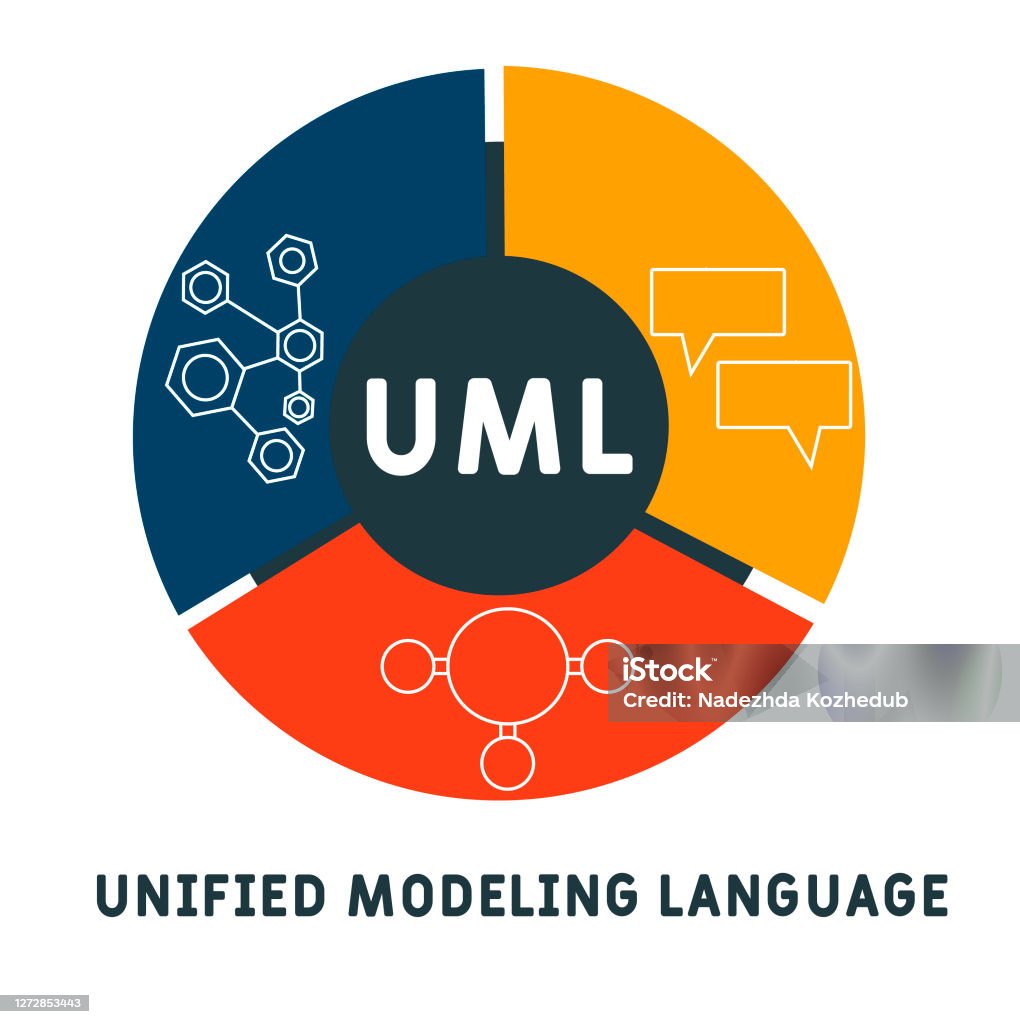
1. Sơ lược về ngôn ngữ mô hình hóa thống nhất UML

