**Chương 2**

Mô hình vòng đời phần mềm

Mục tiêu học tập

Sau khi nghiên cứu chương này, bạn sẽ có thể

• Mô tả cách các sản phẩm phần mềm được phát triển trong thực tế.

• Hiểu mô hình vòng đời cây tiến hóa.

• Đánh giá cao tác động tiêu cực của sự thay đổi đối với các sản phẩm phần mềm.

• Sử dụng mô hình vòng đời lặp đi lặp lại và tăng dần.

• Hiểu tác động của Luật Miller đối với sản xuất phần mềm.

• Mô tả các điểm mạnh của mô hình vòng đời lặp đi lặp lại và tăng dần.

• Nhận thức sớm tầm quan trọng của việc giảm thiểu rủi ro.

• Mô tả các quy trình nhanh nhẹn, bao gồm cả lập trình cực đoan.

• So sánh và đối chiếu một loạt các mô hình vòng đời khác.

Chương 1 mô tả cách các sản phẩm phần mềm sẽ được phát triển trong một thế giới lý tưởng. Chủ đề của chương này là những gì xảy ra trong thực tế. Như sẽ được giải thích, có sự khác biệt lớn giữa lý thuyết và thực hành.

2.1 Phát triển phần mềm trong lý thuyết

Trong một thế giới lý tưởng, một sản phẩm phần mềm được phát triển như mô tả trong Chương 1. Như được mô tả bằng sơ đồ trong Hình 2.1, hệ thống được phát triển từ đầu; biểu thị tập hợp trống. (Xem Chỉ trong Trường hợp Bạn muốn Biết Hộp 2.1 nếu bạn muốn biết nguồn gốc của thuật ngữ từ đầu.) Trước tiên, Yêu cầu của khách hàng được xác định, sau đó thực hiện Phân tích. Khi các hiện vật phân tích hoàn tất, Bản thiết kế sẽ được tạo ra. Tiếp theo là việc Triển khai sản phẩm phần mềm hoàn chỉnh, sau đó được cài đặt trên máy tính của khách hàng.

Tuy nhiên, việc phát triển phần mềm trong thực tế khác nhau đáng kể vì hai lý do. Đầu tiên, các chuyên gia phần mềm là con người và do đó mắc sai lầm. Thứ hai, các yêu cầu của khách hàng có thể thay đổi trong khi phần mềm đang được phát triển. Trong chương này, cả hai vấn đề này đều được thảo luận sâu, nhưng trước tiên chúng tôi trình bày một nghiên cứu trường hợp nhỏ, dựa trên nghiên cứu điển hình trong [Tomer và Schach, 2000], minh họa các vấn đề liên quan. 37+38

thời điểm viết bài, dự án chậm tiến độ 6 tháng và vượt ngân sách 25%. Tuy nhiên, tất cả những người có liên quan đều hiểu rằng hệ thống mới sẽ đáng tin cậy hơn và có chất lượng cao hơn, mặc dù có “sự khác biệt nhỏ” trong việc đáp ứng các yêu cầu về thời gian phản hồi và độ chính xác.

Hình 2.2 mô tả mô hình vòng đời cây tiến hóa của nghiên cứu điển hình nhỏ. Các ô ngoài cùng bên trái đại diện cho Episode 1. Như được hiển thị trong sơ đồ, hệ thống được phát triển từ đầu (). Tiếp theo lần lượt là các yêu cầu (Yêu cầu 1), phân tích (Phân tích 1), thiết kế (Thiết kế 1) và thực hiện (Thực hiện 1). Tiếp theo, như đã mô tả trước đây, các thử nghiệm phiên bản đầu tiên của phần mềm cho thấy rằng không thể đạt được thời gian phản hồi trung bình 1 giây và việc triển khai phải được chỉnh sửa. Triển khai đã sửa đổi xuất hiện trong Hình 2.2 là Thực hiện 2. Tuy nhiên, Thực hiện 2 không bao giờ được hoàn thành. Đó là lý do tại sao hình chữ nhật đại diện cho Thực hiện 2 được vẽ bằng một đường chấm.

Trong tập 3, thiết kế đã được thay đổi. Đặc biệt, một thuật toán nhận dạng hình ảnh nhanh hơn đã được sử dụng. Thiết kế sửa đổi (Thiết kế 3) dẫn đến việc triển khai sửa đổi (Thực hiện 3).

Cuối cùng, trong Tập 4, các yêu cầu đã được thay đổi (Yêu cầu 4) để tăng độ chính xác. Điều này dẫn đến các cation specifi ed modifi (Phân tích 4), thiết kế sửa đổi (Design 4) và triển khai modifi ed (Thực hiện 4).

Trong hình 2.2, các mũi tên liền nét biểu thị sự phát triển và các mũi tên đứt nét biểu thị sự duy trì. Ví dụ, khi thiết kế được thay đổi trong Tập 3, Thiết kế 3 thay thế Thiết kế 1 là thiết kế của Phân tích 1.

Mô hình cây tiến hóa là một ví dụ về mô hình vòng đời (hay gọi tắt là mô hình), nghĩa là chuỗi các bước được thực hiện trong khi sản phẩm phần mềm được phát triển và duy trì. Một mô hình vòng đời khác có thể được sử dụng cho nghiên cứu trường hợp nhỏ là mô hình vòng đời thác nước [Royce, 1970]; một phiên bản đơn giản của mô hình thác nước được mô tả trong Hình 2.3. Mô hình vòng đời cổ điển này có thể được xem như mô hình tuyến tính của Hình 2.1 với các vòng phản hồi. Sau đó, nếu một lỗi được tìm thấy trong quá trình thiết kế gây ra bởi lỗi trong các yêu cầu, theo các mũi tên đi lên, các nhà phát triển phần mềm có thể quay ngược lại từ thiết kế cho đến khi phân tích và do đó các yêu cầu và thực hiện các chỉnh sửa cần thiết ở đó. Sau đó, họ chuyển xuống phần phân tích, chỉnh sửa tài liệu trích dẫn cụ thể để đưa ra các chỉnh sửa cho các yêu cầu, và lần lượt, chỉnh sửa tài liệu thiết kế. Các hoạt động thiết kế giờ đây có thể tiếp tục tại nơi chúng bị đình chỉ khi lỗi được phát hiện. Một lần nữa, các mũi tên rắn biểu thị sự phát triển; các mũi tên đứt nét, duy trì.

Mô hình thác nước chắc chắn có thể được sử dụng để đại diện cho nghiên cứu trường hợp nhỏ Winburg, nhưng, không giống như mô hình cây tiến hóa của Hình 2.2, nó không thể hiển thị thứ tự của các sự kiện. Mô hình cây tiến hóa có lợi thế hơn so với mô hình thác nước. Vào cuối mỗi tập, chúng ta có một đường cơ sở, tức là một tập hợp hoàn chỉnh các hiện vật (nhớ lại rằng một tạo tác là một thành phần cấu thành của một sản phẩm phần mềm). Có bốn đường cơ sở trong Hình 2.2. họ đang

Cuối Tập 1: Yêu cầu 1, Phân tích 1, Thiết kế 1, Thực hiện 1

Cuối Tập 2: Yêu cầu 1, Phân tích 1, Thiết kế 1, Thực hiện 2

Ở cuối Tập 3: Yêu cầu 1, Phân tích 1, Thiết kế 3, Thực hiện 3

Cuối Tập 4: Yêu cầu 4, Phân tích 4, Thiết kế 4, Thực hiện 4

Đường cơ sở đầu tiên là tập hợp hiện vật ban đầu; đường cơ sở thứ hai phản ánh việc thực hiện 2 của Tập 2 đã được sửa đổi (nhưng chưa bao giờ hoàn thành), cùng với các yêu cầu, phân tích và thiết kế không thay đổi của Tập 1. Đường cơ sở thứ ba giống với đường cơ sở đầu tiên nhưng thiết kế và thực hiện đã thay đổi. Đường cơ sở thứ tư là toàn bộ các hiện vật mới được thể hiện trong Hình 2.2. Chúng tôi xem xét lại khái niệm về đường cơ sở trong Chương 5 và 16. 40+41

2.3 Bài học của Nghiên cứu điển hình Winburg Mini

Nghiên cứu trường hợp nhỏ của Winburg mô tả sự phát triển của một sản phẩm phần mềm gặp trục trặc vì một số nguyên nhân không liên quan, chẳng hạn như chiến lược triển khai kém (việc sử dụng các số phân tích kép không cần thiết) và quyết định sử dụng một thuật toán quá chậm. Cuối cùng, dự án đã thành công tốt đẹp. Tuy nhiên, câu hỏi hiển nhiên là, trong thực tế phát triển phần mềm có thực sự hỗn loạn như vậy không? Trên thực tế, nghiên cứu trường hợp nhỏ ít gây tổn thương hơn nhiều, nếu không muốn nói là phần lớn, các dự án phần mềm. Trong nghiên cứu trường hợp nhỏ của Winburg, chỉ có hai phiên bản mới của phần mềm do lỗi (sử dụng số chính xác kép không phù hợp; sử dụng thuật toán không thể đáp ứng yêu cầu về thời gian phản hồi) và chỉ có một phiên bản mới do lỗi thay đổi do khách hàng thực hiện (cần tăng độ chính xác).

