Chương 14: THIẾT KẾ

***Mở đầu:***

Trong hơn 40 năm qua, hàng trăm kỹ thuật thiết kế đã được đưa ra. Một số được biến thể từ những kỹ thuật hiện có, một số là những kỹ thuất mới hoàn toàn. Có những kỹ thuật thiết kế đã được sử dụng bởi hàng chục nghìn kỹ sư phần mềm; Có những ký thuật lại chỉ được sử dụng bởi các tác giả của chúng. Có những chiến lược có cơ sở lý thuyết vững chắc, một số lại thực dụng hơn vì tác giả cho thấy rằng chúng hoạt động tốt hơn trong thực tế

Tuy có nhiều kỹ thuật thiết kế nhưng mô hình thiết kế nhất định vẫn xuất hiện.

Một sản phẩn phầm mềm có 2 khía cạnh: Những hành động thực hiện trên dữ liệu và Dữ liệu mà các hành động thao tác trên dữ liệu đó. Do đó, hai cách cơ bản để thiết kế sản phẩm :

* *Thiết kế hướng hoạt động: trọng tâm là các hoạt động*
* *Thiết kế hướng dữ liệu: dữ liệu được xem xét đầu tiên*

Điểm yếu của các kỹ thuật thiết kế hướng hoạt động là chúng tập trung vào các hoạt động; dữ liệu chỉ có tầm quan trọng thứ yếu. Tương tự như vậy, các kỹ thuật thiết kế hướng dữ liệu nhấn mạnh vào dữ liệu, gây bất lợi cho các hoạt động.

=>Giải pháp là sử dụng các kỹ thuật thiết kế hướng đối tượng, lai giữa 2 kỹ thuật thiết kế trên. Một đối tượng kết hợp cả hoạt động và dữ liệu, thiết kế hướng đối tượng kết hợp các tính năng của thiết kế hướng hoạt động và hướng dữ liệu.

***14.1. Thiết kế và trừu tượng***

Pha thiết kế cổ điển bao gồm 3 hoạt động: thiết kế kiến trúc, thiết kế chi tiết và kiểm thử thiết kế. Đầu vào của quá trình thiết kế là tài liệu chi tiết, mô tả chi tiết sản phẩm cần làm những gì. Đầu ra của quá trình là tài liệu thiết kế, mô tả làm thế nào để sản có thể đạt được những điều đó.

Trong **thiết kế kiến trúc** (hay còn được gọi là **thiết kế chung**, **thiết kế logic**, **thiết kế mức cao**), sự phân rã theo mô-đun của sản phẩm được phát triển. Nghĩa là, các thông số kỹ thuật sẽ được phân tích cẩn thận, và một cấu trúc mô-đun có những chức năng mong muốn sẽ được tạo ra. Đầu vào là những đặc tả. Đầu ra của hoạt động này là một danh sách các mô-đun và sự mô tả các mô-đun kết nối với nhau như thế nào. Từ quan điểm trừu tượng , trong thiết kế kiến trúc, sự tồn tại của một số mô-đun nhất định được giả định, quá trình thiết kế sau đó sẽ phát triển dựa trên những mô-đun này.

Tuy nhiên, khi mô hình hướng đối tượng được sử dụng, hoạt động thiết kế kiến trúc được thực hiện trong pha phân tích hướng đối tượng (chương 12). Điều này là do bước đầu tiên của luồng công việc phân tích là xác định các lớp. Bởi vì một lớp là một loại mô-đun, sự phân rã mô-đun được hình thành trong luồng công việc phân tích.

Hoạt động tiếp theo trong pha thiết kế và cũng là hoạt động chính của luồng công việc thiết kế hướng đối tượng là **thiết kế chi tiết** (hay còn được gọi là **thiết kế mô-đun**, **thiết kế vật lý** hoặc **thiết kế mức thấp**) trong đó mỗi mô-đun được thiết kế chi tiết. Chẳng hạn, như việc lựa chọn thuật toán cụ thể và các cấu trúc dữ liệu. Từ quan điểm trừu tượng, trong hoạt động này, sự kết nối của các mô-đun (hoặc các lớp) để tạo ra một sản phẩm hoàn chỉnh bị bỏ qua.

Như đã liệt kê từ đầu, pha thiết kế cổ điển có 3 hoạt động, hoạt động thứ ba là kiểm thử. Sử dụng từ ‘hoạt động’ thay vì ‘giai đoạn’ hoặc ‘bước’, nhằm nhấn mạnh kiểm thử là một phần không thể thiếu của thiết kế, cũng như là một phần không thể thiếu trong toàn bộ quá trình phát triển phần mềm và quá trình bảo trì. Kiểm thử không chỉ diễn ra khi thiết kế kiến trúc và thiết kế chi tiết đã hoàn thành. Trong thiết kế hướng đối tượng, kiểm thử diễn ra đồng thời trong luồng công việc thiết kế.

Hiện nay, có nhiều kỹ thuật thiết kế được mô tả, đầu tiên là kỹ thuật hướng thao tác, tiếp theo là kỹ thuật hướng dữ liệu, cuối cùng là kỹ thuật hướng đối tượng

***14.2. Thiết kế hướng thao tác***

Một trường hợp lý thuyết để phân rã một sản phẩm thành các mô-đun có độ gắn kết (Cohesion: sự kết nối của các thành phần bên trong một mô-đun) cao và khớp nối (Coupling: sự phụ thuộc lẫn nhau của các mô-đun) thấp. Bây giờ chúng ta sẽ mô tả 2 kỹ thuật cổ điển thực tế để đạt được mục tiêu của loại thiết kế này, phân tích luồng dữ liệu (14.3) và phân tích giao dịch (14.4).

Về lý thuyết, phân tích luồng dữ liệu có thể được áp dụng bất cứ khi nào các thông số kỹ thuật có thể được biểu diễn bằng biểu đồ luồng dữ liệu (DFD), và bởi vì (ít nhất là về mặt lý thuyết) mọi sản phẩm đều có thể được bởi biểu đồ luồng dữ liệu, phân tích luồng dữ liệu có thể được sử dụng phổ biến. Trên thực tế,trong một số trường hợp, có những kỹ thuật thiết kế phù hợp hơn, đặc biệt là trong những trường hợp thiết kế những sản phẩm mà luồng dữ liệu không được coi trọng bằng những yếu tố khác. Ví dụ, các kỹ thuật thiết kế sản phẩm như hệ thống dựa trên quy tắc (hệ thống chuyên gia), cơ sở dữ liệu, xử lý giao dịch. Phân tích giao dịch sẽ được giới thiệu ở phần 14.4

***14.3 Phân tích luồng dữ liệu***

Phân tích luồng dữ liệu (Data flow analysis DFA) là kỹ thuật thiết kế cổ điển nhằm mục đích giúp các mô-đun có độ gắn kết cao. Kỹ thuật này có thể được sử dụng kết hợp với hầu hết các kỹ thuật phân tích.