**- 1.1 Khái niệm về UML**

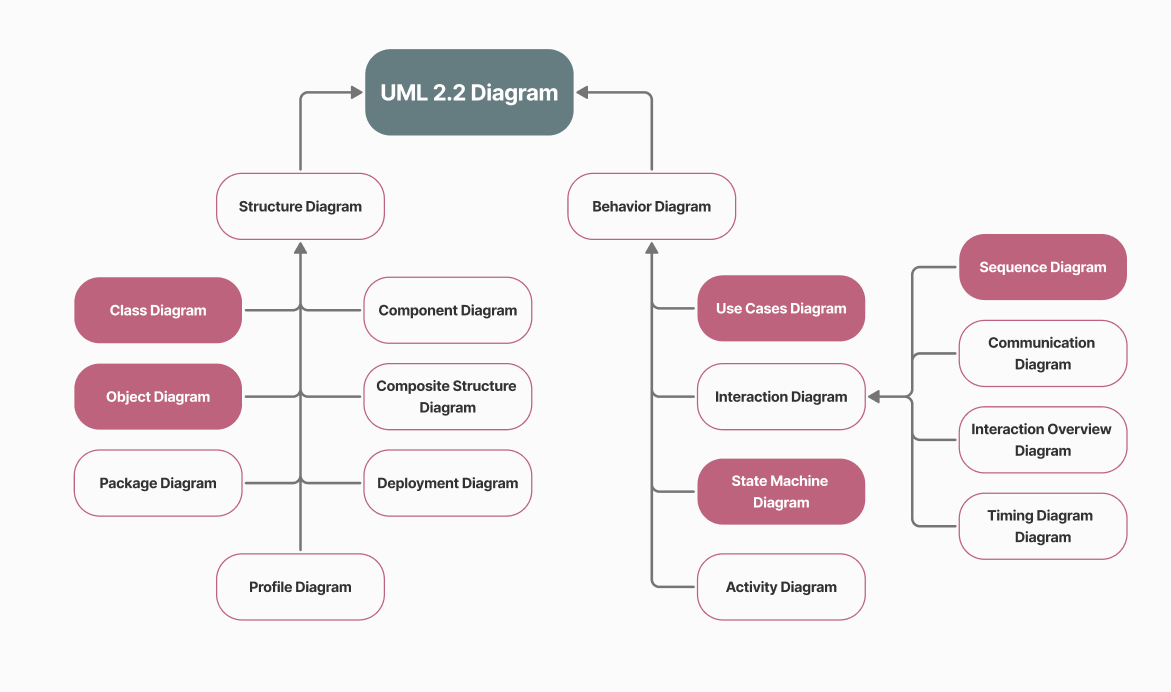
**1. Giới thiệu về UML**



Ngôn ngữ Mô hình hóa Thống nhất (UML – Unified Modeling Language **diu nì phai mó đồ ling láng guỵt t**) là một ngôn ngữ đồ họa mạnh mẽ, được thiết kế để hỗ trợ việc mô hình hóa các hệ thống thông tin phức tạp trong phát triển phần mềm. UML cung cấp một hệ thống các ký hiệu và phương pháp để mô tả cả cấu trúc và hành vi của hệ thống, từ đó giúp các nhà thiết kế, phát triển và kiểm thử hệ thống có cái nhìn toàn diện và chính xác.

* **Vai trò của UML:** UML không chỉ giúp lập mô hình hệ thống mà còn hỗ trợ trong việc phân tích, thiết kế, thẩm định và kiểm thử hệ thống phần mềm. Việc sử dụng UML giúp các nhóm phát triển và người sử dụng nắm bắt và hiểu rõ yêu cầu của hệ thống, đồng thời tạo điều kiện thuận lợi cho việc chuyển giao và duy trì hệ thống.

**2. Phân loại lược đồ trong UML**



UML cung cấp nhiều loại lược đồ khác nhau, mỗi loại lược đồ phục vụ một mục đích cụ thể trong việc mô hình hóa hệ thống. Các lược đồ này được phân thành hai nhóm chính: lược đồ cấu trúc và lược đồ hành vi.

**2.1 Lược đồ Cấu trúc:**

* **Lược đồ lớp (Class Diagram):** Mô tả các lớp, thuộc tính, phương thức và mối quan hệ giữa các lớp trong hệ thống.
* **Lược đồ đối tượng (Object Diagram):** Hiển thị các đối tượng và mối quan hệ của chúng tại một thời điểm cụ thể.
* **Lược đồ thành phần (Component còm pôu nền t Diagram):** Mô tả các thành phần phần mềm và cách chúng kết nối với nhau.
* **Lược đồ triển khai (Deployment đệp loy mần t Diagram):** Mô tả cách các thành phần phần mềm được triển khai trên phần cứng.

