CHƯƠNG 3: KIẾN TRÚC VÀ QUY TRÌNH PHÁT TRIỂN PHẦN MỀM

3.1. Vai trò của kiến trúc phần mềm

3.2. Các kiểu kiến trúc cơ bản

3.3. Chuẩn bị tài liệu

3.4. Các mô hình quy trình phát triển phần mềm

3.1. Vai trò của kiến trúc phần mềm

- **Định nghĩa kiến trúc phần mềm**
- Tầm quan trọng của kiến trúc phần mềm
- Mối quan hệ giữa kiến trúc phần mềm và quy trình phát triển phần mềm

Định nghĩa kiến trúc phần mềm

- Kiến trúc phần mềm là tổng thể **cấu trúc** của một hệ thống phần mềm, bao gồm các **thành phần** phần mềm, **mối quan hệ** giữa chúng và **môi trường thực thi**.
- Kiến trúc phần mềm không chỉ là việc chọn công nghệ mà còn là việc định nghĩa các mô hình tương tác và trách nhiệm của từng thành phần trong hệ thống.
- Kiến trúc phần mềm như một phương tiện giao tiếp giữa các nhóm phát triển, quản lý và khách hàng về cấu trúc và chức năng của hệ thống.Xây dựng sự hiểu biết chung về mục tiêu dự án và công nghệ giữa các nhóm.

3.1. Vai trò của kiến trúc phần mềm

- Định nghĩa kiến trúc phần mềm
- **Tầm quan trọng của kiến trúc phần mềm**
- Mối quan hệ giữa kiến trúc phần mềm và quy trình phát triển phần mềm

Tầm quan trọng của Kiến trúc phần mềm

- Định hướng phát triển: Kiến trúc phần mềm cung cấp một khung làm việc rõ ràng, giúp định hướng cho các quyết định thiết kế và phát triển trong suốt vòng đời của dự án.
- Tối ưu hóa hiệu suất: Một kiến trúc tốt giúp tối ưu hóa hiệu suất của hệ thống, đảm bảo rằng các thành phần hoạt động hiệu quả và có thể mở rộng khi cần thiết.
- Quản lý độ phức tạp: Kiến trúc phần mềm giúp quản lý độ phức tạp của hệ thống bằng cách chia nhỏ thành các thành phần dễ quản lý và phát triển.

Tầm quan trọng của Kiến trúc phần mềm

- Đảm bảo tính linh hoạt: Kiến trúc tốt cho phép hệ thống dễ dàng thích ứng với các thay đổi trong yêu cầu hoặc công nghệ.
- Tăng cường bảo mật: Bằng cách xác định rõ ràng các điểm yếu và thiết kế các biện pháp bảo vệ, kiến trúc phần mềm giúp tăng cường bảo mật cho hệ thống.

3.1. Vai trò của kiến trúc phần mềm

- Định nghĩa kiến trúc phần mềm
- Tầm quan trọng của kiến trúc phần mềm
- **Mối quan hệ giữa kiến trúc phần mềm và quy trình phát triển phần mềm**

Mối quan hệ giữa kiến trúc và quy trình phát triển phần mềm

- Là một sự tương tác chặt chẽ, đóng vai trò quyết định đối với chất lượng, tính ổn định, và khả năng bảo trì của phần mềm. Thể hiện:
  - Kiến trúc phần mềm định hướng quy trình phát triển
  - Quy trình phát triển phải tuân theo kiến trúc
  - Kiến trúc phản ánh yêu cầu và ràng buộc trong quy trình phát triển
  - Phản hồi từ quy trình phát triển ảnh hưởng kiến trúc
  - Phân chia công việc và đội ngũ dựa trên kiến trúc

Mối quan hệ giữa kiến trúc và quy trình phát triển phần mềm

- Kiến trúc phần mềm định hướng quy trình phát triển:
  - Kiến trúc phần mềm là nền tảng của hệ thống, bao gồm việc xác định các thành phần chính và cách chúng tương tác với nhau. Ảnh hưởng trực tiếp đến cách nhóm phát triển sẽ tiến hành công việc.
  - Ví dụ, một kiến trúc phân lớp có thể quy định cách các nhóm phát triển các lớp khác nhau (như giao diện người dùng, logic nghiệp vụ, và cơ sở dữ liệu) riêng rẽ nhưng đồng thời phối hợp với nhau.

Mối quan hệ giữa kiến trúc và quy trình phát triển phần mềm

- Quy trình phát triển phải tuân theo kiến trúc:
  - Quy trình phát triển phần mềm, cho dù theo phương pháp Agile hay Waterfall, đều cần phải thích ứng và tuân thủ các nguyên tắc do kiến trúc phần mềm đề ra.
  - Điều này đảm bảo rằng hệ thống được phát triển theo một cách nhất quán, giảm thiểu lỗi phát sinh và giúp dễ bảo trì hơn.

Mối quan hệ giữa kiến trúc và quy trình phát triển phần mềm

- Kiến trúc phản ánh yêu cầu và ràng buộc trong quy trình phát triển:
  - Các quyết định về kiến trúc phần mềm thường phản ánh các yêu cầu và ràng buộc kỹ thuật, chẳng hạn như khả năng mở rộng, hiệu suất, hoặc khả năng bảo trì.
  - Quy trình phát triển cần phải tuân thủ những yêu cầu này để đảm bảo kiến trúc phần mềm được hiện thực hóa một cách đúng đắn.

Mối quan hệ giữa kiến trúc và quy trình phát triển phần mềm

- Phản hồi từ quy trình phát triển ảnh hưởng kiến trúc:
  - Trong các dự án lớn hoặc phức tạp, việc triển khai có thể tạo ra những phản hồi làm thay đổi kiến trúc ban đầu.
  - Ví dụ, trong một quy trình phát triển theo phương pháp Agile, các giai đoạn thử nghiệm và phản hồi liên tục có thể khiến nhóm phát triển điều chỉnh kiến trúc để phù hợp với thực tế hơn.

Mối quan hệ giữa kiến trúc và quy trình phát triển phần mềm

- Phân chia công việc và đội ngũ dựa trên kiến trúc:
  - Kiến trúc phần mềm thường định hình cách tổ chức nhóm phát triển.
  - Với kiến trúc mô-đun, các đội ngũ có thể được phân chia theo các mô-đun hoặc thành phần riêng biệt, giúp tăng hiệu quả phát triển và giảm thiểu sự chồng chéo.

3.2. Các kiểu kiến trúc cơ bản

- **Kiến trúc MVC MVC (Model-View-Controller).**
- Kiến trúc tầng (Layered Architecture)
- Kiến trúc hướng dịch vụ (Service-Oriented Architecture - SOA)
- Kiến trúc vi dịch vụ (Microservices Architecture)
- Kiến trúc ống và bộ lọc (Pipe and Filter Architecture)
- Kiến trúc sự kiện (Event-Driven Architecture)
- Kiến trúc không máy chủ (Serverless Architecture)

Kiến trúc MVC MVC (Model-View-Controller).

- MVC là một trong những kiến trúc phổ biến nhất trong phát triển phần mềm, đặc biệt là trong các ứng dụng web. Tách biệt ứng dụng thành 3 phần:
  - Model: Quản lý dữ liệu và logic nghiệp vụ. Thực hiện các thao tác liên quan đến dữ liệu và thông báo cho View và Controller khi có sự thay đổi.
  - View: Hiển thị dữ liệu cho người dùng. Nhận dữ liệu từ Model và định dạng nó để hiển thị.
  - Controller: Đóng vai trò trung gian giữa Model và View. Nhận yêu cầu từ người dùng, xử lý chúng, và cập nhật View.

Kiến trúc MVC MVC (Model-View-Controller).

