiểu rõ hơn về cái khách hàng thực sự cần, thay vì phụ thuộc nhiều vào các loại văn bản.

Trong giao tiếp giữa nội bộ nhóm phát triển với nhau, thay vì một lập trình viên (thực hiện việc mã hóa) và một kĩ sư (thực hiện việc thiết kế) giao tiếp với nhau thông qua bản thiết kế.

##### Phát triển dựa trên giá trị (value-based development)

Một trong các nguyên tắc cơ bản của Agile là “phần mềm chạy tốt chính là thước đo của tiến độ”. Nguyên tắc này giúp nhóm dám loại bỏ đi các công việc dư thừa không trực tiếp mang lại giá trị cho sản phẩm.

Để vận hành được cơ chế “làm việc dựa trên giá trị”, nhóm Agile thường làm việc trực tiếp và thường xuyên với khách hàng (hay đại diện của khách hàng), cộng tác trực tiếp với họ để biết yêu cầu nào có độ ưu tiên cao hơn, mang lại giá trị hơn sớm nhất có thể cho dự án.

Nhờ đó các dự án Agile thường giúp khách hàng tối ưu hóa được giá trị của dự án. Một cách gần như trực tiếp, Agile gia tăng đáng kể độ hài lòng của khách hàng.

## **CÁC PHƯƠNG PHÁP CỦA AGILE**

Agile chỉ là một mô hình trên lý thuyết, để triển khai nó thì cần phải phân tích ra cụ thể, sau một thời gian dài, danh sách các phương pháp tăng lên và phát huy hiệu quả rõ rệt.

### Một số phương pháp phổ biến:

* Adaptive software development (ASD)
* Agile modeling
* Agile Unified Process (AUP)
* Crystal Clear methods
* Disciplined agile delivery
* Dynamic systems development method (DSDM)
* Extreme programming (XP)
* Feature-driven development (FDD)
* Lean software development
* [Kanban](https://vi.wikipedia.org/w/index.php?title=Kanban&action=edit&redlink=1)
* [Scrum](https://vi.wikipedia.org/wiki/Scrum_(m%C3%B4_h%C3%ACnh_ph%C3%A1t_tri%E1%BB%83n_ph%E1%BA%A7n_m%E1%BB%81m))
* Scrumban
* RAD(Rapid Application Development)

Trong đó, phương pháp Scrum là phổ biến nhất hiện nay, nhiều công ty tại Việt Nam cũng đang thử nghiệm mô hình này.

## **PHƯƠNG PHÁP SCRUM**

1. Khái niệm scrum :

Là một quy trình phát triển phần mềm theo mô hình linh hoạt (agile). Với nguyên tắc chủ đạo là chia nhỏ phần mềm cần sản xuất ra thành các phần nhỏ để phát triển (các phần nhỏ này phải đọc lập và Release được), lấy ý kiến khách hàng và thay đổi cho phù hợp ngay trong quá trình phát triển để đảm bảo sản phẩm release đáp ứng những gì khách hàng mong muốn. Scrum chia dự án thành các vòng lặp phát triển gọi là các sprint. Mỗi sprint thường mất 2- 4 tuần (30 ngày) để hoàn thành. Nó rất phù hợp cho những dự án có nhiều sự thay đổi và yêu cầu tốc độ cao.

1. Một số đặc điểm của SCRUM :

* Scrum (hay agile nói chung) được xếp vào nhóm “Feature-driven development”. Sản phầm được phát triển theo tính năng, chứ không phát triển sản phẩm theo kiến trúc hệ thống.
* Scrum khác với các mô hình Agile ở chỗ nó là mô hình hướng khách hàng (Customer oriented), vai trò của khách hàng trong việc đánh giá sản phẩm rất quan trọng. Chỉ sau mỗi sprint (2-4 tuần) khách hàng sẽ thấy được sự thay đổi của sản phẩm của mình qua đó đưa ra phản hồi sớm để định hướng.　-> Thích ứng nhanh với sự thay đổi yêu cầu
* Scrum giảm thiểu tài nguyên dành cho việc quản lý mà tập trung nhiều hơn cho những công việc liên quan trực tiếp đến việc làm ra sản phẩm. Bằng cách giảm vai trò quản lý (PM) bằng cách đẩy việc quản lý tới từng người,
* Giảm thời gian dành cho việc viết tài liệu bằng cách tăng thời gian trao đổi trực tiếp. Thông thường khi estimate công việc, thì team estimate cả thời gian dành cho communication để hoàn thành task đó nữa.
* Tập trung vào sản phẩm, sản phẩm mới là đích cuối cùng chứ không phải qui trình.

1. Ưu điểm, nhược điểm của mô hình Scrum

3.1 Ưu điểm:

* + Một người có thể làm nhiều việc ví dụ như dev có thể test
  + Phát hiện lỗi sớm hơn rất nhiều so với các phương pháp truyền thống
  + Khách hàng nhanh chóng thấy được sản phẩm qua đó đưa ra phản hồi sớm.
  + Có khả năng áp dụng được cho những dự án mà yêu cầu khách hàng không rõ ràng ngay từ đầu.

3.2 Nhược điểm:

* + Trình độ của nhóm là có một kỹ năng nhất định
  + Phải có sự hiểu biết về mô hình aglie .
  + Khó khăn trong việc xác định ngân sách và thời gian.
  + Luôn nghe ý kiến phản hồi từ khách hàng và thay đổi theo nên thời gian sẽ kéo dài khi có quá nhiều yêu cầu thay đổi từ khách hàng.
  + Vai trò của PO rất quan trọng, PO là người định hướng sản phẩm. Nếu PO làm không tốt sẽ ảnh hưởng đến kết quả chung

1. Các nhân tố cấu tạo lên 1 qui trình trong Scrum
   * Một cách đơn giản có 03 thành tố quan trọng cấu thành nên SCRUM: Tổ chức (Organization), Qui trình (Process),  Tài liệu (Atifacts). Trong mỗi thành tố có 03 thành tố con. Như vậy, chúng ta chỉ cần hiểu và áp dụng được 9 thành tố này là có thể áp dụng SCRUM.
   * Tổ chức (Organization): Tổ chức nhóm dự án và Roles (Vài trò)
     + Product Owner (Người sở hữu sản phẩm)
     + ScrumMaster (Người điều phối )
     + Development Team (Nhóm phát triển)
   * Tài liệu (Atifacts):  các kết quả đầu ra
     + Product Backlog  (Danh sách các chức năng cần phát triển của sản phẩm)
     + Sprint Backlog (Danh sách các chức năng cần phát triển cho mỗi giai đoạn)
     + Estimation (Kết quả ước lượng của Team)
   * Qui trình(Process):  Qui định cách thức vận hành của SCRUM
     + Sprint Planning meeting (Họp để hoạch định cho mỗi giai đoạn)
     + Sprint Review (Họp để tổng kết cho mỗi giai đoạn)
     + Daily Scrum Meeting (Họp review hàng ngày)

# **CHƯƠNG III : MÔ HÌNH HÓA HỆ THỐNG PHẦN MỀM**

## **PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG (OBJECT ORIENTED ANALYSIS AND DESIGN: OOAD)**

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

### Đặc trưng cơ bản

* + - * Tính bao bọc (encapsulation): quan niệm mối quan hệ giữa đối tượng nhận và đối tượng cung cấp thông qua khái niệm hộp đen. Nghĩa là đối tượng nhận chỉ truy xuất đối tượng cung cấp qua giao diện được định nghĩa bởi đối tượng cung cấp, Phân tích thiết kế hệ thống hướng đối tượng bằng UML đối tượng nhận không được truy cập đến các đặc trưng được xem là “nội bộ” của đối tượng cung cấp.
      * Tính phân loại (classification): gom nhóm các đối tượng có cùng cấu trúc và hành vi vào một lớp (class).
* Tính kết hợp (aggregation): kết hợp các đối tượng và các đối tượng cấu thành nó để mô tả cấu trúc cục bộ của đối tượng (ví dụ: toà nhà <-> phòng, xe <-> sườn xe, bánh xe,... ) , hoặc sự liên kết phụ thuộc lẫn nhau giữa các đối tượng.
* Tính thừa kế (heritage): phân loại tổng quát hoá và chuyên biệt hoá các đối tượng, và cho phép chia sẽ các đặc trưng của một đối tượng.