**2.2 Lược đồ Hành vi:**

* **Lược đồ tình huống sử dụng (Use case Diagram đai ờ ram):** Mô tả các chức năng của hệ thống từ góc nhìn của người dùng (tác nhân) và các Use case mà hệ thống hỗ trợ.
* **Lược đồ trình tự (Sequence si kuỳnh s Diagram):** Mô tả các tương tác giữa các đối tượng theo thứ tự thời gian.
* **Lược đồ hoạt động (Activity Diagram):** Mô tả quy trình và các hoạt động trong hệ thống.
* **Lược đồ trạng thái (State Machine Diagram sì tét mờ chin đai ờ ram):** Mô tả trạng thái của đối tượng và các sự kiện chuyển đổi giữa các trạng thái.

**3. Tạo lược đồ UML**

Việc xây dựng các mô hình trong UML giúp tạo ra cái nhìn tổng quát về hệ thống thông tin. Điều này không chỉ giúp nắm bắt yêu cầu của người dùng mà còn hỗ trợ trong việc thiết kế, kiểm thử và phát triển phần mềm. Các mô hình này cũng tạo cơ sở cho việc sử dụng các công cụ tự động sinh mã trong các ngôn ngữ lập trình hướng đối tượng như C++ và Java.

* **Mô hình đối tượng (Object Model):** Mô hình tĩnh của hệ thống, tập trung vào các lớp và mối quan hệ giữa chúng.
* **Mô hình động (Dynamic đai ná mịc Model):** Mô hình hoạt động của hệ thống, tập trung vào các tương tác và quy trình.

**4. Các tác giả chính của UML**



UML được phát triển bởi ba tác giả chính, được biết đến với tên gọi “Ba tác giả chính” (Three Amigos): Grady Booch(**rấy đi bút**), James Rumbaugh(**chems rum bồ**), và Ivar Jacobson(**ai va chây cóp sờn**).

* **Grady Booch:** tập trung vào mô hình hóa đối tượng.
* **James Rumbaugh:** Phát triển OMT (Object Modeling Technique **téch ni k**), tập trung vào kỹ thuật mô hình hóa đối tượng.
* **Ivar Jacobson:** Phát triển OOSE (Object-Oriented Software Engineering **ọp chéch o ren tịt sóp què en chì nia ring**), tập trung vào kỹ nghệ phần mềm hướng đối tượng.

Họ đã kết hợp các phương pháp mô hình hóa của riêng mình để tạo ra một ngôn ngữ thống nhất, giúp giải quyết các vấn đề phân tán trong thiết kế phần mềm

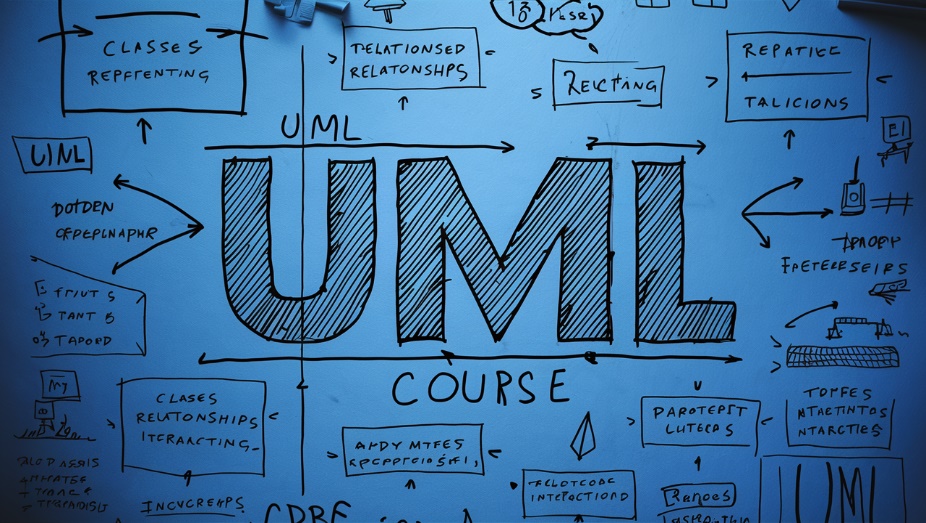
**5. Xu hướng và thành phần kết hợp trong UML**



