HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Khoa Điện tử Viễn thông

Nội dung : Thiết kế và cài đặt

Giảng viên: TS. Lâm Đức Dương

Nội dung

1. Thiết kế hướng đối tượng sử dụng UML
2. Các mẫu thiết kế
3. Các vấn đề triển khai
4. Phát triển mã nguồn mở

Nội dung

1. Quy trình thiết kế hệ thống phần mềm
2. Xác định cấu trúc
3. Thiết kế thành phần
4. Thiết kế giao diện

Vai trò và Tầm quan trọng của Thiết kế Phần mềm

- Thiết kế là một giai đoạn quan trọng, bắc cầu khoảng cách giữa các yêu cầu và triển khai (viết mã).
- Nó rất cần thiết cho chất lượng của sản phẩm phần mềm cuối cùng.
- Các mô hình thiết kế mô tả kiến trúc, giao diện và các thành phần cần thiết để triển khai phần mềm.
- Thiết kế phải xác định cách hệ thống thực thi và cách các yêu cầu được hiện thực hóa.
- Thiết kế khác với mã; thiết kế không phải là mã, và mã không phải là thiết kế.
- Cung cấp thông tin đầy đủ cho việc bảo trì sau này là một vai trò của thiết kế.
- Chất lượng thiết kế nên được đánh giá khi nó đang được tạo ra, không chỉ khi phát sinh vấn đề.

Đặc điểm của Thiết kế Phần mềm Tốt

- Một thiết kế tốt thể hiện sự gắn kết cao trong các mô-đun và sự liên kết lỏng lẻo giữa các mô-đun.
- Nó nên thực hiện tất cả các yêu cầu từ mô hình phân tích.
- Nó phân mô-đun hệ thống một cách hiệu quả.
- Nó nên hoàn chỉnh, cung cấp một bức tranh đầy đủ về phần mềm.

Các Mô hình Thiết kế và Mức độ Trừu tượng

- Thiết kế phần mềm bao gồm việc sử dụng các mô hình khác nhau ở các mức độ trừu tượng khác nhau.

Thiết kế Kiến trúc

- Xác định cấu trúc tổng thể của hệ thống, xác định các hệ thống con và mối quan hệ của chúng.
- Đây là mô hình có mức độ trừu tượng cao nhất.
- Thiết kế kiến trúc xác định các hệ thống con của hệ thống và mối quan hệ của chúng. Nó thường không xác định chi tiết thuật toán hoặc chi tiết thực thi của từng thành phần.
- Các kiểu/mẫu kiến trúc phổ biến bao gồm Luồng dữ liệu, Gọi và trả về, Phân lớp, Máy khách-Máy chủ và Kho dữ liệu.
- Các chế độ xem được sử dụng trong phân tích kiến trúc bao gồm Luồng dữ liệu, Mô-đun và Quy trình.
- Đánh giá chất lượng thiết kế kiến trúc nên tập trung vào khả năng truy cập, độ tin cậy và chức năng.

Thiết kế Dữ liệu

- Xác định cấu trúc và tổ chức dữ liệu, bao gồm chi tiết các cấu trúc dữ liệu được sử dụng để triển khai.
- Nó bao gồm việc chuyển đổi thiết kế dữ liệu logic (như ERD) thành thiết kế dữ liệu vật lý.

Thiết kế Giao diện

- Chỉ định cách các mô-đun, hệ thống con và người dùng sẽ tương tác với hệ thống.
- Bao gồm cả thiết kế giao diện người dùng và thiết kế giao diện giữa các thành phần.

Thiết kế Thành phần

- Chi tiết logic nội bộ và cấu trúc dữ liệu của các thành phần hoặc mô-đun riêng lẻ.
- Nó bao gồm việc thiết kế cách các dịch vụ của một hệ thống con được phân bổ cho các thành phần cấu thành của nó.
- Các mô hình dựa trên kịch bản (như trường hợp sử dụng) có thể được sử dụng trong thiết kế thành phần.

Các loại Trừu tượng khác

- Các loại trừu tượng khác được sử dụng trong thiết kế phần mềm bao gồm
  - **Trừu tượng hóa điều khiển** (biểu diễn luồng điều khiển)
  - **Trừu tượng hóa thủ tục** (biểu diễn các hoạt động).
- Hệ thống phân cấp điều khiển minh họa mối quan hệ tổ chức và giao tiếp giữa các mô-đun.

Các Nguyên tắc và Khái niệm Thiết kế

- **Liên kết (Coupling):** Đo mức độ phụ thuộc lẫn nhau giữa các mô-đun. Liên kết thấp là mong muốn.
- **Gắn kết (Cohesion):** Đo lường mức độ các yếu tố trong một mô-đun có liên quan chặt chẽ với nhau. Gắn kết cao (ví dụ: gắn kết chức năng, nơi tất cả các yếu tố đóng góp vào một chức năng duy nhất) là mong muốn. Gắn kết thời gian đề cập đến các yếu tố được kích hoạt cùng lúc được nhóm lại với nhau.
- **Thiết kế Hướng luồng Dữ liệu:** Chuyển đổi sơ đồ luồng dữ liệu thành một thiết kế kiến trúc. Một luồng giao dịch được đặc trưng bởi một đầu vào duy nhất kích hoạt một trong nhiều đường dẫn có thể có. Một luồng chuyển đổi bao gồm các phép biến đổi dữ liệu tuần tự.
- **Tinh chỉnh (Refinement):** Quá trình xây dựng các mô hình thiết kế trừu tượng thành các mô hình chi tiết hơn.

Thiết kế Giao diện Người dùng (UI)

- Các nguyên tắc chính bao gồm **đa dạng người dùng**, **tính nhất quán**, **khả năng phục hồi**, **tính đơn giản** và **tính linh hoạt**.
- **Tính nhất quán** ngụ ý các kỹ thuật nhập liệu tương tự và giao diện tổng thể nhất quán trong suốt ứng dụng.
- Các nguyên tắc để giảm thiểu tải bộ nhớ người dùng bao gồm cung cấp các lối tắt trực quan, hiển thị thông tin dần dần và cài đặt mặc định có ý nghĩa.
- Thiết kế giao diện người dùng nên cho phép người dùng kiểm soát tương tác, chẳng hạn như cho phép ngắt quãng và hoàn tác các hành động.
- Các kỹ thuật cho thiết kế giao diện người dùng bao gồm **phân tích người dùng** và **phân tích tác vụ**. Phân tích tác vụ bao gồm quan sát các thao tác của người dùng. Phân tích người dùng có thể bao gồm bảng câu hỏi hoặc nghiên cứu các hệ thống tự động liên quan.
- Các vấn đề thiết kế giao diện người dùng phổ biến bao gồm **xử lý lỗi** và **thời gian phản hồi của hệ thống**.
- Xây dựng nguyên mẫu là một cách hợp lệ và được khuyến nghị để tránh phát triển một giao diện người dùng kém. Môi trường phát triển giao diện người dùng thường cung cấp các công cụ để xây dựng nguyên mẫu.
- **Mô hình người dùng** đại diện cho hồ sơ của người dùng cuối của hệ thống. **Hình ảnh hệ thống** đại diện cho giao diện và thông tin hỗ trợ của hệ thống. **Mô hình tinh thần của người dùng** đại diện cho cách người dùng hiểu hệ thống.
- **Giao diện người dùng đồ họa (GUI)** được đặc trưng bởi nhiều cửa sổ và khả năng tương tác song song. **Giao diện dòng lệnh (CLI)** đơn giản hơn, thường bao gồm các thao tác tuần tự thông qua nhập liệu bàn phím.

Biểu diễn và Đánh giá Thiết kế

- Thiết kế có thể được biểu diễn bằng các sơ đồ UML (Lớp, Trình tự, Thành phần, Triển khai), Ngôn ngữ Thiết kế Chương trình (PDL) hoặc bảng quyết định.
- PDL thường là sự kết hợp của các cấu trúc lập trình và văn bản tường thuật.
- Bảng quyết định được sử dụng để biểu diễn các tập hợp điều kiện phức tạp và các hành động kết quả.
- Đầu ra của giai đoạn thiết kế hệ thống thường là một tài liệu thiết kế chi tiết.
- Đánh giá chất lượng thiết kế bao gồm việc đánh giá sự tuân thủ các nguyên tắc, tính mô-đun của nó và liệu nó có đáp ứng các yêu cầu hay không.
- Ma trận truy vết được sử dụng để xác minh rằng tất cả các yêu cầu được giải quyết trong thiết kế.