Đầu vào của kỹ thuật này là một sơ đồ luồng dữ liệu (data flow diagram DFD). Điểm mấu chốt là, khi DFD được hoàn thành, người thiết kế phần mềm sẽ có được thông tin đầy đủ và chính xác về đầu vào và đầu ra của sản phẩm.

Luồng dữ liệu của sản phẩm được biểu diễn trong hình 14.1. Sản phẩm bằng cách nào đó biến đầu vào thành đầu ra. Tại một số điểm trong DFD, đầu vào được biến đổi thành dữ liệu nội bộ (internal data). Sau đó, tại một số điểm khác, những dữ liệu nội bộ này đảm bảo cho chất lượng của đầu ra. Điều này được thể hiện rõ hơn tại hình 14.2. Tại điểm mà đầu vào bị biến đổi thành dữ liệu nội bộ do sản phẩm điều hành được gọi là **điểm trừu tượng cao nhất của đầu vào**. **Điểm trừu tượng cao nhất của đầu ra,** tương tự, là điểm đầu tiên của luồng dữ liệu mà có thể xác định được đầu ra, mà không còn là dữ liệu nội bộ.

Sử dụng điểm trừu tượng cao nhất của đầu vào và đầu ra, sản phẩm được phân rã thành 3 mô-đun: mô-đun đầu vào, mô-đun biến đổi và mô-đun đầu ra. Thực hiện lần lượt từng mô-đun, tìm ra điểm trừu tượng cao nhất của mô-đun, sau đó tiếp tục phân rã mô-đun. Quá trình này được thực hiện lặp đi lặp lại cho đến khi mỗi mô-đun chỉ thực hiện một thao tác đơn lẻ; điều đó, có nghĩa là sự thiết kế bao gồm nhiều mô-đun có tính liên kết cao.

|  |
| --- |
|  |
|  |

Vì vậy, tinh chỉnh từng bước, là nền tảng cho rất nhiều kỹ thuật phát triển phần mềm, đồng thời cũng là nền tảng cho phân tích luồng dữ liệu.

Để nói công bằng, cần chỉ ra những sửa đổi nhỏ cần phải được thực hiện để sự phân rã có thể đạt được khớp nối thấp nhất. Phân tích luồng dữ liệu là một kỹ thuật để có thể đạt được sự kết nối cao. Mục đích của thiết kế tổng quan/ cấu trúc là đạt được sự kết nối cao nhưng khớp nối thấp. Để có thể đạt được mục đích này, đôi khi cần phải thực hiện những sửa đổi nhỏ trong thiết kế. Ví dụ, DFA không tính đến khớp nối, có thể khiến phát sinh control coupling. Trong trường hợp này, chỉ cần sửa đổi hai mô-đun liên quan để dữ liệu (chứ không phải control) được chuyển qua lại giữa hai mô-đun.

**Mini case study**

***14.3.1. Mini case study word counting***

Thiết kế một sản phẩm, đầu vào là một file và đầu ra là số lượng từ trong file đó, tương tự như UNIX wc utility.

Hình 14.3 mô tả sơ đồ luồng dữ liệu. Có 5 mô-đun. Mô-đun read\_file\_name: đọc tên của file, sau đó được xác thực bởi mô-đun validate\_file\_name. Tên đã được xác thực sẽ chuyển qua count\_number\_of\_words, mô-đun thực hiện thao tác đúng như tên gọi. Số lượng từ sau đó được chuyển qua format\_word\_count, và được định dạng, số lượng từ được định dạng cuối cùng chuyển qua display\_word\_count cho đầu ra.

Kiểm tra luồng dữ liệu, đầu vào ban đầu là file-name. Sau khi chuyển qua mô-đun validate\_file\_name, dữ liệu vẫn là file name nên vẫn là dữ liệu đầu vào. Cần phải cân nhắc mô-đun count\_number\_of\_words. Đầu vào của mô-đun này là validate\_file\_name, và đầu ra là word\_count. Đầu ra của mô-đun này khác biệt hoàn toàn so với đầu vào của cả sản phẩm. Vì vậy đây là điểm trừu tượng cao nhất của đầu vào, như được chỉ ra ở hình 14.3. Tương tự, mặc dù đầu ra của count\_number\_of\_words trải qua một số quá trình định dạng, về cơ bản đó vẫn là đầu ra từ khi xuất khỏi mô-đun count\_number\_of\_words. Vì vậy, đây là điểm trừu tượng cao nhất của đầu ra như được chỉ ở hình 14.3

|  |
| --- |
|  |

Kết quả sản phẩm được phân rã bằng cách sử dụng hai điểm trừu tượng cao nhất như được thể hiện qua biểu đồ cấu trúc ở hình 14.4. Hình 14.4 đồng thời cũng thể hiện sơ đồ luồng dữ liệu ở hình 14.3 là quá đơn giản.

|  |
| --- |
|  |
|  |

DFD không thể hiện luồng logic tương ứng trong trường hợp nếu file được chỉ định bởi người dùng không tồn tại. Mô-đun read\_and\_validate\_file\_name cần phải trả về một status\_flag cho perform\_word\_count. Nếu tên file là không hợp lệ, thì perform\_word\_count sẽ không quan tâm đến tên file này, và thông báo lỗi sẽ được in ra màn hình. Nhưng nếu tên file hợp lệ, nó sẽ được chuyển tới count\_number\_of\_words . Nói chung, khi có luồng dữ liệu có điều kiện, thì cần có luồng điều khiển tương ứng.

Một mô-đun có tính gắn kết cao nếu nó thể hiện một chuỗi thao tác liên quan bởi trình tự thực hiện các bước của sản phẩm, và các thao tác cùng thực hiện chung dữ liệu. Trong hình 14.4, hai mô-đun có sự gắn kết trao đổi (các thành phần của mô-đun được nhóm lại với nhau vì chúng thao tác trên cùng dữ liệu): read\_and\_validate\_file\_name và format\_and\_display\_word\_count. Các mô-đun này cần được phân rã thêm. Kết quả cuối cùng được thể hiện trong hình 14.5. Cả 8 mô-đun đều có sự gắn kết về mặt chức năng (các thành phần của mô-đun được nhóm lại với nhau vì chúng thực hiện chung nhiệm vụ duy nhất được định nghĩa rõ ràng bởi mô-đun)

Và bây giờ, việc thiết kế kiến trúc đã hoàn thành, bước tiếp theo là thiết kế chi tiết. Tại bước này, các cấu trúc dữ liệu và thuật toán được lựa chọn. Bản thiết kế chi tiết của mỗi mô-đun sau đó được giao cho lập trình viên để tiến hành cài đặt. Tương tự như các pha công việc khác của phát triển phần mềm, có sự hạn chế về thời gian, việc cài đặt được tiến hành bởi một nhóm, chứ không phải một lập trình viên đơn lẻ hoàn thành viết code cho cả một mô-đun. Vì lý do này, nên các thiết kế chi tiết của mỗi mô-đun cần được trình bày để có thể hiểu được mà không cần phải tham chiếu đến các mô-đun khác. Hình 14.6 là thiết kế chi tiết của 4 trong 8 mô-đun, 4 mô-đun khác được trình bày trong định dạng khác.