- Một số ví dụ tiêu biểu về các phần mềm và nền tảng sử dụng kiến trúc MVC, cho phép phát triển các ứng dụng web theo mô hình MVC: ASP.NET MVC, Spring MVC, AngularJS, Laravel, Symfony…
- Ưu điểm:
  - Tách biệt mối quan tâm, dễ bảo trì và mở rộng.
  - Tăng khả năng tái sử dụng mã nguồn.
  - Dễ dàng kiểm thử từng phần riêng biệt.
- Nhược điểm:
  - Có thể phức tạp hơn cho các ứng dụng nhỏ.
  - Đòi hỏi sự hiểu biết sâu sắc về mô hình để triển khai hiệu quả.

3.2. Các kiểu kiến trúc cơ bản

- Kiến trúc MVC MVC (Model-View-Controller).
- **Kiến trúc tầng (Layered Architecture)**
- Kiến trúc hướng dịch vụ (Service-Oriented Architecture - SOA)
- Kiến trúc vi dịch vụ (Microservices Architecture)
- Kiến trúc ống và bộ lọc (Pipe and Filter Architecture)
- Kiến trúc sự kiện (Event-Driven Architecture)
- Kiến trúc không máy chủ (Serverless Architecture)

Kiến trúc tầng (Layered Architecture)

- Cấu trúc: Hệ thống được chia thành nhiều tầng, mỗi tầng có một chức năng cụ thể và chỉ tương tác với tầng liền kề.
- Mục đích để tách biệt các chức năng như giao diện người dùng, logic nghiệp vụ, và truy cập dữ liệu. Điều này giúp dễ dàng quản lý và mở rộng hệ thống khi cần thiết.
- VD: Ứng dụng Ngân hàng Trực tuyến, Hệ thống Quản lý Tài nguyên Doanh nghiệp (ERP), Ứng dụng Quản lý Học tập Trực tuyến (LMS)….

Kiến trúc tầng (Layered Architecture)

- Các tầng phổ biến:
  - Tầng trình bày (Presentation Layer): Giao diện người dùng và xử lý đầu vào.
  - Tầng logic nghiệp vụ (Business Logic Layer): Xử lý các quy tắc và logic nghiệp vụ.
  - Tầng dữ liệu (Data Access Layer): Quản lý truy cập và thao tác dữ liệu.

Kiến trúc tầng (Layered Architecture)

- Ưu điểm: Dễ bảo trì, phát triển và kiểm thử.
- Nhược điểm: Có thể gây ra độ trễ do phải qua nhiều tầng. Khó khăn trong việc thay đổi 1 tầng mà không ảnh hưởng đến tầng khác

3.2. Các kiểu kiến trúc cơ bản

- Kiến trúc MVC MVC (Model-View-Controller).
- Kiến trúc tầng (Layered Architecture)
- **Kiến trúc hướng dịch vụ (Service-Oriented Architecture - SOA)**
- Kiến trúc vi dịch vụ (Microservices Architecture)
- Kiến trúc ống và bộ lọc (Pipe and Filter Architecture)
- Kiến trúc sự kiện (Event-Driven Architecture)
- Kiến trúc không máy chủ (Serverless Architecture)

Kiến trúc hướng dịch vụ (Service-Oriented Architecture - SOA)

- Cấu trúc: Hệ thống được xây dựng từ các dịch vụ độc lập, có thể giao tiếp qua các giao thức chuẩn như HTTP, SOAP.
- Đặc điểm: Các dịch vụ có thể được phát triển và triển khai độc lập.
- VD: Oracle SOA Suite, PayPal,….
- Ưu điểm: Tái sử dụng dịch vụ, dễ mở rộng, hỗ trợ phân tán và tích hợp.
- Nhược điểm: Phức tạp trong quản lý và bảo mật. Phức tạp trong việc triển khai. Đòi hỏi sự hiểu biết về các giao thức và tiêu chuẩn dịch vụ.

3.2. Các kiểu kiến trúc cơ bản

- Kiến trúc MVC MVC (Model-View-Controller).
- Kiến trúc tầng (Layered Architecture)
- Kiến trúc hướng dịch vụ (Service-Oriented Architecture - SOA)
- **Kiến trúc vi dịch vụ (Microservices Architecture)**
- Kiến trúc ống và bộ lọc (Pipe and Filter Architecture)
- Kiến trúc sự kiện (Event-Driven Architecture)
- Kiến trúc không máy chủ (Serverless Architecture)

Kiến trúc vi dịch vụ (Microservices Architecture)

- Cấu trúc: Hệ thống được chia thành các dịch vụ nhỏ, mỗi dịch vụ thực hiện một chức năng cụ thể và có thể triển khai độc lập.
- VD: Uber, Spotify, Twitter
- Tăng khả năng mở rộng và linh hoạt.
  - Dễ dàng triển khai và phát triển độc lập.
  - Tăng khả năng chịu lỗi.
- Nhược điểm:
  - Phức tạp trong quản lý và giám sát.
  - Đòi hỏi sự hiểu biết về các công cụ và kỹ thuật triển khai phân tán.

3.2. Các kiểu kiến trúc cơ bản

- Kiến trúc MVC MVC (Model-View-Controller).
- Kiến trúc tầng (Layered Architecture)
- Kiến trúc hướng dịch vụ (Service-Oriented Architecture - SOA)
- Kiến trúc vi dịch vụ (Microservices Architecture)
- Kiến trúc ống và bộ lọc (Pipe and Filter Architecture)
- **Kiến trúc sự kiện (Event-Driven Architecture)**
- Kiến trúc không máy chủ (Serverless Architecture)

Kiến trúc sự kiện (Event-Driven Architecture)

- Cấu trúc: Ứng dụng phản ứng với các sự kiện, cho phép xử lý không đồng bộ
- Đặc điểm: Thường sử dụng các hàng đợi hoặc luồng sự kiện để giao tiếp giữa các thành phần.
- VD: LinkedIn, Netflix, Amazon, Uber…..

Kiến trúc sự kiện (Event-Driven Architecture)

- Ưu điểm:
  - Tăng khả năng mở rộng và linh hoạt.
  - Phù hợp với các ứng dụng thời gian thực.
  - Giảm độ trễ và tăng khả năng phản ứng.
- Nhược điểm: Phức tạp trong việc quản lý và giám sát. Khó khăn trong việc đảm bảo tính nhất quán dữ liệu.

3.2. Các kiểu kiến trúc cơ bản

- **Kiến trúc MVC MVC (Model-View-Controller).**
- Kiến trúc tầng (Layered Architecture)
- Kiến trúc hướng dịch vụ (Service-Oriented Architecture - SOA)
- Kiến trúc vi dịch vụ (Microservices Architecture)
- **Kiến trúc ống và bộ lọc (Pipe and Filter Architecture)**
- Kiến trúc sự kiện (Event-Driven Architecture)
- Kiến trúc không máy chủ (Serverless Architecture)

Kiến trúc ống và bộ lọc (Pipe and Filter Architecture)

- Mô tả: Dữ liệu được xử lý qua một chuỗi các bộ lọc, mỗi bộ lọc thực hiện một thao tác cụ thể và chuyển dữ liệu đến bộ lọc tiếp theo.
- VD: Hệ điều hành Unix và Linux sử dụng kiến trúc ống và bộ lọc trong các tiện ích dòng lệnh để xử lý dữ liệu theo chuỗi , Adobe Photoshop có thể áp dụng một chuỗi các bộ lọc và hiệu ứng lên hình ảnh.

Kiến trúc ống và bộ lọc (Pipe and Filter Architecture)

- Ưu điểm:
  - Dễ dàng mở rộng và tái sử dụng các bộ lọc.
  - Tăng khả năng bảo trì và kiểm thử.
- Nhược điểm:
  - Có thể dẫn đến hiệu suất kém do sự phụ thuộc giữa các bộ lọc.
  - Khó khăn trong việc quản lý luồng dữ liệu phức tạp.