Độ phức tạp Cyclomatic (Cyclomatic Complexity)

- Là một số liệu được sử dụng để đo độ phức tạp của luồng điều khiển của một mô-đun. Thay vì đo độ phức tạp về số dòng mã, nó tập trung vào độ phức tạp của luồng điều khiển bên trong mã.

1. Đo độ phức tạp của luồng điều khiển

- Mỗi khi có một quyết định (ví dụ: `if`, `while`, `for`, `switch`, `case`, `AND`, `OR` trong điều kiện), luồng thực thi của chương trình có thể rẽ nhánh. Độ phức tạp Cyclomatic sẽ định lượng số lượng các nhánh rẽ này.
- Một mô-đun có độ phức tạp Cyclomatic cao thường khó hiểu, khó kiểm thử và dễ xảy ra lỗi hơn so với một mô-đun có độ phức tạp thấp.

2. Cung cấp thông tin về số lượng đường dẫn logic độc lập

- **Đường dẫn logic độc lập** là một đường đi trong mã chứa ít nhất một lệnh hoặc một điều kiện chưa được thực thi trong các đường dẫn khác.
- Độ phức tạp Cyclomatic càng cao, số lượng đường dẫn độc lập càng nhiều, đồng nghĩa với việc bạn cần **càng nhiều trường hợp kiểm thử** (test case) để đảm bảo rằng tất cả các đường dẫn có thể có trong mã đều được kiểm tra ít nhất một lần.

Độ phức tạp Cyclomatic - Cách tính

Có hai công thức phổ biến để tính toán độ phức tạp Cyclomatic, dựa trên biểu đồ luồng điều khiển (control flow graph) của mã:

- **Công thức 1: Dựa trên biểu đồ luồng điều khiển** $V(G)=E-N+2P$ Trong đó:
  - V(G): Độ phức tạp Cyclomatic (Complexity Value).
  - E: Số lượng **cạnh** (edges) trong biểu đồ luồng điều khiển (tức là số lượng các liên kết từ một khối mã đến khối mã khác).
  - N: Số lượng **nút** (nodes) trong biểu đồ luồng điều khiển (tức là số lượng các khối lệnh hoặc điểm quyết định).
  - P: Số lượng **thành phần kết nối** (connected components) trong biểu đồ (thường là 1 cho một mô-đun hoặc hàm đơn).
- **Công thức 2: Dựa trên số lượng nút quyết định** V(G)=1+số lượng nút quyết định Trong đó, **nút quyết định** là các điểm mà luồng điều khiển rẽ nhánh (ví dụ: `if`, `while`, `for`, `case`, `AND`, `OR`).
  - Công thức này thường dễ áp dụng hơn trong thực tế.

Thiết kế và cài đặt

- Thiết kế và triển khai phần mềm là giai đoạn trong quy trình kỹ thuật phần mềm mà tại đó một hệ thống phần mềm có thể thực thi được phát triển.
- Các hoạt động thiết kế và triển khai phần mềm luôn xen kẽ nhau.
  - Thiết kế phần mềm là một hoạt động sáng tạo trong đó bạn xác định các thành phần phần mềm và mối quan hệ của chúng, dựa trên yêu cầu của khách hàng.
  - Triển khai là quá trình hiện thực hóa thiết kế thành một chương trình.

Xây dựng hay mua?

- Trong nhiều lĩnh vực, hiện nay có thể mua các hệ thống có sẵn (off-the-shelf systems - COTS) có thể được điều chỉnh và tùy biến theo yêu cầu của người dùng.
  - Ví dụ, nếu bạn muốn triển khai một hệ thống hồ sơ y tế, bạn có thể mua một gói phần mềm đã được sử dụng trong các bệnh viện. Sử dụng phương pháp này có thể rẻ hơn và nhanh hơn so với việc phát triển một hệ thống bằng ngôn ngữ lập trình thông thường.
- Khi bạn phát triển một ứng dụng theo cách này, quy trình thiết kế trở nên liên quan đến việc làm thế nào để sử dụng các tính năng cấu hình của hệ thống đó nhằm đáp ứng **các yêu cầu hệ thống**.

Quy trình thiết kế hướng đối tượng

- Các quy trình thiết kế hướng đối tượng có cấu trúc bao gồm việc phát triển một số mô hình hệ thống khác nhau.
- Chúng đòi hỏi nhiều nỗ lực cho việc phát triển và bảo trì các mô hình này và, đối với các hệ thống nhỏ, điều này có thể không hiệu quả về chi phí.
- Tuy nhiên, đối với các hệ thống lớn được phát triển bởi các nhóm khác nhau, các mô hình thiết kế là một cơ chế giao tiếp quan trọng.

Các giai đoạn quy trình

Có nhiều quy trình thiết kế hướng đối tượng khác nhau phụ thuộc vào tổ chức sử dụng quy trình đó.

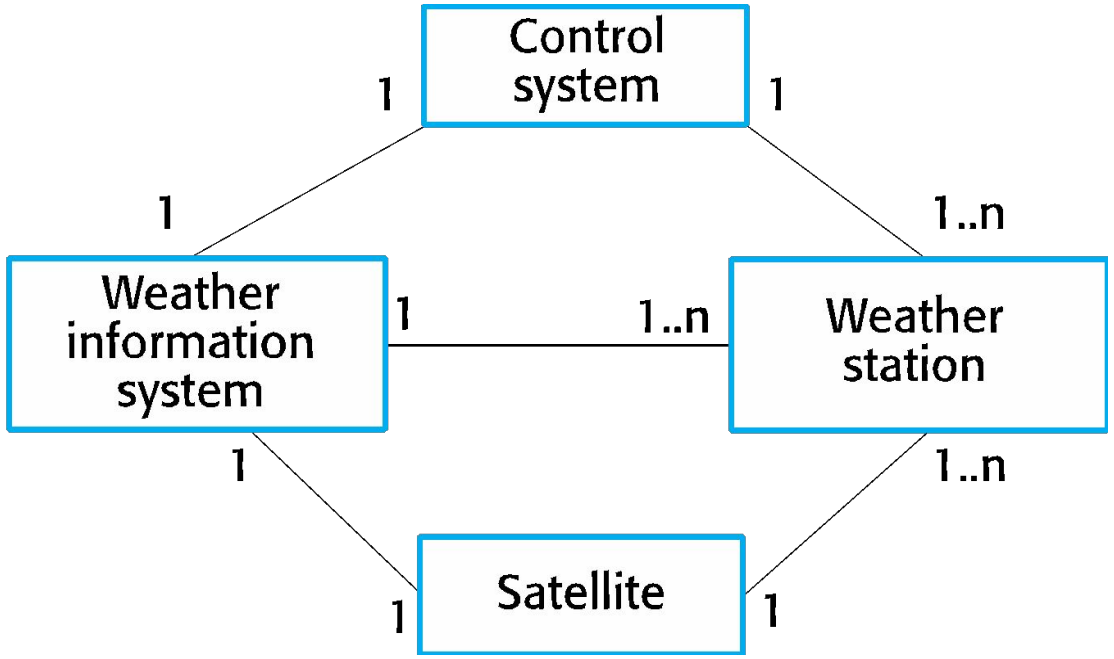
Các hoạt động phổ biến trong các quy trình này bao gồm:

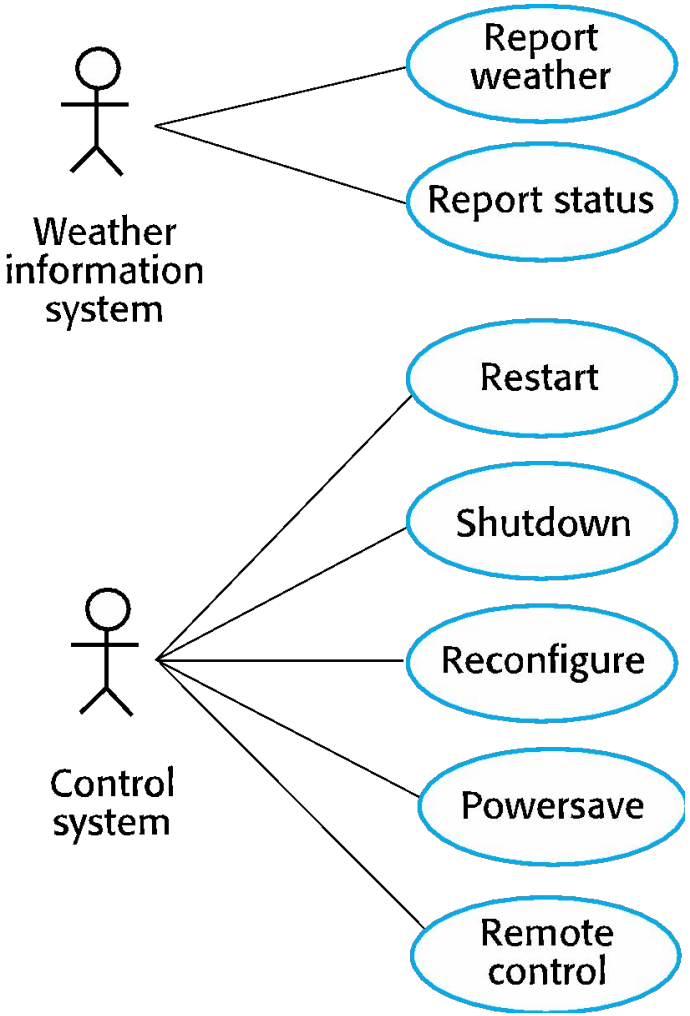
- Định nghĩa ngữ cảnh và các chế độ sử dụng của hệ thống;
- Thiết kế kiến trúc hệ thống;
- Xác định các đối tượng hệ thống chính;
- Phát triển các mô hình thiết kế;
- Đặc tả các giao diện đối tượng.

