**Dịch bài tập lớn**

**16.5 Quản lý bảo trì sau giao hàng**

Các vấn đề về quản lý bảo trì sau giao hàng hiện đang được xem xét

**16.5.1 Báo cáo khiếm khuyết**

Điều đầu tiên cần thiết khi bảo trì một sản phẩm là cơ chế thay đổi sản phẩm. Đối với việc bảo trì khắc phục, nghĩa là loại bỏ các lỗi còn sót lại, nếu sản phẩm có vẻ hoạt động không chính xác, thì người dùng phải gửi báo cáo lỗi. Điều này phải bao gồm đủ thông tin để cho phép lập trình viên bảo trì tạo lại sự cố, thường là một số loại lỗi phần mềm. Ngoài ra, người lập trình bảo trì phải chỉ ra mức độ nghiêm trọng của khiếm khuyết; các danh mục mức độ nghiêm trọng điển hình bao gồm nghiêm trọng, chính, bình thường, nhỏ và tầm thường.

Tốt nhất, mọi lỗi do người dùng báo cáo nên được sửa ngay lập tức. Trong thực tế, các tổ chức lập trình thường thiếu nhân lực, với công việc tồn đọng, cả phát triển và bảo trì. Nếu khiếm khuyết là nghiêm trọng, chẳng hạn như nếu một sản phẩm trả lương bị hỏng vào ngày trước ngày trả lương hoặc trả quá hoặc làm thiếu nhân viên, thì phải thực hiện hành động sửa chữa ngay lập tức. Nếu không, mỗi báo cáo khiếm khuyết ít nhất phải nhận được một cuộc điều tra sơ bộ ngay lập tức.

Lập trình viên bảo trì trước tiên nên tham khảo tệp báo cáo lỗi. Điều này bao gồm tất cả các lỗi được báo cáo mà vẫn chưa được khắc phục, cùng với các đề xuất để khắc phục chúng, nghĩa là, các cách để người dùng bỏ qua phần sản phẩm có vẻ là nguyên nhân gây ra lỗi, cho đến khi lỗi đó có thể được đã sửa. Nếu lỗi đã được báo cáo trước đó, bất kỳ thông tin nào trong tệp báo cáo lỗi phải được cung cấp cho người dùng. Tuy nhiên, nếu những gì người dùng báo cáo dường như là một lỗi mới, thì lập trình viên bảo trì nên nghiên cứu vấn đề và cố gắng tìm ra nguyên nhân và cách khắc phục nó. Ngoài ra, cần cố gắng tìm ra cách giải quyết vấn đề, vì có thể mất 6 hoặc 9 tháng trước khi một người nào đó có thể được chỉ định thực hiện những thay đổi cần thiết đối với phần mềm.

Trong bối cảnh sự thiếu hụt nghiêm trọng của các lập trình viên và đặc biệt là các lập trình viên đủ giỏi để thực hiện bảo trì, đề xuất cách sống chung với lỗi cho đến khi nó có thể được giải quyết thường xuyên là cách duy nhất để đối phó với các báo cáo lỗi không phải là trường hợp khẩn cấp thực sự.

Kết luận của lập trình viên bảo trì phải được thêm vào tệp báo cáo lỗi, cùng với bất kỳ tài liệu hỗ trợ nào, chẳng hạn như danh sách, thiết kế và hướng dẫn sử dụng để đi đến kết luận đó. Người quản lý phụ trách bảo trì sau giao hàng phải thường xuyên tham khảo hồ sơ, đặt mức độ ưu tiên cho các bản sửa lỗi khác nhau. Tệp cũng phải chứa các yêu cầu của khách hàng về bảo trì thích ứng và hoàn hảo. Sau đó, sửa đổi tiếp theo được thực hiện đối với sản phẩm sẽ là sửa đổi có mức độ ưu tiên cao nhất.

