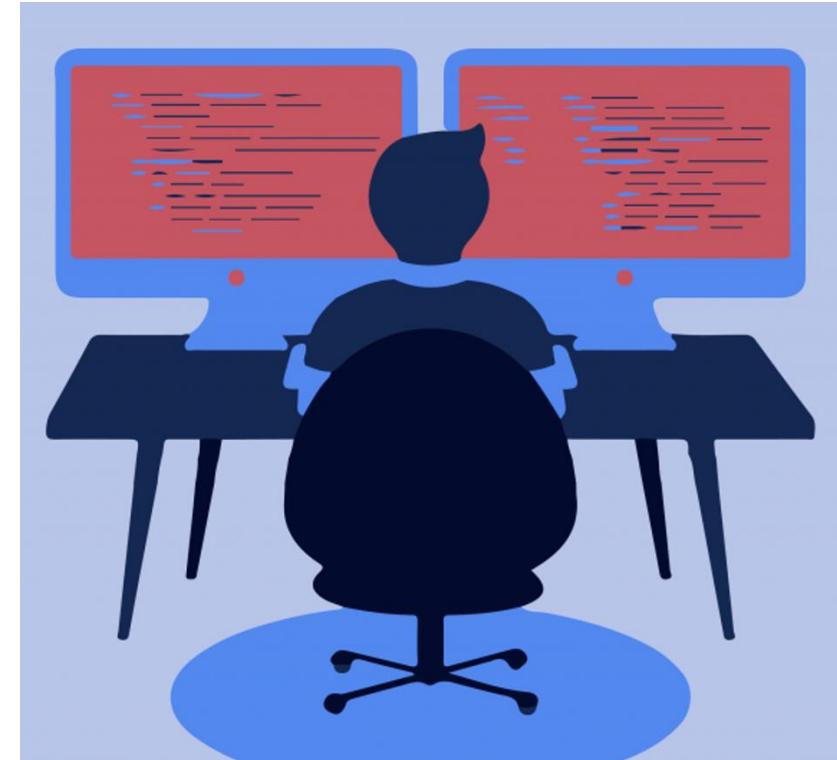


NHẬP MÔN CÔNG NGHỆ PHẦN MỀM

INTRODUCTION TO SOFTWARE ENGINEERING (SWE)

1. Mã học phần: INT1340
2. Số tín chỉ: 3
3. Loại môn học: Bắt buộc.
4. Hoạt động học tập: 45 giờ
 - Nghe giảng lý thuyết: 36 giờ
 - Chữa bài trên lớp: 04 giờ
 - Bài tập/Thảo luận: 04 giờ
 - Dự án: 0 giờ
 - Tự học: 01 giờ



characteristics of software

Lecturer: Chau Van Van|IT Faculty 2

✉: vancv@ptithcm.edu.vn

☎: 0918080300

3/10/2025 9:17 AM

Object-Oriented and Classical Software Engineering

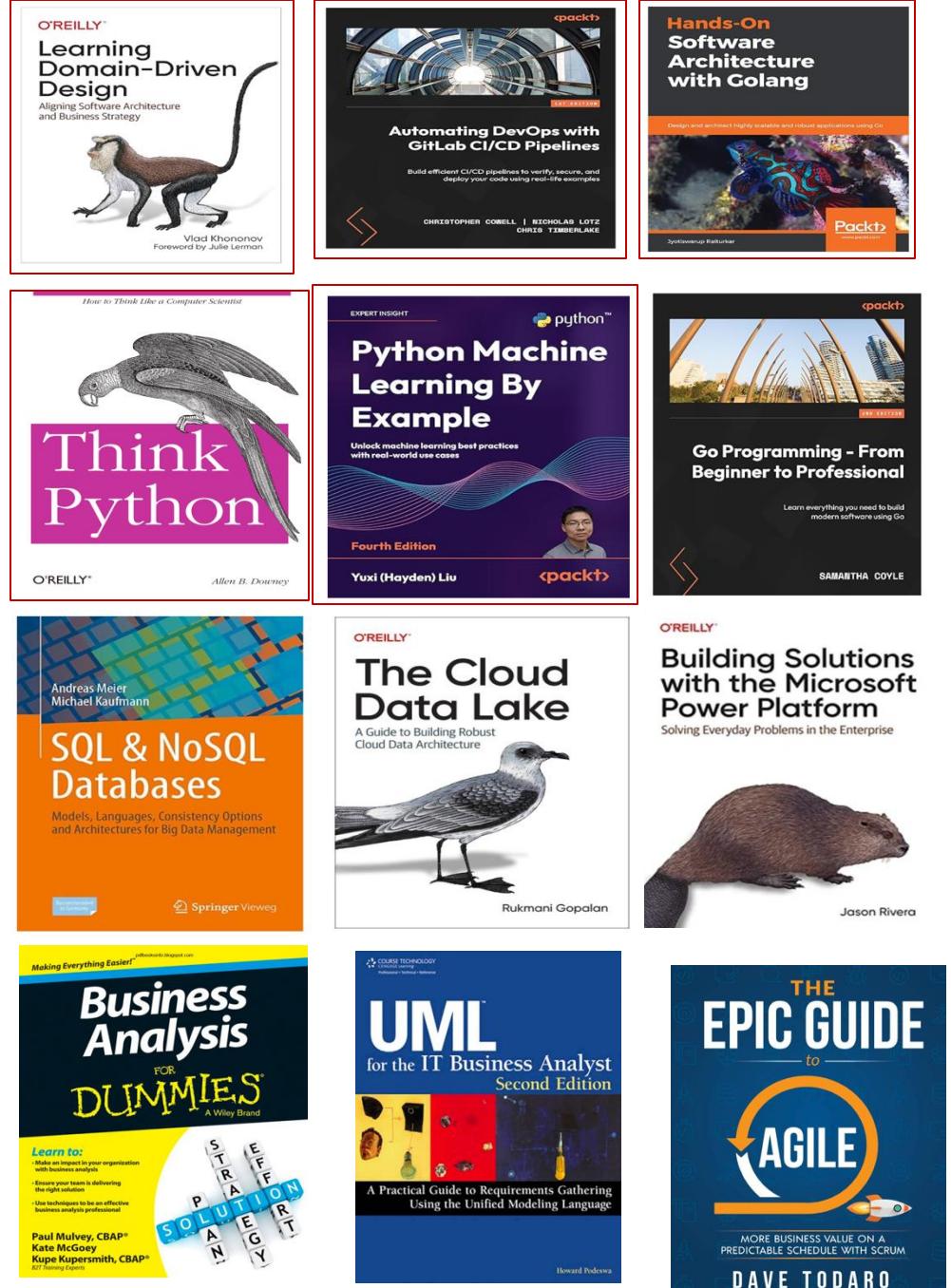
Eighth Edition



Stt	Book title	Authors	Publication Year	Học liệu
1	Object-Oriented and Classical Software Engineering. 8 th edition, McGraw Hill	Stephen R. Schach.	2010	Bắt buộc
2	Giáo trình công nghệ phần mềm, NXB Hà Nội	Nguyễn Xuân Huy	1994	Tham khảo
3	Giáo trình CNPM dành cho SV Ngành CNTT và các trường ĐH, NXB KHKT	Lê Đức Trung	2002	Tham khảo
4	Design Patterns: Elements of Reusable Object-Oriented Software	Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides	1994	Tham khảo
5	Code Complete: A Practical Handbook of Software Construction	Steve McConnell	1993	Tham khảo
6	The Pragmatic Programmer: Your Journey to Mastery	Andrew Hunt, David Thomas	1999	Tham khảo
7	Object-Oriented Software Construction	Bertrand Meyer	1988	Tham khảo
8	Clean Code: A Handbook of Agile Software Craftsmanship	Robert C. Martin	2008	Tham khảo
9	Design Patterns: Elements of Reusable Object-Oriented Software	Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides	1994	Tham khảo
10	Code Complete: A Practical Handbook of Software Construction	Steve McConnell	1993	Tham khảo

12 BEST SOFTWARE ENGINEERING BOOKS FOR DEVELOPERS IN 2024

No.	Book Title	Author(s)
1	Clean Code	Robert C. Martin
2	Design Patterns	Gang of Four (Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides)
3	The Pragmatic Programmer	Andrew Hunt, David Thomas
4	Code Complete	Steve McConnell
5	Refactoring	Martin Fowler
6	Introduction to Algorithms	Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
7	The Clean Coder	Robert C. Martin
8	Domain-Driven Design	Eric Evans
9	Continuous Delivery	Jez Humble, David Farley
10	Building Microservices	Sam Newman
11	Designing Data-Intensive Applications	Martin Kleppmann
12	Cracking the Coding Interview	Gayle Laakmann McDowell



<https://daily.dev/blog/12-best-software-engineering-books-for-developers-in-2024>



SWE | LỊCH TRÌNH CHUNG

Stt	Nội dung	Lý thuyết	BT/TL	Kiểm tra	Thực hành	Tổng cộng
1	Chương 1: Phạm vi công nghệ phần mềm	2	-	-	-	2
2	Chương 2: Tiến trình phần mềm	4	-	-	-	4
3	Chương 3: Một số mô hình vòng đời phát triển	4	-	-	-	4
4	Chương 4: Vấn đề kiểm thử, lập kế hoạch và tài liệu	4	-	-	-	4
5	Chương 5: Lấy yêu cầu	4	-	-	1	5
6	Kiểm tra giữa kỳ	-	-	2	-	2
7	Chương 6: Phân tích	6	-	-	-	6
8	Chữa bài tập – Phân tích	-	4	-	2	6
9	Chương 7: Thiết kế	4	-	-	-	4
10	Chữa bài tập – Thiết kế	-	2	-	-	2
11	Chương 8: Cài đặt và kiểm thử	4	-	-	-	4
12	Chữa bài tập – Viết test case	-	2	-	-	2
13	Ôn tập và giải đáp môn học	-	2	-	-	2
14	Tổng cộng	34	8	2	1	45



SWE| CHÍNH SÁCH ĐỐI VỚI MÔN HỌC VÀ KIỂM TRA ĐÁNH GIÁ ĐỊNH KỲ

- Mỗi nội dung chữa bài tập, hoặc thực hành được chia theo nhóm, mỗi nhóm không quá 30 sinh viên.
- Thiếu một điểm thành phần (bài tập, bài kiểm tra giữa kỳ), hoặc nghỉ quá 20% tổng số giờ của môn học, không được thi hết môn.

Stt	Hình thức kiểm tra	Tỷ lệ đánh giá	Đặc điểm đánh giá	Hình thức thi
1	Đi học đầy đủ (trong lớp gây ảnh hưởng đến người khác, mỗi lần nhắc nhở trừ một điểm, mỗi buổi nghỉ học trừ một điểm). Tích cực thảo luận (không phát biểu buổi nào sẽ được 0 điểm, phát biểu 1 buổi sẽ được 4 điểm, sau đó số buổi học có phát biểu tăng lên 1 thì điểm tăng lên 1)	10%	Cá nhân	
2	Trung bình các điểm bài tập lớn	20%	Cá nhân/Nhóm	
3	Trung bình các bài kiểm tra trên lớp	20%	Cá nhân	
4	Kiểm tra cuối kỳ	50%	Cá nhân	Đồ án môn học

Stt	Nội dung	W01	W02	W03	W04	W05	W06	W07	W08	W09	W10	W11
1	Phạm vi công nghệ phần mềm											
2	Tiến trình phần mềm											
3	Một số mô hình vòng đời phát triển											
4	Vấn đề kiểm thử, lập kế hoạch và tài liệu											
5	Lấy yêu cầu											
6	Kiểm tra giữa kỳ											
7	Phân tích											
8	Chữa bài tập – Phân tích											
9	Thiết kế											
10	Chữa bài tập – Thiết kế											
11	Cài đặt và kiểm thử											
12	Chữa bài tập – Viết test case											
13	Ôn tập và giải đáp môn học											

***Ghi chú:** Lịch trình sẽ có sự thay đổi/điều chỉnh phù hợp và linh hoạt trong suốt quá trình dạy và học

CHƯƠNG 0 – GIỚI THIỆU TỔNG QUÁT

1. HOẠT ĐỘNG CHÍNH TRONG MÔN HỌC.
2. CƠ HỘI NGHỀ NGHIỆP VÀ CÁC VỊ TRÍ LIÊN QUAN
TẠI VIỆT NAM VÀ THẾ GIỚI.

Introduction to Software Engineering

- Các hoạt động chính trong môn học này tập trung vào việc cung cấp nền tảng cơ bản về kỹ thuật phần mềm, bao gồm:

- 1. Giới thiệu về kỹ thuật phần mềm:** Khái niệm phần mềm, vai trò của kỹ sư phần mềm, lịch sử phát triển.
- 2. Mô hình phát triển phần mềm:** Waterfall, Agile, Scrum, Spiral Model.
- 3. Thu thập và phân tích yêu cầu phần mềm:** Xác định nhu cầu của khách hàng và viết tài liệu yêu cầu.
- 4. Thiết kế phần mềm cơ bản:** Sử dụng UML (Use Case Diagram, Class Diagram).
- 5. Lập trình và kiểm thử phần mềm:** Viết mã nguồn, kiểm thử đơn vị (**Unit Test**).
- 6. Quản lý dự án phần mềm sơ cấp:** Nguyên tắc cơ bản về quản lý nhóm, tiến độ và tài liệu dự án.

Software Engineering

- Các hoạt động chính tập trung vào các kỹ thuật nâng cao, quy trình phát triển chuyên nghiệp và tối ưu hóa phần mềm:

- 1. Quản lý vòng đời phần mềm:** Agile, DevOps, CI/CD, quản lý yêu cầu và phiên bản phần mềm.
- 2. Phát triển kiến trúc phần mềm:** Thiết kế hệ thống theo mô hình MVC, Microservices, kiến trúc hướng dịch vụ (SOA).
- 3. Kiểm thử và đảm bảo chất lượng phần mềm:** Tích hợp kiểm thử tự động, kiểm thử bảo mật (Security Testing).
- 4. Bảo trì và nâng cấp phần mềm:** Cải tiến mã nguồn, tối ưu hóa hiệu suất, cập nhật phần mềm.
- 5. Quản lý dự án phần mềm chuyên sâu:** Agile Project Management, Jira, quản lý rủi ro phần mềm.
- 6. Phát triển phần mềm quy mô lớn:** Xây dựng hệ thống phân tán, phần mềm nhúng, điện toán đám mây.
- 7. Ứng dụng AI trong phát triển phần mềm:** Xây dựng phần mềm thông minh, tích hợp học máy (Machine Learning).

□ CƠ HỘI NGHỀ NGHIỆP

- **Introduction to Software Engineering** giúp sinh viên mới bắt đầu có cái nhìn tổng quan, phù hợp với các vị trí thực tập, phát triển phần mềm cơ bản.
- **Software Engineering** giúp nâng cao chuyên môn, mở ra cơ hội làm việc tại các công ty công nghệ lớn, quản lý dự án phần mềm hoặc nghiên cứu AI.

□ CÁC VỊ TRÍ NGHỀ NGHIỆP PHỔ BIẾN (TẠI VIỆT NAM)

1. Lập trình viên phần mềm (Software Developer)

- Phát triển ứng dụng web, mobile hoặc phần mềm nhúng.
- Các công ty tuyển dụng: FPT Software, VNG, VCCorp, TMA Solutions.

2. Kỹ sư kiểm thử phần mềm (Software Tester/QA Engineer)

- Kiểm thử phần mềm, viết test case, automation test.
- Các công ty: KMS Technology, NashTech, Axon Active.

3. Chuyên viên phân tích hệ thống (System Analyst)

- Phân tích yêu cầu và thiết kế hệ thống.
- Công ty công nghệ tài chính, ngân hàng (MB Bank, Techcombank IT, VPBank IT).

4. Quản lý dự án phần mềm (Software Project Manager)

- Lập kế hoạch, giám sát tiến độ dự án phần mềm.
- Các công ty: Viettel Digital, FPT IS, VNPT IT.

5. DevOps Engineer

- Quản lý hạ tầng, CI/CD, tối ưu hiệu suất hệ thống.
- Công ty: Shopee, Lazada, VNG Cloud.

Vị trí	Mức lương (triệu VNĐ/tháng)	Nguồn tham khảo
Lập trình viên phần mềm (Software Developer)	10 - 20	vietnamnet.vn
Kỹ sư kiểm thử phần mềm (Software Tester/QA Engineer)	7 - 18	topcv.vn
Chuyên viên phân tích hệ thống (System Analyst)	10 - 20	vietnamnet.vn
Quản lý dự án phần mềm (Software Project Manager)	27 - 52	vietnamnet.vn
Kỹ sư DevOps (DevOps Engineer)	14 - 30	topcv.vn

Lưu ý: Mức lương trên chỉ mang tính chất tham khảo và có thể thay đổi dựa trên nhiều yếu tố như kinh nghiệm, kỹ năng, vị trí địa lý và quy mô của công ty.

CÁC VỊ TRÍ NGHỀ NGHIỆP PHỔ BIẾN (QUỐC TẾ)

1. Software Engineer (Kỹ sư phần mềm)

- Làm việc tại các công ty công nghệ lớn như Google, Microsoft, Amazon, Meta.

2. Machine Learning Engineer

- Xây dựng mô hình AI, xử lý dữ liệu lớn.
- Công ty: OpenAI, Tesla, NVIDIA.

3. Cloud Engineer

- Phát triển và quản lý ứng dụng trên nền tảng đám mây như AWS, Azure.
- Công ty: Amazon Web Services, IBM, Oracle.

4. Cybersecurity Engineer

- Kiểm thử bảo mật, chống hacker, bảo vệ dữ liệu người dùng.
- Công ty: Palo Alto Networks, Cisco, CrowdStrike.

5. IT Consultant

- Tư vấn giải pháp phần mềm và hệ thống CNTT cho doanh nghiệp.
- Công ty: Accenture, Deloitte, PwC.

Vị trí	Mức lương trung bình (USD/năm)	Nguồn tham khảo
Kỹ sư phần mềm (Software Engineer)	110,140	ditrumyfcg.com
Kỹ sư học máy (Machine Learning Engineer)	160,000 – 308,000	vietnamnet.vn
Kỹ sư đám mây (Cloud Engineer)	116,780	ditrumyfcg.com
Kỹ sư an ninh mạng (Cybersecurity Engineer)	103,590	ditrumyfcg.com
Chuyên viên tư vấn CNTT (IT Consultant)	100,000 – 150,000	vietnamnet.vn

Lưu ý: Mức lương trên chỉ mang tính chất tham khảo và có thể thay đổi dựa trên nhiều yếu tố như kinh nghiệm, kỹ năng, vị trí địa lý và quy mô của công ty.

1. LẬP TRÌNH VIÊN PHẦN MỀM (SOFTWARE DEVELOPER)

MÔ TẢ CÔNG VIỆC

- ✓ Phát triển, bảo trì và tối ưu hóa các ứng dụng phần mềm, website, hoặc mobile app.
- ✓ Viết mã sạch, có thể tái sử dụng và dễ bảo trì theo chuẩn coding.
- ✓ Tham gia vào toàn bộ vòng đời phát triển phần mềm (SDLC), từ phân tích yêu cầu, thiết kế, lập trình đến kiểm thử và triển khai.
- ✓ Phối hợp với các thành viên trong nhóm (BA, QA, Designer) để đảm bảo chất lượng sản phẩm.
- ✓ Cập nhật công nghệ mới, tối ưu codebase hiện có.

YÊU CẦU

- ✓ Thành thạo ít nhất một ngôn ngữ lập trình: **Java, Python, JavaScript, C#, PHP, Go, Kotlin**.
- ✓ Hiểu về mô hình lập trình OOP, MVC, hoặc các thiết kế phần mềm phổ biến.
- ✓ Kinh nghiệm với **cơ sở dữ liệu (SQL, NoSQL)**.
- ✓ Kiến thức về các framework phổ biến: **React, Angular, Spring Boot, Laravel**.
- ✓ Có kinh nghiệm sử dụng **Git, Docker, Kubernetes** là lợi thế.

2. KỸ SƯ KIỂM THỬ PHẦN MỀM (SOFTWARE TESTER/QA ENGINEER)

MÔ TẢ CÔNG VIỆC

- ✓ Lập kế hoạch kiểm thử (test plan), thiết kế kịch bản kiểm thử (test case).
- ✓ Thực hiện kiểm thử thủ công (Manual Testing) và kiểm thử tự động (Automation Testing).
- ✓ Phát hiện, ghi nhận và theo dõi lỗi (bug tracking) bằng Jira, TestRail, hoặc tương đương.
- ✓ Đảm bảo chất lượng phần mềm thông qua kiểm thử tính năng, hiệu năng, bảo mật.
- ✓ Đưa ra các đề xuất cải thiện quy trình phát triển phần mềm.

YÊU CẦU

- ✓ Có hiểu biết về các phương pháp kiểm thử phần mềm: **Unit Test, Integration Test, Regression Test, Performance Test**.
- ✓ Thành thạo ít nhất một công cụ kiểm thử tự động: **Selenium, Appium, Cypress**.
- ✓ Có kiến thức về CI/CD, kiểm thử API bằng **Postman, JMeter**.
- ✓ Kinh nghiệm sử dụng các hệ thống quản lý lỗi **Jira, Redmine** là lợi thế.

1. LẬP TRÌNH VIÊN PHẦN MỀM (SOFTWARE DEVELOPER)

- Yêu cầu tối thiểu: 0 – 1 năm (fresher có thể ứng tuyển với thực tập hoặc dự án cá nhân).

MÔ TẢ CÔNG VIỆC

- ✓ Phát triển, bảo trì và tối ưu hóa các ứng dụng phần mềm, website, hoặc mobile app.
- ✓ Viết mã sạch, có thể tái sử dụng và dễ bảo trì theo chuẩn coding.
- ✓ Tham gia vào toàn bộ vòng đời phát triển phần mềm (SDLC), từ phân tích yêu cầu, thiết kế, lập trình đến kiểm thử và triển khai.
- ✓ Phối hợp với các thành viên trong nhóm (BA, QA, Designer) để đảm bảo chất lượng sản phẩm.
- ✓ Cập nhật công nghệ mới, tối ưu codebase hiện có.

YÊU CẦU

- ✓ Thành thạo ít nhất một ngôn ngữ lập trình: **Java, Python, JavaScript, C#, PHP, Go, Kotlin**.
- ✓ Hiểu về mô hình lập trình OOP, MVC, hoặc các thiết kế phần mềm phổ biến.
- ✓ Kinh nghiệm với **cơ sở dữ liệu (SQL, NoSQL)**.
- ✓ Kiến thức về các framework phổ biến: **React, Angular, Spring Boot, Laravel**.
- ✓ Có kinh nghiệm sử dụng **Git, Docker, Kubernetes** là lợi thế.

2. KỸ SƯ KIỂM THỬ PHẦN MỀM (SOFTWARE TESTER/QA ENGINEER)

- Yêu cầu tối thiểu: 0 – 1 năm (có thể bắt đầu từ fresher QA hoặc internship).

MÔ TẢ CÔNG VIỆC

- ✓ Lập kế hoạch kiểm thử (test plan), thiết kế kịch bản kiểm thử (test case).
- ✓ Thực hiện kiểm thử thủ công (Manual Testing) và kiểm thử tự động (Automation Testing).
- ✓ Phát hiện, ghi nhận và theo dõi lỗi (bug tracking) bằng Jira, TestRail, hoặc tương đương.
- ✓ Đảm bảo chất lượng phần mềm thông qua kiểm thử tính năng, hiệu năng, bảo mật.
- ✓ Đưa ra các đề xuất cải thiện quy trình phát triển phần mềm.

YÊU CẦU

- ✓ Có hiểu biết về các phương pháp kiểm thử phần mềm: **Unit Test, Integration Test, Regression Test, Performance Test**.
- ✓ Thành thạo ít nhất một công cụ kiểm thử tự động: **Selenium, Appium, Cypress**.
- ✓ Có kiến thức về CI/CD, kiểm thử API bằng **Postman, JMeter**.
- ✓ Kinh nghiệm sử dụng các hệ thống quản lý lỗi **Jira, Redmine** là lợi thế.

3. CHUYÊN VIÊN PHÂN TÍCH HỆ THỐNG (SYSTEM ANALYST)

- Yêu cầu tối thiểu: 1 – 3 năm (cần có kinh nghiệm lập trình hoặc làm QA/BA trước đó).

MÔ TẢ CÔNG VIỆC

- ✓ Thu thập, phân tích yêu cầu của khách hàng và chuyển thành tài liệu đặc tả phần mềm (SRS).
- ✓ Phối hợp với đội kỹ thuật để thiết kế hệ thống, đảm bảo đáp ứng yêu cầu kinh doanh.
- ✓ Xây dựng tài liệu đặc tả hệ thống: **ERD, Use Case Diagram, Data Flow Diagram**.
- ✓ Tư vấn cho khách hàng về tính khả thi của hệ thống.
- ✓ Kiểm tra tính hợp lý của hệ thống sau khi triển khai.

YÊU CẦU

- ✓ Kiến thức về **phân tích yêu cầu phần mềm, UML, BPMN**.
- ✓ Kinh nghiệm làm việc với **SQL, NoSQL, RESTful API**.
- ✓ Thành thạo các công cụ hỗ trợ phân tích: **Enterprise Architect, Lucidchart**.
- ✓ Kỹ năng giao tiếp, tư duy logic, giải quyết vấn đề tốt.

4. QUẢN LÝ DỰ ÁN PHẦN MỀM (SOFTWARE PROJECT MANAGER)

- Yêu cầu tối thiểu: 3 – 5 năm (cần có kinh nghiệm quản lý nhóm hoặc dự án nhỏ trước đó).

MÔ TẢ CÔNG VIỆC

- ✓ Lập kế hoạch, quản lý, theo dõi tiến độ và điều phối các dự án phần mềm.
- ✓ Quản lý rủi ro, đảm bảo dự án hoàn thành đúng tiến độ, ngân sách.
- ✓ Giao tiếp với khách hàng, xác định yêu cầu, lên kế hoạch phát triển.
- ✓ Báo cáo tiến độ, chi phí, vấn đề phát sinh lên cấp trên.
- ✓ Quản lý đội nhóm kỹ sư phần mềm, đảm bảo hiệu suất làm việc cao.

YÊU CẦU

- ✓ Hiểu về mô hình quản lý phần mềm: **Agile, Scrum, Waterfall**.
- ✓ Có kinh nghiệm làm việc với các công cụ quản lý: **Jira, Trello, Asana**.
- ✓ Kỹ năng giao tiếp, đàm phán, xử lý vấn đề tốt.
- ✓ Có chứng chỉ quản lý dự án **PMP, CSM** là lợi thế.

5. KỸ SƯ DEVOPS (DEVOPS ENGINEER)

- **Yêu cầu tối thiểu:** 2 – 3 năm (cần có nền tảng sysadmin hoặc backend developer).
- **Kỹ năng cần có:** Docker, Kubernetes, AWS/GCP/Azure, Terraform, Ansible.
- **Vị trí cao hơn:**
 - **Junior DevOps:** 2 – 3 năm.
 - **Senior DevOps:** 4 – 6 năm.
 - **DevOps Architect:** 6+ năm.

❑ MÔ TẢ CÔNG VIỆC

- ✓ Xây dựng, triển khai và quản lý hạ tầng hệ thống phần mềm.
- ✓ Thiết lập và quản lý pipeline CI/CD (Jenkins, GitLab CI, Azure DevOps).
- ✓ Giám sát hệ thống, tối ưu hiệu suất và bảo mật server.
- ✓ Quản lý containerization bằng Docker, Kubernetes.
- ✓ Xây dựng các giải pháp tự động hóa quá trình phát triển.

❑ YÊU CẦU

- ✓ Kinh nghiệm với **AWS, GCP, Azure**.
- ✓ Thành thạo các công cụ **Terraform, Ansible, Kubernetes**.
- ✓ Kiến thức sâu về **Linux, Bash scripting**.
- ✓ Kinh nghiệm với **log monitoring (ELK, Prometheus, Grafana)**.

6. KỸ SƯ HỌC MÁY (MACHINE LEARNING ENGINEER)

- **Yêu cầu tối thiểu:** 2 – 4 năm (có thể bắt đầu từ Data Scientist hoặc AI Researcher).
- **Kỹ năng cần có:** Python, TensorFlow, PyTorch, Big Data, NLP.
- **Vị trí cao hơn:**
 - **Junior ML Engineer:** 2 – 4 năm.
 - **Senior ML Engineer:** 5 – 7 năm.
 - **AI Architect/Researcher:** 7+ năm.

❑ MÔ TẢ CÔNG VIỆC

- ✓ Thiết kế và triển khai các mô hình AI/ML cho sản phẩm.
- ✓ Làm việc với dữ liệu lớn, tối ưu thuật toán.
- ✓ Triển khai mô hình ML lên sản phẩm thực tế (MLOps).
- ✓ Sử dụng các framework AI như **TensorFlow, PyTorch**.
- ✓ Xây dựng hệ thống AI phục vụ phân tích dữ liệu, nhận diện hình ảnh, NLP.

❑ YÊU CẦU

- ✓ Kinh nghiệm với **Python, R, TensorFlow, Scikit-learn**.
- ✓ Hiểu về **Deep Learning, Reinforcement Learning**.
- ✓ Kỹ năng xử lý dữ liệu lớn với **Hadoop, Spark**.
- ✓ Có kinh nghiệm với **MLOps, model deployment**.

7. KỸ SƯ AN NINH MẠNG (CYBERSECURITY ENGINEER)

- **Yêu cầu tối thiểu:** 2 – 4 năm (nền tảng IT Security hoặc Network Admin). **Kỹ năng cần có:** Kali Linux, CEH, OSCP, SIEM.
- **Vị trí cao hơn:**
 - **Cybersecurity Analyst:** 2 – 4 năm.
 - **Security Engineer:** 4 – 6 năm.
 - **Security Architect:** 7+ năm.

MÔ TẢ CÔNG VIỆC

- ✓ Xây dựng và triển khai các giải pháp bảo mật cho hệ thống phần mềm.
- ✓ Kiểm thử bảo mật (Penetration Testing), xác định lỗ hổng bảo mật.
- ✓ Đánh giá và giám sát bảo mật của hệ thống (SIEM, IDS/IPS).
- ✓ Xây dựng chính sách bảo mật, tuân thủ ISO 27001, GDPR.

YÊU CẦU

- ✓ Kiến thức về **network security, cryptography**.
- ✓ Kinh nghiệm với **firewall, IDS/IPS, DLP**.
- ✓ Có chứng chỉ bảo mật **CEH, CISSP, OSCP** là lợi thế.

8. CHUYÊN VIÊN TƯ VẤN CNTT (IT CONSULTANT)

- **Yêu cầu tối thiểu:** 3 – 5 năm (cần hiểu sâu về hệ thống IT, phần mềm doanh nghiệp).
- **Kỹ năng cần có:** ERP, SAP, CRM, kỹ năng giao tiếp, đàm phán.
- **Vị trí cao hơn:**
 - **Junior IT Consultant:** 3 – 5 năm.
 - **Senior IT Consultant:** 6 – 8 năm.
 - **IT Strategy Consultant:** 8+ năm.

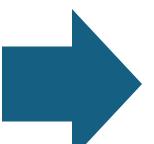
MÔ TẢ CÔNG VIỆC

- ✓ Đánh giá nhu cầu công nghệ thông tin của doanh nghiệp.
- ✓ Đề xuất giải pháp phù hợp với chiến lược kinh doanh.
- ✓ Hỗ trợ triển khai các dự án CNTT.
- ✓ Huấn luyện và hỗ trợ doanh nghiệp trong việc ứng dụng công nghệ.

YÊU CẦU

- ✓ Kiến thức rộng về **hệ thống CNTT, phần mềm doanh nghiệp (ERP, CRM)**.
- ✓ Kỹ năng phân tích, thuyết trình và tư vấn tốt.
- ✓ Có kinh nghiệm với **SAP, Oracle, Microsoft Dynamics** là lợi thế.

Tiêu chí	Software Engineering	Introduction to Software Engineering
Mức độ môn học	Môn chuyên sâu	Môn nhập môn, cơ bản
Mục tiêu chính	Cung cấp kiến thức toàn diện về quy trình phát triển phần mềm, quản lý dự án, kiểm thử, bảo trì, và tối ưu phần mềm.	Giới thiệu các khái niệm cơ bản về kỹ thuật phần mềm, vòng đời phát triển phần mềm, các mô hình phát triển.
Đối tượng học	Sinh viên chuyên ngành Công nghệ Thông tin, Kỹ thuật phần mềm hoặc người làm việc trong ngành.	Sinh viên năm đầu hoặc người mới bắt đầu tìm hiểu về phần mềm.
Chương trình giảng dạy	Bao gồm các chủ đề nâng cao như kiến trúc phần mềm, quy trình phát triển Agile, DevOps, kiểm thử tự động, và quản lý rủi ro.	Tập trung vào các nguyên tắc cơ bản như yêu cầu phần mềm, phân tích, thiết kế, và mô hình phát triển (Waterfall, Agile, Scrum).
Ứng dụng thực tế	Đào sâu vào phát triển phần mềm thực tế, kỹ thuật thiết kế hệ thống lớn, đảm bảo chất lượng phần mềm.	Hướng dẫn cơ bản về lập trình, thiết kế phần mềm đơn giản, giới thiệu cách làm việc nhóm trong dự án phần mềm.
Mức độ phức tạp	Cao hơn, yêu cầu nền tảng về lập trình và hệ thống	Dễ tiếp cận, không yêu cầu kiến thức nền sâu về phần mềm.

- 
- ❑ **Introduction to Software Engineering** phù hợp với người mới bắt đầu, cung cấp kiến thức cơ bản về phần mềm và quy trình phát triển.
 - ❑ **Software Engineering** dành cho người đã có kiến thức nền tảng, tập trung vào các kỹ thuật nâng cao để xây dựng, quản lý và bảo trì phần mềm chất lượng cao.

Tuần 01

NỘI DUNG

1.1. Các khái niệm cơ bản

1.1.1. Phần mềm

1.1.2. Công nghệ phần mềm

1.1.3. Phát triển phần mềm

1.2. Sự cần thiết của công nghệ phần mềm

1.2.1. Khía cạnh kinh tế

1.2.2. Khía cạnh công nghệ

1.2.3. Khía cạnh bảo trì

1.3. Các vấn đề thường gặp khi phát triển phần mềm

1.3.1. Vấn đề vượt chi phí

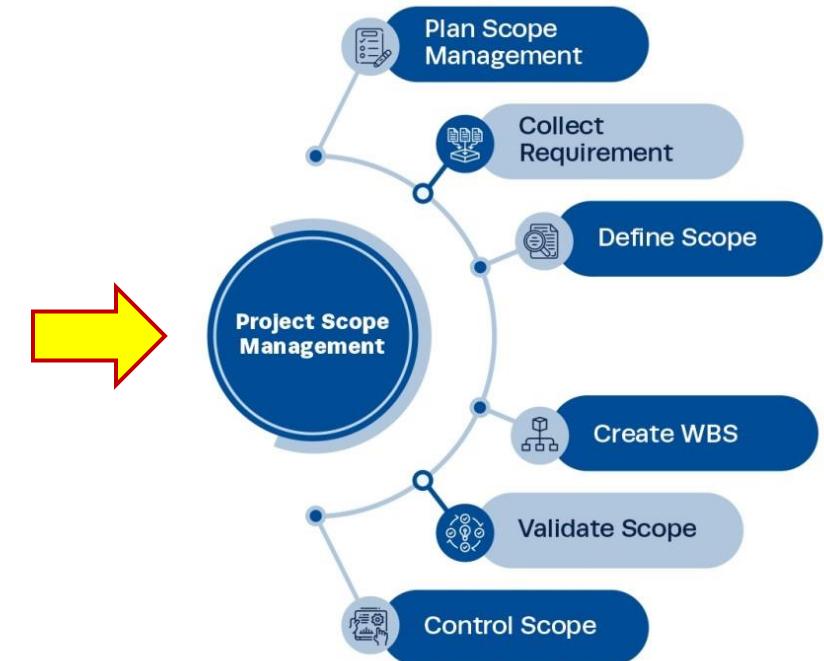
1.3.2. Vấn đề trễ thời hạn

1.3.3. Vấn đề phần mềm vẫn còn lỗi sau khi bàn giao

1.4. Case study: Các nhóm (3-5sv) đọc tài liệu và báo cáo về phạm vi của công nghệ phần mềm



- Hiểu phạm vi của công nghệ phần mềm:** Định nghĩa công nghệ phần mềm và những tác động của nó trong việc phát triển các sản phẩm phần mềm.
- Mô tả các mô hình vòng đời phần mềm:** Giải thích mô hình vòng đời phần mềm cổ điển và các giai đoạn của nó.
- Nhận thức tầm quan trọng của bảo trì phần mềm:** Thảo luận sự khác biệt giữa quan điểm bảo trì phần mềm cổ điển và hiện đại, cùng các tác động của nó đến vòng đời dự án.
- Hiểu về mô hình hướng đối tượng:** Khám phá điểm mạnh và sự chấp nhận rộng rãi của cách tiếp cận hướng đối tượng trong phát triển phần mềm.
- Đánh giá đạo đức nghề nghiệp và làm việc nhóm:** Nhận thức tầm quan trọng của các thực hành đạo đức và sự hợp tác hiệu quả trong nhóm phát triển phần mềm.



Project Scope Management

➤ SAU KHI HOÀN THÀNH CHƯƠNG 1, SINH VIÊN SẼ CÓ THỂ:

- Định nghĩa rõ ràng công nghệ phần mềm** như một lĩnh vực nhằm mục tiêu cung cấp phần mềm không lỗi, đáp ứng đúng nhu cầu người dùng, hoàn thành đúng thời gian và nằm trong ngân sách.
- Xác định và giải thích các giai đoạn trong vòng đời phần mềm**, bao gồm lập kế hoạch, kiểm thử, và tài liệu hóa, cùng vai trò của chúng trong quản lý dự án thành công.
- So sánh các phương pháp cổ điển và hướng đối tượng**, chỉ ra điểm mạnh, hạn chế và bối cảnh áp dụng của từng phương pháp.
- Phân tích tác động của bảo trì phần mềm**, tác động kinh tế và vai trò của nó trong vòng đời phần mềm.
- Áp dụng các nguyên tắc đạo đức trong quá trình ra quyết định** và hiểu rõ tầm quan trọng của chúng trong thực hành chuyên môn.

Đảm nhận nhiều vai trò khác nhau trong dự án phát triển phần mềm:

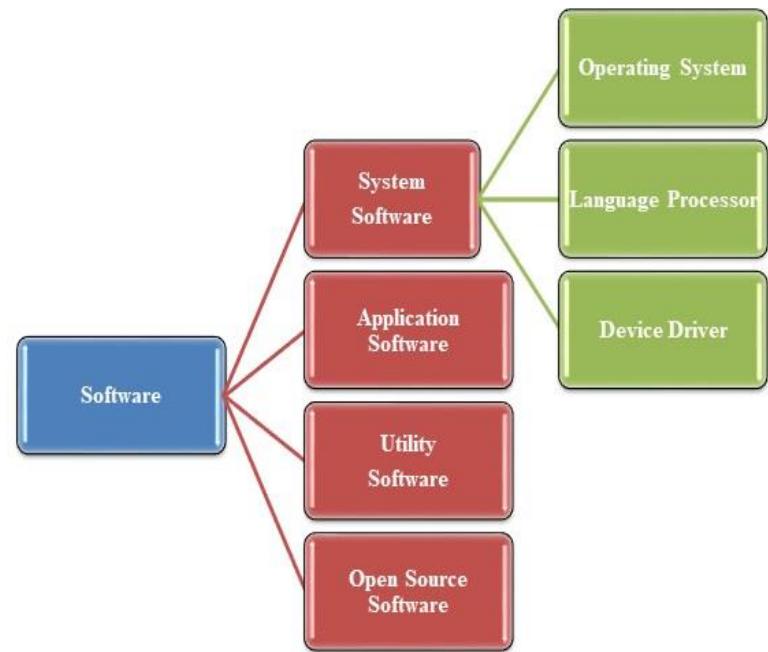
1. Lập trình viên (Developer).
2. Kiểm thử viên (Tester/Quality Control - QC)
3. Phân tích nghiệp vụ (Business Analyst - BA)
4. Quản lý dự án (Project Manager - PM)
5. Thiết kế giao diện người dùng (UI/UX Designer)
6. Quản lý chất lượng (Quality Assurance - QA)

Phần mềm

Định nghĩa: Phần mềm là tập hợp các hướng dẫn, chương trình được viết để máy tính thực thi nhằm thực hiện các chức năng hoặc nhiệm vụ cụ thể.

Phân loại phần mềm:

- Phần mềm hệ thống:** Hệ điều hành, trình điều khiển thiết bị.
- Phần mềm ứng dụng:** Các chương trình phục vụ công việc hoặc nhu cầu của người dùng cuối như Microsoft Office, trình duyệt web.
- Phần mềm nhúng:** Phần mềm điều khiển các thiết bị phần cứng như máy giặt, điều hòa.



Computer - Software Types

❑ **Định nghĩa:** Công nghệ phần mềm là lĩnh vực nghiên cứu, phát triển và áp dụng các phương pháp có hệ thống để xây dựng phần mềm chất lượng cao.

❑ **Mục tiêu:**

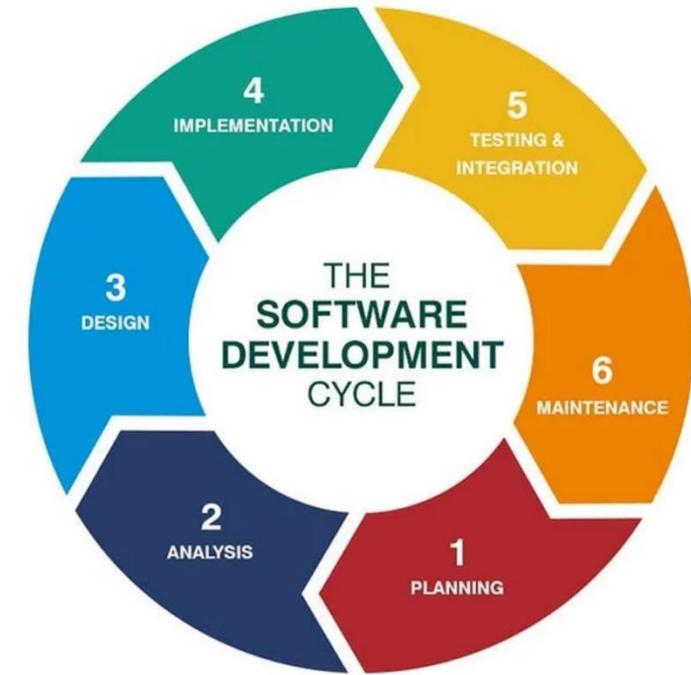
- Tạo ra phần mềm đúng yêu cầu khách hàng.
- Đảm bảo phần mềm có thể bảo trì và nâng cấp dễ dàng.
- Giảm thiểu thời gian và chi phí phát triển.



Factors of Software Quality

□ QUY TRÌNH PHÁT TRIỂN PHẦN MỀM:

- Phân tích yêu cầu:** Thu thập và phân tích các yêu cầu từ khách hàng để hiểu rõ mục tiêu và chức năng của phần mềm.
- Thiết kế:** Lên kế hoạch và cấu trúc hệ thống phần mềm, bao gồm thiết kế kiến trúc và giao diện người dùng.
- Lập trình:** Chuyển đổi thiết kế thành mã nguồn thực thi, phát triển các tính năng và chức năng của phần mềm.
- Kiểm thử:** Đảm bảo phần mềm hoạt động đúng chức năng và không có lỗi thông qua các hoạt động kiểm thử.
- Triển khai:** Cài đặt và bàn giao phần mềm cho khách hàng, đảm bảo phần mềm hoạt động ổn định trong môi trường thực tế.
- Bảo trì:** Khắc phục lỗi và nâng cấp phần mềm sau khi triển khai để đáp ứng nhu cầu thay đổi của người dùng và thị trường.

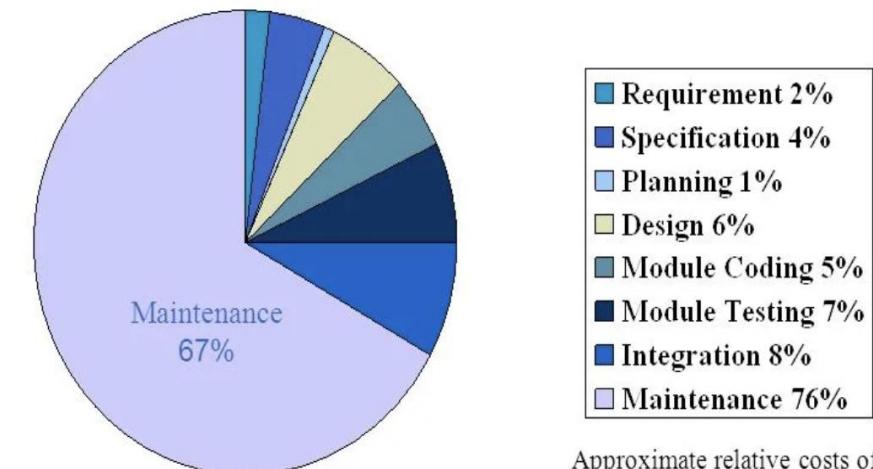


Quy trình phát triển phần mềm gồm nhiều giai đoạn quan trọng, từ phân tích yêu cầu, thiết kế, lập trình, kiểm thử, triển khai đến bảo trì. Mỗi giai đoạn đóng vai trò quan trọng trong việc đảm bảo phần mềm hoạt động ổn định, đáp ứng nhu cầu người dùng và có thể mở rộng trong tương lai.

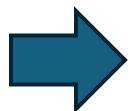
Phạm vi phát triển phần mềm:

- Bảo trì (Maintenance) rất quan trọng đến mức một phần lớn của công nghệ phần mềm bao gồm các kỹ thuật, công cụ và thực hành nhằm giảm chi phí bảo trì.
- Biểu đồ chi phí tương đối của các giai đoạn trong vòng đời phần mềm:

- Yêu cầu (Requirement): 2%
- Đặc tả (Specification): 4%
- Lập kế hoạch (Planning): 1%
- Thiết kế (Design): 6%
- Lập trình module (Module Coding): 5%
- Kiểm thử module (Module Testing): 7%
- Tích hợp (Integration): 8%
- Bảo trì (Maintenance): 76%



Approximate relative costs of the phases of the software life cycle.



Chi phí bảo trì chiếm phần lớn (76%) trong vòng đời phần mềm, nhấn mạnh tầm quan trọng của việc quản lý hiệu quả giai đoạn này.

1.2.1. Khía cạnh kinh tế

- Phần mềm là yếu tố cốt lõi trong nhiều ngành công nghiệp như tài chính, y tế, giáo dục.
- Sự phát triển của phần mềm giúp tăng năng suất lao động và tối ưu hóa chi phí vận hành doanh nghiệp.
- Các dự án phần mềm lớn thường có chi phí rất cao, do đó việc áp dụng công nghệ phần mềm giúp kiểm soát chi phí tốt hơn.

1.2.2. Khía cạnh công nghệ

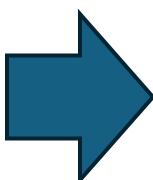
- Sự phát triển nhanh chóng của công nghệ yêu cầu phần mềm phải được cải tiến liên tục để đáp ứng nhu cầu mới.
- Phần mềm giúp kết nối các hệ thống phức tạp và xử lý khối lượng dữ liệu lớn.

1.2.3. Khía cạnh bảo trì

- Sau khi phần mềm được triển khai, bảo trì là hoạt động cần thiết để đảm bảo phần mềm hoạt động ổn định và đáp ứng các yêu cầu thay đổi của người dùng.
- Bảo trì phần mềm chiếm khoảng 60% tổng chi phí vòng đời của một hệ thống phần mềm.

Needed of
SWE

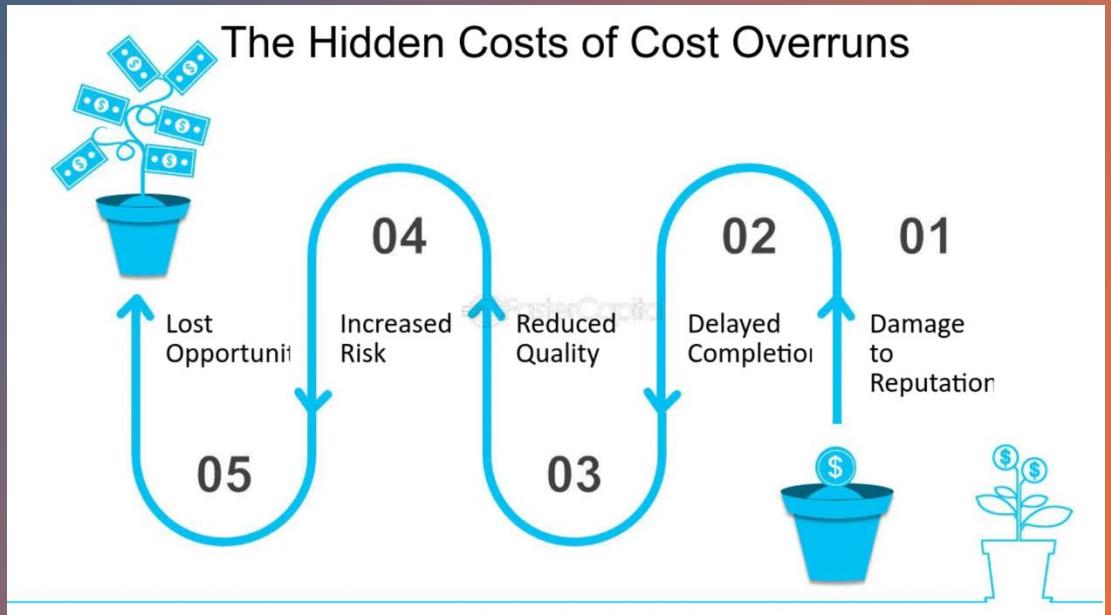
- Phần mềm lớn (Large software)
- Khả năng mở rộng (Scalability)
- Khả năng thích nghi (Adaptability)
- Chi phí (Cost)
- Tính chất động (Dynamic Nature)
- Quản lý chất lượng (Quality Management)



- Công nghệ phần mềm đóng vai trò quan trọng trong sự phát triển của doanh nghiệp và nền kinh tế số.
 - Việc đầu tư vào phần mềm không chỉ giúp tăng hiệu suất mà còn giảm chi phí dài hạn.
 - Bảo trì phần mềm là yếu tố then chốt giúp hệ thống hoạt động ổn định và thích nghi với thay đổi.
 - Hiểu và áp dụng công nghệ phần mềm hiệu quả chính là chìa khóa để doanh nghiệp phát triển bền vững trong thời đại số!

SWE | 1.3. CÁC VẤN ĐỀ THƯỜNG GẶP KHI PHÁT TRIỂN PHẦN MỀM

- 1.3.1. Vấn đề vượt chi phí
- 1.3.2. Vấn đề trễ thời hạn
- 1.3.3. Vấn đề phần mềm vẫn còn lỗi sau khi bàn giao





Nhận định:

Nhiều dự án phần mềm bị đội ngũ sách do **ước lượng sai khối lượng công việc và thay đổi yêu cầu liên tục**. Việc không kiểm soát tốt phạm vi dự án có thể làm chi phí tăng gấp nhiều lần so với kế hoạch ban đầu.



Nguyên nhân:

- Không dự báo chính xác **tài nguyên, thời gian và công sức** cần thiết.
- Khách hàng thay đổi yêu cầu liên tục nhưng không có **quy trình kiểm soát thay đổi (Change Control Process)**.
- Phát sinh **chi phí sửa lỗi** do thiếu kiểm thử hoặc thiết kế kém.



Giải pháp:

- Ước lượng chi phí chính xác** bằng các phương pháp như **COCOMO, Function Point Analysis**.
- Quản lý thay đổi tốt hơn** bằng cách sử dụng mô hình Agile để linh hoạt nhưng vẫn có kiểm soát.
- Xây dựng phạm vi dự án rõ ràng ngay từ đầu**, tránh phát sinh yêu cầu ngoài kế hoạch.



Bài học cần nhớ:

Lập kế hoạch ngân sách chi tiết và kiểm soát phạm vi dự án chặt chẽ ngay từ đầu để tránh rủi ro tài chính.



Nhận định:

- Hầu hết các dự án phần mềm đều gặp **chậm tiến độ do thay đổi yêu cầu, kế hoạch không thực tế hoặc thiếu nhân lực phù hợp**. Điều này dẫn đến áp lực deadline, chất lượng sản phẩm giảm sút.



Nguyên nhân:

- Thay đổi yêu cầu liên tục** từ khách hàng hoặc nội bộ.
- Ước lượng thời gian không chính xác** dẫn đến kế hoạch không khả thi.
- Thiếu nhân lực có chuyên môn** hoặc sự phối hợp kém giữa các nhóm phát triển.



Giải pháp:

- Sử dụng kỹ thuật lập kế hoạch hợp lý** như **Gantt Chart, Critical Path Method (CPM)**.
- Áp dụng phương pháp Agile/Scrum** để chia nhỏ công việc thành các Sprint ngắn, giúp theo dõi tiến độ sát sao.
- Dự trù rủi ro thời gian** bằng cách lập kế hoạch có tính linh hoạt, không đặt lịch trình quá căng thẳng.



Bài học cần nhớ:

Lập kế hoạch ngân sách chi tiết và kiểm soát phạm vi dự án chặt chẽ ngay từ đầu để tránh rủi ro tài chính.

⚠ Nhận định:

- Nhiều phần mềm vẫn còn lỗi sau khi triển khai do kiểm thử không đầy đủ, bỏ qua lỗi nhỏ để chạy kịp deadline, hoặc không có chiến lược kiểm thử rõ ràng. Điều này gây thiệt hại lớn cho doanh nghiệp và làm mất lòng tin của khách hàng.