|  |
| --- |
|  |
|  |
|  |
|  |

Các thiết kế trong hình 14.6 không phụ thuộc vào ngôn ngữ lập trình. Tuy nhiên, nếu ban quản lý quyết định ngôn ngữ cài đặt trước khi bắt đầu thiết kế chi tiết, thì sẽ sử dụng **mã giả** để trình bày bản thiết kế chi tiết. Mã giả về cơ bản bao gồm các chú thích được kết nối bởi các câu lệnh điều khiển của ngôn ngữ cài đặt được lựa chọn trước đó. Hình 14.7 là bản thiết kế chi tiết cho 4 mô-đun còn lại của sản phẩm được viết bằng mã giả của C++ hoặc Java.

|  |
| --- |
|  |

Ưu điểm của mã giả đó là ngắn gọn và rõ ràng, và bước cài đặt thường bao gồm việc chuyển các chú thích thành ngôn ngữ lập trình được chọn. Nhược điểm đó là đôi khi các nhà thiết kế có xu hướng thực hiện quá nhiều chi tiết và tạo ra một bản code cài đặt hoàn chỉnh cho một mô-đun hơn là việc viết ra bản thiết kế chi tiết sử dụng mã giả.

Sau khi hoàn thành tài liệu và kiểm thử thành công, bản thiết kế chi tiết được giao cho nhóm cài đặt và tiến hành code. Sản phẩm, sau đó được trải qua các pha còn lại của vòng đời phần mềm cổ điển

***14.3.2. Mở rộng phân tích luồng dữ liệu***

Trong mini case ở trên, sơ đồ luồng dữ liệu (hình 14.3) chỉ có một luồng đầu vào và một luồng đầu ra. Trong trường hợp mở rộng, xem xét hình 14.8. Ở đây sơ đồ có 4 luồng đầu vào và 5 luồng đầu ra, trường hợp này gần với thực tế hơn.

Khi có nhiều luồng đầu vào và nhiều luồng đầu ra, cách thực hiện là tìm điểm trừu tượng cao nhất của từng luồng đầu vào, và điểm trừu tượng cao nhất của từng luồng đầu ra. Sử dụng các điểm này để phân rã sơ đồ luồng dữ liệu cho trước thành các mô-đun với ít luồng đầu vào và đầu ra hơn ban đầu. Tiếp tục quá trình này cho đến khi các mô-đun kết quả có sự gắn kết cao. Cuối cùng, quyết định khớp nối giữa các cặp mô-đun và thực hiện những điều chỉnh cần thiết.

|  |
| --- |
|  |

Hộp 14.1: Các thực hiện Phân Tích Luồng Dữ Liệu:

Thực hiện lặp đi lặp lại các thao tác:

- Tìm điểm trừu tượng cao nhất của đầu vào của mỗi luồng đầu vào

- Tìm điểm trừu tượng cao nhất của đầu ra của mỗi luồng đầu ra

- Phân rã sơ đồ luồng dữ liệu bằng cách sử dụng những điểm trừu tượng này

Thực hiện lặp lại cho đến khi các mô-đun kết quả có tính gắn kết cao

Nếu các khớp nối của các mô-đun quá cao, điều chỉnh lại thiết kế

***14.4. Phân tích giao dịch***

Một giao dịch là một thao tác từ phía người dùng sản phẩm, chẳng hạn như “xử lý một yêu cầu” hay “in danh sách đơn hàng trong ngày”. Phân tích luồng dữ liệu là không phù hợp cho loại sản phẩm, trong đó phải thể hiện được các thao tác liên quan, tương tự nhau về tổng thể, nhưng lại có sự khác biệt trong chi tiết.

|  |
| --- |
|  |

Hộp 14.2:

Cách thực hiện một Phân Tích Giao Dịch:

* Thiết kế kiến trúc thành 2 phần:

+ Phần phân tích

+ Phần điều phối

* Với mỗi nhóm thao tác liên quan với nhau:

Thiết kế một mô-đun cơ bản và khởi tạo nhiều lần nếu cần thiết

Một ví dụ điển hình cho loại sản phẩm này đó là phần mềm kiểm soát một máy rút tiền tự động. Khách hàng đưa thẻ từ vào trong khe thẻ, nhập mật khẩu thẻ của mình, sau đó có thể thực hiện được các thao tác như kiểm tra hoặc gửi tiết kiệm cho thẻ ngân hàng, rút tiền từ tài khoản, hoặc xác định số dư tài khoản của mình là bao nhiêu. Loại sản phẩm này sẽ được mô tả trong hình 14.9. Một cách tốt để thiết kế những sản phẩm loại này đó là chia thành 2 phần: phần phân tích và phần điều phối. Phần phân tích sẽ xác định loại giao dịch là gì sau đó chuyển thông tin qua cho phần điều phối, phần điều phối sẽ thực hiện giao dịch.

Một mô-đun được gọi là gắn kết logic, khi nó thực hiện một loạt các thao tác liên quan nhau, một trong số đó được chọn bởi mô-đun gọi. Thiết kế được thể hiện trong hình 14.10 là không hợp lý, bởi thiết kế có 2 mô-đun có sự gắn kết logic:edit\_any\_transaction và update\_any\_file. Đồng thời, mặt khác, sẽ là lãng phí khi có 5 mô-đun chỉnh sửa gần như giống nhau và 5 mô-đun cập nhập gần như giống nhau

|  |
| --- |
|  |
|  |

Giải pháp ở đây đó là tái sử dụng phần mềm: một mô-đun chỉnh sửa cơ bản nên được thiết kế, viết code, lập tài liệu, kiểm thử, sau đó được khởi tạo 5 lần. Mỗi phiên bản sẽ có những sự khác biệt nhỏ, không đáng kể. Tương tự, một mô-đun cập nhập cơ bản, được khởi tạo 5 lần, mỗi phiên bản sẽ có những sai khác nhỏ để phục vụ cho 5 loại cập nhập khác nhau. Kết quả thiết kế sẽ có sự gắn kết cao và khớp nối thấp

***14.5 Thiết kế hướng dữ liệu***

Quy tắc cơ bản của thiết kế hướng dữ liệu đó là thiết kế dựa trên cấu trúc dữ liệu mà sản phẩm sẽ hoạt động. Do đó, đầu tiên cần xác định được cấu trúc dữ liệu của sản phẩm. Sau đó, trong mỗi giai đoạn sẽ được cung cấp cùng loại cấu trúc dữ liệu mà sản phẩm sẽ thao tác. Có những kỹ thuật thiết kế hướng dữ liệu, những kỹ thuật được biết đến nhiều nhất là của Michael Jackson (1975), Warnier (1976), and Orr (1981). Cả 3 kỹ thuật này đều có những điểm tương tự nhau.