Khi các bản sao của một sản phẩm đã được phân phối đến nhiều địa điểm khác nhau, các bản sao của báo cáo lỗi phải được lưu hành cho tất cả người dùng sản phẩm, cùng với ước tính về thời điểm có thể sửa từng lỗi. Sau đó, nếu lỗi tương tự xảy ra tại một trang web khác, người dùng có thể tham khảo báo cáo lỗi liên quan để xác định xem có thể khắc phục lỗi đó hay không và khi nào nó sẽ được khắc phục. Tất nhiên, sẽ tốt hơn nếu bạn sửa mọi lỗi ngay lập tức và phân phối phiên bản mới của sản phẩm cho tất cả các trang web. Với sự thiếu hụt các lập trình viên giỏi hiện nay trên toàn thế giới và thực tế của việc bảo trì phần mềm sau giao hàng, việc phân phát các báo cáo lỗi có lẽ là cách tốt nhất có thể làm được.

Có một lý do khác tại sao các khuyết tật thường không được sửa chữa ngay lập tức. Việc thực hiện một số thay đổi, kiểm tra tất cả, thay đổi tài liệu và cài đặt phiên bản mới hầu như luôn rẻ hơn so với thực hiện từng thay đổi riêng lẻ, kiểm tra, lập tài liệu, cài đặt phiên bản mới và sau đó lặp lại toàn bộ chu kỳ cho lần thay đổi tiếp theo. Điều này đặc biệt đúng nếu mọi phiên bản mới phải được cài đặt trên một số lượng đáng kể máy tính (chẳng hạn như một số lượng lớn máy khách trong mạng máy khách-máy chủ) hoặc khi phần mềm đang chạy ở các trang web khác nhau. Kết quả là, các tổ chức thích tập hợp các nhiệm vụ bảo trì phi mã, và sau đó thực hiện các thay đổi như một nhóm.

**16.5.2 Cho phép các thay đổi đối với sản phẩm**

Sau khi quyết định thực hiện bảo trì khắc phục, lập trình viên bảo trì được giao nhiệm vụ xác định lỗi gây ra hỏng hóc và sửa chữa lỗi đó. Sau khi mã đã được thay đổi, việc sửa chữa phải được kiểm tra, cũng như phải kiểm tra toàn bộ sản phẩm (kiểm tra hồi quy). Sau đó, tài liệu phải được cập nhật để phản ánh những thay đổi. Đặc biệt, một mô tả chi tiết về những gì đã được thay đổi, tại sao nó được thay đổi, bởi ai và khi nào phải được thêm vào các bình luận mở đầu của bất kỳ tạo tác mã đã thay đổi nào (Hình 15.1). Nếu cần, các tạo tác phân tích hoặc thiết kế cũng được thay đổi. Một loạt các bước tương tự được tuân theo khi thực hiện bảo trì hoàn thiện hoặc thích ứng; sự khác biệt thực sự duy nhất là bảo trì hoàn hảo và thích ứng được bắt đầu bởi một sự thay đổi trong các yêu cầu chứ không phải bởi một báo cáo khiếm khuyết.

Tại thời điểm này, tất cả những gì có vẻ cần thiết sẽ là phân phối phiên bản mới cho người dùng. Nhưng, điều gì sẽ xảy ra nếu lập trình viên bảo trì chưa kiểm tra sửa chữa đầy đủ? Trước khi sản phẩm được phân phối, nó phải được đảm bảo chất lượng phần mềm do một nhóm độc lập thực hiện; nghĩa là, các thành viên của nhóm SQA bảo trì không được báo cáo với người quản lý giống như người lập trình bảo trì. Điều quan trọng là nhóm SQA phải độc lập về mặt quản lý (Mục 6.1.2).

Các lý do trước đây đã được đưa ra là tại sao việc bảo trì sau giao hàng lại khó khăn. Vì những lý do tương tự, việc bảo trì cũng dễ xảy ra lỗi. Việc kiểm tra trong quá trình bảo trì sau giao hàng rất khó khăn và tốn thời gian, và nhóm SQA không nên đánh giá thấp tác động của việc bảo trì phần mềm đối với việc kiểm tra. Khi phiên bản mới đã được nhóm SQA chấp thuận, nó có thể được phân phối.