📌 Nguyên nhân:

- Không có quy trình kiểm thử bài bản** (chỉ kiểm thử chức năng chính, bỏ qua kiểm thử bảo mật và hiệu suất).
- Bỏ qua lỗi nhỏ để kịp tiến độ**, dẫn đến lỗi nghiêm trọng sau khi phát hành.
- Không thực hiện kiểm thử tự động**, dẫn đến lỗi tái phát sau mỗi lần cập nhật.

✓ Giải pháp:

- Xây dựng quy trình kiểm thử toàn diện** (Unit Test, Integration Test, System Test, UAT).
- Áp dụng kiểm thử tự động** bằng các công cụ như Selenium, JUnit, Postman để giảm thiểu lỗi con người.
- Triển khai mô hình DevOps** để kiểm thử liên tục trong suốt vòng đời phát triển phần mềm.

💡 **Bài học cần nhớ:**

Kiểm thử kỹ càng và có chiến lược rõ ràng giúp phần mềm hoạt động ổn định, hạn chế lỗi phát sinh sau triển khai.

1.3.4. CÓ 5 VẤN ĐỀ THƯỜNG GẶP TRONG PHÁT TRIỂN PHẦN MỀM

1. Không đồng bộ (Misalignment)

Khi các bên liên quan trong doanh nghiệp và bộ phận IT không cùng chung quan điểm, phần mềm được tạo ra có khả năng sẽ không như mong đợi.

2. Thiếu quyết đoán (Indecision)

Khi doanh nghiệp không thể đưa ra quyết định cuối cùng đủ nhanh, dự án có khả năng sẽ không hoàn thành đúng thời hạn.

3. Yêu cầu kém (Poor Requirements)

Khi các yêu cầu không truyền tải chính xác nhu cầu của doanh nghiệp đến đội phát triển, bạn có thể phải thực hiện lại rất nhiều công việc.

4. Phạm vi mở rộng không kiểm soát (Scope Creep)

Khi doanh nghiệp hoặc đội phát triển phần mềm thêm nhiều tính năng mới, dự án có khả năng sẽ không hoàn thành đúng hạn.

5. Thiếu giao tiếp (Miscommunication)

Khi các thành viên trong nhóm không dành thời gian để hiểu nhau, bạn có khả năng sẽ tham gia vào các cuộc thảo luận không cần thiết và phải làm lại công việc.

❑ Mục tiêu:

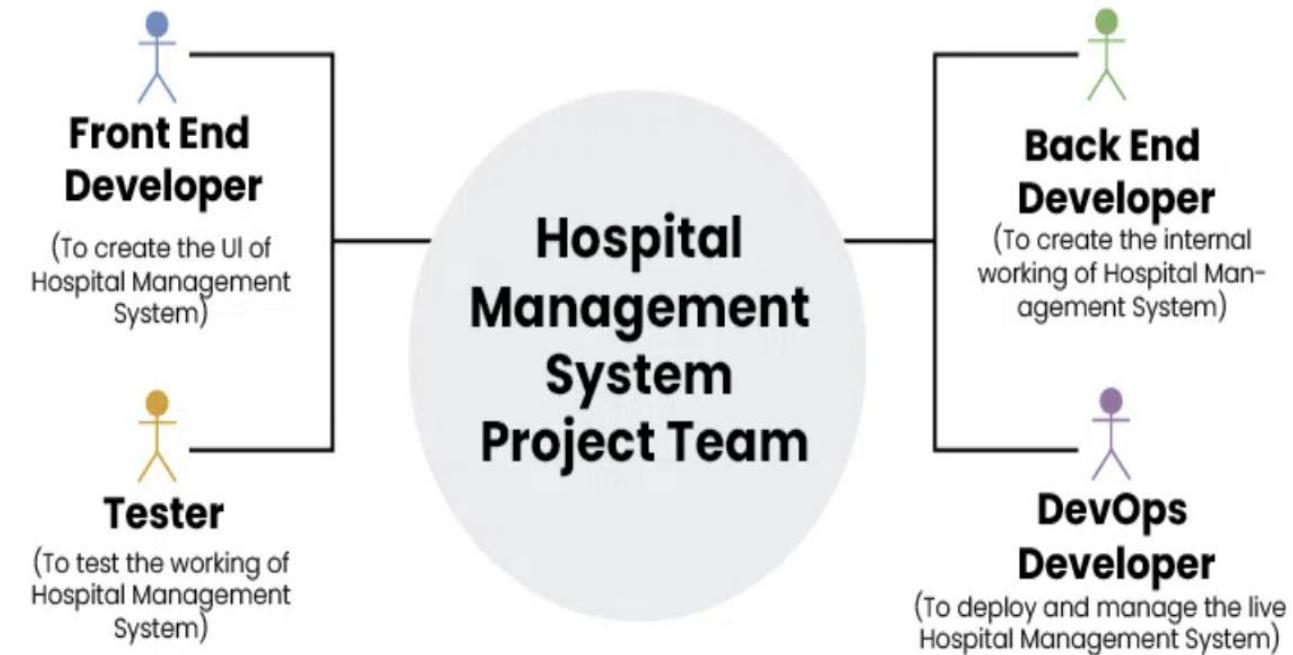
- Giúp sinh viên hiểu rõ hơn về phạm vi của công nghệ phần mềm thông qua ví dụ thực tế.
- Khuyến khích sinh viên thảo luận và trình bày quan điểm.

❑ Hoạt động nhóm:

- Chia lớp thành các nhóm từ 3-5 sinh viên.
- Mỗi nhóm sẽ chọn một chủ đề cụ thể liên quan đến phạm vi công nghệ phần mềm, ví dụ:
 - **Phần mềm quản lý bệnh viện:** Phạm vi và vai trò của phần mềm trong y tế.
 - **Phần mềm thương mại điện tử:** Phạm vi và ứng dụng trong bán hàng trực tuyến.
 - **Phần mềm điều khiển ô tô tự hành:** Vai trò của phần mềm trong ngành công nghiệp ô tô.
- Các nhóm sẽ chuẩn bị báo cáo và thuyết trình trước lớp về chủ đề đã chọn.

Case study - Phần mềm quản lý bệnh viện-Hospital Management System (HMS) là một trong những dự án phát triển phần mềm phổ biến nhất. Việc phát triển phần mềm HMS sẽ thực hiện qua nhiều giai đoạn, chẳng hạn như:

- 1.Thành lập đội
- 2.Lựa chọn chủ đề
- 3.Tạo Tóm tắt Dự án
- 4.Thu thập yêu cầu
- 5.Mã hóa hoặc triển khai
- 6.Kiểm tra
- 7.Trình bày dự án



Case study - Phần mềm quản lý bệnh viện-Hospital Management System (HMS) là một trong những dự án phát triển phần mềm phổ biến nhất. Việc phát triển phần mềm HMS sẽ thực hiện qua nhiều giai đoạn, chẳng hạn như:

CÁC BƯỚC THỰC HIỆN

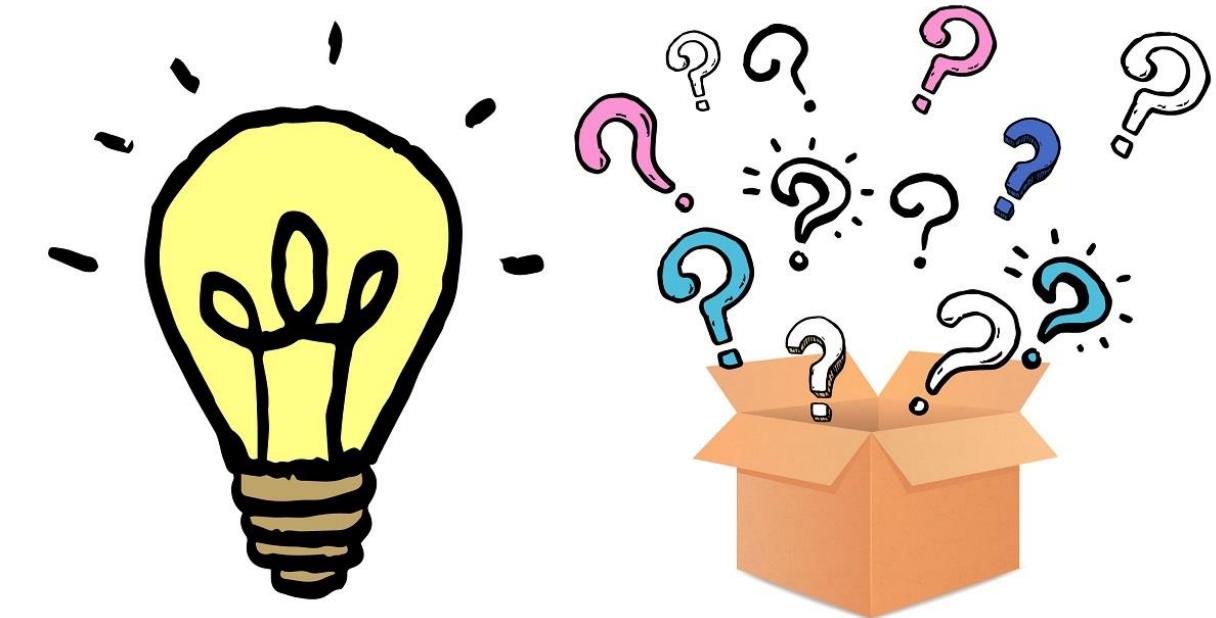
- 1.Thành nhóm (team formation)
- 2.Lựa chọn chủ đề (topic selection)
- 3.Tạo Tóm tắt Dự án Creating project)
- 4.Thu thập yêu cầu (Requirements gathering)
- 5.Mã hóa hoặc triển khai (coding/ implementation)
- 6.Kiểm tra (Testing)
- 7.Trình bày dự án (Proj. Presentation)

THÀNH LẬP NHÓM THỰC HIỆN



HMS Project Team

1. CÂU HỎI TRẮC NGHIỆM
2. CÂU HỎI TRẢ LỜI NGẮN
3. CÂU HỎI THẢO LUẬN NHÓM
4. CÂU HỎI TÌNH HUỐNG



1. **Câu hỏi 1:** Phần mềm bao gồm các loại nào dưới đây?
 - A. Phần mềm hệ thống
 - B. Phần mềm ứng dụng
 - C. Phần mềm nhúng
 - D. Cả A, B và C
2. **Câu hỏi 2:** Công nghệ phần mềm là gì?
 - A. Việc viết mã nguồn cho phần mềm
 - B. Phát triển phần mềm mà không có lỗi
 - C. Ứng dụng các phương pháp khoa học để phát triển phần mềm
 - D. Chỉ bảo trì phần mềm
3. **Câu hỏi 3:** Quy trình phát triển phần mềm gồm mấy giai đoạn chính?
 - A. 3
 - B. 4
 - C. 5
 - D. 6
4. **Câu hỏi 4:** Hoạt động nào dưới đây thuộc quy trình bảo trì phần mềm?
 - A. Lập kế hoạch
 - B. Triển khai phần mềm
 - C. Cập nhật phần mềm để phù hợp với thay đổi môi trường
 - D. Phân tích yêu cầu
5. **Câu hỏi 5:** Chi phí bảo trì phần mềm chiếm bao nhiêu phần trăm tổng chi phí vòng đời phần mềm?
 - A. 46%
 - B. 56%
 - C. 76%
 - D. 86%
6. **Câu hỏi 6:** Nguyên nhân chính gây ra việc vượt chi phí khi phát triển phần mềm là gì?
 - A. Thiếu nhân lực
 - B. Không xác định rõ yêu cầu
 - C. Thay đổi công nghệ
 - D. Cả A và C
7. **Câu hỏi 7:** Yêu cầu nào dưới đây không phải là yêu cầu phi chức năng?
 - A. Hiệu suất xử lý
 - B. Tính bảo mật
 - C. Khả năng mở rộng
 - D. Chức năng đăng nhập
8. **Câu hỏi 8:** Khi nào phần mềm được coi là hoàn thành?
 - A. Khi hoàn thành việc viết mã nguồn
 - B. Khi được bàn giao cho khách hàng và không còn lỗi
 - C. Khi được triển khai trên hệ thống của khách hàng
 - D. Khi được khách hàng chấp nhận và đưa vào sử dụng
9. **Câu hỏi 9:** Vấn đề phổ biến nào thường gặp khi phát triển phần mềm?
 - A. Thiếu công cụ hỗ trợ
 - B. Vượt chi phí, trễ thời hạn và lỗi sau khi bàn giao
 - C. Không có đội kiểm thử
 - D. Tất cả đều đúng
10. **Câu hỏi 10:** Phần mềm có thể được chia thành bao nhiêu loại chính?
 - A. 2
 - B. 3
 - C. 4
 - D. 5

2. CÂU HỎI NGẮN

1. Phần mềm là gì?
2. Công nghệ phần mềm là gì?
3. Các loại phần mềm chính là gì?
4. Tại sao công nghệ phần mềm lại quan trọng?
5. Quy trình phát triển phần mềm gồm những giai đoạn nào?
6. Khía cạnh kinh tế của công nghệ phần mềm là gì?
7. Khía cạnh công nghệ của công nghệ phần mềm là gì?
8. Khía cạnh bảo trì của công nghệ phần mềm là gì?
9. Các nguyên nhân chính gây trễ thời hạn khi phát triển phần mềm là gì?
10. Bảo trì phần mềm bao gồm những hoạt động nào?

1. Phân biệt phần mềm hệ thống và phần mềm ứng dụng.
2. Thảo luận về vai trò của công nghệ phần mềm trong lĩnh vực tài chính.
3. Nêu các thách thức thường gặp trong bảo trì phần mềm.
4. Vì sao phần mềm thương mại điện tử cần được bảo trì thường xuyên?
5. Phân tích những vấn đề khi yêu cầu khách hàng liên tục thay đổi trong quá trình phát triển phần mềm.
6. So sánh chi phí phát triển và chi phí bảo trì phần mềm.
7. Phân biệt các loại yêu cầu trong phát triển phần mềm (chức năng và phi chức năng).
8. Thảo luận về các mô hình quy trình phát triển phần mềm phổ biến.
9. Đề xuất giải pháp giảm thiểu lỗi phần mềm sau khi bàn giao.
10. Vai trò của đội kiểm thử trong quy trình phát triển phần mềm.

Tình huống 1:

Một công ty phát triển phần mềm quản lý tài chính đã hoàn thành dự án và bàn giao cho khách hàng. Tuy nhiên, sau 2 tháng sử dụng, khách hàng phát hiện ra nhiều lỗi phát sinh khi phần mềm xử lý các giao dịch có giá trị lớn. Hãy đề xuất giải pháp xử lý tình huống này.

Tình huống 2:

Trong quá trình phát triển phần mềm quản lý bệnh viện, khách hàng yêu cầu bổ sung thêm tính năng quản lý kho thuốc khi dự án đã đi vào giai đoạn kiểm thử. Là trưởng nhóm phát triển, bạn sẽ xử lý yêu cầu này như thế nào?

Tình huống 3:

Một nhóm phát triển phần mềm gặp phải vấn đề trễ tiến độ do nhiều thành viên không hiểu rõ yêu cầu của khách hàng. Là trưởng dự án, bạn sẽ làm gì để giải quyết vấn đề này và đảm bảo tiến độ dự án?

Tình huống 4:

Sau khi triển khai phần mềm quản lý thư viện, người dùng phản hồi rằng giao diện khó sử dụng và không thân thiện. Đội phát triển cần làm gì để cải thiện trải nghiệm người dùng?

Tình huống 5:

Một dự án phát triển phần mềm đã vượt quá ngân sách dự kiến do thời gian hoàn thành lâu hơn kế hoạch. Là quản lý dự án, bạn sẽ đề xuất những giải pháp nào để hạn chế việc vượt ngân sách trong tương lai?

Tình huống 6:

Trong quá trình bảo trì phần mềm quản lý khách sạn, một nhân viên phát hiện ra một lỗi nhỏ không ảnh hưởng lớn đến hoạt động. Tuy nhiên, chi phí để sửa lỗi này khá cao. Bạn sẽ quyết định sửa lỗi hay không? Vì sao?

Tình huống 7:

Khách hàng yêu cầu đội phát triển phải hoàn thành dự án sớm hơn 1 tháng so với kế hoạch ban đầu. Đội phát triển đang gặp khó khăn về nhân lực và tài nguyên. Bạn sẽ xử lý yêu cầu này như thế nào?

Tình huống 8:

Một công ty phần mềm nhỏ nhận được dự án phát triển ứng dụng di động. Do hạn chế về nguồn lực và kinh nghiệm, công ty đã liên tục thay đổi công nghệ sử dụng trong dự án. Điều này khiến dự án bị kéo dài và chi phí tăng cao. Bạn sẽ đưa ra giải pháp gì để khắc phục?

Tình huống 9:

Sau khi bàn giao phần mềm cho khách hàng, đội phát triển phát hiện ra một lỗi bảo mật nghiêm trọng có thể bị hacker khai thác. Là người phụ trách dự án, bạn sẽ giải quyết tình huống này như thế nào?

Tình huống 10:

Dự án phát triển hệ thống quản lý sản xuất đã được triển khai thành công tại nhà máy. Tuy nhiên, do thay đổi quy trình sản xuất, khách hàng yêu cầu sửa đổi phần mềm để phù hợp với quy trình mới. Đội phát triển cần làm gì để đáp ứng yêu cầu này mà không làm ảnh hưởng đến hoạt động sản xuất của khách hàng?

Software Process !!!

Software Process !!!

& Software Process !!!

Outline

2.1. CÁC WORKFLOW TRONG TIẾN TRÌNH PHẦN MỀM

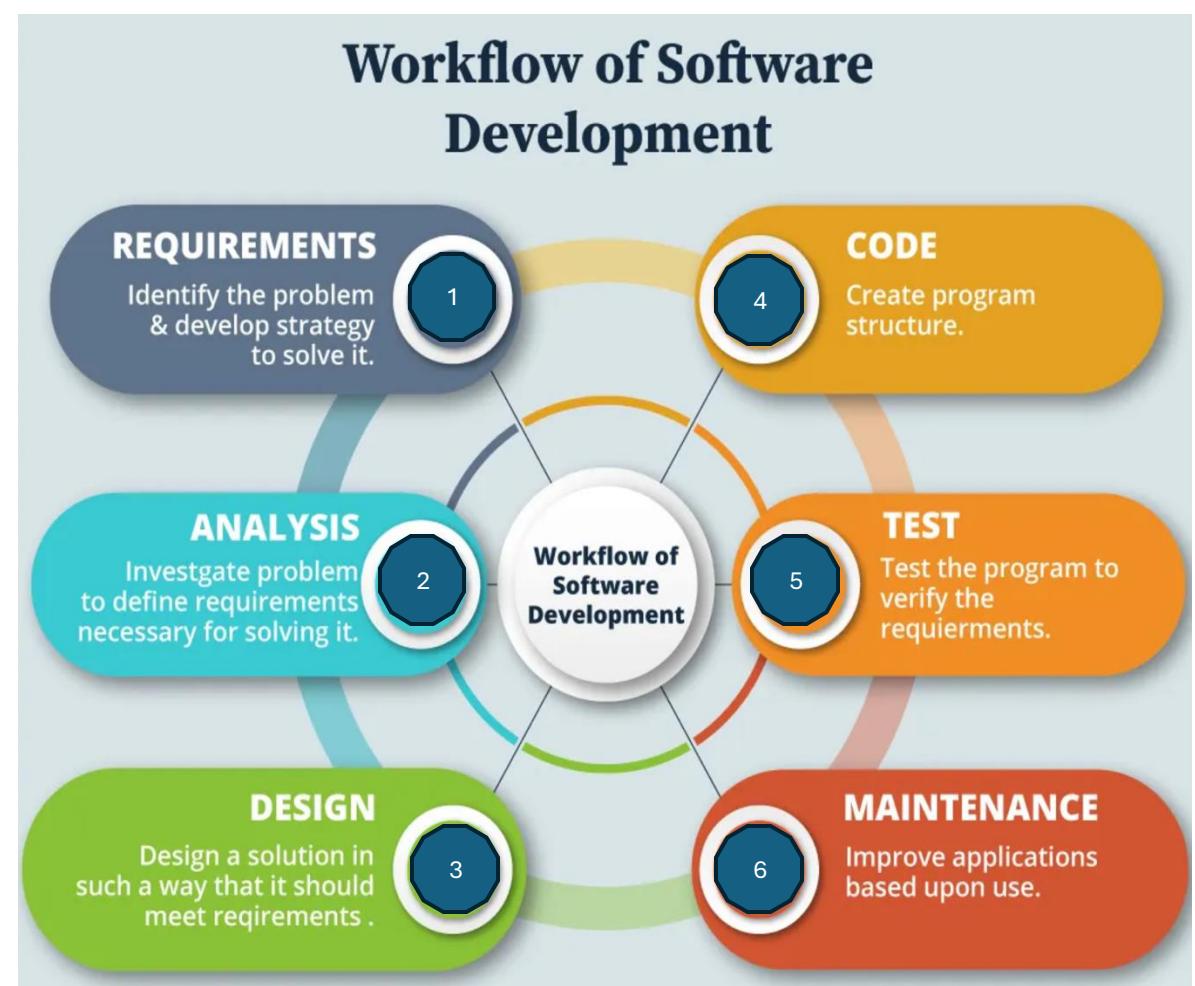
2.2. TIẾN TRÌNH THÔNG NHẤT (UNIFIED PROCESS)

2.3. CÁC MÔ HÌNH QUY CHUẨN CHẤT LƯỢNG CMM

2.4. CASE STUDY

đọc trong CHƯƠNG 3, tài liệu 1

Remember !!!!



Workflow lấy yêu cầu

- **Mục tiêu:** Xác định và ghi nhận tất cả các yêu cầu từ phía khách hàng.
- **Hoạt động chính:**
 - Phỏng vấn, khảo sát người dùng.
 - Ghi nhận yêu cầu chức năng và phi chức năng.
 - Tạo tài liệu yêu cầu phần mềm (SRS - Software Requirements Specification).
- **Kết quả:** Tài liệu yêu cầu phần mềm được duyệt bởi khách hàng.

2.1.2. Workflow phân tích

- **Mục tiêu:** Chuyển đổi các yêu cầu thành các đặc tả kỹ thuật để nhóm phát triển hiểu rõ.
- **Hoạt động chính:**
 - Phân tích hệ thống hiện tại (nếu có).
 - Phân rã yêu cầu thành các module và chức năng.
 - Xây dựng sơ đồ Use Case và ERD.
- **Kết quả:** Sơ đồ Use Case, ERD và tài liệu đặc tả phân tích hệ thống.

2.1.3. Workflow thiết kế

- **Mục tiêu:** Thiết kế chi tiết các thành phần phần mềm dựa trên kết quả phân tích.
- **Hoạt động chính:**
 - Thiết kế kiến trúc tổng thể.
 - Thiết kế chi tiết các module và giao diện.
 - Sử dụng UML để biểu diễn thiết kế (Class Diagram, Sequence Diagram).
- **Kết quả:** Tài liệu thiết kế chi tiết và sơ đồ UML.

2.1.4. Workflow cài đặt

- **Mục tiêu:** Biến các thiết kế thành mã nguồn thực tế.
- **Hoạt động chính:**
 - Viết mã nguồn cho từng module.
 - Tích hợp các module.
 - Tuân thủ quy tắc viết mã và tiêu chuẩn lập trình.
- **Kết quả:** Sản phẩm phần mềm chạy được và hoàn chỉnh.

2.1.5. Workflow kiểm thử

- **Mục tiêu:** Đảm bảo rằng phần mềm hoạt động đúng như mong đợi.
- **Hoạt động chính:**
 - Thực hiện kiểm thử đơn vị, kiểm thử tích hợp và kiểm thử hệ thống.
 - Sửa lỗi phát hiện trong quá trình kiểm thử.
 - Lập báo cáo kiểm thử.
- **Kết quả:** Phần mềm đạt tiêu chuẩn chất lượng và sẵn sàng triển khai.

Các pha trong Tiến trình Thống nhất

1. Pha khởi đầu (**Inception**):

Mục tiêu chính là hiểu rõ các yêu cầu cơ bản và đánh giá tính khả thi của dự án.

2. Pha làm rõ (**Elaboration**):

Tập trung vào phân tích và thiết kế hệ thống, giải quyết các **rủi ro** lớn.

3. Pha xây dựng (**Construction**):

Triển khai các module phần mềm và **kiểm thử** chúng.

4. Pha chuyển giao (**Transition**):

Phần mềm được triển khai và **bàn giao** cho khách hàng sử dụng.

❖ Mục tiêu:

- Xác định phạm vi dự án và đánh giá tính khả thi.
- Hiểu rõ yêu cầu của khách hàng để tránh sai sót sau này.
- Đánh giá nguồn lực và chi phí để đảm bảo dự án có thể thực hiện được.

❖ Giải pháp:

- Thu thập yêu cầu đầy đủ:** Sử dụng **phỏng vấn, khảo sát, mô hình Use Case** để hiểu rõ nhu cầu khách hàng.
- Phân tích tính khả thi:** Đánh giá **kỹ thuật, tài chính, nhân sự** để đảm bảo dự án khả thi.
- Xác định phạm vi dự án:** Tránh mở rộng phạm vi quá lớn ngay từ đầu để tránh rủi ro.

➔ **Bài học cần nhớ:**

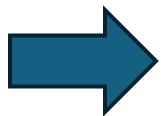
Làm tốt pha khởi đầu giúp tránh các vấn đề về chi phí, tiến độ và phạm vi dự án sau này.

◆ Mục tiêu:

- Xây dựng kiến trúc phần mềm, xác định giải pháp công nghệ.
- Phân tích và giải quyết các rủi ro quan trọng trước khi bắt đầu lập trình.
- Tạo nguyên mẫu (Prototype) để minh họa chức năng hệ thống.

◆ Giải pháp:

- Thiết kế kiến trúc hệ thống:** Xây dựng sơ đồ UML (Use Case, Class, Sequence, Activity, ERD) để mô hình hóa hệ thống.
- Phân tích và giải quyết rủi ro:** Xác định các vấn đề kỹ thuật, bảo mật, hiệu suất có thể gặp phải và đề xuất giải pháp.
- Tạo nguyên mẫu (Prototype):** Giúp khách hàng hiểu rõ sản phẩm trước khi lập trình.



💡 Bài học cần nhớ:

Một thiết kế tốt trong pha làm rõ giúp hệ thống ổn định, dễ mở rộng và bảo trì

📌 Mục tiêu:

- Phát triển phần mềm theo thiết kế đã đề ra.
- Kiểm thử liên tục để đảm bảo chất lượng.
- Hoàn thành sản phẩm theo từng giai đoạn nhỏ để dễ kiểm soát.

📌 Giải pháp:

- **Chia nhỏ công việc theo Sprint (Agile) hoặc Module (Waterfall):** Đảm bảo kiểm soát chất lượng từng phần.
- **Áp dụng DevOps & CI/CD:** Tích hợp liên tục, kiểm thử tự động để giảm lỗi.
- **Kiểm thử sớm (Shift-left testing):** Áp dụng **Unit Test, Integration Test, System Test** ngay trong giai đoạn lập trình.



💡 Bài học cần nhớ:

Lập trình đúng quy trình và kiểm thử sớm giúp tránh lỗi nghiêm trọng và tiết kiệm chi phí sửa lỗi.

➤ Mục tiêu:

- Đưa phần mềm vào vận hành thực tế.
- Đảm bảo phần mềm hoạt động ổn định khi triển khai.
- Cung cấp tài liệu hướng dẫn và hỗ trợ khách hàng sử dụng phần mềm.

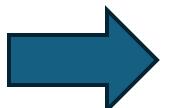
➤ Giải pháp:

- Kiểm thử lần cuối: User Acceptance Testing (UAT)** để đảm bảo phần mềm hoạt động đúng mong đợi.
- Hướng dẫn và hỗ trợ khách hàng**: Cung cấp tài liệu hướng dẫn, đào tạo sử dụng phần mềm.
- Lập kế hoạch bảo trì**: Chuẩn bị sẵn kế hoạch cập nhật, sửa lỗi sau triển khai.



Bài học cần nhớ:

Quản lý tốt pha chuyển giao giúp nâng cao uy tín và giảm chi phí bảo trì sau này.



- ISO - International Organization for Standardization/ IEEE - Electrical and Electronics Engineers

1. ISO/IEC 12207: Quy trình vòng đời phần mềm(Software Life Cycle Processes)

- Tiêu chuẩn này cung cấp một khung quy trình toàn diện cho vòng đời phần mềm, bao gồm các hoạt động từ lập kế hoạch, phát triển đến vận hành và bảo trì. Nó xác định các quy trình, hoạt động và nhiệm vụ có thể áp dụng trong suốt quá trình mua lại và cầu hình sản phẩm, dịch vụ phần mềm.

2. ISO/IEC 29119: Kiểm thử phần mềm (Software Testing)

- ISO/IEC 29119 là một bộ tiêu chuẩn cung cấp các hướng dẫn thực tiễn tốt nhất cho quy trình kiểm thử phần mềm. Nó bao gồm các quy trình kiểm thử, tài liệu kiểm thử, kỹ thuật kiểm thử và phương pháp kiểm thử dựa trên từ khóa, nhằm cải thiện hiệu suất và hiệu quả của quá trình kiểm thử.

3. ISO/IEC 27001: Hệ thống quản lý an toàn thông tin (ISMS) (Information Security Management Systems (ISMS))

- Tiêu chuẩn này quy định các yêu cầu để thiết lập, triển khai, duy trì và cải tiến liên tục hệ thống quản lý an toàn thông tin. Nó giúp các tổ chức bảo vệ tài sản thông tin bằng cách đánh giá có hệ thống các rủi ro bảo mật và triển khai các biện pháp kiểm soát an toàn thông tin toàn diện.

4. ISO/IEC 90003: Kỹ thuật phần mềm – Hướng dẫn áp dụng ISO 9001 cho phần mềm máy tính (Software Engineering—Guidelines for the Application of ISO 9001 to Computer Software)

ISO/IEC 90003 cung cấp hướng dẫn cho các tổ chức về cách áp dụng tiêu chuẩn ISO 9001:2015 vào quá trình mua lại, cung cấp, phát triển, vận hành và bảo trì phần mềm máy tính và các dịch vụ hỗ trợ liên quan. Nó nhấn mạnh các nguyên tắc quản lý chất lượng được điều chỉnh cho lĩnh vực kỹ thuật phần mềm.

5. ISO/IEC 25010: Mô hình chất lượng hệ thống và phần mềm (System and Software Quality Models)

- Tiêu chuẩn này định nghĩa mô hình chất lượng cho hệ thống và sản phẩm phần mềm, mô tả các đặc tính như tính năng, độ tin cậy, khả năng sử dụng, hiệu suất, bảo trì và khả năng di động. Nó đóng vai trò như một khung chuẩn để đánh giá chất lượng phần mềm.

6. ISO/IEC 15288: Quy trình vòng đời hệ thống (System Life Cycle Processes)

- ISO/IEC 15288 thiết lập một khung chung để mô tả vòng đời của các hệ thống do con người tạo ra. Nó định nghĩa một tập hợp các quy trình và thuật ngữ liên quan đến toàn bộ vòng đời hệ thống, có thể áp dụng cho các hệ thống phần mềm trong một bối cảnh rộng hơn.

7. ISO/IEC 15504: Đánh giá quy trình (SPICE) (Process Assessment (SPICE))

- Còn được gọi là "Cải tiến và Xác định Năng lực Quy trình Phần mềm" (SPICE), tiêu chuẩn này cung cấp một khung đánh giá và cải thiện quy trình phát triển phần mềm. Nó giúp các tổ chức xác định năng lực quy trình và xác định các lĩnh vực cần cải tiến.

2.3.1. CMM mức 1 – Initial (Ban đầu)

- Quy trình không ổn định, phụ thuộc nhiều vào cá nhân.
- Phần lớn dự án thành công nhờ vào nỗ lực cá nhân thay vì tổ chức.

2.3.2. CMM mức 2 – Managed (Quản lý)

- Quy trình được quản lý ở mức cơ bản.
- Các hoạt động như lập kế hoạch, quản lý rủi ro được thực hiện.

2.3.3. CMM mức 3 – Defined (Định nghĩa)

- Quy trình được định nghĩa rõ ràng và nhất quán trong toàn tổ chức.
- Các tiêu chuẩn quy trình được xây dựng và áp dụng.

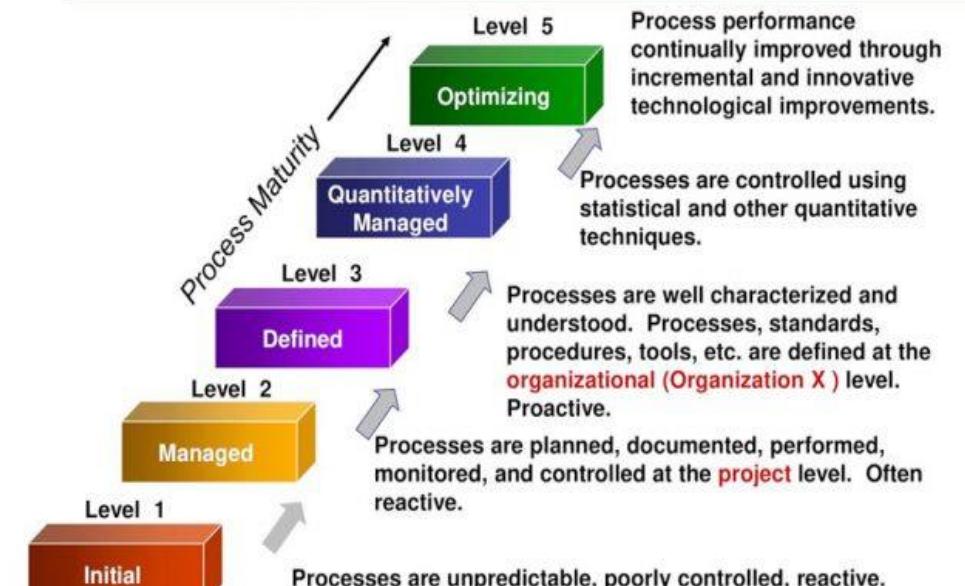
2.3.4. CMM mức 4 – Quantitatively Managed (Quản lý định lượng)

- Quy trình được đo lường và kiểm soát bằng dữ liệu định lượng.
- Tập trung vào việc giảm thiểu sai sót và cải tiến quy trình.

2.3.5. CMM mức 5 – Optimizing (Tối ưu hóa)

- Quy trình liên tục được cải tiến dựa trên phản hồi và dữ liệu.
- Mục tiêu là đạt được sự hoàn hảo trong phát triển phần mềm.

CMMI Staged Representation - 5 Maturity Levels



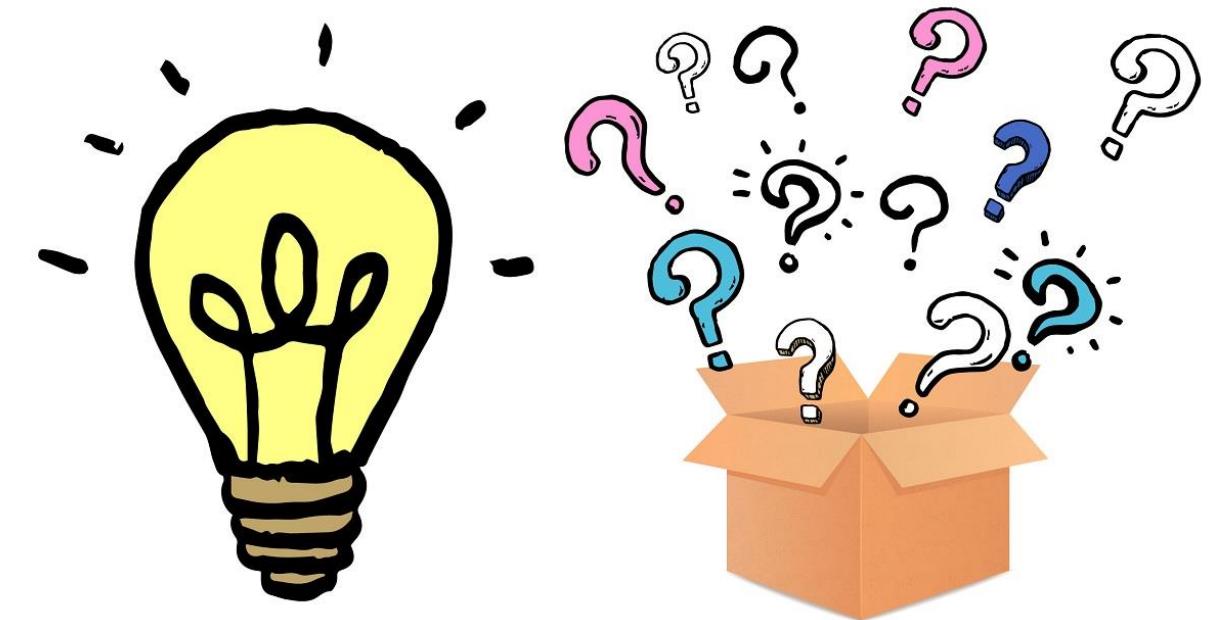
Hoạt động nhóm:

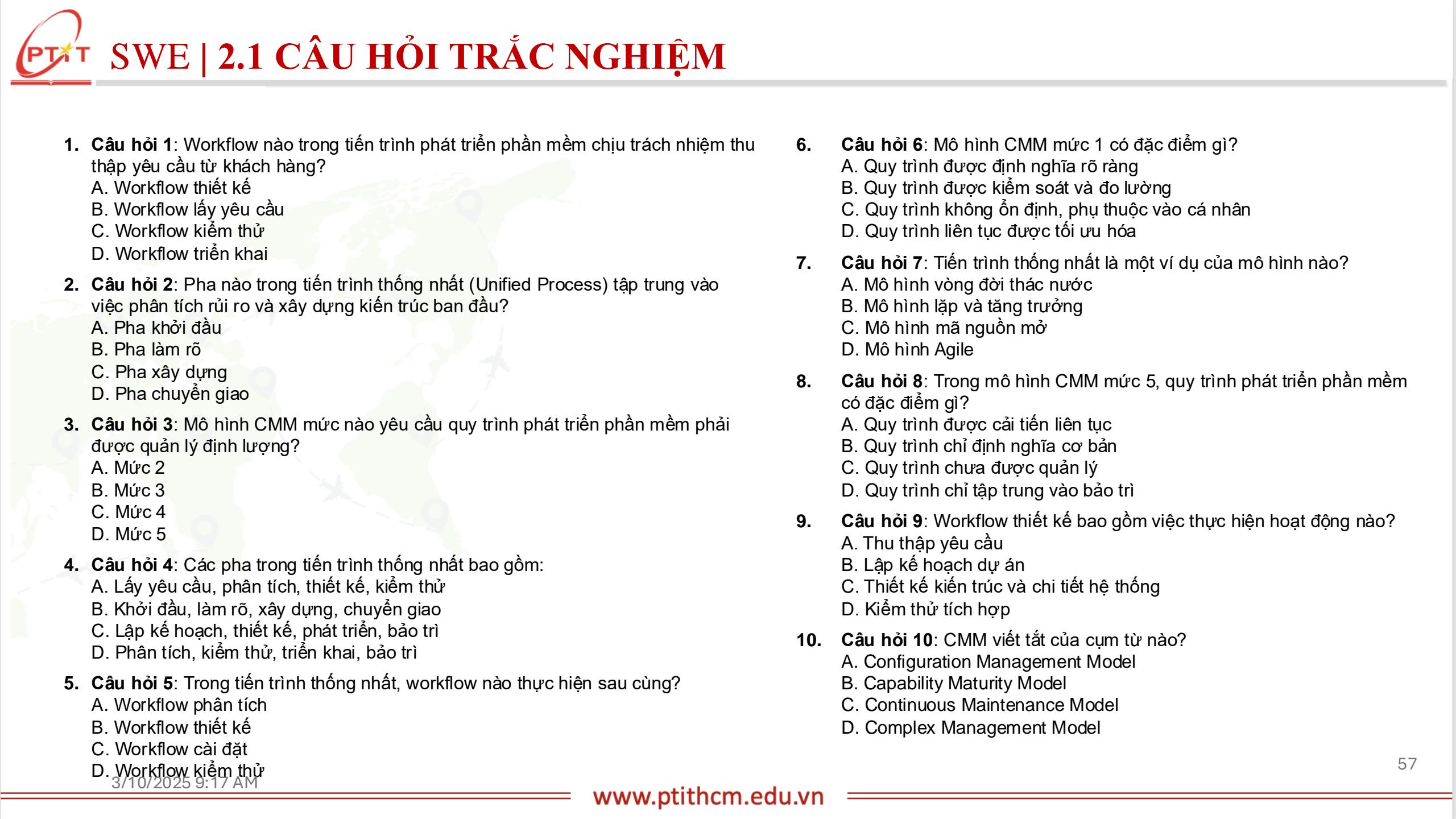
- Chia nhóm (3-5 sinh viên mỗi nhóm) để nghiên cứu về các tiến trình phần mềm thực tế của một công ty hoặc dự án cụ thể.
- Các nhóm sẽ thuyết trình và báo cáo kết quả nghiên cứu của mình trước lớp.

Các công ty hoặc dự án gợi ý:

- Dự án phát triển ứng dụng thương mại điện tử.
- Dự án phát triển phần mềm quản lý bệnh viện.
- Dự án phát triển phần mềm ngân hàng trực tuyến.

1. CÂU HỎI TRẮC NGHIỆM
2. CÂU HỎI TRẢ LỜI NGẮN
3. CÂU HỎI THẢO LUẬN NHÓM
4. CÂU HỎI TÌNH HUỐNG



- 
1. **Câu hỏi 1:** Workflow nào trong tiến trình phát triển phần mềm chịu trách nhiệm thu thập yêu cầu từ khách hàng?
 - A. Workflow thiết kế
 - B. Workflow lấy yêu cầu
 - C. Workflow kiểm thử
 - D. Workflow triển khai
 2. **Câu hỏi 2:** Pha nào trong tiến trình thống nhất (Unified Process) tập trung vào việc phân tích rủi ro và xây dựng kiến trúc ban đầu?
 - A. Pha khởi đầu
 - B. Pha làm rõ
 - C. Pha xây dựng
 - D. Pha chuyển giao
 3. **Câu hỏi 3:** Mô hình CMM mức nào yêu cầu quy trình phát triển phần mềm phải được quản lý định lượng?
 - A. Mức 2
 - B. Mức 3
 - C. Mức 4
 - D. Mức 5
 4. **Câu hỏi 4:** Các pha trong tiến trình thống nhất bao gồm:
 - A. Lấy yêu cầu, phân tích, thiết kế, kiểm thử
 - B. Khởi đầu, làm rõ, xây dựng, chuyển giao
 - C. Lập kế hoạch, thiết kế, phát triển, bảo trì
 - D. Phân tích, kiểm thử, triển khai, bảo trì
 5. **Câu hỏi 5:** Trong tiến trình thống nhất, workflow nào thực hiện sau cùng?
 - A. Workflow phân tích
 - B. Workflow thiết kế
 - C. Workflow cài đặt
 - D. Workflow kiểm thử
 6. **Câu hỏi 6:** Mô hình CMM mức 1 có đặc điểm gì?
 - A. Quy trình được định nghĩa rõ ràng
 - B. Quy trình được kiểm soát và đo lường
 - C. Quy trình không ổn định, phụ thuộc vào cá nhân
 - D. Quy trình liên tục được tối ưu hóa
 7. **Câu hỏi 7:** Tiến trình thống nhất là một ví dụ của mô hình nào?
 - A. Mô hình vòng đời thác nước
 - B. Mô hình lặp và tăng trưởng
 - C. Mô hình mã nguồn mở
 - D. Mô hình Agile
 8. **Câu hỏi 8:** Trong mô hình CMM mức 5, quy trình phát triển phần mềm có đặc điểm gì?
 - A. Quy trình được cải tiến liên tục
 - B. Quy trình chỉ định nghĩa cơ bản
 - C. Quy trình chưa được quản lý
 - D. Quy trình chỉ tập trung vào bảo trì
 9. **Câu hỏi 9:** Workflow thiết kế bao gồm việc thực hiện hoạt động nào?
 - A. Thu thập yêu cầu
 - B. Lập kế hoạch dự án
 - C. Thiết kế kiến trúc và chi tiết hệ thống
 - D. Kiểm thử tích hợp
 10. **Câu hỏi 10:** CMM viết tắt của cụm từ nào?
 - A. Configuration Management Model
 - B. Capability Maturity Model
 - C. Continuous Maintenance Model
 - D. Complex Management Model

1. Pha khởi đầu trong tiến trình thống nhất là gì?
2. Mục tiêu của workflow lấy yêu cầu là gì?
3. Tiến trình thống nhất gồm bao nhiêu pha chính?
4. Sự khác nhau giữa CMM mức 2 và mức 3 là gì?
5. Workflow kiểm thử có nhiệm vụ gì?
6. Mô hình CMM có bao nhiêu mức?
7. Khác biệt giữa mô hình thác nước và mô hình lặp là gì?
8. Tiến trình thống nhất có phải là mô hình lặp không?
9. Mục đích của workflow thiết kế là gì?
10. CMM mức 5 tập trung vào điều gì?

1. Thảo luận về vai trò của từng workflow trong tiến trình phát triển phần mềm.
2. Phân biệt mô hình vòng đòn thác nước và tiến trình thống nhất.
3. Thảo luận về các ưu và nhược điểm của mô hình lặp và tăng trưởng.
4. Vì sao mô hình CMM được sử dụng rộng rãi trong quản lý chất lượng phần mềm?
5. Thảo luận về các khó khăn khi áp dụng mô hình CMM trong thực tế.
6. Đề xuất các giải pháp để cải tiến quy trình phát triển phần mềm.
7. Phân tích ưu điểm của việc áp dụng tiến trình thống nhất trong các dự án lớn.
8. Thảo luận về sự cần thiết của việc kiểm thử trong từng pha của tiến trình thống nhất.
9. So sánh giữa mô hình CMM mức 4 và mức 5.
10. Đề xuất cách tổ chức hoạt động nhóm trong workflow lấy yêu cầu

1. Một công ty phát triển phần mềm gặp khó khăn khi yêu cầu của khách hàng liên tục thay đổi trong pha xây dựng. Đội phát triển nên làm gì để giải quyết vấn đề này?
2. Trong pha chuyển giao của tiến trình thông nhất, khách hàng yêu cầu bổ sung thêm tính năng mới. Đội phát triển nên xử lý ra sao?
3. Dự án phát triển phần mềm bị trễ tiến độ do lỗi phát sinh liên tục trong quá trình kiểm thử. Là trưởng dự án, bạn sẽ làm gì?
4. Trong workflow thiết kế, kiến trúc sư phần mềm muốn thay đổi thiết kế ban đầu để cải thiện hiệu suất. Đội phát triển nên xử lý thế nào?
5. Khách hàng yêu cầu rút ngắn thời gian phát triển dự án mà không thay đổi yêu cầu. Đội phát triển nên phản ứng ra sao?
6. Một công ty nhỏ muốn áp dụng mô hình CMM nhưng gặp khó khăn do thiếu nguồn lực. Hãy đề xuất giải pháp.
7. Trong workflow lấy yêu cầu, khách hàng cung cấp thông tin không rõ ràng. Đội phát triển cần làm gì?
8. Một dự án gặp rủi ro cao trong pha khởi đầu do thiếu tài liệu yêu cầu rõ ràng. Đội phát triển nên làm gì?
9. Dự án phần mềm lớn có nhiều nhóm phát triển ở các địa điểm khác nhau. Làm thế nào để đảm bảo các nhóm phối hợp hiệu quả?
10. Một công ty phát triển phần mềm gặp khó khăn trong việc quản lý quy trình do không có chuẩn hóa. Hãy đề xuất giải pháp.

SWE | 03. MỘT SỐ MÔ HÌNH VÒNG ĐỜI PHÁT TRIỂN PHẦN MỀM

OUTLINE

3.1. MÔ HÌNH LÝ THUYẾT VÒNG ĐỜI PHÁT TRIỂN PHẦN MỀM

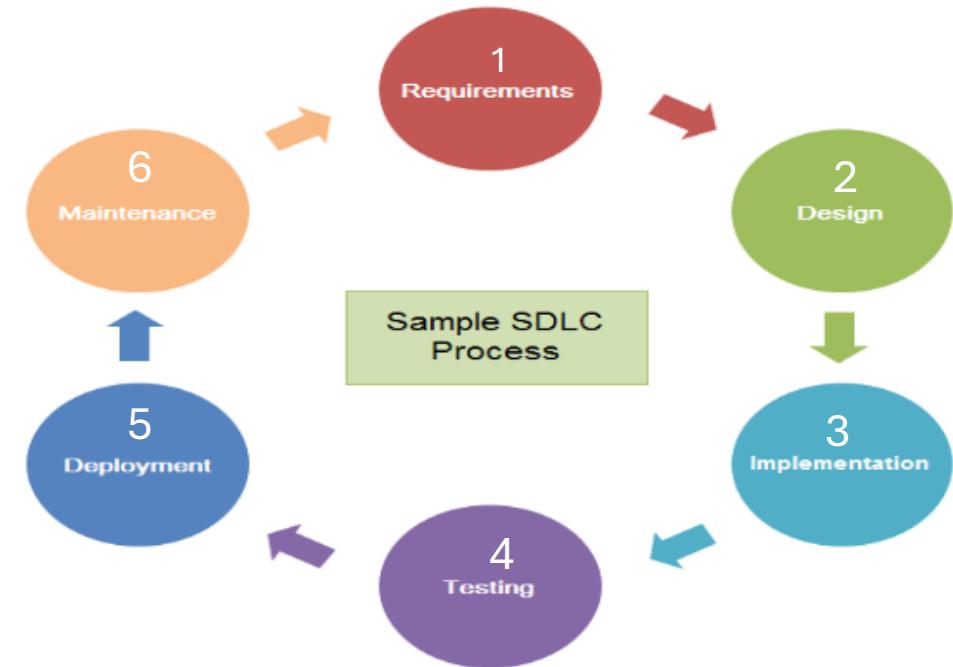
3.2. MỘT SỐ MÔ HÌNH VÒNG ĐỜI PHÁT TRIỂN PHẦN MỀM

3.3. CASE STUDY

3.4. CÂU HỎI VẬN DỤNG LÝ THUYẾT



Remember !!!

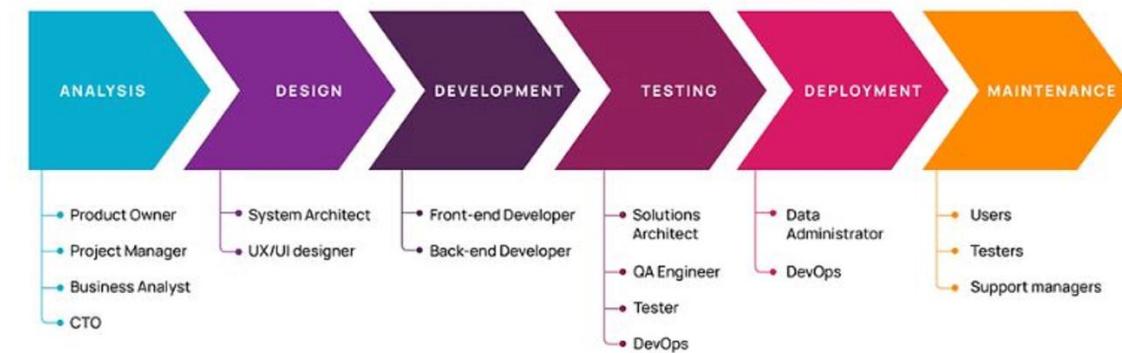


3.1. Mô hình lý thuyết vòng đời phát triển phần mềm

Định nghĩa

- Mô hình lý thuyết vòng đời phát triển phần mềm (**Software Development Life Cycle - SDLC**) là quy trình tiêu chuẩn giúp lập kế hoạch, thiết kế, phát triển, kiểm thử, triển khai và bảo trì phần mềm.
- Mục tiêu của mô hình này là **đảm bảo phần mềm chất lượng cao, giảm thiểu lỗi phát sinh và tối ưu hóa chi phí phát triển**.

6 Phases of the Software Development Life Cycle



CÁC GIAI ĐOẠN CHÍNH CỦA SDLC

❖ Mô hình vòng đời phần mềm thường gồm các bước sau:

◆ 1. Lấy yêu cầu (Requirement Analysis)

- Thu thập và phân tích yêu cầu khách hàng.
- Xác định phạm vi, chức năng chính, tính khả thi của dự án.

◆ 2. Thiết kế (Design)

- Lên kế hoạch thiết kế kiến trúc phần mềm.
- Xây dựng sơ đồ UML, thiết kế cơ sở dữ liệu, giao diện người dùng.

◆ 3. Lập trình (Implementation)

- Chuyển thiết kế thành mã nguồn thực thi.
- Chia nhỏ module, lập trình theo tiêu chuẩn coding.

◆ 4. Kiểm thử (Testing & QA)

- Kiểm thử đơn vị, tích hợp, hệ thống, bảo mật, hiệu suất.
- Đảm bảo phần mềm chạy đúng và ổn định.

◆ 5. Triển khai (Deployment)

- Cài đặt phần mềm trên hệ thống thực tế hoặc máy chủ.
- Đào tạo người dùng và cung cấp tài liệu hướng dẫn.

◆ 6. Bảo trì (Maintenance & Updates)

- Cập nhật, sửa lỗi, nâng cấp phần mềm theo phản hồi từ khách hàng.
- Cải thiện hiệu suất và tối ưu tính năng.

3.1.1. Pha lấy yêu cầu

Mục tiêu:

- Thu thập và phân tích các yêu cầu từ khách hàng để hiểu rõ những gì phần mềm cần thực hiện.

Hoạt động chính:

- Tổ chức phỏng vấn, khảo sát khách hàng.
- Xác định các yêu cầu chức năng và phi chức năng.
- Lập tài liệu yêu cầu phần mềm (SRS).



3.1.2. **Pha phân tích**

Mục tiêu:

- Chuyển đổi các yêu cầu đã thu thập thành các đặc tả kỹ thuật.