Tại sao lại cần nhiều thay đổi đối với một sản phẩm phần mềm? Đầu tiên, như đã nói trước đây, các chuyên gia phần mềm là con người và do đó mắc sai lầm. Thứ hai, một sản phẩm phần mềm là một mô hình của thế giới thực và thế giới thực luôn thay đổi. Vấn đề này được thảo luận nhiều hơn trong Phần 2.4

Nghiên cứu điển hình nhỏ:

Nghiên cứu điển hình về máy kéo màu teal

Teal Tractors, Inc., bán máy kéo ở hầu hết các khu vực của Hoa Kỳ. Công ty đã yêu cầu bộ phận phần mềm của mình phát triển một sản phẩm mới có thể xử lý tất cả các khía cạnh kinh doanh của mình. Ví dụ, sản phẩm phải có khả năng xử lý doanh số bán hàng, hàng tồn kho và tiền hoa hồng trả cho nhân viên bán hàng, cũng như cung cấp tất cả các chức năng kế toán cần thiết. Trong khi sản phẩm phần mềm này đang được triển khai, Máy kéo Teal mua một công ty máy kéo của Canada. Ban quản lý của Teal Tractors quyết định rằng, để tiết kiệm tiền, các hoạt động của Canada sẽ được tích hợp vào các hoạt động của Hoa Kỳ. Điều đó có nghĩa là phần mềm phải được thay đổi trước khi hoàn thành:

1. Nó phải được sửa đổi để xử lý các khu vực bán hàng bổ sung.

2. Nó phải được mở rộng để xử lý những khía cạnh của doanh nghiệp được xử lý khác nhau ở Canada, chẳng hạn như thuế.

3. Nó phải được mở rộng để xử lý hai loại tiền tệ khác nhau, đô la Mỹ và đô la Canada.

Máy kéo Teal là một công ty đang phát triển nhanh chóng với triển vọng tuyệt vời trong tương lai. Việc tiếp quản công ty máy kéo Canada là một bước phát triển tích cực, có thể dẫn đến lợi nhuận lớn hơn nữa trong những năm tới. Tuy nhiên, từ quan điểm của bộ phận phần mềm, việc mua lại công ty Canada có thể là một thảm họa. Trừ khi các yêu cầu, phân tích và thiết kế được thực hiện nhằm mục đích kết hợp các tiện ích mở rộng có thể có trong tương lai, công việc liên quan đến việc thêm các khu vực bán hàng của Canada có thể tuyệt vời đến mức có thể hiệu quả hơn nếu loại bỏ mọi thứ đã làm cho đến nay và bắt đầu lại từ đầu. Lý do là việc thay đổi sản phẩm ở giai đoạn này tương tự như việc cố gắng loại bỏ một sản phẩm phần mềm vào cuối vòng đời của nó (xem Hình 1.6). Việc mở rộng phần mềm để xử lý các khía cạnh cụ thể cho thị trường Canada, cũng như tiền tệ Canada, có thể khó không kém.

Ngay cả khi phần mềm đã được suy nghĩ kỹ lưỡng và thiết kế ban đầu thực sự có thể mở rộng, thiết kế của sản phẩm kết hợp được vá lại không thể gắn kết như nó đã có nếu nó được phát triển ngay từ đầu để phục vụ cho cả Hoa Kỳ. Hoa Kỳ và Canada. Điều này có thể có những tác động nghiêm trọng đến việc bảo trì trong tương lai.

Bộ phận phần mềm của Máy kéo Teal là nạn nhân của vấn đề mục tiêu thay đổi. Đó là, trong khi phần mềm đang được phát triển, các yêu cầu thay đổi. Không quan trọng là lý do thay đổi là vô cùng đáng giá. Thực tế là việc tiếp quản công ty Canada có thể gây bất lợi cho chất lượng của phần mềm đang được phát triển.

Trong một số trường hợp, lý do vì mục tiêu di chuyển ít lành tính hơn. Đôi khi, một nhà quản lý cấp cao quyền lực trong một tổ chức liên tục thay đổi ý kiến ​​của họ về chức năng của một sản phẩm phần mềm đang được phát triển. Trong các trường hợp khác, có sự gia tăng tính năng, một loạt các bổ sung nhỏ, gần như tầm thường, cho các yêu cầu. Nhưng bất kể lý do là gì, những thay đổi thường xuyên, bất kể chúng có vẻ nhỏ đến mức nào, đều có hại cho sức khỏe của một sản phẩm phần mềm. Điều quan trọng là một sản phẩm phần mềm phải được thiết kế như một tập hợp các thành phần độc lập nhất có thể, để việc thay đổi một phần của phần mềm không gây ra lỗi trong một phần dường như không liên quan của mã, một cái gọi là hồi quy lỗi . Khi nhiều thay đổi được thực hiện, hiệu quả là tạo ra sự phụ thuộc trong mã. Cuối cùng, có rất nhiều sự phụ thuộc mà hầu như bất kỳ thay đổi nào đều gây ra một hoặc nhiều lỗi hồi quy. Khi đó, điều duy nhất có thể làm là thiết kế lại toàn bộ sản phẩm phần mềm và thực hiện lại nó.

Thật không may, không có giải pháp nào được biết đến cho vấn đề mục tiêu di chuyển. Liên quan đến những thay đổi tích cực đối với các yêu cầu, các công ty đang phát triển sẽ luôn thay đổi và những thay đổi này phải được hoàn thiện trong các sản phẩm phần mềm quan trọng của công ty. Đối với những thay đổi tiêu cực, nếu cá nhân kêu gọi những thay đổi đó có đủ ảnh hưởng, không thể làm gì để ngăn những thay đổi được thực hiện, gây hại cho khả năng bảo trì thêm của sản phẩm phần mềm.

2.5 Lặp lại và Tăng dần

Do cả vấn đề mục tiêu di chuyển và nhu cầu sửa chữa những sai lầm không thể tránh khỏi trong khi phát triển một sản phẩm phần mềm, vòng đời của các sản phẩm phần mềm thực tế giống như mô hình cây tiến hóa của Hình 2.2 hoặc mô hình thác nước của Hình 2.3 , chứ không phải là chuỗi lý tưởng của Hình 2.1. Một hệ quả của thực tế này là không có nhiều ý nghĩa khi nói về (giả sử) “giai đoạn phân tích”. Thay vào đó, các hoạt động của giai đoạn phân tích được dàn trải trong vòng đời. Tương tự, Hình 2.2 cho thấy bốn phiên bản khác nhau của việc triển khai, một trong số đó (Phiên bản 2) đã không bao giờ được hoàn thành vì vấn đề mục tiêu di chuyển.