3.2. Các kiểu kiến trúc cơ bản

- **Kiến trúc MVC MVC (Model-View-Controller).**
- Kiến trúc tầng (Layered Architecture)
- Kiến trúc hướng dịch vụ (Service-Oriented Architecture - SOA)
- Kiến trúc vi dịch vụ (Microservices Architecture)
- Kiến trúc ống và bộ lọc (Pipe and Filter Architecture)
- Kiến trúc sự kiện (Event-Driven Architecture)
- **Kiến trúc không máy chủ (Serverless Architecture)**

Kiến trúc không máy chủ (Serverless Architecture)

- Mô tả: Ứng dụng được triển khai trên nền tảng đám mây, nơi nhà cung cấp dịch vụ quản lý cơ sở hạ tầng và tự động mở rộng. VD: Netflix, cocacola Vending machine
- Ưu điểm:
  - Giảm chi phí vận hành và quản lý cơ sở hạ tầng.
  - Tăng khả năng mở rộng và linh hoạt.
  - Tập trung vào phát triển chức năng thay vì quản lý máy chủ.
- Nhược điểm:
  - Phụ thuộc vào nhà cung cấp dịch vụ đám mây.
  - Có thể gặp khó khăn trong việc tối ưu hóa hiệu suất và chi phí.
  - Giới hạn trong việc kiểm soát môi trường thực thi.

3.3 Chuẩn bị tài liệu

- Việc chuẩn bị tài liệu phần mềm là một phần không thể thiếu trong quy trình phát triển phần mềm, giúp đảm bảo rằng dự án được thực hiện hiệu quả, đáp ứng các yêu cầu và tiêu chuẩn chất lượng, và có khả năng bảo trì và phát triển trong tương lai.
- Các bước chuẩn bị tài liệu:
  - Mô tả tổng quan
  - Sơ đồ kiến trúc

3.3 Chuẩn bị tài liệu

- Các bước chuẩn bị tài liệu (tt):
  - Mô tả chi tiết các thành phần
  - Yêu cầu phi chức năng
  - Kế hoạch triển khai

3.3.1. Mô tả tổng quan phần mềm

- **Xác định Mục tiêu của Hệ thống**:
  - Mô tả rõ ràng mục tiêu chính mà phần mềm cần đạt được.
  - Giải thích lý do tại sao phần mềm này cần được phát triển.
- **Định nghĩa Phạm vi Dự án**:
  - Xác định các chức năng chính mà phần mềm sẽ cung cấp.
  - Đưa ra các giới hạn của dự án để tránh việc mở rộng không kiểm soát.

3.3.1 Mô tả tổng quan phần mềm

- **Xác định Đối tượng Sử dụng**:
  - Xác định ai sẽ là người sử dụng phần mềm.
  - Mô tả nhu cầu và mong đợi của người dùng đối với phần mềm.
- **Mô tả Bối cảnh Hoạt động**:
  - Giải thích cách phần mềm sẽ được sử dụng trong môi trường thực tế.
  - Đưa ra các kịch bản sử dụng chính để minh họa cách phần mềm hoạt động.

3.3.1 Mô tả tổng quan phần mềm

- **Xác định Các Giả định và Ràng buộc**:
  - Liệt kê các giả định đã được đưa ra trong quá trình phát triển phần mềm.
  - Đưa ra các ràng buộc về công nghệ, ngân sách, thời gian, và nguồn lực.
- **Đánh giá Rủi ro Ban đầu**:
  - Xác định các rủi ro tiềm ẩn có thể ảnh hưởng đến dự án.
  - Đưa ra các biện pháp giảm thiểu rủi ro nếu có thể.

Phân biệt tài liệu tổng quan phần mềm với SRS

- Mức độ chi tiết: Tài liệu tổng quan phần mềm cung cấp cái nhìn tổng thể và khái quát, trong khi tài liệu đặc tả phần mềm đi vào chi tiết cụ thể về các yêu cầu và chức năng.
- Mục tiêu: Tài liệu tổng quan tập trung vào việc định hướng và giao tiếp với các bên liên quan, còn tài liệu đặc tả tập trung vào việc hướng dẫn phát triển và kiểm thử phần mềm.
- Đối tượng sử dụng: Tài liệu tổng quan thường được sử dụng bởi các bên liên quan và quản lý dự án, trong khi tài liệu đặc tả chủ yếu được sử dụng bởi nhóm phát triển và kiểm thử.

3.3.2 Sơ đồ kiến trúc

- **Các bước cơ bản tạo sơ đồ kiến trúc**
- Các loại sơ đồ kiến trúc phổ biến

Các bước cơ bản tạo sơ đồ kiến trúc

- Để tạo ra một sơ đồ kiến trúc phần mềm phản ánh chính xác cấu trúc và các mối quan hệ trong hệ thống. Cần thực hiện **các bước** cơ bản:
  - **Thu thập Yêu cầu:** Thu thập và phân tích các yêu cầu chức năng/phi chức năng. Xác định các ràng buộc kỹ thuật và kinh doanh có thể ảnh hưởng đến kiến trúc.
  - **Xác định Các Thành phần Chính:** Xác định các thành phần chính của hệ thống, bao gồm các mô-đun, dịch vụ, và giao diện. Định nghĩa vai trò và trách nhiệm của từng thành phần.

Các bước cơ bản tạo sơ đồ kiến trúc

- **Lựa chọn Kiến trúc Phù hợp:** Lựa chọn kiểu kiến trúc phù hợp với yêu cầu và ràng buộc của dự án, chẳng hạn như MVC, Microservices, hoặc Layered Architecture. Đánh giá các giải pháp kiến trúc khác nhau và chọn giải pháp tối ưu.
- **Thiết kế Sơ đồ Kiến trúc:**Sử dụng các công cụ và kỹ thuật để thiết kế sơ đồ kiến trúc, chẳng hạn như UML (Unified Modeling Language) hoặc các công cụ vẽ sơ đồ như Lucidchart, Visio, hoặc Draw.io. Vẽ các sơ đồ cần thiết, bao gồm sơ đồ lớp, sơ đồ trình tự, sơ đồ thành phần, và sơ đồ triển khai.

Các bước cơ bản tạo sơ đồ kiến trúc

- **Xác định Mối quan hệ và Giao tiếp:** Xác định cách các thành phần giao tiếp với nhau, bao gồm các giao thức và định dạng dữ liệu. Mô tả các mối quan hệ và phụ thuộc giữa các thành phần.
- **Xem xét và Phê duyệt:** Trình bày sơ đồ kiến trúc cho các bên liên quan để thu thập phản hồi và đề xuất cải tiến. Thực hiện các điều chỉnh cần thiết dựa trên phản hồi và phê duyệt sơ đồ cuối cùng

Các bước cơ bản tạo sơ đồ kiến trúc

- **Tài liệu hóa Kiến trúc:** Tài liệu hóa chi tiết sơ đồ kiến trúc, bao gồm các quyết định thiết kế, lý do lựa chọn kiến trúc, và các ràng buộc. Đảm bảo rằng tài liệu kiến trúc dễ hiểu và có thể được sử dụng làm hướng dẫn cho nhóm phát triển.
- **Duy trì và Cập nhật:** Duy trì và cập nhật sơ đồ kiến trúc khi có thay đổi trong yêu cầu hoặc thiết kế hệ thống

3.3.2 Sơ đồ kiến trúc

- Các bước cơ bản tạo sơ đồ kiến trúc
- **Các loại sơ đồ kiến trúc phổ biến**

Các loại sơ đồ kiến trúc phổ biến

- **Sơ đồ Kiến trúc Tổng thể:** Cung cấp cái nhìn tổng quan về toàn bộ hệ thống, bao gồm các thành phần chính và mối quan hệ giữa chúng
- **Sơ đồ Thành phần (Component Diagram):** Mô tả các thành phần phần mềm và mối quan hệ giữa chúng.
- **Sơ đồ Lớp (Class Diagram):** Mô tả các lớp trong hệ thống, thuộc tính, phương thức, và mối quan hệ giữa các lớp.