Hoạt động chính:

- Phân rã yêu cầu thành các thành phần nhỏ hơn.
- Xây dựng mô hình phân tích như sơ đồ Use Case và ERD.
- Xác định các ràng buộc và giả định.



3.1.3. Pha thiết kế

Mục tiêu:

- Lập kế hoạch chi tiết về cách phần mềm sẽ được phát triển.

Hoạt động chính:

- Thiết kế kiến trúc phần mềm.
- Thiết kế chi tiết từng module và giao diện.
- Sử dụng sơ đồ UML để mô tả chi tiết hệ thống.



3.1.4. **Pha cài đặt**

Mục tiêu:

- Chuyển đổi các thiết kế thành mã nguồn thực tế.

Hoạt động chính:

- Viết mã nguồn cho từng module.
- Thực hiện kiểm thử đơn vị cho từng module.
- Tích hợp các module và kiểm thử tích hợp.



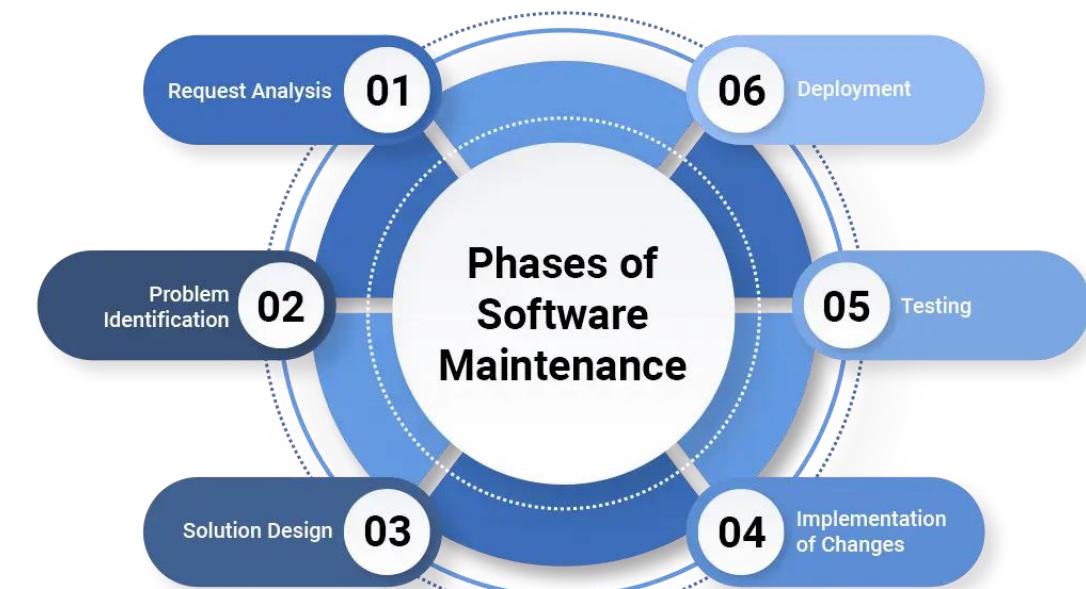
3.1.5. Pha bảo trì

Mục tiêu:

- Đảm bảo phần mềm tiếp tục hoạt động ổn định và đáp ứng các yêu cầu mới.

Hoạt động chính:

- Sửa lỗi phần mềm.
- Cải tiến phần mềm theo yêu cầu mới.
- Điều chỉnh phần mềm để tương thích với môi trường mới.



3.1.6. Pha giải thể

Mục tiêu:

- Kết thúc vòng đời của phần mềm khi phần mềm không còn giá trị sử dụng.

Hoạt động chính:

- Gỡ bỏ phần mềm khỏi hệ thống.
- Lưu trữ tài liệu và dữ liệu cần thiết.
- Đảm bảo dữ liệu được bảo mật khi giải thể phần mềm.

**Đặc Điểm Chính
của Giai Đoạn
Giải Thể.**

- Kết thúc việc sử dụng phần mềm (End of Software Usage).
- Chuyển đổi dữ liệu (Data Migration)
- Ngừng hoạt động hệ thống (System Decommissioning).
- Tài liệu hóa và lưu trữ (Documentation and Archival).
- Phân bổ lại nguồn lực (Resource Reallocation).
- Xem xét hệ thống cũ (Legacy Considerations)

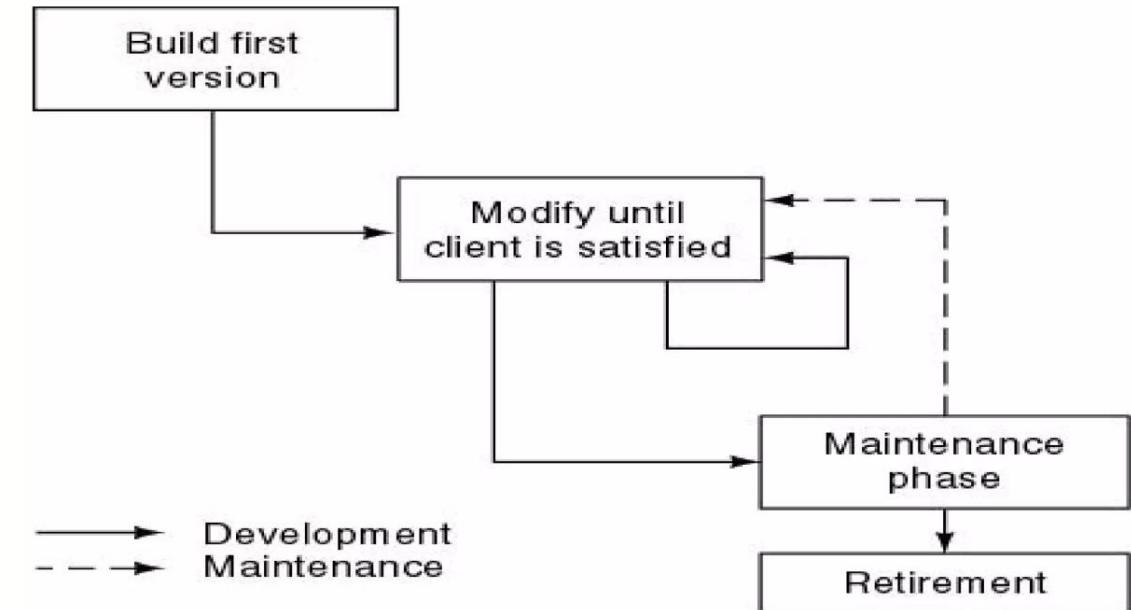
3.2.1. Mô hình xây và sửa (Build and Fix Model)

Đặc điểm:

❑ Phát triển nhanh, không có quy trình rõ ràng, dễ phát sinh lỗi khi mở rộng.

Ưu điểm: Phù hợp cho các dự án nhỏ, yêu cầu đơn giản.

Nhược điểm: Khó bảo trì và nâng cấp.

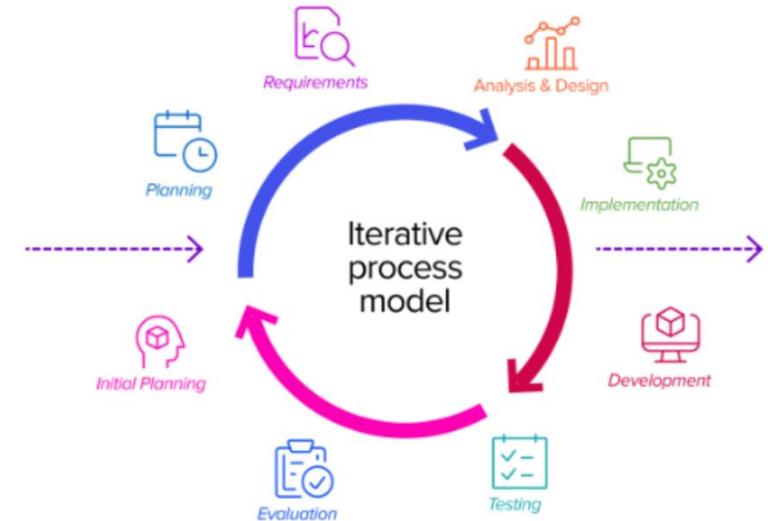


3.2.2. Mô hình lặp và tăng trưởng (Iterative and Incremental Model)

Đặc điểm: Phát triển phần mềm theo từng đợt lặp lại, mỗi đợt cải tiến dần phần mềm.

Ưu điểm: Giảm thiểu rủi ro và thích nghi tốt với thay đổi yêu cầu.

Nhược điểm: Tốn nhiều công sức quản lý và kiểm thử.



□ Định nghĩa

Mô hình thác nước (**Waterfall Model**) là một **mô hình phát triển phần mềm tuyến tính và tuần tự**, trong đó mỗi giai đoạn phát triển phải được hoàn thành trước khi chuyển sang giai đoạn tiếp theo. Đây là một trong những mô hình phát triển phần mềm **cổ điển và truyền thống nhất**.

□ Ưu điểm của mô hình Waterfall:

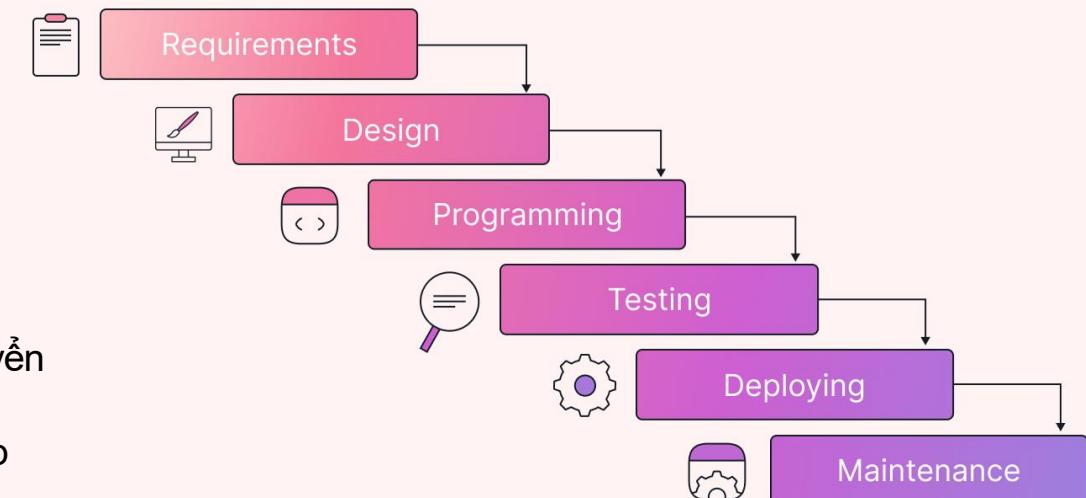
- Dễ hiểu và dễ quản lý do có **cấu trúc chặt chẽ**.
- Phù hợp với **dự án có yêu cầu rõ ràng, ít thay đổi**.
- Tài liệu đầy đủ giúp việc bảo trì và nâng cấp dễ dàng.

□ Nhược điểm của mô hình Waterfall:

- **Khó thay đổi yêu cầu** sau khi đã bắt đầu giai đoạn tiếp theo
- **Tốn nhiều thời gian** vì các giai đoạn phải hoàn thành trước khi chuyển bước
- **Khó kiểm tra sớm lỗi** do kiểm thử chỉ diễn ra sau khi hoàn thành lập trình.

□ Khi nào nên sử dụng mô hình Waterfall?

- Khi yêu cầu dự án rõ ràng và không thay đổi.
- Khi dự án có thời gian và ngân sách cố định.
- Khi nhóm phát triển ưu tiên tài liệu đầy đủ và quy trình chuẩn hóa.



Kết luận:

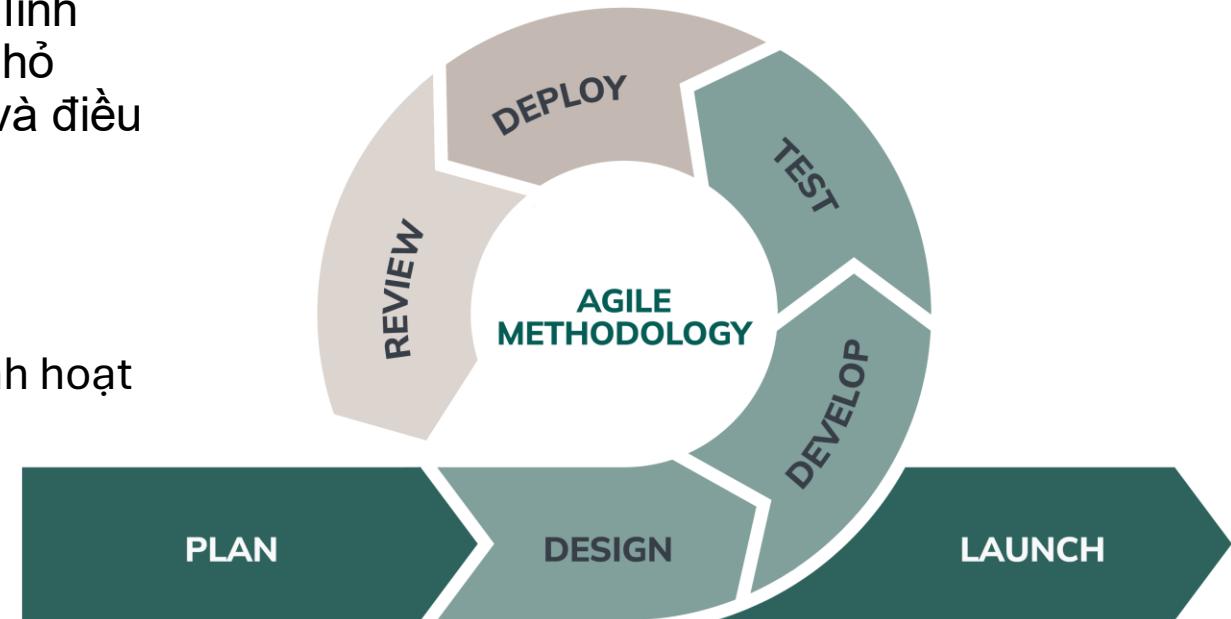
- Mô hình Waterfall phù hợp với những dự án phần mềm **ổn định, có yêu cầu rõ ràng ngay từ đầu**. Tuy nhiên, nếu dự án có **nhiều thay đổi linh hoạt** thì **Agile Model** có thể là lựa chọn phù hợp hơn.

❑ Định nghĩa

- Agile Model là một phương pháp phát triển phần mềm linh hoạt, trong đó dự án được chia thành nhiều vòng lặp nhỏ (iterations/sprints). Mỗi vòng lặp có thể được kiểm tra và điều chỉnh dựa trên phản hồi của khách hàng.

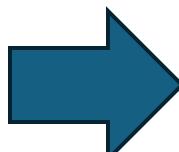
❑ Đặc điểm chính của Agile Model:

- Phát triển theo từng giai đoạn ngắn (Sprints)
- Khách hàng tham gia liên tục, điều chỉnh yêu cầu linh hoạt
- Kiểm thử liên tục giúp phát hiện lỗi sớm
- Nhóm phát triển tự quản lý, tăng hiệu suất làm việc.
- Dễ thay đổi yêu cầu ngay cả khi dự án đang diễn ra.



❑ Agile Development Cycle:

- 1 Xác định yêu cầu
- 2 Lập kế hoạch Sprint
- 3 Thiết kế & phát triển
- 4 Kiểm thử
- 5 Triển khai & phản hồi
- 6 Cải tiến & lập kế hoạch cho Sprint tiếp theo



Khi nào nên dùng Agile?

- Dự án có yêu cầu thay đổi liên tục.
- Cần tương tác thường xuyên với khách hàng.
- Dự án có thời gian dài hạn, cần cải tiến liên tục.

□ CÁC PHƯƠNG PHÁP AGILE PHỔ BIẾN

1 Scrum

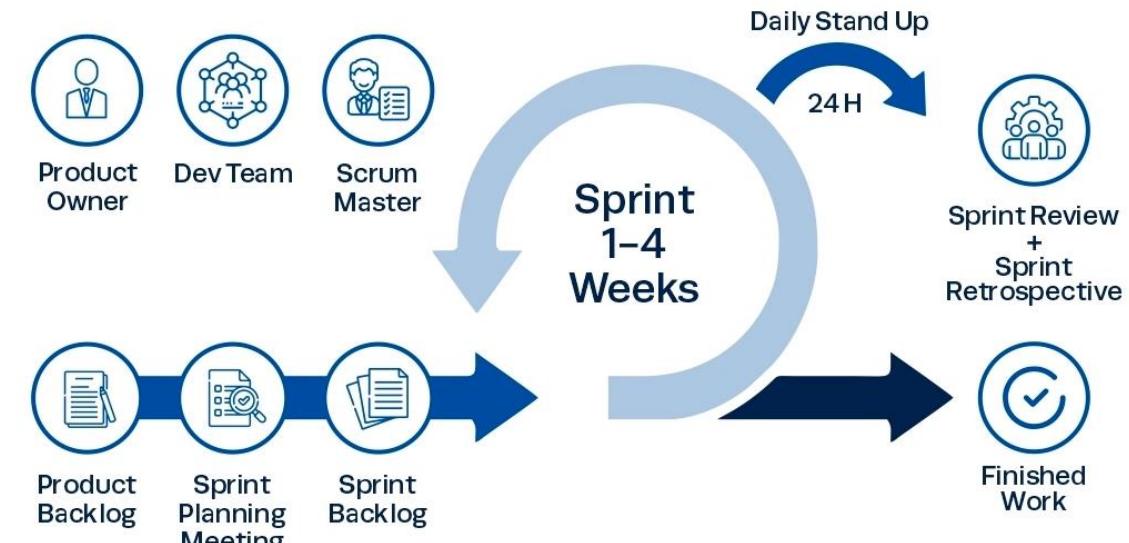
- Chia dự án thành các chu kỳ ngắn (**Sprint**, thường từ 1-4 tuần).

□ Có các vai trò chính:

- Product Owner** – Định nghĩa yêu cầu sản phẩm.
- Scrum Master** – Hỗ trợ nhóm phát triển theo quy trình Scrum.
- Development Team** – Thực hiện các công việc trong Sprint.

□ Các cuộc họp quan trọng:

- Daily Stand-up** – Họp ngắn mỗi ngày (15 phút) để cập nhật tiến độ.
- Sprint Review** – Trình bày kết quả sau mỗi Sprint.
- Sprint Retrospective** – Nhìn lại những gì làm tốt và cần cải thiện.



Khi nào nên dùng Scrum?

- Dự án có yêu cầu thay đổi liên tục.
- Cần tương tác thường xuyên với khách hàng.
- Dự án có thời gian dài hạn, cần cải tiến liên tục.

□ CÁC PHƯƠNG PHÁP AGILE PHỔ BIẾN (tt)

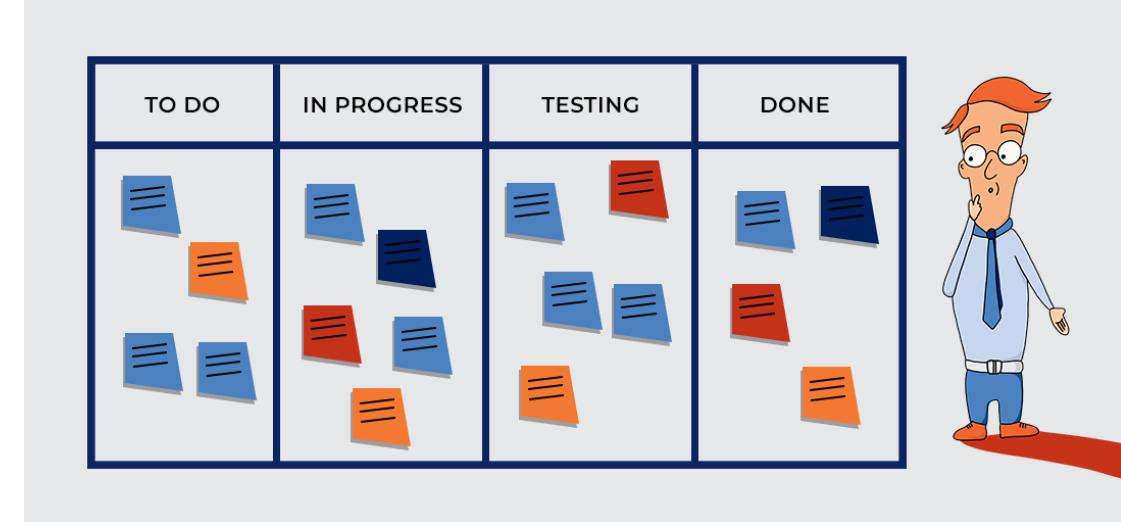
2 Kanban

Tập trung vào trực quan hóa quy trình làm việc bằng bảng Kanban.

Mỗi công việc được chia thành **các cột trạng thái** như:

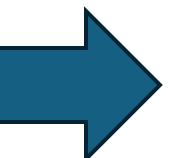
- To Do (Việc cần làm)
- In Progress (Đang thực hiện)
- Done (Hoàn thành)

Hạn chế số lượng công việc đang thực hiện cùng lúc để tăng hiệu suất.



Khi nào nên dùng Kanban?

- Khi Nhóm Cần Quản Lý Công Việc Liên Tục (Continuous Workflow).
- Khi Nhóm Muốn Tối Ưu Hiệu Suất & Giảm Tồn Đọng Công Việc.
- Khi Cần Minh Bạch Quy Trình Làm Việc
- Khi Làm Việc Với Các Đội Nhóm Khác Nhau



□ CÁC PHƯƠNG PHÁP AGILE PHỔ BIẾN (tt)

3 Extreme Programming (XP)

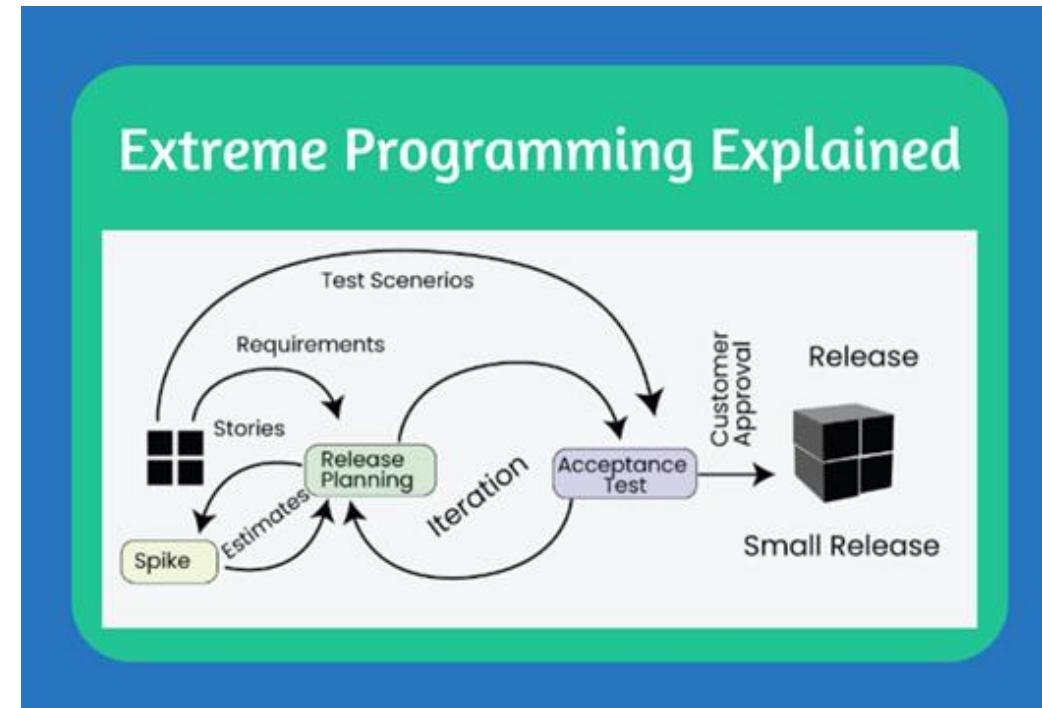
- Tập trung vào **chất lượng code** và **liên tục cải tiến phần mềm**.

Các thực hành quan trọng:

Pair Programming – Lập trình theo cặp để giảm lỗi

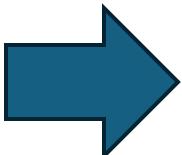
Test-Driven Development (TDD) – Viết test trước khi viết code.

Continuous Integration (CI) – Tích hợp và kiểm thử liên tục.



Khi nào nên dùng XP?

- Khi Dự Án Có Yêu Cầu Thay Đổi Liên Tục.
- Khi Cần Tốc Độ Phát Triển Cao & Ra Mắt Nhanh
- Khi Dự Án Cần Chất Lượng Code Cao & Ít Lỗi
- Khi Nhóm Làm Việc Nhỏ & Tương Tác Thường Xuyên.
- Khi Cần Cải Tiến Liên Tục Trong Quá Trình Phát Triển
- XP khuyến khích Continuous Integration (CI) – Liên tục cập nhật và kiểm tra code.



Tiêu chí	Waterfall	Agile
Phương pháp tiếp cận	Tuần tự, mỗi giai đoạn phải hoàn thành trước khi chuyển sang giai đoạn tiếp theo.	Linh hoạt, phát triển theo từng phần nhỏ với các vòng lặp lặp lại.
Thay đổi yêu cầu	Khó khăn trong việc thay đổi sau khi dự án đã bắt đầu.	Dễ dàng thích ứng với thay đổi, ngay cả khi dự án đang tiến hành.
Sự tham gia của khách hàng	Thường chỉ tham gia ở giai đoạn đầu và cuối.	Tham gia liên tục trong suốt quá trình phát triển.
Kiểm thử	Thực hiện sau khi hoàn thành phát triển.	Thực hiện song song với quá trình phát triển trong mỗi vòng lặp.
Quy mô dự án phù hợp	Thích hợp cho các dự án nhỏ với yêu cầu rõ ràng.	Phù hợp cho các dự án có yêu cầu thay đổi hoặc không chắc chắn.

Định nghĩa:

- Mô hình **bản mẫu nhanh** (**Rapid Prototyping Model**) là một phương pháp phát triển phần mềm tập trung vào **tạo ra các nguyên mẫu** (**prototype**) nhanh chóng để kiểm tra, đánh giá và hoàn thiện yêu cầu trước khi bắt đầu phát triển hệ thống chính thức, điều này cũng có thể gọi là **thử nghiệm beta**.

Ưu điểm của mô hình Rapid Prototyping:

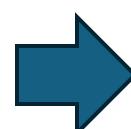
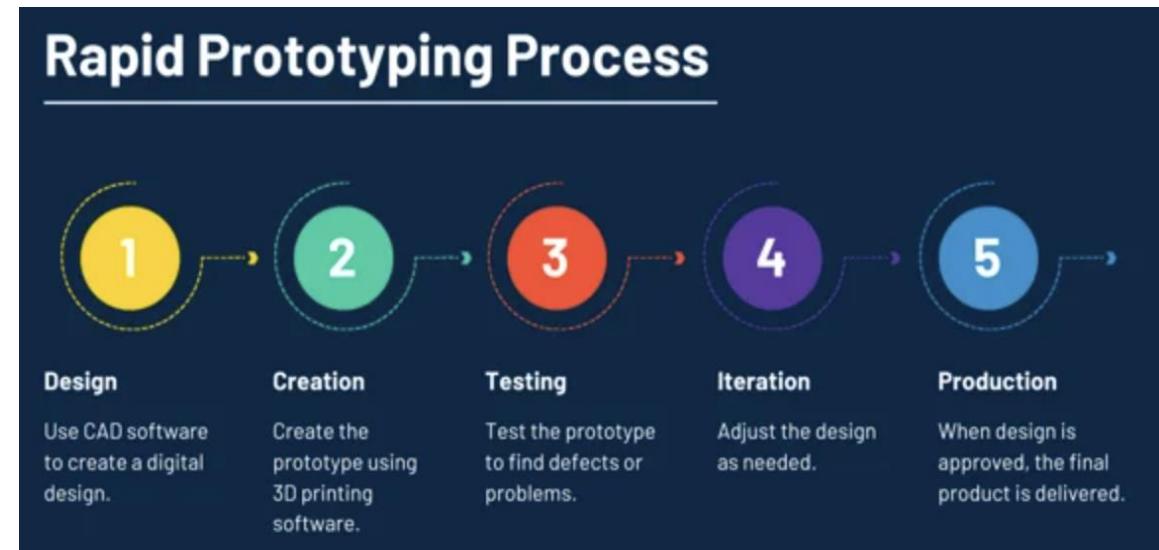
- Giúp khách hàng dễ hình dung về sản phẩm** ngay từ giai đoạn đầu (mẫu thử nghiệm beta).
- Dễ dàng thay đổi yêu cầu** dựa trên phản hồi thực tế.
- Tiết kiệm thời gian và chi phí** so với mô hình Waterfall nếu yêu cầu chưa rõ ràng.

Nhược điểm của mô hình Rapid Prototyping:

- Dễ bị lạm dụng**, dẫn đến kéo dài thời gian phát triển.
- Chi phí có thể tăng cao** nếu thay đổi liên tục.
- Không phù hợp với hệ thống phức tạp** yêu cầu cấu trúc vững chắc ngay từ đầu.

Khi nào nên sử dụng mô hình Rapid Prototyping?

- Khi khách hàng chưa xác định rõ yêu cầu và cần một mô hình trực quan để hiểu rõ hơn.
- Khi cần nhanh chóng kiểm tra khả thi của một ý tưởng.
- Khi phát triển **phần mềm có giao diện người dùng quan trọng** (**UI/UX**).



Kết luận:

- Mô hình Rapid Prototyping là một lựa chọn lý tưởng cho **dự án có yêu cầu linh hoạt và chưa rõ ràng**. Tuy nhiên, cần kiểm soát chặt chẽ số lần chỉnh sửa để tránh lãng phí thời gian và chi phí phát triển.

Định nghĩa:

- Mô hình Ôn Định và Đồng Bộ (Synchronize and Stabilize Model) là một phương pháp phát triển phần mềm được Microsoft áp dụng rộng rãi, đặc biệt trong việc phát triển các phần mềm lớn và phức tạp như hệ điều hành Windows và các ứng dụng doanh nghiệp.

Đặc điểm chính của mô hình Ôn Định và Đồng Bộ

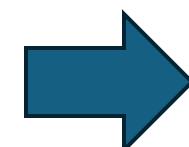
- Phát triển song song (Parallel Development)
- Đồng bộ thường xuyên (Frequent Synchronization)
- Ôn định sản phẩm (Stabilization Phase).
- Dựa trên phản hồi của người dùng

Quy trình phát triển trong mô hình này

- Giai đoạn Lập Kế Hoạch (Planning Phase)
- Giai đoạn Phát Triển và Đồng Bộ (Development & Synchronization Phase)
- Giai đoạn Ôn Định (Stabilization Phase)
- Giai đoạn Chuyển Giao (Final Release Phase)

Ưu điểm của mô hình Ôn Định và Đồng Bộ

- Cho phép phát triển song song, giúp đẩy nhanh tiến độ.
- Linh hoạt với yêu cầu thay đổi, đảm bảo sản phẩm hoàn thiện hơn.



Nhược điểm của mô hình

- Quá trình ôn định có thể kéo dài, làm chậm tiến độ nếu không được kiểm soát tốt.
- Đòi hỏi tài nguyên lớn, không phù hợp với các dự án nhỏ.

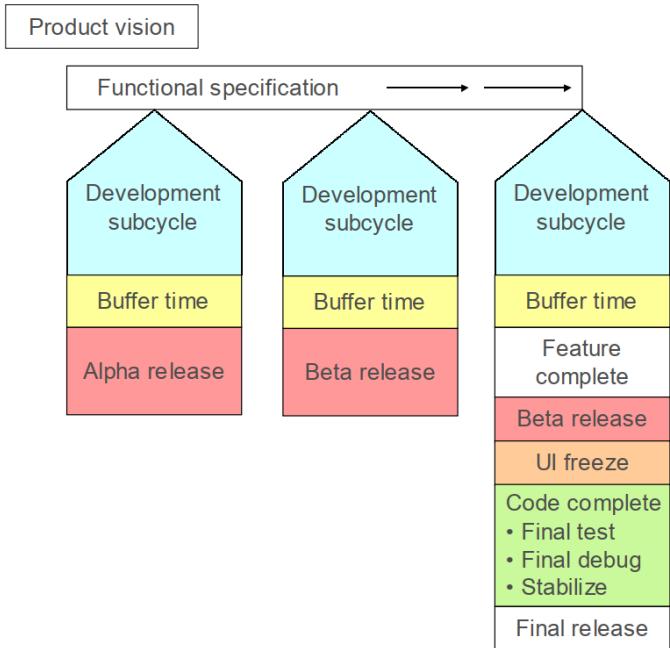


Figure 3. The Synchronize-and-Stabilize process

- Mô hình Ôn Định và Đồng Bộ là một phương pháp mạnh mẽ cho các dự án phần mềm **lớn và phức tạp**. Việc kết hợp phát triển song song, đồng bộ liên tục và kiểm thử kỹ lưỡng giúp đảm bảo sản phẩm cuối cùng **chất lượng cao và ít lỗi nhất**.

Định nghĩa:

- Mô hình Mã Nguồn Mở (Open-Source Model) là phương pháp phát triển phần mềm trong đó mã nguồn của phần mềm được công khai và cho phép mọi người xem, sửa đổi, và phân phối lại.
- Các dự án mã nguồn mở thường được phát triển bởi cộng đồng lập trình viên trên toàn thế giới, thay vì một công ty hoặc tổ chức duy nhất.

Đặc điểm chính của Mô hình Mã Nguồn Mở

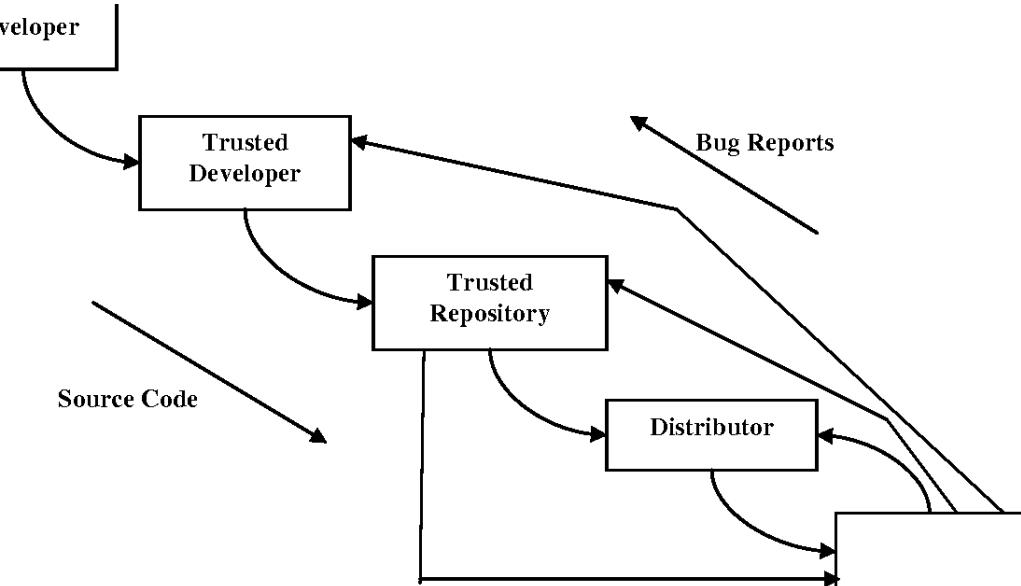
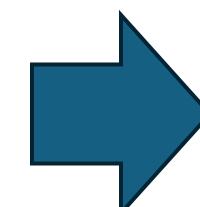
- Mã nguồn công khai .
- Cộng đồng phát triển
- Cải tiến liên .
- Phân phối miễn phí.

Ưu điểm của Mô hình Mã Nguồn Mở

- Chi phí thấp .
- Tính linh hoạt cao .
- Bảo mật tốt hơn
- Không phụ thuộc vào nhà cung cấp .
- Cộng đồng hỗ trợ mạnh mẽ.

Nhược điểm của Mô hình Mã Nguồn Mở

- Hỗ trợ kỹ thuật hạn chế .
- Không phải lúc nào cũng ổn định.
- Có thể phức tạp .
- Không phải tất cả đều miễn phí.



Kết luận

- Mô hình Mã Nguồn Mở mang lại **nhiều lợi ích về chi phí, tính linh hoạt và bảo mật, nhưng cũng có một số thách thức như thiếu hỗ trợ chính thức và có thể khó triển khai.** Việc lựa chọn phần mềm mã nguồn mở hay không phụ thuộc vào nhu cầu và khả năng kỹ thuật của doanh nghiệp hoặc cá nhân

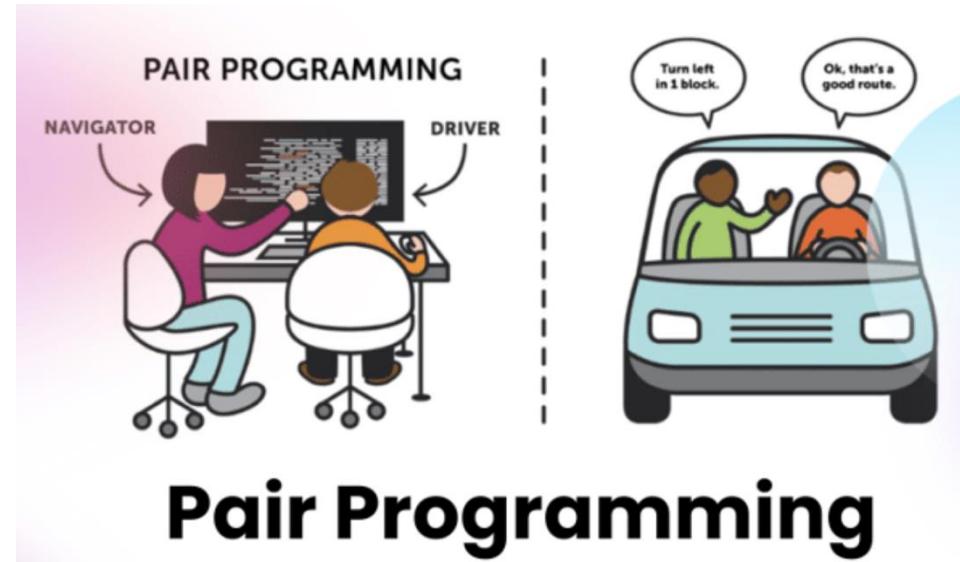
□ Định nghĩa:

- Mô hình **Pair Programming** (Lập trình cặp) là một phương pháp phát triển phần mềm trong đó **hai lập trình viên cùng làm việc trên một máy tính** để viết mã, kiểm tra lỗi và tối ưu hóa mã trong thời gian thực.
- Phương pháp này thuộc về **Extreme Programming (XP)**, một phần của **Agile**, nhằm cải thiện chất lượng phần mềm và tăng cường sự hợp tác trong nhóm.

□ Nguyên tắc chính của Pair Programming

👤 Hai vai trò chính:

- Driver (Người lập trình chính):** Trực tiếp viết mã, tập trung vào việc lập trình chính xác theo yêu cầu.
- Observer/Navigator (Người quan sát và hỗ trợ):** Giám sát mã, kiểm tra lỗi, đề xuất cải tiến và hỗ trợ tư duy chiến lược.
- Liên tục hoán đổi vai trò** để đảm bảo cả hai lập trình viên cùng hiểu sâu về mã nguồn.
- Tập trung vào chất lượng** – Mã được kiểm tra ngay lập tức, giảm lỗi và cải thiện thiết kế.



Pair Programming

Ưu điểm của Mô hình Pair Programming

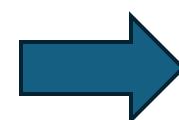
- Chất lượng mã cao hơn** – Mã được kiểm tra và tối ưu hóa ngay lập tức.
- Giảm lỗi phần mềm** – Hai người cùng kiểm tra giúp phát hiện lỗi nhanh hơn.
- Nâng cao kỹ năng lập trình** – Người mới học hỏi từ người có kinh nghiệm.
- Tăng cường hợp tác trong nhóm** – Cải thiện giao tiếp và kỹ năng làm việc nhóm.
- Tư duy phản biện tốt hơn** – Các quyết định thiết kế được thảo luận kỹ lưỡng.

Nhược điểm của Mô hình Pair Programming

- Tốn thời gian hơn** – Hai người làm việc trên một tác vụ thay vì phân chia nhiệm vụ.
- Chi phí nhân lực cao** – Do sử dụng hai lập trình viên cho cùng một công việc.
- Cần kỹ năng giao tiếp tốt** – Nếu không có sự hợp tác, dễ xảy ra xung đột và mất hiệu quả.

Khi nào nên sử dụng Pair Programming?

- Khi cần **đảm bảo chất lượng mã cao nhất**.
- Khi làm việc với **các tính năng quan trọng và phức tạp**.
- Khi có **lập trình viên mới**, giúp học nhanh hơn.
- Khi muốn **cải thiện hợp tác nhóm** và chia sẻ kiến thức.



Note

- Mô hình **Pair Programming** là một phương pháp hiệu quả để nâng cao chất lượng phần mềm, giảm lỗi và cải thiện kỹ năng lập trình. Tuy nhiên, nó cần **cân nhắc chi phí và thời gian** khi áp dụng vào dự án thực tế

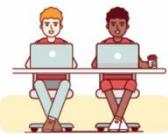
Pair programming styles



Unstructured
In unstructured pair programming, the developers can trade off who takes the lead, and should discuss decisions about the code.



Driver/Navigator
In the driver/navigator approach to pair programming, one developer sets the architectural or strategic direction, and the other implements these decisions as code.



Ping-pong
Ping-pong pair programming shifts rapidly back-and-forth between the two developers, like a game of ping pong, where the software is the ball.

❑ Định nghĩa:

Mô hình Xoắn Ốc (Spiral Model) là một phương pháp phát triển phần mềm kết hợp giữa mô hình thác nước (Waterfall) và mô hình phát triển lặp (Iterative Development). Nó tập trung vào quản lý rủi ro bằng cách chia quy trình phát triển thành các chu kỳ lặp, giúp đánh giá và giảm thiểu rủi ro trong từng giai đoạn.

- Các giai đoạn trong mô hình Xoắn Ốc
- Mô hình này bao gồm **bốn pha chính**, được lặp đi lặp lại trong từng vòng xoắn (iteration):

1 Xác định yêu cầu và lập kế hoạch (Planning)

- Thu thập yêu cầu, phân tích và đặt mục tiêu phát triển.
- Xác định rủi ro tiềm ẩn và phương án xử lý.

2 Phân tích rủi ro (Risk Analysis)

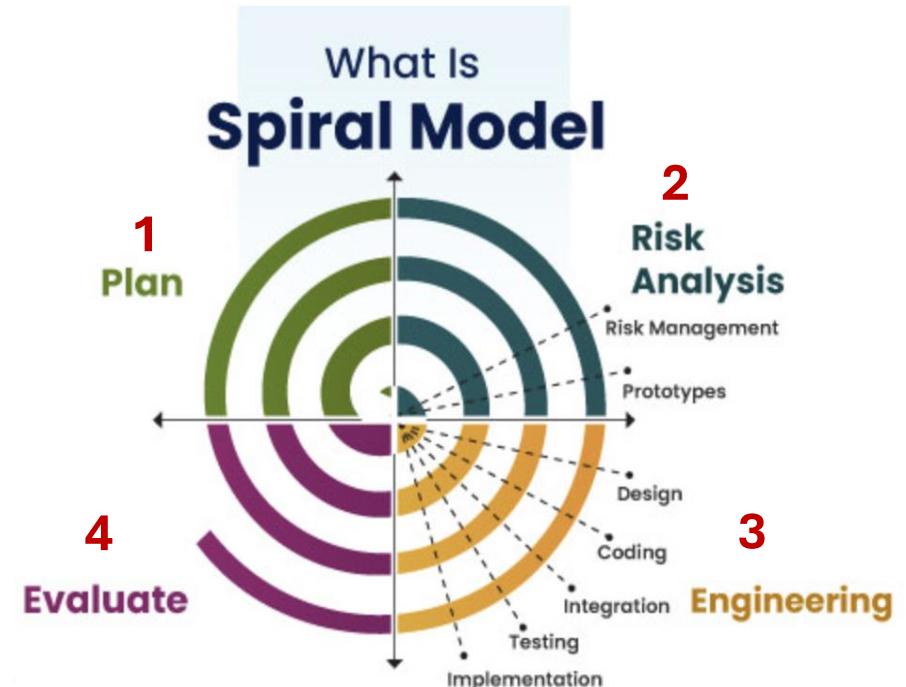
- Đánh giá rủi ro liên quan đến yêu cầu, công nghệ và thiết kế.
- Lập kế hoạch đối phó với các rủi ro có thể xảy ra.

3 Thiết kế, phát triển và kiểm thử (Engineering)

- Xây dựng nguyên mẫu (prototype).
- Tiến hành lập trình, kiểm thử và tối ưu hóa sản phẩm.

4 Đánh giá và tiếp tục phát triển (Evaluation & Refinement)

- Đánh giá kết quả của vòng lặp.
- Nếu sản phẩm chưa hoàn thiện, tiếp tục một vòng xoắn mới để cải thiện.



Ưu điểm của mô hình Xoắn Ốc

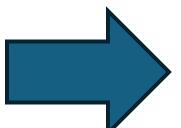
- Quản lý rủi ro tốt.
- Linh hoạt với yêu cầu thay đổi.
- Phù hợp với dự án lớn.
- Cải thiện chất lượng phần mềm.

 Nhược điểm của mô hình Xoắn Ốc

- Chi phí cao.
- Đòi hỏi kỹ năng quản lý cao.
- Không phù hợp với dự án nhỏ.

 Khi nào nên sử dụng mô hình Xoắn Ốc?

- Khi phát triển dự án lớn và phức tạp, có nhiều rủi ro.
- Khi phần mềm yêu cầu tính linh hoạt cao và dễ thay đổi.
- Khi cần nghiên cứu và phát triển công nghệ mới trước khi triển khai toàn bộ dự án.

 **Kết luận**

- Mô hình Xoắn Ốc là một cách tiếp cận **hiện đại và hiệu quả** cho việc phát triển phần mềm phức tạp. Nó giúp **kiểm soát rủi ro, cải thiện chất lượng sản phẩm và linh hoạt trong quá trình phát triển**. Tuy nhiên, do chi phí cao và yêu cầu quản lý tốt, mô hình này thường **được áp dụng cho các dự án phần mềm lớn, quan trọng** như hệ thống ngân hàng, hàng không, và các ứng dụng doanh nghiệp.



3.3. Case Study

Hoạt động nhóm:

- Chia nhóm (3-5 sinh viên mỗi nhóm).
- Mỗi nhóm sẽ chọn một mô hình vòng đồi phát triển phần mềm và trình bày về đặc điểm, ưu điểm, nhược điểm của mô hình đó.
- Thảo luận về tình huống thực tế áp dụng mô hình và đưa ra bài học kinh nghiệm.

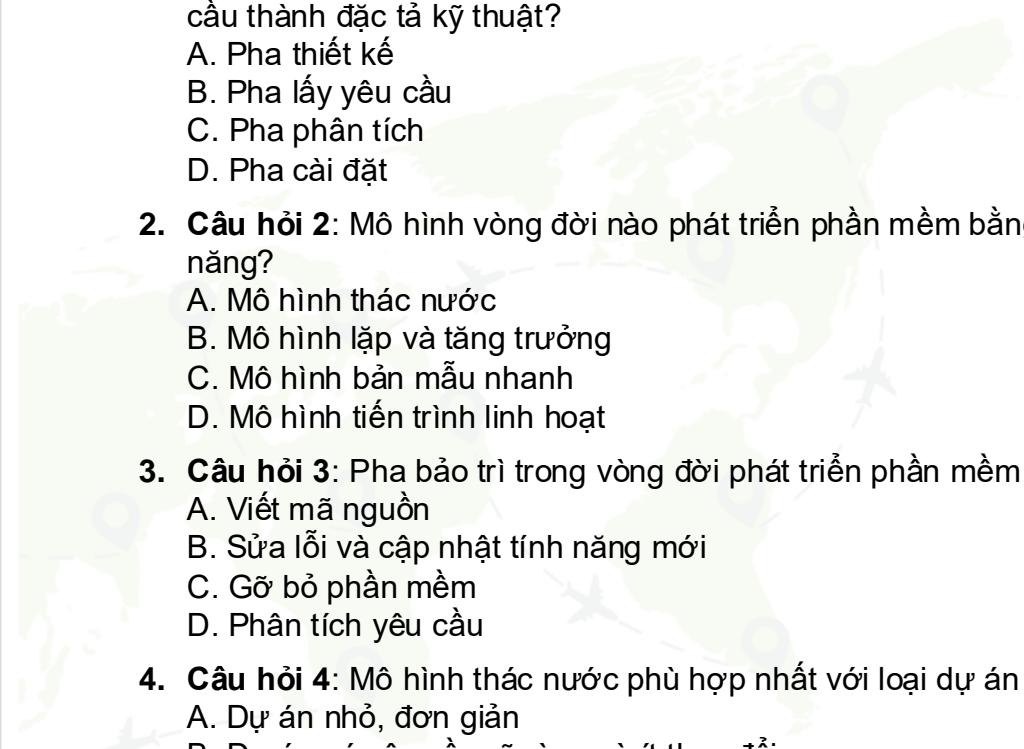


3.1 CÂU HỎI TRẮC NGHIỆM

3.2 CÂU HỎI TRẢ LỜI NGẮN

3.3. CÂU HỎI THẢO LUẬN NHÓM

3.4 CÂU HỎI TÌNH HUỐNG

- 
- Câu hỏi 1:** Pha nào trong mô hình lý thuyết vòng đời phát triển phần mềm chịu trách nhiệm chuyển đổi yêu cầu thành đặc tả kỹ thuật?
 - A. Pha thiết kế
 - B. Pha lấy yêu cầu
 - C. Pha phân tích
 - D. Pha cài đặt
 - Câu hỏi 2:** Mô hình vòng đời nào phát triển phần mềm bằng cách tạo các phiên bản nhỏ và tăng dần tính năng?
 - A. Mô hình thác nước
 - B. Mô hình lặp và tăng trưởng
 - C. Mô hình bản mẫu nhanh
 - D. Mô hình tiến trình linh hoạt
 - Câu hỏi 3:** Pha bảo trì trong vòng đời phát triển phần mềm bao gồm hoạt động nào?
 - A. Viết mã nguồn
 - B. Sửa lỗi và cập nhật tính năng mới
 - C. Gỡ bỏ phần mềm
 - D. Phân tích yêu cầu
 - Câu hỏi 4:** Mô hình thác nước phù hợp nhất với loại dự án nào?
 - A. Dự án nhỏ, đơn giản
 - B. Dự án có yêu cầu rõ ràng và ít thay đổi
 - C. Dự án yêu cầu linh hoạt cao
 - D. Dự án mã nguồn mở
 - Câu hỏi 5:** Trong mô hình xoắn ốc, mỗi vòng xoắn tương ứng với:
 - A. Một pha kiểm thử
 - B. Một chu kỳ lặp của toàn bộ quy trình phát triển
 - C. Một phiên bản phần mềm nhỏ
 - D. Một lần phân tích rủi ro

6. **Câu hỏi 6:** Điểm yếu lớn nhất của mô hình xây và sửa là gì?
- A. Khó phát triển nhanh
 - B. Tốn nhiều chi phí bảo trì
 - C. Khó kiểm soát chất lượng
 - D. Không phù hợp với dự án nhỏ
7. **Câu hỏi 7:** Mô hình nào tập trung vào việc tạo các nguyên mẫu nhanh để thu thập phản hồi từ khách hàng?
- A. Mô hình lặp và tăng trưởng
 - B. Mô hình bản mẫu nhanh
 - C. Mô hình xoắn ốc
 - D. Mô hình tiến trình linh hoạt
8. **Câu hỏi 8:** Pha nào kết thúc vòng đời phát triển phần mềm?
- A. Pha bảo trì
 - B. Pha cài đặt
 - C. Pha giải thể
 - D. Pha thiết kế
9. **Câu hỏi 9:** Điểm khác biệt chính giữa mô hình lặp và tăng trưởng với mô hình thác nước là gì?
- A. Mô hình thác nước lặp lại nhiều lần
 - B. Mô hình lặp và tăng trưởng phát triển theo từng đợt nhỏ
 - C. Mô hình lặp và tăng trưởng không có giai đoạn kiểm thử
 - D. Mô hình thác nước không có giai đoạn bảo trì
10. **Câu hỏi 10:** Mô hình nào có khả năng thích nghi tốt nhất với sự thay đổi của yêu cầu khách hàng?
- A. Mô hình thác nước
 - B. Mô hình xoắn ốc
 - C. Mô hình xây và sửa
 - D. Mô hình tiến trình linh hoạt

1. Pha lầy yêu cầu là gì và có vai trò gì trong vòng đời phát triển phần mềm?
2. Mô hình thác nước hoạt động như thế nào?
3. Mô hình lặp và tăng trưởng khác gì so với mô hình thác nước?
4. Mục tiêu của pha bảo trì là gì?
5. Mô hình xây và sửa có nhược điểm gì?
6. Mô hình bản mẫu nhanh là gì?
7. Pha giải thể là gì?
8. Mô hình xoắn ốc là gì?
9. Tại sao mô hình tiến trình linh hoạt được đánh giá cao?
10. Điểm khác biệt chính giữa mô hình mã nguồn mở và các mô hình khác là gì?

1. So sánh ưu và nhược điểm của mô hình thác nước và mô hình xoắn ốc.
2. Thảo luận về tình huống thực tế có thể áp dụng mô hình lặp và tăng trưởng.
3. Tại sao mô hình xây và sửa không phù hợp với các dự án lớn?
4. So sánh giữa mô hình bản mẫu nhanh và mô hình tiến trình linh hoạt.
5. Phân tích vai trò của quản lý rủi ro trong mô hình xoắn ốc.
6. Khi nào nên sử dụng mô hình thác nước thay vì mô hình tiến trình linh hoạt?
7. Thảo luận về những khó khăn khi áp dụng mô hình mã nguồn mở.
8. Phân tích cách mô hình tiến trình linh hoạt giúp cải thiện chất lượng phần mềm.
9. Thảo luận về vai trò của pha bảo trì trong vòng đời phát triển phần mềm.
10. Đề xuất mô hình vòng đời phù hợp cho dự án phát triển phần mềm ngân hàng và giải thích lý do.