Xem xét các phiên bản kế tiếp của một cấu phần, ví dụ, tài liệu cation đặc biệt hoặc một mô-đun mã. Từ quan điểm này, quá trình cơ bản là lặp đi lặp lại. Đó là, chúng tôi sản xuất phiên bản đầu tiên của hiện vật, sau đó chúng tôi sửa đổi nó và tạo ra phiên bản thứ hai, v.v. Mục đích của chúng tôi là mỗi phiên bản đều gần với mục tiêu của chúng tôi hơn phiên bản tiền nhiệm và cuối cùng chúng tôi tạo ra một phiên bản ưng ý. Lặp lại là một khía cạnh nội tại của kỹ thuật phần mềm và các mô hình vòng đời lặp đi lặp lại đã được sử dụng trong hơn 30 năm [Larman và Basili, 2003]. Ví dụ, mô hình thác nước, được đưa ra lần đầu tiên vào năm 1970, là mô hình lặp lại (nhưng không tăng dần). 42+43

Khía cạnh thứ hai của việc phát triển phần mềm trong thế giới thực là hạn chế mà Luật Miller áp đặt cho chúng tôi. Năm 1956, George Miller, một giáo sư tâm lý học, đã chỉ ra rằng, tại bất kỳ thời điểm nào, con người chúng ta chỉ có khả năng tập trung vào khoảng bảy khối (đơn vị thông tin) [Miller, 1956]. Tuy nhiên, một tạo tác phần mềm điển hình có hơn bảy phần. Ví dụ, một tạo tác mã có thể có nhiều hơn đáng kể bảy biến và một tài liệu yêu cầu có thể có nhiều hơn bảy yêu cầu. Một cách mà con người chúng ta xử lý hạn chế này về số lượng thông tin mà chúng ta có thể xử lý bất kỳ lúc nào là sử dụng tinh chỉnh theo từng bước. Đó là, chúng tôi tập trung vào những khía cạnh hiện đang là quan trọng nhất và hoãn lại cho đến sau những khía cạnh hiện ít quan trọng hơn. Nói cách khác, mọi khía cạnh cuối cùng cũng được xử lý nhưng theo thứ tự quan trọng hiện tại. Điều này có nghĩa là chúng ta bắt đầu bằng cách xây dựng một tạo tác chỉ giải quyết một phần nhỏ những gì chúng ta đang cố gắng đạt được. Sau đó, chúng tôi xem xét các khía cạnh khác của vấn đề và thêm các mảnh mới vào hiện vật hiện có. Ví dụ: chúng tôi có thể xây dựng một tài liệu yêu cầu bằng cách xem xét bảy yêu cầu mà chúng tôi cho là quan trọng nhất. Sau đó, chúng tôi sẽ xem xét bảy yêu cầu quan trọng nhất tiếp theo, v.v. Đây là một quá trình gia tăng. Sự gia tăng cũng là một khía cạnh nội tại của kỹ thuật phần mềm; phát triển phần mềm gia tăng đã hơn 45 tuổi [Larman và Basili, 2003].

Trong thực tế, lặp lại và tăng dần được sử dụng kết hợp với nhau. Có nghĩa là, một tạo tác được xây dựng từng phần (tăng dần), và mỗi phần gia tăng trải qua nhiều phiên bản (lặp lại). Những ý tưởng này được minh họa trong Hình 2.2, đại diện cho vòng đời của nghiên cứu trường hợp nhỏ Winburg (Phần 2.2 và 2.3). Như được hiển thị trong thiết bị fi đó, không có "giai đoạn yêu cầu" duy nhất như vậy. Thay vào đó, các yêu cầu của khách hàng được trích xuất và phân tích hai lần, tạo ra các yêu cầu ban đầu (Yêu cầu 1) và các yêu cầu sửa đổi (Yêu cầu 4). Tương tự, không có “giai đoạn triển khai” duy nhất, mà là bốn giai đoạn riêng biệt, trong đó mã được tạo và sau đó sửa đổi.

Những ý tưởng này được khái quát trong Hình 2.4, phản ánh các khái niệm cơ bản nằm trong mô hình vòng đời lặp đi lặp lại và tăng dần [Jacobson, Booch, và Rumbaugh, 1999]. Hình vẽ cho thấy sự phát triển của một sản phẩm phần mềm theo bốn mức tăng, được gắn nhãn Mức tăng A, Mức tăng B, Mức tăng C và Mức tăng D. Trục hoành là thời gian và trục tung là giờ người (một người-giờ là số lượng công việc mà một người có thể làm trong 1 giờ), vì vậy vùng tô bóng dưới mỗi đường cong là tổng nỗ lực cho phần gia tăng đó.

Điều quan trọng cần đánh giá là Hình 2.4 mô tả chỉ một cách khả thi mà một sản phẩm phần mềm có thể được phân rã thành từng phần. Một sản phẩm phần mềm khác có thể được xây dựng chỉ với 2 bước, trong khi một phần ba có thể yêu cầu 14. Hơn nữa, lược đồ không nhằm thể hiện chính xác cách một sản phẩm phần mềm được phát triển. Thay vào đó, nó cho thấy cách nhấn mạnh thay đổi từ lặp sang lặp.

Các giai đoạn tuần tự của Hình 2.1 là các cấu trúc nhân tạo. Thay vào đó, như được phản ánh rõ ràng trong Hình 2.4, chúng ta phải thừa nhận rằng các luồng công việc (hoạt động) khác nhau được thực hiện trong toàn bộ vòng đời. Có năm luồng công việc cốt lõi, quy trình công việc yêu cầu, quy trình phân tích, quy trình thiết kế, quy trình thực hiện và quy trình kiểm tra, và như đã nêu trong câu trước, tất cả năm quy trình này đều được thực hiện trong vòng đời của một sản phẩm phần mềm. Tuy nhiên, đôi khi một quy trình làm việc chiếm ưu thế hơn bốn quy trình còn lại. 44

Ví dụ, vào đầu vòng đời, các nhà phát triển phần mềm trích xuất một bộ yêu cầu ban đầu. Nói cách khác, khi bắt đầu vòng đời lặp đi lặp lại và tăng dần, quy trình công việc yêu cầu chiếm ưu thế. Những yêu cầu tạo tác này được mở rộng và sửa đổi trong phần còn lại của vòng đời. Trong thời gian đó, bốn quy trình công việc khác (phân tích, thiết kế, triển khai và kiểm tra) chiếm ưu thế. Nói cách khác, dòng công việc yêu cầu là dòng công việc chính ở đầu chu kỳ sống, nhưng tầm quan trọng tương đối của nó giảm dần sau đó. Ngược lại, quy trình thực hiện và kiểm tra công việc chiếm nhiều thời gian của các thành viên trong nhóm phát triển phần mềm vào cuối vòng đời hơn so với lúc đầu.

Các hoạt động lập kế hoạch và tài liệu được thực hiện trong suốt vòng đời lặp đi lặp lại và gia tăng. Hơn nữa, kiểm tra là một hoạt động chính trong mỗi lần lặp, và đặc biệt là vào cuối mỗi lần lặp. Ngoài ra, toàn bộ phần mềm được kiểm tra kỹ lưỡng sau khi hoàn thành; vào thời điểm đó, việc kiểm tra và sau đó sửa đổi việc triển khai dựa trên kết quả của các bài kiểm tra khác nhau hầu như là hoạt động duy nhất của nhóm phần mềm. Điều này được phản ánh trong quy trình làm việc thử nghiệm của Hình 2.4.

Hình 2.4 cho thấy bốn giá trị gia tăng. Hãy xem xét Phần tăng A, được mô tả bằng cột bên trái. Khi bắt đầu bước tăng này, các thành viên trong nhóm yêu cầu xác định yêu cầu của khách hàng. Khi hầu hết các yêu cầu đã được xác định, phiên bản đầu tiên của phần phân tích có thể được bắt đầu. Khi đã đạt đủ tiến độ với việc phân tích, phiên bản đầu tiên của thiết kế có thể được bắt đầu. Thậm chí, một số mã hóa thường được thực hiện trong lần tăng đầu tiên này, có lẽ dưới dạng một nguyên mẫu thử nghiệm khái niệm để kiểm tra tính khả thi của một phần sản phẩm phần mềm được đề xuất. Cuối cùng, như đã đề cập trước đó, các hoạt động lập kế hoạch, thử nghiệm và tài liệu bắt đầu từ Ngày thứ nhất và tiếp tục từ đó trở đi, cho đến khi sản phẩm phần mềm cuối cùng được giao cho khách hàng.

Tương tự, sự tập trung chính trong Phần tăng trưởng B là các yêu cầu và quy trình phân tích, sau đó là quy trình thiết kế. Sự nhấn mạnh trong Phần tăng trưởng C trước tiên là quy trình thiết kế, sau đó là quy trình thực hiện và quy trình công việc kiểm tra. Cuối cùng, trong Phần tăng trưởng D, quy trình thực hiện và quy trình kiểm tra chiếm ưu thế.

Như được phản ánh trong Hình 1.4, khoảng một phần năm tổng số nỗ lực được dành cho các yêu cầu và quy trình phân tích (cùng nhau), một phần năm khác cho quy trình thiết kế và khoảng ba phần năm cho quy trình thực hiện. Tổng kích thước tương đối của các vùng bóng mờ trong Hình 2.4 sẽ loại bỏ các giá trị này.