Các loại sơ đồ kiến trúc phổ biến

- **Sơ đồ Trình tự (Sequence Diagram):** Mô tả cách các đối tượng tương tác với nhau theo thời gian, thể hiện thứ tự các thông điệp được gửi.
- **Sơ đồ Hoạt động (Activity Diagram):** Mô tả các hoạt động và luồng công việc trong hệ thống.
- **Sơ đồ Triển khai (Deployment Diagram):** Mô tả cách các thành phần phần mềm được triển khai trên phần cứng, bao gồm máy chủ, thiết bị, và mạng.

Các loại sơ đồ kiến trúc phổ biến

- **Sơ đồ Giao diện (Interface Diagram):** Mô tả các giao diện giữa các thành phần trong hệ thống.
- **Sơ đồ Dòng dữ liệu (Data Flow Diagram):** Mô tả cách dữ liệu di chuyển qua hệ thống, từ nguồn đến đích.

3.4. Các mô hình quy trình phát triển phần mềm

- **Mô hình thác nước**
- Mô hình nguyên mẫu
- Mô hình gia tăng
- Mô hình xoắn ốc
- Mô hình Agile Scrum

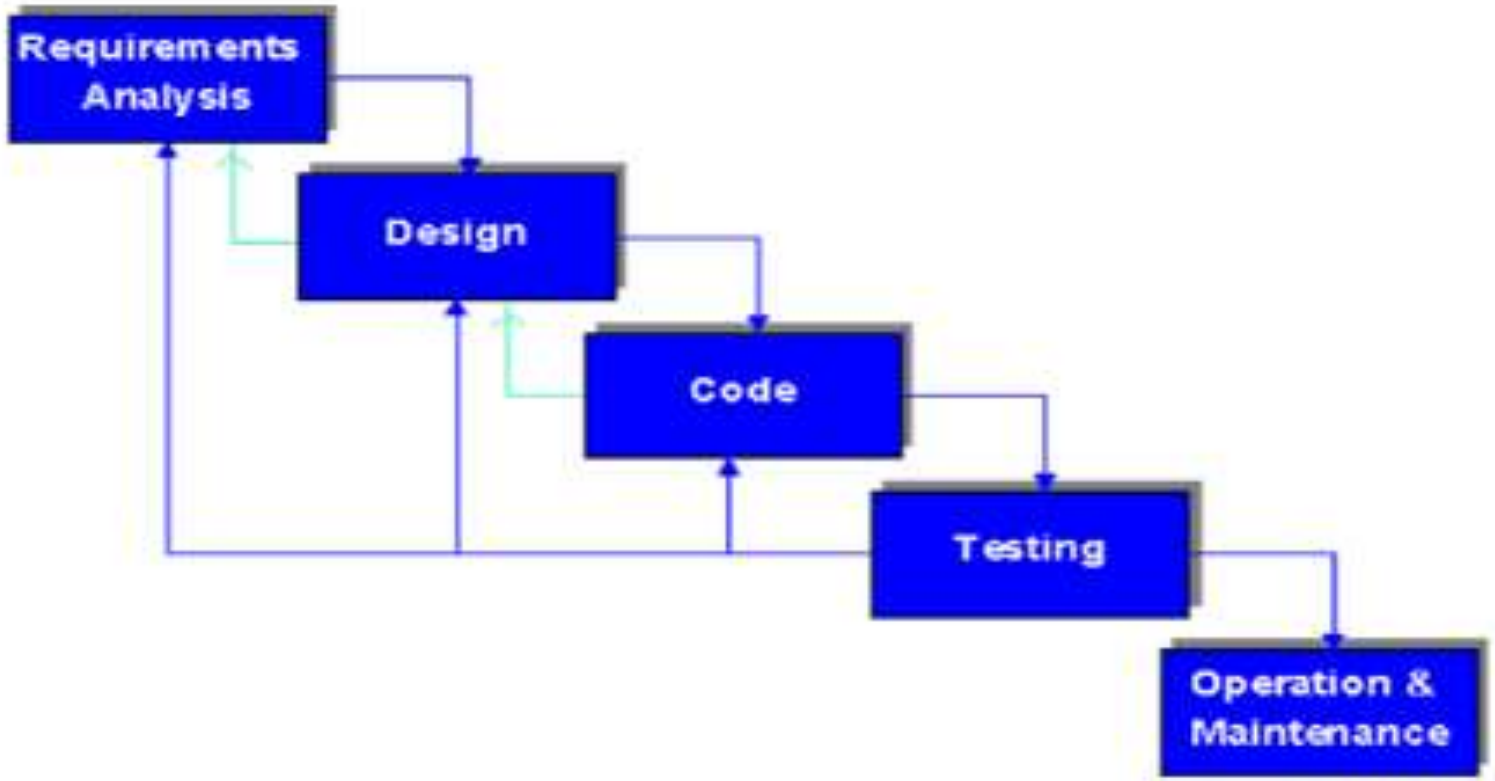
Mô hình thác nước

- Mô hình thác nước là mô hình vòng đời lâu đời nhất; được đề xuất bởi Winston Royce vào năm 1970.
- Mô hình này được gọi là thác nước vì nó thường được vẽ với một chuỗi các hoạt động qua các giai đoạn của vòng đời "xuống dốc" từ trái sang phải:
  - phân tích, yêu cầu, đặc tả, thiết kế, cài đặt, kiểm thử, bảo trì
- Có nhiều phiên bản của mô hình thác nước:
  - các giai đoạn / hoạt động có thể được cấu trúc theo các mức độ chi tiết khác nhau
  - phản hồi có thể linh hoạt hơn hoặc ít hơn

Mô hình thác nước

- Đặc điểm của Mô hình Thác nước
  - Tuân thủ trình tự tuyến tính: Mô hình thác nước thực hiện các giai đoạn phát triển theo một trình tự tuyến tính, từ giai đoạn này sang giai đoạn khác mà không quay lại.
  - Rõ ràng và dễ hiểu: Mô hình này có cấu trúc rõ ràng, dễ hiểu và dễ quản lý, giúp các bên liên quan dễ dàng theo dõi tiến độ.
  - Tài liệu hóa đầy đủ: Mỗi giai đoạn đều được tài liệu hóa chi tiết, giúp đảm bảo rằng tất cả các yêu cầu và quyết định đều được ghi lại.

Mô hình thác nước



Mô hình thác nước

- Các Giai đoạn của Mô hình Thác nước:
  - **Phân tích yêu cầu (Requirements Analysis):** Thu thập và phân tích các yêu cầu từ khách hàng và các bên liên quan. Tạo ra tài liệu yêu cầu phần mềm
  - **Thiết kế hệ thống (System Design)**:Thiết kế kiến trúc tổng thể của hệ thống dựa trên các yêu cầu đã phân tích. Tạo ra tài liệu thiết kế chi tiết.

Mô hình thác nước

- **Lập trình (Code)**: Phát triển mã nguồn dựa trên tài liệu thiết kế. Thực hiện các kiểm thử đơn vị để đảm bảo rằng các thành phần hoạt động đúng.
- **Kiểm thử (Testing)**: Thực hiện kiểm thử hệ thống để phát hiện và sửa lỗi. Đảm bảo rằng phần mềm đáp ứng các yêu cầu đã đề ra.
- **Triển khai (Deployment)**: Triển khai phần mềm vào môi trường sản xuất. Đào tạo người dùng và cung cấp tài liệu hướng dẫn.
- **Bảo trì (Maintenance)**: Thực hiện bảo trì và cập nhật phần mềm khi cần thiết. Giải quyết các vấn đề phát sinh sau khi triển khai.