1. Một công ty phát triển phần mềm theo mô hình thác nước gấp vấn đề khi khách hàng yêu cầu thay đổi sau khi hoàn thành pha thiết kế. Đội phát triển nên xử lý như thế nào?
2. Dự án phát triển phần mềm theo mô hình lặp và tăng trưởng liên tục bị trễ tiến độ do thiếu nhân lực. Là quản lý dự án, bạn sẽ làm gì?
3. Trong quá trình phát triển phần mềm theo mô hình tiến trình linh hoạt, khách hàng không đưa ra phản hồi kịp thời. Đội phát triển nên xử lý ra sao?
4. Khách hàng yêu cầu bổ sung một số tính năng mới khi phần mềm đã bước vào pha cài đặt. Nên áp dụng mô hình nào để xử lý tốt nhất yêu cầu này?
5. Một công ty nhỏ muốn áp dụng mô hình bản mẫu nhanh nhưng gặp khó khăn do thiếu nguồn lực. Hãy đề xuất giải pháp.
6. Trong dự án phần mềm thương mại điện tử, khách hàng liên tục yêu cầu thay đổi giao diện. Mô hình nào sẽ phù hợp nhất?
7. Dự án phát triển phần mềm lớn với nhiều nhóm phát triển ở các quốc gia khác nhau nên áp dụng mô hình nào?
8. Phân tích rủi ro là hoạt động chính trong mô hình xoắn ốc. Đề xuất cách giảm thiểu rủi ro khi áp dụng mô hình này.
9. Một dự án phát triển phần mềm ngân hàng cần yêu cầu bảo mật cao. Nên áp dụng mô hình nào để đảm bảo yêu cầu này?
10. Khi nào nên kết thúc vòng đời phần mềm và thực hiện pha giải thể?

OUTLINE

4.1. KIỂM THỬ

4.2. LẬP KẾ HOẠCH

4.3. LÀM TÀI LIỆU

4.4. CASE STUDY

4.5 CÂU HỎI VẬN DỤNG LÝ THUYẾT

ĐỌC TÀI LIỆU CHƯƠNG 4 &9, CUỐN TÀI LIỆU 1

□ 4.1.1. Hoạt động kiểm thử trong tiến trình phát triển phần mềm

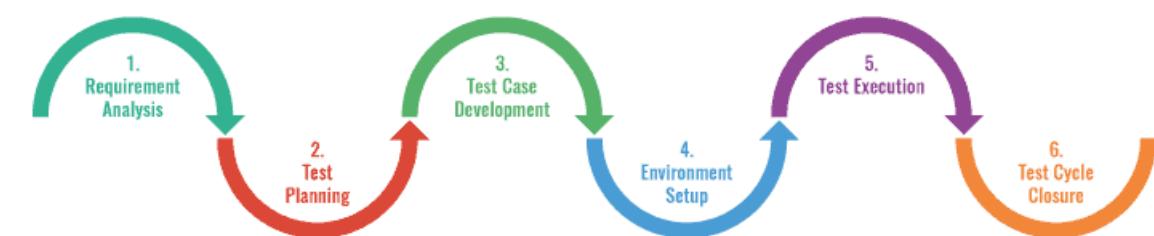
- **Mục tiêu:**

Đảm bảo phần mềm hoạt động đúng như yêu cầu của khách hàng và không chứa lỗi nghiêm trọng trước khi bàn giao.

- **Các loại kiểm thử chính:**

- Kiểm thử đơn vị (Unit Testing).
- Kiểm thử tích hợp (Integration Testing).
- Kiểm thử hệ thống (System Testing.)
- Kiểm thử chấp nhận (Acceptance Testing.)

Software Testing Life Cycle (STLC)



4.1.2. Nhóm SQA (Software Quality Assurance)

Khái niệm:

Nhóm SQA chịu trách nhiệm đảm bảo rằng các quy trình và sản phẩm phần mềm tuân thủ các tiêu chuẩn chất lượng đã đề ra.

Vai trò của nhóm SQA:

Xây dựng và duy trì quy trình kiểm thử.

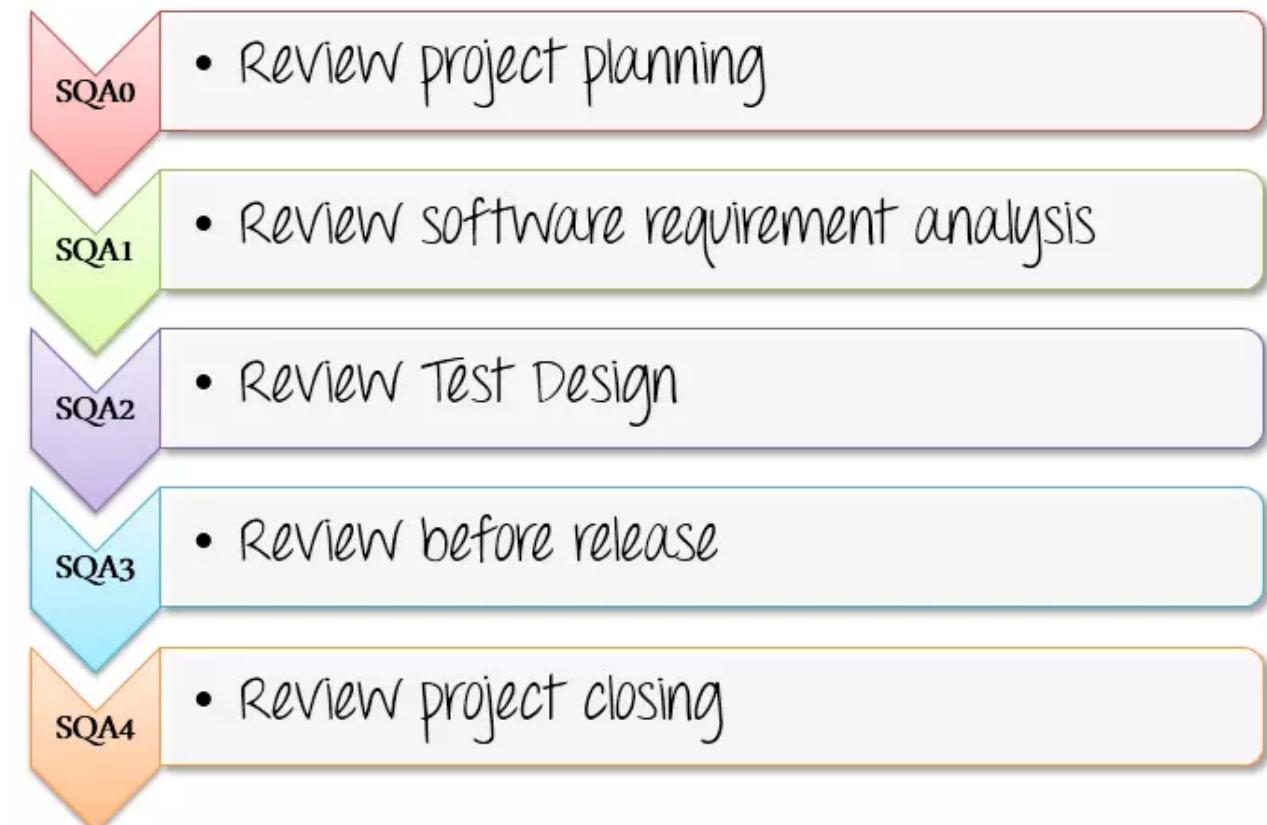
Đánh giá kết quả kiểm thử.

Đảm bảo rằng sản phẩm cuối cùng đáp ứng các tiêu chuẩn chất lượng và yêu cầu của khách hàng.

Hoạt động chính của nhóm SQA:

Thực hiện các cuộc đánh giá chất lượng phần mềm định kỳ.

Giám sát việc thực hiện các quy trình phát triển phần mềm.



❑ Định nghĩa:

Kiểm thử các sản phẩm phi thực thi là kiểm thử các tài liệu liên quan đến phần mềm như tài liệu yêu cầu, tài liệu thiết kế, và tài liệu hướng dẫn sử dụng.

❑ Phương pháp kiểm thử:

- **Walkthrough:** Các thành viên nhóm phát triển cùng nhau xem xét tài liệu để phát hiện lỗi.
- **Inspection:** Một nhóm chuyên gia độc lập thực hiện đánh giá tài liệu.
- **Review:** Người quản lý tổ chức buổi họp để thảo luận và rà soát các tài liệu đã hoàn thành.



Định nghĩa:

Kiểm thử các sản phẩm thực thi là kiểm thử phần mềm đã được triển khai trên hệ thống thực tế.

Loại kiểm thử:

- Kiểm thử chức năng (functional)
- Kiểm thử hiệu suất (performance)
- Kiểm thử bảo mật (security)

Types of Testing

Non-Execution Based Testing

- Applicable to earlier phases as well as Coding phase
- Mostly comprised of Walkthroughs and Inspections of Documents, Design and Code

Execution Based Testing

- Corresponds to the phases when code is available
- Applicable to Implementation, Integration and Maintenance phases

❑ Mục tiêu:

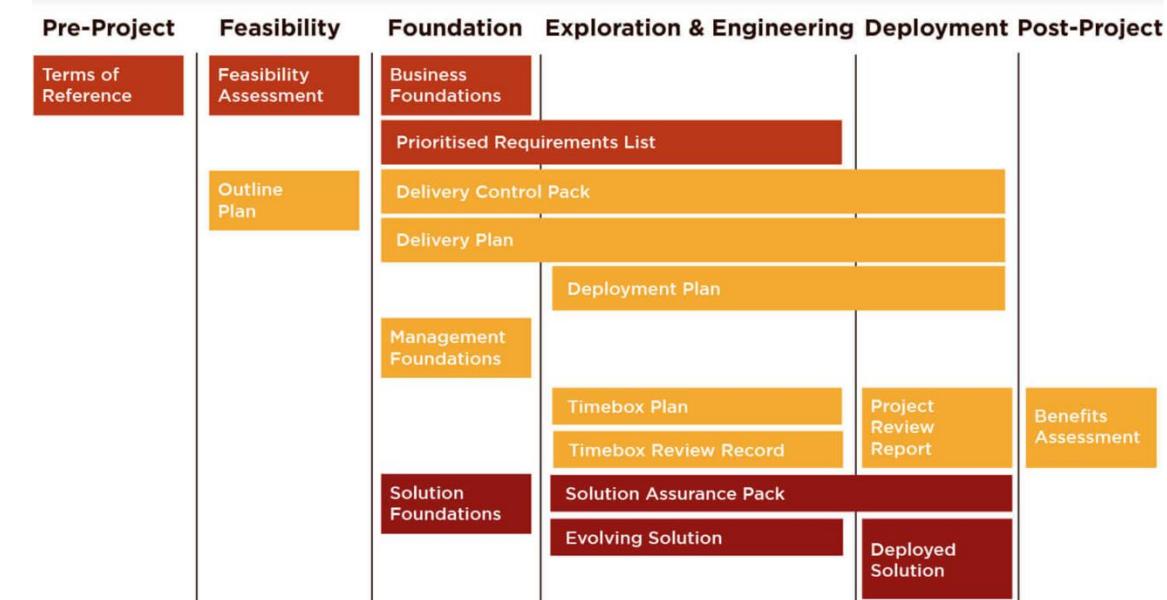
Xác định các công việc cần thực hiện, phân bổ tài nguyên và ước tính thời gian hoàn thành.

❑ Hoạt động chính:

Phân tích yêu cầu và phạm vi dự án.

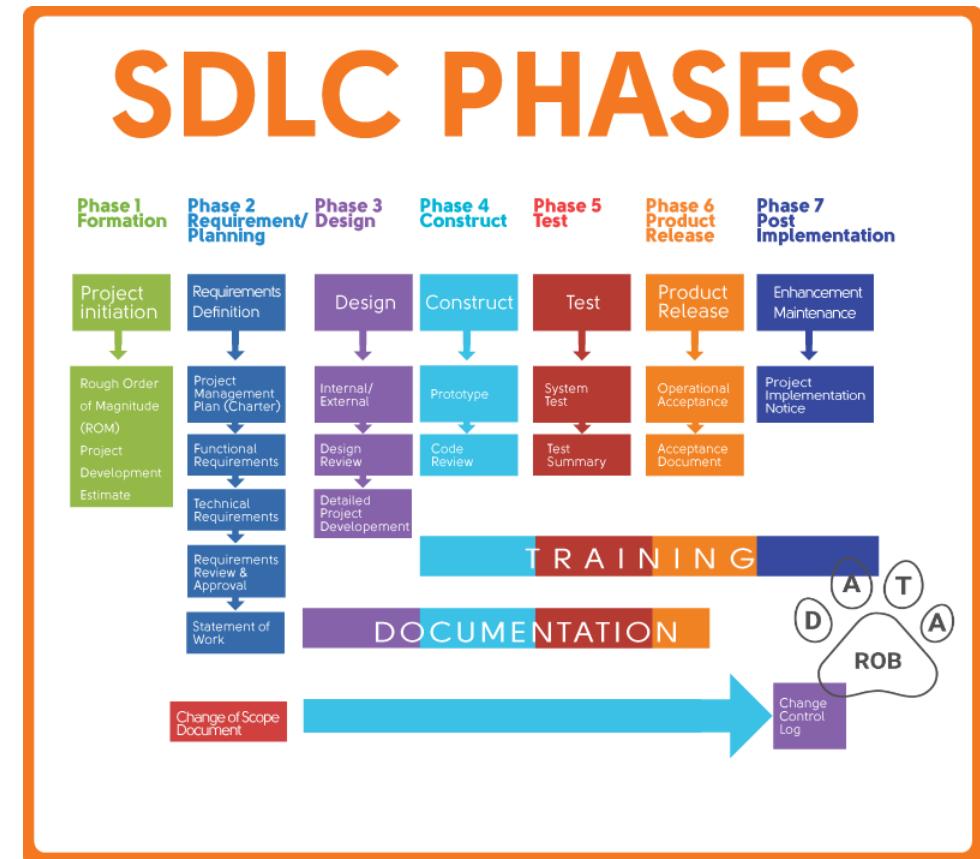
Xác định các công việc và sắp xếp thứ tự ưu tiên.

Lập kế hoạch quản lý rủi ro.



Lập kế hoạch cho từng pha:

- Pha lấy yêu cầu:** Lập kế hoạch về các hoạt động thu thập và phân tích yêu cầu.
- Pha thiết kế:** Xác định thời gian và tài nguyên cần thiết cho việc thiết kế kiến trúc hệ thống.
- Pha cài đặt:** Lập kế hoạch viết mã nguồn và kiểm thử đơn vị.
- Pha kiểm thử:** Lập kế hoạch kiểm thử hệ thống, bao gồm cả kiểm thử chấp nhận.
- Pha bảo trì:** Dự kiến các hoạt động bảo trì sau khi phần mềm được bàn giao.

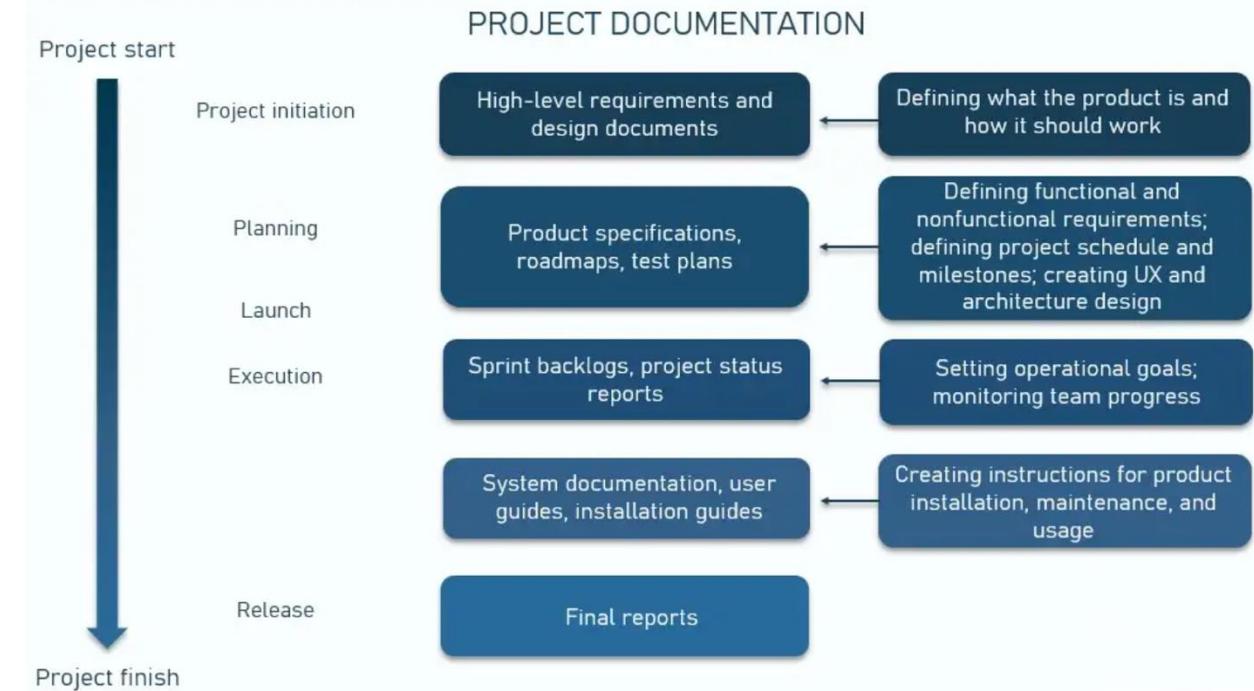


Mục tiêu:

Đảm bảo tất cả các giai đoạn phát triển phần mềm đều có tài liệu đầy đủ và rõ ràng để hỗ trợ phát triển và bảo trì.

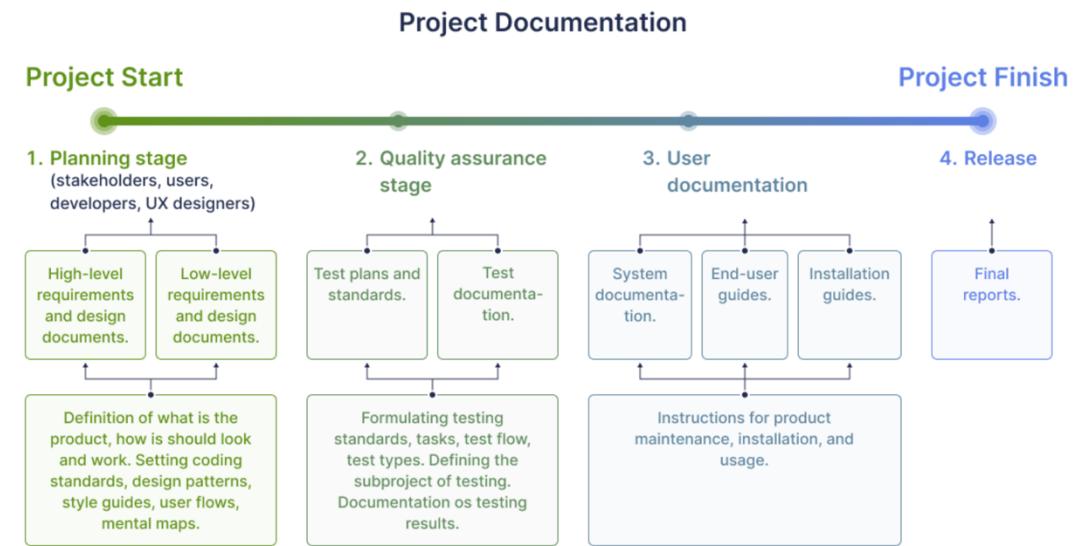
Các loại tài liệu chính:

- Tài liệu yêu cầu.
- Tài liệu thiết kế.
- Tài liệu hướng dẫn sử dụng.
- Tài liệu kiểm thử.



- ❑ **Pha lấy yêu cầu:** Lập tài liệu mô tả chi tiết yêu cầu của khách hàng.
- ❑ **Pha thiết kế:** Xây dựng tài liệu kiến trúc và tài liệu thiết kế chi tiết.
- ❑ **Pha cài đặt:** Viết tài liệu hướng dẫn sử dụng mã nguồn và tài liệu cấu hình hệ thống.
- ❑ **Pha kiểm thử:** Lập tài liệu kế hoạch kiểm thử và báo cáo kết quả kiểm thử.

...



Mục tiêu:

Đảm bảo rằng các phiên bản tài liệu được quản lý chặt chẽ để tránh nhầm lẫn và đảm bảo tính nhất quán.

Hoạt động quản lý phiên bản:

- Ghi nhận thay đổi của tài liệu.
- Sử dụng công cụ quản lý phiên bản như Git hoặc SVN.
- Lưu trữ và bảo mật các phiên bản tài liệu.

Status	Version	Author	Date	Changes
Draft	v0.1	Name Surname Email	20200115	First draft
Draft	V2.0	Name Surname Email	20200120	Inserted images and tables
Final	V1.0	Name Surname Email	20200130	Inserted references and typos correction
Approved	V1.0	Name Surname Email	20200205	Approved by XX

Mục tiêu:

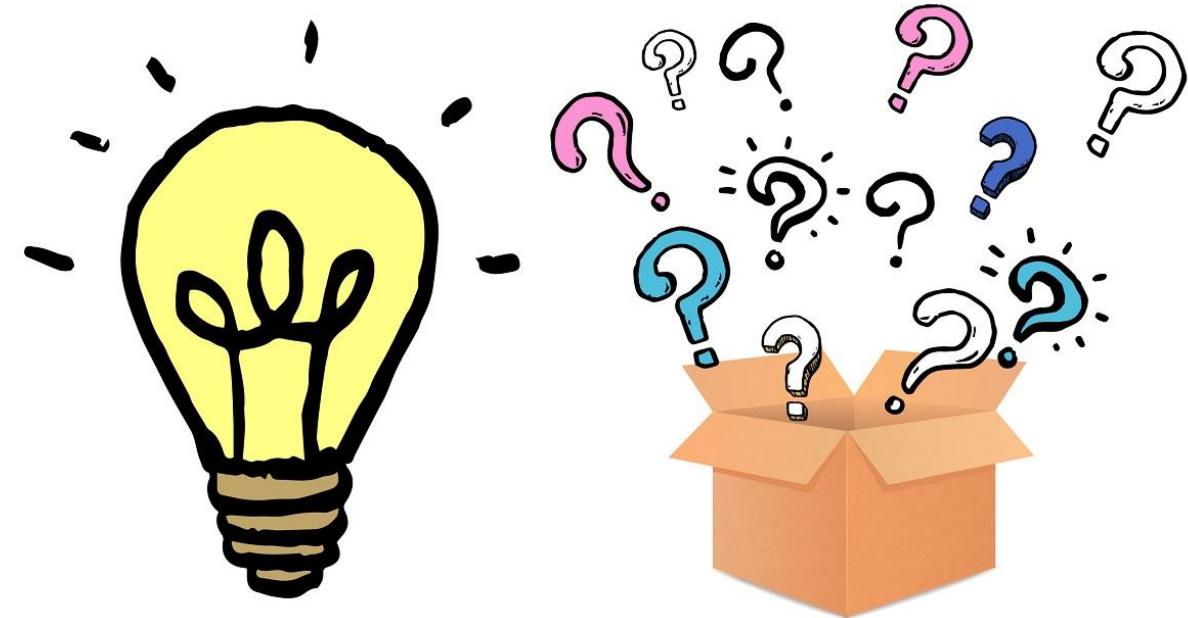
Giúp sinh viên hiểu rõ vai trò của nhóm SQA và việc làm tài liệu kiểm thử thông qua tình huống thực tế.

Hoạt động nhóm:

- Chia lớp thành các nhóm 3-5 sinh viên.
- Mỗi nhóm sẽ đọc tài liệu liên quan đến nhóm SQA và làm tài liệu kiểm thử.
- Thảo luận và trình bày về vai trò của nhóm SQA và các loại tài liệu kiểm thử cần thiết trong quá trình phát triển phần mềm.
- Đánh giá tài liệu kiểm thử do các nhóm khác chuẩn bị và đưa ra nhận xét.



1. CÂU HỎI TRẮC NGHIỆM
2. CÂU HỎI TRẢ LỜI NGẮN
3. CÂU HỎI THẢO LUẬN NHÓM
4. CÂU HỎI TÌNH HUỐNG



5. CÂU HỎI TRẮC NGHIỆM

1. **Câu hỏi 1:** Mục tiêu chính của kiểm thử phần mềm là gì?
 - A. Đảm bảo phần mềm không chứa lỗi
 - B. Đảm bảo phần mềm đáp ứng đúng yêu cầu
 - C. Hoàn thành dự án đúng thời gian
 - D. Phát hiện tất cả các lỗi bảo mật
2. **Câu hỏi 2:** Nhóm nào chịu trách nhiệm đảm bảo chất lượng phần mềm trong dự án?
 - A. Nhóm kiểm thử tự động
 - B. Nhóm bảo trì
 - C. Nhóm SQA
 - D. Nhóm thiết kế
3. **Câu hỏi 3:** Kiểm thử các sản phẩm phi thực thi bao gồm hoạt động nào dưới đây?
 - A. Kiểm thử đơn vị
 - B. Walkthrough và review tài liệu
 - C. Kiểm thử hệ thống
 - D. Kiểm thử hiệu suất
4. **Câu hỏi 4:** Lập kế hoạch trong tiến trình phát triển phần mềm nhằm mục đích gì?
 - A. Xác định phạm vi công việc và ước tính thời gian hoàn thành
 - B. Giảm thiểu chi phí phát triển
 - C. Phát hiện lỗi sớm trong quá trình phát triển
 - D. Xác định các công cụ kiểm thử tự động
5. **Câu hỏi 5:** Loại kiểm thử nào sau đây tập trung vào việc phát hiện các lỗi chức năng của phần mềm?
 - A. Kiểm thử hiệu suất
 - B. Kiểm thử chức năng
 - C. Kiểm thử bảo mật
 - D. Kiểm thử tích hợp

5. CÂU HỎI TRẮC NGHIỆM

6. Câu hỏi 6: Quản lý phiên bản tài liệu có mục tiêu gì?

- A. Đảm bảo tài liệu luôn được cập nhật và có thể truy xuất phiên bản cũ khi cần
- B. Giảm thiểu số lượng tài liệu cần làm
- C. Tăng tốc độ phát triển phần mềm
- D. Tự động hóa việc kiểm thử tài liệu

7. Câu hỏi 7: Nhóm SQA có vai trò gì trong kiểm thử phần mềm?

- A. Viết mã nguồn cho phần mềm
- B. Đánh giá và đảm bảo quy trình phát triển phần mềm tuân thủ tiêu chuẩn chất lượng
- C. Triển khai phần mềm lên môi trường thực tế
- D. Giám sát hoạt động của hệ thống sau khi triển khai

8. Câu hỏi 8: Loại kiểm thử nào thường được thực hiện cuối cùng trước khi phần mềm được bàn giao cho khách hàng?

- A. Kiểm thử đơn vị
- B. Kiểm thử hệ thống
- C. Kiểm thử tích hợp
- D. Kiểm thử chấp nhận

9. Câu hỏi 9: Đâu là một trong những công cụ phổ biến dùng để quản lý phiên bản tài liệu?

- A. Selenium
- B. Git
- C. Postman
- D. JIRA

10. Câu hỏi 10: Hoạt động lập tài liệu trong mỗi pha phát triển phần mềm nhằm mục đích gì?

- A. Giảm thời gian phát triển phần mềm
- B. Hỗ trợ quá trình bảo trì và nâng cấp phần mềm sau khi triển khai
- C. Tăng cường bảo mật cho phần mềm
- D. Tự động hóa việc phát triển phần mềm

5. CÂU HỎI NGẮN

1. Nhóm SQA là gì và vai trò của nhóm này trong phát triển phần mềm?
2. Kiểm thử đơn vị là gì?
3. Mục tiêu chính của kiểm thử chấp nhận là gì?
4. Các hoạt động chính trong kiểm thử sản phẩm phi thực thi là gì?
5. Tại sao việc lập tài liệu cho mỗi pha phát triển phần mềm lại quan trọng?
6. Quản lý phiên bản tài liệu là gì?
7. Các loại kiểm thử chính trong kiểm thử sản phẩm thực thi là gì?
8. Kiểm thử tích hợp là gì?
9. Hoạt động lập kế hoạch cho các pha phát triển phần mềm bao gồm những gì?
10. Làm tài liệu kiểm thử bao gồm những gì?

5. CÂU HỎI THẢO LUẬN NHÓM

1. Vai trò của nhóm SQA trong việc đảm bảo chất lượng phần mềm là gì?
2. Thảo luận về sự khác nhau giữa kiểm thử đơn vị và kiểm thử tích hợp.
3. Tại sao việc lập tài liệu kiểm thử lại quan trọng trong mỗi dự án phần mềm?
4. Thảo luận về các thách thức khi lập kế hoạch cho các pha phát triển phần mềm.
5. Quản lý phiên bản tài liệu có ảnh hưởng như thế nào đến quá trình bảo trì phần mềm?
6. So sánh giữa kiểm thử sản phẩm phi thực thi và kiểm thử sản phẩm thực thi.
7. Thảo luận về cách cải thiện quy trình lập kế hoạch để giảm thiểu rủi ro trong dự án phần mềm.
8. Đề xuất các công cụ hỗ trợ việc lập tài liệu kiểm thử và quản lý phiên bản.
9. Tại sao kiểm thử chấp nhận lại là một giai đoạn quan trọng trong phát triển phần mềm?
10. Thảo luận về các phương pháp hiệu quả để quản lý chất lượng phần mềm trong các dự án lớn.

5. CÂU HỎI TÌNH HUỐNG

1. Một dự án phần mềm đã hoàn thành và sắp bàn giao cho khách hàng, nhưng khách hàng yêu cầu kiểm tra lại toàn bộ tài liệu yêu cầu và thiết kế. Đội phát triển nên xử lý thế nào?
2. Trong quá trình kiểm thử hệ thống, nhóm phát triển phát hiện một lỗi nghiêm trọng nhưng thời hạn bàn giao đang đến gần. Bạn sẽ xử lý tình huống này như thế nào?
3. Một nhóm phát triển gặp khó khăn trong việc quản lý phiên bản tài liệu do tài liệu liên tục thay đổi. Hãy đề xuất giải pháp.
4. Dự án phát triển phần mềm gặp vấn đề khi khách hàng yêu cầu thay đổi lớn trong pha cài đặt. Đội phát triển nên xử lý thế nào?
5. Nhóm kiểm thử phát hiện nhiều lỗi chức năng trong phần mềm. Tuy nhiên, nhóm phát triển lại cho rằng đây không phải lỗi mà là tính năng. Là trưởng dự án, bạn sẽ làm gì?
6. Khách hàng yêu cầu bổ sung một tính năng mới khi phần mềm đã hoàn thành pha kiểm thử tích hợp. Đội phát triển nên làm gì?
7. Một công ty phát triển phần mềm nhỏ muốn xây dựng nhóm SQA nhưng gặp khó khăn về ngân sách. Hãy đề xuất giải pháp.
8. Trong quá trình làm tài liệu kiểm thử, nhóm phát triển không thống nhất được về nội dung cần đưa vào tài liệu. Là trưởng nhóm, bạn sẽ giải quyết vấn đề này như thế nào?
9. Dự án phát triển phần mềm cho một ngân hàng yêu cầu bảo mật cao. Đề xuất cách lập kế hoạch kiểm thử cho dự án này.
10. Sau khi triển khai phần mềm, khách hàng phát hiện ra một số lỗi bảo mật nghiêm trọng. Đội phát triển cần xử lý ra sao để khắc phục vấn đề và lấy lại niềm tin từ khách hàng?

SWE | 05. LẤY YÊU CẦU, MÔ TẢ BẰNG NGÔN NGỮ TỰ NHIÊN

OUTLINE

5.1. XÁC ĐỊNH CÁI KHÁCH HÀNG CẦN

5.2. TÌM HIỂU LĨNH VỰC CỦA ỨNG DỤNG

5.4. CASE STUDY

5.5 CÂU HỎI VẬN DỤNG LÝ THUYẾT



ĐỌC TÀI LIỆU CHƯƠNG 11

❑ **Mục tiêu:** Tìm hiểu và ghi nhận chi tiết những yêu cầu của khách hàng.

❑ **Các kỹ thuật phỏng vấn:**

Phỏng vấn không cấu trúc: Khách hàng tự do trình bày các yêu cầu.

Phỏng vấn có cấu trúc: Câu hỏi được chuẩn bị trước và khách hàng trả lời theo các chủ đề rõ ràng.

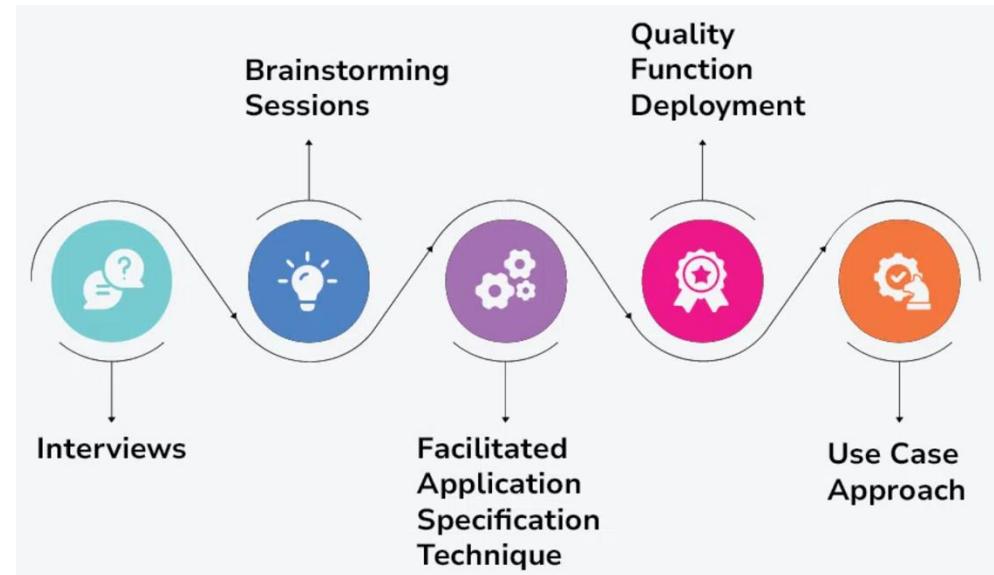
Phỏng vấn bán cấu trúc: Kết hợp cả hai loại trên, vừa đặt câu hỏi theo trình tự, vừa tạo không gian để khách hàng bổ sung thông tin.

❑ **Lưu ý khi phỏng vấn:**

Lắng nghe kỹ và ghi chú đầy đủ.

Đặt câu hỏi rõ ràng, tránh gây hiểu lầm.

Xác nhận lại với khách hàng để đảm bảo thông tin đúng.



Kỹ thuật thu thập yêu cầu

Các bước tổng hợp:

- Phân loại yêu cầu thành yêu cầu chức năng và phi chức năng.
- Loại bỏ các yêu cầu mâu thuẫn hoặc trùng lặp.
- Sắp xếp các yêu cầu theo mức độ ưu tiên.
- Ghi chép lại thành tài liệu yêu cầu ban đầu.

Aspect	Functional Requirements	Non-functional Requirements
Definition	Specifies what the system should do (actions and tasks).	Describes how well the system performs (quality attributes).
Purpose	To ensure the system performs necessary functions for users.	To ensure the system meets quality standards for performance, reliability, and usability.
Focus	Focuses on features and capabilities of the system.	Focuses on system quality and user experience.
Example	User login, data retrieval, payment processing.	Performance speed, system availability, security protocols.
Testing Method	Functional testing (to validate feature behavior).	Performance, security, and usability testing.
Measurement	Measurable through specific functions or actions.	Measured by quality attributes (e.g., response time, uptime).
Impact on Development	Defined early to shape core system functionality.	Often refined throughout development to ensure system stability and scalability.
Goal	To meet user and business needs for system capabilities.	To enhance user experience and ensure system robustness.
Importance	Provides essential functions to meet user requirements.	Ensures the system is efficient, reliable, and user-friendly.

❑ Phương pháp mô tả:

Viết yêu cầu dưới dạng câu ngắn gọn, dễ hiểu.

Sử dụng các thuật ngữ phổ thông, hạn chế dùng thuật ngữ kỹ thuật trừ khi cần thiết.

Đảm bảo mỗi yêu cầu đều rõ ràng, không gây hiểu nhầm.

❑ Ví dụ mô tả yêu cầu:

"Hệ thống phải cho phép người dùng tạo tài khoản với tên đăng nhập và mật khẩu."

Needs (I need to...)	Requirements
build a big house for my large family	<ul style="list-style-type: none">The house shall have 3 bedrooms<ul style="list-style-type: none">2 small bedrooms1 master bedroomThe house shall have a backyardThe house shall be 3 levelsThe floors shall be wood<ul style="list-style-type: none">The wood color shall be dark brown
get a BA job	<ul style="list-style-type: none">I should take a BA courseI should create a resumeI should practice mockup interviews
build my portfolio website	<ul style="list-style-type: none">The website shall allow clients to contact meThe website shall have a blogThe website shall present my projects

Mục tiêu: Giúp đội phát triển hiểu rõ các thuật ngữ chuyên ngành mà khách hàng sử dụng.

Cách thực hiện:

- Thu thập các thuật ngữ qua phỏng vấn và tài liệu.
- Lập danh sách các thuật ngữ kèm định nghĩa.

Step	Description
1. Gather Information	<ul style="list-style-type: none">- Conduct interviews with customers or field experts.- Request internal documents, workflows, or reports.
2. Analyze Domain Documents	<ul style="list-style-type: none">- Study specialized documents like research papers, industry standards, or guidelines.- Gather insights from existing systems.
3. Identify Key Terms	<ul style="list-style-type: none">- Note down frequently used terms in the application domain.- Categorize keywords into technical terms, business concepts, and processes.
4. Validate with Customer	<ul style="list-style-type: none">- Share the keyword list with the customer for verification.- Clarify meanings of terms to ensure a common understanding.
5. Update the List	<ul style="list-style-type: none">- Revise and expand the list based on customer feedback and development team input.

- Gửi danh sách từ khóa cho khách hàng xác nhận.
- Thống nhất ý nghĩa và cách sử dụng các thuật ngữ trong suốt quá trình phát triển dự án.

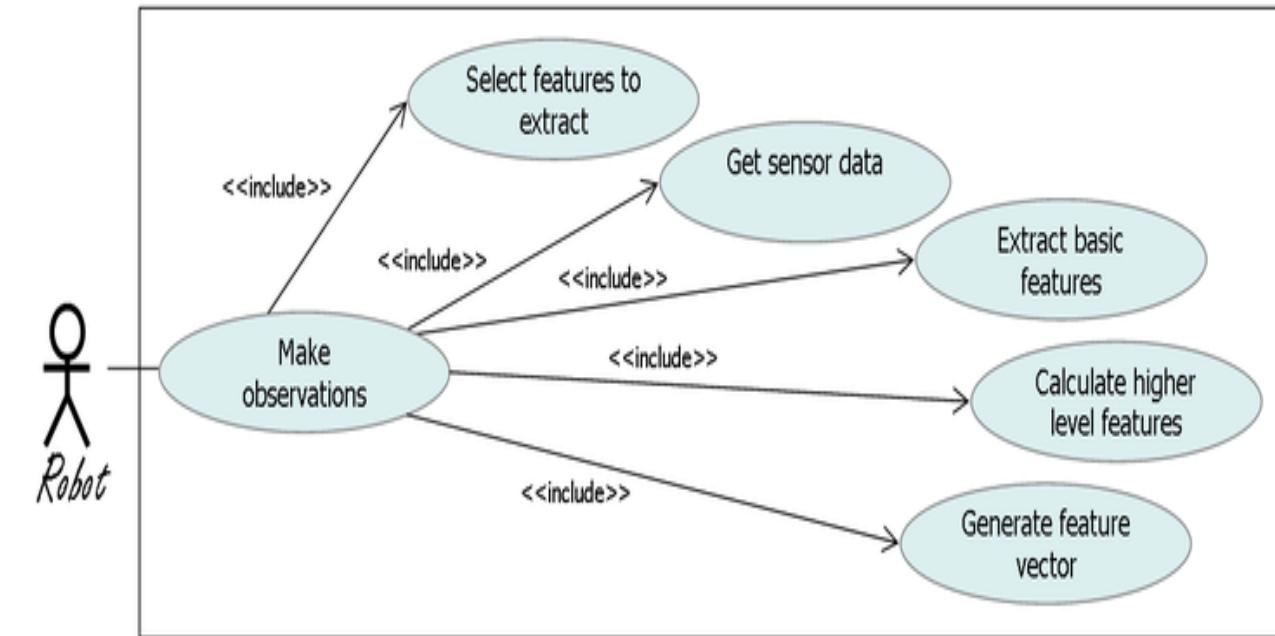


Use case: Mô tả các chức năng mà hệ thống cung cấp cho người dùng.

Các bước trích use case:

Xác định các tác nhân bên ngoài hệ thống (người dùng hoặc hệ thống khác).

Xác định các hành động chính mà tác nhân hiện với hệ thống.



Mô tả chi tiết nghiệp vụ cho từng use case**Nội dung mô tả:**

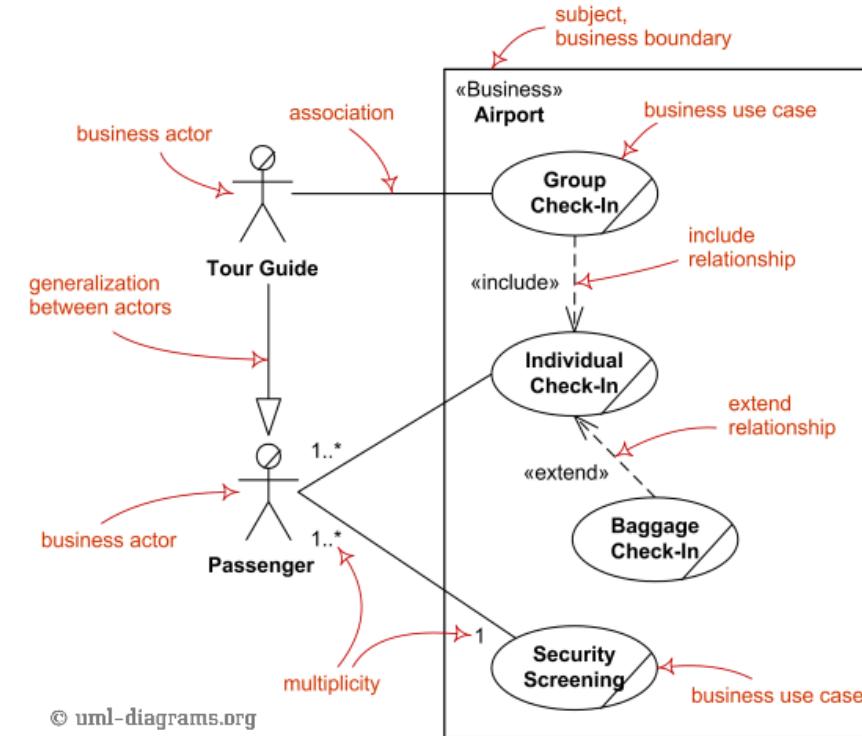
Tên use case.

Tác nhân tham gia.

Dòng sự kiện chính.

Các điều kiện trước và sau khi thực hiện use case.

Dòng sự kiện thay thế hoặc ngoại lệ (nếu có).



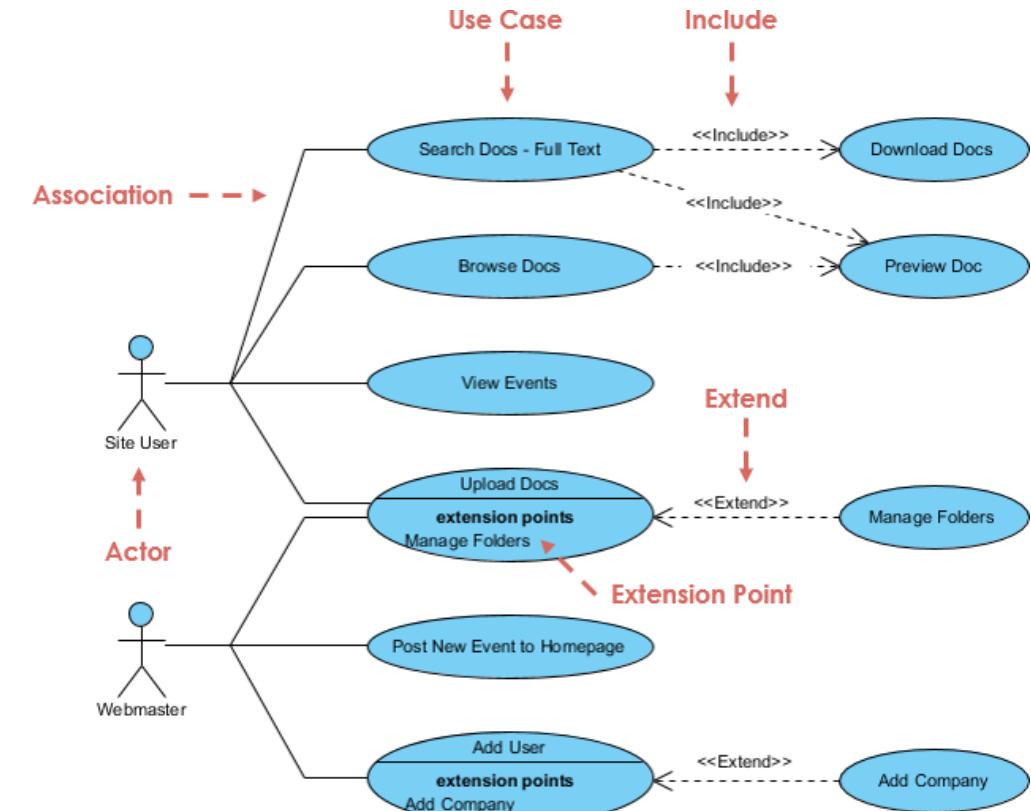
□ Mô tả quan hệ giữa các use case

Các loại quan hệ:

Include: Một use case bao gồm một use case khác.

Extend: Một use case mở rộng thêm chức năng cho một use case khác.

Generalization: Use case tổng quát hóa hoặc cụ thể hóa các use case khác.



Một số lưu ý khi trích các use case

Đảm bảo mỗi use case thể hiện một chức năng rõ ràng.

Tránh viết quá chi tiết hoặc quá chung chung.

Kiểm tra sự nhất quán giữa các use case.

Element	Description
Use Case Name	Clearly describe the goal (e.g., "Place an Order").
Actors	List primary and secondary actors involved in the interaction.
Preconditions	State conditions that must be true before the use case can be executed.
Basic Flow	Describe the step-by-step interaction for achieving the goal.
Alternative Flow	Outline variations in the flow due to different conditions.
Exception Flow	Describe error-handling or unexpected scenarios.
Postconditions	Define the state of the system after the use case is successfully executed.

☐ Xây dựng danh sách các từ khóa chuyên môn của ứng dụng Ví dụ với ứng dụng Hệ thống thương mại điện tử.

Loại từ khóa	Từ khóa chuyên môn
Thuật ngữ kỹ thuật	CMS (Content Management System), CRM (Customer Relationship Management), API (Application Programming Interface).
Quy trình nghiệp vụ	Thêm vào giỏ hàng, Thanh toán trực tuyến, Hoàn trả sản phẩm, Theo dõi đơn hàng.
Chức năng chính	Quản lý sản phẩm, Quản lý khách hàng, Quản lý giao hàng, Thống kê doanh thu.

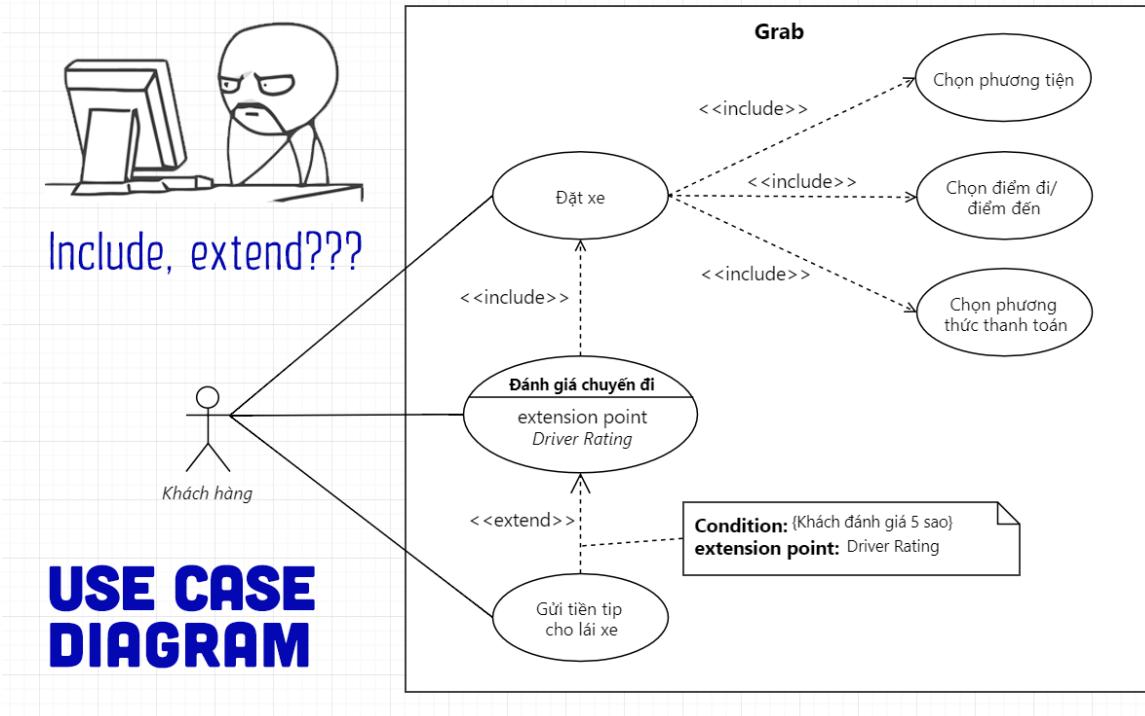
Trích các use case của ứng dụng

Ví dụ:

Use case 1: Quản lý sản phẩm.

Use case 2: Quản lý khách hàng

Use case 3: Quản lý giao hàng.



SWE | 5.4.3. MÔ TẢ CHI TIẾT CHO TỪNG USE CASE

❑ Use case: Mượn sách

Tác nhân: Độc giả, Thủ thư.

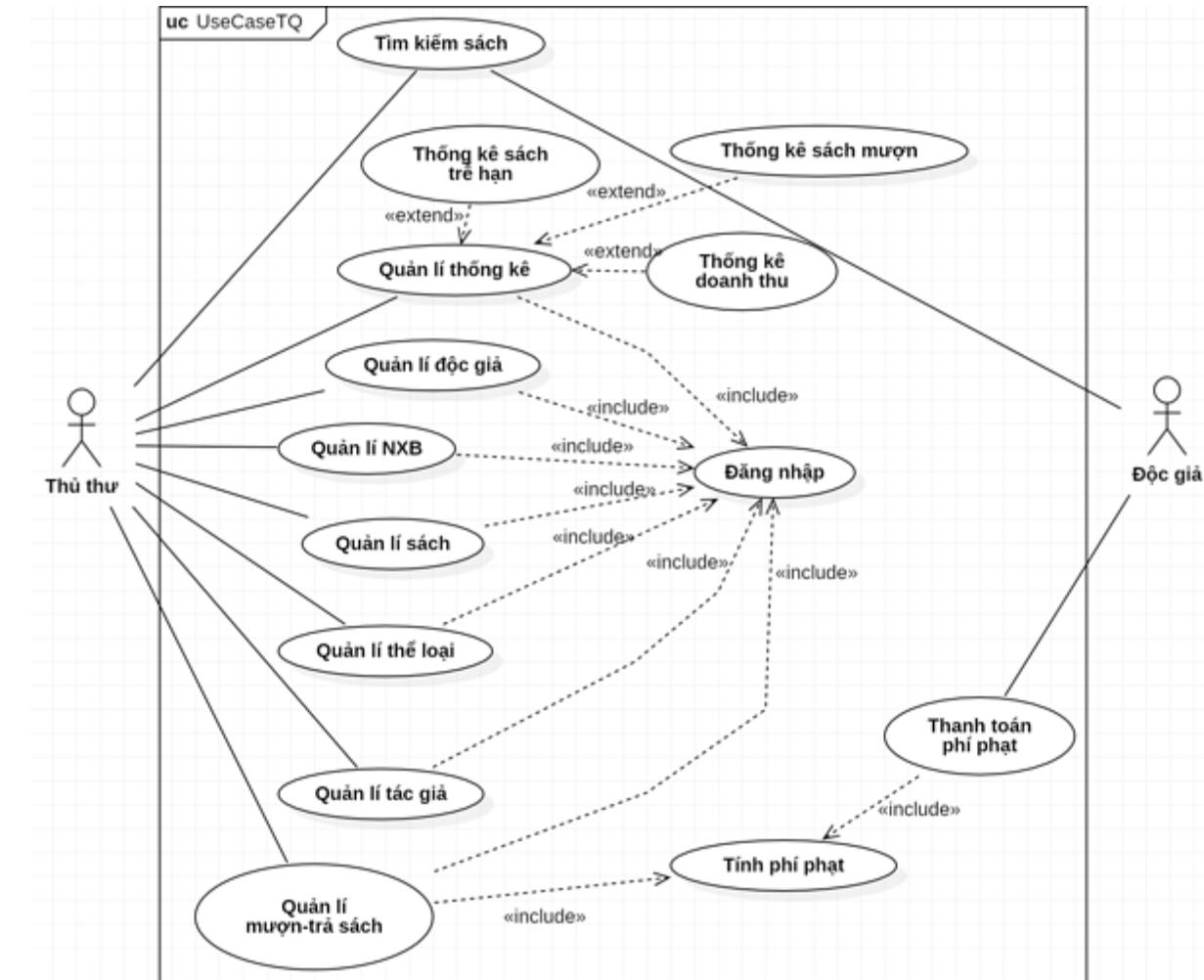
Dòng sự kiện chính:

Độc giả yêu cầu mượn sách.