Một lĩnh vực khác trong đó quản lý phải đảm bảo rằng các thủ tục được tuân thủ cẩn thận là khi kỹ thuật đường cơ sở và bản sao riêng (Mục 5.10.2) được sử dụng. Giả sử một lập trình viên muốn thay đổi **Lớp cung cấp thuế(Tax Provision Class)**. Lập trình viên tạo ra các bản sao của **Lớp cung cấp thuế** và tất cả các tạo tác mã khác cần thiết để thực hiện nhiệm vụ bảo trì cần thiết; thường điều này bao gồm tất cả các lớp khác trong sản phẩm. Lập trình viên thực hiện các thay đổi cần thiết đối với **Lớp cung cấp thuế** và kiểm tra chúng. Bây giờ, phiên bản trước của **Lớp dự phòng thuế** đã bị đóng băng và phiên bản sửa đổi của **Lớp cung cấp thuế** kết hợp các thay đổi được cài đặt trong đường cơ sở. Nhưng, khi sản phẩm sửa đổi được giao cho người dùng, nó ngay lập tức gặp sự cố. Điều sai lầm là lập trình viên bảo trì đã thử nghiệm phiên bản sửa đổi của **Lớp cung cấp thuế** bằng cách sử dụng các bản sao không gian làm việc riêng của mình, nghĩa là các bản sao của các tạo tác mã khác nằm trong đường cơ sở tại thời điểm việc duy trì **Lớp cung cấp thuế** được bắt đầu. Trong khi đó, một số tạo tác mã khác đã được cập nhật bởi các lập trình viên bảo trì khác làm việc trên cùng một sản phẩm. Bài học rất rõ ràng: Trước khi cài đặt một tạo tác mã, nó phải được kiểm tra bằng cách sử dụng các phiên bản cơ bản hiện tại của tất cả các tạo tác mã khác chứ không phải các phiên bản riêng tư của lập trình viên. Đây là một lý do nữa để quy định một nhóm SQA độc lập - các thành viên của nhóm SQA chỉ đơn giản là không có quyền truy cập vào không gian làm việc riêng tư của các lập trình viên. Một lý do thứ ba là người ta ước tính rằng sự điều chỉnh ban đầu của một lỗi là chính nó không chính xác khoảng 70 phần trăm thời gian [Parnas, 1999].

**16.5.3 Đảm bảo khả năng bảo trì**

Bảo trì sau giao hàng không phải là nỗ lực một lần. Một sản phẩm được viết tốt trải qua một loạt các phiên bản trong suốt vòng đời của nó. Do đó, cần phải lập kế hoạch bảo trì sau giao hàng trong toàn bộ quy trình phần mềm. Trong quy trình thiết kế, ví dụ, nên sử dụng các kỹ thuật che giấu thông tin (Phần 7.6); trong quá trình thực hiện, các tên biến nên được chọn sẽ có ý nghĩa đối với các lập trình viên bảo trì trong tương lai (Phần 15.3). Tài liệu phải đầy đủ, chính xác và phản ánh phiên bản hiện tại của mọi tạo tác mã thành phần của sản phẩm.

Trong quá trình bảo trì sau giao hàng, điều quan trọng là không được làm ảnh hưởng đến khả năng bảo trì đã được tích hợp sẵn trong sản phẩm ngay từ đầu. Nói cách khác, cũng như nhân viên phát triển phần mềm luôn phải có ý thức về việc bảo trì sau giao hàng không thể tránh khỏi, do đó, nhân viên bảo trì phần mềm luôn phải có ý thức về việc bảo trì sau giao hàng trong tương lai không thể tránh khỏi. Các nguyên tắc được thiết lập để bảo trì trong quá trình phát triển áp dụng như nhau đối với bảo trì sau giao hàng.

**16.5.4 Vấn đề bảo trì lặp đi lặp lại**

Một trong những khó khăn khó chịu hơn của việc phát triển phần mềm là **vấn đề mục tiêu chuyển động (moving-target problem)** (Phần 2.4). Ngay khi nhà phát triển xây dựng sản phẩm, khách hàng có thể thay đổi các yêu cầu. Điều này không chỉ gây khó chịu cho nhóm phát triển, những thay đổi thường xuyên có thể dẫn đến một sản phẩm được xây dựng kém. Ngoài ra, những thay đổi như vậy làm tăng thêm giá thành của sản phẩm

Vấn đề trở nên trầm trọng hơn trong quá trình bảo trì sau giao hàng. Một sản phẩm đã hoàn thiện càng được thay đổi, nó càng đi chệch khỏi thiết kế ban đầu và càng khó thay đổi hơn. Trong điều kiện bảo trì lặp đi lặp lại, tài liệu có thể trở nên kém tin cậy hơn bình thường và các tệp kiểm tra hồi quy có thể không được cập nhật. Nếu vẫn phải bảo dưỡng nhiều hơn, trước tiên có thể phải thực hiện lại toàn bộ sản phẩm.