Mô hình thác nước

- Ưu điểm:
  - Dễ hiểu và dễ quản lý: Cấu trúc tuyến tính giúp dễ dàng theo dõi tiến độ và quản lý dự án.
  - Tài liệu hóa đầy đủ: Mỗi giai đoạn đều có tài liệu rõ ràng, giúp dễ dàng tham khảo và bảo trì.
  - Phù hợp với các dự án nhỏ: Mô hình này hiệu quả cho các dự án có yêu cầu rõ ràng và không thay đổi nhiều.

Mô hình thác nước

- Nhược điểm:
  - Thiếu linh hoạt: Khó khăn trong việc thay đổi yêu cầu sau khi đã bắt đầu phát triển, vì mỗi giai đoạn phải hoàn thành trước khi chuyển sang giai đoạn tiếp theo.
  - Rủi ro cao: Nếu phát hiện lỗi hoặc vấn đề trong giai đoạn kiểm thử, có thể phải quay lại nhiều giai đoạn trước đó, gây tốn thời gian và chi phí.
  - Không phù hợp với dự án lớn và phức tạp: Mô hình này không hiệu quả cho các dự án lớn có yêu cầu thay đổi thường xuyên.

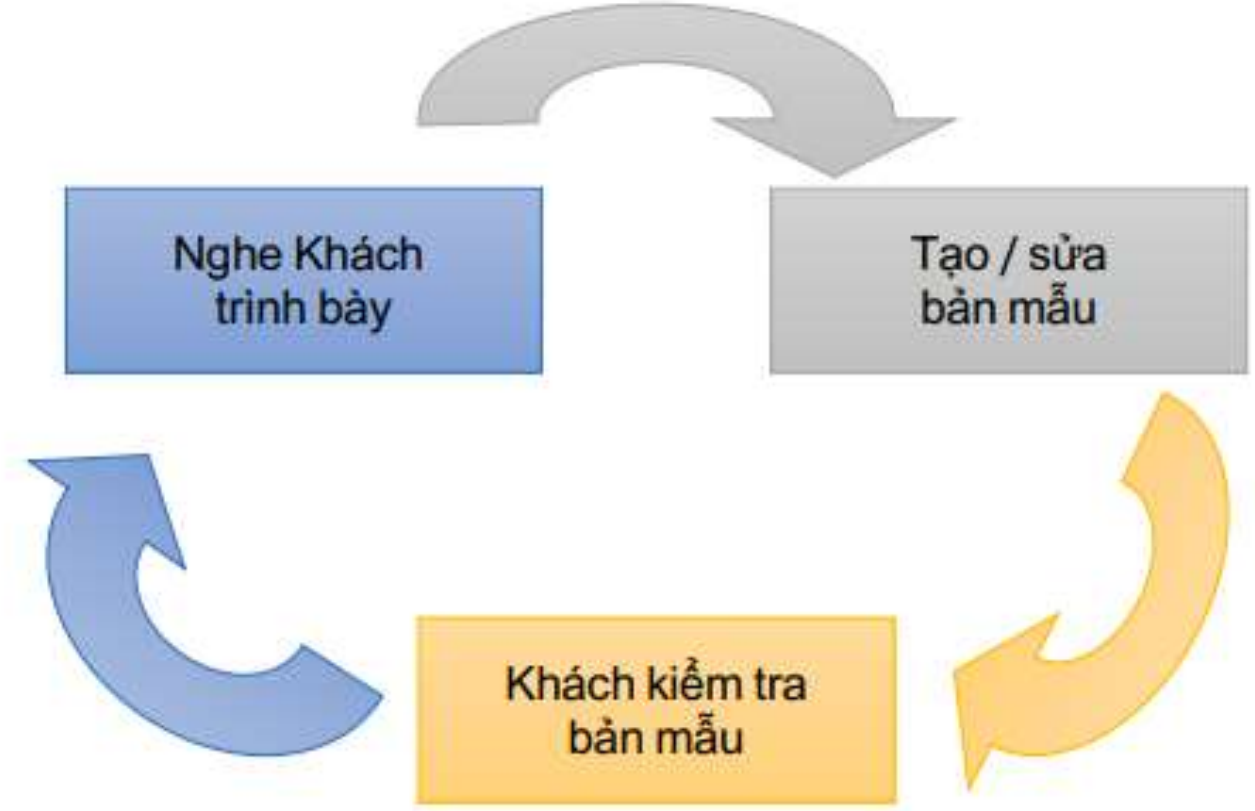
Mô hình thác nước

- Mô hình thác nước phù hợp trong các tình huống sau::
  - Dự án nhỏ và đơn giản: Khi yêu cầu rõ ràng và không thay đổi nhiều.
  - Dự án có yêu cầu pháp lý hoặc quy định nghiêm ngặt: Khi cần tài liệu hóa đầy đủ và tuân thủ quy trình.
  - Dự án với thời gian và ngân sách cố định: Khi cần một kế hoạch phát triển rõ ràng và có thể dự đoán được..

3.4. Các mô hình quy trình phát triển phần mềm

- Mô hình thác nước
- **Mô hình nguyên mẫu**
- Mô hình gia tăng
- Mô hình xoắn ốc
- Mô hình Agile Scrum

Mô hình nguyên mẫu



Mô hình nguyên mẫu

- Đặc điểm của Mô hình nguyên mẫu
  - Tạo mẫu nhanh: Mô hình Prototype cho phép tạo ra các mẫu nhanh chóng để kiểm tra ý tưởng và chức năng.
  - Phản hồi từ người dùng: Người dùng có thể tương tác với nguyên mẫu và cung cấp phản hồi, giúp điều chỉnh yêu cầu và thiết kế.
  - Tính linh hoạt: Dễ dàng thay đổi và điều chỉnh dựa trên phản hồi từ người dùng.
  - Tập trung vào giao diện người dùng: Thường được sử dụng để phát triển giao diện người dùng và trải nghiệm người dùng.

Mô hình nguyên mẫu

- Các Giai đoạn của Mô hình nguyên mẫu
  - Xác định yêu cầu: Thu thập và phân tích yêu cầu từ người dùng và các bên liên quan. Xác định các chức năng chính mà nguyên mẫu cần có.
  - Tạo nguyên mẫu: Phát triển một phiên bản mẫu của sản phẩm với các chức năng cơ bản. Nguyên mẫu có thể là một mô hình tĩnh (wireframe) hoặc một ứng dụng hoạt động.
  - Thử nghiệm và thu thập phản hồi: Người dùng tương tác với nguyên mẫu và cung cấp phản hồi về chức năng, giao diện, và trải nghiệm. Ghi nhận các vấn đề và yêu cầu mới từ người dùng.

Mô hình nguyên mẫu

- Ưu điểm của Mô hình Prototype
  - Phản hồi sớm: Giúp thu thập phản hồi từ người dùng sớm trong quá trình phát triển, giảm thiểu rủi ro.
  - Giảm hiểu lầm: Giúp làm rõ yêu cầu và mong đợi của người dùng, giảm thiểu sự hiểu lầm.
  - Tăng sự hài lòng của người dùng: Người dùng có thể thấy sản phẩm trong quá trình phát triển và cảm thấy hài lòng hơn với kết quả cuối cùng.
  - Khả năng điều chỉnh cao: Dễ dàng thay đổi và điều chỉnh dựa trên phản hồi từ người dùng.

Mô hình nguyên mẫu

- Nhược điểm của Mô hình Prototype
  - Chi phí cao: Việc tạo ra nhiều nguyên mẫu có thể tốn kém về thời gian và nguồn lực.
  - Có thể dẫn đến yêu cầu không rõ ràng: Người dùng có thể yêu cầu thêm nhiều tính năng không cần thiết chỉ vì họ thấy nguyên mẫu.
  - Không phải là sản phẩm hoàn chỉnh: Nguyên mẫu có thể không phản ánh đầy đủ các vấn đề kỹ thuật và hiệu suất của sản phẩm cuối cùng.
  - Khó khăn trong việc quản lý: Việc lặp lại quá trình thử nghiệm và cải tiến có thể dẫn đến khó khăn trong việc quản lý dự án.