Có sự lặp lại trong mỗi lần tăng của Hình 2.4. Điều này được thể hiện trong Hình 2.5, mô tả ba lần lặp trong quá trình Tăng dần B. (Hình 2.5 là hình ảnh phóng to của cột thứ hai của Hình 2.4.) Như trong Hình 2.5, mỗi lần lặp bao gồm tất cả năm quy trình công việc nhưng một lần nữa với tỷ lệ khác nhau.

Một lần nữa, cần phải nhấn mạnh rằng Hình 2.5 không nhằm chỉ ra rằng mọi gia số liên quan đến chính xác ba lần lặp. Số lần lặp thay đổi theo từng mức tăng dần. Mục đích của Hình 2.5 là cho thấy sự lặp lại trong mỗi bước tăng và lặp lại rằng tất cả năm quy trình công việc (yêu cầu, phân tích, thiết kế, thực hiện và thử nghiệm, cùng với lập kế hoạch và tài liệu) được thực hiện trong hầu hết mọi lần lặp, mặc dù mỗi quy trình với tỷ lệ khác nhau thời gian.

Như đã giải thích trước đây, Hình 2.4 phản ánh sự gia tăng nội tại của sự phát triển của mọi sản phẩm phần mềm. Hình 2.5 hiển thị rõ ràng sự lặp lại làm cơ sở cho sự gia tăng. Cụ thể, Hình 2.5 mô tả ba bước lặp lại liên tiếp, trái ngược với một bước tăng lớn. Chi tiết hơn, Lặp lại B.1 bao gồm các quy trình công việc yêu cầu, phân tích, thiết kế, triển khai và thử nghiệm, được thể hiện bằng hình chữ nhật đứt nét ngoài cùng bên trái với các góc tròn. Việc lặp lại tiếp tục cho đến khi các thành phần của mỗi quy trình trong số năm quy trình làm việc đạt yêu cầu.

Tiếp theo, tất cả năm bộ tạo tác được lặp lại trong Lặp lại B.2. Lần lặp thứ hai này có bản chất tương tự như lần đầu tiên. Nghĩa là, các tạo tác yêu cầu được cải thiện, từ đó kích hoạt các cải tiến đối với các tạo tác phân tích, v.v., như được phản ánh trong lần lặp thứ hai của Hình 2.5 và tương tự đối với lần lặp thứ ba.

Quá trình lặp lại và tăng dần bắt đầu từ đầu Phần tăng A và tiếp tục cho đến khi kết thúc Phần tăng D. Sau đó, sản phẩm phần mềm hoàn chỉnh sẽ được cài đặt trên máy tính của khách hàng. 45+46

Winburg Mini Case Study đã được duyệt lại

Hình 2.6 cho thấy mô hình cây tiến hóa của nghiên cứu trường hợp nhỏ Winburg (Hình 2.2) được xếp chồng lên mô hình lặp lại và tăng dần (quy trình công việc thử nghiệm không được hiển thị vì mô hình cây tiến hóa giả định thử nghiệm liên tục, giải thích trong Phần 1.7) . Hình 2.6 làm sáng tỏ thêm về bản chất của sự gia tăng:

• Phần tăng A tương ứng với Tập 1, Phần tăng B tương ứng với Tập 2, v.v.

• Từ quan điểm của mô hình lặp lại và tăng dần, hai trong số các bước tăng thêm không bao gồm tất cả bốn quy trình công việc. Chi tiết hơn, Phần tăng thêm B (Phần 2) chỉ bao gồm quy trình thực hiện và Phần tăng C (Phần 3) chỉ bao gồm quy trình công việc thiết kế và quy trình thực hiện. Mô hình lặp lại và tăng dần không yêu cầu mọi quy trình làm việc phải được thực hiện trong mỗi lần tăng.

• Hơn nữa, trong Hình 2.4, hầu hết quy trình công việc yêu cầu được thực hiện trong Phần tăng dần A và Phần tăng dần B, trong khi trong Hình 2.6 nó được thực hiện trong Phần tăng dần A và Phần tăng dần D. Ngoài ra, trong Hình 2.4 hầu hết các phân tích được thực hiện trong Phần tăng dần B, ngược lại trong Hình 2.6, quy trình phân tích được thực hiện trong Phần tăng dần A và Phần tăng dần D. Điều này cho thấy rằng cả Hình 2.4 và Hình 2.6 đều không đại diện cho cách mọi sản phẩm phần mềm được xây dựng. Thay vào đó, mỗi hình cho thấy cách mà một sản phẩm phần mềm cụ thể được xây dựng, làm nổi bật sự lặp lại và gia tăng cơ bản.

• Quy mô nhỏ và sự chấm dứt đột ngột của quy trình thực hiện trong Phần tăng thêm B (Tập 2) của Hình 2.6 cho thấy rằng Thực hiện 2 đã không được hoàn thành. Phần màu xám phản ánh một phần của quy trình triển khai không được thực hiện.

• Ba mũi tên đứt nét của mô hình cây tiến hóa cho thấy rằng mỗi phần tăng thêm cấu thành sự duy trì của phần tăng trước đó. Trong ví dụ này, gia số thứ hai và thứ ba là các trường hợp bảo dưỡng khắc phục. Có nghĩa là, mỗi gia số sửa chữa các lỗi trong gia số trước đó. Như đã giải thích trước đó, Phần tăng B (Tập 2) sửa quy trình thực hiện bằng cách thay thế các biến độ chính xác kép bằng các biến độ chính xác đơn thông thường. Phần tăng C (Tập 3) điều chỉnh quy trình thiết kế bằng cách sử dụng thuật toán nhận dạng hình ảnh nhanh hơn, do đó cho phép đáp ứng yêu cầu về thời gian phản hồi. Các thay đổi tương ứng sau đó phải được thực hiện đối với quy trình thực hiện. Cuối cùng, trong Phần tăng dần D (Tập 4), các yêu cầu được thay đổi để quy định độ chính xác tổng thể được cải thiện, một ví dụ của bảo trì hoàn hảo. Sau đó, các thay đổi tương ứng được thực hiện đối với quy trình phân tích, quy trình thiết kế và quy trình thực hiện. 47+48

2.7 Rủi ro và các khía cạnh khác của việc lặp lại và gia tăng

Một cách khác để xem xét sự lặp lại và sự gia tăng là toàn bộ dự án được chia thành các dự án nhỏ nhỏ hơn (hoặc phần tăng dần). Mỗi dự án nhỏ mở rộng các yêu cầu, phân tích, thiết kế, thực hiện và thử nghiệm các hiện vật. Cuối cùng, tập hợp các hiện vật tạo thành sản phẩm phần mềm hoàn chỉnh.

Trên thực tế, mỗi dự án nhỏ không chỉ bao gồm mở rộng các hiện vật. Điều cần thiết là phải kiểm tra xem mỗi hiện vật có đúng không (quy trình kiểm tra) và thực hiện bất kỳ thay đổi cần thiết nào đối với các hiện vật có liên quan. Quá trình kiểm tra và sửa đổi, sau đó kiểm tra lại và sửa đổi, v.v., rõ ràng là có tính chất lặp lại. Nó tiếp tục cho đến khi các thành viên của nhóm phát triển hài lòng với tất cả các hiện vật của dự án nhỏ hiện tại (hoặc phần tăng thêm). Khi điều đó xảy ra, họ tiến hành bước tăng tiếp theo.

So sánh Hình 2.3 (mô hình thác nước) với Hình 2.5 (chế độ xem các lần lặp trong Phần tăng dần B) cho thấy rằng mỗi lần lặp có thể được xem như một mô hình thác nước nhỏ nhưng hoàn chỉnh. Có nghĩa là, trong mỗi lần lặp lại, các thành viên của nhóm phát triển trải qua các giai đoạn yêu cầu, phân tích, thiết kế và triển khai cổ điển trên một phần cụ thể của sản phẩm phần mềm. Từ quan điểm này, mô hình lặp lại và tăng dần của Hình 2.4 và 2.5 có thể được xem như một chuỗi các mô hình thác nước liên tiếp.