Thủ thư kiểm tra tình trạng sách.

Thủ thư ghi nhận thông tin mượn sách vào hệ thống.

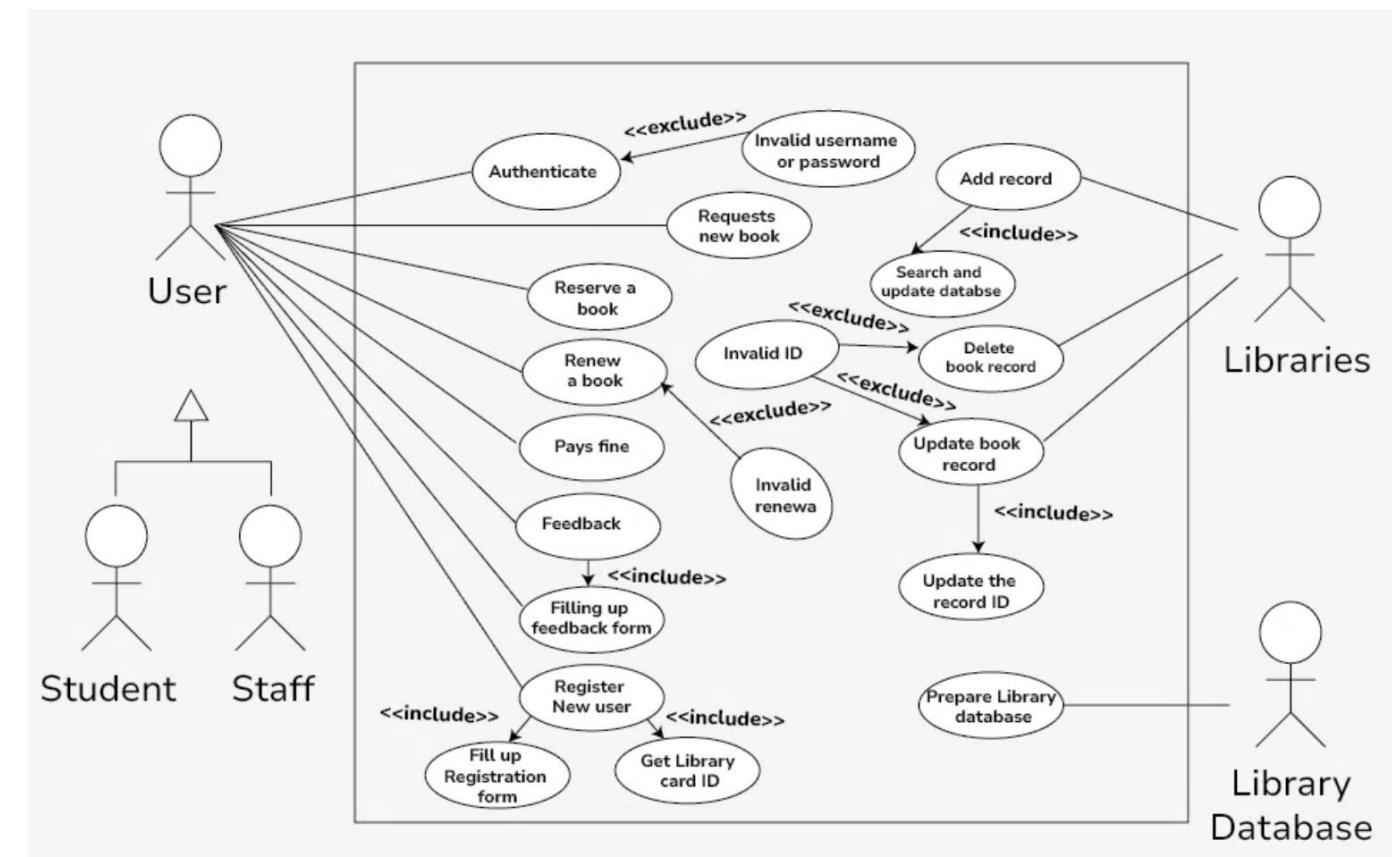
Hệ thống cập nhật trạng thái của sách thành “Đang được mượn”.



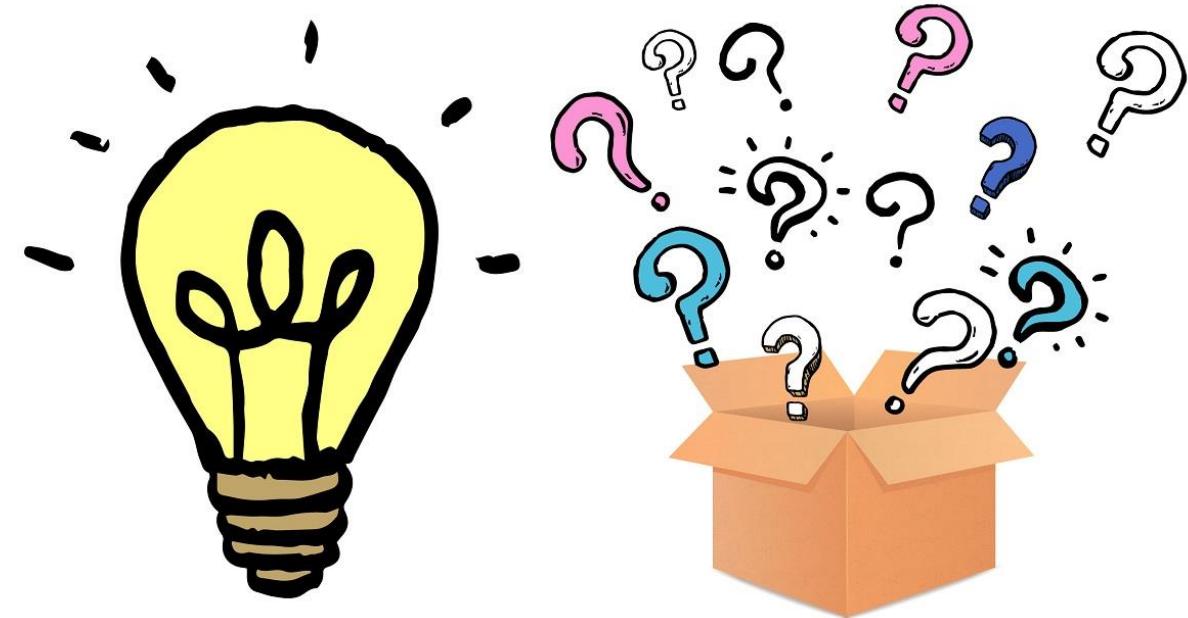
□ Mô tả quan hệ giữa các use case

Include: Use case "Mượn sách" bao gồm use case "Kiểm tra tình trạng sách".

Extend: Use case "Quản lý danh mục sách" có thể mở rộng thêm chức năng "Tìm kiếm sách".



1. CÂU HỎI TRẮC NGHIỆM
2. CÂU HỎI TRẢ LỜI NGẮN
3. CÂU HỎI THẢO LUẬN NHÓM
4. CÂU HỎI TÌNH HUỐNG



1. Câu hỏi 1: Kỹ thuật nào sau đây thường được sử dụng để thu thập yêu cầu khách hàng?

- A. Lập trình theo cặp
- B. Phỏng vấn khách hàng
- C. Kiểm thử đơn vị
- D. Triển khai hệ thống

2. Câu hỏi 2: Trong bước tổng hợp kết quả yêu cầu, việc nào sau đây là cần thiết?

- A. Xây dựng sơ đồ lớp
- B. Phân loại yêu cầu và loại bỏ thông tin trùng lặp
- C. Viết mã nguồn
- D. Thiết kế giao diện

3. Câu hỏi 3: Use case mô tả:

- A. Cách kiểm thử phần mềm
- B. Cách người dùng tương tác với hệ thống
- C. Cách quản lý tài liệu
- D. Cách bảo trì phần mềm

4. Câu hỏi 4: Tại sao cần xây dựng danh sách từ khóa chuyên môn khi tìm hiểu lĩnh vực ứng dụng?

- A. Để giảm thời gian lập trình
- B. Để tăng tốc độ kiểm thử
- C. Để đảm bảo đội phát triển và khách hàng hiểu rõ các thuật ngữ
- D. Để giảm chi phí phát triển

5. Câu hỏi 5: Quan hệ nào sau đây trong use case mô tả một use case có thể mở rộng thêm chức năng trong một số điều kiện nhất định?

- A. Include
- B. Extend
- C. Generalization
- D. Aggregation

6. **Câu hỏi 6:** Khi mô tả yêu cầu bằng ngôn ngữ tự nhiên, điều quan trọng nhất là gì?
 - A. Sử dụng nhiều thuật ngữ kỹ thuật
 - B. Viết thật ngắn gọn và không cần kiểm tra lại với khách hàng
 - C. Viết rõ ràng, dễ hiểu và tránh từ đa nghĩa
 - D. Chỉ tập trung vào các yêu cầu phi chức năng
7. **Câu hỏi 7:** Mục tiêu chính của việc trích các use case là gì?
 - A. Thiết kế giao diện người dùng
 - B. Mô tả các chức năng mà hệ thống cung cấp cho người dùng
 - C. Viết mã nguồn cho hệ thống
 - D. Lập kế hoạch kiểm thử
8. **Câu hỏi 8:** Quan hệ nào giữa các use case thể hiện việc một use case phải gọi một use case khác để thực hiện chức năng đầy đủ?
 - A. Include
 - B. Extend
 - C. Generalization
 - D. Dependency
9. **Câu hỏi 9:** Hoạt động nào sau đây là bước đầu tiên trong việc xây dựng mô hình nghiệp vụ?
 - A. Thiết kế giao diện người dùng
 - B. Trích các use case
 - C. Viết mã nguồn
 - D. Thực hiện kiểm thử hệ thống
10. **Câu hỏi 10:** Một yêu cầu chức năng là gì?
 - A. Một yêu cầu mô tả cách hệ thống xử lý dữ liệu
 - B. Một yêu cầu về tốc độ xử lý của hệ thống
 - C. Một yêu cầu về khả năng bảo trì phần mềm
 - D. Một yêu cầu về giao diện người dùng

1. Kỹ thuật phỏng vấn khách hàng là gì?
2. Tại sao cần tổng hợp kết quả yêu cầu?
3. Use case là gì?
4. Mục tiêu của việc xây dựng danh sách từ khóa chuyên môn là gì?
5. Quan hệ Include giữa các use case là gì?
6. Quan hệ Extend giữa các use case là gì?
7. Yêu cầu phi chức năng là gì?
8. Mô hình nghiệp vụ là gì?
9. Điều kiện trước và điều kiện sau của use case là gì?
10. Một số lưu ý khi trích các use case là gì?

1. Khách hàng không hiểu rõ các thuật ngữ kỹ thuật trong tài liệu yêu cầu. Là trưởng nhóm phát triển, bạn sẽ làm gì?
2. Trong quá trình lấy yêu cầu, khách hàng liên tục thay đổi ý kiến về chức năng cần thiết. Đội phát triển nên xử lý ra sao?
3. Một dự án phần mềm quản lý bán hàng gặp vấn đề khi các yêu cầu được mô tả quá chung chung, khó thực hiện. Đội phát triển cần làm gì để khắc phục?
4. Khách hàng đưa ra yêu cầu không rõ ràng và chỉ có thể mô tả một cách sơ lược. Làm thế nào để thu thập yêu cầu đầy đủ từ khách hàng?
5. Trong quá trình xây dựng mô hình nghiệp vụ, một số use case bị trùng lặp về chức năng. Bạn sẽ xử lý tình huống này như thế nào?
6. Đội phát triển và khách hàng có ý kiến khác nhau về ý nghĩa của một số từ khóa chuyên môn. Là trưởng dự án, bạn sẽ làm gì?
7. Sau khi hoàn thành việc trích các use case, đội phát triển nhận ra rằng một số chức năng quan trọng bị bỏ sót. Hãy đề xuất cách giải quyết.
8. Trong quá trình xây dựng mô hình nghiệp vụ, khách hàng yêu cầu bổ sung thêm một số chức năng mới. Đội phát triển nên làm gì?
9. Một nhóm phát triển gặp khó khăn trong việc mô tả chi tiết các use case vì chưa hiểu rõ quy trình nghiệp vụ. Hãy đề xuất giải pháp.
10. Khách hàng yêu cầu thêm một chức năng mới khi hệ thống đã bước vào giai đoạn thiết kế chi tiết. Đội phát triển cần xử lý như thế nào?

LẤY YÊU CẦU- REQUIREMENT

YÊU CẦU PHẦN MỀM GÌ ?
(What is a Software Requirement?)

AI CẦN DÙNG APP GÌ
? (WHAT?)

AI YÊU PHẦN MỀM ?
(Who Defines Software Requirements?)

AI LÀ NGƯỜI DÙNG
APP? (WHO?)

TẠI SAO Y/C PHẦN MỀM QUAN
TRỌNG?
(Why Are Software Requirements Important?)

AI TẠI SAO PHẢI
DÙNG APP (WHY?)

1. Giới thiệu dự án

- **Tên dự án:** Hệ thống quản lý khám chữa bệnh cho Bệnh viện Chợ Rẫy (**HIS - Hospital Information System**)

Mục tiêu:

- Xây dựng một hệ thống phần mềm hỗ trợ toàn diện công tác quản lý, khám chữa bệnh, hồ sơ bệnh án điện tử, lịch hẹn và thanh toán viện phí.
- Giảm tải công việc hành chính, nâng cao hiệu suất và độ chính xác trong điều trị.

2. Đối tượng sử dụng

- **Bệnh nhân:** Đăng ký khám, theo dõi hồ sơ bệnh án, đặt lịch hẹn.
- **Bác sĩ:** Xem lịch khám, hồ sơ bệnh nhân, nhập liệu và kê đơn thuốc.
- **Y tá & Điều dưỡng:** Hỗ trợ bác sĩ, cập nhật tình trạng bệnh nhân
- **Nhân viên tiếp nhận:** Quản lý lịch hẹn, đăng ký khám, thông tin bệnh nhân.
- **Quản trị viên:** Quản lý hệ thống, phân quyền người dùng, thống kê báo cáo.

3. Đặc tả yêu cầu phần mềm

3.1. Yêu cầu chức năng (Functional Requirements)

◆ Đăng ký khám bệnh & Quản lý bệnh nhân

- Bệnh nhân có thể đăng ký khám online hoặc tại quầy tiếp nhận.
- Hệ thống lưu trữ thông tin cá nhân, lịch sử khám, đơn thuốc, kết quả xét nghiệm.
- Hỗ trợ tìm kiếm, tra cứu bệnh án theo mã bệnh nhân, tên, số CMND/CCCD.

◆ Hệ thống hồ sơ bệnh án điện tử (EMR - Electronic Medical Record)

- Bác sĩ có thể ghi chú, chẩn đoán bệnh, kê đơn thuốc, chỉ định xét nghiệm trực tuyến.
- Hồ sơ bệnh án được cập nhật tự động và đồng bộ hóa giữa các khoa.
- Hỗ trợ tích hợp dữ liệu hình ảnh y khoa (X-ray, MRI, CT scan).

◆ Quản lý lịch khám & Điều phối bác sĩ

- Tạo, xem và quản lý lịch khám bệnh của từng bác sĩ.
- Hỗ trợ đặt lịch hẹn trực tuyến và gửi nhắc lịch qua SMS/Zalo.
- Điều phối bác sĩ linh hoạt, tránh quá tải phòng khám.

◆ Quản lý xét nghiệm & Chẩn đoán hình ảnh

- Hệ thống tự động cập nhật kết quả xét nghiệm vào hồ sơ bệnh nhân.
- Hỗ trợ tích hợp thiết bị xét nghiệm y tế (máy xét nghiệm sinh hóa, huyết học).
- Lưu trữ và truy xuất hình ảnh chẩn đoán y khoa.

◆ Quản lý thuốc & Nhà thuốc bệnh viện

- Theo dõi kê đơn thuốc điện tử, kết nối với nhà thuốc bệnh viện.
- Quản lý tồn kho thuốc, hạn sử dụng, nhập/xuất thuốc.
- Kiểm tra tương tác thuốc, cảnh báo thuốc bị cấm hoặc hết hạn.

1.1. Các thực thể chính trong hệ thống

- Hệ thống quản lý khám chữa bệnh cho Bệnh viện Chợ Rẫy bao gồm các thực thể chính:
 - Bệnh nhân (Patient)**: Lưu trữ thông tin bệnh nhân.
 - Bác sĩ (Doctor)**: Thông tin bác sĩ, chuyên khoa.
 - Lịch hẹn khám (Appointment)**: Quản lý lịch hẹn khám của bệnh nhân.
 - Hồ sơ bệnh án (MedicalRecord)**: Ghi lại lịch sử khám, chẩn đoán, đơn thuốc.
 - Dịch vụ y tế (MedicalService)**: Danh sách dịch vụ khám chữa bệnh.
 - Xét nghiệm (LabTest)**: Quản lý xét nghiệm và kết quả xét nghiệm.
 - Đơn thuốc (Prescription)**: Kê đơn thuốc của bác sĩ cho bệnh nhân.
 - Nhà thuốc (Pharmacy)**: Quản lý kho thuốc bệnh viện.
 - Hóa đơn (Invoice)**: Quản lý thanh toán viện phí.
 - Nhân viên tiếp nhận (Receptionist)**: Tiếp nhận bệnh nhân và sắp xếp lịch hẹn.

4.2. Quan hệ giữa các thực thể

- Một **bệnh nhân** có thể có nhiều **lịch hẹn khám**.
- Một **bệnh nhân** có thể có nhiều **hồ sơ bệnh án**.
- Một **hồ sơ bệnh án** chứa **kê đơn thuốc, dịch vụ y tế và xét nghiệm**.
- Một **bác sĩ** có thể khám cho nhiều **bệnh nhân**.
- Một **đơn thuốc** bao gồm nhiều **thuốc** từ **nha thuốc**.
- Một **hóa đơn** liên kết với **bệnh nhân** và các **dịch vụ đã sử dụng**.

5.1. Các tác nhân trong hệ thống

- 1. Bệnh nhân:** Đặt lịch khám, tra cứu hồ sơ bệnh án, thanh toán viện phí.
- 2. Bác sĩ:** Xem lịch khám, cập nhật hồ sơ bệnh án, kê đơn thuốc.
- 3. Nhân viên tiếp nhận:** Đăng ký khám bệnh, điều phối lịch khám.
- 4. Nhân viên xét nghiệm:** Nhập kết quả xét nghiệm vào hệ thống.
- 5. Nhà thuốc:** Cấp thuốc theo đơn, quản lý kho thuốc.
- 6. Hệ thống bảo hiểm y tế:** Tính toán chi phí và hỗ trợ thanh toán bảo hiểm.

5.2. Danh sách các Use Case

- ◆ Bệnh nhân
- ◆ Bác sĩ
- ◆ Nhân viên tiếp nhận
- ◆ Nhà thuốc
- ◆ Hệ thống bảo hiểm y tế

6. YÊU CẦU THIẾT KẾ CƠ SỞ DỮ LIỆU

1. Bảng BenhNhan (Patient)
2. Bảng BacSi (Doctor)
3. Bảng LichHen (Appointment)
4. Bảng HoSoBenhAn (MedicalRecord)
5. Bảng XetNghiem (LabTest)
6. Bảng DonThuoc (Prescription)
7. Bảng Thuoc (Medicine)
8. Bảng HoaDon (Invoice)

OUTLINE

6.1. PHÂN LOẠI CÁC LỚP TRONG HỆ THỐNG

6.2. VIẾT CÁC KỊCH BẢN SỬ DỤNG (SCENARIO)

6.3. TRÍCH CÁC LỚP

6.4. XÂY DỰNG SƠ ĐỒ LỚP

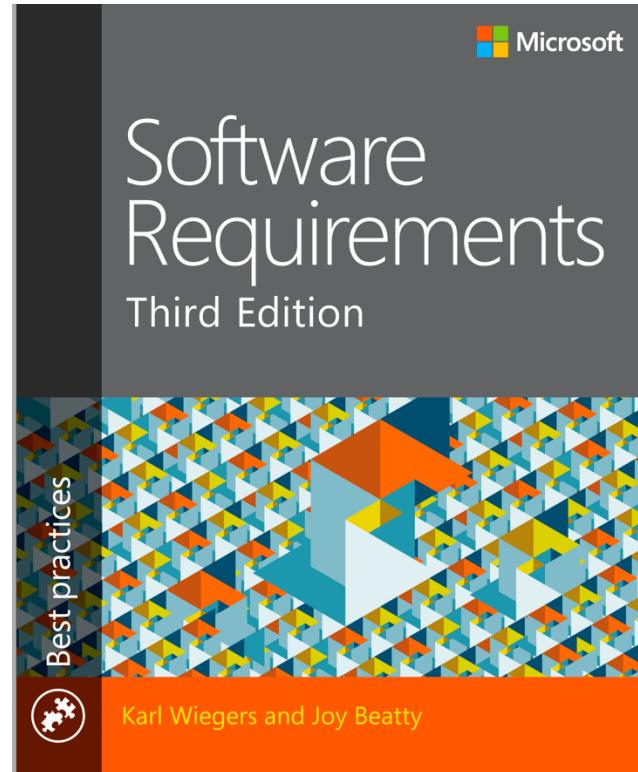
6.5. XÂY DỰNG SƠ ĐỒ TUẦN TỰ, CỘNG TÁC

6.6. CASE STUDY

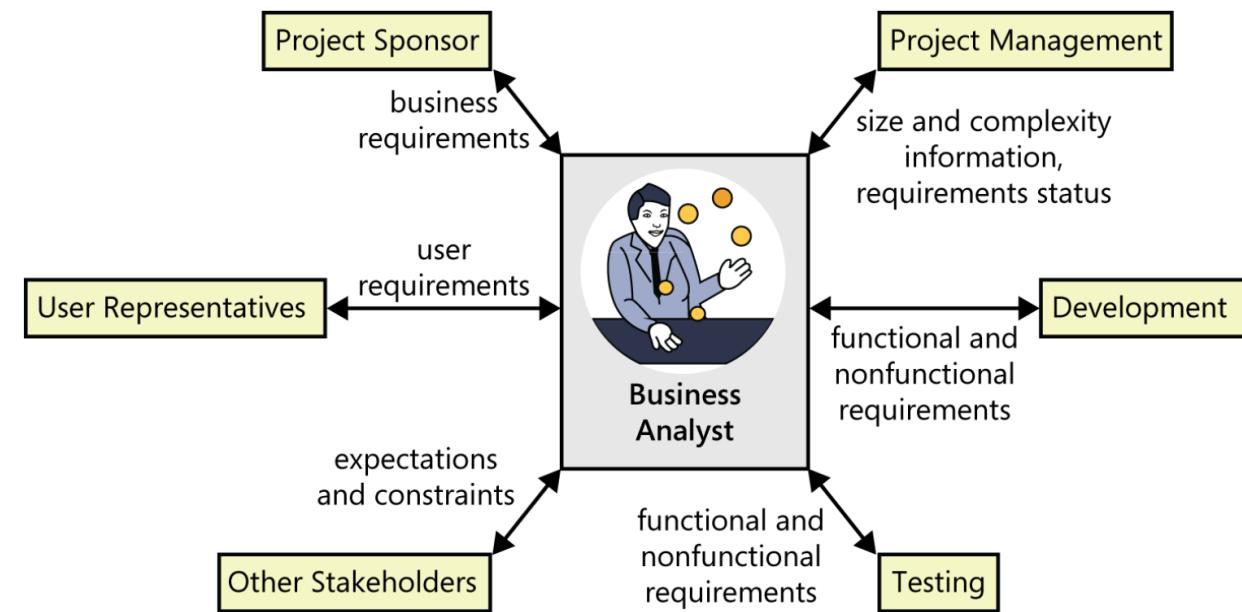
6.7 CÂU HỎI CÙNG CỔ LÝ THUYẾT

NỘI DUNG TÌM ĐỌC CHƯƠNG 11

THE BUSINESS ANALYST ROLE



- Kết nối giao tiếp giữa khách hàng và đội ngũ phát triển.
- Hoạt động như một cầu nối hợp tác để đảm bảo sản phẩm đáp ứng nhu cầu của các bên liên quan.



Miscommunication in Software Development – The Gap Between Expectations and Reality



How the customer explained it



How the project leader understood it



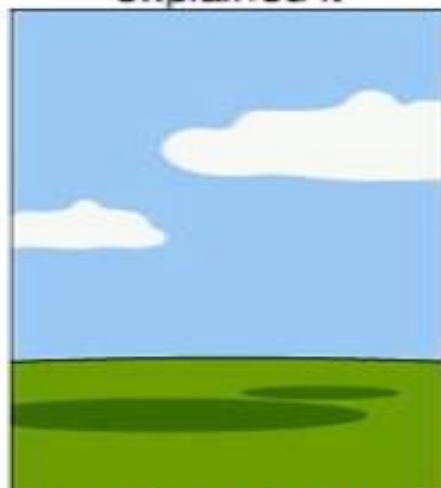
How the engineer designed it



How the programmer wrote it



How the sales executive described it



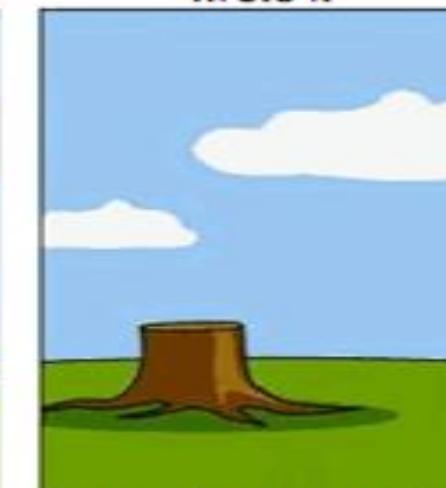
How the project was documented



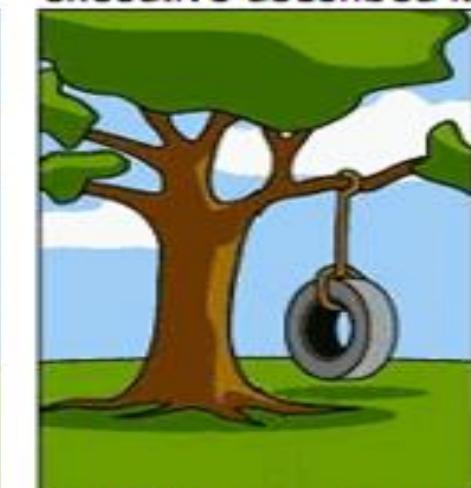
What operations installed



How the customer was billed

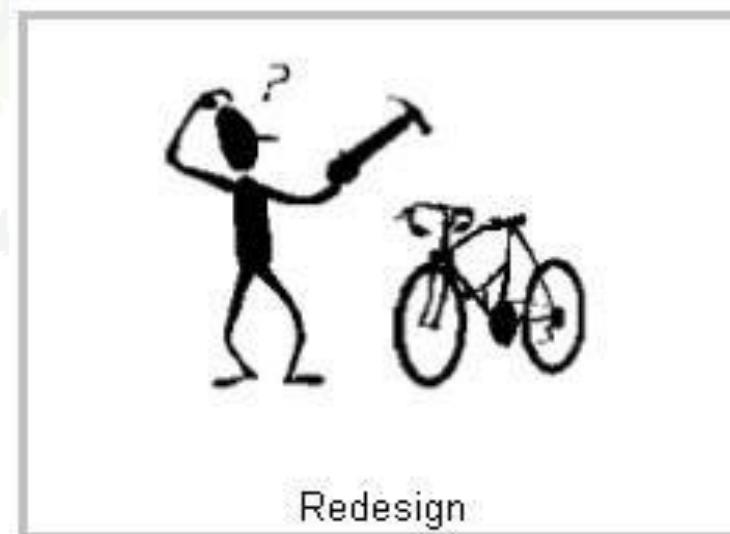
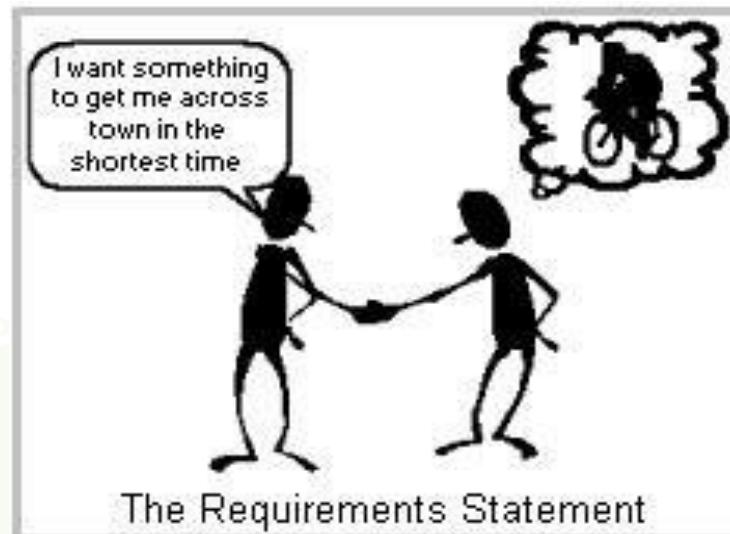


How the helpdesk supported it



What the customer really needed

1. How the customer explained it (Khách hàng giải thích yêu cầu)
- Khách hàng muốn **một chiếc xích đu đơn giản** với dây treo trên cây.
 - Yêu cầu này là **dễ hiểu và thực tế**, nhưng chưa có chi tiết kỹ thuật cụ thể.
2. How the project leader understood it (Trưởng dự án hiểu yêu cầu như thế nào)
- Trưởng dự án hiểu rằng cần một **chiếc xích đu có hai dây buộc vào cành cây**, nhưng không hiểu hết mong muốn của khách hàng.
 - Đây là **sự khác biệt đầu tiên** trong quy trình.
3. How the engineer designed it (Kỹ sư thiết kế nó như thế nào)
- Kỹ sư thiết kế một **chiếc xích đu có ba dây**, nhưng không treo đúng vị trí.
 - Cho thấy **việc chuyển đổi từ hiểu biết sang thiết kế không đúng hoàn toàn**.
4. How the programmer wrote it (Lập trình viên thực hiện như thế nào)
- Lập trình viên viết ra một phiên bản **hoàn toàn sai lầm** với một chiếc dây buộc vào thân cây.
 - Điều này minh họa rằng **lập trình viên có thể không hiểu đúng yêu cầu hoặc không được cung cấp đầy đủ thông tin**.
5. How the sales executive described it (Nhân viên kinh doanh mô tả nó như thế nào)
- Bộ phận kinh doanh có xu hướng **cường điệu sản phẩm**, mô tả một **chiếc ghế sang trọng** thay vì một xích đu đơn giản.
 - Điều này cho thấy **việc truyền đạt sai giữa các bộ phận có thể gây ra kỳ vọng sai lệch cho khách hàng**.
6. How the project was documented (Tài liệu dự án ghi lại như thế nào)
- Không có nội dung gì trong tài liệu (trống trơn), thể hiện **thiếu tài liệu hoặc tài liệu kém chất lượng**.
 - Điều này là vấn đề phổ biến trong các dự án, khi thông tin không được ghi chép đúng.
7. What operations installed (Nhóm triển khai thực tế cài đặt gì)
- Họ chỉ **cài đặt một sợi dây**, bỏ qua toàn bộ khái niệm về xích đu.
 - Đây là **sai lệch nghiêm trọng giữa triển khai thực tế và yêu cầu ban đầu**.
8. How the customer was billed (Khách hàng bị tính phí như thế nào)
- Khách hàng nhận được một hệ thống **phức tạp như tàu lượn siêu tốc**, dù họ chỉ yêu cầu một chiếc xích đu.
 - Điều này thể hiện **chi phí dự án có thể bị đội lên do quản lý kém**.
9. How the helpdesk supported it (Bộ phận hỗ trợ khách hàng xử lý như thế nào)
- Chỉ có **một gốc cây bị chặt**, tương ứng cho việc **hỗ trợ kỹ thuật không có giá trị thực tế**.
 - Khách hàng không nhận được **giải pháp hữu ích**.
10. What the customer really needed (Khách hàng thực sự cần gì)
- Khách hàng chỉ cần **một chiếc lốp xe treo trên cây**, đơn giản và hiệu quả.
 - Điều này thể hiện **sự khác biệt lớn giữa nhu cầu thực tế và những gì được triển khai**.



❑ MÔ TẢ TƯNG KHUNG HÌNH:

1. The Requirements Statement (Yêu cầu ban đầu)

- Người dùng ban đầu chỉ **yêu cầu một phương tiện giúp họ di chuyển nhanh nhất trong thị trấn**.
- Đây là một yêu cầu rất chung chung, không đủ chi tiết để phát triển đúng ngay từ đầu.

2. Emerging Requirements (Các yêu cầu phát sinh)

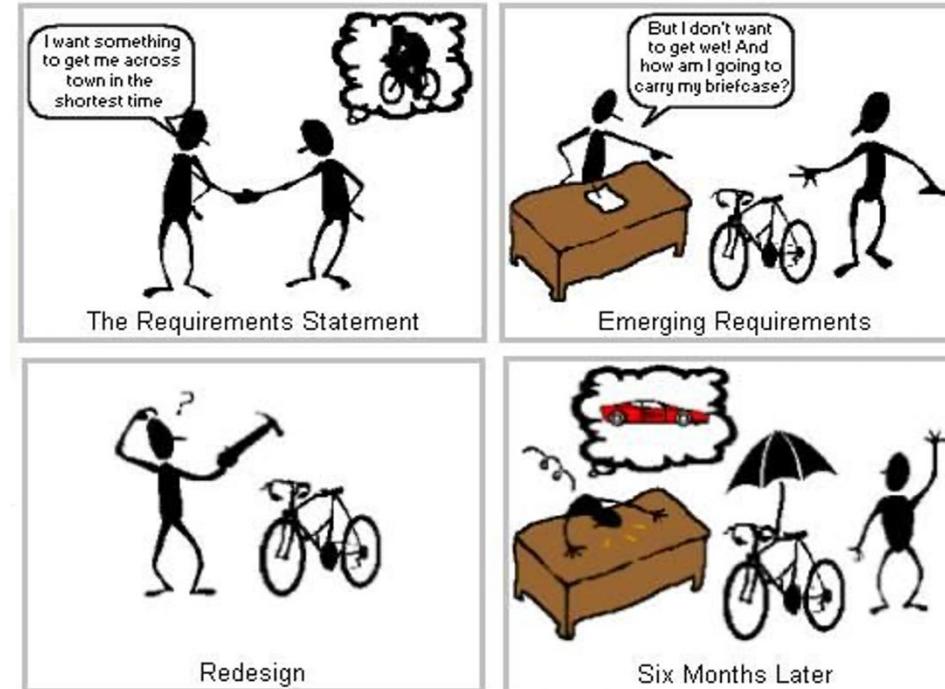
- Khi bắt đầu thực hiện, khách hàng **bắt đầu nêu thêm yêu cầu mới** như không muốn bị ướt khi trời mưa và cần chỗ để cất tài liệu.
- Điều này thường xảy ra trong thực tế khi khách hàng không nghĩ đến toàn bộ yêu cầu ngay từ đầu, dẫn đến nhiều thay đổi trong quá trình phát triển.

3. Redesign (Thiết kế lại)

- Nhóm phát triển phải thay đổi thiết kế **để đáp ứng các yêu cầu mới phát sinh**.
- Đây là một **vấn đề phổ biến trong phát triển phần mềm**, khi yêu cầu thay đổi liên tục dẫn đến công sức thiết kế lại rất nhiều.

4. Six Months Later (Sáu tháng sau)

- Sản phẩm cuối cùng hoàn toàn **khác xa so với mong đợi ban đầu**, có nhiều tính năng không cần thiết.
- Điều này phản ánh việc **thiếu quản lý yêu cầu đúng cách dẫn đến sản phẩm bị làm quá mức hoặc không đúng nhu cầu thực sự của khách hàng**.



❖ MÔ TẢ TƯNG KHUNG HÌNH:

1. Câu hỏi về bảo mật

Đội ngũ phát triển nhận thấy sản phẩm của họ có rất nhiều **lỗ hổng bảo mật** và đang tìm cách khắc phục.

2. Các cách giải quyết được đề xuất

Một số thành viên đề xuất **chạy trình quét bảo mật** hoặc **gọi lỗ hổng là tính năng**, thay vì giải quyết vấn đề từ gốc rễ.

Một thành viên đề xuất **phát triển bảo mật ngay từ đầu**, nhưng không ai quan tâm đến giải pháp này.

3. Hậu quả

Một cảnh hài hước cho thấy **hệ thống có thể bị tấn công hoặc sập** do không được xây dựng bảo mật đúng cách.



❑ Phân tích từng phần trong bức ảnh:

1. Requirements (Yêu cầu)
2. Brainstorming (Lên ý tưởng)
3. Budget (Ngân sách)
4. Implementation (Triển khai thực tế)

❑ Ý nghĩa thực tế:

👉 Khoảng cách giữa yêu cầu KH và sản phẩm thực tế:

- KH mong đợi một sản phẩm đơn giản, dễ sử dụng.
- Nhưng khi lên ý tưởng, đội ngũ phát triển thường làm phức tạp vấn đề.
- Khi triển khai, ngân sách bị giới hạn, dẫn đến một sản phẩm hoàn toàn khác xa với yêu cầu ban đầu.

👉 Sai lầm trong quy trình phát triển phần mềm / sản phẩm:

- Các nhóm phát triển có thể **thêm quá nhiều tính năng không cần thiết**.
- **Ngân sách và nguồn lực hạn chế** khiến sản phẩm cuối cùng kém chất lượng.
- KH cuối cùng **không nhận được sản phẩm như mong đợi**.

REQUIREMENTS VS. IMPLEMENTATION

- SIMPLE INTERFACE - ACCOMODATE ALL USERS
- CUSTOMIZABLE
- SECURE
- LOW MAINTENANCE

REQUIREMENTS



BRAINSTORMING



BUDGET



IMPLEMENTATION



MONKEYUSER.COM

1 Định nghĩa lớp thực thể (Entity Class)

- Lớp thực thể là **các lớp mô tả các thực thể trong hệ thống phần mềm**, thường được ánh xạ trực tiếp đến bảng trong cơ sở dữ liệu.
- Lớp thực thể lưu trữ dữ liệu **có trạng thái** và thường **tồn tại lâu dài** trong hệ thống.

2 Đặc điểm của lớp thực thể

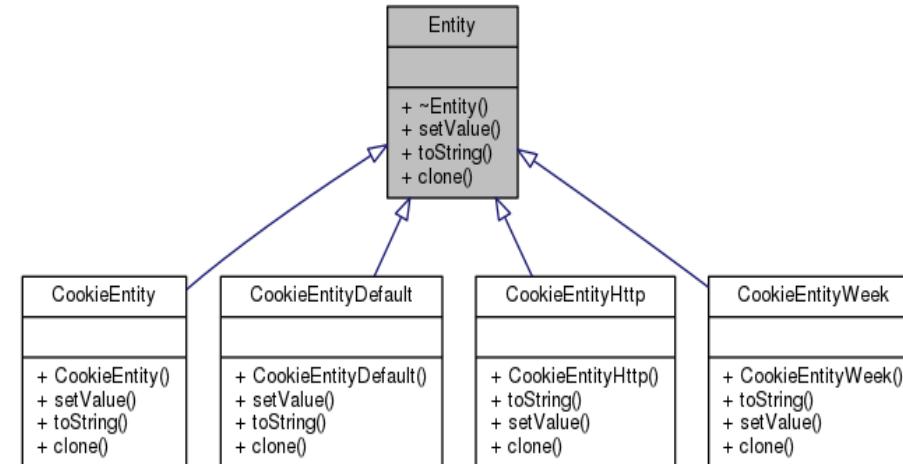
- Chứa dữ liệu (Attributes):** Các thuộc tính biểu diễn thông tin của thực thể (ví dụ: tên, tuổi, địa chỉ).
- Có định danh duy nhất (ID hoặc Primary Key):** Giúp xác định một thực thể cụ thể.
- Có các phương thức (Methods):** Thường chỉ để truy xuất hoặc cập nhật dữ liệu chứ không chứa logic nghiệp vụ phức tạp.

3 Vai trò trong hệ thống phần mềm

- Lưu trữ thông tin** quan trọng trong hệ thống.
- Hỗ trợ truy xuất và quản lý dữ liệu**, giúp hệ thống dễ dàng xử lý thông tin.
- Tạo mối quan hệ với các lớp khác**, như lớp điều khiển (Controller) hoặc lớp giao diện (UI).

➡ **Tóm tắt:**

- Lớp thực thể là thành phần quan trọng trong hệ thống phần mềm vì nó quản lý dữ liệu cốt lõi.
- Thiết kế lớp thực thể đúng cách giúp **tăng hiệu suất truy xuất dữ liệu** và **cải thiện khả năng mở rộng hệ thống**.
- Cần đảm bảo các lớp thực thể **được tổ chức chặt chẽ** và **có mối quan hệ hợp lý** với các thành phần khác của hệ thống.



LỚP THỰC THỂ (ENTITY CLASS) (tt)

❑ Ví dụ về lớp thực thể:

Hệ thống quản lý sinh viên

java

```
class SinhVien {  
    private int maSinhVien;  
    private String hoTen;  
    private String ngaySinh;  
    private String diaChi;  
  
    // Constructor  
    public SinhVien(int maSinhVien, String hoTen, String ngaySinh, String diaChi) {  
        this.maSinhVien = maSinhVien;  
        this.hoTen = hoTen;  
        this.ngaySinh = ngaySinh;  
        this.diaChi = diaChi;  
    }  
  
    // Getters and Setters  
    public int getMaSinhVien() { return maSinhVien; }  
    public String getHoTen() { return hoTen; }  
    public String getNgaySinh() { return ngaySinh; }  
    public String getDiaChi() { return diaChi; }  
}
```

□ Lớp Điều Khiển (Control Class) là gì?

1. Khái niệm

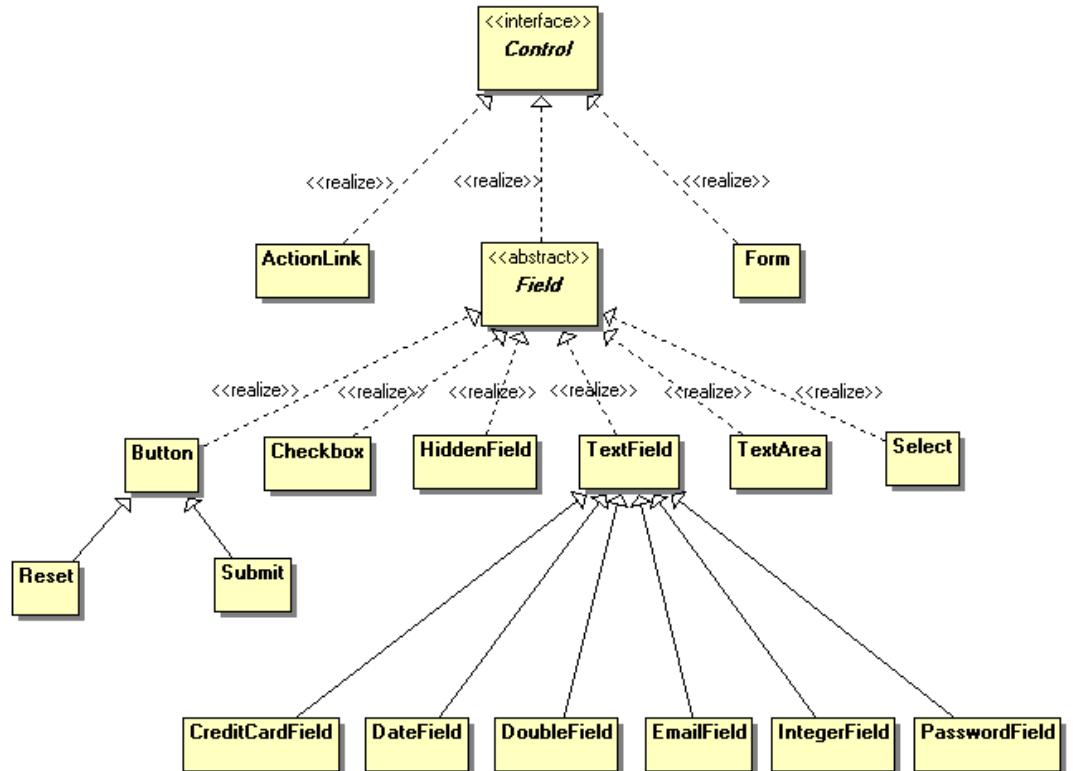
- Lớp điều khiển (Control Class) là một loại lớp trong mô hình hướng đối tượng, đảm nhiệm vai trò trung gian giữa **lớp giao diện (Boundary Class)** và **lớp thực thể (Entity Class)**. Lớp này chứa logic xử lý nghiệp vụ và điều phối luồng dữ liệu trong hệ thống.

2. Vai trò của lớp điều khiển

- Xử lý logic nghiệp vụ chính của ứng dụng.
- Điều phối các tương tác giữa giao diện người dùng và dữ liệu.
- Tạo điều kiện cho các lớp khác giao tiếp mà không làm tăng sự phụ thuộc lẫn nhau.

3. Ví dụ minh họa -> (next slide)

- Trong một hệ thống đặt hàng trực tuyến, ta có các lớp:
- **Lớp giao diện (Boundary Class)**: OrderForm (Giao diện đặt hàng).
- **Lớp thực thể (Entity Class)**: Order (Đơn hàng), Customer (Khách hàng).
- **Lớp điều khiển (Control Class)**: OrderController chịu trách nhiệm xử lý yêu cầu đặt hàng, kiểm tra tồn kho, và tạo đơn hàng.



❑ Example of a Control Class in a Shopping Cart System

Scenario:

A shopping cart system where users can add items to their cart, remove items, and proceed to checkout.

1. Identifying the Classes

- **Boundary Class (User Interface Layer)**

- ShoppingCartUI: Handles user interactions, such as clicking buttons to add/remove items.

- **Entity Class (Data Layer)**

- Product: Represents an item available for purchase.
- ShoppingCart: Stores the list of items the user wants to buy.

- **Control Class (Business Logic Layer)**

- ShoppingCartController: Manages the logic for adding/removing items and calculating totals.

2. Implementation of Control Class

java

 Copy  Edit

```
public class ShoppingCartController {
    private ShoppingCart cart;

    public ShoppingCartController() {
        this.cart = new ShoppingCart();
    }

    // Add item to cart
    public void addItem(Product product, int quantity) {
        cart.addProduct(product, quantity);
        System.out.println(quantity + " " + product.getName() + " added to cart.");
    }

    // Remove item from cart
    public void removeItem(Product product) {
        cart.removeProduct(product);
        System.out.println(product.getName() + " removed from cart.");
    }

    // Get total cost of items in cart
    public double getTotalCost() {
        return cart.calculateTotal();
    }
}
```

3. Supporting Entity Class for ShoppingCart

```
java

import java.util.HashMap;
import java.util.Map;

public class ShoppingCart {
    private Map<Product, Integer> items = new HashMap<>();

    public void addProduct(Product product, int quantity) {
        items.put(product, items.getOrDefault(product, 0) + quantity);
    }

    public void removeProduct(Product product) {
        items.remove(product);
    }

    public double calculateTotal() {
        double total = 0;
        for (Map.Entry<Product, Integer> entry : items.entrySet()) {
            total += entry.getKey().getPrice() * entry.getValue();
        }
        return total;
    }

    public int getTotalItems() {
        return items.size();
    }
}
```

4. Entity Class for Product

3/10/2025 9:17 AM

java

```
public class Product {  
    private String name;  
    private double price;  
  
    public Product(String name, double price) {  
        this.name = name;  
        this.price = price;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public double getPrice() {  
        return price;  
    }  
}
```

5. Explanation (Giải thích)

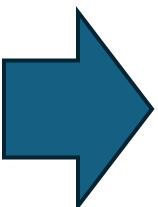
- The ShoppingCartController is a **Control Class** because it:
 - Acts as an intermediary between the UI and the data.
(Hoạt động như một trung gian giữa giao diện người dùng và dữ liệu.)
 - Implements business logic (e.g., adding/removing products, checking total cost).
(Thực hiện logic nghiệp vụ như thêm/xóa sản phẩm, kiểm tra tổng chi phí.)
 - Ensures that the **ShoppingCart** (Entity Class) is updated properly.
(Đảm bảo rằng lớp thực thể ShoppingCart được cập nhật đúng cách.)
- Why use a Control Class? (Tại sao sử dụng Lớp Điều Khiển?)
 - It separates UI logic from business logic.
(Nó tách biệt logic giao diện người dùng khỏi logic nghiệp vụ.)
 - It makes the system more modular and easier to maintain.
(Làm cho hệ thống có tính mô-đun cao hơn và dễ bảo trì hơn.)
 - It prevents the UI layer from directly modifying the data layer.
(Ngăn chặn lớp giao diện người dùng thao tác trực tiếp với lớp dữ liệu.)

4. Đặc điểm của lớp điều khiển

- Không chứa dữ liệu trạng thái lâu dài.
- Chỉ tập trung vào xử lý nghiệp vụ và điều phối thông tin.
- Giúp duy trì tính tách biệt giữa giao diện và dữ liệu, dễ dàng mở rộng và bảo trì hệ thống.

5. Khi nào nên sử dụng lớp điều khiển?

- Khi cần tổ chức logic nghiệp vụ một cách rõ ràng và có thể tái sử dụng.
- Khi hệ thống có nhiều tầng xử lý phức tạp cần điều phối.
- Khi muốn giảm sự phụ thuộc giữa giao diện và dữ liệu để dễ dàng thay đổi từng phần riêng lẻ.



👉 **Ghi chú:**

- Lớp điều khiển đóng vai trò quan trọng trong kiến trúc phần mềm, giúp hệ thống dễ bảo trì, mở rộng và tách biệt rõ ràng các thành phần.

❑ Định nghĩa:

- **Lớp Biên (Boundary Class)** là một lớp đóng vai trò giao tiếp giữa hệ thống và môi trường bên ngoài, bao gồm người dùng và các hệ thống khác. Nó chịu trách nhiệm quản lý tương tác, xử lý đầu vào từ người dùng và hiển thị kết quả.

❑ Characteristics of Boundary Class (Đặc điểm của Lớp Biên):

- Hoạt động như một cầu nối giữa người dùng và hệ thống.
- Xử lý và xác thực đầu vào trước khi chuyển cho lớp điều khiển.
- Hiển thị thông tin phù hợp cho người dùng.
- Có thể tương tác với hệ thống bên ngoài, API hoặc cơ sở dữ liệu.

❑ Examples of Boundary Classes (Ví dụ về Lớp Biên):

1. User Interface Components (Thành phần giao diện người dùng): Forms, buttons, menus in web applications.
2. API Handlers (Bộ xử lý API): Classes that process API requests and responses.
3. Data Input Validators (Bộ kiểm tra đầu vào dữ liệu): Classes that check and format user input before sending it to the system.

❑ Ghi chú:

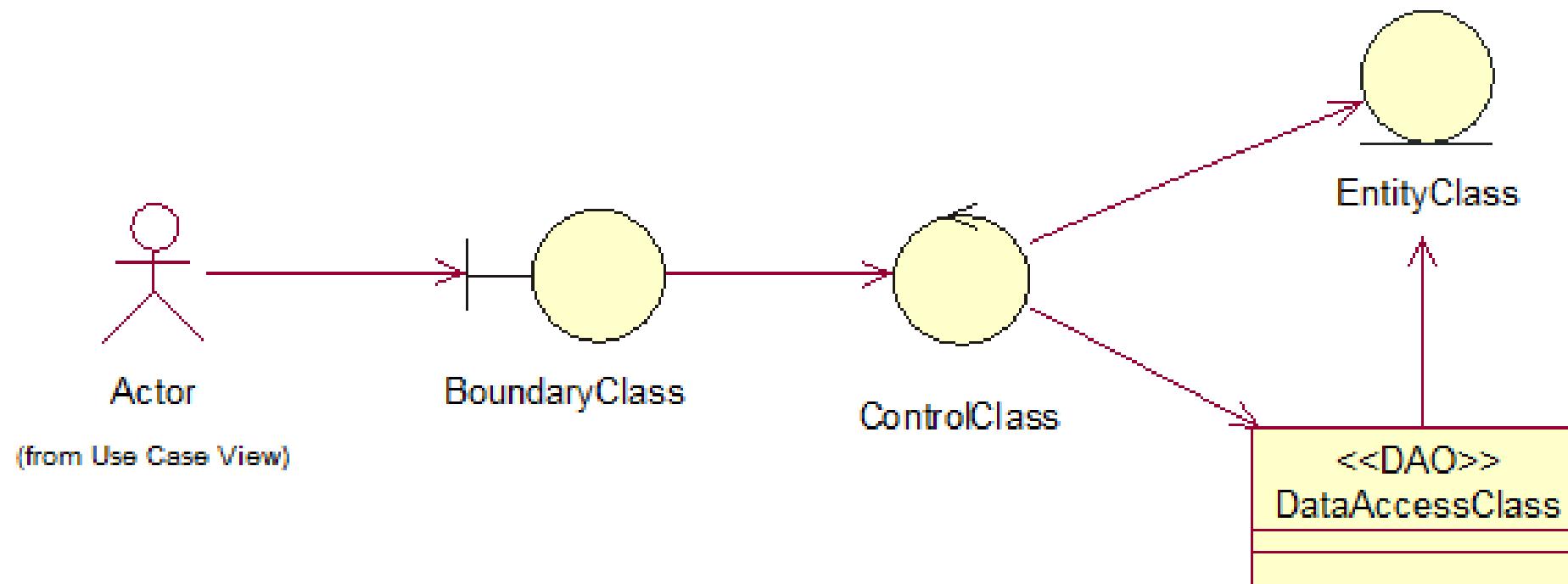
- Lớp Biên đóng vai trò quan trọng trong việc đảm bảo sự **giao tiếp mượt mà** giữa người dùng và hệ thống.