UML kết hợp nhiều kỹ thuật mô hình hóa khác nhau từ các phương pháp luận của ba tác giả chính và các phương pháp mô hình hóa khác như:

* **Kỹ thuật mô hình hóa dữ liệu (Entity-Relationship Diagrams én ti đì rì lây sờn shíp đai à ram):** Mô tả các thực thể và mối quan hệ giữa chúng.
* **Mô hình hóa quy trình (Workflows quớt ph lầu s):** Mô tả quy trình và các bước trong hệ thống.
* **Mô hình hóa đối tượng:** Tập trung vào các lớp và đối tượng trong hệ thống.
* **Mô hình hóa cấu phần:** Mô tả các thành phần cấu trúc của hệ thống.

**6. Mục tiêu và ứng dụng của UML**



UML hướng đến việc tạo ra một chuẩn ngôn ngữ mô hình hóa chung cho tất cả các hệ thống phần mềm, bao gồm cả các hệ thống phân tán và tương tranh. UML không chỉ giúp mô hình hóa hệ thống mà còn hỗ trợ trong việc phát triển phần mềm qua các giai đoạn của vòng đời phát triển, từ phân tích yêu cầu đến thiết kế và kiểm thử.

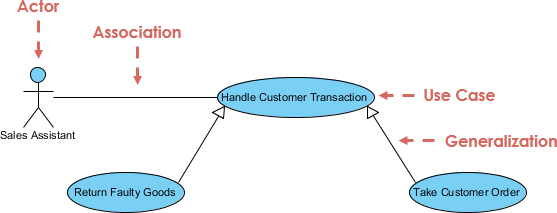
* **Ứng dụng thực tiễn:** UML có thể được sử dụng trong nhiều quy trình phát triển phần mềm và công nghệ thực thi hệ thống khác nhau, từ các ứng dụng đơn giản đến các hệ thống phức tạp và phân tán.

**Kết luận**

UML là một công cụ mạnh mẽ và linh hoạt trong thiết kế và phát triển phần mềm, cung cấp một ngôn ngữ chuẩn hóa để mô hình hóa các hệ thống phức tạp. Việc sử dụng UML giúp đảm bảo rằng hệ thống được thiết kế và phát triển theo đúng yêu cầu, đồng thời cung cấp cái nhìn rõ ràng và toàn diện về hệ thống.

**### 2. Lược đồ Use case (Use case s)**

* 1. **Giới thiệu về Use case (Use case s)**  
     Use case là một khái niệm quan trọng trong kỹ nghệ phần mềm, giúp mô tả cách mà hệ thống tương tác với các tác nhân bên ngoài để đáp ứng các mục tiêu cụ thể. Ý tưởng này bắt nguồn từ Ivar Jacobson, người đã phát triển cấu trúc Use case nhằm hỗ trợ mô tả các yêu cầu chức năng của hệ thống phần mềm một cách rõ ràng và trực quan. Việc sử dụng Use case không chỉ giúp cho nhóm phát triển phần mềm hiểu rõ yêu cầu từ khách hàng mà còn tạo ra cơ sở cho việc kiểm thử hệ thống sau này.

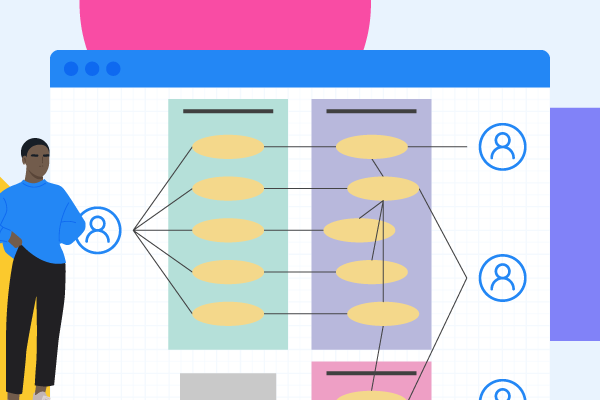


* **Ví dụ thực tế:** Trong hệ thống quản lý thư viện, Use case có thể là việc mượn sách. Mục tiêu của Use case này là tự động hóa quy trình mượn sách giữa độc giả và thư viện, tương tác với các tác nhân như thủ thư và độc giả.