Mô hình lặp lại và tăng dần có nhiều điểm mạnh:

1. Nhiều cơ hội được cung cấp để kiểm tra xem sản phẩm phần mềm có đúng không. Mỗi lần lặp đều kết hợp quy trình làm việc thử nghiệm, vì vậy mỗi lần lặp lại là một cơ hội khác để kiểm tra tất cả các tạo tác được phát triển cho đến thời điểm này. Các lỗi được phát hiện và sửa chữa muộn hơn thì chi phí sẽ cao hơn, như trong Hình 1.6. Không giống như mô hình thác nước cổ điển, mỗi trong số nhiều lần lặp lại của mô hình lặp đi lặp lại và tăng dần đều mang lại cơ hội tiếp tục để tìm ra các lỗi và sửa chúng, do đó tiết kiệm tiền.

2. Độ bền của kiến ​​trúc bên dưới có thể được xác định tương đối sớm trong vòng đời. Kiến trúc của một sản phẩm phần mềm bao gồm các thành phần tạo tác khác nhau và cách chúng kết hợp với nhau. Một phép tương tự là kiến ​​trúc của một nhà thờ, có thể được mô tả là theo kiểu Romanesque, Gothic hoặc Baroque, trong số các khả năng khác. Tương tự, kiến ​​trúc của một sản phẩm phần mềm có thể được mô tả là hướng đối tượng (Chương 7), đường ống và bộ lọc (các thành phần UNIX hoặc Linux) hoặc máy khách-máy chủ (với máy chủ trung tâm cung cấp lưu trữ tệp cho mạng máy tính khách). Kiến trúc của một sản phẩm phần mềm được phát triển bằng cách sử dụng mô hình lặp và tăng dần phải có thuộc tính mà nó có thể được mở rộng liên tục (và, nếu cần, có thể dễ dàng thay đổi) để kết hợp các bước tiếp theo. Khả năng xử lý các phần mở rộng và thay đổi như vậy mà không bị phá vỡ được gọi là tính mạnh mẽ. Mạnh mẽ là một chất lượng quan trọng trong quá trình phát triển một sản phẩm phần mềm; nó rất quan trọng trong quá trình bảo trì sau giao hàng. Vì vậy, nếu một sản phẩm phần mềm phải tồn tại qua 12, 15 hoặc nhiều năm bảo trì sau phân phối thông thường, thì kiến ​​trúc cơ bản phải mạnh mẽ. Khi một mô hình lặp lại và tăng dần được sử dụng, nó sẽ sớm trở nên rõ ràng liệu kiến ​​trúc có mạnh mẽ hay không. Nếu, trong quá trình kết hợp (ví dụ) bước thứ ba, rõ ràng là phần mềm được phát triển cho đến nay phải được tổ chức lại mạnh mẽ và các phần lớn được thực hiện lại, thì rõ ràng là kiến ​​trúc không đủ mạnh. Khách hàng phải quyết định bỏ dự án hay bắt đầu lại từ đầu. Một khả năng khác là thiết kế lại kiến ​​trúc để mạnh mẽ hơn và sau đó tái sử dụng càng nhiều đồ tạo tác hiện tại càng tốt trước khi tiếp tục bước tiếp theo. Một lý do khác tại sao một kiến ​​trúc mạnh mẽ lại quan trọng như vậy là vấn đề mục tiêu di chuyển (Phần 2.4). Tất cả nhưng chắc chắn rằng các yêu cầu của khách hàng sẽ thay đổi, do sự phát triển trong tổ chức của khách hàng hoặc vì khách hàng liên tục thay đổi ý định về những gì phần mềm mục tiêu phải làm. Kiến trúc càng mạnh thì phần mềm càng có khả năng thay đổi được. Không thể thiết kế một công trình kiến ​​trúc có thể đương đầu với quá nhiều thay đổi mạnh mẽ. Nhưng, nếu những thay đổi được yêu cầu là hợp lý trong phạm vi, thì một kiến ​​trúc mạnh mẽ sẽ có khả năng kết hợp những thay đổi đó mà không cần phải cấu trúc lại. 48+49

3. Mô hình lặp đi lặp lại và gia tăng cho phép chúng ta giảm thiểu rủi ro sớm. Rủi ro luôn liên quan đến việc phát triển và bảo trì phần mềm. Ví dụ, trong nghiên cứu trường hợp nhỏ của Winburg, thuật toán nhận dạng hình ảnh ban đầu không đủ nhanh; Có một rủi ro luôn hiện hữu là một sản phẩm phần mềm đã hoàn thành sẽ không đáp ứng được các hạn chế về thời gian. Việc phát triển từng bước một sản phẩm phần mềm cho phép chúng tôi giảm thiểu những rủi ro như vậy sớm trong vòng đời. Ví dụ: giả sử một mạng cục bộ (LAN) mới đang được phát triển và có lo ngại rằng phần cứng mạng hiện tại không đủ cho sản phẩm phần mềm mới. Sau đó, một hoặc hai lần lặp lại đầu tiên được hướng tới việc xây dựng những phần của phần mềm giao diện với phần cứng mạng. Nếu hóa ra, trái ngược với lo ngại của các nhà phát triển, mạng có khả năng cần thiết, các nhà phát triển có thể tiến hành dự án, chắc chắn rằng rủi ro này đã được giảm thiểu. Mặt khác, nếu mạng thực sự không thể đối phó với lượng truy cập bổ sung mà mạng LAN mới tạo ra, điều này sẽ được báo cáo cho khách hàng sớm trong vòng đời, khi chỉ một phần nhỏ ngân sách đã được chi tiêu. Giờ đây, khách hàng có thể quyết định hủy dự án, mở rộng khả năng của mạng hiện có, mua một mạng mới và mạnh hơn hoặc thực hiện một số hành động khác.

4. Chúng tôi luôn có một phiên bản hoạt động của phần mềm. Giả sử một sản phẩm phần mềm được phát triển bằng cách sử dụng mô hình vòng đời cổ điển của Hình 2.1. Chỉ khi kết thúc dự án mới có phiên bản hoạt động của sản phẩm phần mềm. Ngược lại, khi mô hình vòng đời lặp đi lặp lại và gia tăng được sử dụng, ở cuối mỗi lần lặp lại, có một phiên bản hoạt động của một phần sản phẩm phần mềm mục tiêu tổng thể. Khách hàng và những người dùng dự định có thể thử nghiệm với phiên bản đó và xác định những thay đổi nào cần thiết để đảm bảo rằng việc triển khai hoàn chỉnh trong tương lai đáp ứng nhu cầu của họ. Những thay đổi này có thể được thực hiện với mức tăng tiếp theo và sau đó khách hàng và người dùng có thể xác định xem có cần thay đổi thêm hay không. Một biến thể của điều này là cung cấp các phiên bản một phần của sản phẩm phần mềm, không chỉ để thử nghiệm mà còn để giới thiệu sản phẩm phần mềm mới một cách suôn sẻ trong tổ chức khách hàng. Thay đổi hầu như luôn được coi là một mối đe dọa. Thông thường, người dùng lo sợ rằng việc giới thiệu một sản phẩm phần mềm mới tại nơi làm việc sẽ khiến họ mất việc với máy tính. Tuy nhiên, việc giới thiệu một sản phẩm phần mềm dần dần có thể có hai lợi ích. Đầu tiên, nỗi sợ hãi dễ hiểu về việc bị thay thế bởi một máy tính được giảm bớt. Thứ hai, nhìn chung sẽ dễ dàng hơn để tìm hiểu chức năng của một sản phẩm phần mềm phức tạp nếu chức năng đó được giới thiệu từng bước trong khoảng thời gian hàng tháng, thay vì toàn bộ.

