Chương 8

Thiết kế hướng đối tượng

8.1 Nhiệm vụ của thiết kế

- Cụ thể hóa, chi tiết hóa các bản phát họa cách thức giải quyết chức năng tương ứng. Nếu dùng kỹ thuật thiết kế hướng đối tượng, bản thiết kế cách giải quyết chức năng là các class đối tượng cụ thể, mối quan hệ giữa chúng và các thông tin cụ thể, chi tiết kèm theo. Thí dụ mỗi class đều có tên, có các thuộc tính chi tiết và các tác vụ chức năng (có thể kèm theo giải thuật của tác vụ đó)
- Workflow thiết kế sẽ cụ thể hóa, chi tiết hóa tất cả các bản phát họa cách giải quyết mọi yêu cầu chức năng của hệ thống phần mềm.
- Workflow thiết kế cũng sẽ đặc tả được kiến trúc cụ thể, chi tiết của hệ thống phần mềm.
- Toàn bộ các artifacts được tạo ra và duy trì trong workflow thiết kế được gọi là mô hình thiết kế và mô hình triển khai.

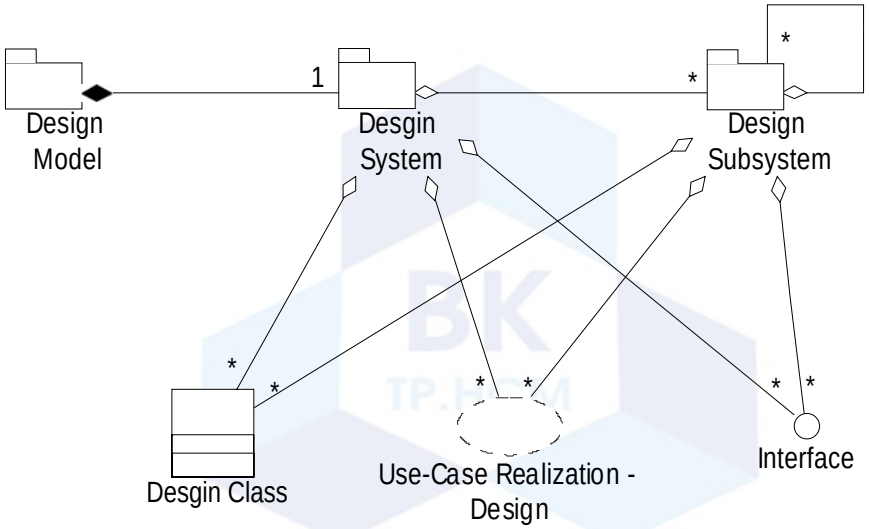
8.1 Nhiệm vụ của thiết kế

- Mục đích của các artifacts được tạo ra trong workflow thiết kế là :
  - Giúp nắm bắt các hệ thống con, các class thiết kế, interface giữa chúng (interface ↔ interface, interface ↔ class class ↔ class).
  - Giúp ta xem xét dễ dàng bảng thiết kế bằng cách dùng các ký hiệu của ngôn ngữ đặc tả để miêu tả, hiển thị artifacts.
  - Giúp người nghiên cứu hệ thống đạt được sự hiểu biết sâu sắc các ràng buộc, các yêu cầu không chức năng liên quan đến ngôn ngữ lập trình được dùng để hiện thực, việc dùng lại linh kiện có sẵn, HĐH, công nghệ phân tán, xử lý đồng thời, database, giao diện, quản lý giao tác…
  - Tạo ra mức trừu tượng để làm đầu vào trực tiếp cho hoạt động hiện thực hệ thống phần mềm.