### Phân loại;

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

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

Có 2 cách tiếp cận riêng biệt:

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

### Ưu điểm

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

## **ĐỊNH NGHĨA UML**

**UML** (Tiếng Anh là Unified Modeling Language) là ngôn ngữ trực quan được dùng cho việc đặc tả, hình dung, xây dựng cho các hệ thống phần mềm. Cách tiếp cận theo mô hình UML giúp ích rất nhiều cho những người thiết kế cũng như những người sử dụng nó. UML tạo nên một cái nhìn bao quát và đầy đủ về hệ thống phần mềm dự định xây dựng.

UML là một tập các phần tử và một tập các quy tắc riêng. Các phần tử của UML là các đối tượng đồ họa như đường thẳng, hình chữ nhật và các quy tắc trong UML xác định các kết hợp giữa các phần tử này.

Trong UML, các đối tượng trao đổi với nhau bằng cách gửi các thông điệp cho nhau. Sự trao đổi này thể hiện sự tương tác giữa các đối tượng trong hệ thống. Mô hình động thái hệ thống được biểu diễn bằng các sơ đồ tương tác (Interaction Diagram)

- Sơ đồ lớp (Class Diagram): Class Diagram là xương sống của hầu như tất cả các phương pháp hướng đối tượng, bao gồm cả UML. Chúng mô tả các cấu trúc tĩnh của hệ thống.

- Sơ đồ tình huống sử dụng (*Use Cases Diagram*): *Use case diagram* mô tả sự tương tác đặc trưng giữa người dùng và hệ thống. Sơ đồ này thể hiện các ứng xử của hệ thống đối với bên ngoài, trong một hoàn cảnh nhất định, xét từ quan điểm của người sử dụng. Nó mô tả các yêu cầu đối với hệ thống, có nghĩa là những gì hệ thống phải làm chứ không phải mô tả hệ thống làm như thế nào. Tập hợp tất cả các sơ đồ tình huống sử dụng của hệ thống thể hiện tất cả các trường hợp mà hệ thống có thể được sử dụng. Một sơ đồ tình huống sử dụng có thể có những biến thể. Mỗi một biến thể được gọi là một kịch bản (scenario). Phạm vi của sơ đồ thường được giới hạn bởi các hoạt động mà người dùng thực hiện trên hệ thống trong một chu kì hoạt động để thực hiện một sự kiện nghiệp vụ.

Việc xây dựng các sơ đồ này nhằm mục đích mô hình hóa hoạt động của hệ thống, thực hiện việc gán trách nhiệm cho các đối tượng. Sơ đồ tương tác chỉ ra các tương tác, bao gồm tập đối tượng, quan hệ và các thông điệp trao đổi giữa chúng. Sơ đồ trình tự và sơ đồ tương tác chỉ ra từng bước của một luồng điều khiển cụ thể trong ca sử dụng. Từ sơ đồ tương tác, người thiết kế có thể phát hiện thêm các lớp, các thao tác cần thực hiện của mỗi lớp,... Do vậy, sơ đồ tương tác trở thành nền tảng cho các bước còn lại của quá trình phát triển phần mềm.

Không phải tất cả các hệ thống đều cần cả bốn sơ đồ tương tác trên mà nó còn tùy thuộc vào mức độ phức tạp của của hệ thống. Đối với những hệ thống tương đối đơn giản thì chỉ cần sơ đồ trình tự và sơ đồ trạng thái là đủ.

### Sơ đồ lớp:

Trong [kỹ thuật phần mềm](https://en.wikipedia.org/wiki/Software_engineering) , một sơ đồ lớp trong [Unified Modeling Language](https://en.wikipedia.org/wiki/Unified_Modeling_Language) (UML) là một loại sơ đồ cấu trúc tĩnh mô tả cấu trúc của một hệ thống bằng cách hiển thị các [lớp](https://en.wikipedia.org/wiki/Class_(computer_science)) , thuộc tính, hoạt động (hoặc các phương thức) của hệ thống , và các mối quan hệ giữa các đối tượng.

Sơ đồ lớp là khối xây dựng chính của mô hình [hướng đối tượng](https://en.wikipedia.org/wiki/Object-oriented_programming) . Nó được sử dụng cho [mô hình khái niệm](https://en.wikipedia.org/wiki/Conceptual_model) chung của hệ thống ứng dụng, và để mô hình chi tiết chuyển các mô hình thành [mã lập trình](https://en.wikipedia.org/wiki/Programming_code) . Sơ đồ lớp cũng có thể được sử dụng cho [mô hình hóa dữ liệu](https://en.wikipedia.org/wiki/Data_modeling) . [[1]](https://en.wikipedia.org/wiki/Class_diagram#cite_note-1) Các lớp trong một sơ đồ lớp đại diện cho cả các yếu tố chính, tương tác trong ứng dụng, và các lớp học được lập trình.

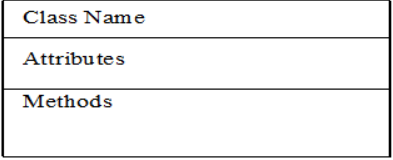
Trong sơ đồ, các lớp học được biểu diễn bằng các hộp chứa ba ngăn:

* Khoang trên cùng chứa tên của lớp. Nó được in đậm và trung tâm, và chữ cái đầu tiên được viết hoa.
* Khoang giữa chứa các thuộc tính của lớp. Chúng được căn lề trái và chữ cái đầu tiên là chữ thường.
* Ngăn dưới cùng chứa các hoạt động mà lớp có thể thực hiện. Chúng cũng được căn lề trái và chữ cái đầu tiên là chữ thường

Trong thiết kế của một hệ thống, một số lớp được xác định và nhóm lại với nhau trong một sơ đồ lớp để giúp xác định mối quan hệ tĩnh giữa chúng. Với mô hình chi tiết, các lớp học của thiết kế khái niệm thường được chia thành một số lớp con.

Các thành phần của sơ đồ lớp:

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



Trong đó,

–    Class Name: là tên của lớp.

–     Attributes (thuộc tính): mô tả tính chất của các đối tượng. Ví dụ như khách hàng có Mã khách hàng, Tên khách hàng, Địa chỉ, Ngày sinh v.v…

–      Method (Phương thức): chỉ các hành động mà đối tượng này có thể thực hiện trong hệ thống. Nó thể hiện hành vi của các đối tượng do lớp này tạo ra.