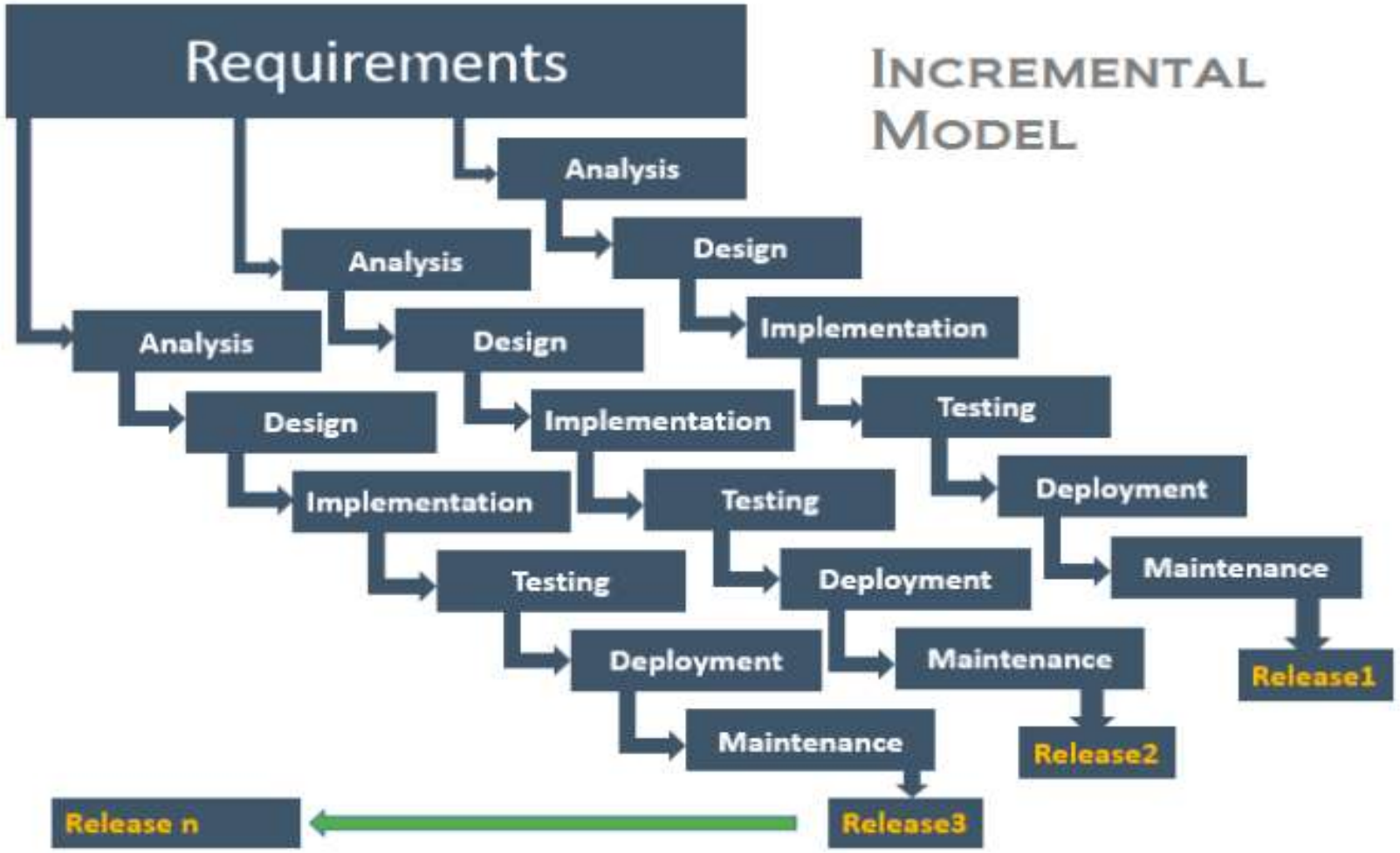
Mô hình nguyên mẫu

- Tình huống áp dụng phù hợp
  - Dự án có yêu cầu không rõ ràng: Khi yêu cầu của người dùng chưa rõ ràng hoặc có thể thay đổi, mô hình Prototype giúp làm rõ và điều chỉnh yêu cầu.
  - Phát triển giao diện người dùng: Khi cần phát triển giao diện người dùng và trải nghiệm người dùng, mô hình Prototype cho phép thử nghiệm và thu thập phản hồi nhanh chóng.
  - Dự án có tính chất sáng tạo cao: Trong các dự án yêu cầu sự sáng tạo và đổi mới, việc tạo ra nguyên mẫu giúp khám phá các ý tưởng mới.
  - Dự án có thời gian phát triển ngắn: Khi cần phát triển nhanh chóng và thu thập phản hồi từ người dùng, mô hình Prototype là một lựa chọn tốt.

3.4. Các mô hình quy trình phát triển phần mềm

- Mô hình thác nước
- Mô hình nguyên mẫu
- **Mô hình gia tăng**
- Mô hình xoắn ốc
- Mô hình Agile Scrum

Mô hình gia tăng



Mô hình gia tăng

- Đặc điểm
  - Phát triển theo từng phần: Sản phẩm được chia thành các phần nhỏ (increments), mỗi phần được phát triển và hoàn thiện một cách độc lập.
  - Tính linh hoạt: Có khả năng điều chỉnh và thay đổi yêu cầu trong quá trình phát triển, cho phép phản hồi từ người dùng.
  - Giao tiếp liên tục: Thường xuyên tương tác với khách hàng để thu thập phản hồi và điều chỉnh sản phẩm.

Mô hình gia tăng

- Các giai đoạn
  - Mô hình gia tăng thường bao gồm các giai đoạn sau:
  - Lập kế hoạch: Xác định yêu cầu và lập kế hoạch cho từng phần của sản phẩm.
  - Phân tích yêu cầu: Phân tích và xác định yêu cầu cho phần đầu tiên của sản phẩm.
  - Thiết kế: Thiết kế kiến trúc và các thành phần cho phần đầu tiên.
  - Phát triển: Lập trình và phát triển phần đầu tiên của sản phẩm.

Mô hình gia tăng

- Các giai đoạn (tt)
  - Kiểm thử: Kiểm thử phần đầu tiên để đảm bảo chất lượng và đáp ứng yêu cầu.
  - Triển khai: Triển khai phần đầu tiên cho người dùng.
  - Phản hồi: Thu thập phản hồi từ người dùng và điều chỉnh yêu cầu cho các phần tiếp theo.
  - Lặp lại: Lặp lại các bước từ phân tích yêu cầu đến triển khai cho các phần tiếp theo cho đến khi hoàn thành sản phẩm.

Mô hình gia tăng

- Ưu điểm
  - Tính linh hoạt: Dễ dàng điều chỉnh yêu cầu và thay đổi trong quá trình phát triển.
  - Giảm rủi ro: Các phần nhỏ được phát triển và kiểm thử độc lập, giúp phát hiện lỗi sớm.
  - Phản hồi nhanh chóng: Khách hàng có thể sử dụng các phần đã hoàn thành và cung cấp phản hồi ngay lập tức.
  - Tăng cường sự tham gia của khách hàng: Khách hàng có thể thấy sản phẩm dần hoàn thiện và tham gia vào quá trình phát triển.

Mô hình gia tăng

- Nhược điểm
  - Quản lý phức tạp: Cần phải quản lý nhiều phần và yêu cầu khác nhau, có thể dẫn đến sự phức tạp trong quản lý dự án.
  - Khó khăn trong việc xác định yêu cầu: Nếu yêu cầu không được xác định rõ ràng từ đầu, có thể dẫn đến sự thay đổi lớn trong các phần sau.
  - Chi phí cao: Có thể phát sinh chi phí cao hơn do việc phát triển và kiểm thử nhiều phần khác nhau.