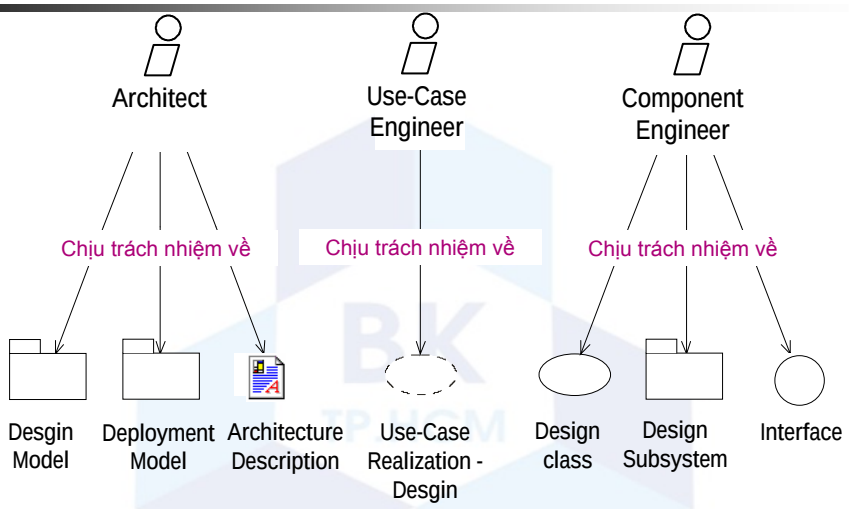
8.2 Các artifacts cần tạo ra

- Mô hình thiết kế = hệ thống các kết quả thiết kế, nó chứa :
  - các hệ thống con, nếu có, mỗi hệ thống con chứa :
    - các dẫn xuất use-case ở cấp thiết kế, mỗi dẫn xuất chứa :
      - các lược đồ class ở cấp thiết kế.
      - các lược đồ tương tác giữa các đối tượng cấp thiết kế.
      - 'flow of events' ở cấp thiết kế.
      - các yêu cầu đặc biệt của từng use-case, hay của toàn bộ các use-case cho workflow hiện thực.
  - Đặc tả kiến trúc hệ thống phần mềm theo góc nhìn thiết kế (view of design model)
- Mô hình triển khai :
  - Sẽ là đặc tả kiến trúc phần mềm theo góc nhìn triển khai.

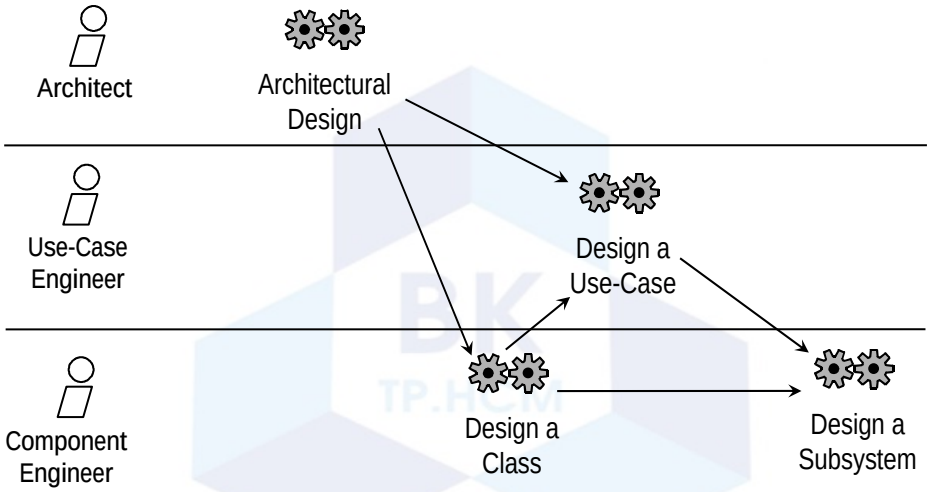
8.2 Các artifacts cần tạo ra



8.3 Các worker tham gia thiết kế



8.4 Qui trình thiết kế



8.5 Thiết kế kiến trúc

- Nhiệm vụ của hoạt động thiết kế kiến trúc là xây dựng mô hình thiết kế, mô hình triển khai và kiến trúc của hệ thống phần mềm theo 2 góc nhìn tương ứng.
- Để xây dựng mô hình triển khai, ta nhận dạng các thông tin sau :
  - Các nút tính toán và các cấu hình mạng của chúng.
- Để phục vụ xây dựng mô hình thiết kế, ta nhận dạng các thông tin sau :
  - Các hệ thống con và interface của chúng.
  - Các class thiết kế có ý nghĩa kiến trúc (như class chủ động).
  - Các cơ chế thiết kế tổng quát để xử lý các yêu cầu chung như tính bền vững, tính hiệu quả… mà ta đã nắm bắt được trong workflow phân tích.