Ngữ cảnh và tương tác hệ thống

- Hiểu rõ các mối quan hệ giữa phần mềm đang được thiết kế và môi trường bên ngoài của nó là cần thiết để quyết định cách cung cấp chức năng hệ thống được yêu cầu và cách cấu trúc hệ thống để giao tiếp với môi trường của nó.
- Việc hiểu rõ ngữ cảnh cũng cho phép bạn thiết lập các ranh giới của hệ thống. Việc đặt ra ranh giới hệ thống giúp bạn quyết định những tính năng nào được triển khai trong hệ thống đang được thiết kế và những tính năng nào nằm trong các hệ thống liên quan khác.
- Một mô hình ngữ cảnh hệ thống là một mô hình cấu trúc chỉ ra các hệ thống khác trong môi trường của hệ thống đang được phát triển.
- Một mô hình tương tác là một mô hình động chỉ ra cách hệ thống tương tác với môi trường của nó khi nó được sử dụng.

Ngữ cảnh hệ thống và ca sử dụng của Trạm thời tiết



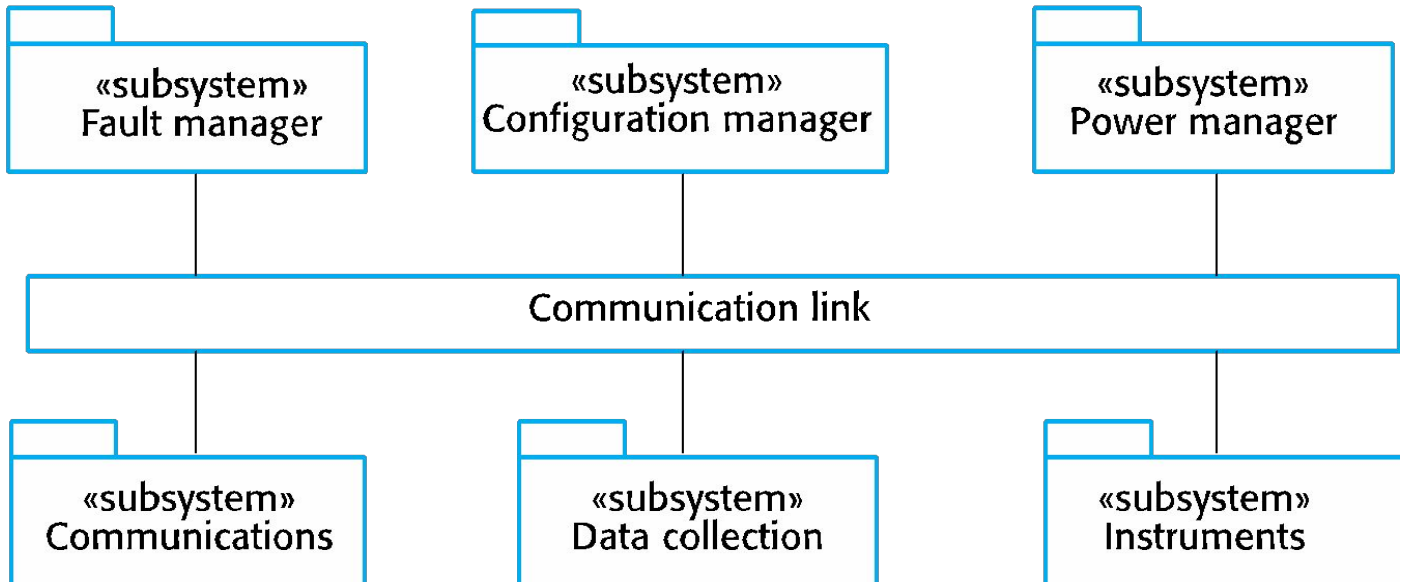


Thiết kế kiến trúc

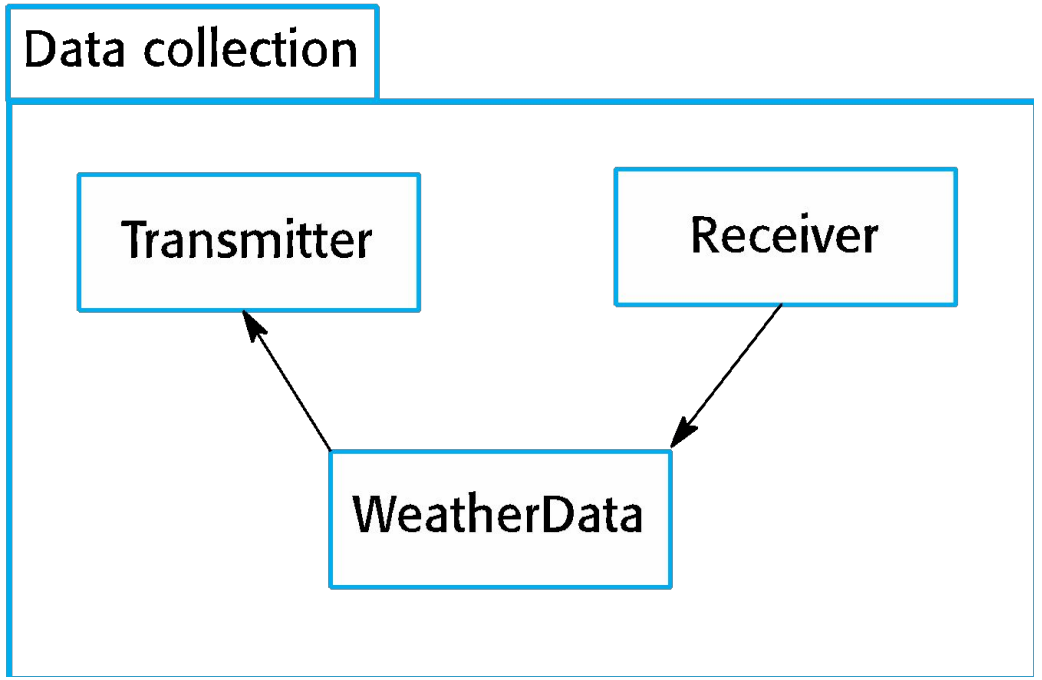
- Một khi các tương tác giữa hệ thống và môi trường của nó đã được hiểu rõ, sử dụng thông tin này để thiết kế kiến trúc hệ thống.
- Xác định các thành phần chính tạo nên hệ thống và các tương tác của chúng, và sau đó có thể tổ chức các thành phần bằng cách sử dụng một mẫu kiến trúc như mô hình phân lớp hoặc mô hình client-server.

Ví dụ Kiến trúc mức cao của trạm thời tiết

- Trạm thời tiết bao gồm các hệ thống con độc lập giao tiếp bằng cách phát thông điệp quảng bá (broadcasting messages) trên một cơ sở hạ tầng chung.



Kiến trúc của hệ thống thu thập dữ liệu



Xác định lớp đối tượng

- Việc xác định các lớp đối tượng thường là một phần khó khăn của thiết kế hướng đối tượng.
- Không có 'công thức thần kỳ' nào cho việc xác định đối tượng. Nó phụ thuộc vào kỹ năng, kinh nghiệm và kiến thức miền của các nhà thiết kế hệ thống.
- Việc xác định đối tượng là một quy trình lặp đi lặp lại. Bạn khó có thể làm đúng ngay lần đầu tiên.