Thiết kế hướng dữ liệu chưa từng phổ biến được như thiết kế hướng đối tượng, cùng với sự phát triển của các mô hình hướng đối tượng, thiết kế hướng dữ liệu dần trở nên lỗi thời.

***14.6 Thiết kế hướng đối tượng***

Mục tiêu của thiết kế hướng đối tượng (object-oriented design – OOD) là thiết kế sản phẩm dưới dạng các đối tượng, nghĩa là, khởi tạo các lớp và lớp con được trích xuất trong quá trình phân tích hướng đối tượng. Những ngôn ngữ lập trình cổ điển như C, phiên bản cũ (trước 2000) của COBOL và Fortran không hỗ trợ lập trình hướng đối tượng. Vì vậy, OOD chỉ phù hợp với những người sử dụng các ngôn ngữ lập trình hướng đối tượng như: Smalltalk (Goldberg and Robson, 1989), C++ (Stroustrup, 2003), Ada 95 (ISO/IEC 8652, 1995), và Java (Flanagan, 2005).

Mặc dù OOD không hỗ trợ các ngôn ngữ lập trình cổ điển, nhưng một tập con lớn của OOD có thể được sử dụng. Khi ngôn ngữ cài đặt không hỗ trợ thừa kế, giải pháp đó là sử dụng **thiết kế dữ liệu trừu tượng**. Kiểu dữ liệu trừu tượng có thể được cài đặt bởi hầu hết các ngôn ngữ hỗ trợ câu lệnh **type**. Kể cả, những ngôn ngữ cổ điển không hỗ trợ những câu lệnh type, và do đo không hỗ trợ kiểu dữ liệu trừu tượng, vẫn có thể cài đặt đóng gói dữ liệu.

(Type programming language: các ngôn ngữ lập trình mà các kiểu dữ liệu như integer, character… đã được định nghĩa trước như một phần của ngôn ngữ, tất cả các hằng và biến khi được khởi tạo phải được mô tả là kiểu dữ liệu nào)

Hai bước quan trọng của OOD đó là hoàn thành sơ đồ lớp và thực hiện thiết kế chi tiết.

Trong bước đầu tiên, hoàn thành sơ đồ lớp, định dạng của các thuộc tính cần phải được quyết định, các phương thức cần được gán cho các lớp liên quan.

Định dạng của các thuộc tính có thể được suy ra trực tiếp từ việc phân tích artifacts (tạo tác). Ví dụ, Ở Hoa Kỳ, ngày 3 tháng 12 năm 1947 được viết là 12/03/1947 (mm/dd/yyyy), còn ở châu Âu viết là 03/12/1947 (dd/mm/yyyy). Bất kể là định dạng nào, đều cần tổng cộng 10 ký tự để thể hiện.

Thông tin để quyết định định dạng của thuộc tính có được từ luồng công việc phân tích, do đó định dạng có thể được thêm sơ đồ lớp từ luồng công việc đó. Tuy nhiên, mô hình hướng đối tượng là lặp đi lặp lại. Mỗi lần lặp sẽ dẫn đến những thay đổi trên những gì đã hoàn thành được. Do đó các thông tin nên được thêm vào mô hình UML càng muộn càng tốt.

Ví dụ, sơ đồ lớp của tổ chức MSG:

|  |
| --- |
|  |
|  |

Đây là 2 lần lặp đầu tiên của sơ đồ lớp case study tổ chức MSG. Nếu các thuộc tính được thêm vào sớm, chúng có thể bị thay đổi hoặc chuyển từ lớp này sang lớp khác cho đến khi nhóm phân tích hài lòng với sơ đồ lớp. Nói chung, sẽ là không hợp lý khi thêm các thành phần vào sơ đồ lớp (hoặc bất kỳ sơ đồ UML nào) trước khi thực sự cần thiết phải làm vậy, vì sẽ làm khó khăn khi thực hiện các bước lặp. Việc thực hiện định dạng khi chưa thực sự cần thiết là không hợp lý.

Thành phần chính khác trong bước đầu tiên của OOD là gán phương thức (cài đặt các thao tác) cho lớp. Việc quyết định tất cả các thao tác của sản phẩm được thực hiện cách xem xét các biểu đồ tương tác của từng kịch bản. Đây là việc đơn giản. Phần khó khăn đó là quyết định phương thức nào được liên kết với mỗi mô-đun.

Một phương thức có thể được gán cho một lớp hoặc một client gửi thông điệp cho đối tượng của lớp đó (client của một đối tượng là một chương trình đơn vị gửi thông điệp cho đối tượng đó).

Một quy tắc có thể hỗ trợ quyết định cách gán một thao tác đó là ẩn thông tin. Nghĩa là, các biến state của lớp nên được khai báo là private (chỉ có thể được truy cập bởi các đối tượng bên trong lớp) hoặc protected (chỉ có thể được truy cập bởi các đối tượng bên trong lớp hoặc các đối tượng của lớp con). Theo đó, các thao tác thực hiện trên các biến state phải là cục bộ bên trong lớp.

Quy tắc thứ 2, đó là nếu một thao tác nhất định được liên quan tới nhiều client khác nhau của một đối tượng, nên có 1 bản sao chép cài đặt thao tác như là một phương thức của đối tượng hơn là có bản sao chép của mỗi client của đối tượng đó. (Nếu một phương thức được gọi bởi nhiều client của một đối tượng thì sẽ gán phương thức đó cho đối tượng chứ không phải các client. Nếu có nhiều đối tượng X gọi đến một hành động k của đối tượng Y, thì phương thức để thực hiện hành động k nên gán cho lớp của đối tượng Y, mà không nên gán cho lớp của đối tượng X)

Quy tắc thứ 3, hỗ trợ việc quyết định nơi để đặt phương thức là sử dụng thiết kế hướng trách nhiệm. Thiết kế hướng đối tượng là một khía cạnh quan trọng của mô hình hướng đối tượng. Nếu một client gửi thông điệp tới một đối tượng, thì đối tượng có trách nhiệm về mọi khía cạnh thực hiện yêu cầu của client. Client không được biết và cũng không được phép biết cách đối tượng thực hiện yêu cầu. Khi yêu cầu được thực hiện, quyền kiểm soát trả về cho client. Lúc này, client chỉ biết yêu cầu đã được thực hiện mà không biết làm sao để đạt được kết quả như vậy.( Thiết kế hướng trách nhiệm. Nếu một hành động mà không thể gán thành phương thức cho lớp khác, thì lớp của đối tượng cần thực hiện hành động đó phải chứa phương thức tương ứng hành động đó)

Để xem các nguyên tắc này được sử dụng như thế nào, bây giờ chúng tôi minh họa OOD bằng 2 case study sau đây

***14.7. Thiết kế hướng đối tượng: Case study Thang Máy***

*Bước 1: Hoàn thành sơ đồ lớp*

Sơ đồ lớp chi tiết được hoàn thành bằng cách thêm thao tác (phương thức) cho sơ đồ lớp hình 13.12. Trong trường hợp cài đặt bằng java, cần thêm 2 lớp.

|  |
| --- |
|  |

Lớp Elevator Application tương ứng với hàm main của C++, lớp Elevator Utilities chứa Java routines tương ứng với hàm C++ được khai báo bên ngoài lớp C++.