8.5 Thiết kế kiến trúc

Nhận dạng các nút và cấu hình mạng nối kết

- Cấu hình mạng vật lý sẽ ảnh hưởng đến kiến trúc phần mềm, gồm các khía cạnh sau :
  - Các nút nào liên quan, khả năng về bộ nhớ và công suất tính toán của từng nút.
  - Kiểu kết nối và giao thức giữa các nút.
  - Các tính chất của kết nối và giao thức như băng thông, độ sẵn sàng, chất lượng,…
  - Mức độ cần thiết của tính dư thừa, đề kháng lỗi, di cư process, sao lưu dữ liệu…

8.5 Thiết kế kiến trúc

Nhận dạng các hệ thống con và interface của chúng

- Được thực hiện từ đầu hay khi mô hình thiết kế phát triển lên độ phức tạp cao nên cần phải chia nhỏ.
- Một số hệ thống con có thể được dùng lại từ các project khác.
- Các hoạt động cụ thể :
  - Nhận dạng các hệ thống con ở cấp ứng dụng.
  - Nhận dạng các hệ thống con cấp giữa (middleware) và cấp hệ thống.
  - Định nghĩa sự phụ thuộc giữa các hệ thống con.
  - Nhận dạng interface giao tiếp của từng hệ thống con

8.5 Thiết kế kiến trúc

Nhận dạng các hệ thống con và interface của chúng

- Các hệ thống xử lý cùng lĩnh vực thường có kiến trúc giống nhau, do đó ta có thể chọn 1 trong các mẫu kiến trúc phổ biến có sẵn để làm kiến trúc của 1 hệ thống phần mềm cần thiết kế.
- Mẫu là phương tiện miêu tả, dùng chung và dùng lại kiến thức của nhiều người. Một mẫu kiến trúc là đặc tả kiến trúc có rồi, được thiết kế tốt, đã được dùng và kiểm chứng trong nhiều ứng dụng khác nhau.

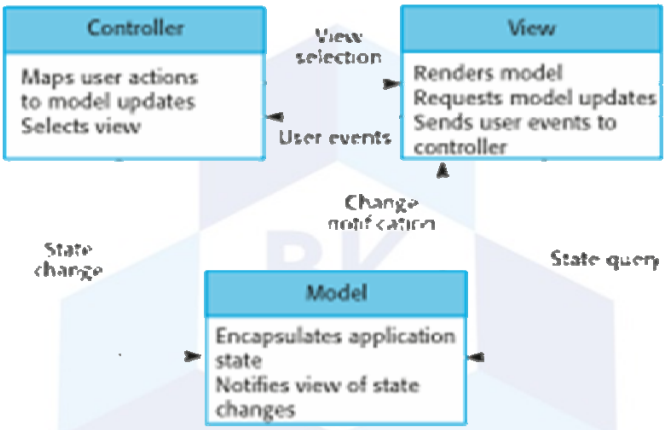
8.5 Thiết kế kiến trúc

Kiến trúc MVC (Model-View-Controller)

- **Đặc tả** : Hệ thống gồm 3 thành phần luận lý tương tác lẫn nhau :
  - Model quản lý dữ liệu và các tác vụ liên quan đến dữ liệu này.
  - View định nghĩa và quản lý cách thức dữ liệu được trình bày cho user.
  - Controller quản lý các tương tác với user như ấn phím, click chuột… và gởi thông tin tương tác này tới View và/hoặc Model.

8.5 Thiết kế kiến trúc

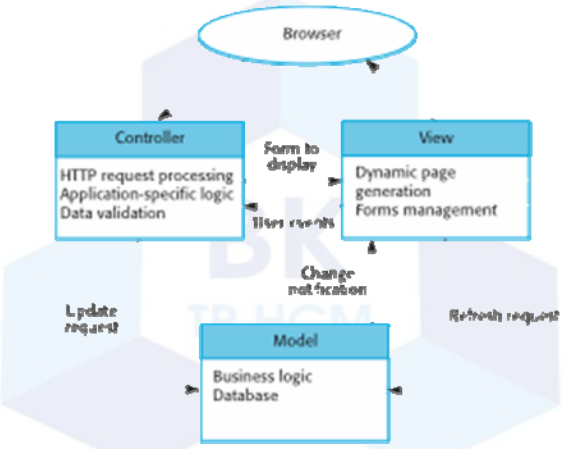
Kiến trúc MVC (Model-View-Controller)



8.5 Thiết kế kiến trúc

Kiến trúc MVC (Model-View-Controller)

- **Thí dụ** : Hệ thống web dùng kiến trúc MVC :



8.5 Thiết kế kiến trúc

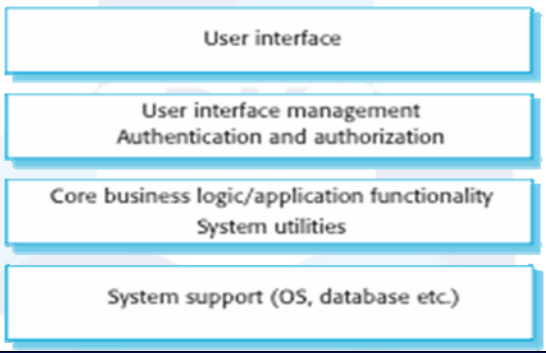
Kiến trúc MVC (Model-View-Controller)