Các phương pháp xác định

- Sử dụng phương pháp ngữ pháp dựa trên mô tả hệ thống bằng ngôn ngữ tự nhiên.
- Dựa vào việc xác định các đối tượng hữu hình trong lĩnh vực ứng dụng.
- Sử dụng phương pháp hành vi và xác định các đối tượng dựa trên những gì tham gia vào hành vi nào.
- Sử dụng phân tích dựa trên kịch bản. Các đối tượng, thuộc tính và phương thức trong mỗi kịch bản được xác định.

Các mô hình thiết kế

- Các mô hình thiết kế chỉ ra các đối tượng và lớp đối tượng cùng các mối quan hệ giữa các thực thể này.
- Có hai loại mô hình thiết kế:
  a. Các mô hình cấu trúc mô tả cấu trúc tĩnh của hệ thống xét về các lớp đối tượng và mối quan hệ.
  b. Các mô hình động mô tả các tương tác động giữa các đối tượng.

Các ví dụ về mô hình thiết kế

- Các mô hình hệ thống con (Subsystem models) chỉ ra các nhóm logic của các đối tượng thành các hệ thống con mạch lạc.
- Các mô hình tuần tự (Sequence models) chỉ ra chuỗi tương tác đối tượng.
- Các mô hình máy trạng thái (State machine models) chỉ ra cách các đối tượng riêng lẻ thay đổi trạng thái của chúng khi phản ứng với các sự kiện.
- Các mô hình khác bao gồm mô hình ca sử dụng (use-case models), mô hình kết tập (aggregation models), mô hình tổng quát hóa (generalisation models), v.v.

Các mô hình hệ thống con

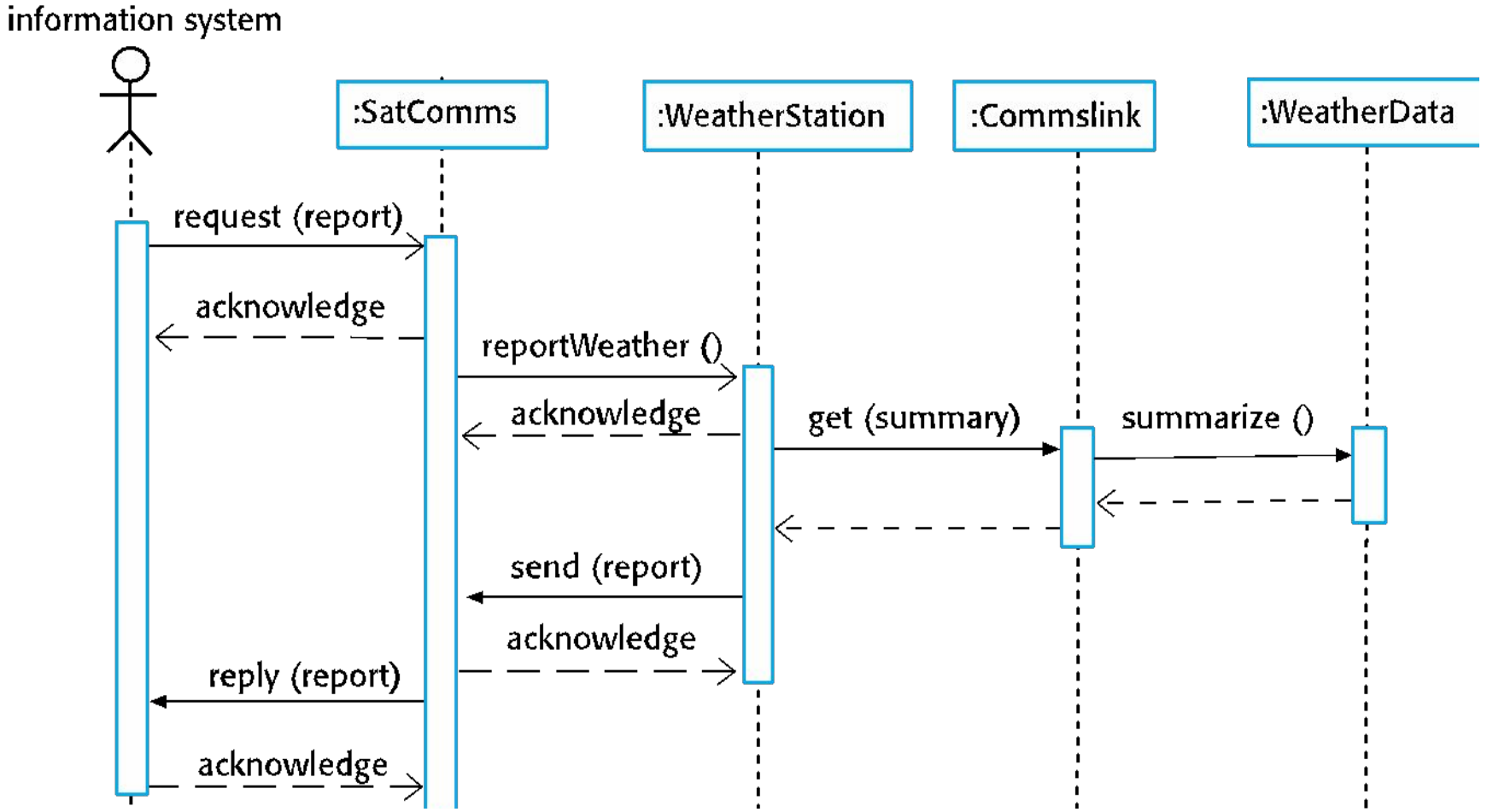
- Chỉ ra cách thiết kế được tổ chức thành các nhóm đối tượng liên quan logic.
- Trong UML, chúng được thể hiện bằng cách sử dụng các gói (packages) - một cấu trúc đóng gói (encapsulation construct). Đây là một mô hình logic. Tổ chức thực tế của các đối tượng trong hệ thống có thể khác.

Các mô hình tuần tự

Các mô hình tuần tự chỉ ra chuỗi các tương tác đối tượng diễn ra

- Các đối tượng được sắp xếp theo chiều ngang ở phía trên cùng;
- Thời gian được biểu diễn theo chiều dọc nên các mô hình được đọc từ trên xuống dưới;
- Các tương tác được biểu diễn bằng các mũi tên có nhãn, Các kiểu mũi tên khác nhau đại diện cho các loại tương tác khác nhau;
- Một hình chữ nhật mỏng trong đường đời đối tượng (object lifeline) đại diện cho thời gian khi đối tượng đó là đối tượng kiểm soát trong hệ thống.

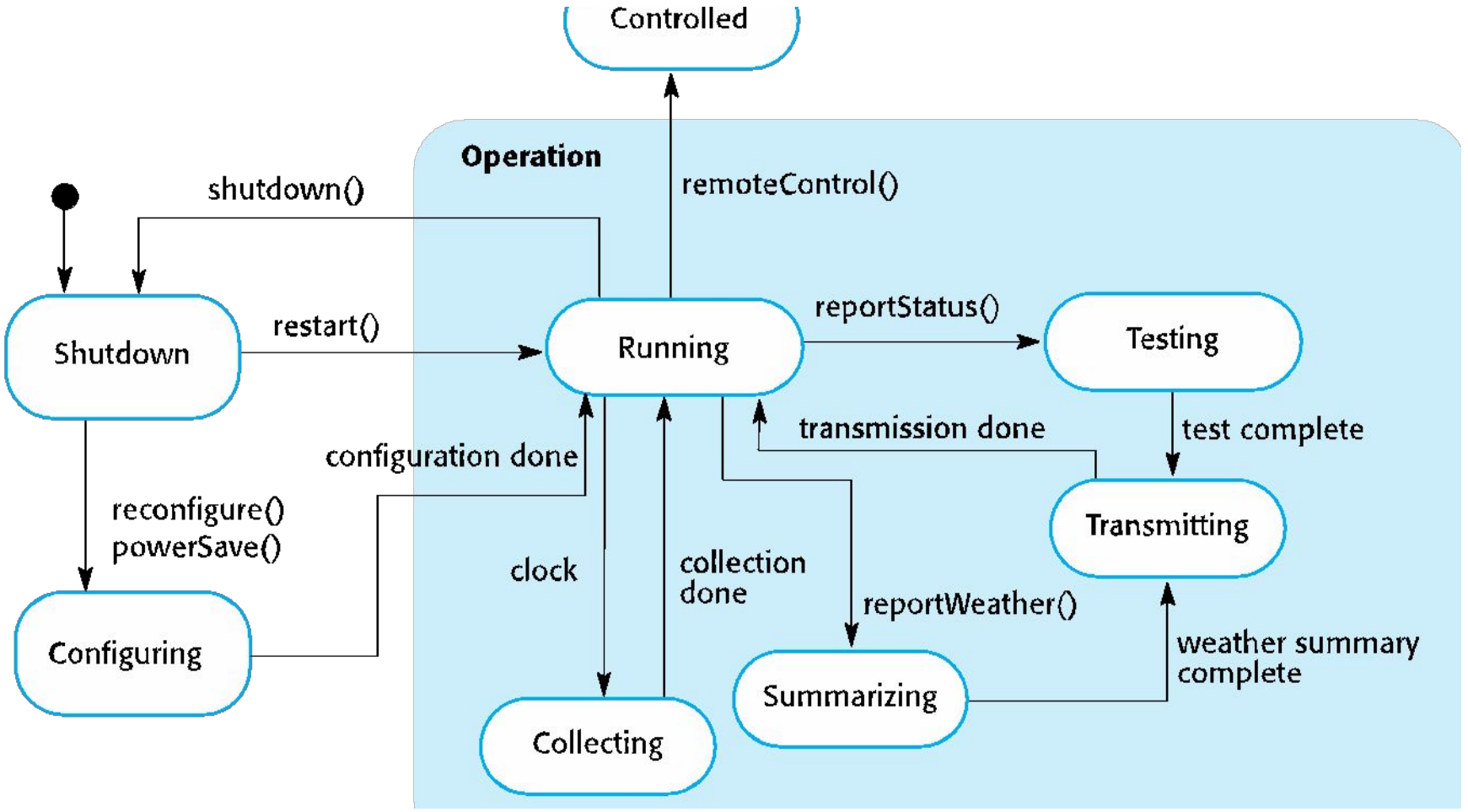
Ví dụ Sơ đồ tuần tự mô tả việc thu thập dữ liệu