SWE | 6.1.3. LỚP BIÊN (BOUNDARY CLASS)

❑ Diagram Representation (Biểu diễn sơ đồ):

1. Actor (Tác nhân - Người dùng) interacts with BoundaryClass (UI).
Người dùng tương tác với BoundaryClass (Giao diện người dùng).
2. BoundaryClass passes data to ControlClass for processing.
BoundaryClass chuyển dữ liệu đến ControlClass để xử lý.
3. ControlClass interacts with EntityClass to retrieve or store data.
ControlClass tương tác với EntityClass để truy xuất hoặc lưu trữ dữ liệu.



HOMEWORK PRACTICES

❑ Example of Boundary Class (Ví dụ về Lớp Biên)

- Online Shopping System (Hệ thống mua sắm trực tuyến)
- Scenario: A customer wants to buy a product online.

❑ Coding with Java IDE for these classes below:

1. Boundary Class (Lớp Biên): ProductPageUI

- Displays product details, price, and "Add to Cart" button.
- Accepts user input (e.g., selecting quantity).
- Sends user actions to the control class.

2. Control Class (Lớp Điều Khiển): CartController

- Processes the request to add a product to the cart.
- Calls ProductInventory (Entity Class) to check stock availability.

3. Entity Class (Lớp Thực Thể): ProductInventory

- Stores product details and stock availability.
- Updates stock when an item is added to the cart.

1. Boundary Class (Lớp Biên): ProductPageUI

Example Code
in Java

java

 Copy  Edit

```
// Boundary Class – ProductPageUI
public class ProductPageUI {
    private CartController cartController;

    public ProductPageUI(CartController cartController) {
        this.cartController = cartController;
    }

    public void displayProduct(String productName, double price) {
        System.out.println("Product: " + productName + " - Price: $" + price);
    }

    public void addToCart(String productId, int quantity) {
        cartController.addProductToCart(productId, quantity);
    }
}
```

2. CONTROL CLASS (LỚP ĐIỀU KHIỂN): CARTCONTROLLER

Control Class -
CartController

java

Copy Edit

```
// Control Class - CartController
public class CartController {
    private ProductInventory inventory;

    public CartController(ProductInventory inventory) {
        this.inventory = inventory;
    }

    public void addProductToCart(String productId, int quantity) {
        if (inventory.isProductAvailable(productId, quantity)) {
            System.out.println("Product added to cart.");
        } else {
            System.out.println("Out of stock!");
        }
    }
}
```

3. ENTITY CLASS (LỚP THỰC THỂ): PRODUCTINVENTORY

Entity Class -
ProductInventory

java

Copy Edit

```
// Entity Class – ProductInventory
import java.util.HashMap;
import java.util.Map;

public class ProductInventory {
    private Map<String, Integer> stock = new HashMap<>();

    public void addStock(String productId, int quantity) {
        stock.put(productId, quantity);
    }

    public boolean isProductAvailable(String productId, int quantity) {
        return stock.getOrDefault(productId, 0) >= quantity;
    }
}
```

□ Key Takeaways (Điểm quan trọng cần nhớ)

1. Boundary Class interacts with users (UI components like forms, buttons, and input fields).

- **Lớp Biên** tương tác với người dùng (các thành phần giao diện như biểu mẫu, nút bấm, ô nhập liệu).

2. Control Class processes logic and connects the boundary class with entity classes.

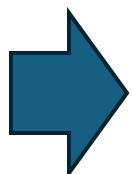
- **Lớp Điều Khiển** xử lý logic và kết nối lớp biên với lớp thực thể.

3. Entity Class stores and manages data, interacting with the database if necessary.

- **Lớp Thực Thể** lưu trữ và quản lý dữ liệu, tương tác với cơ sở dữ liệu nếu cần.

□ **Ghi chú:**

Các ví dụ trên thể hiện cách áp dụng mô hình BCE trong các ứng dụng thực tế để đảm bảo phân tách trách nhiệm và dễ bảo trì.



- ❑ **Mô hình Model-View-Controller (MVC)** là một mẫu thiết kế phần mềm giúp tách một ứng dụng thành ba thành phần liên kết với nhau:

1. Model (Dữ liệu & Logic nghiệp vụ)

- Quản lý dữ liệu và logic nghiệp vụ của ứng dụng.
- Lưu trữ, xử lý và cập nhật dữ liệu.
- Gửi thông báo cập nhật đến View khi dữ liệu thay đổi.

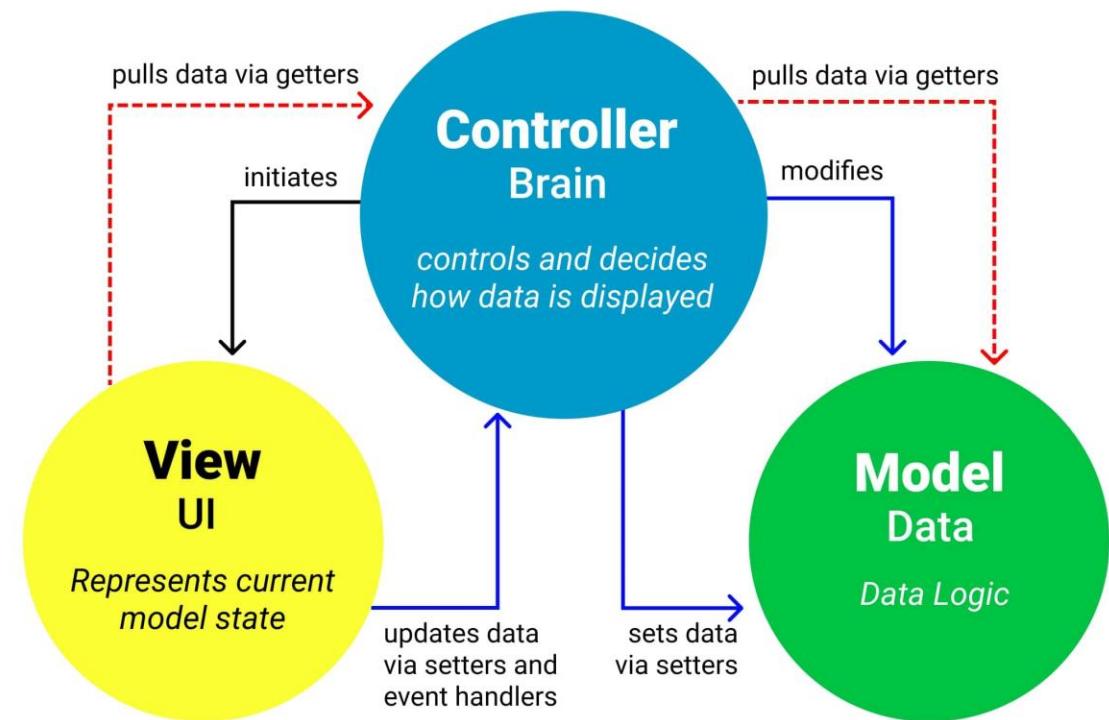
2. View (Giao diện người dùng - UI)

- Đại diện cho giao diện của người dùng.
- Hiển thị dữ liệu từ Model.
- Cập nhật khi Model thay đổi.

3. Controller (Bộ não xử lý)

- Đóng vai trò trung gian giữa Model và View.
- Xử lý đầu vào của người dùng và cập nhật Model.
- Quyết định cách dữ liệu được hiển thị trên View.

MVC Architecture Pattern



❑ **Mối quan hệ giữa các lớp trong MVC:**

- View lấy dữ liệu từ Model thông qua **các phương thức getter**.
- Controller thay đổi Model khi có sự kiện từ người dùng.
- Model cập nhật View khi dữ liệu thay đổi.
- Controller cập nhật View thông qua các trình xử lý sự kiện và setter.

❑ **Ưu điểm của mô hình MVC:**

- **Phân tách rõ ràng các thành phần:** Mỗi phần có một trách nhiệm riêng, giúp dễ dàng bảo trì.
- **Mở rộng linh hoạt:** Cho phép thay đổi một phần của ứng dụng mà không ảnh hưởng đến phần còn lại.
- **Tái sử dụng mã nguồn:** Các thành phần có thể được tái sử dụng trong các phần khác của ứng dụng.



VÍ DỤ MINH HỌA VỀ MÔ HÌNH MVC (MODEL - VIEW - CONTROLLER)

3/10/2025 9:17 AM

❑ Ví dụ

1. Minh họa về mô hình MVC (Model - View - Controller)

Tình huống: Ứng dụng Quản lý Sinh Viên

Giả sử chúng ta có một ứng dụng web để quản lý thông tin sinh viên, bao gồm các thao tác như thêm, sửa, xóa và xem danh sách sinh viên.

example



The screenshot shows a code editor window with the following Python code:

```
python
.
.
.
class Student:
    def __init__(self, student_id, name, age):
        self.student_id = student_id
        self.name = name
        self.age = age

    def get_info(self):
        return f"ID: {self.student_id}, Name: {self.name}, Age: {self.age}"
```

The code defines a `Student` class with an `__init__` method that initializes `student_id`, `name`, and `age`. It also defines a `get_info` method that returns a formatted string containing the student's ID, name, and age.

2. View (Giao diện người dùng - UI)

View hiển thị thông tin sinh viên lên giao diện người dùng.

Ví dụ: HTML để hiển thị danh sách sinh viên

example

```
html
<!DOCTYPE html>
<html>
<head>
    <title>Danh sách sinh viên</title>
</head>
<body>
    <h1>Danh sách Sinh Viên</h1>
    <table>
        <tr>
            <th>ID</th>
            <th>Họ và Tên</th>
            <th>Tuổi</th>
        </tr>
        {% for student in students %}
        <tr>
            <td>{{ student.student_id }}</td>
            <td>{{ student.name }}</td>
            <td>{{ student.age }}</td>
        </tr>
        {% endfor %}
    </table>
</body>
</html>
```

3. View (Giao diện người dùng - UI)

View hiển thị thông tin sinh viên lên giao diện người dùng.

Ví dụ: HTML để hiển thị danh sách sinh viên

example

python

Copy Edit

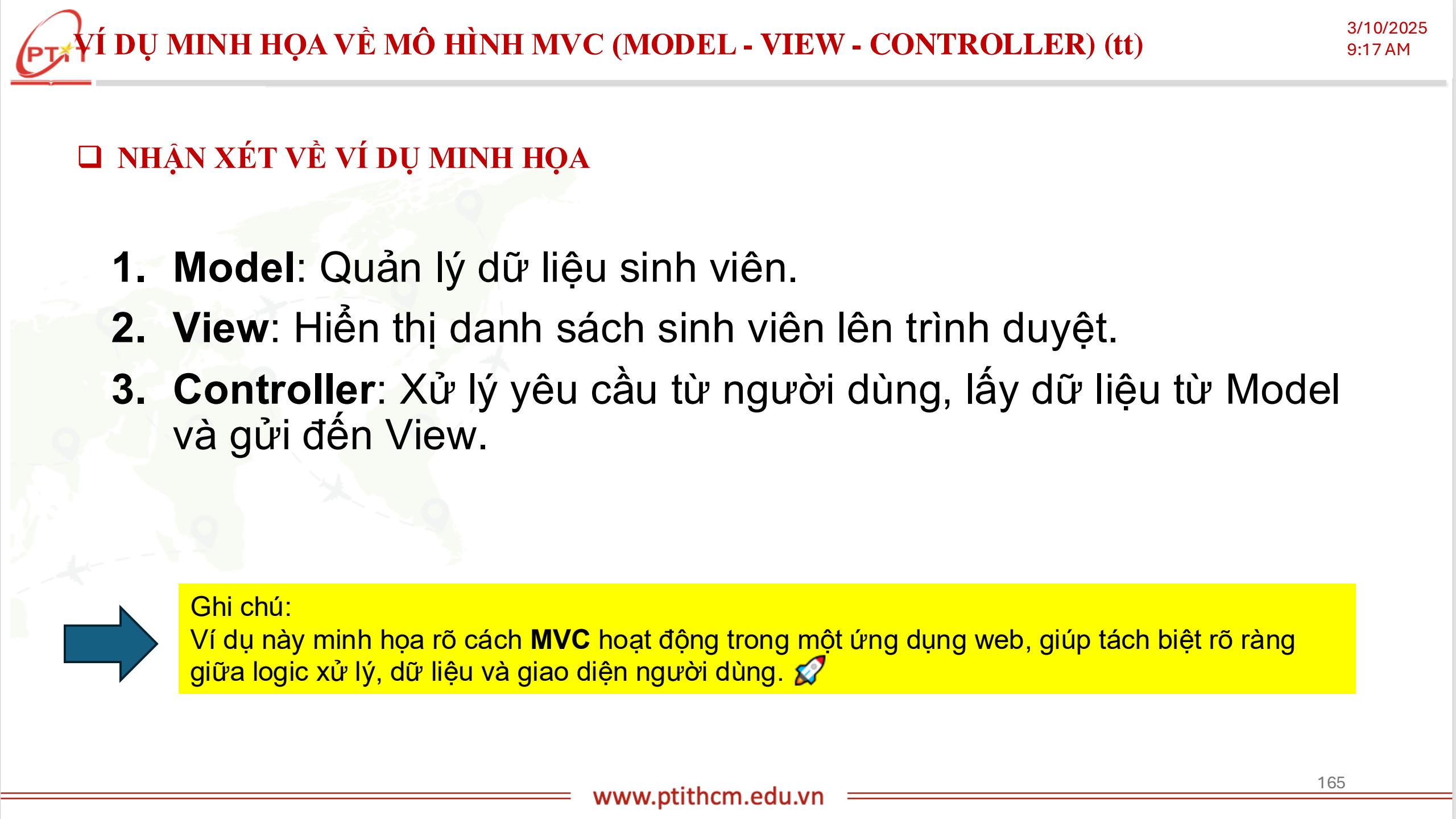
```
from flask import Flask, render_template
from model import Student

app = Flask(__name__)

# Danh sách sinh viên giả lập
students = [
    Student(1, "Nguyễn Văn A", 20),
    Student(2, "Trần Thị B", 21),
    Student(3, "Lê Văn C", 22),
]

@app.route('/students')
def list_students():
    return render_template('students.html', students=students)

if __name__ == '__main__':
    app.run(debug=True)
```



VÍ DỤ MINH HỌA VỀ MÔ HÌNH MVC (MODEL - VIEW - CONTROLLER) (tt)

□ NHẬN XÉT VỀ VÍ DỤ MINH HỌA

- Model:** Quản lý dữ liệu sinh viên.
- View:** Hiển thị danh sách sinh viên lên trình duyệt.
- Controller:** Xử lý yêu cầu từ người dùng, lấy dữ liệu từ Model và gửi đến View.

Ghi chú:
Ví dụ này minh họa rõ cách **MVC** hoạt động trong một ứng dụng web, giúp tách biệt rõ ràng giữa logic xử lý, dữ liệu và giao diện người dùng. 

1. Định nghĩa Scenario

- Scenario là **tập hợp các bước mô tả chi tiết về cách người dùng tương tác với hệ thống** để hoàn thành một nhiệm vụ cụ thể. Mỗi scenario thể hiện một tình huống sử dụng phần mềm theo góc nhìn của người dùng.

2. Mục đích của Scenario

- Giúp hiểu rõ các yêu cầu của hệ thống.
- Mô tả cụ thể các luồng tương tác giữa người dùng và hệ thống.
- Hỗ trợ phát triển **use case** bằng cách cung cấp bối cảnh chi tiết hơn.
- Làm cơ sở để kiểm thử phần mềm, đảm bảo hệ thống hoạt động đúng theo yêu cầu.

3. Cấu trúc của một Scenario

- Một scenario thường bao gồm các thành phần sau:
 - Tên Scenario:** Mô tả ngắn gọn mục tiêu của kịch bản.
 - Mô tả ngữ cảnh:** Cung cấp thông tin về người dùng, tình huống xảy ra.
 - Các bước thực hiện:** Trình bày tuần tự các hành động mà người dùng thực hiện.
 - Kết quả mong đợi:** Kết quả cuối cùng sau khi scenario kết thúc thành công.
 - Scenario ngoại lệ (nếu có):** Các tình huống phát sinh lỗi hoặc sai sót trong quá trình thực hiện.



➡️ **Ghi nhớ:**

- Khi viết Scenario, cần bám sát thực tế, đảm bảo rõ ràng, dễ hiểu và bao quát được các trường hợp ngoại lệ để hệ thống có thể hoạt động tốt nhất

4. VÍ DỤ VỀ SCENARIO

- Scenario:** Đăng nhập vào hệ thống quản lý sinh viên
- Ngữ cảnh:** Người dùng là sinh viên cần đăng nhập vào hệ thống để xem điểm.
- Các bước thực hiện:**
 - Sinh viên truy cập trang đăng nhập.
 - Nhập tên đăng nhập và mật khẩu.
 - Nhấn nút "Đăng nhập".
 - Hệ thống kiểm tra thông tin và hiển thị màn hình chính nếu hợp lệ.
- Kết quả mong đợi:** Sinh viên đăng nhập thành công và có thể sử dụng hệ thống.
- Scenario ngoại lệ:**
 - Nếu sai mật khẩu, hệ thống hiển thị thông báo lỗi.
 - Nếu tài khoản bị khóa, hệ thống yêu cầu liên hệ quản trị viên.

5. PHÂN BIỆT SCENARIO VÀ USE CASE

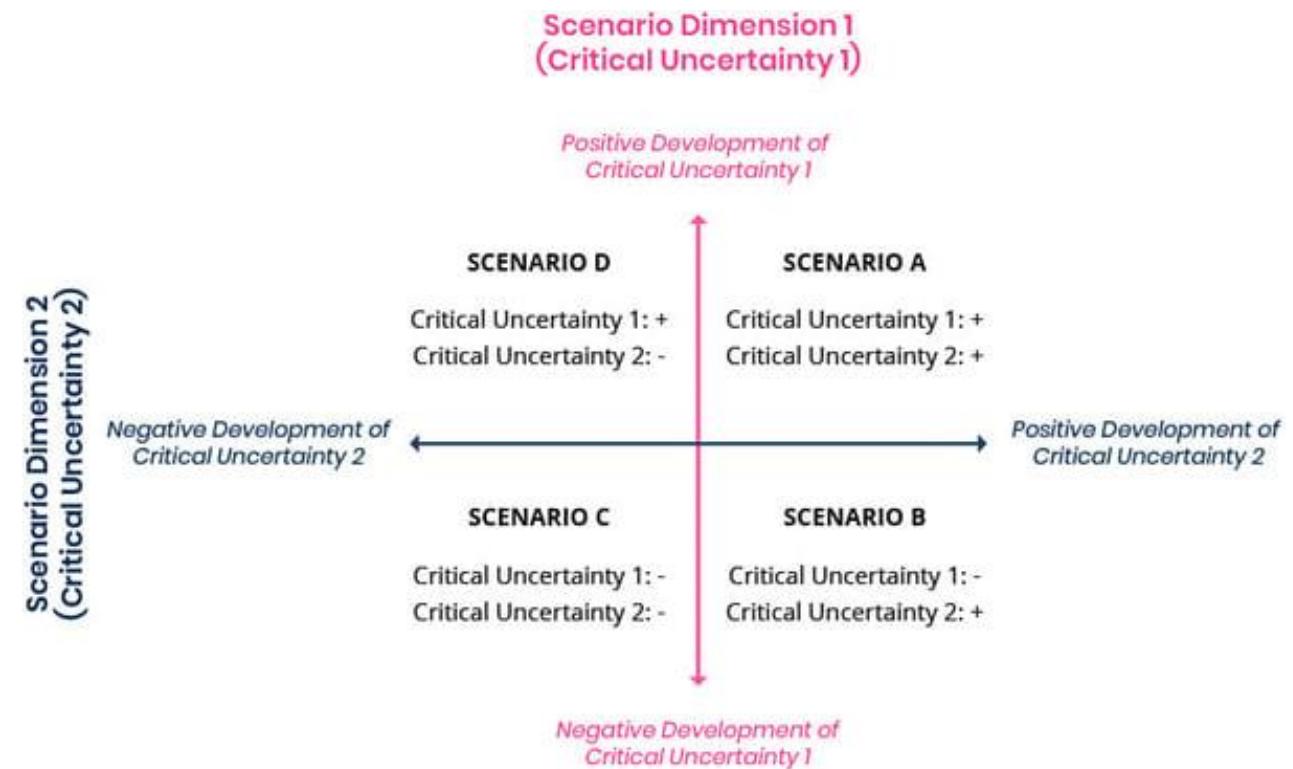
Tiêu chí	Scenario	Use Case
Mục đích	Mô tả chi tiết một tình huống cụ thể	Mô tả chung một chức năng của hệ thống
Mức độ chi tiết	Rất cụ thể, gồm từng bước thao tác	Tổng quát hơn, tập trung vào mục tiêu tổng thể
Phạm vi	Một instance cụ thể của Use Case	Bao gồm nhiều Scenario khác nhau

1. Định nghĩa

- Scenario Chuẩn (Normal Scenario):** Là chuỗi các bước mô tả luồng hoạt động chính của một Use Case, tức là khi mọi thứ diễn ra như mong đợi, không có lỗi hoặc tình huống bất thường.
- Scenario Ngoại Lệ (Exception Scenario):** Mô tả các trường hợp xảy ra lỗi hoặc điều kiện bất thường trong quá trình thực hiện Use Case, giúp hệ thống xử lý tốt hơn khi gặp lỗi.

2. Tại sao cần viết cả Scenario Chuẩn và Ngoại Lệ?

- Đảm bảo hệ thống hoạt động đúng trong điều kiện bình thường.
- Giúp phát hiện các tình huống bất thường và xử lý lỗi để nâng cao độ tin cậy của phần mềm.
- Hỗ trợ kiểm thử phần mềm, đảm bảo hệ thống có thể xử lý mọi tình huống có thể xảy ra.



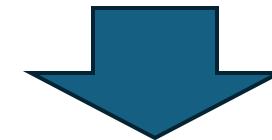
3. Ví dụ: Use Case "Đăng nhập vào hệ thống"

Scenario Chuẩn (Normal Scenario)

- **Ngữ cảnh:** Người dùng cần đăng nhập vào hệ thống để sử dụng các tính năng.
- **Các bước thực hiện:**
 1. Người dùng mở trang đăng nhập.
 2. Nhập tên đăng nhập và mật khẩu hợp lệ.
 3. Nhấn nút "Đăng nhập".
 4. Hệ thống kiểm tra thông tin và xác thực thành công.
 5. Hệ thống chuyển hướng đến trang chủ.
- **Kết quả mong đợi:** Người dùng đăng nhập thành công và có thể sử dụng hệ thống.

Scenario Ngoại Lệ (Exception Scenario)

- **Trường hợp 1: Nhập sai mật khẩu**
 - Người dùng nhập tên đăng nhập hợp lệ nhưng sai mật khẩu.
 - Hệ thống hiển thị thông báo "Sai mật khẩu, vui lòng thử lại".
- **Trường hợp 2: Tài khoản bị khóa**
 - Người dùng nhập sai mật khẩu quá 5 lần.
 - Hệ thống khóa tài khoản và yêu cầu liên hệ quản trị viên.
- **Trường hợp 3: Tên đăng nhập không tồn tại**
 - Người dùng nhập một tên đăng nhập không có trong hệ thống.
 - Hệ thống hiển thị thông báo "Tài khoản không tồn tại".



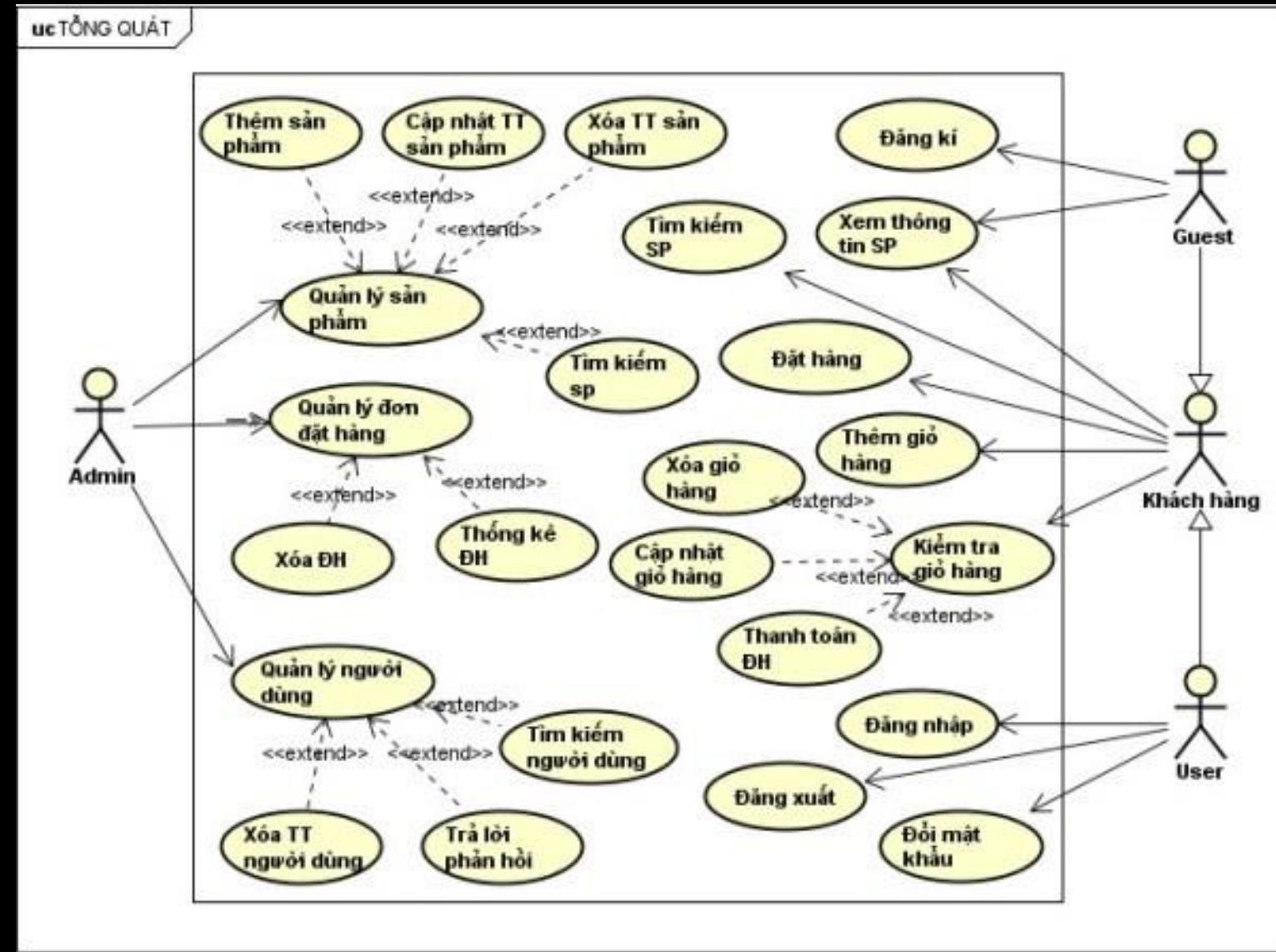
Ghi nhớ :

- **Scenario Chuẩn** đảm bảo hệ thống hoạt động trơn tru khi mọi thứ diễn ra đúng như dự kiến.
- **Scenario Ngoại Lệ** giúp hệ thống xử lý tình huống lỗi, đảm bảo trải nghiệm người dùng và bảo mật.
- Khi viết Use Case, cần xem xét đầy đủ cả hai loại Scenario để phần mềm **vận hành ổn định và đáng tin cậy**.

VÍ DỤ VỀ SCENARIO CHUẨN VÀ SCENARIO NGOẠI LỆ

❑ Use Case: Đặt hàng trên website thương mại điện tử

- **Mô tả:** Người dùng thực hiện đặt hàng trên một trang thương mại điện tử.
- **Mục tiêu:** Xác nhận đơn hàng thành công và tiến hành thanh toán.





Scenario Chuẩn (Normal Scenario)

Đặt hàng thành công.

❑ **Bối cảnh:** Người dùng có tài khoản hợp lệ và đặt hàng một sản phẩm có sẵn trong kho.

❑ **Các bước thực hiện:**

1. Người dùng đăng nhập vào hệ thống.
2. Chọn sản phẩm cần mua và nhấn "Thêm vào giỏ hàng".
3. Nhấn vào giỏ hàng, kiểm tra thông tin đơn hàng.
4. Chọn phương thức thanh toán và địa chỉ giao hàng.
5. Nhấn nút "Xác nhận đặt hàng".
6. Hệ thống kiểm tra tồn kho và xác nhận đơn hàng hợp lệ.
7. Hệ thống hiển thị thông báo "Đặt hàng thành công" và gửi email xác nhận.

🎯 **Kết quả mong đợi:**

- Đơn hàng được lưu vào hệ thống, người dùng nhận được thông tin xác nhận qua email.

Scenario Ngoại Lệ (Exception Scenario) Lỗi do sản phẩm hết hàng

❑ **Bối cảnh:** Người dùng đặt hàng nhưng sản phẩm đã hết hàng trong kho.

❑ **Các bước thực hiện:**

1. Người dùng đăng nhập vào hệ thống.
2. Chọn sản phẩm cần mua và nhấn "Thêm vào giỏ hàng".
3. Nhấn vào giỏ hàng, kiểm tra thông tin đơn hàng.
4. Chọn phương thức thanh toán và địa chỉ giao hàng.
5. Nhấn nút "Xác nhận đặt hàng".
6. Hệ thống kiểm tra tồn kho và phát hiện sản phẩm đã hết hàng.
7. Hệ thống hiển thị thông báo: ! "Sản phẩm bạn chọn đã hết hàng. Vui lòng chọn sản phẩm khác hoặc chờ cập nhật hàng".

⚠ **Kết quả mong đợi:**

- Người dùng không thể đặt hàng với sản phẩm hết hàng, hệ thống thông báo rõ ràng để họ có thể chọn phương án khác.

6.3.1. TRÍCH CÁC LỚP BIÊN

6.3.2. ĐỀ XUẤT CÁC LỚP ĐIỀU KHIỂN

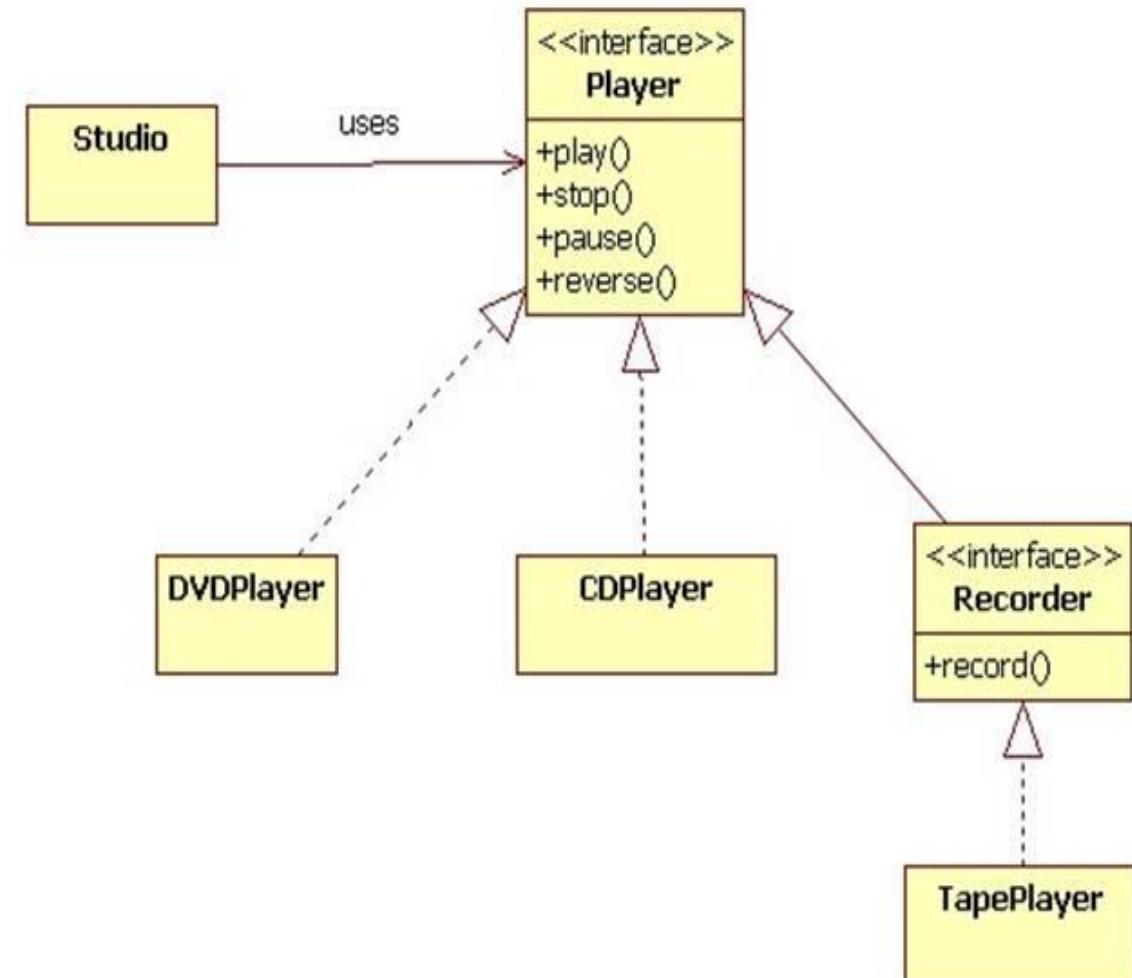
6.3.3. ĐỀ XUẤT CÁC LỚP BIÊN

1. Định nghĩa

- Trích các lớp biên (Interface Classes) là bước quan trọng trong phân tích và thiết kế hệ thống, trong đó các lớp biên được xác định để quản lý giao tiếp giữa hệ thống và người dùng hoặc các hệ thống bên ngoài.

2. Vai trò của lớp biên

- Giao diện người dùng:** Xử lý tương tác giữa người dùng và hệ thống.
- Giao tiếp với hệ thống bên ngoài:** Nhận dữ liệu từ API, hệ thống khác.
- Kiểm tra và xử lý đầu vào:** Đảm bảo dữ liệu nhập vào hệ thống hợp lệ.





3. Xác định lớp biên trong sơ đồ UML

- Trong sơ đồ UML trên, lớp **biên (interface class)** có thể được xác định như sau:
- Lớp Player (<<interface>> Player)**
 - Đây là **lớp giao diện (interface class)** cung cấp các phương thức cơ bản để điều khiển thiết bị phát nhạc (DVDPlayer, CDPlayer).
 - Các phương thức bao gồm:
 - play(): Phát nhạc.
 - stop(): Dừng phát.
 - pause(): Tạm dừng.
 - reverse(): Tua ngược.
 - Vai trò:** Làm trung gian giữa hệ thống phát nhạc và các thiết bị phát nhạc cụ thể.
- Lớp Recorder (<<interface>> Recorder)**
 - Đây cũng là một **lớp giao diện (interface class)** nhưng chuyên về chức năng ghi âm.
 - Cung cấp phương thức record() để ghi lại âm thanh.
 - Vai trò:** Tạo chuẩn chung để các thiết bị ghi âm (ví dụ TapePlayer) triển khai.

4. Vai trò của các lớp biên trong sơ đồ

- Player** và **Recorder** đảm bảo các thiết bị phát và ghi âm tuân theo một giao diện nhất định.
- Các lớp như **DVDPlayer**, **CDPlayer**, **TapePlayer** kế thừa từ các lớp giao diện này và triển khai các chức năng cụ thể.
- Điều này giúp hệ thống dễ dàng mở rộng và bảo trì khi cần bổ sung các thiết bị mới.

Ghi nhớ:

- Lớp biên trong sơ đồ này chính là các Interface Classes (Player, Recorder).
- Chúng quy định cách các thành phần giao tiếp với nhau, giúp hệ thống có kiến trúc linh hoạt và dễ mở rộng
- Nhờ có Interface Classes, các thiết bị phát nhạc và ghi âm có thể triển khai các chức năng chung một cách thống nhất.

1. Định nghĩa lớp thực thể (Entity Classes)

Lớp thực thể (Entity Classes) là các lớp biểu diễn dữ liệu hoặc trạng thái trong hệ thống. Chúng chịu trách nhiệm quản lý thông tin và có thể được lưu trữ trong cơ sở dữ liệu.

2. Xác định các lớp thực thể trong sơ đồ UML

Trong sơ đồ trên, các lớp thực thể có thể được xác định như sau:

- **DVDPlayer**

- Là một thực thể đại diện cho đầu phát DVD.
- Thực hiện các phương thức từ giao diện Player như play(), stop(), pause(), reverse().
- **Vai trò:** Quản lý chức năng phát nhạc/video từ đĩa DVD.

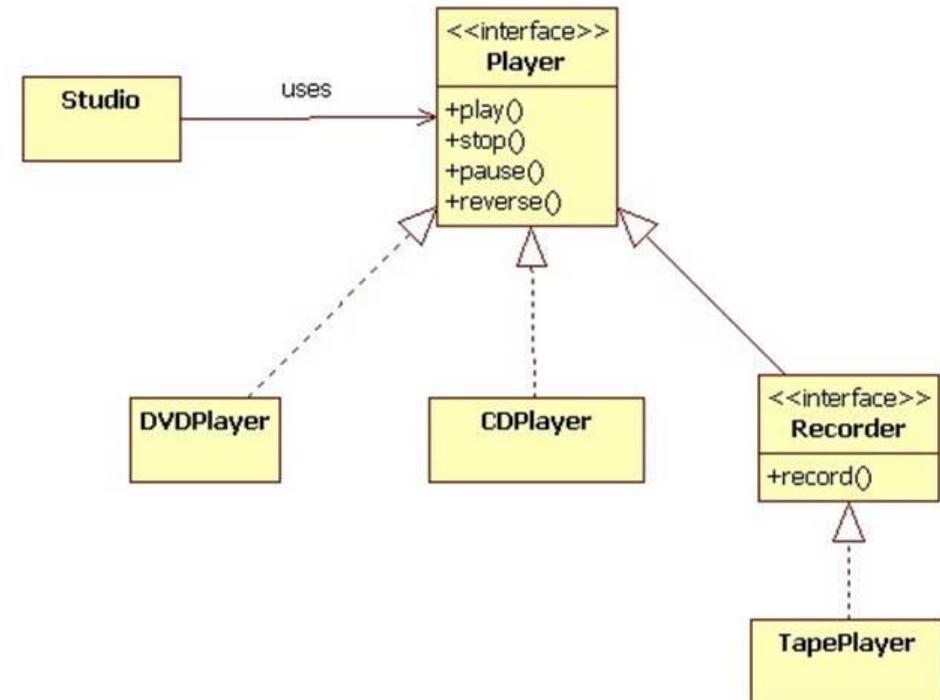
- **CDPlayer**

- Là một thực thể đại diện cho đầu phát CD.
- Cũng kế thừa từ giao diện Player, thực hiện các chức năng tương tự DVDPlayer.
- **Vai trò:** Phát nhạc từ đĩa CD.

- **TapePlayer**

- Là một thực thể thực thi giao diện Recorder, có phương thức record().

Vai trò: Ghi âm từ băng cassette.



Ghi chú:

- Các lớp thực thể từ sơ đồ UML trên là: DVDPlayer, CDPlayer và TapePlayer.
- Chúng kế thừa từ các lớp giao diện (Player, Recorder) để triển khai chức năng cụ thể trong hệ thống.
- Các lớp thực thể chịu trách nhiệm chính trong việc lưu trữ và quản lý trạng thái của hệ thống.

Đề xuất bài tập thực hành thiết kế sơ đồ UML

Chủ đề: Hệ thống quản lý cửa hàng bán lẻ

1. Mô tả bài toán

Một cửa hàng bán lẻ trực tuyến cần một hệ thống phần mềm để quản lý bán hàng. Hệ thống sẽ bao gồm ba lớp giao diện (Boundary Classes) để tương tác với người dùng và năm lớp thực thể (Entity Classes) để quản lý dữ liệu.

2. Xác định các lớp trong hệ thống

◆ Lớp biên (Interface Classes)

1. CustomerInterface

1. Cho phép khách hàng đăng ký, đăng nhập, xem thông tin sản phẩm và đặt hàng.
2. Các chức năng chính:
 - register()
 - login()
 - viewProducts()
 - placeOrder()

2. AdminInterface

1. Cho phép quản trị viên quản lý danh sách sản phẩm, đơn hàng và khách hàng.
2. Các chức năng chính:
 - addProduct()
 - updateProduct()
 - deleteProduct()
 - viewOrders()

3. PaymentInterface

1. Cho phép khách hàng thanh toán đơn hàng qua nhiều phương thức khác nhau.
2. Các chức năng chính:
 - selectPaymentMethod()
 - processPayment()
 - confirmTransaction()

◆ Lớp thực thể (Entity Classes)

1. Customer

- Chứa thông tin khách hàng như tên, địa chỉ, số điện thoại.
- Thuộc tính:
 - customerID
 - name
 - address
 - phoneNumber

2. Product

- Chứa thông tin sản phẩm được bán trên hệ thống.
- Thuộc tính:
 - productID
 - name
 - price
 - stockQuantity

3. Order

- Đại diện cho đơn hàng mà khách hàng đã đặt.
- Thuộc tính:
 - orderID
 - customerID
 - orderDate
 - totalAmount

4. Order

- Đại diện cho đơn hàng mà khách hàng đã đặt.
- Thuộc tính:
 - orderID
 - customerID
 - orderDate
 - totalAmount

5. Payment

- Quản lý thông tin thanh toán của đơn hàng.
- Thuộc tính:
 - paymentID
 - orderID
 - paymentMethod
 - paymentStatus

6. Admin

- Đại diện cho người quản trị cửa hàng, có quyền quản lý sản phẩm và đơn hàng.
- Thuộc tính:
 - adminID
 - username
 - password

3. YÊU CẦU BÀI TẬP UML

1. Vẽ sơ đồ lớp UML thể hiện mối quan hệ giữa các lớp trên.

2. Xác định quan hệ giữa các lớp:

- Lớp biên (**Interface Classes**) giao tiếp với lớp thực thể (**Entity Classes**).
- Lớp thực thể có thể có quan hệ **kế thừa**, **kết hợp** hoặc **liên kết** với nhau.

3. Thêm các phương thức và thuộc tính cần thiết để mô hình hóa hệ thống một cách đầy đủ.

📌 HƯỚNG DẪN :

- Sử dụng **quan hệ kết hợp** giữa Order và Customer.
- Sử dụng **quan hệ kế thừa** nếu cần mở rộng các lớp thực thể.
- PaymentInterface có thể tương tác với Payment để xử lý thanh toán.

6.4. Xây dựng sơ đồ lớp

6.4.1. Khái niệm sơ đồ lớp

6.4.2. Xây dựng sơ đồ quan hệ giữa các lớp

1. Định nghĩa sơ đồ lớp (Class Diagram Definition)

- Sơ đồ lớp (Class Diagram) là một loại sơ đồ trong UML (Unified Modeling Language) dùng để mô tả cấu trúc tĩnh của hệ thống phần mềm bằng cách biểu diễn các lớp, thuộc tính, phương thức và mối quan hệ giữa các lớp.

2. Vai trò của sơ đồ lớp (Role of Class Diagram)

- Xác định các thành phần chính của hệ thống và cách chúng tương tác.
- Giúp các nhà phát triển hiểu rõ cấu trúc hệ thống và lập kế hoạch triển khai.
- Là tài liệu tham khảo quan trọng trong thiết kế và lập trình phần mềm.
- Hỗ trợ kiểm tra và bảo trì hệ thống dễ dàng hơn.

3. Thành phần của sơ đồ lớp (Components of Class Diagram)

- Lớp (Class):** Đại diện cho một thực thể trong hệ thống, gồm tên lớp, thuộc tính (attributes), và phương thức (methods).
- Thuộc tính (Attributes):** Định nghĩa trạng thái hoặc dữ liệu của lớp.
- Phương thức (Methods):** Xác định hành vi hoặc chức năng của lớp.
- Quan hệ giữa các lớp (Relationships between Classes):** Bao gồm các loại quan hệ như:
 - Association (Quan hệ liên kết)
 - Aggregation (Quan hệ tập hợp)
 - Composition (Quan hệ thành phần)
 - Inheritance (Quan hệ kế thừa)
 - Dependency (Quan hệ phụ thuộc)



Kết luận

- Sơ đồ lớp giúp mô tả trực quan các thành phần trong hệ thống phần mềm, tạo cơ sở để thiết kế và lập trình một cách có hệ thống và hiệu quả.

4. Ví dụ minh họa (Example Illustration)

- Một sơ đồ lớp cho hệ thống quản lý sinh viên có thể gồm các lớp như:
- SinhVien (Student)
- MonHoc (Course)
- GiaoVien (Teacher)
- DangKyHoc (Enrollment)

Các lớp này có thể có mối quan hệ với nhau, chẳng hạn:

- Một sinh viên có thể đăng ký nhiều môn học (Association).
- Một giáo viên có thể giảng dạy nhiều môn học (Aggregation).
- Một môn học có thể có nhiều tài liệu tham khảo, nhưng tài liệu này không thể tồn tại độc lập nếu không có môn học đó (Composition).

Kết luận

- Sơ đồ lớp giúp mô tả trực quan các thành phần trong hệ thống phần mềm, tạo cơ sở để thiết kế và lập trình một cách có hệ thống và hiệu quả.

1. Định nghĩa quan hệ giữa các lớp (Definition of Class Relationships)

- Trong sơ đồ lớp của UML, các lớp không tồn tại độc lập mà có mối quan hệ với nhau để phản ánh logic nghiệp vụ và cấu trúc phần mềm. Việc xác định đúng quan hệ giữa các lớp giúp hệ thống dễ hiểu, bảo trì và mở rộng hơn.

2. Các loại quan hệ giữa các lớp (Types of Relationships Between Classes)

1. Quan hệ liên kết (Association)

- Là mối quan hệ giữa hai hoặc nhiều lớp có liên kết với nhau nhưng vẫn độc lập.
- Ví dụ: Một lớp SinhVien có thể liên kết với lớp MonHoc thông qua quan hệ "đăng ký học".

2. Quan hệ kế thừa (Generalization/Inheritance)

- Một lớp con kế thừa các thuộc tính và phương thức của lớp cha.
- Ví dụ: Lớp NhanVien kế thừa từ lớp Nguoi.

3. Quan hệ phụ thuộc (Dependency)

- Một lớp thay đổi có thể ảnh hưởng đến một lớp khác nhưng không có liên kết cố định.
- Ví dụ: Lớp HoaDon phụ thuộc vào lớp SanPham, nếu sản phẩm thay đổi, hóa đơn có thể bị ảnh hưởng.

4. Quan hệ tập hợp (Aggregation)

- Một lớp chứa một hoặc nhiều thực thể của lớp khác, nhưng các thực thể này vẫn có thể tồn tại độc lập.
- Ví dụ: Lớp LopHoc có một danh sách SinhVien, nhưng sinh viên vẫn có thể tồn tại ngoài lớp học.

5. Quan hệ thành phần (Composition)

- Một lớp chứa một hoặc nhiều thực thể của lớp khác, nhưng các thực thể này không thể tồn tại độc lập.
- Ví dụ: Lớp XeHoi có DongCo, nếu XeHoi bị xóa thì DongCo cũng bị xóa theo.

Class Diagram Relationship Type	Notation
Association	
Inheritance	
Realization/ Implementation	
Dependency	
Aggregation	
Composition	

3. Cách thể hiện quan hệ trong sơ đồ UML (How to Represent Relationships in UML)

- Dùng đường thẳng có mũi tên để chỉ quan hệ kế thừa.
- Dùng đường thẳng đơn giản để chỉ quan hệ liên kết.
- Dùng hình thoi rỗng để biểu diễn quan hệ tập hợp (Aggregation).
- Dùng hình thoi đặc để biểu diễn quan hệ thành phần (Composition).
- Dùng đường đứt nét để thể hiện quan hệ phụ thuộc.

4. Ví dụ minh họa (Example)

- Xây dựng sơ đồ quan hệ giữa các lớp trong hệ thống quản lý thư viện:
- Lớp **DocGia** liên kết với Sach qua quan hệ "**mượn sách**".
- Lớp **ThuThu** kế thừa từ lớp **NhanVien**.
- Lớp **Sach** có quan hệ thành phần với **TacGia** vì mỗi cuốn sách phải có tác giả.

Ghi nhớ:

- Xác định đúng mối quan hệ giữa các lớp giúp thiết kế phần mềm rõ ràng, dễ bảo trì và mở rộng, đảm bảo sự liên kết giữa các thành phần trong hệ thống.

6.5. Xây dựng sơ đồ tuần tự, cộng tác

6.5.1. Khái niệm sơ đồ tuần tự, cộng tác

6.5.2. Viết lại scenario phiên bản 2 cho các use case

6.5.3. Vẽ sơ đồ tuần tự, cộng tác tương ứng với các scenario

1. Lược đồ lớp (Class Diagram)

- Hiển thị **các lớp (classes)**, **thuộc tính (attributes)**, **phương thức (methods)**, và **mối quan hệ giữa các lớp** (kế thừa, kết hợp, phụ thuộc, liên kết,...).

2. Lược đồ đối tượng (Object Diagram)

- Dùng để kiểm tra cách các **đối tượng tương tác với nhau** trong quá trình thực thi.

3. Lược đồ tuần tự (Sequence Diagram)

- Mô tả cách **các đối tượng** trong hệ thống tương tác với nhau theo **thứ tự thời gian**.

4. Lược đồ cộng tác (Collaboration Diagram) (Communication Diagram)

- Cũng mô tả tương tác giữa các đối tượng như **lược đồ tuần tự**, nhưng tập trung vào **mối quan hệ giữa các đối tượng** thay vì trình tự thời gian.

5. Lược đồ ca sử dụng (Use Case Diagram)

- Thể hiện các **chức năng (use case)** của hệ thống thông qua **người dùng (actors)** và **mối quan hệ giữa chúng**.

6. Lược đồ trạng thái (State Diagram)

- Mô tả **các trạng thái khác nhau** của một đối tượng và **cách nó thay đổi trạng thái** dựa trên các sự kiện.

7. Lược đồ hoạt động (Activity Diagram)

- Biểu diễn luồng công việc (workflow) hoặc **luồng điều khiển (control flow)** của hệ thống.

8. Lược đồ triển khai (Deployment Diagram)

- Thể hiện **cách phần mềm được triển khai** trên **phần cứng (server, database, client, network...)**.

9. Lược đồ thành phần (Component Diagram)

- Mô tả **các thành phần phần mềm** và **cách chúng kết nối với nhau**.

10. Lược đồ tổng quan tương tác (Interaction Overview Diagram)

- Kết hợp **lược đồ hoạt động** và **lược đồ tuần tự** để hiển thị **luồng tương tác của hệ thống**.

11. Lược đồ thời gian (Timing Diagram)

- Mô tả cách một **đối tượng thay đổi trạng thái** theo **thời gian**.

☐ Sơ đồ tuần tự (Sequence Diagram)

1. Định nghĩa:

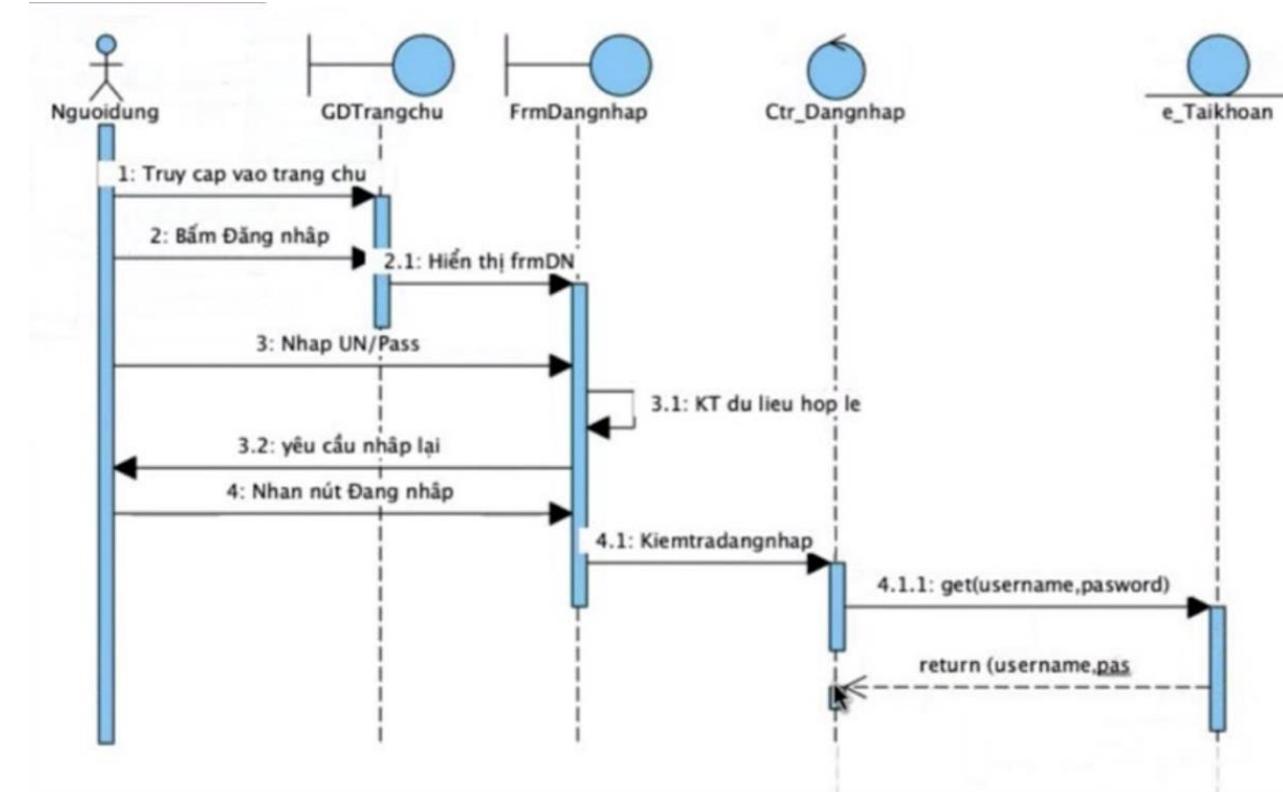
- Sơ đồ tuần tự là một loại sơ đồ UML mô tả cách các đối tượng trong hệ thống tương tác với nhau theo thứ tự thời gian.
- Nó thể hiện trình tự các thông điệp (messages) được gửi giữa các đối tượng để thực hiện một chức năng cụ thể.