- **Tình huống nên dùng** : Hệ thống có nhiều cách để view và tương tác với dữ liệu, hoặc ta chưa biết trước các yêu cầu tương lai về sự tương tác và biểu diễn dữ liệu của chương trình.
- **Ưu điểm** : cho phép dữ liệu thay đổi độc lập với cách thức thể hiện nó và ngược lại.
- **Khuyết điểm** : có thể cần nhiều code hơn và code có thể phức tạp hơn khi mô hình dữ liệu và sự tương tác chỉ ở mức độ đơn giản.

8.5 Thiết kế kiến trúc

Kiến trúc nhiều cấp (Layered architecture)

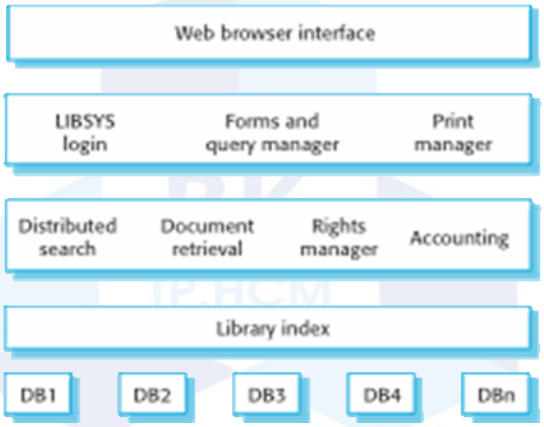
- **Đặc tả** : Hệ thống gồm nhiều cấp dạng chồng lên nhau, mỗi layer có chức năng cụ thể, rõ ràng và cung cấp các dịch vụ cho layer ngay trên mình. Các layer cấp thấp nhất chứa các dịch vụ cơ bản nhất và được dùng cho toàn hệ thống.



8.5 Thiết kế kiến trúc

Kiến trúc nhiều cấp (Layered architecture)

- **Thí dụ** : Hệ thống dùng chung các tài liệu copy ở các thư viện khác nhau.



8.5 Thiết kế kiến trúc

Kiến trúc nhiều cấp (Layered architecture)

- **Tình huống nên dùng** : xây dựng các khả năng mới trên hệ thống có sẵn, hay khi có nhiều nhóm phát triển khác nhau, mỗi nhóm chịu trách nhiệm về 1 layer chức năng cụ thể, hay khi có yêu cầu bảo mật nhiều cấp.
- **Ưu điểm** : cho phép thay đổi toàn bộ layer bất kỳ sao cho interface không đổi. Có thể giải quyết 1 chức năng nào đó (xác nhận user) ở nhiều cấp theo cách thức tăng dần.
- **Khuyết điểm** : khó tách bạch chức năng của từng cấp, layer trên khó tương tác với layer phía dưới nó nhưng không liền kề. Hiệu quả giảm sút khi nhiều layer phải tương tác nhau để giải quyết 1 chức năng nào đó.

8.5 Thiết kế kiến trúc

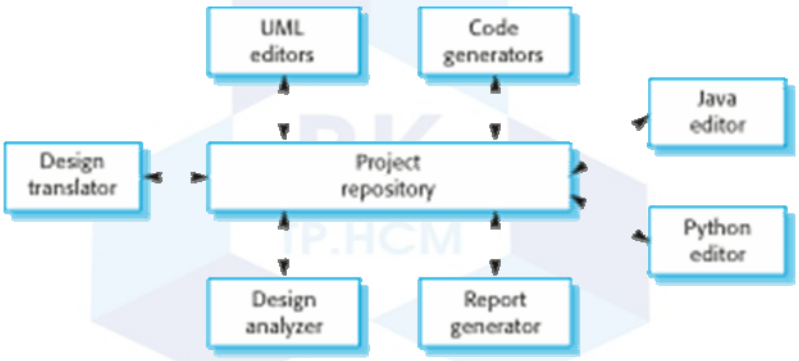
Kiến trúc kho (Repository Architecture)

- **Đặc tả** : Tất cả dữ liệu của hệ thống được quản lý trong 1 kho chứa tập trung, mọi thành phần chức năng của hệ thống đều có thể truy xuất kho chứa này. Các thành phần không tương tác trực tiếp với nhau, chỉ thông qua kho chứa tập trung.