Sơ đồ trạng thái (State diagrams)

- Sơ đồ trạng thái được sử dụng để chỉ ra cách các đối tượng phản ứng với các yêu cầu dịch vụ khác nhau và các chuyển đổi trạng thái được kích hoạt bởi các yêu cầu này.
- Sơ đồ trạng thái là các mô hình mức cao hữu ích về hành vi thời gian chạy (run-time behavior) của một hệ thống hoặc một đối tượng.
- Thường không cần sơ đồ trạng thái cho tất cả các đối tượng trong hệ thống. Nhiều đối tượng trong hệ thống tương đối đơn giản và một mô hình trạng thái sẽ thêm chi tiết không cần thiết vào thiết kế.

Ví dụ sơ đồ trạng thái cho Trạm thời tiết



Đặc tả giao diện

- Các giao diện đối tượng phải được đặc tả để các đối tượng và các thành phần khác có thể được thiết kế song song.
- Các nhà thiết kế nên tránh thiết kế biểu diễn giao diện mà nên ẩn điều này trong chính đối tượng đó.
- Các đối tượng có thể có nhiều giao diện, đó là các góc nhìn về các phương thức được cung cấp.

Các mẫu thiết kế

- Một mẫu thiết kế là một cách tái sử dụng kiến thức trừu tượng về một vấn đề và giải pháp của nó.
  - Các Mẫu (Patterns) và Ngôn ngữ Mẫu (Pattern Languages) là những cách để mô tả các thực hành tốt nhất (best practices), các thiết kế tốt, và thu thập kinh nghiệm theo cách mà người khác có thể tái sử dụng kinh nghiệm này.
- Một mẫu là một mô tả về vấn đề và bản chất của giải pháp cho vấn đề đó.
- Nó phải đủ trừu tượng để có thể tái sử dụng trong các bối cảnh khác nhau.
- Các mô tả mẫu thường sử dụng các đặc tính hướng đối tượng như kế thừa và đa hình.

Các Thành phần của mẫu

- Tên
  - Một định danh mẫu có ý nghĩa.
- Mô tả vấn đề.
- Mô tả giải pháp.

  Không phải là một thiết kế cụ thể mà là một khuôn mẫu cho một giải pháp thiết kế có thể được hiện thực hóa (instantiated) theo những cách khác nhau.
- Các hệ quả
  - Các kết quả và sự đánh đổi (trade-offs) khi áp dụng mẫu.

Mẫu Người quan sát (Observer)

- Tên
  - Người quan sát (Observer).
- Mô tả
  - Tách biệt việc hiển thị trạng thái đối tượng khỏi chính đối tượng đó.
- Mô tả vấn đề
  - Được sử dụng khi cần nhiều cách hiển thị trạng thái.
- Mô tả giải pháp

- Các hệ quả
  - Các tối ưu hóa để nâng cao hiệu năng hiển thị là không thực tế/khó thực hiện

Tên mẫu Người quan sát (Observer)

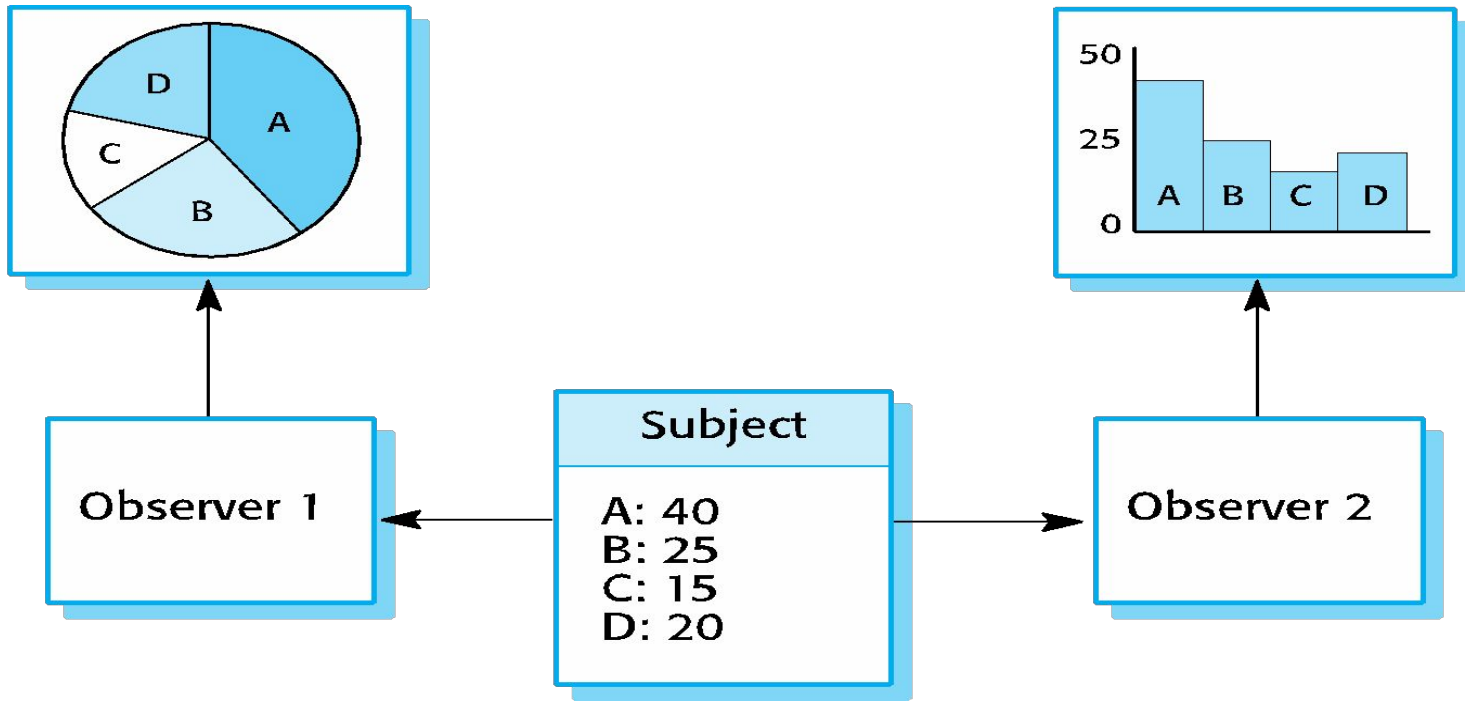
Mô tả Tách biệt việc hiển thị trạng thái của một đối tượng khỏi chính đối tượng đó và cho phép cung cấp các cách hiển thị thay thế. Khi trạng thái đối tượng thay đổi, tất cả các màn hình hiển thị sẽ tự động được thông báo và cập nhật để phản ánh sự thay đổi.