**Chen nờ lai zấy sờn**

**ệch sô si ấy sần**

* 1. **Sự khác biệt giữa Use case và tương tác người dùng**

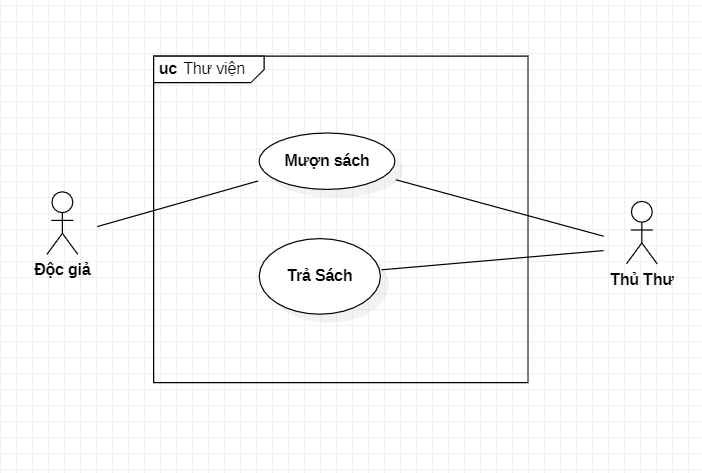
  
Use case tập trung vào mục tiêu mà người dùng muốn đạt được, trong khi tương tác người dùng mô tả chi tiết các bước phải trải qua để đạt được mục tiêu đó. Điều này tạo ra sự phân biệt rõ ràng ~~giữa cái What (cái gì) và cái How (làm như thế nào).~~

* **Phân tích cụ thể:** Khi xây dựng một ca sử dụng, điều quan trọng là tập trung vào mục tiêu cuối cùng mà người dùng muốn đạt được. Ví dụ, trong hệ thống quản lý thư viện, mục tiêu có thể là "mượn sách", nhưng các bước cụ thể như quét mã vạch, xác nhận thông tin, hay in biên lai chính là phần tương tác người dùng.

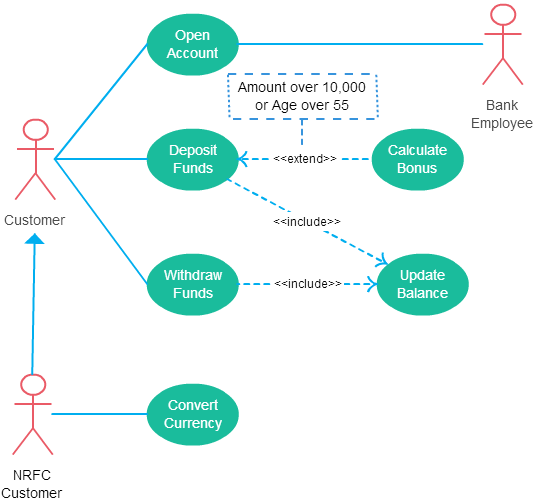
Sự phân biệt này giúp chúng ta tinh chỉnh Use case từ mục tiêu tổng quát thành các tương tác cụ thể giữa hệ thống và tác nhân, giúp xây dựng hệ thống rõ ràng và dễ hiểu hơn.

**3. Các thành phần của lược đồ ca sử dụng**  
Lược đồ Use case bao gồm những thành tố quan trọng như:

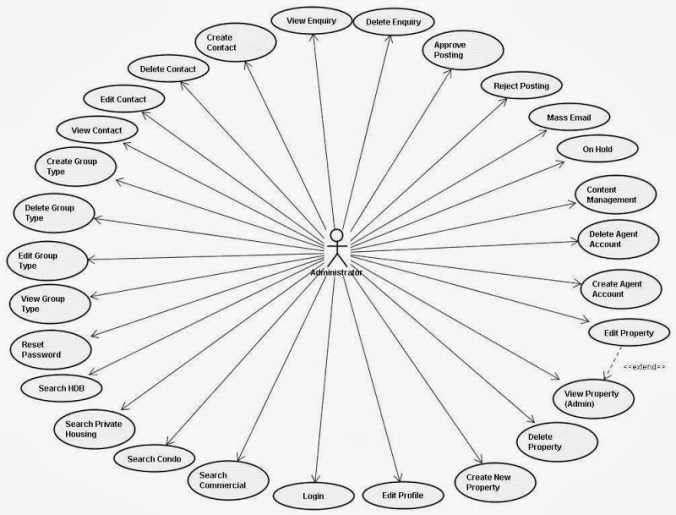
* **Ranh giới hệ thống (system sýps tèm boundary bấu đờ ri):** Giới hạn những gì hệ thống sẽ xử lý.
* **Tác nhân (actor ác tò):** Đại diện cho người dùng hoặc các hệ thống khác tương tác với hệ thống.
* **Usecase:** Là các chức năng cụ thể mà hệ thống cung cấp để đáp ứng mục tiêu của tác nhân.