8.5 Thiết kế kiến trúc

Kiến trúc kho (Repository Architecture)

- **Thí dụ** : Môi trường IDE gồm nhiều thành phần dùng chung kho thông tin, mỗi tool tạo thông tin và để trong kho để các tool khác dùng.



8.5 Thiết kế kiến trúc

Kiến trúc kho (Repository Architecture)

- **Tình huống nên dùng** : khi hệ thống tạo và chứa 1 lượng rất lớn thông tin trong thời gian dài, hay trong các hệ thống dựa vào dữ liệu, ở đó việc chứa thông tin vào kho sẽ kích hoạt 1 tool hay 1 chức năng hoạt động.
- **Ưu điểm** : các thành phần độc lập nhau, không ai biết gì về ai khác.
- **Khuyết điểm** : kho là điểm yếu nhất, nếu có lỗi sẽ ảnh hưởng toàn bộ các thành phần chức năng. Có vấn đề về truy xuất đồng thời kho, phân tán kho trên nhiều máy cũng khó khăn.

8.5 Thiết kế kiến trúc

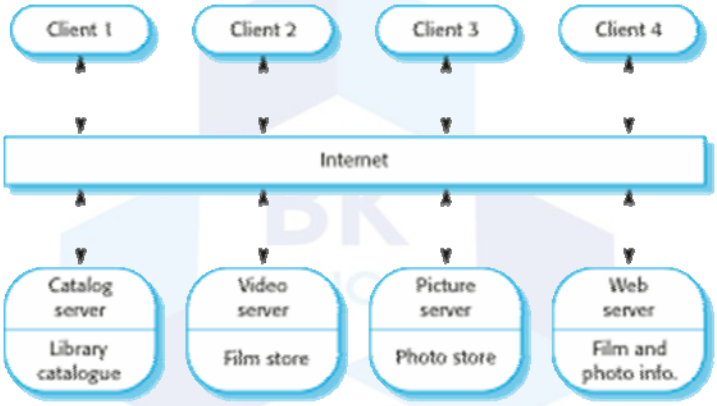
Kiến trúc client-server (client-server Architecture)

- **Đặc tả** : Hệ thống gồm 2 loại phần tử chức năng : server cung cấp 1 số dịch vụ, client là phần tử sử dụng dịch vụ bằng cách truy xuất đến server tương ứng.

8.5 Thiết kế kiến trúc

Kiến trúc client-server (client-server Architecture)

- **Thí dụ** : Hệ thống quản lý phim ảnh dùng mô hình client-server



8.5 Thiết kế kiến trúc

Kiến trúc client-server (client-server Architecture)

- **Tình huống nên dùng** : khi database dùng chung từ nhiều vị trí khác nhau hay khi tải hệ thống thay đổi động (nhân bản server thành nhiều phần tử).
- **Ưu điểm** : server có thể phân tán tự do trên mạng.
- **Khuyết điểm** : độ hiệu quả phụ thuộc vào mạng và hệ thống nên khó lường trước. Nếu các server được quản lý bởi các tổ chức khác nhau thì có vấn đề về quản lý chúng.

8.5 Thiết kế kiến trúc

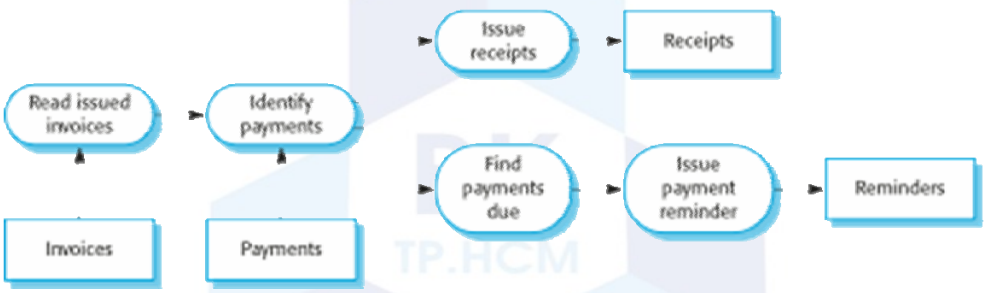
Kiến trúc đường ống và lọc (Pipe and filter Architecture)

- **Đặc tả** : Xử lý dữ liệu thông qua 1 ống gồm nhiều thành phần xử lý (filter) rời rạc và nối tiếp nhau, mỗi filter thực hiện 1 hoạt động chuyển đổi dữ liệu từ dạng đầu vào thành dạng đầu ra để thành phần sau xử lý tiếp.