Mô tả vấn đề Trong nhiều tình huống, bạn phải cung cấp nhiều cách hiển thị thông tin trạng thái, chẳng hạn như hiển thị đồ họa và hiển thị dạng bảng. Không phải tất cả những cách này đều có thể được biết khi thông tin được đặc tả. Tất cả các cách trình bày thay thế nên hỗ trợ tương tác và, khi trạng thái thay đổi, tất cả các màn hình hiển thị phải được cập nhật. Mẫu này có thể được sử dụng trong mọi tình huống mà cần nhiều hơn một định dạng hiển thị cho thông tin trạng thái và khi không cần thiết đối tượng duy trì thông tin trạng thái phải biết về các định dạng hiển thị cụ thể được sử dụng.

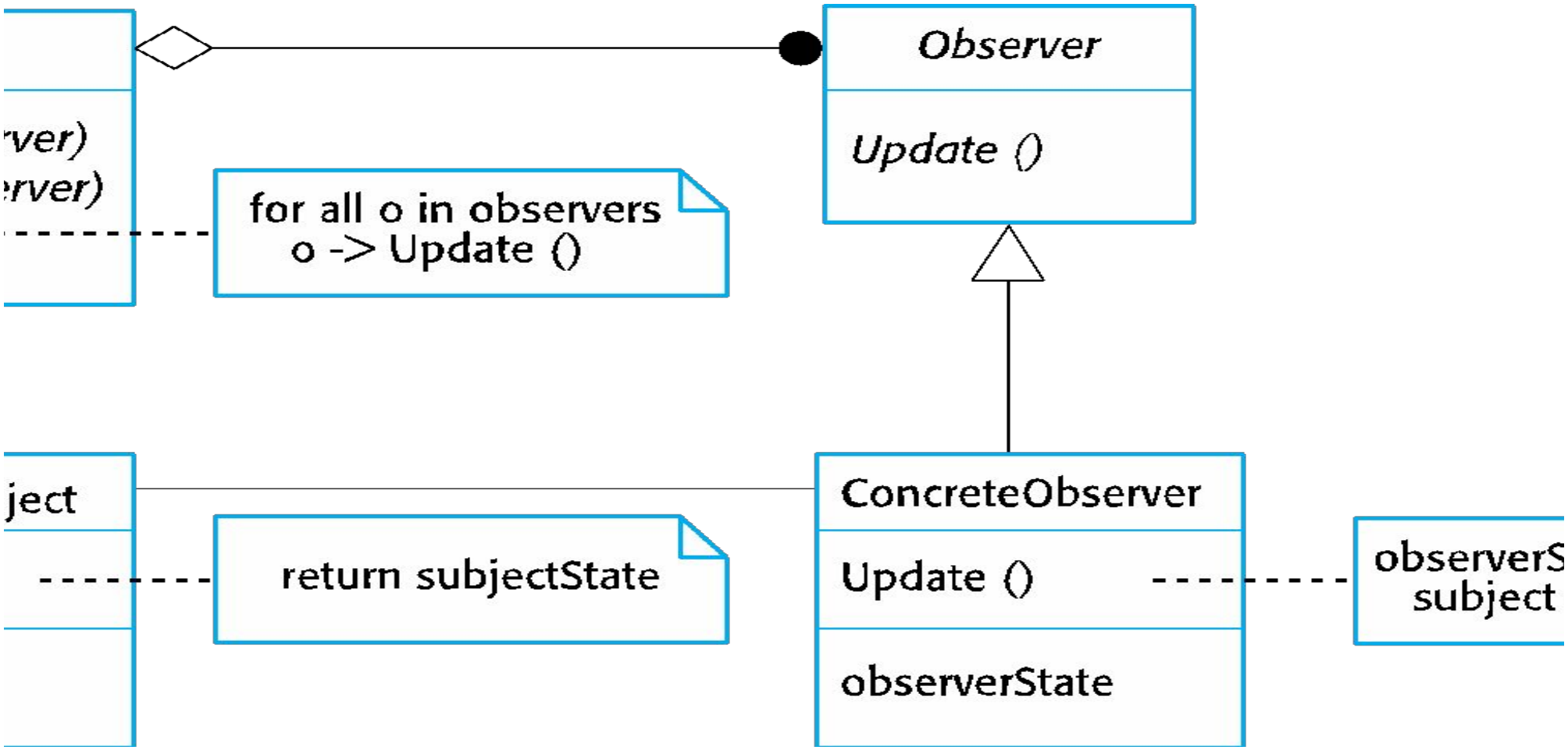
Mô tả giải pháp Giải pháp này bao gồm hai đối tượng trừu tượng, Chủ thể (Subject) và Người quan sát (Observer), và hai đối tượng cụ thể, Chủ thể Cụ thể (ConcreteSubject) và Người quan sát Cụ thể (ConcreteObserver), kế thừa các thuộc tính của các đối tượng trừu tượng liên quan. Các đối tượng trừu tượng bao gồm các hoạt động chung áp dụng được trong mọi tình huống. Trạng thái cần hiển thị được duy trì trong ConcreteSubject, lớp này kế thừa các hoạt động từ Subject cho phép nó thêm và xóa các Người quan sát (mỗi người quan sát tương ứng với một màn hình hiển thị) và đưa ra thông báo khi trạng thái đã thay đổi.

Các hệ quả Chủ thể chỉ biết đến Observer trừu tượng và không biết chi tiết về lớp cụ thể. Do đó, có sự ghép nối tối thiểu (minimal coupling) giữa các đối tượng này. Vì sự thiếu hiểu biết này, các tối ưu hóa nhằm nâng cao hiệu năng hiển thị là không thực tế. Các thay đổi đối với chủ thể có thể gây ra một loạt các cập nhật liên kết đến các người quan sát, một số trong đó có thể không cần thiết.

Multiple displays using the Observer pattern



A UML model of the Observer pattern



Các vấn đề thiết kế

Để sử dụng các mẫu trong thiết kế của bạn, bạn cần nhận ra rằng bất kỳ vấn đề thiết kế nào bạn đang đối mặt đều có thể có một mẫu liên quan có thể được áp dụng.

- Thông báo cho nhiều đối tượng rằng trạng thái của một đối tượng khác đã thay đổi (mẫu Observer).
- Dọn dẹp các giao diện cho một số đối tượng liên quan thường được phát triển dần dần (mẫu Façade).
- Cung cấp một cách chuẩn để truy cập các phần tử trong một tập hợp, bất kể tập hợp đó được triển khai như thế nào (mẫu Iterator).
- Cho phép khả năng mở rộng chức năng của một lớp hiện có tại thời gian chạy (mẫu Decorator).

Các vấn đề triển khai

Trọng tâm ở đây không phải là lập trình, mặc dù điều này rõ ràng là quan trọng, mà là về các vấn đề triển khai khác thường không được đề cập trong các sách về lập trình:

- Tái sử dụng Hầu hết phần mềm hiện đại được xây dựng bằng cách tái sử dụng các thành phần hoặc hệ thống hiện có. Khi bạn đang phát triển phần mềm, bạn nên tận dụng tối đa mã nguồn hiện có.
- Quản lý cấu hình Trong quá trình phát triển, bạn phải theo dõi nhiều phiên bản khác nhau của mỗi thành phần phần mềm trong một hệ thống quản lý cấu hình.
- Phát triển kiểu host-target Phần mềm sản xuất thường không thực thi trên cùng một máy tính với môi trường phát triển phần mềm. Thay vào đó, bạn phát triển nó trên một máy tính (hệ thống chủ - host system) và thực thi nó trên một máy tính riêng biệt (hệ thống đích - target system).