* Relationship (Quan hệ)

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

–          Association

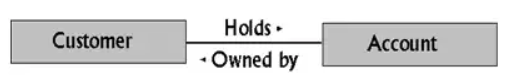
–          Aggregation

–          Composition

–          Generalization

***+ Association***

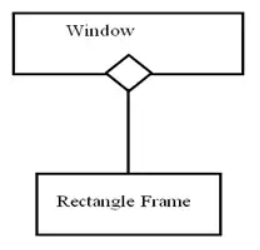
Association là quan hệ giữa hai lớp với nhau, thể hiện chúng có liên quan với nhau. Association thể hiện qua các quan hệ như “has: có”, “Own: sở hữu” v.v…



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

***+ Aggregation***

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



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

***+ Composition***

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

[](http://iviettech.vn/wp-content/uploads/2014/03/Composition1.png)

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

***+Generalization***

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

[](http://iviettech.vn/wp-content/uploads/2014/03/Generalization1.png)

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

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

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

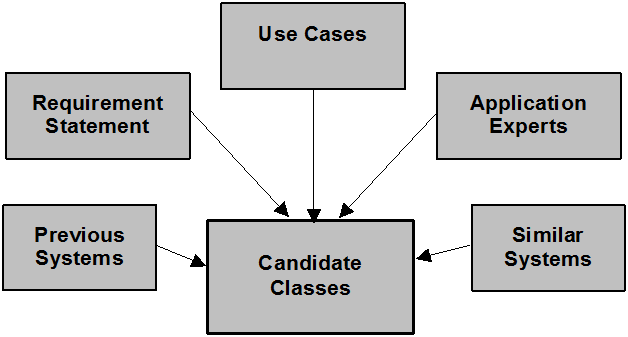
### Cách xây dựng bản vẽ Class

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

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

#### Bước 1: Tìm các Classes dự kiến

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

[](http://iviettech.vn/wp-content/uploads/2014/03/Class_Candidate.png)

**Hình 7. Các nguồn thông tin có thể tìm Class dự kiến**

–   **Requirement statement**:  Các yêu cầu. Chúng ta phân tích các danh từ trong các yêu cầu để tìm ra các thực thể.

–   **Use Cases**: Phân tích các Use Case sẽ cung cấp thêm các Classes dự kiến.

–   **Previous và Similar System:** có thể sẽ cung cấp thêm cho bạn các lớp dự kiến.

–   **Application Experts:** các chuyên gia ứng dụng cũng có thể giúp bạn.

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

–    **Customers**: khách hàng giao dịch là một thực thể có thật và quản lý trong hệ thống.

–    **Accounts**:  Tài khoản của khách hàng cũng là một đối tượng thực tế.

–    **ATM Cards**: Thẻ dùng để truy cập ATM cũng được quản lý trong hệ thống.

–    **ATM Transactions**: Các giao dịch được lưu giữ lại, nó cũng là một đối tượng có thật.

–    **Banks**: Thông tin ngân hàng bạn đang giao dịch, nếu có nhiều nhà Bank tham gia vào hệ thống bạn phải quản lý nó. Lúc đó Bank trở thành đối tượng bạn phải quản lý.

–   **ATM**: Thông tin ATM bạn sẽ giao dịch. Nó cũng được quản lý tương tự như Banks.

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

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

–   **Tìm thuộc tính**: phân tích thông tin từ các form mẫu có sẵn, bạn sẽ tìm ra thuộc tính cho các đối tượng của lớp. Ví dụ các thuộc tính của lớp Customer sẽ thể hiện trên Form đăng ký thông tin khách hàng.

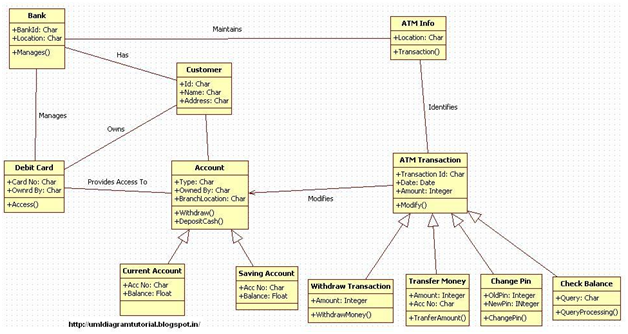
–   **Tìm phương thức**: phương thức là các hoạt động mà các đối tượng của lớp này có thể thực hiện. Chúng ta sẽ bổ sung phương thức đầy đủ cho các lớp khi phân tích Sequence Diagram sau này.

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

–    Phân tích các quan hệ giữa các lớp và định nghĩa các lớp phát sinh do các quan hệ sinh ra. Chúng ta phân tích các thực thể ở trên và nhận thấy.

* Lớp ***Accounts*** có thể chia thành nhiều loại tài khoản như ***Current Accounts***và ***Saving Accounts*** và có quan hệ thừa kế với nhau.
* Lớp ***ATM Transactions*** cũng có thể chia thành nhiều loại giao dịch như ***Deposit***, ***Withdraw, Transfer***v.v.. và chúng cũng có quan hệ thừa kế với nhau.

–   Tách chúng ta và vẽ chúng lên bản vẽ chúng ta sẽ có Class Diagram cho hệ thống ATM như sau:

[](http://iviettech.vn/wp-content/uploads/2014/03/ClassDiagram2.png)

**Hình 8. Ví dụ về Class Diagram cho hệ thống ATM**

### Đặc tả Class

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

–          Các thuộc tính: Tên, kiểu dữ liệu, kích thước

–          Các phương thức:

* + Tên
* + Mô tả
* + Tham số đầu vào: Tên, kiểu dữ liệu, kích thươcs
* + Kết quả đầu ra: Tên, kiểu dữ liệu, kích thước
* + Luồng xử lý
* + Điều kiện bắt đầu
* + Điều kiện kết thúc

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

### Sử dụng bản vẽ Class

Có thể tóm tắt một số ứng dụng của bản vẽ Class Diagram như sau:

–   Hiểu cấu trúc của hệ thống

–   Thiết kế hệ thống

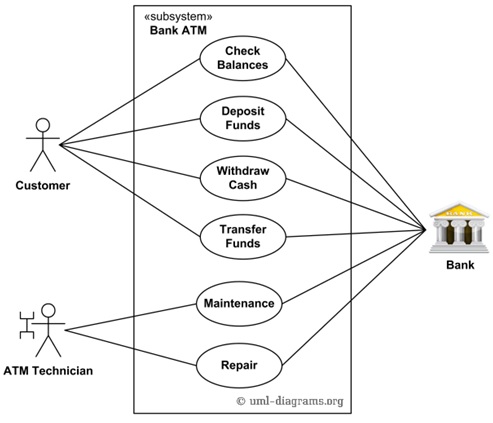
–   Sử dụng để phân tích chi tiết các chức năng (Sequence Diagram, State Diagram v.v…)

–   Sử dụng để cài đặt (coding)

* **USE CASE DIAGRAM**

#### 1. Các thành phần trong bản vẽ Use Case

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

[](http://iviettech.vn/wp-content/uploads/2014/01/UseCase-Diagram-1.jpg)

**Hình 1. Bản vẽ Use Case về ứng dụng ATM**

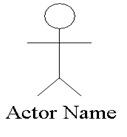
Nhìn bản vẽ này chúng ta thấy có hai người dùng là Customer và ATM Technician và một đối tượng sử dụng hệ thống là Bank. Bên cạnh đó nó mô tả các chức năng của hệ thống và người dùng nào dùng chức năng gì. Điều này giúp chúng ta hình dung được là chúng ta sẽ xây dựng hệ thống với những chức năng gì?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

Bây giờ chúng ta sẽ tìm hiểu kỹ hơn về các thành phần của bản vẽ.