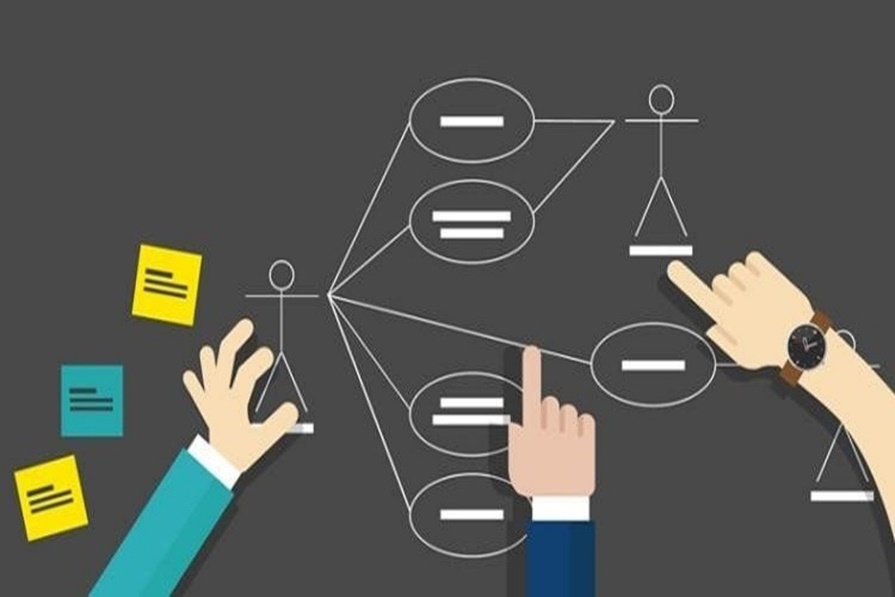
* **Ví dụ chi tiết hơn:** Trong hệ thống quản lý thư viện, ranh giới hệ thống bao gồm việc mượn, trả, và quản lý sách, tạp chí. Các tác nhân chính là độc giả và thủ thư. Ranh giới rõ ràng giúp tránh việc mở rộng hệ thống sang các lĩnh vực không cần thiết như quản lý tài liệu hành chính hoặc hỗ trợ sự kiện.
  1. **Mối quan hệ giữa các ca sử dụng**  
     Trong một hệ thống phức tạp, các Use case có thể liên kết với nhau, và mối quan hệ giữa các Use case thường có hai dạng chính:



* **Include (bao hàm):** Use case A có thể bao hàm một Use case khác (B), tức là một phần của Use case A sẽ yêu cầu thực hiện Use case B.
* **Extend (mở rộng):** Use case A có thể được mở rộng thêm các bước, khi cần thực hiện một số chức năng bổ sung.
* **Ví dụ minh họa:** Trong hệ thống quản lý thư viện, Use case "Mượn sách" có thể bao hàm Use case "Xác thực thông tin độc giả", vì trước khi mượn sách, hệ thống cần kiểm tra thông tin của người mượn. Trong khi đó, Use case "Gia hạn sách" có thể mở rộng Use case "Mượn sách" để xử lý việc gia hạn thời gian mượn thêm.
  1. **Vai trò của ranh giới hệ thống trong phát triển phần mềm**

  
Việc xác định ranh giới hệ thống là một bước quan trọng để đảm bảo rằng hệ thống chỉ xử lý các chức năng cần thiết, tránh việc mở rộng không cần thiết và phức tạp hóa hệ thống. Nếu không xác định ranh giới rõ ràng, hệ thống có thể bị phình to và khó quản lý.