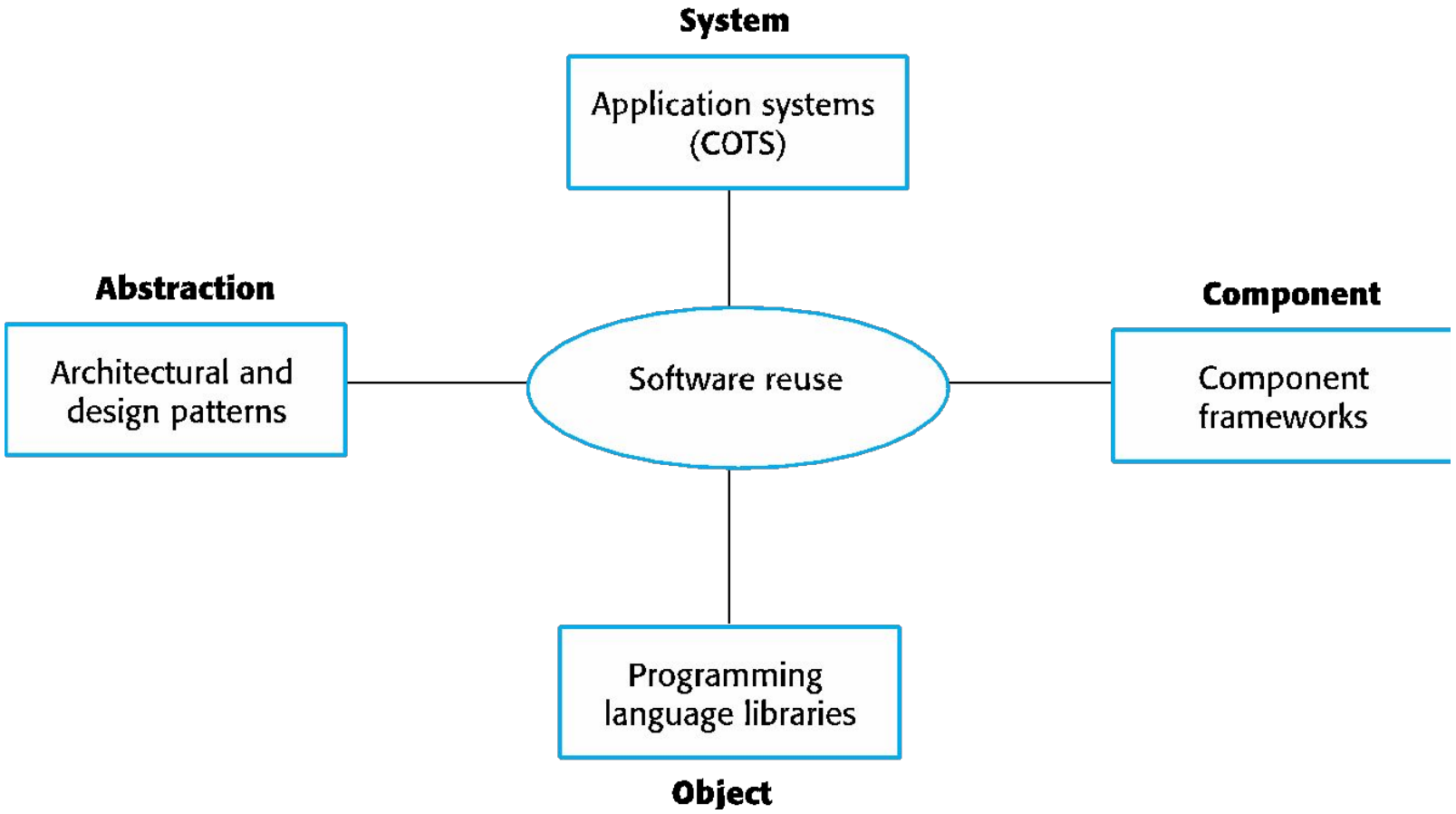
Tái sử dụng

- Từ những năm 1960 đến những năm 1990, hầu hết phần mềm mới được phát triển từ đầu, bằng cách viết tất cả mã nguồn bằng một ngôn ngữ lập trình bậc cao.
  - Sự tái sử dụng phần mềm đáng kể duy nhất là việc tái sử dụng các hàm và đối tượng trong các thư viện ngôn ngữ lập trình.
- Chi phí và áp lực về tiến độ có nghĩa là phương pháp này ngày càng trở nên không khả thi, đặc biệt đối với các hệ thống thương mại và dựa trên Internet.
- Một phương pháp phát triển dựa trên việc tái sử dụng phần mềm hiện có đã xuất hiện và hiện nay thường được sử dụng cho phần mềm kinh doanh và khoa học.

Các cấp độ tái sử dụng

- Mức độ trừu tượng
  - Ở mức độ này, bạn không tái sử dụng trực tiếp phần mềm mà sử dụng kiến thức về các trừu tượng hóa thành công trong thiết kế phần mềm của bạn.
- Mức độ đối tượng
  - Ở mức độ này, bạn tái sử dụng trực tiếp các đối tượng từ một thư viện thay vì tự viết mã nguồn.
- Mức độ thành phần
  - Các thành phần là tập hợp các đối tượng và lớp đối tượng mà bạn tái sử dụng trong các hệ thống ứng dụng.
- Mức độ hệ thống
  - Ở mức độ này, bạn tái sử dụng toàn bộ các hệ thống ứng dụng.

Tái sử dụng phần mềm



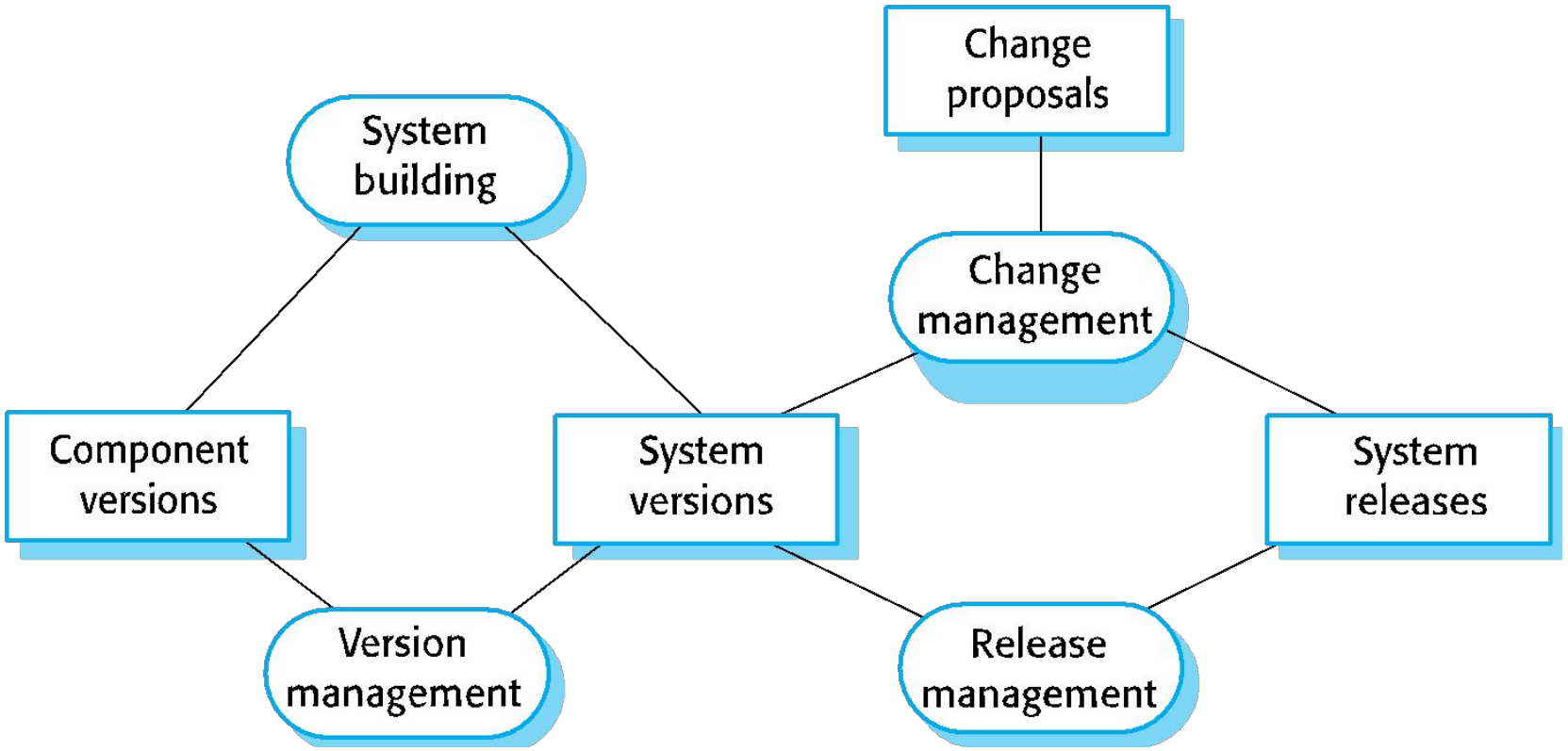
Chi phí tái sử dụng

- Chi phí thời gian bỏ ra để tìm kiếm phần mềm tái sử dụng và đánh giá xem nó có đáp ứng nhu cầu của bạn hay không.
- Nếu có, chi phí mua phần mềm tái sử dụng. Đối với các hệ thống có sẵn (off-the-shelf) lớn, các chi phí này có thể rất cao.
- Chi phí điều chỉnh và cấu hình các thành phần hoặc hệ thống phần mềm tái sử dụng để phản ánh các yêu cầu của hệ thống mà bạn đang phát triển.
- Chi phí tích hợp các yếu tố phần mềm tái sử dụng với nhau (nếu bạn đang sử dụng phần mềm từ các nguồn khác nhau) và với mã nguồn mới mà bạn đã phát triển.