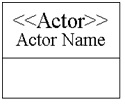
##### *1.1 Actor*

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

Actor được biểu diễn như sau:

[](http://iviettech.vn/wp-content/uploads/2014/01/Actor-1.jpg)

Hoặc

[](http://iviettech.vn/wp-content/uploads/2014/01/Actor-2.jpg)

**Hình 2. Các ký hiệu của Actor**

##### *1.2 Use Case*

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

[](http://iviettech.vn/wp-content/uploads/2014/01/Use-Case-Notation.jpg)

**Hình 3. Ký hiệu về Use Case**

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

##### *1.3 Relationship(Quan hệ)*

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

–          Association

–          Generalization

–          Include

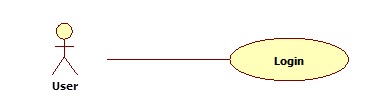
–          Extend

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

**+ Quan hệ Association**

[Association](http://iviettech.vn/wp-content/uploads/2014/01/Association.jpg)

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

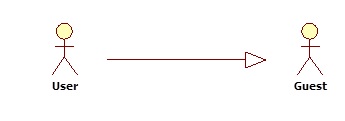
[](http://iviettech.vn/wp-content/uploads/2014/01/Use-Case-Association.jpg)

**Hình 4. Ví dụ thể hiện Actor User sử dụng Use Case Login**

**+Quan hệ  Generalization**

[Generalization](http://iviettech.vn/wp-content/uploads/2014/01/Generalization.jpg)

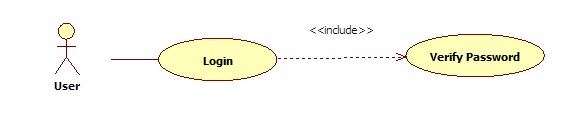
Generalization được sử dụng để thể hiện quan hệ thừa kế giữa các Actor hoặc giữa các Use Case với nhau.

[](http://iviettech.vn/wp-content/uploads/2014/01/Actor-Generation.jpg)  
**Hình 5. Ví dụ Actor User thừa kế toàn bộ quyền của Actor Guest**

**+ Quan hệ Include**

[](http://iviettech.vn/wp-content/uploads/2014/01/Include.jpg)

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

[](http://iviettech.vn/wp-content/uploads/2014/01/UseCase-Include.jpg)

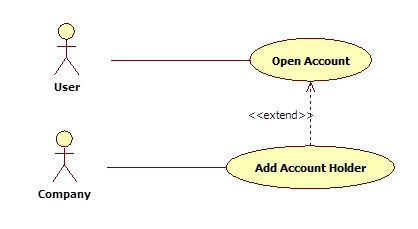
**Hình 6. Ví dụ về quan hệ Include giữa các Use Case**

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

**+ Quan hệ  Extend**

[](http://iviettech.vn/wp-content/uploads/2014/01/Extend.jpg)

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

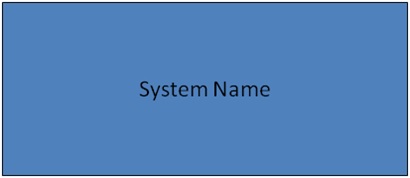
[](http://iviettech.vn/wp-content/uploads/2014/01/UseCase-Extend.jpg)

**Hình 7. Ví dụ về quan hệ Extend giữa các Use Case**

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

##### *1.4 System Boundary*

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

[](http://iviettech.vn/wp-content/uploads/2014/01/System.jpg)

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

#### 2. Các bước xây dựng Use Case Diagram

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

***+ Bước 1: Tìm các Actor***

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

–          Ai sử dụng hệ thống này?

–          Hệ thống nào tương tác với hệ thống này?

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

–          Ai sử dụng hệ thống?  -> Customer, ATM Technician

–          Hệ thống nào tương tác với hệ thống này? -> Bank

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

***+ Bước 2: Tìm các Actor***

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

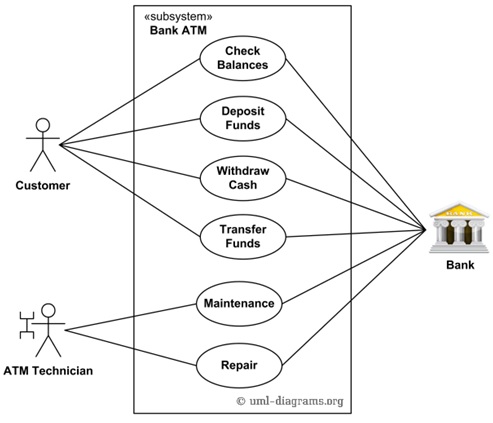
Xem xét ví dụ ở trên ta thấy:

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

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

***+ Bước 3: Xác định các quan hệ***

Phân tích và các định các quan loại hệ giữa các Actor và Use Case, giữa các Actor với nhau, giữa các Use Case với nhau  sau đó nối chúng lại chúng ta sẽ được bản vẽ Use Case.

[](http://iviettech.vn/wp-content/uploads/2014/01/UseCase-Diagram-1.jpg)

**Hình 8. Bản vẽ Use Case về ATM**

#### 3. Đặc tả Use Case

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

Có 2 cách để đặc tả Use Case.

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

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

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

**Cách 2**: **Sử dụng các bản vẽ để đặc tả**

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

#### 4. Sử dụng UseCase Diagram

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

–          Phân tích và hiểu hệ thống

–          Thiết kế hệ thống.

–          Làm cơ sở cho việc phát triển, kiểm tra các bản vẽ như Class Diagram, Activity Diagram, Sequence Diagram, Component Diagram.

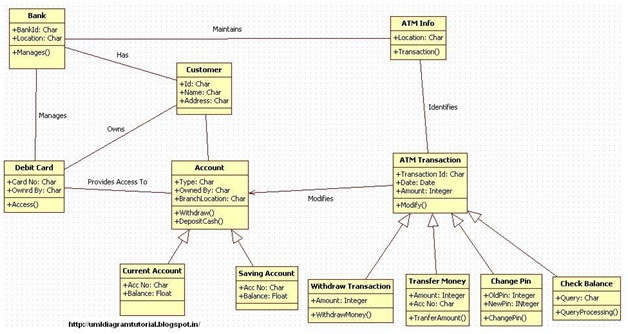
–          Làm cơ sở để giao tiếp với khách hàng, các nhà đầu tư.

–          Giúp cho việc kiểm thử chức năng, kiểm thử chấp nhận.

### **CLASS DIAGRAM**

#### 1. Các thành phần trong bản vẽ Class

Trước tiên, chúng ta xem một bản vẽ Class.

[](http://iviettech.vn/wp-content/uploads/2014/03/Class_Diagram1.jpg)

**Hình 1. Ví dụ về Class Diagram của ATM**

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

Classes (Các lớp)

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

[](http://iviettech.vn/wp-content/uploads/2014/03/Class_Notation1.png)

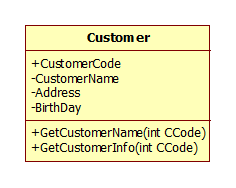
**Hình 2. Ký hiệu về Class**

Trong đó,

–          Class Name: là tên của lớp.

–          Attributes (thuộc tính): mô tả tính chất của các đối tượng. Ví dụ như khách hàng có Mã khách hàng, Tên khách hàng, Địa chỉ, Ngày sinh v.v…

–          Method (Phương thức): chỉ các hành động mà đối tượng này có thể thực hiện trong hệ thống. Nó thể hiện hành vi của các đối tượng do lớp này tạo ra.

[](http://iviettech.vn/wp-content/uploads/2014/03/Class_Notation2.png)

**Hình 3. Ví dụ về một Class**

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

#### 2. Relationship (Quan hệ)

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

–          Association

–          Aggregation

–          Composition

–          Generalization

Chúng ta sẽ lần lượt tìm hiểu về chúng.