8.5 Thiết kế kiến trúc

Kiến trúc đường ống và lọc (Pipe and filter Architecture)

- **Thí dụ** : Hệ thống xử lý hóa đơn đặt hàng



8.5 Thiết kế kiến trúc

Kiến trúc đường ống và lọc (Pipe and filter Architecture)

- **Tình huống nên dùng** : trong các ứng dụng xử lý dữ liệu mà dữ liệu nhập cần được xử lý bởi nhiều công đoạn khác nhau trước khi tạo ra kết quả cuối cùng (compiler).
- **Ưu điểm** : dễ dàng dùng lại từng filter của hệ thống cũ, phù hợp với nhiều hoạt động nghiệp vụ, dễ dàng nâng cấp bằng cách thêm filter mới.
- **Khuyết điểm** : định dạng dữ liệu phải thỏa mãn đồng thời bởi 2 filter liền kề : filter tạo kết quả và filter dùng kết quả đó.

8.5 Thiết kế kiến trúc

Nhận dạng các class quan trọng về mặt kiến trúc

- Cần nhận dạng các class thiết kế quan trọng về mặt kiến trúc để làm tiền đề cho hoạt động thiết kế chi tiết, các class khác sẽ được nhận dạng trong khi thiết kế từng use-case :
  - Nhận dạng các class thiết kế từ các class phân tích tương ứng.
  - Nhận dạng các class chủ động khi chú ý yêu cầu đồng thời của hệ thống :
    - Các yêu cầu về độ hiệu quả, độ sẵn sàng, công suất hệ thống
    - Sự phân tán hệ thống phần mềm trên các nút.
    - Các yêu cầu khác như khởi động, kết thúc, tránh deadlock, tránh bảo hòa, cấu hình lại các nút, khả năng nối kết…

8.5 Thiết kế kiến trúc

Nhận dạng các cơ chế thiết kế tổng quát

- Từ các yêu cầu chung và đặc biệt đã được nhận dạng trong workflow phân tích, ta quyết định xử lý chúng dựa trên công nghệ thiết kế và hiện thực sẵn có. Kết quả là 1 tập các cơ chế thiết kế tổng quát. Các yêu cầu cần xử lý thường liên quan đến :
  - Tính bền vững.
  - Sự phân tán và đồng thời.
  - Các tính chất an toàn dữ liệu.
  - Đề kháng với lỗi.
  - Quản lý giao tác.

8.6 Thiết kế từng use-case

- Nhiệm vụ thiết kế use-case là để :
  - Nhận dạng các hệ thống con và các class thiết kế có đối tượng của mình tham gia vào việc thực hiện các hoạt động tồn tại trong “flow of events ở cấp thiết kế” của use-case tương ứng.
  - Thể hiện sự tương tác giữa các hệ thống con, giữa hệ thống con với đối tượng thiết kế, giữa các đối tượng thiết kế trong việc thực hiện use-case thông qua các lược đồ động như lược đồ trình tự, lược đồ cộng tác, lược đồ hoạt động, lược đồ trạng thái.
  - Nhận dạng thêm 1 số yêu cầu đặc biệt và phi chức năng cho việc thực hiện từng tác vụ, từng class và từng hệ thống con.

8.6 Thiết kế từng use-case

Nhận dạng các class thiết kế thực hiện 1 use-case

- Ở bước này, ta sẽ :
  - Nghiên cứu các class biên, thực thể, điều khiển trong dẫn xuất use-case cấp phân tích tương ứng để nhận dạng các class thiết kế được dẫn xuất từ class phân tích này.
  - Nghiên cứu các yêu cầu đặc biệt trong dẫn xuất use-case cấp phân tích tương ứng để nhận dạng các class thiết kế hiện thực được các yêu cầu đặc biệt này.
  - Có thể liên hệ với kiến trúc sư và kỹ sư linh kiện để bàn bạc hầu nhận dạng thêm các class khác.

8.6 Thiết kế từng use-case

Nhận dạng các class thiết kế thực hiện 1 use-case

- Ở bước này, ta sẽ (tt) :
  - Gán nghĩa vụ cho từng class thiết kế tìm được.
  - Xác định cụ thể các mối quan hệ giữa các class thiết kế tìm được.
  - Tập hợp các class tìm được, mối quan hệ giữa chúng thành 1 hay nhiều lược đồ class. Các lược đồ class này sẽ là nội dung thiết yếu để xây dựng dẫn xuất use-case tương ứng.