Xem xét lần lặp đầu tiên của CRC card cho bộ điều khiển phụ thang máy (hình 13.14).

|  |
| --- |
|  |

Trách nhiệm chia thành 2 nhóm. Nhóm đầu tiên là cho 5.Start time, được gán cho bộ điều khiển thang máy trên cơ sở thiết kế hướng trách nhiệm, nhiệm vụ này được thực hiện bởi bộ điều khiển của thang máy.

11 trách nhiệm còn lại(1 đến 4, 6 đến 12) đều có dạng “Gửi thông điệp tới một lớp khác để thực hiện một điều gì đó”. Chúng ta nên sử dụng thiết kế hướng trách nhiệm để gán các phương thức liên quan cho các lớp. Ngoài ra để đảm bảo an toàn, quy tắc ẩn thông tin được áp dụng với cả 11 case trên.

Có 2 lý do để phương thức closeDoors và openDoor được gán cho lớp **Elevator Doors**. Đó là, một client của lớp **Elevator Doors** (ở đây, là một đối tượng của lớp **Elevator Subcontroller** ) gửi thông điệp tới một đối tượng của lớp **Elevator Doors** để đóng hoặc mở của thang máy, yêu cầu được thực hiện bởi các phương thức liên quan. Mọi khía cạnh của cả hai phương thức này đều được thực hiện bên trong lớp **Elevator Doors**. Ngoài ra, kết quả ẩn thông tin hoàn toàn độc lập với lớp **Elevator Doors**.

Hai phương thức moveDownOneFloor và moveUpOneFloor được gán cho lớp **Elevator**. Không cần chỉ rõ cách làm thang máy dừng lại. Nếu cả hai phương này đều không được gọi thì thang máy sẽ không di chuyển, không có cách nào làm thang máy di chuyển ngoài việc gọi một trong hai phương thức trên.

Cuối cùng, phương thức turnOffButton và turnOnButton được gán cho cả lớp **Elevator Button** và lớp **Floor Button**. Lý do tương tự như khi gán các phương thức cho lớp **Elevator Doors** và lớp **Elevator**. Đầu tiên, quy tắc của thiết kế hướng trách nhiệm yêu cầu các nút có toàn quyền kiểm soát việc chúng bật hay tắt. Tiếp theo, quy tắc ẩn thông tin yêu cầu trạng thái bên trong của nút cần được ẩn. Phương thức trả về kết quả nút đang bật hay tắt cần được nằm bên trong lớp **Elevator Button**, và tương tự nằm trong lớp **Floor Button**. Phương thức turnOffButton và turnOnButton được khai báo trừu tượng bên trong lớp **Button.** Trong thời gian chạy, phiên bản đúng của turnOffButton hoặc turnOnButton sẽ được gọi.

*Bước 2: Thể hiện thiết kế chi tiết*

Một bản thiết kế chi tiết được phát triển cho tất cả các lớp. Có thể sử dụng bất kỳ kỹ thuật nào phù hợp. Bản thiết kế chi tiết cho phương thức elevatorSubcontrollerEventLoop được trình bày dưới dạng mã giả trong hình 14.12, ngoài ra cũng có thể sử dụng dạng bảng như hình 14.6

|  |
| --- |
|  |

Hình 14.12 được xây dựng từ biểu đồ trạng thái hình 13.13. Ví dụ, 2 sự kiện elevator button pushed và elevator button turned off được cài đặt bởi hai câu lệnh if đầu hình. Tiếp theo là thao tác của trạng thái Processing New **Request**. Điều kiện **else-if** tương ứng cho sự kiện tiếp theo dẫn tới **Elevator Subcontroller Event Loop**.

***14.8. Thiết kế hướng đối tượng Case study Tổ chức MSG***

*Bước 1: Hoàn thành sơ đồ lớp*

Hình 14.13 thể hiện mô hình lớp tổng thể của Case study tổ chức MSG

|  |
| --- |
|  |

Lớp Date được vẽ nét đứt thể hiện lớp này chỉ cần thiết cho cài đặt C++, Java có những lớp được xây dựng sẵn để xử lý ngày tháng: java.text.Dateformat và java.util.Calendar.

Định dạng của các thuộc tính được rút ra từ cuộc thảo luận với khách hàng và người dùng. Xem xét các form (11.4.2) cũng rất hữu ích trong trường hợp này. Hình 14.14 thể hiện một phần kết quả.

Các phương thức của sản phẩm được tìm thấy trong các sơ đồ tương tác. Nhiệm vụ của người thiết kế là quyết định gán phương thức nào cho mỗi lớp. Ví dụ, quy ước của một sản phẩm phần mềm hướng đối tượng là các thuộc tính của các lớp cần có phương thức mutator (setter) setAttribute, dùng để gán giá trị cho thuộc tính và phương thức accessor (getter) getAttribute, dùng để lấy giá trị của thuộc tính.

Ví dụ, phương thức setAssetNumber để gán giá trị cho tài sản (đầu tư hoặc thế chấp).

|  |
| --- |
|  |

Trong mô hình cổ điển cần có hai hàm riêng set\_investment\_number và set\_mortgage\_number. Nhưng mô hình hướng đối tượng hỗ trợ kế thừa nên phương thức setAssetNumber nên được gán cho lớp **Asset**.

|  |
| --- |
|  |

Hình 14.15 trình bày phương thức được áp dụng không chỉ cho thể hiện của lớp **Asset** mà còn là kết quả của kế thừa cho các thể hiện của các lớp con của **Asset**, lớp **Investment** và lớp Mortgage.

*Bước 2: Thực hiện thiết kế chi tiết*

Tiếp theo, xây dựng thiết kế chi tiết bằng cách dựng các phương thức và quyết định phương đó làm gì.

Hình 14.16 là bản thiết kế chi tiết (mã giả java) của phương thức computeEstimatedFunds thuộc lớp **EstimateFundsForWeek**.