Vấn đề của mục tiêu di chuyển rõ ràng là một vấn đề về quản lý. Về lý thuyết, nếu ban quản lý đủ kiên quyết với khách hàng và giải thích vấn đề khi bắt đầu dự án, thì các yêu cầu có thể được giữ nguyên kể từ khi ký kết thông số kỹ thuật cho đến khi sản phẩm được giao. Một lần nữa, sau mỗi lần yêu cầu bảo trì hoàn hảo, các yêu cầu có thể bị đóng băng, chẳng hạn như 3 tháng hoặc 1 năm. Trong thực tế, nó không hoạt động theo cách đó. Ví dụ: nếu khách hàng là chủ tịch của tập đoàn và tổ chức phát triển là bộ phận phần mềm của tập đoàn đó, thì chủ tịch có thể ra lệnh thay đổi vào thứ Hai và thứ Năm hàng tuần và chúng sẽ được thực hiện. Câu tục ngữ cổ, "Ai trả tiền cho người đánh đàn gọi là giai điệu", thật không may, tất cả đều quá phù hợp trong tình huống này. Có lẽ, điều tốt nhất mà phó chủ tịch phụ trách phần mềm có thể làm là cố gắng giải thích cho chủ tịch ảnh hưởng của sản phẩm của việc bảo trì lặp đi lặp lại và sau đó chỉ cần thực hiện lại sản phẩm hoàn chỉnh bất cứ khi nào bảo trì thêm sẽ gây nguy hiểm cho tính toàn vẹn của sản phẩm. .

Cố gắng ngăn cản việc bảo trì bổ sung bằng cách đảm bảo rằng các thay đổi được yêu cầu được thực hiện chậm có thể có nghĩa là nhân viên có liên quan được thay thế bằng những người khác đã chuẩn bị để thực hiện công việc nhanh hơn. Tóm lại, nếu người yêu cầu thay đổi lặp đi lặp lại có đủ ảnh hưởng, không có giải pháp nào cho vấn đề của mục tiêu di chuyển.

**16.6 Bảo trì phần mềm hướng đối tượng**

Một lý do được đưa ra để sử dụng mô hình hướng đối tượng là nó thúc đẩy khả năng bảo trì. Rốt cuộc, một đối tượng là một đơn vị độc lập của một chương trình. Cụ thể hơn, một đối tượng được thiết kế tốt thể hiện sự độc lập về mặt khái niệm, hay còn được gọi là tính **đóng gói(encapsulation)** (Phần 7.4). Mọi khía cạnh của sản phẩm liên quan đến phần của thế giới thực được mô hình hóa bởi đối tượng đó đều được bản địa hóa cho chính đối tượng đó. Ngoài ra, các đối tượng thể hiện tính độc lập về vật chất; ẩn thông tin được sử dụng để đảm bảo rằng các chi tiết triển khai không thể nhìn thấy bên ngoài đối tượng đó (Phần 7.6). Hình thức giao tiếp duy nhất được phép là gửi một thông điệp đến đối tượng để gọi một phương thức cụ thể.

Kết quả là, lập luận đi, thật dễ dàng để duy trì một đối tượng vì hai lý do. Thứ nhất, tính độc lập về mặt khái niệm có nghĩa là dễ dàng xác định bộ phận nào của sản phẩm phải được thay đổi để đạt được mục tiêu bảo trì cụ thể, có thể là bảo trì nâng cao hoặc bảo dưỡng khắc phục. Thứ hai, ẩn thông tin đảm bảo rằng một thay đổi được thực hiện đối với một đối tượng không có tác động bên ngoài đối tượng đó, và do đó số lỗi hồi quy được giảm đáng kể.