***+ Association***

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

[Association 1](http://iviettech.vn/wp-content/uploads/2014/03/Association-1.png)

**Hình 4. Ví dụ về Association**

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

***+ Aggregation***

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

[](http://iviettech.vn/wp-content/uploads/2014/03/Aggregation1.png)

**Hình 5. Ví dụ về Aggregation**

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

***+ Composition***

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

[](http://iviettech.vn/wp-content/uploads/2014/03/Composition1.png)

**Hình 5. Ví dụ về Composition**

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

***+Generalization***

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

[](http://iviettech.vn/wp-content/uploads/2014/03/Generalization1.png)

**Hình 6. Ví dụ về Genelization**

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

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

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

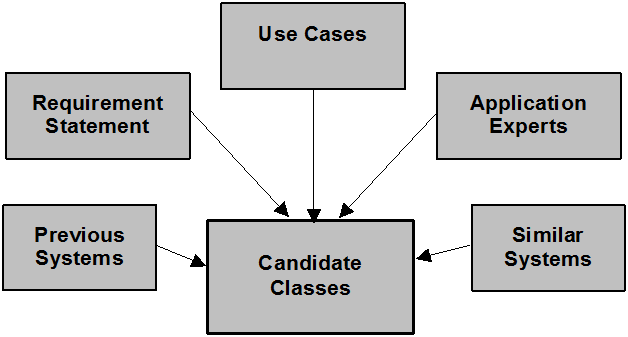
#### 3. Cách xây dựng bản vẽ Class

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

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

Bước 1: Tìm các Classes dự kiến

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

[](http://iviettech.vn/wp-content/uploads/2014/03/Class_Candidate.png)

**Hình 7. Các nguồn thông tin có thể tìm Class dự kiến**

–          **Requirement statement**:  Các yêu cầu. Chúng ta phân tích các danh từ trong các yêu cầu để tìm ra các thực thể.

–          **Use Cases**: Phân tích các Use Case sẽ cung cấp thêm các Classes dự kiến.

–          **Previous và Similar System:** có thể sẽ cung cấp thêm cho bạn các lớp dự kiến.

–          **Application Experts:** các chuyên gia ứng dụng cũng có thể giúp bạn.

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

–          **Customers**: khách hàng giao dịch là một thực thể có thật và quản lý trong hệ thống.

–          **Accounts**:  Tài khoản của khách hàng cũng là một đối tượng thực tế.

–          **ATM Cards**: Thẻ dùng để truy cập ATM cũng được quản lý trong hệ thống.

–          **ATM Transactions**: Các giao dịch được lưu giữ lại, nó cũng là một đối tượng có thật.

–          **Banks**: Thông tin ngân hàng bạn đang giao dịch, nếu có nhiều nhà Bank tham gia vào hệ thống bạn phải quản lý nó. Lúc đó Bank trở thành đối tượng bạn phải quản lý.

–          **ATM**: Thông tin ATM bạn sẽ giao dịch. Nó cũng được quản lý tương tự như Banks.

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

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

–          **Tìm thuộc tính**: phân tích thông tin từ các form mẫu có sẵn, bạn sẽ tìm ra thuộc tính cho các đối tượng của lớp. Ví dụ các thuộc tính của lớp Customer sẽ thể hiện trên Form đăng ký thông tin khách hàng.

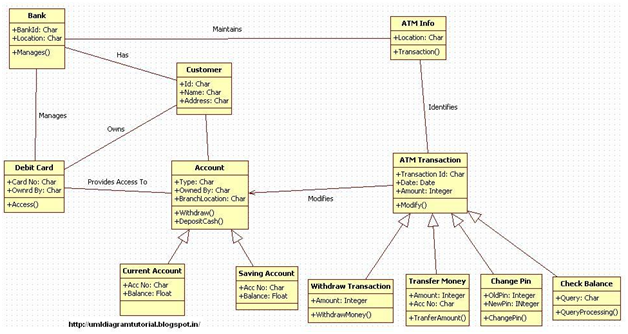
–          **Tìm phương thức**: phương thức là các hoạt động mà các đối tượng của lớp này có thể thực hiện. Chúng ta sẽ bổ sung phương thức đầy đủ cho các lớp khi phân tích Sequence Diagram sau này.

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

–          Phân tích các quan hệ giữa các lớp và định nghĩa các lớp phát sinh do các quan hệ sinh ra. Chúng ta phân tích các thực thể ở trên và nhận thấy.

* Lớp ***Accounts*** có thể chia thành nhiều loại tài khoản như ***Current Accounts***và ***Saving Accounts*** và có quan hệ thừa kế với nhau.
* Lớp ***ATM Transactions*** cũng có thể chia thành nhiều loại giao dịch như ***Deposit***, ***Withdraw, Transfer***v.v.. và chúng cũng có quan hệ thừa kế với nhau.

–          Tách chúng ta và vẽ chúng lên bản vẽ chúng ta sẽ có Class Diagram cho hệ thống ATM như sau:

[](http://iviettech.vn/wp-content/uploads/2014/03/ClassDiagram2.png)

**Hình 8. Ví dụ về Class Diagram cho hệ thống ATM**

#### 4. Đặc tả Class

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

–          Các thuộc tính: Tên, kiểu dữ liệu, kích thước

–          Các phương thức:

* + Tên
* + Mô tả
* + Tham số đầu vào: Tên, kiểu dữ liệu, kích thươcs
* + Kết quả đầu ra: Tên, kiểu dữ liệu, kích thước
* + Luồng xử lý
* + Điều kiện bắt đầu
* + Điều kiện kết thúc

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

#### 5. Sử dụng bản vẽ Class

Có thể tóm tắt một số ứng dụng của bản vẽ Class Diagram như sau:

–          Hiểu cấu trúc của hệ thống

–          Thiết kế hệ thống

–          Sử dụng để phân tích chi tiết các chức năng (Sequence Diagram, State Diagram v.v…)

–          Sử dụng để cài đặt (coding)

* **ACTIVITY DIAGRAM**

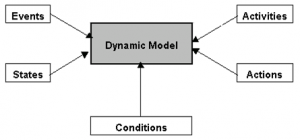
**Theo UML 2.0 thì hệ thống có thể được mô tả theo 2 mô hình tĩnh (Static Model) và mô hình động (Dynamic Model).**

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

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

Trong loạt bài này chúng ta chỉ bàn về hai bản vẽ của mô hình động được sử dụng thường xuyên trong thiết kế hệ thống phần mềm là Activity Diagram và Sequence Diagram. Các bản vẽ khác các bạn tự tìm hiểu hoặc sẽ được giới thiệu sau.

#### 1. Các thành phần cơ bản của Dynamic Model

[](http://iviettech.vn/wp-content/uploads/2014/04/Activity-Diagram1.png)

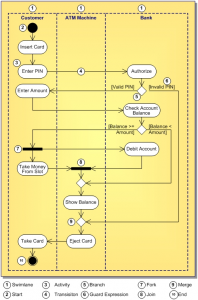
**Hình 1. Các thành phần của mô hình động**

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

#### 2. Activity Diagarm

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

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

[](http://iviettech.vn/wp-content/uploads/2014/04/Activity-Diagram2.png)

Hình 2. Ví dụ về Activity Diagram của hoạt động rút tiền từ ATM

**Chúng ta thấy chúng có các ký hiệu sau:**a.  Swimlance

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

[](http://iviettech.vn/wp-content/uploads/2014/04/Activity-Diagram3.png)

Hình 3. Ký hiệu về Swimlance      b. Nút Start, End

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

[Start](http://iviettech.vn/wp-content/uploads/2014/04/Start.png)  
Hình 4. Ký hiệu về nút Start

[](http://iviettech.vn/wp-content/uploads/2014/04/End.png)

Hình 5. Ký hiệu về nút kết thúc

c. Activity

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

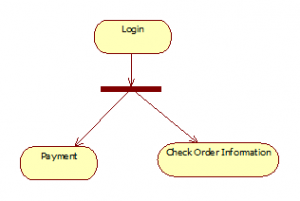
[](http://iviettech.vn/wp-content/uploads/2014/04/Activity.png)

Hình 4. Ký hiệu về Activityd. Branch

Branch thể hiện rẽ nhánh trong mệnh đề điều kiện.