2. Thành phần chính:

- Actors (tác nhân):** Người dùng hoặc hệ thống bên ngoài.
- Objects (đối tượng):** Các thành phần trong hệ thống.
- Lifelines (đường sinh mệnh):** Biểu diễn vòng đời của một đối tượng trong quá trình tương tác.
- Messages (thông điệp):** Giao tiếp giữa các đối tượng theo thứ tự thời gian.

3. Ví dụ minh họa:

(Một sơ đồ tuần tự thể hiện quy trình đăng nhập vào hệ thống)



Sơ đồ tuần tự- Sequence diagram

1. Phân tích sơ đồ tuần tự

Quan sát sơ đồ, sinh viên cần xác định:

- **Các đối tượng tham gia trong hệ thống:**

- **Người dùng (Nguoidung):** Thực hiện đăng nhập vào hệ thống.
- **Giao diện trang chủ (GDTrangchu):** Xử lý yêu cầu hiển thị form đăng nhập.
- **Form đăng nhập (FrmDangnhap):** Nơi người dùng nhập thông tin đăng nhập.
- **Controller đăng nhập (Ctr_Dangnhap):** Xử lý kiểm tra dữ liệu hợp lệ.
- **Thực thể tài khoản (e_Taikhoan):** Cung cấp thông tin tài khoản để xác thực.

- **Các bước xử lý trong sơ đồ:**

- Người dùng truy cập trang chủ.
- Người dùng bấm **đăng nhập**, hệ thống hiển thị **form đăng nhập**.
- Người dùng **nhập tên đăng nhập và mật khẩu**.
- Hệ thống kiểm tra thông tin:
 - Nếu hợp lệ, kiểm tra đăng nhập.
 - Nếu sai, yêu cầu nhập lại.
- Hệ thống gửi thông tin đến **bảng tài khoản** để xác thực.
- Hệ thống nhận phản hồi và thông báo kết quả đăng nhập.

1. Yêu cầu sinh viên vẽ lại sơ đồ tuần tự nhưng bổ sung chi tiết hơn:

- **Thêm trường hợp đăng nhập sai quá số lần quy định** → hệ thống khóa tài khoản.
- **Thêm bước thông báo lỗi cụ thể** (ví dụ: "Sai mật khẩu", "Tài khoản không tồn tại").
- **Thêm tùy chọn quên mật khẩu** để người dùng có thể khôi phục tài khoản.

2. Thiết kế sơ đồ tuần tự cho các quy trình khác:

- **Quy trình đăng ký tài khoản mới:** Người dùng tạo tài khoản, hệ thống xác nhận qua email.
- **Quy trình đặt lại mật khẩu:** Gửi yêu cầu, nhận mã OTP, tạo mật khẩu mới.
- **Quy trình đăng xuất:** Người dùng đăng xuất, hệ thống xóa session.

☐ Sơ đồ cộng tác (Collaboration Diagram)

1. Định nghĩa:

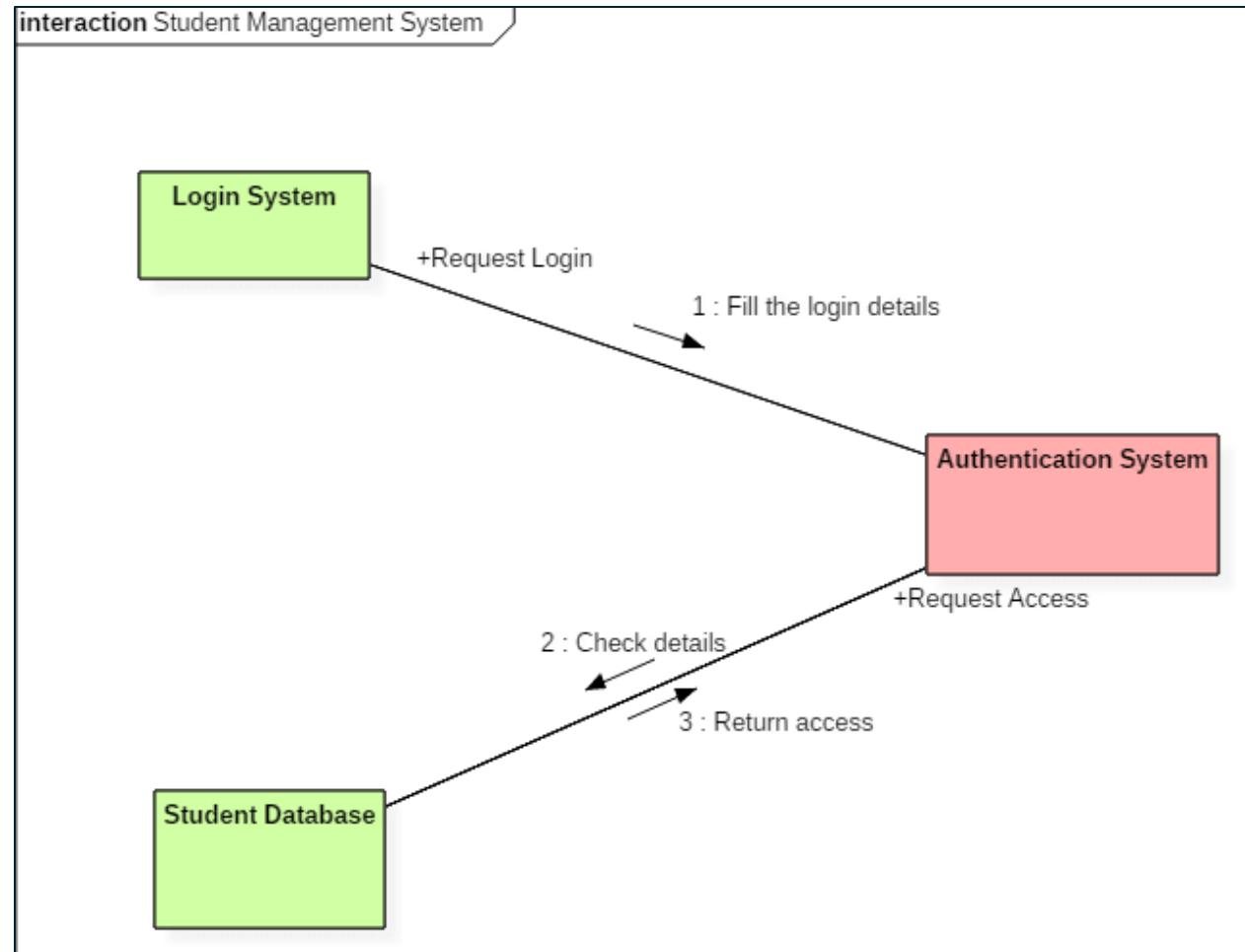
- Sơ đồ cộng tác là một loại sơ đồ UML tập trung vào mối quan hệ giữa các đối tượng thay vì thứ tự thời gian.
- Nó thể hiện cách các đối tượng liên kết với nhau và trao đổi thông điệp để hoàn thành một chức năng.

2. Thành phần chính:

- Objects (đối tượng):** Các thành phần trong hệ thống.
- Links (liên kết):** Kết nối giữa các đối tượng thể hiện quan hệ của chúng.
- Messages (thông điệp):** Mô tả sự tương tác giữa các đối tượng.

3. Ví dụ minh họa:

(Một sơ đồ cộng tác thể hiện quy trình – quản lý sv)



**Sơ đồ cộng tác-
Colaboration diagram**

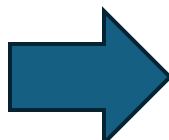
2. BÀI TẬP THỰC HÀNH

1. Yêu cầu sinh viên vẽ lại sơ đồ cộng tác nhưng bổ sung chi tiết hơn:

1. Thêm trường hợp sai thông tin đăng nhập (hệ thống từ chối truy cập).
2. Thêm hành động thông báo lỗi hoặc yêu cầu nhập lại.
3. Mở rộng sơ đồ với các bước xử lý khi đăng nhập thành công (chuyển hướng đến trang quản lý sinh viên).

2. Thiết kế sơ đồ cộng tác cho các quy trình khác:

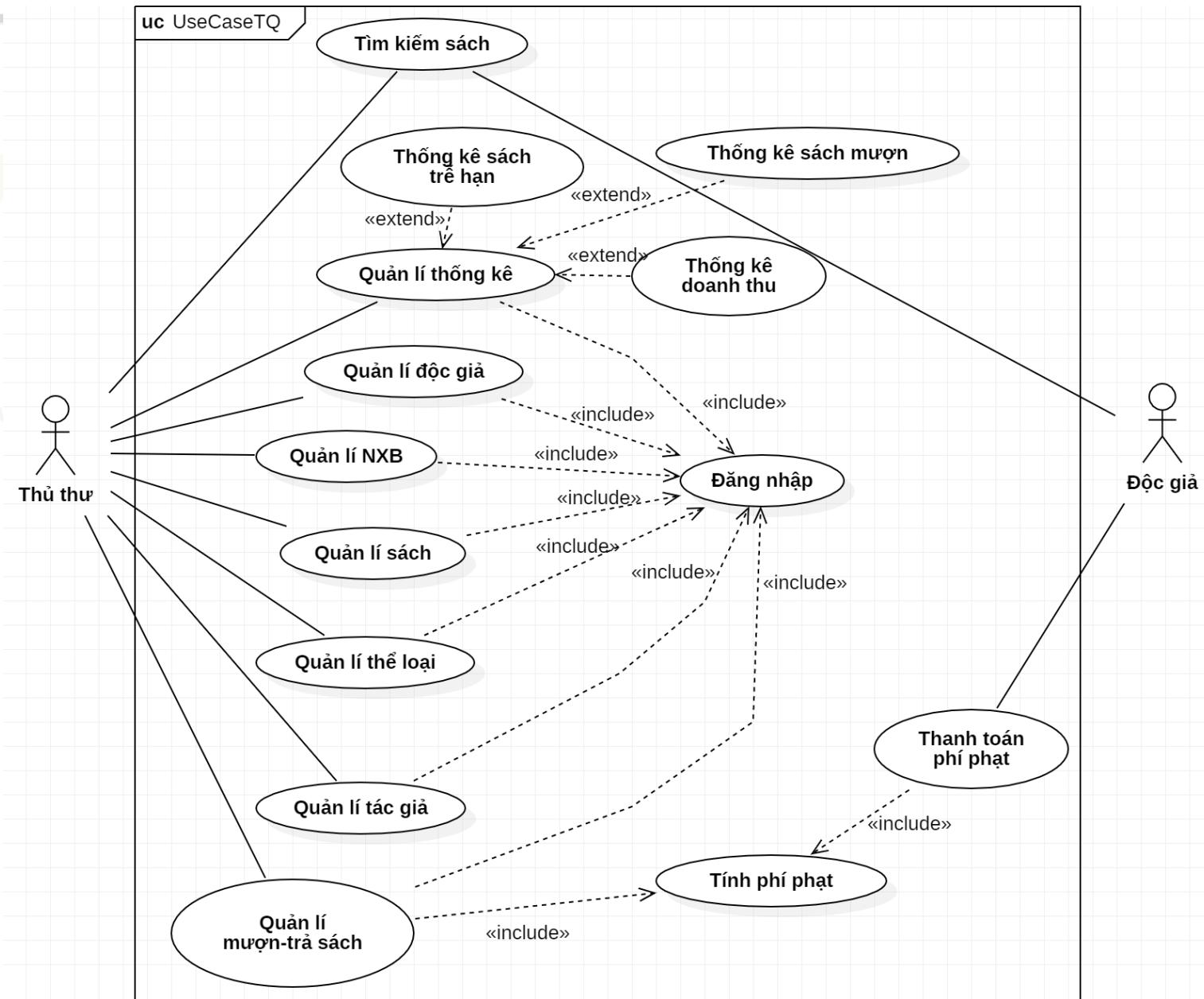
1. Quy trình đăng ký môn học: Tương tác giữa sinh viên, hệ thống đăng ký, cơ sở dữ liệu.
2. Quy trình cập nhật điểm: Tương tác giữa giảng viên, hệ thống điểm số, cơ sở dữ liệu sinh viên.



Báo cáo và thảo luận:

- Sinh viên trình bày sơ đồ đã thiết kế.
- Giảng viên đặt câu hỏi để sinh viên giải thích:
 - Các đối tượng có hợp lý không?
 - Luồng xử lý có đầy đủ không?
 - Các tương tác có thể tối ưu hơn không?
- Góp ý và chỉnh sửa sơ đồ nếu cần.

SCENARIO USE CASE PHIÊN BẢN 1



Scenario "Mượn sách" - Phiên bản 2

1. Tác nhân liên quan

- **Độc giả:** Người mượn sách.
- **Thủ thư:** Người quản lý và thực hiện các thao tác liên quan đến mượn/trả sách.
- **Hệ thống quản lý thư viện:** Hệ thống xử lý thông tin và xác nhận giao dịch.

2. Kịch bản chi tiết:

Trường hợp chuẩn (Normal Scenario)

1. **Độc giả đăng nhập vào hệ thống** (Use Case: *Đăng nhập*).
2. Độc giả thực hiện **tìm kiếm sách** cần mượn (Use Case: *Tìm kiếm sách*).
3. Hệ thống hiển thị thông tin sách, bao gồm:
 - Tên sách, tác giả, thể loại, số lượng còn lại.
4. Nếu sách có sẵn, độc giả thực hiện yêu cầu mượn sách.
5. Hệ thống kiểm tra tài khoản độc giả:
 - Độc giả có sách quá hạn chưa trả không?
 - Độc giả có khoản phí phạt chưa thanh toán không?
6. Nếu tài khoản hợp lệ, hệ thống ghi nhận lệnh mượn sách và thông báo thành công.
7. **Thủ thư xác nhận mượn sách** (Use Case: *Quản lý mượn-trả sách*).
8. Hệ thống cập nhật số lượng sách trong kho.
9. **Hoàn tất quy trình mượn sách.**

□ TRƯỜNG HỢP NGOẠI LỆ (EXCEPTIONAL SCENARIO)

1. Sách không có sẵn:

- Hệ thống thông báo sách đã hết và đề xuất sách tương tự nếu có.
- Độc giả có thể đặt trước sách nếu thư viện cho phép.

2. Tài khoản độc giả có vi phạm:

- Nếu độc giả có sách quá hạn, hệ thống hiển thị cảnh báo và yêu cầu trả sách trước.
- Nếu độc giả có khoản **phí phạt**, hệ thống yêu cầu thanh toán trước khi tiếp tục (Use Case: *Thanh toán phí phạt*).

3. Độc giả không hoàn thành đăng nhập:

- Nếu sai thông tin đăng nhập, hệ thống yêu cầu nhập lại hoặc đặt lại mật khẩu.

4. Các Use Case liên quan

- **Đăng nhập** (*Include* với tất cả Use Case khác).
- **Tìm kiếm sách** (*Extend* với quản lý sách, nhà xuất bản, thể loại, tác giả).
- **Quản lý mượn-trả sách** (*Include* với tính phí phạt nếu có sách trễ hạn).
- **Tính phí phạt** (*Include* với thanh toán phí phạt nếu cần).

4. YÊU CẦU THỰC HÀNH CHO SINH VIÊN

1. Vẽ lại sơ đồ tuần tự (Sequence Diagram) thể hiện kịch bản trên.
2. Vẽ sơ đồ cộng tác (Collaboration Diagram) để thể hiện mối quan hệ giữa các đối tượng.
3. Mô hình hóa lớp thực thể (Entity Classes) liên quan đến mượn sách, độc giả, thủ thư, tài khoản.

❑ LỢI ÍCH CỦA PHIÊN BẢN 2

- **Chi tiết hơn:** Mô tả từng bước để đảm bảo hệ thống vận hành chính xác.
- **Xử lý ngoại lệ:** Bao gồm các trường hợp đặc biệt khi độc giả có sách trễ hạn hoặc phí phạt.
- **Liên kết chặt chẽ với Use Case:** Mỗi bước trong kịch bản đều tương ứng với một Use Case trong sơ đồ.

1. SƠ ĐỒ TUẦN TỰ (SEQUENCE DIAGRAM)

- **Mục đích:** Mô tả **thứ tự** các **tin nhắn** (messages) được trao đổi giữa các đối tượng trong một tình huống cụ thể.
- **Cấu trúc:**
 - Các **đối tượng** (objects) được thể hiện theo cột dọc.
 - Các **thông điệp** (messages) giữa đối tượng được biểu diễn bằng mũi tên.
 - **Dòng thời gian** đi từ trên xuống dưới.
- **Ứng dụng:** Dùng để **phân tích quy trình** trong hệ thống, đặc biệt hữu ích trong việc kiểm tra luồng xử lý nghiệp vụ.

Ví dụ: **Đăng nhập hệ thống**

1. Người dùng nhập tài khoản/mật khẩu.
2. Hệ thống kiểm tra thông tin.
3. Nếu đúng, hệ thống cho phép truy cập.
4. Nếu sai, yêu cầu nhập lại.

□ HƯỚNG DẪN VẼ SƠ ĐỒ TUẦN TỰ (SEQUENCE DIAGRAM)

Bước 1: Xác định Use Case cần vẽ

- Ví dụ: Quy trình đăng nhập vào hệ thống

Bước 2: Xác định các đối tượng liên quan

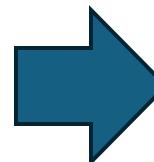
- **Người dùng (Actor)**: Sinh viên hoặc nhân viên đăng nhập
- **Hệ thống giao diện (UI)**: Màn hình đăng nhập
- **Hệ thống xác thực (Authentication System)**: Kiểm tra tài khoản
- **Cơ sở dữ liệu (Database)**: Lưu trữ thông tin tài khoản

Bước 3: Vẽ các đối tượng theo cột dọc

- Ký hiệu **Hình chữ nhật** có gạch dưới (*ObjectName*) thể hiện đối tượng.
- Vẽ **Actor** bên trái, sau đó vẽ **các hệ thống liên quan** từ trái sang phải.

Bước 4: Thêm các dòng thời gian và tin nhắn

- Mỗi đối tượng có một **dòng thời gian (lifeline)** kéo dài theo chiều dọc.
- Vẽ **mũi tên (messages)** thể hiện tương tác giữa các đối tượng.
- Ghi rõ **thứ tự thông điệp** theo từng bước:
 - Người dùng nhập tài khoản/mật khẩu.
 - UI gửi thông tin đến Authentication System.
 - Authentication System gửi truy vấn đến Database.
 - Database kiểm tra thông tin và phản hồi.
 - Nếu đúng, hệ thống hiển thị giao diện thành công.
 - Nếu sai, yêu cầu nhập lại.



Kiểm tra lại sơ đồ:

- Đảm bảo đúng **luồng xử lý** của hệ thống.
- Kiểm tra sự **nhất quán** giữa sơ đồ và thực tế.

HƯỚNG DẪN VẼ SƠ ĐỒ CỘNG TÁC (COLLABORATION DIAGRAM)

Bước 1: Xác định Use Case cần vẽ

- Ví dụ: Quy trình mượn sách tại thư viện

Bước 2: Xác định các đối tượng liên quan

- Người dùng (Actor)**: Độc giả mượn sách
- Hệ thống giao diện (UI)**: Màn hình mượn sách
- Hệ thống quản lý thư viện (Library System)**: Xử lý yêu cầu
- Cơ sở dữ liệu sách (Book Database)**: Kiểm tra tình trạng sách

Bước 3: Vẽ các đối tượng

- Sử dụng **hình chữ nhật** đại diện cho mỗi đối tượng.
- Kết nối các đối tượng bằng **đường liên kết**.

Bước 4: Thêm các thông điệp giao tiếp

- Mỗi đường kết nối thể hiện **sự hợp tác** giữa các đối tượng.
- Đánh số **thứ tự thông điệp** để thể hiện trình tự:
 - Người dùng gửi yêu cầu mượn sách đến UI.
 - UI chuyển yêu cầu đến Library System.
 - Library System kiểm tra tình trạng sách từ Book Database.
 - Nếu sách có sẵn, hệ thống cập nhật thông tin mượn.
 - Hệ thống gửi phản hồi về UI, hiển thị thông báo cho người dùng.



Kiểm tra lại sơ đồ
•Đảm bảo **các liên kết hợp tác** phản ánh đúng cách hệ thống hoạt động.
•Kiểm tra **tính logic** của luồng giao tiếp giữa các đối tượng.

6.6.1. Viết các **scenario** phiên bản 1 cho từng **use case** (*Write scenario version 1 for each use case*)

6.6.2. Trích xuất các **lớp** cho hệ thống (*Extract classes for the system*)

6.6.3. Xây dựng sơ đồ lớp (*Construct class diagrams*)

6.6.4. Viết **scenario** phiên bản 2 (*Write scenario version 2*)

6.6.5. Xây dựng sơ đồ tuần tự, cộng tác cho từng **use case** (*Construct sequence and collaboration diagrams for each use case*)

6.6.1. VIẾT CÁC SCENARIO PHIÊN BẢN 1 CHO TỪNG USE CASE

Ví dụ: Use case "Đăng ký khóa học" – Scenario phiên bản 1:

- Sinh viên chọn chức năng đăng ký khóa học.
- Sinh viên nhập mã khóa học cần đăng ký.
- Hệ thống kiểm tra điều kiện đăng ký và hiển thị thông báo thành công.

■ MỤC TIÊU BÀI THỰC HÀNH

- Hiểu cách viết **scenario phiên bản 1** cho từng **Use Case** trong hệ thống.
- Xây dựng kịch bản xử lý cơ bản dựa trên **các bước tuần tự** của hệ thống.
- Ứng dụng UML để thể hiện kịch bản thành **sơ đồ tuần tự hoặc sơ đồ cộng tác**.

BÀI TẬP THỰC HÀNH

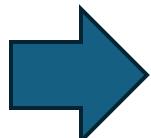
Bài tập 1: Viết Scenario phiên bản 1 cho Use Case "**Mượn sách thư viện**".

Bài tập 2: Viết Scenario phiên bản 1 cho Use Case "**Thanh toán học phí**".

Bài tập 3: Vẽ sơ đồ tuần tự UML cho Use Case "**Đăng ký khóa học**" dựa trên kịch bản đã viết.

Lưu ý cho sinh viên:

- Sử dụng **công cụ UML** như **Draw.io**, **StarUML**, hoặc **Lucidchart** để vẽ sơ đồ minh họa.
- So sánh kịch bản với thực tế để đảm bảo tính chính xác và khả thi.
- Nếu gặp khó khăn, hỏi giảng viên để được hướng dẫn thêm.



HƯỚNG DẪN TỪNG BƯỚC

Bước 1: Chọn một Use Case cụ thể

Ví dụ: **Đăng ký khóa học**

Các Use Case khác có thể chọn:

- **Mượn sách trong thư viện**
- **Thanh toán học phí**
- **Đăng nhập hệ thống**

Bước 2: Viết Scenario phiên bản 1 theo các bước cơ bản

Use Case: Đăng ký khóa học

Bước 1: Sinh viên chọn chức năng đăng ký khóa học.

Bước 2: Sinh viên nhập mã khóa học cần đăng ký.

Bước 3: Hệ thống kiểm tra điều kiện đăng ký (có đủ điều kiện hay không).

Bước 4: Nếu hợp lệ, hệ thống hiển thị thông báo "**Đăng ký thành công**".

Bước 5: Nếu không hợp lệ, hệ thống hiển thị thông báo "**Không thể đăng ký khóa học này**".

Bước 3: Kiểm tra lại kịch bản

- **Tính logic:** Đảm bảo từng bước **đúng trình tự** thực hiện trong hệ thống.
- **Đầy đủ thông tin:** Kiểm tra xem có cần bổ sung bước nào không.
- **Đúng đối tượng:** Mỗi bước phải rõ ràng về ai **thực hiện hành động** và **hệ thống phản hồi ra sao**.

□ Phạm vi thực hành tiếp theo sinh viên tiếp tục hoàn thiện:

1. Trích xuất các lớp cho hệ thống
2. Xây dựng sơ đồ lớp (*Construct class diagrams*)
3. Viết scenario phiên bản 2
4. Xây dựng sơ đồ tuần tự, cộng tác cho từng use case.

BÀI THỰC HÀNH TIẾP THEO: HOÀN THIỆN USE CASE "ĐĂNG KÝ KHÓA HỌC"

- Sau khi đã viết scenario phiên bản 1 cho Use Case "Đăng ký khóa học", sinh viên tiếp tục hoàn thiện hệ thống bằng các bước sau:

1. Trích xuất các lớp cho hệ thống

- Xác định **các lớp biên (Boundary classes)**, **lớp điều khiển (Control classes)** và **lớp thực thể (Entity classes)** phù hợp với Use Case "Đăng ký khóa học".
- Ví dụ: Lớp **GiaoDienDangKy**, **QuanLyDangKy**, **SinhVien**, **KhoaHoc**, v.v.

2. Xây dựng sơ đồ lớp (*Construct class diagrams*)

- Thiết kế sơ đồ lớp thể hiện các **mối quan hệ** giữa các lớp được trích xuất ở bước 1.
- Xác định **các thuộc tính và phương thức quan trọng** của từng lớp.

3. Viết scenario phiên bản 2 (*Write scenario version 2*)

- Hoàn thiện kịch bản chi tiết hơn cho Use Case "Đăng ký khóa học".
- Ví dụ: Bổ sung điều kiện ràng buộc như kiểm tra số lượng tín chỉ tối đa, xác thực thông tin đăng ký, xử lý trường hợp lỗi, v.v.

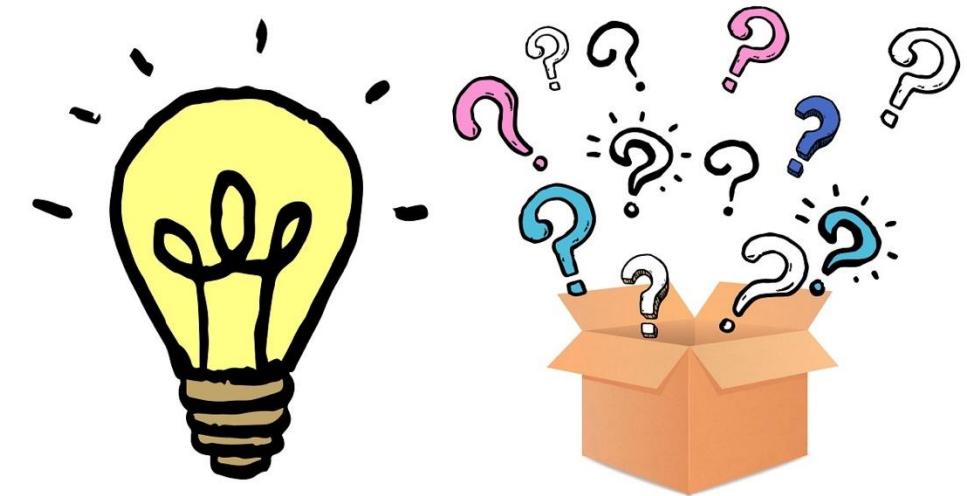
4. Xây dựng sơ đồ tuần tự, cộng tác cho từng Use Case (*Construct sequence and collaboration diagrams for each use case*)

- Sơ đồ tuần tự:** Biểu diễn trình tự trao đổi tin nhắn giữa các đối tượng trong quá trình đăng ký khóa học.
- Sơ đồ cộng tác:** Thể hiện các đối tượng tham gia và các kết nối hợp tác giữa chúng để thực hiện chức năng đăng ký khóa học.

Yêu cầu sinh viên thực hành:

- Thiết kế và vẽ sơ đồ bằng công cụ UML (Enterprise Architect, Visual Paradigm, StarUML, v.v.).
- Chuẩn bị báo cáo mô tả chi tiết các bước thực hiện.
- Trình bày kết quả và phản biện trước lớp.

- 6.1 CÂU HỎI TRẮC NGHIỆM
- 6.2 CÂU HỎI TRẢ LỜI NGẮN
- 6.3 CÂU HỎI THẢO LUẬN NHÓM
- 6.4 CÂU HỎI TÌNH HUỐNG



1. Câu hỏi 1: Lớp nào dưới đây đại diện cho các đối tượng có dữ liệu và trạng thái cần quản lý trong hệ thống?

- A. Lớp biên
- B. Lớp điều khiển
- C. Lớp thực thể
- D. Lớp giao diện

2. Câu hỏi 2: Lớp biên trong hệ thống có vai trò gì?

- A. Xử lý dữ liệu
- B. Điều khiển luồng công việc
- C. Giao tiếp với người dùng hoặc hệ thống bên ngoài
- D. Lưu trữ dữ liệu

3. Câu hỏi 3: Quan hệ nào giữa các lớp thể hiện sự kế thừa?

- A. Association
- B. Aggregation
- C. Composition
- D. Inheritance

4. Câu hỏi 4: Sơ đồ lớp mô tả:

- A. Quan hệ giữa các đối tượng trong một luồng xử lý
- B. Các lớp và quan hệ giữa các lớp trong hệ thống
- C. Giao diện người dùng của hệ thống
- D. Thứ tự luồng xử lý giữa các đối tượng

5. Câu hỏi 5: Quan hệ Include giữa các use case được dùng khi:

- A. Một use case mở rộng một use case khác
- B. Một use case cần gọi một use case khác để hoàn thành chức năng
- C. Một use case kế thừa một use case khác
- D. Một use case được sử dụng bởi hệ thống bên ngoài

6. Câu hỏi 6: Scenario là gì?

- A. Một sơ đồ mô tả lớp trong hệ thống
- B. Một kịch bản mô tả cách hệ thống và người dùng tương tác
- C. Một sơ đồ mô tả các luồng dữ liệu
- D. Một tài liệu thiết kế giao diện

7. Câu hỏi 7: Quan hệ nào sau đây biểu diễn việc một lớp chứa một lớp khác nhưng lớp con vẫn có thể tồn tại độc lập?

- A. Aggregation
- B. Composition
- C. Association
- D. Inheritance

8. Câu hỏi 8: Sơ đồ tuần tự mô tả điều gì?

- A. Quan hệ giữa các lớp
- B. Thứ tự các thông điệp được trao đổi giữa các đối tượng
- C. Cấu trúc dữ liệu của hệ thống
- D. Các chức năng mà hệ thống cung cấp

9. Câu hỏi 9: Lớp điều khiển trong mô hình MVC tương ứng với thành phần nào?

- A. Model
- B. View
- C. Control
- D. Entity

10. Câu hỏi 10: Để biểu diễn quan hệ giữa các lớp, ta sử dụng sơ đồ nào?

- A. Sơ đồ tuần tự
- B. Sơ đồ cộng tác
- C. Sơ đồ lớp
- D. Sơ đồ use case

1. Lớp thực thể là gì?
2. Lớp điều khiển có vai trò gì trong hệ thống?
3. Scenario là gì?
4. Quan hệ Include giữa các use case là gì?
5. Mục đích của sơ đồ lớp là gì?
6. Quan hệ Aggregation khác gì so với Composition?
7. Sơ đồ tuần tự là gì?
8. Quan hệ Extend giữa các use case là gì?
9. Lớp biên có vai trò gì trong hệ thống?
10. Sơ đồ cộng tác là gì?

1. Thảo luận về vai trò của từng loại lớp (thực thể, biên, điều khiển) trong hệ thống.
2. So sánh sự khác nhau giữa Aggregation và Composition.
3. Thảo luận về tầm quan trọng của việc xây dựng sơ đồ lớp trong quá trình phân tích hệ thống.
4. Phân biệt sơ đồ tuần tự và sơ đồ cộng tác.
5. Thảo luận về vai trò của lớp điều khiển trong mô hình MVC.
6. Tại sao cần viết các scenario khi phân tích hệ thống?
7. Làm thế nào để đảm bảo rằng các use case được trích đầy đủ và chính xác?
8. Thảo luận về mối quan hệ giữa use case và scenario.
9. Phân tích ưu và nhược điểm của việc sử dụng sơ đồ tuần tự trong thiết kế hệ thống.
10. Thảo luận về cách cải thiện chất lượng kịch bản sử dụng (scenario) trong quá trình phân tích.

1. Trong quá trình phân tích hệ thống quản lý thư viện, nhóm phát triển phát hiện một số yêu cầu mới từ khách hàng sau khi đã viết xong các scenario. Nhóm phát triển nên xử lý như thế nào?
2. Một nhóm phát triển gặp khó khăn khi xác định các lớp điều khiển trong hệ thống. Hãy đề xuất giải pháp.
3. Sau khi hoàn thành sơ đồ lớp, khách hàng yêu cầu thêm một số chức năng mới. Nhóm phát triển cần làm gì để cập nhật sơ đồ lớp?
4. Khi viết các scenario cho use case "Đăng ký khóa học", nhóm phát triển gặp tình huống có nhiều trường hợp ngoại lệ. Làm thế nào để xử lý tình huống này?
5. Trong quá trình xây dựng sơ đồ tuần tự, một số đối tượng không có vai trò rõ ràng. Nhóm phát triển nên làm gì?
6. Sau khi xây dựng sơ đồ lớp, nhóm phát triển phát hiện ra một số quan hệ giữa các lớp bị sai. Hãy đề xuất cách sửa chữa.
7. Một nhóm phát triển gặp khó khăn khi mô tả các quan hệ giữa các use case. Hãy đề xuất giải pháp.
8. Trong dự án phát triển phần mềm quản lý bán hàng, nhóm phát triển cần xác định các lớp biên cho hệ thống. Hãy đưa ra đề xuất phù hợp.
9. Khách hàng yêu cầu thêm chức năng mới sau khi các scenario đã được hoàn thiện. Nhóm phát triển nên làm gì?
10. Trong quá trình xây dựng sơ đồ cộng tác, một số đối tượng không tương tác đúng theo yêu cầu. Hãy đề xuất cách giải quyết.

A faint, semi-transparent background image of a world map in green and light blue. Numerous small, grey airplane icons are scattered across the map, representing flight routes or paths.

REVIEW LESSION (1-3)

I. Câu hỏi trắc nghiệm (Multiple Choice Questions)

- 1. Pha nào trong tiến trình phát triển phần mềm tập trung vào việc thu thập yêu cầu từ khách hàng?**
A. Pha thiết kế
B. Pha kiểm thử
C. Pha lấy yêu cầu
D. Pha triển khai
- 2. Mô hình phát triển phần mềm nào có các vòng lặp liên tục và cải tiến sản phẩm sau mỗi lần lặp?**
A. Mô hình thác nước
B. Mô hình xoắn ốc
C. Mô hình Agile
D. Mô hình kiểm thử
- 3. Trong kỹ thuật lấy yêu cầu phần mềm, phương pháp nào giúp khai thác thông tin từ khách hàng bằng cách quan sát cách họ làm việc?**
A. Phỏng vấn
B. Khảo sát
C. Quan sát
D. Phân tích tài liệu
- 4. Trong chuẩn hóa cơ sở dữ liệu, dạng chuẩn nào loại bỏ các phụ thuộc bắc cầu?**
A. 1NF
B. 2NF
C. 3NF
D. BCNF
- 5. Trong UML, sơ đồ nào giúp mô tả sự tương tác giữa các đối tượng theo thời gian?**
A. Sơ đồ lớp
B. Sơ đồ tuần tự
C. Sơ đồ trạng thái
D. Sơ đồ hoạt động

6. UML, sơ đồ nào giúp mô tả sự tương tác giữa các đối tượng theo thời gian?

- A. Sơ đồ lớp
- B. Sơ đồ tuần tự
- C. Sơ đồ trạng thái
- D. Sơ đồ hoạt động

7. CMMI mức 5 – Optimizing tập trung vào điều gì?

- A. Kiểm soát quy trình
- B. Định lượng quy trình
- C. Cải tiến quy trình
- D. Xác định quy trình

8. Đâu là ưu điểm chính của mô hình phát triển phần mềm Agile?

- A. Linh hoạt và thay đổi nhanh chóng
- B. Yêu cầu chi tiết ngay từ đầu
- C. Kiểm thử chỉ diễn ra ở giai đoạn cuối
- D. Phù hợp với tất cả các loại dự án

9. Nguyên tắc DRY (Don't Repeat Yourself) trong lập trình có ý nghĩa gì?

- A. Hạn chế viết code lặp lại
- B. Viết code dễ đọc hơn
- C. Tăng tốc độ thực thi của chương trình
- D. Giảm chi phí phát triển phần mềm

10. Yếu tố nào quan trọng nhất trong việc thiết kế một lớp trong lập trình hướng đối tượng?

- A. Đặt tên biến dễ hiểu
- B. Đảm bảo tính đóng gói và tái sử dụng
- C. Viết phương thức càng nhiều càng tốt
- D. Dùng số nguyên thay vì số thực

11. Phát biểu nào đúng về kiểm thử phần mềm?

- A. Kiểm thử chỉ diễn ra sau khi phát triển xong sản phẩm
- B. Kiểm thử giúp tìm ra tất cả lỗi trong phần mềm
- C. Kiểm thử có thể thực hiện song song với phát triển phần mềm
- D. Kiểm thử không quan trọng nếu phần mềm được viết tốt

1. Định nghĩa phần mềm và công nghệ phần mềm.
2. Mô tả ngắn gọn về các mô hình vòng đời phát triển phần mềm phổ biến.
3. Liệt kê ba loại yêu cầu phần mềm chính và giải thích từng loại.
4. Mô tả vai trò của sơ đồ lớp UML trong thiết kế phần mềm.
5. Tại sao kiểm thử phần mềm quan trọng trong phát triển phần mềm?
6. Định nghĩa nguyên lý **SOLID** trong lập trình hướng đối tượng.
7. Mô tả sự khác biệt giữa kiểm thử hộp đen và kiểm thử hộp trắng.
8. Mô tả quy trình thiết kế cơ sở dữ liệu từ sơ đồ lớp UML.
9. Nêu ba ưu điểm của việc sử dụng mô hình phát triển phần mềm Agile.
10. Liệt kê các giai đoạn chính trong quá trình chuẩn hóa cơ sở dữ liệu.

III. Câu hỏi thảo luận nhóm (Discussion Questions)

1. So sánh mô hình phát triển phần mềm Agile và mô hình Waterfall.
2. Lợi ích của việc sử dụng UML trong thiết kế phần mềm là gì?
3. Làm thế nào để đảm bảo phần mềm có thể bảo trì tốt trong tương lai?
4. Tại sao các công ty phần mềm thường sử dụng mô hình phát triển lặp (Iterative Development)?
5. Vai trò của kiến trúc phần mềm trong việc xây dựng một hệ thống phần mềm phức tạp.
6. Làm thế nào để đảm bảo rằng một hệ thống phần mềm đáp ứng được yêu cầu bảo mật?
7. So sánh kiểm thử đơn vị (Unit Testing) và kiểm thử tích hợp (Integration Testing).
8. Những thách thức chính trong việc thu thập yêu cầu phần mềm là gì?
9. Cách áp dụng quy trình Scrum vào dự án phát triển phần mềm thực tế.
10. Tại sao việc tối ưu hóa mã nguồn lại quan trọng trong phát triển phần mềm?

IV. Câu hỏi tình huống (Situational Questions)

1. Bạn là một kỹ sư phần mềm trong một nhóm phát triển, khách hàng liên tục thay đổi yêu cầu. Bạn sẽ xử lý như thế nào?
2. Trong quá trình kiểm thử, bạn phát hiện một lỗi nghiêm trọng nhưng trưởng nhóm quyết định không sửa chữa. Bạn sẽ làm gì?
3. Một dự án phần mềm gặp tình trạng chậm tiến độ do thay đổi yêu cầu liên tục từ khách hàng. Bạn sẽ đề xuất giải pháp gì?
4. Nhóm của bạn đang thiết kế cơ sở dữ liệu cho một hệ thống thương mại điện tử. Làm thế nào để đảm bảo thiết kế cơ sở dữ liệu không bị dư thừa?
5. Bạn cần lựa chọn giữa hai mô hình phát triển phần mềm: Waterfall và Agile. Bạn sẽ chọn mô hình nào cho một dự án startup công nghệ? Tại sao?
6. Một dự án phần mềm lớn gặp vấn đề về hiệu suất. Những bước nào cần thực hiện để tối ưu hiệu suất phần mềm?
7. Một hệ thống đang hoạt động có nhiều lỗi bảo mật. Bạn sẽ làm gì để tăng cường bảo mật mà không ảnh hưởng đến người dùng hiện tại?
8. Khi thiết kế phần mềm, nhóm của bạn có nhiều ý kiến trái ngược nhau về cách triển khai một tính năng. Làm thế nào để đưa ra quyết định tốt nhất?
9. Khách hàng yêu cầu hệ thống phải có giao diện thân thiện với người dùng. Bạn sẽ làm gì để đảm bảo hệ thống đáp ứng tiêu chí này?
10. Công ty bạn vừa tiếp nhận một dự án phần mềm cũ, nhưng không có tài liệu hướng dẫn. Bạn sẽ làm gì để hiểu và tiếp tục phát triển hệ thống này?

1. Viết một chương trình Java để quản lý thông tin sinh viên sử dụng lập trình hướng đối tượng.
2. Xây dựng sơ đồ lớp UML cho hệ thống quản lý thư viện.
3. Viết test case cho tính năng đăng ký tài khoản trong một hệ thống website.
4. Thiết kế cơ sở dữ liệu cho hệ thống bán hàng trực tuyến và chuẩn hóa đến 3NF.
5. Viết một đoạn mã Java đơn giản để minh họa nguyên tắc đóng gói trong lập trình hướng đối tượng.
6. Xây dựng sơ đồ tuần tự UML cho chức năng "Đăng nhập hệ thống".
7. Viết chương trình kiểm thử đơn vị (Unit Test) cho một phương thức tính tổng hai số trong Java.
8. Tạo kịch bản kiểm thử chức năng thanh toán trong hệ thống thương mại điện tử.
9. Xây dựng mô hình thực thể (ERD) cho hệ thống quản lý bệnh viện.
10. Phân tích và mô tả một use case cụ thể trong hệ thống đặt vé máy bay.

OUTLINE

7.1. THIẾT KẾ THUỘC TÍNH CHO LỚP.

7.2. THIẾT KẾ PHƯƠNG THỨC CHO CÁC LỚP

7.3. XÂY DỰNG THẺ CRC CHO CÁC LỚP

7.4. THIẾT KẾ SƠ ĐỒ THUẬT TOÁN CHO CÁC PHƯƠNG THỨC

7.5. HOÀN THIỆN SƠ ĐỒ LỚP CHI TIẾT

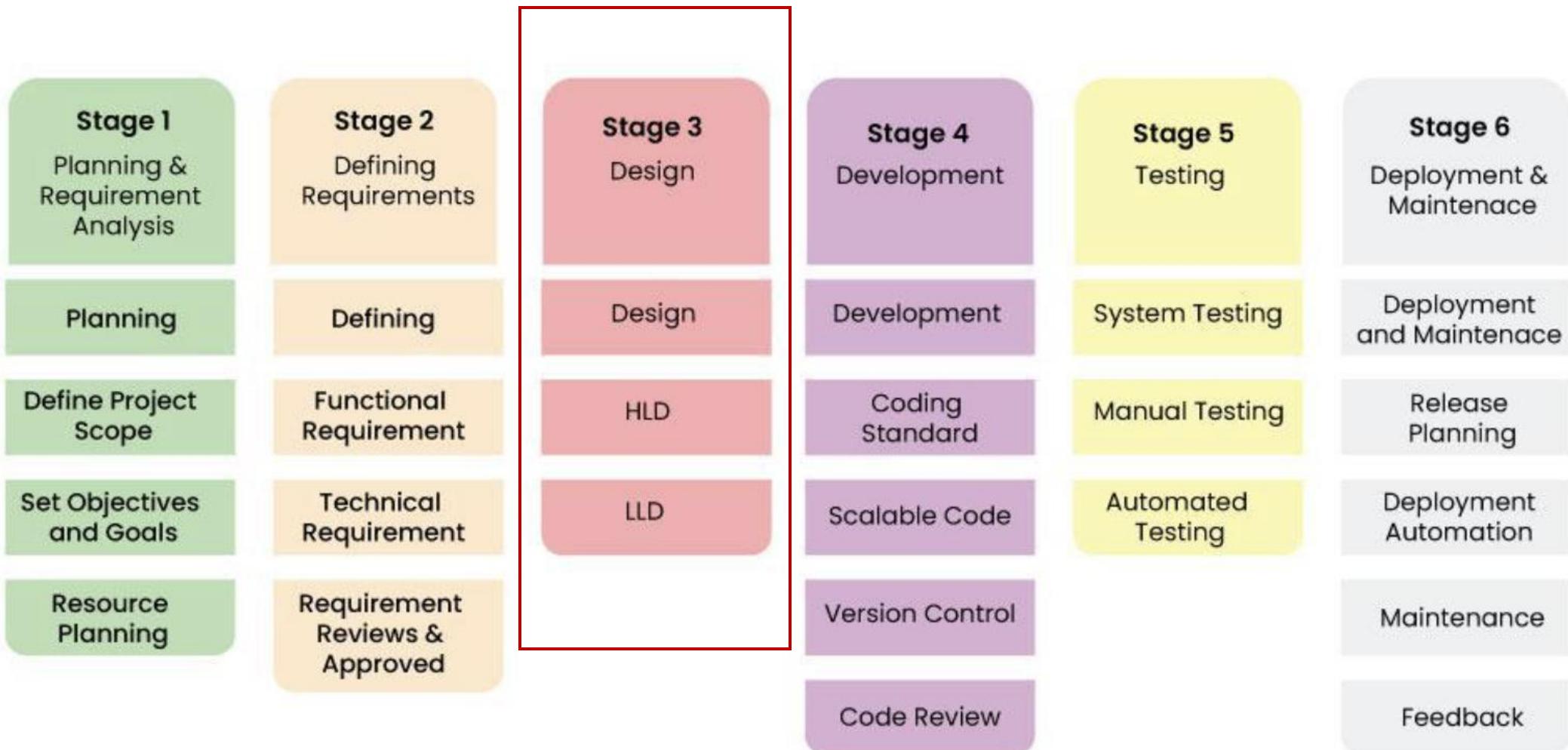
7.6. THIẾT KẾ CƠ SỞ DỮ LIỆU

7.7. CASE STUDY

7.8 CÂU HỎI CÙNG CÔ LÝ THUYẾT







- **Mục đích:** Xác định cấu trúc tổng thể của hệ thống, bao gồm các thành phần chính và cách chúng tương tác.
- **Hoạt động:**
 - Lựa chọn kiến trúc hệ thống (VD: **MVC**, **Microservices**, **Client-Server**).
 - Xác định công nghệ sử dụng (ngôn ngữ lập trình, database, framework).
- **Kết quả:**
 - Sơ đồ kiến trúc hệ thống (VD: **Deployment Diagram**, **Component Diagram**).

- **Mục đích:** Thiết kế logic tổng thể của hệ thống và các module chính.
- **Hoạt động:**
 - Phân chia hệ thống thành các module (VD: **Authentication, Payment, Reporting**).
 - Xác định luồng dữ liệu và giao tiếp giữa các module.
- **Kết quả:**
 - Tài liệu HLD mô tả chức năng từng module và interface.
 - Sử dụng **Flowchart** hoặc **Sequence Diagram**.

3. LOW-LEVEL DESIGN (LLD – THIẾT KẾ MỨC THẤP)

- **Mục đích:** Chi tiết hóa từng module thành các thành phần nhỏ (lớp, hàm, thuộc tính).
- **Hoạt động:**
 - Thiết kế lớp (class) với các phương thức và thuộc tính.
 - Xác định thuật toán cho các chức năng phức tạp.
- **Kết quả:**
 - Tài liệu LLD kèm **Class Diagram**, **ER Diagram** (cho database).
 - Ví dụ: Thiết kế chi tiết lớp User với phương thức login(), logout().

- **Mục đích:** Thiết kế cách người dùng và hệ thống tương tác.
- **Hoạt động:**
 - Thiết kế **UI/UX** cho ứng dụng (VD: wireframe, prototype).
 - Thiết kế **API endpoints** cho giao tiếp giữa các hệ thống.
- **Kết quả:**
 - Prototype giao diện người dùng (dùng Figma, Adobe XD).
 - Tài liệu API (VD: Swagger/OpenAPI).

- **Mục đích:** Xác định cấu trúc lưu trữ dữ liệu và mối quan hệ giữa các thực thể.
- **Hoạt động:**
 - Thiết kế Entity-Relationship Diagram (ERD).
 - Chuẩn hóa database (Normalization) để tránh dư thừa dữ liệu.
- **Kết quả:**
 - ERD chi tiết.
 - Script tạo bảng và quan hệ (VD: SQL).

- **Hệ thống Đặt vé Xem phim:**

- 1. Architectural Design:** Chọn kiến trúc Client-Server.
- 2. HLD:** Phân module thành **User Management, Booking, Payment.**
- 3. LLD:** Thiết kế lớp Booking với phương thức `reserveSeat()`, `cancelBooking()`.
- 4. Interface Design:** Prototype trang đặt vé với nút "Chọn ghế", "Thanh toán".
- 5. Database Design:** ERD cho bảng Users, Movies, Bookings.

07. CÔNG CỤ HỖ TRỢ THIẾT KẾ

Giai đoạn	Công cụ
Architectural Design	Lucidchart, Draw.io
HLD & LLD	Enterprise Architect, UML
Interface Design	Figma, Adobe XD
Database Design	MySQL Workbench, pgAdmin

7.1. Thiết kế thuộc tính cho lớp

7.2. Thiết kế phương thức cho các lớp

7.2.1. Nguyên lí A - Đóng gói dữ liệu

7.2.2. Nguyên lí B - Thực hiện lời gọi nhiều lần

7.2.3. Nguyên lí C - Hướng trách nhiệm

"THIẾT KẾ THUỘC TÍNH CHO LỚP" là quá trình xác định và định nghĩa các thuộc tính (fields/variables) cho một lớp (class) trong lập trình hướng đối tượng (OOP).

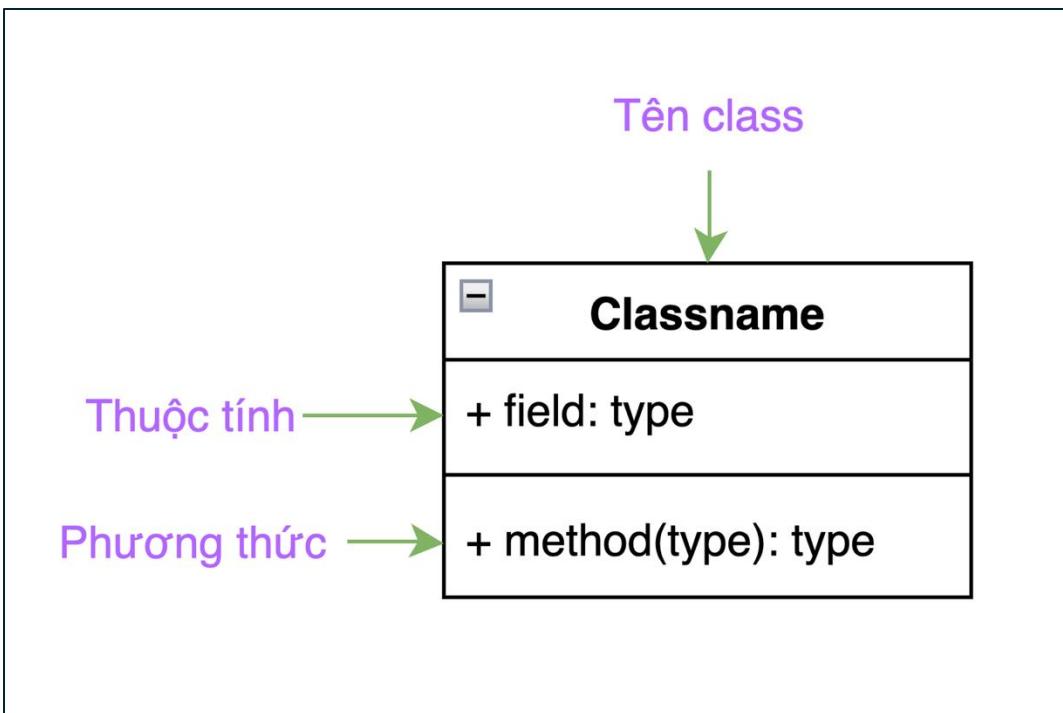
- **Giải thích**
- **Thuộc tính (Attributes/Fields):**

- Là các biến lưu trữ dữ liệu đặc trưng của một đối tượng thuộc lớp đó.
- Được khai báo với kiểu dữ liệu cụ thể như int, string, boolean,...
- Có thể có các phạm vi truy cập như private, public, protected.

❑ VÍ DỤ MINH HỌA

java

```
public class Student {  
    private String name; // Thuộc tính name  
    private int age; // Thuộc tính age  
}
```

 Copy Edit

❑ Mô tả:

Sinh viên hãy thiết kế một lớp "**Sản phẩm**" (**Product**) với các thuộc tính phù hợp. Xác định kiểu dữ liệu và phạm vi truy cập cho từng thuộc tính.

❑ Yêu cầu:

1. Xác định các thuộc tính chính của lớp **Product**.
2. Xác định kiểu dữ liệu (int, double, String, boolean, v.v.) cho từng thuộc tính.
3. Quy định phạm vi truy cập (private, public, protected).
4. Biểu diễn lớp dưới dạng sơ đồ UML và triển khai mã nguồn bằng ngôn ngữ lập trình Java hoặc Python.