5. Có bằng chứng thực nghiệm cho thấy chu kỳ sống lặp đi lặp lại và gia tăng hoạt động. Biểu đồ tròn của Hình 1.1 cho thấy kết quả của báo cáo từ Standish Group về các dự án đã hoàn thành trong năm 2006 [Rubenstein, 2007]. Trên thực tế, báo cáo này (được gọi là Báo cáo CHAOS— xem Chỉ trong trường hợp bạn muốn biết ở Hộp 2.2) được lập 2 năm một lần. Hình 2.7 cho thấy kết quả từ năm 1994 đến năm 2006. Tỷ lệ sản phẩm thành công tăng đều đặn từ 16 phần trăm năm 1994 lên 34 phần trăm năm 2002, nhưng sau đó giảm xuống 29 phần trăm năm 2004. Trong cả năm 2002 [Softwaremag.com, 2004] và 2004 [Hayes, 2004] báo cáo, một trong những yếu tố liên quan đến các dự án thành công là việc sử dụng một quy trình lặp đi lặp lại. (Những lý do được đưa ra cho việc giảm tỷ lệ các dự án thành công trong năm 2004 bao gồm: nhiều dự án lớn hơn năm 2002, sử dụng mô hình thác nước, thiếu sự tham gia của người dùng và thiếu sự hỗ trợ từ các giám đốc điều hành cấp cao [Hayes, 2004].) , tỷ lệ các dự án thành công đã tăng trở lại trong nghiên cứu năm 2006 lên 35 phần trăm. Chủ tịch của Standish Group, Jim Johnson, cho rằng sự gia tăng này là do ba yếu tố: quản lý dự án tốt hơn, cơ sở hạ tầng Web mới nổi và (một lần nữa) phát triển lặp đi lặp lại [Rubenstein, 2007]. 50

Chỉ trong trường hợp bạn muốn biết

Thuật ngữ CHAOS là một từ viết tắt. Vì một số lý do không xác định, Tập đoàn Standish giữ bí mật hàng đầu của từ viết tắt này. Họ nêu [Standish, 2003]:

Chỉ có một số người tại The Standish Group và bất kỳ ai trong số 360 người đã nhận và lưu những chiếc áo phông mà chúng tôi đã phát sau khi họ hoàn thành cuộc khảo sát đầu tiên vào năm 1994, biết chữ CHAOS đại diện cho điều gì.

2.8 Quản lý sự lặp lại và gia tăng

Thoạt nhìn, mô hình lặp đi lặp lại và tăng dần của Hình 2.4 và 2.5 trông hoàn toàn hỗn loạn. Thay vì tiến trình có trật tự từ các yêu cầu đến việc triển khai mô hình thác nước (Hình 2.3), có vẻ như các nhà phát triển làm bất cứ điều gì họ thích, có thể là viết mã vào buổi sáng, một hoặc hai giờ thiết kế sau bữa trưa và sau đó nửa giờ xác định trước khi về nhà. Đó không phải là tình huống. Ngược lại, mô hình lặp lại và tăng dần cũng giống như mô hình thác nước, bởi vì như đã chỉ ra trước đây, phát triển một sản phẩm phần mềm sử dụng mô hình lặp lại và tăng dần không hơn không kém so với phát triển một loạt các sản phẩm phần mềm nhỏ hơn , tất cả đều sử dụng mô hình thác nước.

Chi tiết hơn, như trong Hình 2.3, phát triển một sản phẩm phần mềm sử dụng mô hình thác nước có nghĩa là thực hiện liên tiếp các yêu cầu, phân tích, thiết kế và giai đoạn thực hiện (theo thứ tự đó) trên toàn bộ sản phẩm phần mềm. Nếu gặp sự cố, các vòng phản hồi của Hình 2.3 (mũi tên đứt nét) được tuân theo; nghĩa là, lặp đi lặp lại (bảo trì) được thực hiện. Tuy nhiên, nếu cùng một sản phẩm phần mềm được phát triển bằng mô hình lặp đi lặp lại và tăng dần, thì sản phẩm phần mềm đó được coi là một tập hợp các giá trị gia tăng. Đối với mỗi bước tăng lần lượt, các giai đoạn yêu cầu, phân tích, thiết kế và thực hiện (theo thứ tự đó) được thực hiện lặp lại trên mức tăng đó cho đến khi rõ ràng rằng không cần lặp lại nữa. Nói cách khác, tổng thể dự án được chia thành một loạt các dự án mini thác nước. Trong mỗi dự án nhỏ, việc lặp lại được thực hiện khi cần thiết, như trong Hình 2.5. Do đó, lý do đoạn trước nói rằng mô hình lặp và tăng dần giống như mô hình thác nước là bởi vì mô hình lặp và tăng dần là mô hình thác nước, được áp dụng liên tiếp.51+52

2.9 Các mô hình vòng đời khác

Bây giờ chúng ta xem xét một số mô hình vòng đời khác, bao gồm mô hình xoắn ốc và mô hình đồng bộ hóa và ổn định. Chúng tôi bắt đầu với mô hình sửa lỗi và viết mã khét tiếng.

2.9.1 Mô hình vòng đời mã và sửa lỗi

Thật không may là rất nhiều sản phẩm được phát triển bằng cách sử dụng cái có thể được gọi là mô hình vòng đời mã và sửa lỗi. Sản phẩm được thực hiện mà không có yêu cầu hoặc thông số kỹ thuật, hoặc bất kỳ nỗ lực nào về thiết kế. Thay vào đó, các nhà phát triển chỉ cần ném mã lại với nhau và làm lại nó nhiều lần nếu cần để làm hài lòng khách hàng. Cách tiếp cận này được thể hiện trong Hình 2.8, hiển thị rõ ràng sự vắng mặt của các yêu cầu, trích dẫn cụ thể và thiết kế. Mặc dù cách tiếp cận này có thể hoạt động tốt trên các bài tập lập trình ngắn dài 100 hoặc 200 dòng, nhưng mô hình mã và sửa lỗi hoàn toàn không đạt yêu cầu đối với các sản phẩm có kích thước hợp lý. Hình 1.6 cho thấy chi phí thay đổi một sản phẩm phần mềm là tương đối nhỏ nếu thay đổi được thực hiện trong các giai đoạn yêu cầu, phân tích hoặc thiết kế nhưng lại phát triển lớn đến mức không thể chấp nhận được nếu các thay đổi được thực hiện sau khi sản phẩm đã được mã hóa hoặc tệ hơn là nếu nó đã đã được phân phối và cài đặt trên máy tính của khách hàng. Do đó, chi phí của phương pháp mã và sửa lỗi thực sự lớn hơn nhiều so với chi phí của một sản phẩm được thiết kế tỉ mỉ và có cấu hình phù hợp. Ngoài ra, việc bảo trì một sản phẩm có thể cực kỳ khó khăn nếu không có tài liệu thiết kế hoặc trích dẫn cụ thể, và khả năng xảy ra lỗi hồi quy là lớn hơn đáng kể. Thay vì cách tiếp cận mã và sửa, điều cần thiết là, trước khi bắt đầu phát triển một sản phẩm, một mô hình vòng đời thích hợp phải được chọn.

Đáng tiếc, tất cả các dự án đều sử dụng mô hình mã và sửa. Vấn đề đặc biệt nghiêm trọng trong các tổ chức chỉ đo lường tiến độ bằng các dòng mã, vì vậy các thành viên của nhóm phát triển phần mềm bị áp lực phải tạo ra càng nhiều dòng mã càng tốt, bắt đầu từ Ngày đầu tiên của dự án. Mô hình mã và sửa lỗi là cách dễ nhất để phát triển phần mềm — và cho đến nay là cách tồi tệ nhất.

Một phiên bản đơn giản nhất của mô hình thác nước đã được trình bày trong Phần 2.2. Bây giờ chúng ta xem xét mô hình đó chi tiết hơn. 53

2.9.2 Mô hình vòng đời thác nước

Mô hình vòng đời thác nước lần đầu tiên được Royce đưa ra [1970]. Hình 2.9 cho thấy các vòng phản hồi để bảo trì trong khi sản phẩm đang được phát triển, như được trình bày trong Hình 2.3, mô hình thác nước đơn giản hóa. Hình 2.9 cũng cho thấy các vòng phản hồi để bảo trì giao hàng sau.

Một điểm quan trọng liên quan đến mô hình thác nước là không có giai đoạn nào là hoàn thành cho đến khi tài liệu cho giai đoạn đó được hoàn thành và các sản phẩm của giai đoạn đó đã được nhóm đảm bảo chất lượng phần mềm (SQA) phê duyệt. Điều này chuyển thành các sửa đổi; nếu các sản phẩm của giai đoạn trước đó phải được thay đổi do tuân theo một vòng phản hồi, thì giai đoạn trước đó chỉ được coi là hoàn thành khi tài liệu cho giai đoạn đã được sửa đổi và các sửa đổi đã được kiểm tra bởi nhóm SQA. Vốn có trong mọi giai đoạn của mô hình thác nước đang thử nghiệm. Thử nghiệm không phải là một giai đoạn riêng biệt chỉ được thực hiện sau khi sản phẩm đã được xây dựng, cũng không phải chỉ được thực hiện vào cuối mỗi giai đoạn. Thay vào đó, như đã nêu trong Phần 1.7, việc kiểm thử phải được tiến hành liên tục trong suốt quá trình phần mềm. Đặc biệt, trong quá trình bảo trì, cần phải đảm bảo không chỉ rằng phiên bản sửa đổi của sản phẩm vẫn thực hiện những gì phiên bản trước đó đã làm — và vẫn thực hiện đúng (kiểm tra hồi quy) —nhưng nó cũng đáp ứng mọi yêu cầu mới do khách hàng.