* **Phân tích thêm:** Khi phát triển hệ thống quản lý thư viện, ranh giới hệ thống sẽ được xác định dựa trên các yêu cầu của thư viện, như mượn, trả và kiểm soát kho sách. Những chức năng này cần được tách biệt với các hệ thống khác như hệ thống quản lý tài chính hoặc nhân sự của thư viện, để tránh phức tạp hóa hệ thống.
  1. **Tầm quan trọng của Use case trong phát triển phần mềm**

  
Use case không chỉ giúp mô tả yêu cầu chức năng của hệ thống mà còn đóng vai trò quan trọng trong việc kiểm thử và đánh giá hệ thống sau này. Một hệ thống thành công thường là kết quả của việc đầu tư nhiều thời gian vào việc phân tích và xác định Use case một cách rõ ràng.

* **Tăng cường phân tích:** Khi sử dụng Use case để phát triển hệ thống quản lý thư viện, chúng ta có thể xác định một cách chính xác các yêu cầu chức năng từ người dùng. Điều này giúp cho việc phát triển hệ thống trở nên hiệu quả, tiết kiệm thời gian và chi phí, đồng thời tăng độ chính xác trong việc đáp ứng yêu cầu của khách hàng.
  1. **Mở rộng về kỹ thuật " Use case Driven d rí vờn Development đì ve lóp mờn t" (Phát triển phần mềm dựa trên ca sử dụng)**

  
Phát triển phần mềm dựa trên Use case (Use case Driven **d rí vờ**n Development **đì ve lóp mờn t**) là một phương pháp tiếp cận phổ biến trong kỹ nghệ phần mềm hiện đại. Phương pháp này tập trung vào việc sử dụng các Use case làm cơ sở để thiết kế và phát triển hệ thống.

* **Phân tích thêm:** Phương pháp này giúp đảm bảo rằng hệ thống được phát triển theo đúng yêu cầu của người dùng, đồng thời dễ dàng kiểm thử và bảo trì. Khi mỗi Use case được xác định rõ ràng, nhóm phát triển có thể dễ dàng hình dung cách hệ thống sẽ vận hành và tương tác với các tác nhân bên ngoài.

**Kết luận** Use case đóng vai trò nền tảng trong kỹ nghệ phần mềm, giúp đảm bảo rằng hệ thống được phát triển theo đúng yêu cầu chức năng và phù hợp với mục tiêu của người dùng. Thông qua việc sử dụng lược đồ Use case và các kỹ thuật liên quan, nhóm phát triển có thể tạo ra một hệ thống mạnh mẽ, linh hoạt và dễ mở rộng trong tương lai.

**Phần kết thúc mở:** Với sự phát triển không ngừng của công nghệ, Use case ngày càng trở nên quan trọng hơn trong việc xác định rõ ràng các yêu cầu của người dùng, góp phần vào việc xây dựng các hệ thống phần mềm hiệu quả và có tính khả thi cao.

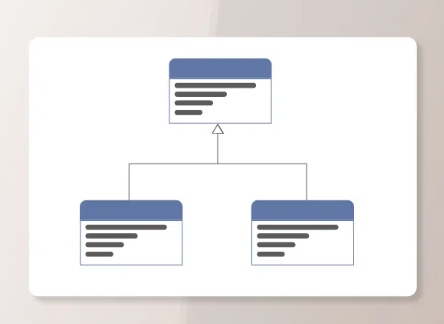
**### 3. Mô hình hóa các lớp đối tượng tĩnh (Class Modeling)**

* 1. **Giới thiệu về phát triển phần mềm hướng đối tượng (OOSD)**

  
Hiện nay, **phát triển phần mềm định hướng đối tượng (Object-oriented (o rì en tịt) software(sóp què) development(đì ve lóp mờn) - OOSD)** là phương pháp phổ biến và tiên tiến nhất trong công nghiệp phần mềm. OOSD cho phép các nhà phát triển xây dựng phần mềm bằng cách mô hình hóa thế giới thực qua các **đối tượng** và **lớp** (class). Mô hình này phản ánh thực tế một cách gần gũi và dễ hiểu nhất thông qua việc đưa vào các yếu tố như thuộc tính và hành vi của các đối tượng.