Mô hình gia tăng

- Mô hình gia tăng phù hợp trong các tình huống sau:
  - Dự án có yêu cầu không rõ ràng: Khi yêu cầu có thể thay đổi hoặc chưa được xác định rõ ràng, mô hình gia tăng cho phép điều chỉnh linh hoạt.
  - Dự án cần phản hồi nhanh: Khi cần thu thập phản hồi từ người dùng để cải thiện sản phẩm, mô hình gia tăng cho phép phát triển và triển khai từng phần.
  - Dự án lớn và phức tạp: Khi dự án quá lớn để phát triển một lần, mô hình gia tăng giúp chia nhỏ công việc và quản lý dễ dàng hơn.
  - Dự án có thời gian hạn chế: Khi cần có một sản phẩm khả dụng nhanh chóng, mô hình gia tăng cho phép phát triển các phần quan trọng trước.

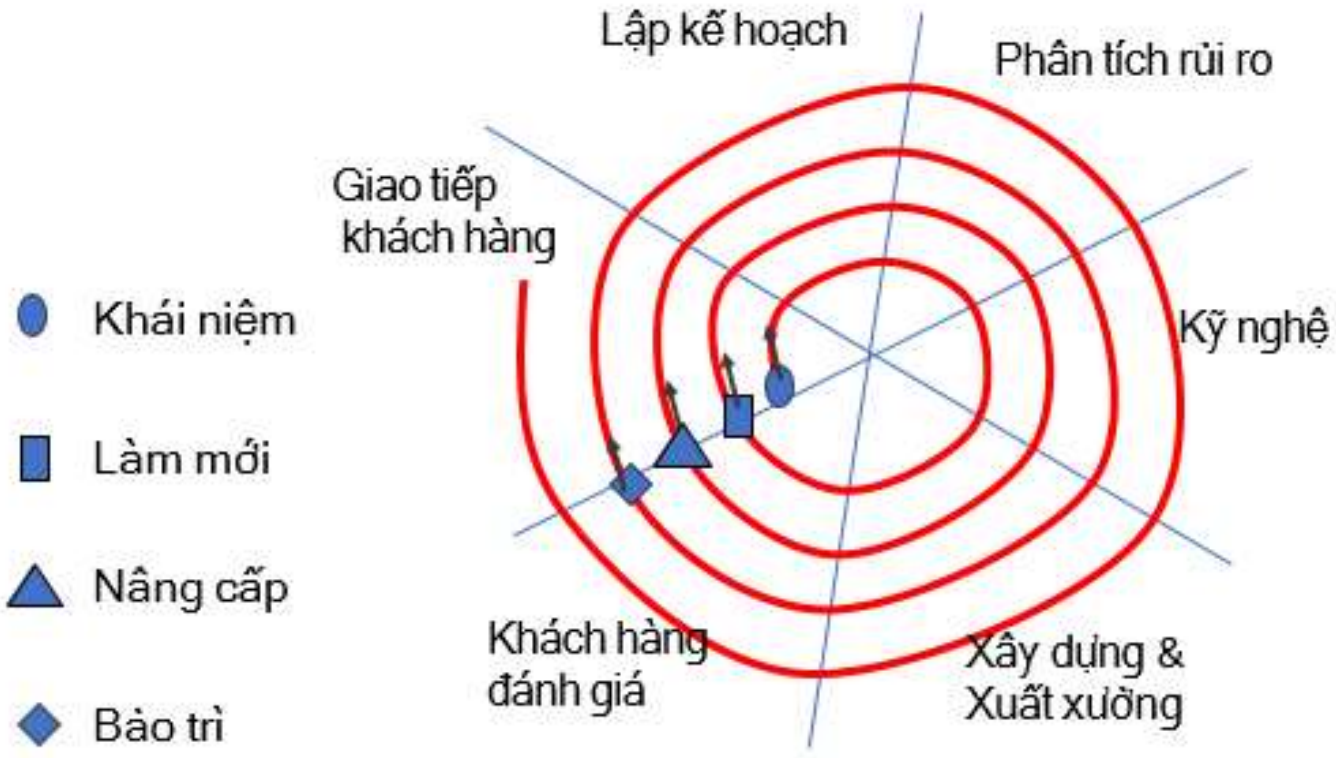
3.4. Các mô hình quy trình phát triển phần mềm

- Mô hình thác nước
- Mô hình nguyên mẫu
- Mô hình gia tăng
- **Mô hình xoắn ốc**
- Mô hình Agile Scrum

Mô hình xoắn ốc

- Mô hình xoắn ốc (Spiral Model) là một phương pháp phát triển phần mềm kết hợp giữa các yếu tố của mô hình thác nước (Waterfall Model) và mô hình lặp đi lặp lại (Iterative Model). Mô hình này được thiết kế để quản lý rủi ro và cải tiến sản phẩm qua từng vòng lặp.
- Đặc điểm
  - Tập trung vào Rủi ro: Mô hình xoắn ốc nhấn mạnh việc xác định và quản lý rủi ro trong từng giai đoạn phát triển.
  - Lặp đi lặp lại: Mỗi vòng lặp (spiral) trong mô hình đại diện cho một giai đoạn phát triển, cho phép cải tiến và điều chỉnh sản phẩm dựa trên phản hồi.
  - Tích hợp Phân tích và Thiết kế: Mô hình này cho phép phân tích và thiết kế được thực hiện đồng thời, giúp phát hiện sớm các vấn đề.

Mô hình xoắn ốc



Mô hình xoắn ốc

- Mô hình xoắn ốc thường bao gồm các giai đoạn chính trong mỗi vòng lặp:
  - Giao tiếp khách hàng: giữa người phát triển và khách hàng để tìm hiểu yêu cầu, ý kiến
  - Lập kế hoạch: Xác lập tài nguyên, thời hạn và những thông tin khác
  - Phân tích rủi ro: Xem xét mạo hiểm kỹ thuật và mạo hiểm quản lý
  - Kỹ nghệ: Xây dựng một hay một số biểu diễn của ứng dụng

Mô hình xoắn ốc

- Mô hình xoắn ốc thường bao gồm các giai đoạn chính trong mỗi vòng lặp (tt):
  - Xây dựng và xuất xưởng: xây dựng, kiểm thử, cài đặt và cung cấp hỗ trợ người dùng (tư liệu, huấn luyện, . . .)
  - Đánh giá của khách hàng: Nhận các phản hồi của người sử dụng về biểu diễn phần mềm trong giai đoạn kỹ nghệ và cài đặt

Mô hình xoắn ốc

- Ưu điểm
  - Quản lý Rủi ro Tốt: Mô hình xoắn ốc cho phép phát hiện và giảm thiểu rủi ro sớm trong quá trình phát triển.
  - Linh hoạt và Thích ứng: Có khả năng điều chỉnh và cải tiến sản phẩm qua từng vòng lặp dựa trên phản hồi từ người dùng.
  - Phát triển Từng Phần: Cho phép phát triển và kiểm thử từng phần của sản phẩm, giúp cải thiện chất lượng

Mô hình xoắn ốc

- Nhược điểm
  - Phức tạp: Mô hình có thể trở nên phức tạp và khó quản lý, đặc biệt là trong các dự án lớn.
  - Chi phí Cao: Việc quản lý rủi ro và thực hiện các vòng lặp có thể dẫn đến chi phí cao hơn so với các mô hình khác.
  - Cần Kinh nghiệm: Đòi hỏi đội ngũ phát triển có kinh nghiệm trong việc quản lý rủi ro và thực hiện phân tích.