8.6 Thiết kế từng use-case

Xây dựng các lược đồ trình tự

- Lược đồ trình tự chứa các phần tử actor, đối tượng thiết kế và các thông báo được gởi giữa chúng theo thứ tự thời gian.
- Nếu use-case có nhiều luồng điều khiển khác nhau, ta nên tạo lược đồ trình tự cho từng luồng.
- Nên chuyển lược đồ cộng tác ở cấp phân tích thành lược đồ trình tự ban đầu, từ đó phát triển thêm chi tiết.
- Dùng "flow of events ở cấp thiết kế", duyệt các bước trong nó để xác định các thông báo cần thiết giữa actor và đối tượng thiết kế, hay giữa 2 đối tượng thiết kế.

8.6 Thiết kế từng use-case

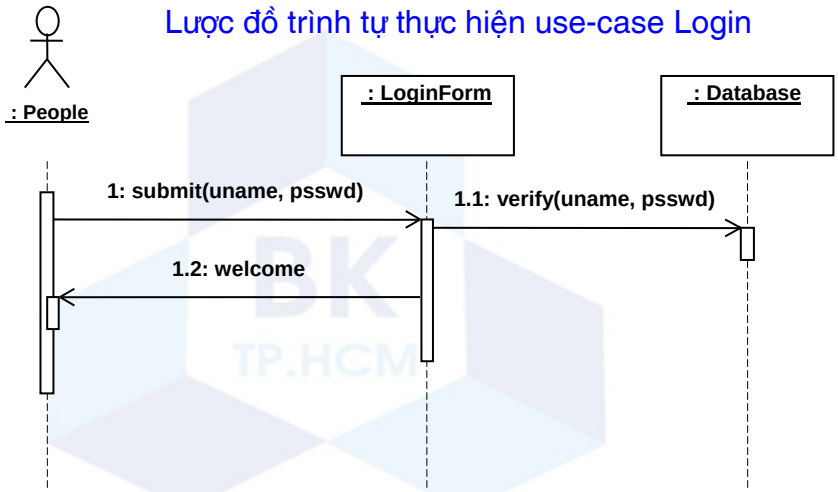
Xây dựng các lược đồ trình tự

- Cần chú ý các điểm sau trong việc xây dựng các lược đồ trình tự :
  - Nên tập trung thông tin thiết yếu là trình tự các thông báo.
  - Thường lúc đầu, actor sẽ gởi thông báo đến đối tượng thiết kế để yêu cầu thực hiện use-case.
  - Mỗi class thiết kế trong lược đồ class nên có ít nhất 1 đối tượng tham gia vào lược đồ trình tự.
  - Các thông báo được vẽ từ đường đời sống đối tượng gởi đến đường đời sống đối tượng nhận, có thể gán tên tạm, sau này nó trở thành tên tác vụ tượng ứng (của đối tượng nhận).
  - Lược đồ trình tự nên xử lý tất cả các mối quan hệ của use-case cần hiện thực.

8.6 Thiết kế từng use-case

Xây dựng các lược đồ trình tự

Lược đồ trình tự thực hiện use-case Login



8.7 Thiết kế từng class

- Nhiệm vụ của việc thiết kế từng class là tạo ra được class thiết kế hoàn thành vai trò của nó trong dẫn xuất use-case và các yêu cầu phụ, tập trung các thông tin sau :
  - Các tác vụ & vá các method kèm theo (thuật giải hiện thực).
  - Các thuộc tính
  - Các mối quan hệ với các phần tử khác.
  - Các trạng thái quan trọng của đối tượng thuộc class đó.
  - Các yêu cầu đặc biệt có liên quan đến hiện thực class đó.
  - Đảm bảo hiện thực đúng từng interface mà class phải hiện thực.

8.7 Thiết kế từng class

- Từ 1 interface, ta nhận dạng ít nhất 1 class thiết kế hiện thực nó.
- Từ 1 class phân tích :
  - Thiết kế class biên phụ thuộc công nghệ tạo giao diện : Form, ActiveX… Để ý dùng các prototype giao diện trong nắm bắt yêu cầu.
  - Thiết kế class thực thể thường phụ thuộc vào công nghệ database. Việc ánh xạ từ mô hình hướng đối tượng sang mô hình dữ liệu quan hệ có thể cần worker, mô hình và công việc riêng.

8.7 Thiết kế từng class

- Khi thiết kế class điều khiển nên chú ý các nhu cầu sau :
  - Phân tán : cần nhiều class thiết kế trên các nút khác nhau để thực hiện 1 class điều khiển.
  - Hiệu quả : nên dùng chỉ 1 class thiết kế cho 1 class điều khiển.
  - Giao tác : class thiết kế cần tích hợp công nghệ quản lý giao tác.