Hai yếu tố then chốt đưa OOSD đến vị trí này là: khả năng phản ánh các vấn đề thế giới thực qua việc trừu tượng hóa đối tượng và quá trình phát triển phần mềm thông qua các bước lặp tinh chỉnh từ mức độ cao xuống mức thực thi cụ thể. Quá trình này giảm thiểu sự khác biệt ngữ nghĩa giữa các mô hình đặc tả và mô hình thực thi bằng cách thực hiện liên tục việc chuyển đổi từ các khái niệm trừu tượng sang các chi tiết cụ thể.

* 1. **Phân biệt lớp và đối tượng trong Class Modeling**  
     Một nhóm các đối tượng tương tự nhau trong thế giới thực được trừu tượng hóa thành một **lớp** trong mô hình lớp. Các đối tượng (objects) là các thể hiện cụ thể của lớp và mang các giá trị cụ thể cho các thuộc tính. Ví dụ, lớp Person có thể có các thuộc tính như name, age, và các hành vi như speak(), nhưng đối tượng John sẽ có tên cụ thể và tuổi cụ thể.



* **Lớp (Class)**: Là tập hợp các đối tượng có thuộc tính và hành vi giống nhau. Lớp là một trừu tượng hóa của các đối tượng. Ví dụ: Person là lớp đại diện cho các cá nhân.
* **Đối tượng (Object)**: Là thể hiện cụ thể của lớp, có các giá trị cụ thể cho từng thuộc tính. Ví dụ, John là một đối tượng của lớp Person.

**3. Các khái niệm quan trọng trong Class Modeling**

* **Phân lớp (Classification cờ lát si phi cấy sờn)**: Là quá trình trừu tượng hóa các thực thể có cùng cấu trúc và hành vi thành một lớp đại diện.
* **Đa hình (Polymorphism po lì mó phì sừm)**: Một hành vi có thể thể hiện khác nhau trên các đối tượng thuộc cùng một lớp.
* **Kế thừa (Inheritance ìn hé ri tềnh sss)**: Một lớp con có thể kế thừa thuộc tính và phương thức từ lớp cha, tạo ra mối quan hệ phân cấp giữa các lớp.
* **Tổng hợp (Aggregation ác ri rấy sờn)**: Là một loại quan hệ giữa các lớp, trong đó một lớp là thành phần của lớp khác. ~~Ví dụ: Một chiếc xe hơi bao gồm các bánh xe.~~

**4. Mối quan hệ giữa các lớp trong Class Modeling**  
Mối quan hệ giữa các lớp thể hiện qua các liên kết trong sơ đồ lớp. Ví dụ, mối quan hệ “Works for” giữa lớp Person và lớp Company mô tả một cá nhân làm việc cho một công ty cụ thể. Các mối quan hệ giữa các lớp được phân loại theo nhiều dạng như:

* **Liên kết (Association ạt sô sì ấy sờn)**: Mô tả sự tương tác giữa các lớp. Ví dụ, một Employee làm việc cho một Company.
* **Tổng hợp (Aggregation ác ri rấy sờn)**: Một lớp là một phần của lớp khác, chẳng hạn như một đoạn văn chứa nhiều câu.
* **Kế thừa (Generalization chén nờ lai zấy sờn/Inheritance ìn hé ri tềnh sss)**: Một lớp có thể được kế thừa từ một lớp cha, ~~tạo ra mối quan hệ "is-a."~~ Ví dụ, Manager là một loại Employee.

**5. Kết luận**  
**Class Modeling** là một phần không thể thiếu trong OOSD, giúp các nhà phát triển phần mềm mô hình hóa rõ ràng cấu trúc và mối quan hệ giữa các đối tượng. Với sự hỗ trợ của UML, quá trình này trở nên trực quan và dễ hiểu hơn, từ đó giúp thiết kế, lập trình và bảo trì phần mềm hiệu quả. Việc nắm bắt các khái niệm như lớp, đối tượng, phân lớp, đa hình, kế thừa, tổng hợp không chỉ giúp hiểu rõ hơn về hệ thống mà còn tạo điều kiện để xây dựng các ứng dụng phức tạp theo cách có tổ chức và dễ quản lý.