Hình 14.17 phương thức EstimateFundsForWeek lớp **Mortgage**.

|  |
| --- |
|  |
|  |

***14.9: Luồng công việc thiết kế***

Mục đích của luồng công việc thiết kế là tinh chỉnh các tạo tác của luồng công việc phân tích cho đến khi tài liệu ở dạng mà các lập trình viên có thể thực hiện được. Đầu vào của luồng công việc thiết kế là các tạo tác của luồng công việc phân tích. Trong luồng công việc này, các tạo tác được lặp đi lặp lại và phát triển cho đến khi chúng ở định dạng mà các lập trình viên có thể thực hiện được.

|  |
| --- |
|  |

Thực hiện thiết kế hướng đối tượng:

- Hoàn thành sơ đồ lớp

- Thực hiện thiết kế chi tiết

Ngoài thiết kế hướng đối tượng, luồng công việc thiết kế cũng yêu cầu thực hiện một số các quyết định khác:

- Lựa chọn ngôn ngữ lập trình cho sản phẩm phần mềm.

- Tái sử dụng bao nhiêu sản phẩm phần mềm hiện có cho sản phẩm phần mềm mới.

- Tính di động.

- Phân bổ từng thành phần phần mềm cho thành phần phần cứng mà nó sẽ chạy.

Quy trình hợp nhất thường được dùng cho những sản phẩm phần mềm lớn, thường khoảng 500.000 dòng code hoặc hơn. Case study tổ chức MSG chỉ khoảng 5000 dòng code. Do đó nhiều khía cạnh của quy trình hợp nhất không thể áp dụng được vào case study này. Ví dụ, một phần quan trọng của luồng công việc phân tích là chia sản phẩm thành các gói phân tích. Mỗi gói gồm các lớp liên quan đến nhau, thường là liên quan đến một tập nhỏ các actors, và có thể cài đặt như một đơn vị riêng lẻ.

Ý tưởng phân rã một luồng công việc lớn thành các luồng công việc nhỏ, độc lập được chuyển sang luồng công việc. Ở đây, mục tiêu là chia nhỏ các luồng công việc sắp thực hiện thành các phần có thể quản lý được, gọi là hệ thống con. Và tất nhiên chỉ hợp lý khi chia nhỏ đối với quy trình hợp nhất, còn đối với case study của MSG Foundation thì việc chia nhỏ là không hợp lý, bởi Tổ chức này quá nhỏ. Rất dễ hiểu, việc chia nhỏ một sản phẩm phần mềm thành các gói là một ví dụ về “chia để trị”.

Có hai lý do tại sao quy trình công việc lớn được chia thành các hệ thống con:

- Việc triển khai cài đặt các hệ thống con dễ dàng hơn việc cài đặt một hệ thống lớn. Đây là một ví dụ cho chia để trị.

- Nếu hệ thống con được triển khai thực sự độc lập thì chúng có thể được thực hiện bởi các lập trình viên một cách song song. Điều này giúp sản phẩm phần mềm có thể được hoàn thành nhanh hơn

Kiến trúc của một sản phẩm phần mềm bao gồm các thành phần khác nhau và cách chúng kêt hợp với nhau. Việc phân bổ các thành phần cho hệ thống con là một phần của nhiệm vụ kiến trúc. Và nhiệm vụ này được quyết định bởi kiến trúc sư phần mềm.

Một kiến trúc sư cần phải biết “đánh đổi”. Một sản phẩm phần mềm cần đáp ứng các yêu cầu chức năng(là các trường hợp sử dụng) và các yêu cầu phi chức năng(tính di động, độ tin cậy, bảo mật). Nhưng cần thực hiện các yêu cầu này trong giới hạn ngân sách và thời gian. Đây là điều hầu như không thể làm được. Cho nên luôn luôn cần có sự thỏa hiệp. Khách hàng cần nới lỏng một số yêu cầu, tăng ngân sách và dời thời gian giao hàng. Và kiến trúc sư phần mềm cần hỗ trợ việc đưa ra quyết định cho khách hàng bằng cách vạch ra rõ ràng những sự “đánh đổi”.

Trong một số trường hợp sự đánh đổi là điều hiển nhiên. Ví dụ: kiến trúc sư chỉ ra rằng một tập hợp các yêu cầu bảo mật mới sẽ cần thêm 3 tháng và 350.000 đô la để đưa vào sản phẩm. Nếu sản phẩm là một mạng lưới ngân hàng quốc tế thì tất nhiên khách hàng sẽ không đồng ý thỏa hiệp. Tuy nhiên, đôi lúc khách hàng cũng cần phải đưa ra các quyết định quan trọng. Ví dụ: Một kiến trúc sư chỉ ra rằng việc trì hoãn một yêu cầu cụ thể cho đến khi phần mềm được phân phối sẽ có thể tiết kiệm được 150.000 đô la bây giờ nhưng sẽ mất 300.000 đô la để kết hợp sau này. Quyết định cuối cùng vẫn là của khách hàng, tuy nhiên kiến trúc sư cần hỗ trợ để khách hàng đưa ra quyết định đúng đắn nhất.

Kiến trúc của một sản phẩm quyết định đến sự thành công của một phần mềm. Và các quyết định về kiến trúc được thực hiện trong luồng công việc thiết kế. Nếu luồng công việc yêu cầu được thực hiện không tốt, vẫn có thể có một dự án thành công miễn là dành thêm thời gian và tiền bạc. Tương tự nếu luồng công việc phân tích không đầy đủ có thể dẫn đến việc phải bỏ nhiều công sức hơn trong luồng công việc thiết kế. Tuy nhiên nếu kiến trúc không tối ưu thì không thể nào khắc phục, phải thiết kế lại ngay lập tức.

Do đó, điều cần thiết là nhóm phát triển phải bao gồm nhóm kiến trúc sư phần mềm có chuyên môn cao.

***14.10. Luồng công việc kiểm thử: thiết kế***

Mục tiêu của kiểm thử là để xác minh các thông số kỹ thuật được đưa vào thiết kế đã chính xác và hoàn chỉnh chưa. Ví dụ: thiết kế phải không có lỗi logic và các giao diện phải được định nghĩa trực tiếp. Điếu quan trọng là bất kỳ lỗi nào trong thiết kế cũng phải được phát hiện trước khi viết code. Nếu không chi phí cho việc sửa lỗi sau này sẽ rất lớn. Các lỗi thiết kế có thể được phát hiện bằng các phương tiện kiểm tra thiết kế hay hướng dẫn thiết kế. (design walkthroughs)

Khi sản phẩm được định hướng giao dịch, việc kiểm tra thiết kế phải được thực hiện. Các cuộc kiểm tra bao gồm tất cả các loại giao dịch có thể có phải được lên lịch. Người kiểm tra nên liên hệ từng giao dịch trong thiết kế với các thông số kỹ thuật. Cho biết giao dịch phát sinh như thế nào từ tài liệu đặc tả. Ví dụ nếu sản phẩm là một máy rút tiền tự động, một giao dịch tương ứng với mỗi hoạt động mà khách hàn có thể thực hiện, chẳng hạn như rút tiền, đổi mật khẩu,…Trong các trường hợp khác, sự tương ứng giữa thông số kỹ thuật và giao dịch không nhất thiết phải là 1-1. Ví dụ: Trong hệ thống điều khiển đèn giao thông, nếu một chiếc ô tô lái qua một tấm cảm biến dẫn đến hệ thống quyết định thay đổi một đèn cụ thể từ đỏ sang xanh trong 15 giây, thì các tác động khác từ tấm cảm biến đó sẽ bị bỏ qua. Ngược lại, để tăng tốc luồng giao thông, một tác động duy nhất có thể khiến toàn bộ loạt đèn bị thay đổi từ đỏ sang xanh.