Quản lý cấu hình

- Quản lý cấu hình là tên gọi chung cho quy trình quản lý một hệ thống phần mềm đang thay đổi.
- Mục đích của quản lý cấu hình là hỗ trợ quy trình tích hợp hệ thống để tất cả các nhà phát triển có thể truy cập mã nguồn và tài liệu dự án một cách có kiểm soát, tìm hiểu những thay đổi đã được thực hiện, và biên dịch và liên kết các thành phần để tạo ra một hệ thống.

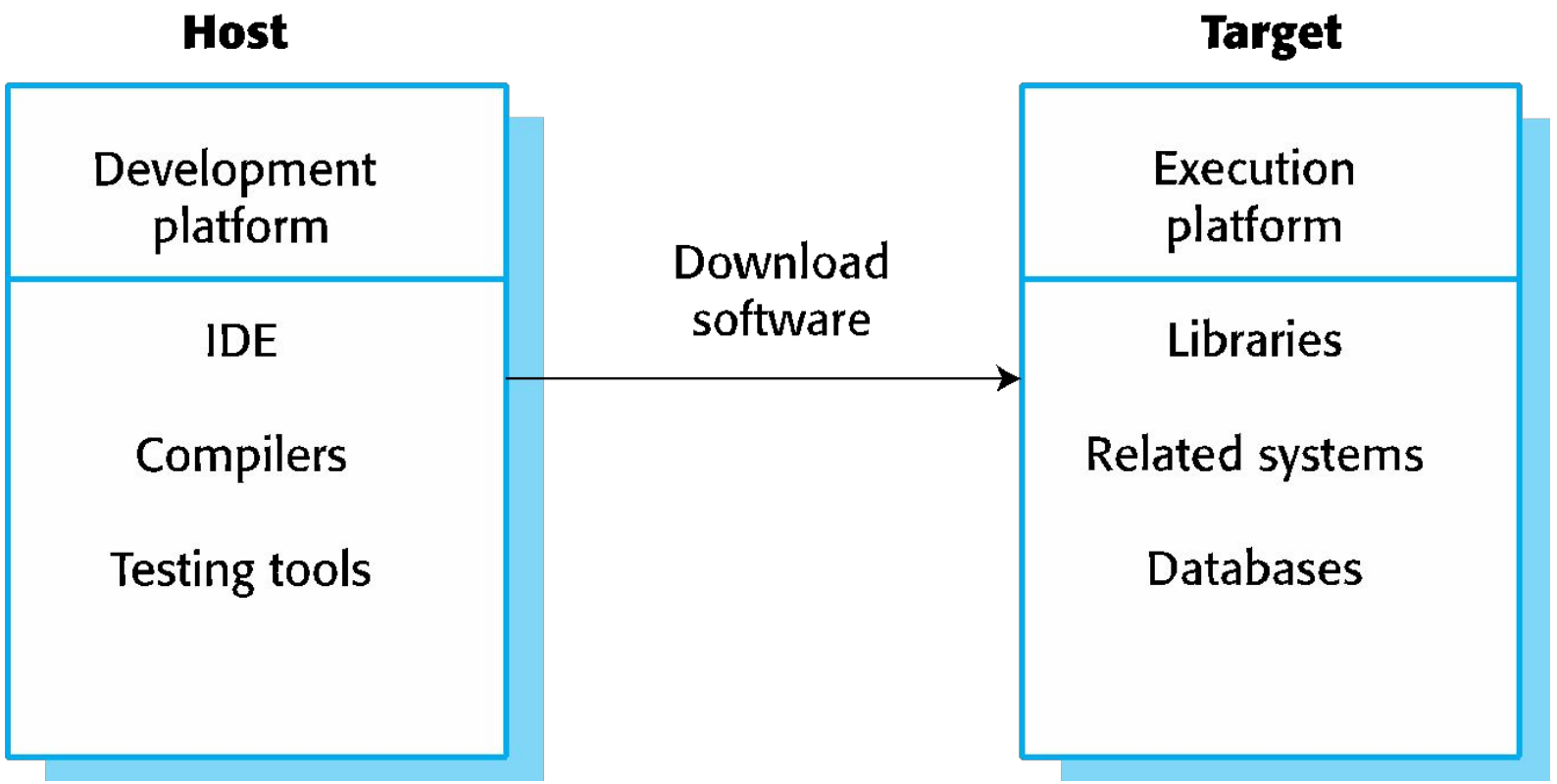
Configuration management tool interaction



Phát triển kiểu host-target

Hầu hết phần mềm được phát triển trên một máy tính (máy chủ - the host), nhưng chạy trên một máy riêng biệt (máy đích - the target). Tổng quát hơn, chúng ta có thể nói về một nền tảng phát triển và một nền tảng thực thi. Một nền tảng không chỉ là phần cứng. Nó bao gồm hệ điều hành đã cài đặt cộng với các phần mềm hỗ trợ khác như hệ quản trị cơ sở dữ liệu hoặc, đối với nền tảng phát triển, một môi trường phát triển tương tác. Nền tảng phát triển thường có phần mềm được cài đặt khác với nền tảng thực thi; các nền tảng này có thể có kiến trúc khác nhau.

Host-target development



Phát triển mã nguồn mở

- Phát triển mã nguồn mở là một phương pháp phát triển phần mềm trong đó mã nguồn của một hệ thống phần mềm được công bố và các tình nguyện viên được mời tham gia vào quy trình phát triển
- Nguồn gốc của nó nằm ở Tổ chức Phần mềm Tự do (Free Software Foundation - www.fsf.org), tổ chức ủng hộ rằng mã nguồn không nên là độc quyền mà thay vào đó phải luôn có sẵn để người dùng xem xét và sửa đổi theo ý muốn.
- Phần mềm mã nguồn mở đã mở rộng ý tưởng này bằng cách sử dụng Internet để tuyển dụng một số lượng lớn hơn nhiều các nhà phát triển tình nguyện. Nhiều người trong số họ cũng là người sử dụng mã nguồn.
  Sản phẩm mã nguồn mở nổi tiếng nhất, tất nhiên, là hệ điều hành Linux, được sử dụng rộng rãi như một hệ thống máy chủ và, ngày càng tăng, như một môi trường máy tính để bàn.
- Các sản phẩm mã nguồn mở quan trọng khác là Java, máy chủ web Apache và hệ quản trị cơ sở dữ liệu mySQL

Các điểm chính

- Thiết kế và triển khai phần mềm là các hoạt động xen kẽ nhau. Mức độ chi tiết trong thiết kế phụ thuộc vào loại hệ thống và việc bạn đang sử dụng phương pháp hướng kế hoạch hay linh hoạt.
- Quy trình thiết kế hướng đối tượng bao gồm các hoạt động để thiết kế kiến trúc hệ thống, xác định các đối tượng trong hệ thống, mô tả thiết kế bằng các mô hình đối tượng khác nhau và tài liệu hóa các giao diện thành phần.
- Một loạt các mô hình khác nhau có thể được tạo ra trong quá trình thiết kế hướng đối tượng. Chúng bao gồm các mô hình tĩnh (mô hình lớp, mô hình tổng quát hóa, mô hình liên kết) và các mô hình động (mô hình tuần tự, mô hình máy trạng thái).

Các điểm chính

- Các giao diện thành phần phải được định nghĩa chính xác để các đối tượng khác có thể sử dụng chúng. Một stereotype giao diện UML có thể được sử dụng để định nghĩa các giao diện.
- Khi phát triển phần mềm, bạn nên luôn xem xét khả năng tái sử dụng phần mềm hiện có, dưới dạng các thành phần, dịch vụ hoặc hệ thống hoàn chỉnh.
- Quản lý cấu hình là quá trình quản lý các thay đổi đối với một hệ thống phần mềm đang tiến hóa. Nó rất cần thiết khi một nhóm người đang hợp tác để phát triển phần mềm.
- Hầu hết việc phát triển phần mềm là phát triển kiểu host-target (máy chủ-máy đích). Bạn sử dụng một IDE trên máy chủ để phát triển phần mềm, sau đó phần mềm được chuyển sang máy đích để thực thi.
- Phát triển mã nguồn mở bao gồm việc công khai mã nguồn của một hệ thống. Điều này có nghĩa là nhiều người có thể đề xuất các thay đổi và cải tiến cho phần mềm