Mô hình xoắn ốc

- Tình huống áp dụng phù hợp
  - Dự án lớn và phức tạp: Mô hình xoắn ốc phù hợp cho các dự án có quy mô lớn, yêu cầu nhiều tính năng và có độ phức tạp cao.
  - Dự án có yêu cầu thay đổi thường xuyên: Khi yêu cầu của người dùng có thể thay đổi trong quá trình phát triển, mô hình xoắn ốc cho phép điều chỉnh linh hoạt.
  - Dự án có rủi ro cao: Khi có nhiều rủi ro liên quan đến công nghệ, yêu cầu, hoặc thị trường, mô hình xoắn ốc giúp quản lý và giảm thiểu rủi ro hiệu quả.

3.4. Các mô hình quy trình phát triển phần mềm

- Mô hình thác nước
- Mô hình nguyên mẫu
- Mô hình gia tăng
- Mô hình xoắn ốc
- **Mô hình Agile Scrum**

Mô hình Agile Scrum

- Agile Scrum là một phương pháp phát triển phần mềm thuộc mô hình Agile, tập trung vào việc quản lý và phát triển sản phẩm thông qua các vòng lặp ngắn (sprint) và cải tiến liên tục. Scrum giúp các nhóm phát triển làm việc hiệu quả hơn bằng cách chia nhỏ công việc thành các phần có thể quản lý và dễ dàng điều chỉnh theo phản hồi từ khách hàng.

Mô hình Agile Scrum

- Agile Scrum là một phương pháp phát triển phần mềm thuộc mô hình Agile, tập trung vào việc quản lý và phát triển sản phẩm thông qua các vòng lặp ngắn (sprint) và cải tiến liên tục. Scrum giúp các nhóm phát triển làm việc hiệu quả hơn bằng cách chia nhỏ công việc thành các phần có thể quản lý và dễ dàng điều chỉnh theo phản hồi từ khách hàng.
- Đặc điểm
  - Tính lặp đi lặp lại: Scrum sử dụng các vòng lặp ngắn để phát triển và cải tiến sản phẩm liên tục.

Mô hình Agile Scrum

- Đặc điểm
  - Tính linh hoạt: Nhóm có thể điều chỉnh kế hoạch và ưu tiên dựa trên phản hồi từ khách hàng và các bên liên quan.
  - Tập trung vào khách hàng: Scrum nhấn mạnh việc nhận phản hồi từ khách hàng để cải thiện sản phẩm.
  - Tự tổ chức: Nhóm Scrum tự quản lý và quyết định cách thức thực hiện công việc.
  - Vai trò rõ ràng: Scrum xác định các vai trò cụ thể như Product Owner, Scrum Master và Development Team.

Các giai đoạn chính mô hình Agile Scrum

- Sprint Planning (Lập kế hoạch Sprint):
  - Nhóm xác định các mục tiêu cho sprint tiếp theo và chọn các mục từ danh sách sản phẩm (Product Backlog) để thực hiện trong sprint.
  - Daily Scrum (Họp hàng ngày):
  - Cuộc họp ngắn (thường 15 phút) diễn ra hàng ngày, nơi các thành viên trong nhóm báo cáo về tiến độ, kế hoạch cho ngày tiếp theo và các trở ngại gặp phải.

Các giai đoạn chính mô hình Agile Scrum

- Sprint Review (Đánh giá Sprint): Cuộc họp diễn ra vào cuối mỗi sprint, nơi nhóm trình bày các tính năng đã hoàn thành cho các bên liên quan và nhận phản hồi.
- Sprint Retrospective (Họp tổng kết Sprint): Cuộc họp diễn ra sau Sprint Review, nơi nhóm thảo luận về những gì đã làm tốt, những gì cần cải thiện và lập kế hoạch cho các hành động trong sprint tiếp theo.
- Sprint (Vòng lặp phát triển): Thời gian phát triển (thường từ 1 đến 4 tuần) trong đó nhóm làm việc để hoàn thành các mục tiêu đã xác định.

Mô hình Agile Scrum

- Ưu điểm
  - Tăng cường sự hợp tác: Scrum khuyến khích sự giao tiếp và hợp tác giữa các thành viên trong nhóm và các bên liên quan.
  - Phản hồi nhanh chóng: Nhóm có thể nhận phản hồi từ khách hàng sớm và điều chỉnh sản phẩm theo yêu cầu.
  - Cải tiến liên tục: Scrum thúc đẩy việc cải tiến quy trình và sản phẩm qua các cuộc họp tổng kết.
  - Giảm rủi ro: Việc phát triển theo từng sprint giúp phát hiện và giải quyết vấn đề sớm hơn.

Mô hình Agile Scrum

- Ưu điểm
  - Tăng cường sự hợp tác: Scrum khuyến khích sự giao tiếp và hợp tác giữa các thành viên trong nhóm và các bên liên quan.
  - Phản hồi nhanh chóng: Nhóm có thể nhận phản hồi từ khách hàng sớm và điều chỉnh sản phẩm theo yêu cầu.
  - Cải tiến liên tục: Scrum thúc đẩy việc cải tiến quy trình và sản phẩm qua các cuộc họp tổng kết.
  - Giảm rủi ro: Việc phát triển theo từng sprint giúp phát hiện và giải quyết vấn đề sớm hơn.

Mô hình Agile Scrum

- Nhược điểm
  - Cần sự cam kết: Scrum yêu cầu sự cam kết cao từ tất cả các thành viên trong nhóm, điều này có thể khó khăn trong một số tổ chức.
  - Khó khăn trong việc áp dụng: Một số tổ chức có thể gặp khó khăn trong việc chuyển đổi từ các phương pháp truyền thống sang Scrum.
  - Quản lý khối lượng công việc: Nếu không quản lý tốt, khối lượng công việc có thể trở nên quá tải trong các sprint.
  - Phụ thuộc vào sự tham gia của khách hàng: Scrum yêu cầu sự tham gia liên tục của khách hàng để đảm bảo rằng sản phẩm đáp ứng đúng nhu cầu

Mô hình Agile Scrum

- Dưới đây là một số tình huống và điều kiện khi mô hình Agile Scrum nên được áp dụng:
  - **Dự án có yêu cầu thay đổi thường xuyên:** Khi yêu cầu của khách hàng có thể thay đổi trong suốt quá trình phát triển, Scrum cho phép điều chỉnh linh hoạt và nhanh chóng.
  - **Dự án có tính phức tạp cao:** Khi dự án có nhiều yếu tố không chắc chắn và phức tạp, Scrum giúp quản lý rủi ro và cải tiến sản phẩm qua từng sprint. Scrum cho phép nhóm phát triển kiểm tra và điều chỉnh sản phẩm thường xuyên, giảm thiểu rủi ro.

Mô hình Agile Scrum

- **Dự án cần sự hợp tác chặt chẽ giữa các bên liên quan:** Scrum khuyến khích giao tiếp và phản hồi liên tục.
- **Dự án yêu cầu phát hành nhanh và thường xuyên:** Scrum cho phép phát triển theo từng sprint ngắn.
- **Dự án có đội ngũ phát triển nhỏ và tự quản lý:** Đội ngũ có thể nhanh chóng đưa ra quyết định và điều chỉnh quy trình làm việc để tối ưu hóa hiệu suất.
- **Dự án cần cải tiến liên tục:** Scrum cung cấp các buổi họp như Sprint Retrospective để đánh giá và cải tiến quy trình.

Mô hình Agile Scrum

- **Dự án có tính chất đổi mới hoặc sáng tạo:** Scrum cho phép nhóm phát triển thử nghiệm và khám phá các giải pháp mới. Khuyến khích sự sáng tạo và đổi mới trong quy trình phát triển.

Agile Scrum là một phương pháp linh hoạt và hiệu quả cho những dự án có yêu cầu thay đổi thường xuyên, tính phức tạp cao, và cần sự hợp tác chặt chẽ giữa các bên liên quan. Tuy nhiên, việc áp dụng Scrum cũng cần xem xét đến văn hóa tổ chức và khả năng của đội ngũ phát triển.