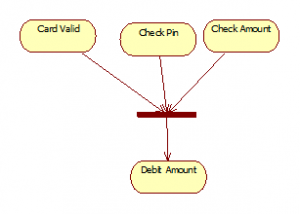
**[Branch](http://iviettech.vn/wp-content/uploads/2014/04/Branch.png)**e. Fork

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

**[](http://iviettech.vn/wp-content/uploads/2014/04/Fork.png)**

Hình 6. Ký hiệu về Forkf. Join

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

[](http://iviettech.vn/wp-content/uploads/2014/04/join.png)

Hình 7. Ký hiệu về Join

#### 3. Cách xây dựng Activity Diagram

Thực hiện các bước sau đây để xây dựng bản vẽ Activity Diagarm.

**Bước 1**: **Xác định các nghiệp vụ cần mô tả**

Xem xét bản vẽ Use Case  để xác định nghiệp vụ nào bạn cần mô tả.

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

**Bước 3**: **Xác định các hoạt động tiếp theo**

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

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

#### 4. Sử dụng bản vẽ Activity Diagram

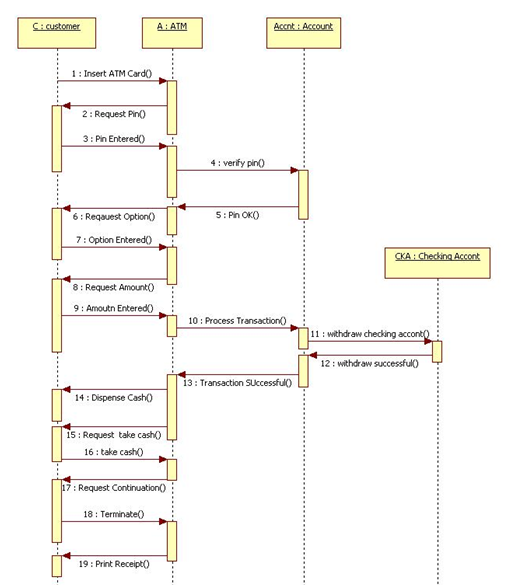
–          Phân tích nghiệp vụ để hiểu rõ hệ thống

–          Phân tích Use Case

–          Cung cấp thông tin để thiết kế bản vẽ Sequence Diagram

* **SEQUENCE DIAGRAM**

Sequence Diagarm là bản vẽ mô tả sự tương tác của các đối tượng để tạo nên các chức năng của hệ thống. Bản vẽ này mô tả sự tương tác theo thời gian nên rất phù hợp với việc sử dụng để thiết kế và cài đặt chức năng cho hệ thống phần mềm.  
Chúng ta hãy xem một ví dụ Sequence Diagram.

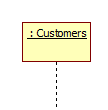
[](http://iviettech.vn/wp-content/uploads/2014/04/Sequence1.png)

**Hình 1. Ví dụ Sequence Diagram cho hoạt động rút tiền ở ATM**

#### 1. Các thành phần của Sequence Diagram

##### *a. Objects*

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

[](http://iviettech.vn/wp-content/uploads/2014/04/seq-Notation1.png)

**Hình 2. Ký hiệu về đối tượng trong bản vẽ sequence Diagram**

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

##### *b. Stimulus (message)*

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

[seq-Notation2](http://iviettech.vn/wp-content/uploads/2014/04/seq-Notation2.png)

**Hình 3. Ký hiệu về Stimulus trong bản vẽ Sequence Diagram**

##### *c. Axes*

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

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

#### 2. Xây dựng Sequence Diagram

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

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

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

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

**Bước 4**: Vẽ Sequence Diagarm

**Bước 5**: Cập nhật lại bản vẽ Class Diagram

#### 3. Ứng dụng Sequence Diagram

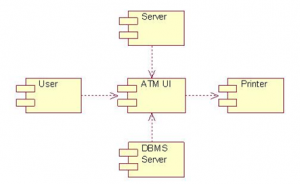
–          Thiết kế các chức năng

–          Kiểm chứng và bổ sung method cho các Class

–          Sử dụng trong việc coding các chức năng

### **COMPONENT DIAGRAM**

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

Component Diagram là bản vẽ cho biết cấu trúc của hệ thống theo thành phần phần mềm. Chúng ta xem một ví dụ về Component Diagram như sau:[](http://iviettech.vn/wp-content/uploads/2014/06/Component-Diagram1.png)

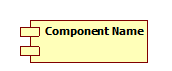
**Hình 1. Ví dụ về Component Diagram cho ứng dụng máy ATM**

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

#### 1. Các thành phần của Component Diagram

##### Component

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

[](http://iviettech.vn/wp-content/uploads/2014/06/Component-1.png)

**Hình 2. Ký hiệu về Component**

##### Component Dependence

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

[Component 2](http://iviettech.vn/wp-content/uploads/2014/06/Component-2.png)

**Hình 3. Ký hiệu về dependence**

#### 2. Xây dựng Component Diagram

Để xây dựng bản vẽ Component chúng ta thực hiện các bước sau:

**Bước 1**: Chia hệ thống thành những SubSystem

**Bước 2**: Xác định các thành phần và vẽ

#### 3. Thực hành xây dựng Component Diagram cho hệ thống eCommerce

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

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

–          Chia phần Website phục vụ cho đối tượng bên ngoài công ty là Guest và Customer ra một gói riêng để dễ triển khai và bảo mật. Thành phần này gọi là **EcommerceWeb**.

–          Phần Website phục vụ cho đối tượng bên trong công ty được chia thành một gói gọi là **ManagementWeb** .

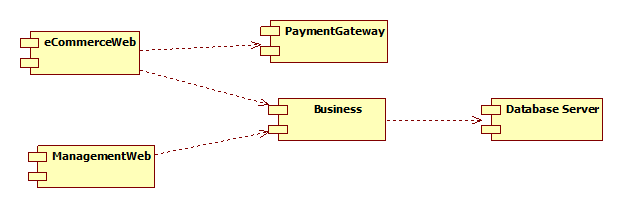
–          Phần **Bussiness** được sử dụng để tương tác CSDL và xử lý các nghiệp vụ.

–          Phần **PaymentGateway** để xử lý thanh toán trực tuyến.

–          Phần **Database Server** cũng được tách ra một gói riêng.

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

##### Bước 2:  Xác định quan hệ và vẽ ta được bản vẽ Component Diagram

[](http://iviettech.vn/wp-content/uploads/2014/06/Component-Diagram2.png)

**Hình 4. Component Diagram cho hệ thống eCommerce**

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

#### 4. Ứng dụng của Component Diagram

Component được sử dụng vào các công việc sau:

–          Thể hiện cấu trúc của hệ thống

–          Cung cấp đầu vào cho bản vẽ Deployment

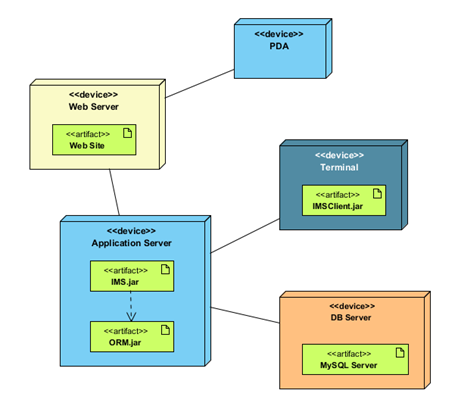
–          Hỗ trợ cho việc thiết kế kiến trúc phần mềm

### **DEPLOYMENT DIAGRAM**

**Deployment Diagram là bản vẽ giúp chúng ta xác định sẽ triển khai hệ thống phần mềm như thế nào. Đồng thời, xác định chúng ta sẽ đặt các thành phần phần mềm (component) lên hệ thống ra sao.**

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

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

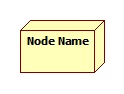
[](http://iviettech.vn/wp-content/uploads/2014/06/Deployment-Diagram-1.png)**Hình 1. Ví dụ về Deployment Diagram**

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

#### 1. Các thành phần của Deployment Diagram

Node

Node là một thành phần vật lý, nó có thể là thiết bị phần cứng hoặc một môi trường nào đó mà các thành phần phần mềm được thực hiện.