Mô hình thác nước có nhiều điểm mạnh, bao gồm cách tiếp cận có kỷ luật được thực thi — quy định rằng tài liệu được cung cấp ở mỗi giai đoạn và yêu cầu rằng tất cả các sản phẩm của mỗi giai đoạn (bao gồm cả tài liệu) phải được kiểm tra tỉ mỉ bởi SQA. Tuy nhiên, thực tế là mô hình thác nước được điều khiển bằng tài liệu cũng có thể là một điểm yếu. Để thấy điều này, hãy xem xét hai kịch bản có phần kỳ lạ sau đây.

Đầu tiên, Joe và Jane Johnson quyết định xây một ngôi nhà. Họ tham khảo ý kiến ​​của một kiến ​​trúc sư. Thay vì cho họ xem các bản phác thảo, kế hoạch và có lẽ là một mô hình tỷ lệ, kiến ​​trúc sư đưa cho họ một tài liệu đánh máy dài 20 trang mô tả ngôi nhà theo các thuật ngữ kỹ thuật cao. Mặc dù cả Joe và Jane đều không có kinh nghiệm kiến ​​trúc trước đây và hầu như không hiểu tài liệu, họ vẫn nhiệt tình ký tên vào nó và nói, "Hãy tiến hành xây dựng ngôi nhà!"

Một kịch bản khác như sau: Mark Marberry mua bộ quần áo của anh ấy qua đường bưu điện. Thay vì gửi cho anh ta những bức ảnh về bộ quần áo của họ và những mẫu vải có sẵn qua thư, công ty gửi cho Mark một bản mô tả bằng văn bản về đường cắt và chất liệu vải sản phẩm của họ. Sau đó, Mark đặt hàng một bộ đồ chỉ dựa trên bản mô tả bằng văn bản.

Hai trường hợp trước rất khó xảy ra. Tuy nhiên, họ mô tả chính xác cách phần mềm thường được xây dựng bằng mô hình thác nước. Quá trình bắt đầu với các cation cụ thể. Nhìn chung, các tài liệu trích dẫn cụ thể dài, chi tiết và khá thú vị là đọc khá nhàm chán. Khách hàng thường thiếu kinh nghiệm trong việc đọc các trích dẫn đặc tả phần mềm và khó khăn khác biệt này là do thực tế là các tài liệu cation đặc biệt thường được viết theo phong cách mà khách hàng không quen thuộc. Khó khăn khác thậm chí còn tồi tệ hơn khi các cation đặc tả được viết bằng ngôn ngữ cation đặc tả chính thức như Z [Spivey, 1992] (Phần 12.9). Tuy nhiên, khách hàng tiến hành ký tên vào tài liệu cation cụ thể, cho dù đã hiểu đúng hay chưa. Theo nhiều cách, có rất ít sự khác biệt giữa Joe và Jane Johnson ký hợp đồng xây dựng một ngôi nhà từ bản mô tả bằng văn bản mà họ chỉ hiểu được một phần và khách hàng phê duyệt sản phẩm phần mềm được mô tả dưới dạng tài liệu cation cụ thể mà họ chỉ hiểu được một phần.

Mark Marberry và những bộ quần áo đặt hàng qua thư của anh ấy có vẻ kỳ quái ở cực điểm, nhưng đó chính xác là những gì sẽ xảy ra khi mô hình thác nước được sử dụng trong phát triển phần mềm. Lần đầu tiên khách hàng nhìn thấy một sản phẩm đang hoạt động chỉ là sau khi toàn bộ sản phẩm đã được mã hóa. Một điều ngạc nhiên nhỏ là các nhà phát triển phần mềm luôn sợ hãi câu nói, "Tôi biết đây là điều tôi yêu cầu, nhưng nó không thực sự là điều tôi muốn."

Điều gì đã xảy ra? Có một sự khác biệt đáng kể giữa cách khách hàng hiểu sản phẩm như được mô tả trong tài liệu đặc điểm kỹ thuật và sản phẩm thực tế. Các thông số kỹ thuật chỉ tồn tại trên giấy; do đó khách hàng không thể thực sự hiểu bản thân sản phẩm sẽ như thế nào. Mô hình thác nước, phụ thuộc rất quan trọng vào các thông số kỹ thuật bằng văn bản, có thể dẫn đến việc xây dựng các sản phẩm đơn giản là không đáp ứng được nhu cầu thực sự của khách hàng. 54

Công bằng mà nói, cần phải chỉ ra rằng, giống như một kiến trúc sư có thể giúp khách hàng hiểu những gì sẽ được xây dựng bằng cách cung cấp các mô hình tỷ lệ, bản phác thảo và kế hoạch, vì vậy kỹ sư phần mềm có thể sử dụng các kỹ thuật đồ họa, chẳng hạn như sơ đồ luồng dữ liệu (Phần 12.3 ) hoặc các sơ đồ UML (Chương 17) để giao tiếp với máy khách. Vấn đề là những hỗ trợ đồ họa này không mô tả thành phẩm sẽ hoạt động như thế nào. Ví dụ, có một sự khác biệt đáng kể giữa lưu đồ (mô tả sơ đồ về sản phẩm) và chính sản phẩm đang hoạt động. Trong cuốn sách này, hai giải pháp được đưa ra để giải quyết vấn đề mà tài liệu đặc tả thường không mô tả một sản phẩm theo cách cho phép khách hàng xác định liệu sản phẩm được đề xuất có đáp ứng nhu cầu của họ hay không. Giải pháp hướng đối tượng được mô tả trong Chương 11 và 13. Giải pháp cổ điển là mô hình tạo mẫu nhanh, được mô tả trong Phần 2.9.3. 55

2.9.3 Mô hình vòng đời tạo mẫu nhanh

Một mẫu thử nhanh là một mô hình hoạt động có chức năng tương đương với một tập hợp con của sản phẩm. Ví dụ: nếu sản phẩm mục tiêu là xử lý các khoản phải trả, khoản phải thu và kho bãi, thì nguyên mẫu nhanh có thể bao gồm một sản phẩm thực hiện xử lý màn hình để thu thập dữ liệu và in báo cáo, nhưng không cập nhật tệp hoặc xử lý lỗi. Một mẫu thử nhanh cho sản phẩm mục tiêu nhằm xác định nồng độ của một loại enzyme trong dung dịch có thể thực hiện phép tính và hiển thị câu trả lời, nhưng không thực hiện bất kỳ xác nhận hoặc kiểm tra tính hợp lý nào của dữ liệu đầu vào.

Bước đầu tiên trong mô hình vòng đời tạo mẫu nhanh được mô tả trong Hình 2.10 là xây dựng một nguyên mẫu nhanh và cho phép khách hàng và người dùng tương lai tương tác và thử nghiệm với nguyên mẫu nhanh. Sau khi khách hàng hài lòng rằng nguyên mẫu nhanh thực sự thực hiện hầu hết những gì được yêu cầu, các nhà phát triển có thể tạo tài liệu cation cụ thể với một số đảm bảo rằng sản phẩm đáp ứng nhu cầu thực sự của khách hàng. 55

Sau khi tạo ra nguyên mẫu nhanh, quá trình phần mềm tiếp tục như trong Hình 2.10. Một điểm mạnh chính của mô hình tạo mẫu nhanh là sự phát triển của sản phẩm về cơ bản là tuyến tính, tiến hành từ nguyên mẫu nhanh đến sản phẩm được giao; các vòng phản hồi của mô hình thác nước (Hình 2.9) ít có khả năng cần thiết hơn trong mô hình tạo mẫu nhanh. Có một số lý do cho điều này. Đầu tiên, các thành viên của nhóm phát triển sử dụng nguyên mẫu nhanh để xây dựng tài liệu cation cụ thể. Bởi vì nguyên mẫu nhanh đang hoạt động đã được xác nhận thông qua tương tác với khách hàng, điều hợp lý để mong đợi rằng tài liệu đặc tả kết quả sẽ chính xác. Thứ hai, hãy xem xét thiết kế. Mặc dù nguyên mẫu thử nghiệm nhanh (hoàn toàn đúng) đã được lắp ráp vội vã, nhóm thiết kế có thể hiểu rõ về nó — tệ nhất là nó sẽ thuộc loại “không nên làm như thế nào”. Một lần nữa, các vòng phản hồi của mô hình thác nước ít có khả năng cần thiết hơn ở đây.