Tuy nhiên, trên thực tế, tình hình không hoàn toàn bình thường như vậy. Trên thực tế, có ba trở ngại cụ thể đối với việc bảo trì phần mềm hướng đối tượng. Một trong những vấn đề có thể được giải quyết thông qua việc sử dụng các công cụ CASE thích hợp, nhưng những vấn đề khác thì khó xử lý hơn:

1. Hãy xem xét cấu trúc phân cấp lớp C được hiển thị trong Hình 16.1. Phương thức displayNode được định nghĩa trong **UndirectedTreeClass**, được kế thừa bởi **DirectedTreeClass**, và sau đó được xác định lại trong **RootedTreeClass**. Phiên bản được định nghĩa lại này được kế thừa bởi **BinaryTreeClass** và **BalancedBinaryTreeClass** và được sử dụng trong **BalancedBinaryTreeClass**. Do đó, một lập trình viên bảo trì phải nghiên cứu hệ thống phân cấp kế thừa hoàn chỉnh để hiểu **BalancedBinaryTreeClass**. Tệ hơn nữa, hệ thống phân cấp có thể không được hiển thị theo kiểu tuyến tính của Hình 16.1 nhưng thường được trải rộng trên toàn bộ sản phẩm. Vì vậy, để hiểu những gì displayNode làm trong **BalancedBinaryTreeClass**, lập trình viên bảo trì có thể phải nghiên cứu một phần chính của sản phẩm. Điều này khác xa với đối tượng "độc lập" được mô tả ở đầu phần này. Giải pháp cho vấn đề này rất đơn giản: sử dụng công cụ CASE thích hợp. Cũng giống như trình biên dịch C ++ có thể giải quyết chính xác phiên bản displayNode trong các trường hợp của lớp **BalancedBinaryTreeClass**, vì vậy một bàn làm việc lập trình có thể cung cấp một phiên bản "phẳng" của một lớp, nghĩa là định nghĩa của lớp với tất cả các tính năng được kế thừa trực tiếp hoặc gián tiếp xuất hiện rõ ràng, với bất kỳ đổi tên hoặc xác định lại nào được kết hợp. Dạng phẳng của **BalancedBinaryTreeClass** của Hình 16.1 bao gồm định nghĩa displayNode từ **RootedTreeClass**.

**class UndirectedTreeClass**

{

…

**void** displayNode (Node a);

…

}// **class UndirectedTreeClass**

**class DirectedTreeClass: public UndirectedTreeClass**

{

…

}// **class DirectedTreeClass**

**class RootedTreeClass: public DirectedTreeClass**

{

…

**void** displayNode (Node a);

…

}// **class RootedTreeClass**

**class BinaryTreeClass: public RootedTreeClass**

{

…

}// **class BinaryTreeClass**

**class BalancedBinaryTreeClass: public BinaryTreeClass**

{

Node hhh;

displayNode (hhh);

}// **class BalancedBinaryTreeClass**

HÌNH 16.1 C++ thực hiện hệ thống phân cấp lớp.