[](http://iviettech.vn/wp-content/uploads/2014/06/Node-1.jpg)**Hình 2. Ký hiệu về Node**

Relationship

Deployment Diagram sử dụng quan hệ Association và Dependence để thể hiện mối quan hệ giữa các node với nhau. Các ký hiệu của chúng như sau:

[Associate](http://iviettech.vn/wp-content/uploads/2014/06/Associate.jpg)

**Hình 3. Ký hiệu về Association**

[Component 2](http://iviettech.vn/wp-content/uploads/2014/06/Component-2.png)

**Hình 4. Ký hiệu về Dependence**

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

#### 2. Xây dựng Deployment Diagram

Thực hiện các bước sau đây để xây dựng bản vẽ Deployment Diagarm.

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

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

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

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

Bước 3: Xác định các quan hệ và hoàn tất bản vẽ

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

#### 3. Thực hành xây dựng Deployment Diagram cho hệ thống eCommerce

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

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

–          Để tăng cường an ninh và sức chịu đựng cho hệ thống chúng ta bố trí phần cho người dùng bên ngoài công ty (Guest và Customer) ra một Server riêng gọi là Web eCommerce Server.

–          Website chứa phần tương tác với nhân viên công ty đặt lên một Node riêng gọi là Web Management Server.

–          Phần Bussiness được đưa ra một Server ứng dụng gọi là Application Server.

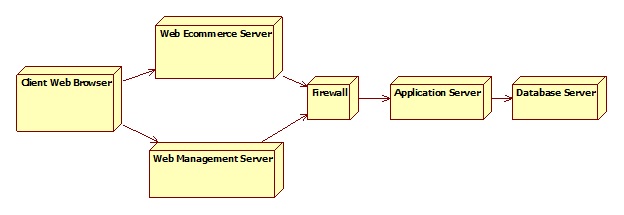
–          Database được đặt lên một Server gọi là Database Server.

–          Phần PaymentGateWay có thể được đặc trên Web eCommerce Server

–          Bổ sung thêm các thiết bị bảo mật và hạ tầng.

Bước 2: Xác định quan hệ giữa các thành phần và hoàn tất bản vẽ

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

[](http://iviettech.vn/wp-content/uploads/2014/06/Deployment-Diagram-2.jpg)**Hình 5. Bản vẽ deployment diagram cho hệ thống eCommerce**

Nếu theo mô hình này, hệ thống cần 4 Server và 01 thiết bị mạng là Firewall để triển khai nhằm đảm bảo an ninh, sức chịu đựng khi có đông người sử dụng và khả năng mở rộng hệ thống. Tuy nhiên, nếu hệ thống của bạn ít người sử dụng và cần tiết kiệm chi phí, bạn có thể triển khai trên một hoặc hai Server chứ không cần thiết phải triển khai trên nhiều Server như mô hình trên. Ngoài ra, bạn cũng có thể hiệu chỉnh lại để bản vẽ trên để trở nên hấp dẫn và chi tiết hơn như trên hình 1.

#### 4. Ứng dụng của Deployment Diagram

Deployment Diagram có thể ứng dụng vào các trường hợp sau:

–          Làm tài liệu để triển khai hệ thống.

–          Sử dụng trong thiết kế kiến trúc cho hệ thống.

–          Dùng trong giao tiếp với khách hàng, các nhà đầu tư.

## **ÁP DỤNG OOAD VÀO PHẦN MỀM QUẢN LÍ PHÒNG MẠCH TƯ SỬ DỤNG UML**

A screenshot of a cell phone

Description generated with high confidenceA screenshot of a cell phone

Description generated with high confidenceA close up of a device

Description generated with high confidenceA screenshot of a cell phone

Description generated with high confidenceA close up of a map

Description generated with very high confidence

* USE CASE DIAGRAM
* ClASS DIAGRAM
* ACTIVITY DIAGRAM
* SEQUENCE DIAGRAM
* COMPONENT DIAGRAM
* DEPLOYMENT DIAGRAM

# **CHƯƠNG IV: QUẢN LÍ CODE VỚI MÔ HÌNH 3 LỚP**

## **MÔ HÌNH 3 LỚP**

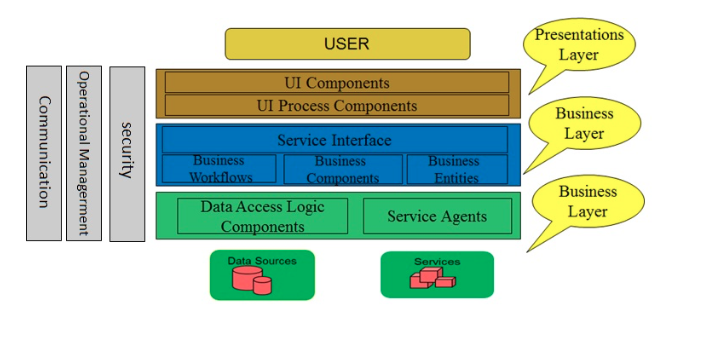
3-tiers là một kiến trúc kiểu client/server mà trong đó giao diện người dùng (UI-user interface), các quy tắc xử lý (BR-business rule hay BL-business logic), và việc lưu trữ dữ liệu được phát triển như những module độc lập, và hầu hết là được duy trì trên các nền tảng độc lập, và mô hình 3 tầng (3-tiers) được coi là một kiến trúc phần mềm và là một mẫu thiết kế.” (dịch lại từ wikipedia tiếng Anh).

Đây là kiến trúc triển khai ứng dụng ở mức vật lý .Kiến trúc gồm 3 module chính và riêng biệt :

* Tầng Presentation: hiển thị các thành phần giao diện để tương tác với người dùng như tiếp nhận thông tin, thông báo lỗi, …
* Tầng Business Logic: thực hiện các hành động nghiệp vụ của phần mềm như tính toán, đánh giá tính hợp lệ của thông tin, … Tầng này còn di chuyển, xử lí thông tin giữa 2 tầng trên dưới.
* Tầng Data: nơi lưu trữ và trích xuất dữ liệu từ các hệ quản trị CSDL hay các file trong hệ thống. Cho phép tầng Business logic thực hiện các truy vấn dữ liệu .

Mọi người vẫn hay nhầm lẫn giữa tier và layer vì cấu trúc phân chia giống nhau (presentation, bussiness , data). Tuy nhiên, thực tế chúng hoàn toàn khác nhau. Nếu 3 tiers có tính vật lí thì 3 layer có tính logic. Nghĩa là ta phân chia ứng dụng thành các phần (các lớp) theo chức năng hoặc vai trò một cách logic. Các layer khác nhau được thực thi trong 1 phân vùng bộ nhớ của process. Vì thế nên một tier có thể có nhiều layer.

### Giới thiệu mô hình 3-layer ( 3 lớp)



Mô hình 3-layer gồm có 3 phần chính :

– Presentation Layer (GUI) : Lớp này có nhiệm vụ chính giao tiếp với người dùng. Nó gồm các thành phần giao diện ( win form, web form,…) và thực hiện các công việc như nhập liệu, hiển thị dữ liêu, kiểm tra tính đúng đắn dữ liệu trước khi gọi lớp Business Logic Layer (BLL).