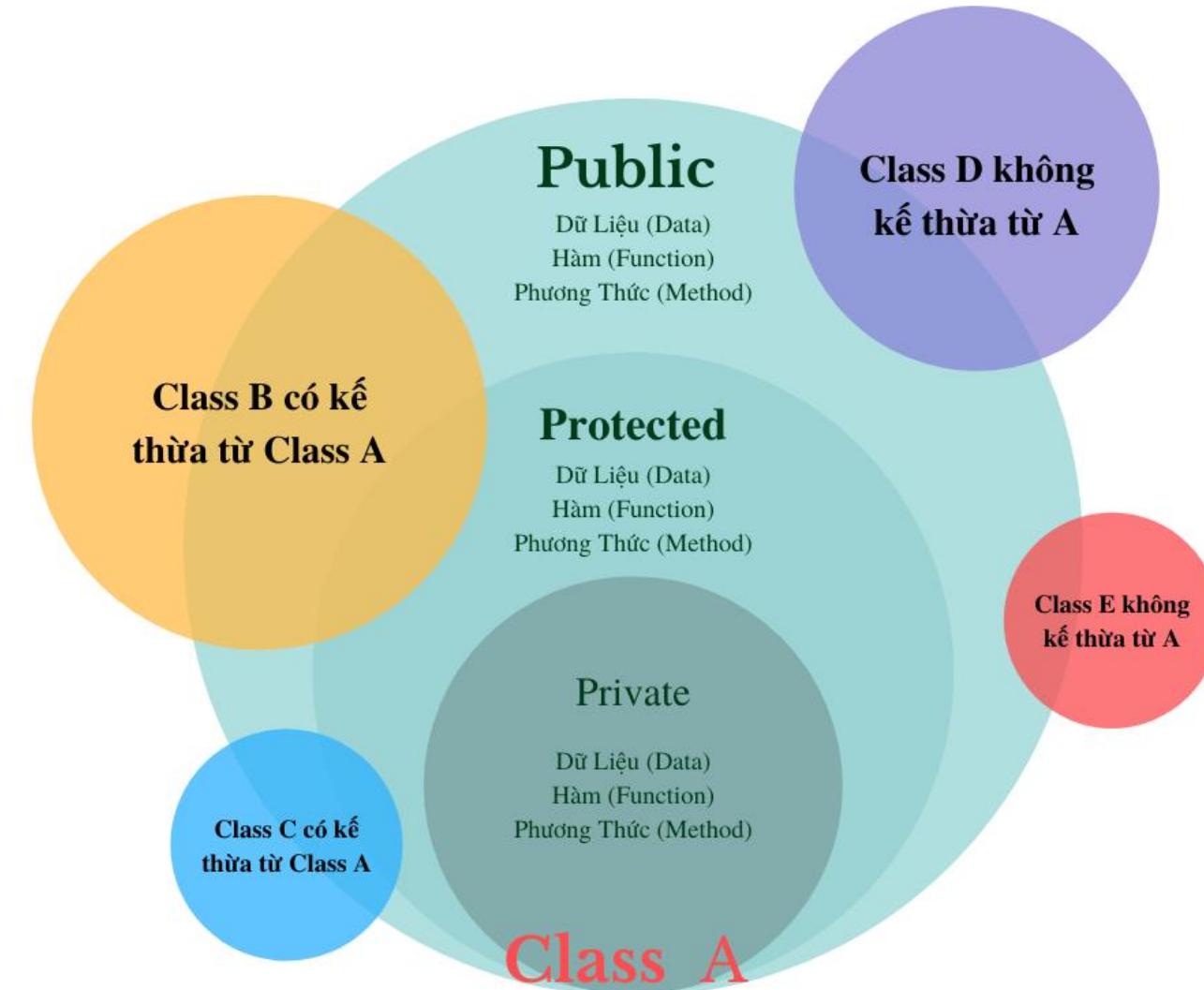
❑ GỢI Ý BÀI LÀM:**Xác định các thuộc tính cho lớp Product**

- Một sản phẩm thường có những đặc điểm sau:
- **Mã sản phẩm** (productId) – Kiểu String (hoặc int nếu chỉ dùng số).
- **Tên sản phẩm** (productName) – Kiểu String.
- **Giá sản phẩm** (price) – Kiểu double.
- **Số lượng tồn kho** (stockQuantity) – Kiểu int.
- **Sản phẩm còn hàng hay không** (isAvailable) – Kiểu boolean.

TÍNH ĐÓNG GÓI (ENCAPSULATION) TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

4 đặc tính của lập trình hướng đối tượng (OOP)

- 1.Tính đóng gói (Encapsulation)
- 2.Tính kế thừa (Inheritance)
- 3.Tính đa hình (Polymorphism)
- 4.Tính trừu tượng (Abstraction)



☐ Nguyên lý A – Đóng gói dữ liệu

- Đóng gói dữ liệu (Encapsulation) là một trong những nguyên lý cơ bản của lập trình hướng đối tượng (OOP). Nó đề cập đến việc gộp nhóm các dữ liệu (thuộc tính) và các phương thức (hàm) liên quan vào cùng một đơn vị, thường là một lớp (class). Điều này giúp quản lý và sử dụng mã nguồn hiệu quả hơn, đồng thời che giấu các chi tiết cài đặt nội bộ, chỉ cho phép truy cập thông qua các phương thức được xác định.

☐ Ý nghĩa của đóng gói dữ liệu:

- Quản lý và sử dụng hiệu quả:** Mỗi "gói" được thiết kế để thực hiện một nhóm chức năng cụ thể, giúp mã nguồn rõ ràng và dễ bảo trì.
- Che giấu thông tin (Data Hiding):** Bằng cách ẩn các chi tiết cài đặt nội bộ, đóng gói ngăn chặn truy cập trái phép và bảo vệ tính toàn vẹn của dữ liệu. Người lập trình có thể kiểm soát cách thức và phạm vi mà các phần bên ngoài tương tác với dữ liệu bên trong.

java

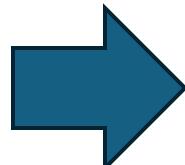
```
public class SinhVien {  
    private double diemTrungBinh; // Thuộc tính được đóng gói  
  
    // Phương thức thiết lập giá trị cho diemTrungBinh với kiểm tra hợp lệ  
    public void setDiemTrungBinh(double diem) {  
        if (diem >= 0.0 && diem <= 10.0) {  
            this.diemTrungBinh = diem;  
        } else {  
            System.out.println("Điểm không hợp lệ! Vui lòng nhập giá trị từ 0.0 đến 10.0");  
        }  
    }  
  
    // Phương thức lấy giá trị của diemTrungBinh  
    public double getDiemTrungBinh() {  
        return this.diemTrungBinh;  
    }  
}
```

[Sao chép](#) [Chỉnh sửa](#)

Phân tích:

- Thuộc tính diemTrungBinh được khai báo là private, ngăn chặn truy cập trực tiếp từ bên ngoài lớp.
- Phương thức setDiemTrungBinh() cho phép thay đổi giá trị của diemTrungBinh, nhưng chỉ khi giá trị nằm trong khoảng hợp lệ (0.0 đến 10.0).
- Phương thức getDiemTrungBinh() cung cấp cách truy cập giá trị của diemTrungBinh từ bên ngoài một cách an toàn.

Ví dụ minh họa: Xem xét lớp SinhVien với thuộc tính diemTrungBinh. Để bảo vệ dữ liệu này, chúng ta sẽ khai báo nó là **private** và cung cấp phương thức **setDiemTrungBinh()** để thay đổi giá trị, kèm theo kiểm tra tính hợp lệ.



☐ Nguyên lý B – Thực hiện lời gọi nhiều lần

Nguyên lý này nhấn mạnh việc thiết kế các phương thức sao cho chúng có thể tái sử dụng và được gọi nhiều lần trong nhiều tình huống khác nhau, nhằm giảm thiểu trùng lặp mã nguồn và tăng tính bảo trì cho hệ thống.

☐ Ví dụ minh họa:

Hãy xem xét một chương trình quản lý điểm số của sinh viên, trong đó có một phương thức **tinhTong()** giúp tính tổng điểm của các môn học.

Giải thích:

1. **tinhTong()** được thiết kế để **có thể tái sử dụng** và gọi lại nhiều lần khi cần tính tổng điểm.
2. **tinhDiemTrungBinh()** sử dụng **tinhTong()** để tính điểm trung bình.
3. **kiemTraHocBong()** cũng gọi **tinhDiemTrungBinh()**, tức là gián tiếp sử dụng **tinhTong()** để kiểm tra xem sinh viên có đủ điều kiện nhận học bổng hay không.

```
public class SinhVien {  
    private String ten;  
    private int[] diemMonHoc;  
  
    // Constructor  
    public SinhVien(String ten, int[] diemMonHoc) {  
        this.ten = ten;  
        this.diemMonHoc = diemMonHoc;  
    }  
  
    // Phương thức tính tổng điểm  
    public int tinhTong() {  
        int tong = 0;  
        for (int diem : diemMonHoc) {  
            tong += diem;  
        }  
        return tong;  
    }  
  
    // Phương thức tính điểm trung bình, sử dụng tinhTong()  
    public double tinhDiemTrungBinh() {  
        return (double) tinhTong() / diemMonHoc.length;  
    }  
  
    // Phương thức kiểm tra học bổng, sử dụng tinhTong()  
    public boolean kiemTraHocBong() {  
        return tinhDiemTrungBinh() >= 8.0;  
    }  
}
```

Nguyên lý C – Hướng trách nhiệm

- Nguyên lý này nhấn mạnh việc phân chia trách nhiệm rõ ràng giữa các lớp trong hệ thống để đảm bảo tính **tổ chức, bảo trì dễ dàng và tái sử dụng hiệu quả**.

Khái niệm:

- Mỗi lớp trong hệ thống nên chỉ chịu trách nhiệm về một nhiệm vụ cụ thể.
- Tránh để một lớp đảm nhiệm quá nhiều chức năng, làm tăng độ phức tạp và khó bảo trì.
- Áp dụng nguyên lý **SRP (Single Responsibility Principle - Nguyên lý đơn trách nhiệm)** trong lập trình hướng đối tượng.

Ví dụ minh họa:

- Giả sử chúng ta có một hệ thống quản lý sinh viên. Nếu không áp dụng nguyên lý này, chúng ta có thể thiết kế một lớp duy nhất **SinhVien** chứa cả thông tin sinh viên, xử lý tính điểm và thao tác với cơ sở dữ liệu. Điều này khiến lớp trở nên quá tải và khó bảo trì.

Thay vào đó, chúng ta **phân chia trách nhiệm** thành các lớp cụ thể như sau:



1. Lớp SinhVien (Chỉ lưu thông tin sinh viên)

```
java
public class SinhVien {
    private String ten;
    private String maSV;

    public SinhVien(String ten, String maSV) {
        this.ten = ten;
        this.maSV = maSV;
    }

    public String getTen() {
        return ten;
    }

    public String getMaSV() {
        return maSV;
    }
}
```

2. Lớp DiemSinhVien (Xử lý tính điểm)

```
java
public class DiemSinhVien {
    private SinhVien sinhVien;
    private int[] diemMonHoc;

    public DiemSinhVien(SinhVien sinhVien, int[] diemMonHoc) {
        this.sinhVien = sinhVien;
        this.diemMonHoc = diemMonHoc;
    }

    public double tinhDiemTrungBinh() {
        int tong = 0;
        for (int diem : diemMonHoc) {
            tong += diem;
        }
        return (double) tong / diemMonHoc.length;
    }
}
```

3. Lớp DatabaseHelper (Chỉ làm nhiệm vụ lưu trữ dữ liệu)

```
java
public class DatabaseHelper {
    public static void luuThongTinSinhVien(SinhVien sinhVien) {
        System.out.println("Lưu thông tin sinh viên: " + sinhVien.getTen() + " - Mã SV: " + sinhVien.getMaSV());
    }
}
```

Giải thích:

- **Lớp SinhVien** chỉ chịu trách nhiệm lưu trữ thông tin sinh viên.
- **Lớp DiemSinhVien** chịu trách nhiệm tính điểm trung bình.
- **Lớp DatabaseHelper** chịu trách nhiệm làm việc với cơ sở dữ liệu.

TỔNG KẾT NGẮN GỌN VỀ THIẾT KẾ PHƯƠNG THỨC CHO CÁC LỚP

Việc thiết kế phương thức trong lập trình hướng đối tượng cần tuân theo ba nguyên lý quan trọng để đảm bảo hệ thống phần mềm **bảo mật, dễ bảo trì và tái sử dụng hiệu quả**:

1. Đóng gói dữ liệu (Encapsulation):

- Bảo vệ dữ liệu khỏi sự can thiệp trực tiếp từ bên ngoài.
- Kiểm soát truy cập thông qua các phương thức cụ thể.

2. Thực hiện lời gọi nhiều lần (Reusability):

- Thiết kế các phương thức có thể tái sử dụng nhiều lần.
- Giảm trùng lặp mã nguồn, giúp tối ưu hóa hiệu suất lập trình.

3. Hướng trách nhiệm (Single Responsibility):

- Mỗi lớp chỉ nên đảm nhận một nhiệm vụ cụ thể.
- Hệ thống dễ bảo trì, mở rộng và giảm rủi ro thay đổi không cần thiết.

Bài học rút ra:

- Áp dụng tốt ba nguyên lý trên giúp viết code gọn gàng, dễ hiểu, tiết kiệm thời gian phát triển và nâng cao chất lượng phần mềm.

7.3. Xây dựng thẻ CRC cho các lớp

7.3.1. Khái niệm thẻ CRC (Class - Responsibility - Collaboration)

7.3.2. Xây dựng thẻ CRC cho các lớp

□ KHÁI NIỆM:

1. Khái niệm thẻ CRC (Class - Responsibility - Collaboration)

- Thẻ CRC (Class - Responsibility - Collaboration) là một công cụ được sử dụng trong lập trình hướng đối tượng để giúp thiết kế và tổ chức các lớp trong hệ thống.
 - **Class (Lớp):** Đại diện cho một thực thể hoặc đối tượng trong hệ thống.
 - **Responsibility (Trách nhiệm):** Mô tả các nhiệm vụ hoặc chức năng mà lớp đảm nhận.
 - **Collaboration (Cộng tác):** Các lớp khác mà lớp này tương tác để thực hiện chức năng của nó.

2. MỤC ĐÍCH CỦA THẺ CRC

- Giúp xác định rõ trách nhiệm của mỗi lớp trong hệ thống.
- Hỗ trợ việc thiết kế phần mềm theo hướng đối tượng bằng cách định nghĩa các mối quan hệ giữa các lớp.
- Dễ dàng điều chỉnh thiết kế trước khi lập trình.

3. VÍ DỤ VỀ THẺ CRC

- Giả sử chúng ta có một hệ thống quản lý thư viện, trong đó có lớp SinhVien, Sach, và ThuVien.

Class	Responsibility (Trách nhiệm)	Collaboration (Cộng tác)
SinhVien	- Mượn sách, trả sách	Sach, ThuVien
Sach	- Cung cấp thông tin sách, trạng thái	SinhVien, ThuVien
ThuVien	- Quản lý sách, kiểm tra tình trạng mượn	Sach, SinhVien

4. ỨNG DỤNG TRONG THỰC TẾ

- Thẻ CRC giúp nhóm phát triển dễ dàng thảo luận về thiết kế hệ thống trước khi viết mã.
 - Được sử dụng trong các giai đoạn phân tích và thiết kế hệ thống phần mềm.
 - Hỗ trợ việc tạo ra các sơ đồ UML như sơ đồ lớp hoặc sơ đồ tuần tự.
- **Bước tiếp theo:** Dựa vào thông tin này, sinh viên có thể thực hành xây dựng thẻ CRC cho một hệ thống đơn giản như quản lý cửa hàng, đặt vé máy bay hoặc hệ thống bán hàng trực tuyến.

❑ XÂY DỰNG THẺ CRC CHO CÁC LỚP

- Thẻ CRC (Class - Responsibility - Collaboration) là một công cụ hữu ích trong thiết kế hướng đối tượng, giúp mô tả trách nhiệm và sự tương tác giữa các lớp trong hệ thống.

❑ Ví dụ về thẻ CRC cho lớp SinhVien

Class	Responsibility (Trách nhiệm)	Collaboration (Cộng tác với)
Sinhvien	- Lưu trữ thông tin sinh viên	- Lớp MonHoc
	- Quản lý đăng ký môn học	- Lớp LopHoc

❑ HƯỚNG DẪN SINH VIÊN THỰC HÀNH

1.Bước 1: Xác định các lớp cần thiết trong hệ thống.

- Ví dụ: Trong hệ thống quản lý sinh viên, các lớp có thể bao gồm SinhVien, MonHoc, LopHoc, GiangVien, DangKyHoc.

2.Bước 2: Xác định trách nhiệm của từng lớp.

- Mỗi lớp nên có những trách nhiệm cụ thể, không chồng chéo với lớp khác.

3.Bước 3: Xác định các lớp có liên quan (Collaboration).

- Xác định lớp nào cần tương tác với lớp nào để thực hiện chức năng hệ thống.

4.Bước 4: Vẽ thẻ CRC cho từng lớp.

- Sử dụng bảng hoặc sơ đồ để mô tả chi tiết.

7.4. Thiết kế sơ đồ thuật toán cho các phương thức

7.4.1. Khái niệm sơ đồ FSM (Finite State Machine) - statechart

7.4.2. Xây dựng sơ đồ FSM – statechart cho mỗi lớp

Chapter 14

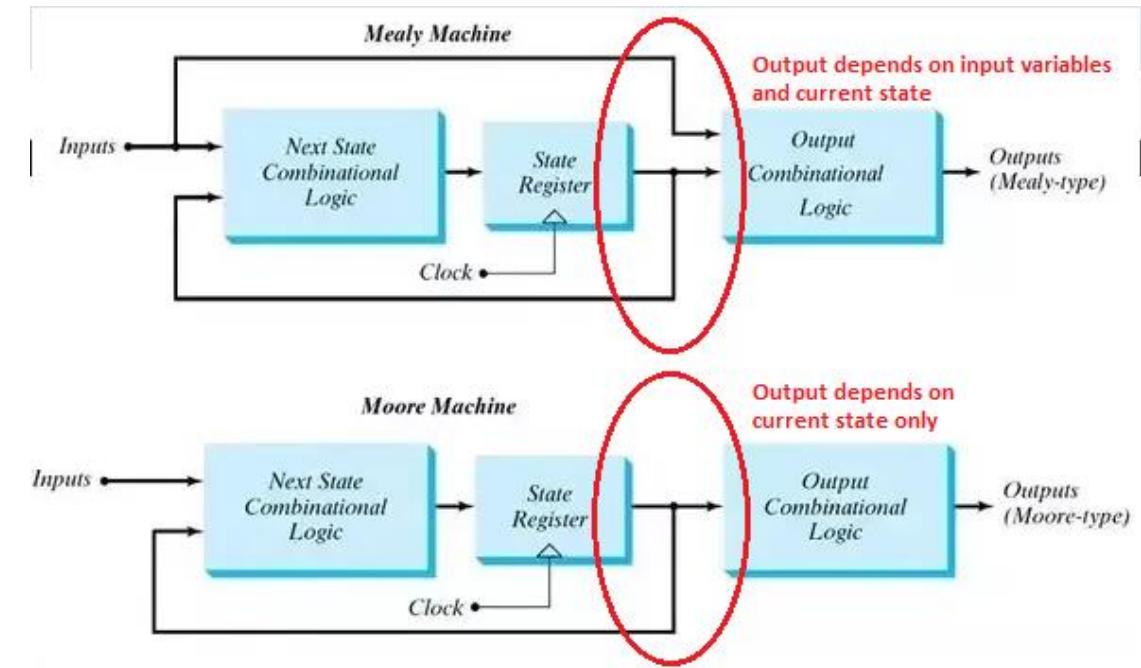
❑ **Máy trạng thái hữu hạn (Finite State Machine - FSM)** là một mô hình toán học được sử dụng để thiết kế và mô tả hành vi của các hệ thống số, đặc biệt trong thiết kế vi mạch số. FSM giúp kiểm soát quá trình hoạt động và dễ dàng debug hệ thống.

❑ **Cấu trúc cơ bản của FSM:**

1. **Mạch tạo trạng thái kế tiếp (Next State Logic):** Xác định trạng thái kế tiếp theo dựa trên trạng thái hiện tại và đầu vào.
2. **Bộ nhớ trạng thái (State Memory):** Lưu trữ trạng thái hiện tại, thường sử dụng các Flip-Flop hoặc Latch.
3. **Mạch tạo đầu ra (Output Logic):** Xác định đầu ra của hệ thống dựa trên trạng thái hiện tại và/hoặc đầu vào.

❑ **Phân loại FSM:**

1. **FSM Moore** là loại có mạch tạo ngõ ra không phụ thuộc trực tiếp vào ngõ vào FSM
2. **FSM Mealy** là loại có mạch tạo ngõ ra phụ thuộc trực tiếp vào ngõ vào FSM



Đặc điểm	FSM Moore	FSM Mealy
Định nghĩa	Đầu ra chỉ phụ thuộc vào trạng thái hiện tại .	Đầu ra phụ thuộc vào trạng thái hiện tại và đầu vào .
Biểu thức đầu ra	$Output = f(state)$	$Output = f(state, input)$
Thay đổi đầu ra	Chỉ thay đổi khi chuyển sang trạng thái mới.	Có thể thay đổi ngay khi đầu vào thay đổi.
Phản ứng với đầu vào	Chậm hơn vì phải đợi thay đổi trạng thái.	Nhanh hơn vì có thể thay đổi ngay khi đầu vào thay đổi.
Thiết kế đơn giản hơn	Dễ thiết kế hơn, ít bị ảnh hưởng bởi nhiều.	Phức tạp hơn do đầu ra phụ thuộc vào đầu vào.
Ứng dụng thực tế	- Bộ điều khiển đồng hồ, bộ mã hóa trạng thái.	- Bộ điều khiển giao tiếp (ví dụ: UART, I2C, SPI).
	- Bộ đếm sự kiện tuần tự.	- Bộ giải mã tín hiệu

Ví dụ minh họa

FSM Moore - Máy bán hàng tự động

- Trạng thái xác định đầu ra.
- Nếu người dùng đưa vào đủ tiền, trạng thái mới sẽ kích hoạt đầu ra để nhả sản phẩm.

FSM Mealy - Bộ giải mã tín hiệu

- Đầu ra thay đổi ngay khi có tín hiệu đầu vào.
- Ví dụ: Bộ điều khiển nhận tín hiệu từ cảm biến, ngay lập tức thay đổi trạng thái của hệ thống.

Yêu cầu thiết kế FSM điều khiển đèn giao thông

- Sử dụng **FSM** (Finite State Machine) để thiết kế hệ thống điều khiển đèn giao thông cho một ngã tư đường, đáp ứng các yêu cầu sau:

1.Thời gian dừng tại mỗi trạng thái của đèn có thể cấu hình trước khi biên dịch RTL code.

2.Tần số điều khiển đèn là 1 Hz, tương ứng với **chu kỳ 1 giây**.

3.Sự chuyển trạng thái đèn được xác định theo bảng quy tắc.

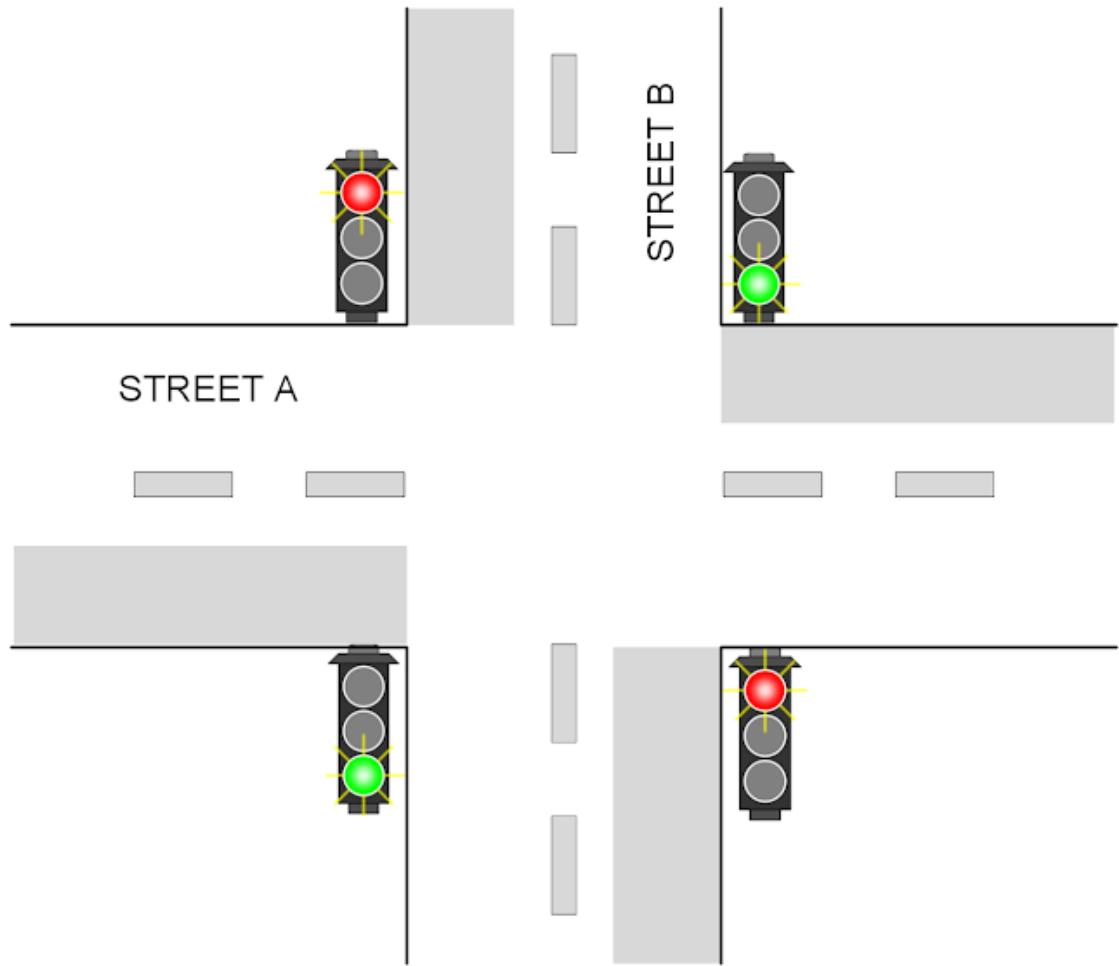
4.Mỗi đường có 3 tín hiệu điều khiển đèn:

- Green (Xanh)**
- Yellow (Vàng)**
- Red (Đỏ)**

5.Trạng thái khởi động: Đèn của cả hai đường **STREET A** và **STREET B** đều ở màu **ĐỎ**.

Bảng giá trị ngõ ra tương ứng với các trạng thái đèn như sau:

STREET A	STREET B	street_a[2:0]	street_b[2:0]
Green	Red	100	001
Yellow	Red	010	001
Red	Red	001	001
Red	Green	001	100
Red	Yellow	001	010
Red	Red	001	001



❑ Example: Traffic Light System

- **States:** Red, Green, Yellow
- **Inputs:** Timer or sensor detection
- **Transitions:**
 - Red → Green (after timer expires)
 - Green → Yellow (after timer expires)
 - Yellow → Red (after timer expires)
- **Outputs:** Traffic lights changing colors.

❑ Applications of FSM

- Digital circuit design
- Control systems (e.g., elevators, vending machines)
- Software development (e.g., workflow automation)
- Network protocols (e.g., TCP/IP state management)

7.5. Hoàn thiện sơ đồ lớp chi tiết

7.5.1. Hoàn thiện thuộc tính cho lớp

7.5.2. Hoàn thiện các phương thức cho lớp

7.5.3. Vẽ lại sơ đồ lớp chi tiết cho hệ thống

- Trong lập trình hướng đối tượng (OOP), **thuộc tính của lớp** là các biến được định nghĩa bên trong lớp, dùng để lưu trữ trạng thái hoặc đặc điểm của đối tượng. Các thuộc tính này thường được khai báo với các mức truy cập như **public**, **private**, **hoặc protected**, xác định phạm vi truy cập của chúng.

Example: Completing Attributes for a Student Class

Attribute	Data Type	Description
studentID	String	Unique identifier for the student.
name	String	Full name of the student.
email	String	Contact email address.
dateOfBirth	Date	Student's date of birth.
enrolledCourses	List<Course>	List of courses the student is enrolled in.

Ghi chú:

- 1.Khai báo các thuộc tính (studentID, name, email, dateOfBirth, enrolledCourses).
- 2.Sử dụng **ArrayList<String>** để lưu danh sách các khóa học đã đăng ký.
- 3.Cung cấp các phương thức **getter và setter** để truy xuất và cập nhật dữ liệu.

```
// Định nghĩa lớp Student
public class Student {
    // Thuộc tính của lớp Student
    private String studentID;
    private String name;
    private String email;
    private Date dateOfBirth;
    private List<String> enrolledCourses;

    // Constructor - Hàm khởi tạo
    public Student(String studentID, String name, String email, Date dateOfBirth) {
        this.studentID = studentID;
        this.name = name;
        this.email = email;
        this.dateOfBirth = dateOfBirth;
        this.enrolledCourses = new ArrayList<>();
    }

    // Getter và Setter cho từng thuộc tính
    public String getStudentID() {
        return studentID;
    }

    public void setStudentID(String studentID) {
        this.studentID = studentID;
    }
}
```

❑ Định nghĩa:

- Trong lập trình hướng đối tượng (OOP), **phương thức** là các hành vi (behavior) hoặc chức năng (function) mà một lớp có thể thực hiện. Việc **hoàn thiện các phương thức cho lớp** là quá trình thiết kế và triển khai các phương thức sao cho mỗi phương thức thực hiện đúng một nhiệm vụ cụ thể, giúp lớp hoạt động một cách hiệu quả và dễ bảo trì.

❑ Nguyên tắc thiết kế phương thức tốt

- Chỉ thực hiện một nhiệm vụ cụ thể** (Single Responsibility Principle – SRP).
- Đặt tên phương thức rõ ràng, dễ hiểu**
- Hạn chế sử dụng quá nhiều tham số đầu vào**
- Sử dụng từ khóa private nếu phương thức chỉ dùng nội bộ trong lớp**
- Tránh lặp lại code bằng cách tái sử dụng phương thức**

Methods:

```
java
// Getters and Setters
public String getStudentID() {
    return studentID;
}

public void setName(String name) {
    this.name = name;
}

public void setEmail(String email) {
    this.email = email;
}
```

Copy Edit

❑ Tóm tắt:

- Hoàn thiện phương thức cho lớp** là quá trình thiết kế các hành vi của đối tượng.
- Các phương thức cần có trách nhiệm **rõ ràng, cụ thể, dễ bảo trì**.
- Ví dụ Java** giúp minh họa cách tổ chức và thiết kế phương thức hiệu quả.



Bài 1: Hoàn thiện lớp Student

- **Yêu cầu:**
 - Viết một lớp Student với các thuộc tính:
 - studentID: Mã sinh viên (String)
 - name: Họ và tên (String)
 - gpa: Điểm trung bình (double)
 - enrolledCourses: Danh sách khóa học đã đăng ký (List<String>)
 - **Cài đặt các phương thức:**
 - **displayInfo()**: Hiển thị thông tin sinh viên.
 - **updateGPA(double newGPA)**: Cập nhật điểm trung bình của sinh viên.
 - **enrollCourse(String courseName)**: Thêm một khóa học vào danh sách đăng ký.
 - **isEligibleForScholarship()**: Kiểm tra xem sinh viên có đủ điều kiện nhận học bổng không (GPA ≥ 3.5).

Bài 2: Mở rộng lớp Student

- Yêu cầu:
- Viết phương thức compareGPA(Student otherStudent): So sánh điểm trung bình của sinh viên hiện tại với một sinh viên khác.
- Viết phương thức dropCourse(String courseName): Xóa một khóa học khỏi danh sách đăng ký nếu có.

Hướng dẫn:

java

 Copy  Edit

```
public void dropCourse(String courseName) {  
    if (enrolledCourses.contains(courseName)) {  
        enrolledCourses.remove(courseName);  
        System.out.println(name + " đã hủy khóa học: " + courseName);  
    } else {  
        System.out.println("Khóa học không tồn tại trong danh sách!");  
    }  
}
```

Bài 3: Xây dựng lớp BankAccount

- Yêu cầu:
- Viết một lớp BankAccount có các thuộc tính:
 - accountNumber: Số tài khoản (String)
 - balance: Số dư (double)
 - accountHolder: Chủ tài khoản (String)
- Cài đặt các phương thức:
 - deposit(double amount): Nạp tiền vào tài khoản.
 - withdraw(double amount): Rút tiền nếu số dư đủ.
 - displayBalance(): Hiển thị số dư tài khoản.

Bài 4: Xây dựng lớp LibraryBook

Yêu cầu:

- Viết một lớp LibraryBook có các thuộc tính:
 - bookID: Mã sách (String)
 - title: Tiêu đề sách (String)
 - author: Tác giả (String)
 - isBorrowed: Trạng thái mượn sách (boolean)
- Cài đặt các phương thức:
 - borrowBook(): Đánh dấu sách là đã mượn (isBorrowed = true).
 - returnBook(): Đánh dấu sách là đã trả (isBorrowed = false).
 - displayInfo(): Hiển thị thông tin sách.

java

```
public void borrowBook() {  
    if (!isBorrowed) {  
        isBorrowed = true;  
        System.out.println("Bạn đã mượn sách: " + title);  
    } else {  
        System.out.println("Sách đã được mượn!");  
    }  
}
```

Bài 5: Ứng dụng lớp Student trong thực tế

 **Tình huống:** Bạn là một lập trình viên phát triển hệ thống quản lý sinh viên cho trường đại học. Hệ thống cần:

- Cho phép sinh viên đăng ký và hủy khóa học.
- Cập nhật điểm trung bình của sinh viên.
- Kiểm tra xem sinh viên có đạt học bổng hay không.
- So sánh điểm của hai sinh viên.



Yêu cầu:

- Cài đặt các chức năng trên bằng Java.
- Viết chương trình main() để tạo **danh sách sinh viên** và thực hiện các thao tác.

Vẽ lại sơ đồ lớp chi tiết cho hệ thống

Hoàn thiện sơ đồ lớp chi tiết, bao gồm đầy đủ các thuộc tính, phương thức và các mối quan hệ giữa các lớp.

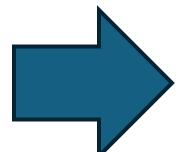
Dùng UML vẽ Class Diagram và Sequence Diagram

Classes and Details

Class	Attributes	Methods
Student	studentID, name, email	registerCourse(), dropCourse(), getEnrolledCourses()
Course	courseID, courseName, credits	addStudent(), removeStudent(), listStudents()
Teacher	teacherID, name, department	assignCourse(), listCourses()
Enrollment	enrollmentID, studentID, courseID	createEnrollment(), cancelEnrollment()

Note:

- **Class Diagram:** Defines relationships between Student, Course, Teacher, and Enrollment.
- **Sequence Diagram:** Describes step-by-step interactions for a student registering for a course.



□ Thiết Kế Cơ Sở Dữ Liệu

- Dựa trên sơ đồ lớp đã hoàn thiện để xác định các bảng trong cơ sở dữ liệu, mỗi lớp thực thể tương ứng với một bảng dữ liệu.

• Ví dụ:

Ghi chú:

- **Khóa chính (Primary Key):** Định danh duy nhất của một bản ghi trong bảng.
- **Khóa ngoại (Foreign Key):** Được sử dụng để liên kết bảng này với bảng khác.
- **Ràng buộc (Constraints):**
 - **NOT NULL:** Trường bắt buộc nhập giá trị.
 - **AUTO_INCREMENT:** Tự động tăng giá trị cho khóa chính.
- Viết câu lệnh SQL để tạo bảng Course và Enrollment.
- Thêm một số dữ liệu mẫu vào các bảng này.
- Viết truy vấn SQL để lấy danh sách sinh viên đăng ký một khóa học cụ thể.

Example: Other Tables Based on Class Diagram

Table: Course

Column Name	Data Type	Description	Constraints
courseID	VARCHAR(10)	Unique identifier for the course.	Primary Key
courseName	VARCHAR(100)	Name of the course.	NOT NULL
credits	INT	Number of credits for the course.	NOT NULL

Table: Enrollment

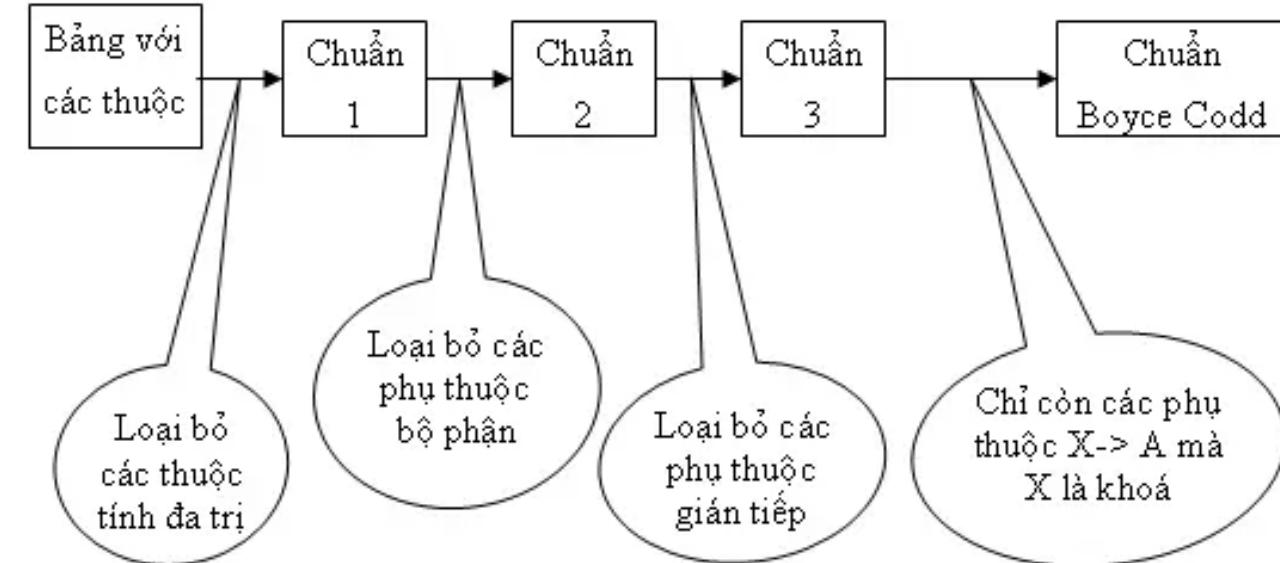
Column Name	Data Type	Description	Constraints
enrollmentID	INT AUTO_INCREMENT	Unique identifier for the enrollment.	Primary Key
studentID	VARCHAR(10)	Foreign key referencing SinhVien(studentID).	Foreign Key
courseID	VARCHAR(10)	Foreign key referencing Course(courseID).	Foreign Key
enrollmentDate	DATE	Date of enrollment.	NOT NULL

Mục tiêu:

Loại bỏ dư thừa dữ liệu và đảm bảo tính nhất quán
trong cơ sở dữ liệu.

 Các bước chuẩn hóa:

- Đưa các bảng về dạng 1NF (loại bỏ các thuộc tính lặp).
- Đưa về dạng 2NF (loại bỏ phụ thuộc một phần).
- Đưa về dạng 3NF (loại bỏ phụ thuộc bắc cầu).



Các dạng chuẩn hóa (Normal Form)

Ghi chú:

- **1NF:** Loại bỏ các giá trị lặp, mỗi ô chứa một giá trị duy nhất.
- **2NF:** Xác định khóa chính hợp lệ, loại bỏ phụ thuộc một phần.
- **3NF:** Loại bỏ các phụ thuộc bắc cầu giữa các thuộc tính.

Ví dụ 1: Dữ liệu thông tin sinh viên đăng ký môn học

- Dữ liệu ban đầu (Chưa chuẩn hóa)

StudentID	StudentName	Course	Instructor	InstructorPhone
001	Nam	Math	Mr. A	123456789
001	Nam	Physics	Mr. B	987654321
002	Lan	Math	Mr. A	123456789



Yêu cầu:
Hãy chuẩn hóa cơ sở dữ liệu về dạng (1NF, 2NF, 3NF)

□ THIẾT KẾ THUỘC TÍNH CHO CÁC LỚP TRONG HỆ THỐNG

Xác định đầy đủ các thuộc tính cần thiết cho từng lớp.

Ví dụ: Lớp KhachHang có các thuộc tính: maKhachHang, hoTen, diaChi, soDienThoai.

Example: Class KhachHang (**Customer**)

Attributes

Attribute Name	Data Type	Description	Constraints
maKhachHang	VARCHAR(10)	Unique identifier for the customer.	Primary Key
hoTen	VARCHAR(50)	Full name of the customer.	NOT NULL
diaChi	VARCHAR(100)	Customer's address.	NULABLE
soDienThoai	VARCHAR(15)	Contact phone number.	UNIQUE, NOT NULL
email	VARCHAR(100)	Email address of the customer.	NULABLE, UNIQUE
ngayDangKy	DATE	Registration date of the customer.	NOT NULL

□ Thiết kế phương thức cho các lớp trong hệ thống

- Thiết kế các phương thức cần thiết cho từng lớp, đảm bảo mỗi phương thức thực hiện một trách nhiệm cụ thể.

Ví dụ:

- Phương thức `dangKy()` cho lớp `KhachHang`.
- Phương thức `themSanPham()` cho lớp `DonHang`.

Class: `KhachHang` (Customer)

Method Name	Parameters	Return Type	Description
<code>dangKy()</code>	<code>String hoTen, String email, String soDienThoai</code>	<code>boolean</code>	Registers a new customer in the system.
<code>capNhatThongTin()</code>	<code>String hoTen, String diaChi, String soDienThoai</code>	<code>void</code>	Updates the customer's information.
<code>xemDonHang()</code>	<code>None</code>	<code>List<DonHang></code>	Retrieves all orders made by the customer.

Class: `DonHang` (Order)

Method Name	Parameters	Return Type	Description
<code>themSanPham()</code>	<code>SanPham sanPham, int soLuong</code>	<code>boolean</code>	Adds a product to the order.
<code>xoaSanPham()</code>	<code>SanPham sanPham</code>	<code>boolean</code>	Removes a product from the order.
<code>tinhTongTien()</code>	<code>None</code>	<code>double</code>	Calculates the total cost of the order.

□ Thiết kế CSDL cho hệ thống

- Dựa trên sơ đồ lớp để xây dựng các bảng và mối quan hệ giữa các bảng trong cơ sở dữ liệu.

Steps to Design the Database

1. Identify Tables:
 - Each class in the class diagram corresponds to a table in the database.
 - Example: `KhachHang` class becomes the `KhachHang` table.
2. Define Columns:
 - Attributes of the class become columns in the table.
 - Specify the appropriate data types for each attribute.
 - Add constraints such as `Primary Key`, `Unique`, `Not Null`, etc.
3. Establish Relationships:
 - Define foreign keys to represent associations between tables.
 - Relationships:
 - **One-to-Many:** E.g., One `KhachHang` can place many `DonHang`.
 - **Many-to-Many:** E.g., Many `SanPham` can belong to many `DonHang` (via a join table).
4. Normalize Tables:
 - Ensure normalization to avoid redundancy and maintain data integrity.
 - Normalize to at least 3NF (Third Normal Form).

☐ Hoàn thiện sơ đồ lớp chi tiết cho hệ thống

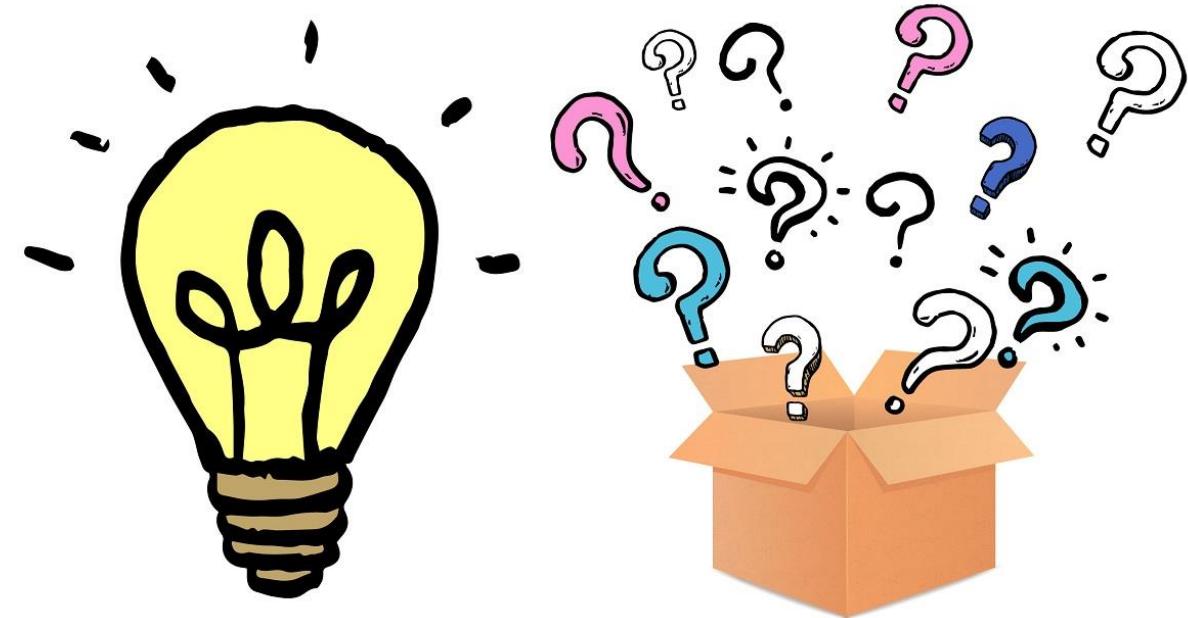
- Bổ sung đầy đủ các thuộc tính và phương thức cho mỗi lớp.
- Vẽ lại sơ đồ lớp chi tiết bao gồm các quan hệ giữa các lớp.

Completing a Detailed Class Diagram for the System

To create a detailed class diagram:

1. **Include all attributes:**
 - Add all the necessary properties for each class.
 - Ensure attributes align with the role of the class.
2. **Define all methods:**
 - Include methods that allow the class to perform its responsibilities.
 - Ensure methods are specific and follow the Single Responsibility Principle.
3. **Add relationships:**
 - Clearly indicate associations, aggregations, compositions, and generalizations between classes.

1. CÂU HỎI TRẮC NGHIỆM
2. CÂU HỎI TRẢ LỜI NGẮN
3. CÂU HỎI THẢO LUẬN NHÓM
4. CÂU HỎI TÌNH HUỐNG



- 1. Câu hỏi 1:** Thuộc tính của một lớp nên được thiết kế như thế nào để đảm bảo tính đóng gói dữ liệu?
A. Public
B. Private
C. Protected
D. Internal

- 2. Câu hỏi 2:** Nguyên lý nào đảm bảo rằng dữ liệu chỉ có thể được truy cập thông qua các phương thức công khai của lớp?
A. Hướng trách nhiệm
B. Đóng gói dữ liệu
C. Thực hiện lời gọi nhiều lần
D. Kế thừa

- 3. Câu hỏi 3:** Thẻ CRC bao gồm những thành phần nào?
A. Class, Relation, Composition
B. Class, Responsibility, Collaboration
C. Class, Reference, Control
D. Class, Role, Connection

- 4. Câu hỏi 4:** Quan hệ nào giữa các lớp thể hiện việc một lớp là thành phần của lớp khác và không thể tồn tại độc lập?
A. Association
B. Aggregation
C. Composition
D. Inheritance

- 5. Câu hỏi 5:** Sơ đồ FSM mô tả điều gì?
A. Cấu trúc dữ liệu của hệ thống
B. Các trạng thái và chuyển đổi giữa các trạng thái của đối tượng
C. Các lớp và quan hệ giữa các lớp
D. Cách người dùng tương tác với hệ thống

- 6. Câu hỏi 6:** Nguyên lý nào khuyến khích việc thiết kế phương thức để tái sử dụng nhiều lần trong các ngữ cảnh khác nhau?
- A. Hướng trách nhiệm
 - B. Thực hiện lời gọi nhiều lần
 - C. Đóng gói dữ liệu
 - D. Phân lớp
- 7. Câu hỏi 7:** Chuẩn hóa quan hệ giữa các bảng về dạng 3-NF nhằm mục đích gì?
- A. Tăng tốc độ xử lý dữ liệu
 - B. Giảm thiểu dư thừa dữ liệu và đảm bảo tính nhất quán
 - C. Đảm bảo các bảng chứa đầy đủ dữ liệu
 - D. Tăng khả năng mở rộng của hệ thống
- 8. Câu hỏi 8:** Lớp nào trong hệ thống thường chứa các thuộc tính lưu trữ dữ liệu và không chứa nhiều logic xử lý?
- A. Lớp điều khiển
 - B. Lớp biên
 - C. Lớp thực thể
 - D. Lớp giao diện
- 9. Câu hỏi 9:** Khi xây dựng sơ đồ lớp chi tiết, điều gì cần được bổ sung?
- A. Thêm các quan hệ giữa các lớp
 - B. Thêm các thuộc tính và phương thức đầy đủ cho từng lớp
 - C. Thêm giao diện người dùng
 - D. Thêm các sơ đồ tuần tự
- 10. Câu hỏi 10:** Trong mô hình cơ sở dữ liệu quan hệ, một thuộc tính của bảng được gọi là:
- A. Field
 - B. Row
 - C. Record
 - D. Table

- 1.Tại sao cần đóng gói dữ liệu khi thiết kế lớp?
- 2.Nguyên lý hướng trách nhiệm trong thiết kế phương thức là gì?
- 3.Thẻ CRC là gì?
- 4.Quan hệ Aggregation và Composition khác nhau như thế nào?
- 5.FSM (Finite State Machine) là gì?
- 6.Mục tiêu của việc chuẩn hóa cơ sở dữ liệu là gì?
- 7.Lớp điều khiển có vai trò gì trong hệ thống?
- 8.Nguyên lý thực hiện lời gọi nhiều lần giúp ích gì trong thiết kế phương thức?
- 9.Sơ đồ lớp là gì?
- 10.Dạng chuẩn 3-NF của cơ sở dữ liệu là gì?

1. Thảo luận về vai trò của việc đóng gói dữ liệu khi thiết kế lớp.
2. So sánh giữa Aggregation và Composition trong thiết kế lớp.
3. Thảo luận về cách xây dựng thẻ CRC hiệu quả cho hệ thống.
4. Tại sao cần xây dựng sơ đồ FSM cho các lớp trong hệ thống?
5. Thảo luận về các bước chuẩn hóa cơ sở dữ liệu và vai trò của từng bước.
6. Làm thế nào để đảm bảo rằng các phương thức của lớp tuân thủ nguyên lý hướng trách nhiệm?
7. Thảo luận về ưu và nhược điểm của việc sử dụng sơ đồ lớp chi tiết trong thiết kế hệ thống.
8. Tại sao cần chuẩn hóa cơ sở dữ liệu đến dạng 3-NF?
9. Thảo luận về tầm quan trọng của lớp điều khiển trong mô hình MVC.
10. Làm thế nào để xây dựng sơ đồ lớp chi tiết cho một hệ thống lớn và phức tạp?

1. Trong quá trình thiết kế hệ thống quản lý sinh viên, bạn nhận ra rằng nhiều lớp có các thuộc tính giống nhau. Bạn sẽ xử lý vấn đề này như thế nào?
2. Sau khi hoàn thành sơ đồ lớp, nhóm phát triển phát hiện thiếu một số phương thức cần thiết. Làm thế nào để bổ sung vào sơ đồ lớp?
3. Một nhóm phát triển gặp khó khăn trong việc phân biệt giữa Aggregation và Composition khi thiết kế sơ đồ lớp. Làm thế nào để giúp nhóm giải quyết vấn đề này?
4. Trong quá trình xây dựng thẻ CRC, nhóm phát triển phát hiện nhiều lớp có trách nhiệm trùng lặp. Làm thế nào để xử lý tình huống này?
5. Khách hàng yêu cầu thêm một chức năng mới sau khi sơ đồ lớp đã được hoàn thiện. Nhóm phát triển cần làm gì để cập nhật sơ đồ lớp?
6. Khi xây dựng sơ đồ FSM cho lớp DonHang, nhóm phát triển gặp khó khăn trong việc xác định các trạng thái và sự kiện. Bạn sẽ hướng dẫn nhóm như thế nào?
7. Trong quá trình chuẩn hóa cơ sở dữ liệu, nhóm phát triển gặp vấn đề khi các bảng chứa nhiều dữ liệu dư thừa. Làm thế nào để xử lý vấn đề này?
8. Một nhóm phát triển gặp khó khăn khi thiết kế các phương thức cho lớp vì chưa hiểu rõ nguyên lý hướng trách nhiệm. Làm thế nào để hướng dẫn nhóm áp dụng nguyên lý này?
9. Trong quá trình xây dựng sơ đồ lớp chi tiết, khách hàng yêu cầu thay đổi một số yêu cầu đã thống nhất trước đó. Nhóm phát triển nên xử lý như thế nào?
10. Khi kiểm tra lại sơ đồ lớp, nhóm phát triển nhận thấy một số quan hệ giữa các lớp bị sai. Làm thế nào để sửa lại sơ đồ lớp mà không ảnh hưởng đến tiến độ dự án?

SWE | 08.CÀI ĐẶT VÀ KIỂM THỬ

8.1. CODE CONVENTION

8.2. TEST UNIT

8.3. TÍCH HỢP

8.4. TEST CASE

8.5. CASE STUDY

8.6 CÂU HỎI CÙNG CỐ LÝ THUYẾT



Đọc tài liệu CHƯƠNG 8

Thank You

Q&A

1. Technical support engineer
2. **Software Developer**
3. Quality assurance engineer
4. Software architect
5. **Software project manager**
6. Hardware engineer
7. **A Data scientist**
8. Information security analyst
9. **Technical writer**
10. Machine learning engineer