Có một hạn chế trong việc kiểm tra hướng giao dịch đó là không phát hiện ra các trường hợp mà nhà thiết kế bỏ qua. Ví dụ: Các thông số kỹ thuật cho bộ điều khiển đèn giao thông quy định khoảng thời gian từ 11:00 đêm đến 6:00 sáng thì tất cả đèn vàng phải nhấp nháy theo một hướng và các đèn đỏ phải nhấp nhánh theo hướng khác. Tuy nhiên nếu nhà thiết kế bỏ qua điều này, thì các giao dịch được đặt lịch để tạo ra lúc 11:00 đêm đến 6:00 sáng sẽ không được đưa vào thiết kế và tất nhiên chúng không thể được kiểm tra khi dựa trên các giao dịch. Do đó kiểm tra hướng giao dịch thôi là không đủ, còn cần phải có kiểm tra dựa trên thông số kỹ thuật để đảm bảo các báo cáo trong tài liệu không bị bỏ sót hoặc hiểu sai.

***14.11. Luồng công việc kiểm thử: Case study tổ chức MSG***

Khi thiết kế đã hoàn thành thì các khía cạnh của thiết kế MSG Foundation phải được kiểm tra bằng cách kiểm tra thiết kế. Mọi tạo tác thiết kế đều cần được kiểm tra, xem xet. Ngay cả khi không tìm thấy lỗi nào thì thiết kế vẫn có thể bị thay đổi một lần nữa, có thể là thay đổi hoàn toàn, khi case study được cài đặt.

***14.12. Các kỹ thuật để thiết kế chi tiết.***

Một số kĩ thuật để thiết kế chi tiết đã được nêu ra ở các chương trước như là kĩ thuật chỉnh sửa từng bước trong chương 5, hay chương 6 từng gợi ý rằng việc triển khai một sản phẩm hoàn chỉnh sau đó chứng minh nó đúng có thể sẽ phản tác dụng. Ngoài ra việc phát triển bản thử, thiết kế chi tiết song song hay kiểm tra cẩn thận code cũng là một kĩ thuật để thiết kế chi tiết.

Các kĩ thuật chính thức được áp dụng cho thiết kế chi tiết hỗ trợ theo 3 cách sau:

- Kỹ thuật hiện đại chứng minh tính đúng đắn, không thường được áp dụng cho cả sản phẩm, mà được áp dụng cho các phần kích thước mô-đun của sản phẩm

- Việc phát triển bản thử cùng với thiết kế chi tiết sẽ tạo nên một thiết kế có ít lỗi hơn.

- Nếu cùng một lập trình viên chịu trách nhiệm về cả thiết kế chi tiết và thực hiện thì lập trình viên đó sẽ cảm thấy chắc tin tưởng rằng thiết kế chi tiết là chính xác. Thái độ tích cực này đối với thiết kế sẽ dẫn đến ít lỗi hơn.

***14.13.Kỹ thuật thiết kế thời gian thực***

Phần mềm thời gian thực được đặc trưng bởi ràng buộc về thời gian, nếu một ràng buộc không được đáp ứng thì thông tin sẽ bị mất. Đặc biệt mỗi đầu vào phải được xử lý trước khi đầu vào tiếp theo đến. Ví dụ: Một lò phản ứng hạt nhân được điều khiển bằng máy tính. Các thông tin đầu vào như nhiệt độ của lõi và mức nước trong buồng phản ứng liên tục được gửi đến, máy tính đọc giá trị từng đầu vào và thực hiện xử lý cần thiết trước khi đầu vào tiếp theo đến.

Một đặc điểm của hệ thống thời gian thực là chúng được thực hiện trên phần cứng phân tán. Ví dụ: Phần mềm điều khiển máy bay chiến đấu có thể triển khai trên năm máy tính, một máy tính để điều hướng, một máy tính dùng hệ thống vũ khí, một máy tính dùng để điều khiển máy bay chiến đấu điện tử, một máy tính điều khiển động cơ, cánh máy bay và một máy tính đề xuất chiến thuật chiến đấu.

Việc thiết kế hệ thống phân tán cũng phát sinh một số vấn đề mang tính chất phân tán. Ví dụ: Trong lúc điều khiến chiến đấu, máy tính chiến thuật đề xuất phi công nên lao lên, trong khi máy tính vũ khí đề xuất phi công lao xuống để một vũ khí được phóng trong điều kiện tối ưu. Tuy nhiên phi công lại lựa chọn di chuyển sang bên phải, từ đó gửi tín hiệu đến phần cứng máy tính của chuyến bay để thực hiện điều chỉnh cần thiết sao cho máy bay di chuyển theo hướng đã chỉ định. Quyết định cuối cùng vẫn ở con người. Ngoài ra, chuyển động thực tế cần được chuyển tiếp tới máy tính chiến thuật và máy tính vũ khí để tiếp tục đề xuất các phương án mới trên tình hình thực tế.

Một khó khăn nữa của hệ thống thời gian thực là vấn đề động bộ hóa. Có thể phát sinh các tính huống như bế tắc. Tình huống này đặc biệt rắc rối trong hệ thống thời gian thực vì trong hệ thống không có quyền kiểm soát thứ tự hay thời gian của các đầu vào.

Như vậy để có thể khắc phục các khó khăn trên thì kỹ thuật thiết kế phải cung cấp một cơ chế để kiểm tra rằng khi được triển khai, thiết kế có thể đọc và xử lý dữ liệu vào ở tốc độ cần thiết. Hơn nữa các vấn đề đồng bộ hóa trong thiết kế cũng đã được giải quyết chính xác.

Hiện nay, công nghệ phần cứng đã có những tiến bộ vượt xa về mọi mặt so với công nghệ phần mềm. Dù không phát triển bằng công nghệ phần cứng nhưng vẫn còn rất nhiều thời gian để phát triển công nghệ phần mềm, để hướng tới thiết kế các hệ thống thời gian thực có thể đáp ứng mọi yêu cầu và giải quyết mọi vấn đề về đồng bộ hóa có thể nảy sinh.

***14.14.CASE:Công cụ cho thiết kế.***

Như đã nói trong phần trước, một khía cạnh quan trọng của thiết kế là kiểm tra xem các tạo tác thiết kế có kết hợp chính xác tất cả các khía cạnh của tạo tác phân tích hay không. Do đó cần có một công cụ Case để có thể sử dụng cho cả tạo tác phân tích và tạo tác thiết kế. Được gọi là front-end hoặc công cụ upperCase(trái ngược với back-end hay công cụ lowCase).