1. Một trở ngại khác đối với việc duy trì một sản phẩm được thực hiện bằng ngôn ngữ hướng đối tượng khó giải quyết hơn. Nó phát sinh do hậu quả của đa hình và liên kết động, các khái niệm được giải thích trong Phần 7.8. Một ví dụ đã được đưa ra trong phần đó, một lớp cơ sở có tên **File Class**, cùng với ba lớp con: **Disk File Class**, **Tape File Class** và **Diskette File Class**. Điều này được thể hiện trong Hình 7.33 (b), được sao chép ở đây để thuận tiện như Hình 16.2 . Trong lớp tệp lớp cơ sở, một phương pháp giả (**trừu tượng** hoặc **ảo**) được khai báo. Sau đó, một sự triển khai cụ thể của phương pháp xuất hiện trong mỗi trong ba lớp con; mỗi phương pháp được đặt tên giống hệt nhau, mở, như được hiển thị trong Hình 16.2. Giả sử myFile được tuyên bố là một đối tượng, một phiên bản của **Lớp Tệp** và mã được duy trì chứa thông báo myFile.open(). Do hậu quả của đa hình và ràng buộc động, tại thời điểm chạy, myFile có thể là thành viên của bất kỳ trong ba lớp có nguồn gốc của **Lớp tệp**, nghĩa là tệp đĩa, tệp băng hoặc tệp diskette. Một khi hệ thống thời gian chạy đã xác định lớp có nguồn gốc nào, phiên bản mở thích hợp được gọi. Điều này có thể có hậu quả bất lợi cho việc bảo trì. Nếu một lập trình viên bảo trì gặp phải cuộc gọi myFile.open() trong mã, sau đó, để hiểu rằng một phần của sản phẩm, anh ta hoặc cô ta phải xem xét điều gì sẽ xảy ra nếu myFile là một ví dụ của mỗi trong ba lớp con. Một công cụ CASE không thể giúp đỡ ở đây bởi vì, nói chung, không có cách nào để giải quyết các vấn đề ràng buộc động bằng cách sử dụng các phương pháp tĩnh. Cách duy nhất để xác định ràng buộc động nào thực sự xảy ra trong một tập hợp các trường hợp cụ thể là theo dõi thông qua mã, bằng cách chạy nó trên máy tính hoặc theo dõi thông qua nó theo cách thủ công. Đa hình và liên kết năng động thực sự là những khía cạnh cực kỳ mạnh mẽ của công nghệ hướng đối tượng thúc đẩy sự phát triển của một sản phẩm hướng đối tượng. Tuy nhiên, chúng có thể có tác động xấu đến việc bảo trì, bằng cách buộc lập trình viên bảo trì điều tra một loạt các ràng buộc có thể xảy ra tại thời điểm chạy và do đó xác định phương pháp nào trong số một số phương pháp khác nhau có thể được gọi tại thời điểm đó trong mã.

|  |
| --- |
| **File Class** |
| **Phương** pháp trừu tượng mở |

|  |
| --- |
| **Lớp Tệp Disktte** |
| Triển khai phương thức mở cho tệp diskette |

|  |
| --- |
| **Disk File Class** |
| Triển khai phương thức mở cho tệp đĩa |

|  |
| --- |
| **Lớp Tệp Băng** |
| Triển khai phương thức mở cho tệp băng |

Hình 16.2: định nghĩa của **Lớp tệp** lớp cơ sở với các lớp có nguồn gốc **Disk File Class**, **Tape File Class** và **Diskette File Class**.

1. Vấn đề cuối cùng phát sinh do hậu quả của **thừa kế**. Giả sử một lớp cơ sở cụ thể làm hầu hết, nhưng không phải tất cả, về những gì cần thiết cho việc thiết kế một sản phẩm mới. Một lớp có nguồn gốc bây giờ được định nghĩa, nghĩa là, một lớp giống hệt với lớp cơ sở theo nhiều cách, nhưng các tính năng mới có thể được thêm vào và các tính năng hiện có được đổi tên, tái cấu trúc, đàn áp hoặc thay đổi theo những cách khác. Hơn nữa, những thay đổi này có thể được thực hiện mà không ảnh hưởng đến lớp cơ sở hoặc bất kỳ lớp có nguồn gốc nào khác. Tuy nhiên, giả sử bây giờ rằng lớp cơ sở chính nó đã được thay đổi. Nếu điều này xảy ra, tất cả các lớp có nguồn gốc được thay đổi theo cùng một cách. Nói cách khác, sức mạnh của thừa kế là lá mới có thể được thêm vào cây thừa kế (hoặc đồ thị, nếu ngôn ngữ thực hiện hỗ trợ nhiều thừa kế, như C) mà không làm thay đổi bất kỳ lớp nào khác trong cây. Nhưng, nếu một nút bên trong của cây được thay đổi theo bất kỳ cách nào, thì sự thay đổi này được nhân rộng cho tất cả các hậu duệ của nó ( **vấn đề lớp cơ sở mong manh**).

Do đó, kế thừa là một tính năng khác của công nghệ hướng đối tượng có thể có ảnh hưởng tích cực lớn đến sự phát triển nhưng tác động tiêu cực đến việc bảo trì.