8.7 Thiết kế từng class

Nhận dạng các tác vụ cho class thiết kế

- Dựa vào các thông tin đầu vào sau :
  - Các trách nhiệm của bất kỳ class phân tích nào mà dẫn đến việc phát sinh class thiết kế. Mỗi trách nhiệm tương ứng với 1 hay nhiều tác vụ cụ thể, nhưng ở đây mới phát họa thông số hình thức các tham số.
  - Các yêu cầu đặc biệt của bất kỳ class phân tích nào mà dẫn đến class thiết kế.
  - Các interface mà class thiết kế phải cung cấp.
  - Dẫn xuất use-case cấp thiết kế mà class tham gia.

8.7 Thiết kế từng class

Nhận dạng các thuộc tính cho class thiết kế

Dựa vào các thông tin đầu vào sau :

- Các thuộc tính của bất kỳ class phân tích nào mà dẫn đến việc phát sinh class thiết kế. Mỗi thuộc tính cấp phân tích ám chỉ 1 hay nhiều thuộc tính cụ thể.
- Mỗi tác vụ thiết kế có thể cần 1 hay nhiều thuộc tính dữ liệu.
- Cố gắng dùng kiểu đã có cho thuộc tính mới. Nếu không có ý định sinh mã tự động thì hạn chế dùng kiểu cụ thể của ngôn ngữ lập trình. Còn nếu muốn sinh mã tự động thì phải dùng kiểu cụ thể của ngôn ngữ dự định viết code.
- Nếu có quá nhiều thuộc tính, có thể tách riêng từng nhóm thành các class riêng và dùng lược đồ class riêng để miêu t3 mối quan hệ giữa các class phát sinh này.

8.7 Thiết kế từng class

Nhận dạng các mối quan hệ của class thiết kế

Dựa vào các hướng dẫn sau :

- Chú ý các mối quan hệ kết hợp và bao gộp của bất kỳ class phân tích nào dẫn tới class thiết kế này.
- Tinh chế số lượng phân tử tham gia ở mỗi đầu, tên vai trò, tính chất của vai trò, class kết hợp, kết hợp n-ary.
- Tinh chế hướng của mối quan hệ kết hợp từ lược đồ tương tác.
- Nhận dạng và miêu tả mối quan hệ tổng quát hóa. Nếu ngôn ngữ lập trình dự định dùng không hỗ trợ tính thừa kế, dùng mối quan hệ kết hợp hay bao gộp để thay thế.

8.7 Thiết kế từng class

Xác định method cho từng tác vụ

- Method đặc tả cách tác vụ được hiện thực. Có thể dùng ngôn ngữ tự nhiên hay ngôn ngữ pseudo-code. Nếu kỹ sư linh kiện thực hiện cả 2 khâu thiết kế và hiện thực thì họ thường ít khi đặc tả method cho tác vụ trong workflow thiết kế, họ chỉ làm tại thời điểm viết code.
- Một vài đối tượng thiết kế được điều khiển hoạt động bởi trạng thái : trạng thái hiện hành se xác định hành vi của đối tượng khi xử lý thông báo từ ngoài gởi đến. Trong trường hợp này ta nên miêu tả chi tiết các sự chuyển trạng thái của đối tượng bằng lược đồ chuyển trạng thái.
- Tiếp tục xử lý các yêu cầu đặc biệt chưa được chú ý ở các bước trước.

8.8 Thiết kế các hệ thống con

- Nhiệm vụ của việc thiết kế các hệ thống con là :
  - Đảm bảo các hệ thống con độc lập với nhau nhiều như có thể có (tính phụ thuộc thấp nhất).
  - Đảm bảo interface của hệ thống con độc lập nhiều như có thể có với chi tiết bên trong nó.
  - Đảm bảo hệ thống con cung cấp chính xác interface theo yêu cầu.
  - Đảm bảo hệ thống con hoàn thành mục đích, hiện thực đúng cho các tác vụ.

8.9 Kết chương

- Chương này đã giới thiệu các thông tin cơ bản về workflow thiết kế như nhiệm vụ, các artifact cần tạo ra, các worker tham gia, qui trình thực hiện. Chương này cũng đã giới thiệu chi tiết về hoạt động thiết kế kiến trúc phần mềm, thiết kế từng use-case, thiết kế từng class, thiết kế các hệ thống con.