Các công cụ upperCase thường được xây dựng dựa trên một từ điển dữ liệu. Công cụ Case có thể kiểm tra mọi trường của mọi bản ghi trong từ điển được đề cập ở đâu đó trong thiết kế hoặc mọi mục trong thiết kế được trình bày trong sơ đồ luồng dữ liệu. Ngoài ra, nhiều công cụ upperCase còn kết hợp công cụ kiểm tra tính nhất quán sử dụng từ điển dữ liệu để xác định rằng mọi mục trong thiết kế đã được khai báo trong thông số kỹ thuật và ngược lại rằng mọi mục trong thông số kỹ thuật đều xuất hiện trong thiết kế.

Hơn nữa, nhiều công cụ upperCase còn kết hợp trình tạo màn hình và trình tạo báo cáo. Nghĩa là, khách hàng có thể chỉ định mục nào sẽ xuất hiện trong báo cáo hay trên màn hình và vị trí mỗi mục xuất hiện. Bởi lẽ, mọi chi tiết đều nằm trong từ điển dữ liệu, công cụ CASE có thể dễ dàng sinh code để in báo cáo hoặc hiển thị ra màn hình như yêu cầu của khách hàng.

Một số công cụ Case cũng tích hợp các công cụ quản lý tỷ lệ để ước tính và lập kế hoạch.

***14.15. Số liệu cho thiết kế***

Nhiều số liệu được sử dụng để mô tả các khía cạnh của thiết kế. Ví dụ: số lượng mô-đun hoặc lớp là thước đo thô về kích thước của sản phẩm.

Độ phức tạp chu kỳ M là số nhánh trong mã thực tế. Một số quan điểm cho rằng M là thước đo chất lượng thiết kế. M càng nhỏ thì chất lượng càng tốt. M có một điểm mạnh đó là dễ tính tuy nhiên nó cũng có vấn đề. M hoàn toàn là một thước đo của độ phức tạp điều khiển nhưng lại bỏ qua độ phức tạp của dữ liệu.

Khi mô hình cổ điển được sử dụng, một lớp số liệu liên quan cho giai đoạn thiết kế dựa trên việc biểu diễn thiết kế kiến trúc(dưới dạng một đồ thị có hướng) với các module được gửi bởi các nút và sự liên kết giữa các mô-đun(được biểu diễn bằng các cung).

Fan-in của một module được xác định bằng số lượng luồng vào mô-đun cộng với số lượng cấu trúc dữ liệu chung mà module đã truy cập.

Fan-out của một module được xác định bằng số lượng luồng ra khỏi mô-đun cộng với số lượng cấu trúc dữ liệu toàn cục được mô-đun cập nhật.

Thước đo độ phức tạp của mô-đun=length x (fan-in x fan-out)2; Với length là kích thước mô-đun.

Tuy phức tạp như vậy nhưng một số thí nghiệm đã chứng minh đây chưa phải là một số liệu tối ưu để đo độ phức tạp.

Vấn đề về số liệu thiết kế càng phức tạp hơn khi mô hình hướng đối tượng được sử dụng. Ví dụ: độ phức tạp chu kỳ M không đảm bảo vì nó bỏ qua độ phức tạp dữ liệu, mà độ phức tạp dữ liệu ảnh hưởng vào độ phức tạp của hướng đối tượng. Chính vì vậy mà một số các số liệu về độ phức tạp lại không được áp dụng.

***14.16: Thách thức của luồng công việc thiết kế.***

Không làm quá nhiều trong quy trình phân tích, có nghĩa là nhóm phân tích không được bắt đầu sớm phần của luồng công việc thiết kế.

Trong luồng công việc thiết kế, nhóm thiết kế có thể bị đi sai hướng: làm quá nhiều hoặc làm quá ít.

Xem xét bản thiết kế chi tiết bằng mã giả trong hình 14.7, nhiều nhà thiết kế có thể sẽ tiến hành thực hiện viết bản thiết kế chi tiết bằng C++ hoặc Java. Có nghĩa là họ sẽ tiến hành viết code các lớp chứ không phải viết mã giả. Việc này sẽ gây ra tốn thời gian, bởi lẽ thay vì chỉ viết phác thảo các lớp, mô-đun họ lại ngồi viết code, và đồng thời cũng sẽ mất thời gian hơn khi sửa chữa lỗi trong thiết kế. Các nhà thiết kế cần phải chống lại ham muốn làm nhiều hơn những gì học được yêu cầu.

Đồng thời, các nhà thiết kế cũng có thể phạm vào lỗi làm quá ít. Xem xét đến bản thiết kế chi tiết trong hình 14.16. Nếu nhóm thiết kế làm vội vã, và quyết định thu nhỏ bản thiết kế chi tiết chỉ còn lại như bản tường thuật. Thậm chí nhóm thiết kế còn quyết định rằng những lập trình viên nên tự mình làm các bản thiết kế chi tiết. Cả hai quyết định trên đều sẽ dẫn đến sai lầm. Lý do quan trọng nhất của thiết kế chi tiết đó là để đảm bảo rằng mọi giao diện đều đúng. Bản tường thuật là không đủ cho mục đích này, không có thiết kế chi tiết rõ ràng là thiếu sót. Chính vì vậy, một thách thức lớn đặt ra cho nhóm thiết kế đó là họ thực hiện đúng lượng công việc được giao.

Một thách thức lớn khác đó là, nhóm thiết kế thiếu một người trưởng nhóm là một nhà thiết kế tài giỏi. Một nhóm thiết kế xuất sắc thì cần một người dẫn dắt xuất sắc. Tuy nhiên người thiết kế giỏi thì có thể được đào tạo còn người thiết kế xuất sắc thì chỉ có thể được tạo ra bởi người thiết kế xuất sắc, mà những người thiết kế xuất sắc thì lại rất hiếm. Vậy nên, thử thách đó là cần nuôi dưỡng người thiết kế xuất sắc. Họ cần được tìm ra và xác định càng sớm càng tốt, không phải lúc nào người thiết kế giỏi nhất cũng là người có nhiều kinh nghiệm nhất. Khi tìm được cần phải cho họ được theo học những người thiết kế xuất sắc để giúp học phát triển tốt nhất.

***Tổng kết chương 14:***

* 3 kỹ thuật thiết kế cơ bản được giới thiệu trong chương:

1. Thiết kế hướng thao tác
2. Thiết kế hướng dữ liệu
3. Thiết kế hướng đối tượng

* Thiết kế hướng đối tượng có:

1. Phân tích luồng dữ liệu
2. Phân tích giao dịch

* Các bước thực hiện thiết kế hướng đối tượng

1. Hoàn thành sơ đồ lớp
2. Thực hiện thiết kế chi tiết