– Business Logic Layer (BLL) : Layer này phân ra 2 thành nhiệm vụ :

* Đây là nơi đáp ứng các yêu cầu thao tác dữ liệu của GUI layer, xử lý chính nguồn dữ liệu từ Presentation Layer trước khi truyền xuống Data Access Layer và lưu xuống hệ quản trị CSDL.
* Đây còn là nơi kiểm tra các ràng buộc, tính toàn vẹn và hợp lệ dữ liệu, thực hiện tính toán và xử lý các yêu cầu nghiệp vụ, trước khi trả kết quả về Presentation Layer.

– Data Access Layer (DAL) : Lớp này có chức năng giao tiếp với hệ quản trị CSDL như thực hiện các công việc liên quan đến lưu trữ và truy vấn dữ liệu ( tìm kiếm, thêm, xóa, sửa,…).

### Ưu điểm:

* Việc phân chia thành từng lớp giúp cho code được tường minh hơn. Nhờ vào việc chia ra từng lớp đảm nhận các chức năng khác nhau và riêng biệt như giao diện, xử lý, truy vấn thay vì để tất cả lại một chỗ. Nhằm giảm sự kết dính.
* Dễ bảo trì khi được phân chia, thì một thành phần của hệ thống sẽ dễ thay đổi. Việc thay đổi này có thể được cô lập trong 1 lớp, hoặc ảnh hưởng đến lớp gần nhất mà không ảnh hưởng đến cả chương trình.
* Dễ phát triển, tái sử dụng: khi chúng ta muốn thêm một chức năng nào đó thì việc lập trình theo một mô hình sẽ dễ dàng hơn vì chúng ta đã có chuẩn để tuân theo. Và việc sử dụng lại  khi có sự thay đổi giữa hai môi trường ( Winform sang Webfrom ) thì chỉ việc thay đổi lại lớp GUI.
* Dễ bàn giao. Nếu mọi người đều theo một quy chuẩn đã được định sẵn, thì công việc bàn giao, tương tác với nhau sẽ dễ dàng hơn và tiết kiệm được nhiều thời gian.
* Dễ phân phối khối lượng công việc. Mỗi một nhóm, một bộ phận sẽ nhận một nhiệm vụ trong mô hình 3 lớp. Việc phân chia rõ ràng như thế sẽ giúp các lập trình viên kiểm soát được khối lượng công việc của mình.

\* Phân tích chi tiết từng layer trong mô hình 3 lớp :

### https://static.techtalk.vn/wp-content/uploads/2017/03/Screen-Shot-2017-03-18-at-8.44.06-AM.png1.Presentation Layer (GUI):

Có hai thành phần chính sau đây với những tác vụ cụ thể :

* UI Components : gồm các thành phần tạo nên giao diện của ứng dụng (GUI). Chúng chịu trách nhiệm thu nhận và hiển thị dữ liệu cho người dùng… Ví dụ : textbox, button, combobox, …
* UI Process Components : là thành phần chịu trách nhiệm quản lý các quá trình chuyển đổi giữa các UI… Ví dụ : Sắp xếp quá trình kiểm tra thông tin khách hàng:
* 1.Hiển thị màn hình tra cứu ID
* 2.Hiển thị màn hình thông tin chi tiết khách hàng tương ứng
* 3.Hiển thị màn hình liên lạc với khách hàng.

### https://static.techtalk.vn/wp-content/uploads/2017/03/Screen-Shot-2017-03-18-at-8.44.32-AM.png2. Bussiness Layer (BLL) :

Lớp này gồm 4 thành phần:

* Service Interface : là thành phần giao diện lập trình mà lớp này cung cấp cho lớp Presentation sử dụng.
* Bussiness Workflows : chịu trách nhiệm xác định và điều phối các quy trình nghiệp vụ gồm nhiều bước và kéo dài. Những quy trình này phải được sắp xếp và thực hiện theo một thứ tự chính xác.
* Ví dụ : Thực hiện mua một đơn hàng trên tiki qua nhiều bước : kiểm tra gói hàng còn không?, tính tổng chi phí, cho phép giao dịch và sắp xếp việc giao hàng.
* Bussiness Components : chịu trách nhiệm kiểm tra các quy tắc nghiệp vụ, ràng buộc logic và thực hiện các công việc . Các thành phần này cũng thực hiện các dịch vụ mà Service Interface cung cấp và Business Workflows sẽ sử dụng nó.
* Ví dụ : Tiếp tục ví dụ ở trên. Bạn sẽ cần một Bussiness Component để kiểm tra gói hàng có khả dụng không ? hay một component để tính tổng chi phí,…
* Bussiness Entities : thường được sử dụng như Data Transfer Objects ( DTO ) . Bạn có thể sử dụng để truyền dữ liệu giữa các lớp (Presentation và Data Layer). Chúng thường là cấu trúc dữ liệu ( DataSets, XML,… ) hay các lớp đối tượng đã được tùy chỉnh.
* Ví dụ : tạo 1 class Student lưu trữ các dữ liệu về tên, ngày sinh, ID, lớp.

### https://static.techtalk.vn/wp-content/uploads/2017/03/Screen-Shot-2017-03-18-at-8.46.12-AM.png 3. Data Layer (DAL) :

* Data Access Logic Components : chịu trách nhiệm chính lưu trữ và truy xuất dữ liệu từ các nguồn dữ liệu (Data Sources) như XML, file system,… Hơn nữa còn tạo thuận lợi cho việc dễ cấu hình và bảo trì.
* Service Agents : giúp bạn gọi và tương tác với các dịch vụ từ bên ngoài một cách dễ dàng và đơn giản.

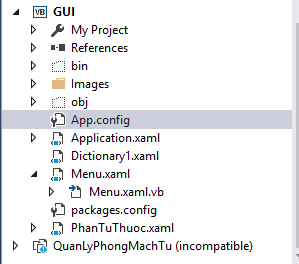
Cấu trúc mô hình 3 lớp

Để hiểu rõ hơn về cấu trúc và cách xây dựng của mô hình 3 lớp, chúng ta cùng tham khảo một ví dụ về mô hình quản lí công nhân

gồm các lớp BUS, DAO, GUI. (Các đoạn code sẽ bị lược bỏ bớt )

1. **ỨNG DỤNG MÔ HÌNH 3 LỚP VÀO PHẦN MỀM**
   1. Cấu trúc

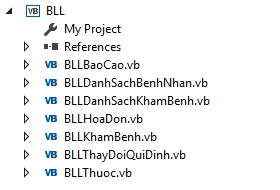
Lớp Layer (GUI):

Hình ảnh:

Phân tích:

|  |  |
| --- | --- |
| Tên lớp | Ý nghĩa |
| menu |  |
| phantuthuoc |  |

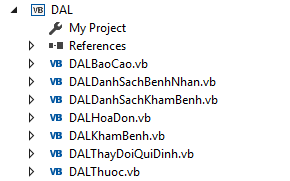
Lớp Bussiness (BLL)

****Hình ảnh:

Phân tích:

|  |  |
| --- | --- |
| Tên lớp | Ý nghĩa |
| BLLBaoCao |  |
| BLLDanhSachBenhNhan |  |
| BLLDanhSachKhamBenh |  |
| BLLHoaDon |  |
| BLLKhamBenh |  |
| BLLThayDoiQuiDinh |  |
| BLLThuoc |  |

Lớp Data (DAL)

****Hình ảnh:

Phân tích:

|  |  |
| --- | --- |
| Tên lớp | Ý nghĩa |
| DALBaoCao |  |
| DALDanhSachBenhNhan |  |
| DALDanhSachKhamBenh |  |
| DALHoaDon |  |
| DALKhamBenh |  |
| DALThayDoiQuiDinh |  |
| DALThuoc |  |