Việc thực hiện sẽ đến tiếp theo. Trong mô hình thác nước, việc thực hiện thiết kế đôi khi dẫn đến lỗi thiết kế. Trong mô hình tạo mẫu nhanh, thực tế là một phiên bản làm việc ban đầu của sản phẩm phần mềm đã được xây dựng có xu hướng làm giảm nhu cầu sửa chữa thiết kế trong hoặc sau khi thực hiện. Nguyên mẫu đã cung cấp một số thông tin chi tiết cho nhóm thiết kế, mặc dù nó có thể chỉ phản ánh một phần chức năng của sản phẩm mục tiêu hoàn chỉnh.

Khi sản phẩm đã được khách hàng chấp nhận và cài đặt, việc bảo trì sau giao hàng sẽ bắt đầu. Tùy thuộc vào nhiệm vụ bảo trì cụ thể phải được thực hiện, chu trình được đưa vào lại hoặc ở giai đoạn yêu cầu, phân tích, thiết kế hoặc thực hiện.

Một khía cạnh thiết yếu của một nguyên mẫu nhanh được thể hiện trong từ nhanh chóng. Các nhà phát triển nên cố gắng xây dựng nguyên mẫu nhanh càng nhanh càng tốt để tăng tốc quá trình phát triển phần mềm. Xét cho cùng, công dụng duy nhất của nguyên mẫu nhanh là xác định nhu cầu thực sự của khách hàng là gì; một khi điều này đã được xác định, việc triển khai nguyên mẫu nhanh sẽ bị loại bỏ nhưng các bài học kinh nghiệm được giữ lại và sử dụng trong các giai đoạn phát triển tiếp theo. Vì lý do này, cấu trúc bên trong của nguyên mẫu nhanh không phù hợp. Điều quan trọng là nguyên mẫu phải được xây dựng nhanh chóng và sửa đổi nhanh chóng để phản ánh nhu cầu của khách hàng. Do đó, tốc độ là điều cốt yếu.

Tạo mẫu nhanh được thảo luận chi tiết hơn trong Chương 11.

2.9.4 Mô hình vòng đời nguồn mở

Hầu hết tất cả các dự án phần mềm nguồn mở thành công đều trải qua hai giai đoạn không chính thức. Đầu tiên, một cá nhân có ý tưởng về một chương trình, chẳng hạn như hệ điều hành (Linux), trình duyệt Net (Firefox) hoặc máy chủ Web (Apache). Người đó xây dựng một phiên bản ban đầu, sau đó được cung cấp để phân phối miễn phí cho bất kỳ ai muốn có một bản sao; ngày nay, điều này được thực hiện thông qua Internet, tại các trang web như SourceForge.net và FreshMeat.net. Nếu ai đó tải xuống bản sao của phiên bản ban đầu và nghĩ rằng chương trình đáp ứng được nhu cầu, người đó sẽ bắt đầu sử dụng chương trình đó.

Nếu có đủ sự quan tâm đến chương trình, dự án sẽ chuyển dần sang giai đoạn hai không chính thức. Người dùng trở thành đồng phát triển, trong đó một số người dùng báo cáo các lỗi và những người khác đề xuất cách sửa các lỗi đó. Một số người dùng đưa ra ý tưởng để mở rộng chương trình và những người khác thực hiện những ý tưởng đó. Khi chương trình mở rộng về chức năng, nhưng những người dùng khác chuyển chương trình để nó có thể chạy trên các kết hợp hệ điều hành / phần cứng bổ sung. Một khía cạnh chính là các cá nhân thường làm việc trên một dự án mã nguồn mở trong thời gian rảnh rỗi của họ trên cơ sở tự nguyện; họ không được trả tiền để tham gia. 56

Bây giờ hãy xem xét kỹ hơn ba hoạt động của giai đoạn không chính thức thứ hai:

1. Báo cáo và sửa chữa các khuyết tật là bảo trì sửa chữa.

2. Thêm chức năng bổ sung là bảo trì hoàn hảo.

3. Chuyển chương trình sang một môi trường mới là bảo trì thích ứng.

Nói cách khác, giai đoạn không chính thức thứ hai của mô hình vòng đời nguồn mở chỉ bao gồm bảo trì sau giao hàng, như thể hiện trong Hình 2.11. Trên thực tế, thuật ngữ đồng phát triển trong đoạn thứ hai của phần này nên là đồng bảo trì.

Có một số điểm khác biệt chính giữa mô hình vòng đời phần mềm nguồn đóng và nguồn mở:

• Phần mềm mã nguồn đóng được duy trì và kiểm tra bởi các nhóm nhân viên của tổ chức sở hữu phần mềm. Người dùng đôi khi gửi báo cáo lỗi. Tuy nhiên, chúng bị hạn chế đối với các báo cáo thất bại (báo cáo về hành vi không chính xác được quan sát thấy); người dùng không có quyền truy cập vào mã nguồn, vì vậy họ không thể gửi báo cáo lỗi (báo cáo mô tả nơi mã nguồn không chính xác và cách sửa lỗi).

Ngược lại, phần mềm nguồn mở thường được duy trì bởi các tình nguyện viên không được trả lương. Người dùng được khuyến khích gửi báo cáo lỗi. Mặc dù tất cả người dùng đều có quyền truy cập vào mã nguồn, nhưng chỉ một số thiểu số có khuynh hướng và thời gian, cũng như các kỹ năng cần thiết, để sử dụng mã nguồn và gửi báo cáo lỗi (“bản sửa lỗi”); do đó hầu hết các báo cáo lỗi là báo cáo lỗi. Nhìn chung, có một nhóm cốt lõi gồm những người bảo trì chuyên dụng chịu trách nhiệm quản lý dự án mã nguồn mở. Một số thành viên của nhóm ngoại vi, tức là những người dùng không phải là thành viên của nhóm cốt lõi, thỉnh thoảng chọn gửi báo cáo lỗi. Các thành viên của nhóm cốt lõi có trách nhiệm đảm bảo rằng những khiếm khuyết này được sửa chữa. Chi tiết hơn, khi một báo cáo lỗi được gửi, một thành viên nhóm cốt lõi sẽ kiểm tra xem bản sửa lỗi có thực sự giải quyết được vấn đề hay không và sửa đổi mã nguồn một cách thích hợp. Khi một báo cáo lỗi được gửi, một thành viên của nhóm cốt lõi sẽ tự mình xác định cách khắc phục hoặc giao nhiệm vụ đó cho một tình nguyện viên khác, thường là một thành viên của nhóm ngoại vi mong muốn tham gia nhiều hơn vào dự án mã nguồn mở. Một lần nữa, quyền cài đặt bản sửa lỗi trong phần mềm bị hạn chế đối với các thành viên của nhóm cốt lõi. 57

• Các phiên bản mới của phần mềm nguồn đóng thường được phát hành khoảng một năm một lần. Mỗi phiên bản mới đều được kiểm tra cẩn thận bởi nhóm đảm bảo chất lượng phần mềm trước khi phát hành; một loạt các trường hợp thử nghiệm được chạy.

Ngược lại, một câu châm ngôn của phong trào mã nguồn mở là “Phát hành sớm. Phát hành thường xuyên ”[Raymond, 2000]. Đó là, nhóm cốt lõi phát hành phiên bản mới của một sản phẩm mã nguồn mở ngay khi nó sẵn sàng, có thể là một tháng hoặc thậm chí chỉ một ngày sau khi phiên bản trước đó được phát hành. Phiên bản mới này được phát hành sau khi thử nghiệm tối thiểu; giả định rằng thử nghiệm mở rộng hơn sẽ được thực hiện bởi các thành viên của nhóm ngoại vi. Một phiên bản mới có thể được cài đặt bởi hàng trăm nghìn người dùng trong vòng một hoặc hai ngày kể từ ngày phát hành. Những người dùng này không chạy các trường hợp thử nghiệm như vậy. Tuy nhiên, trong quá trình sử dụng phiên bản mới trên máy tính của họ, họ gặp phải những lỗi hỏng hóc và thông báo qua e-mail. Bằng cách này, các lỗi trong phiên bản mới (cũng như các lỗi sâu hơn trong các phiên bản trước) được đưa ra ánh sáng và được sửa chữa.

So sánh các Hình 2.8, 2.10 và 2.11, chúng ta thấy rằng mô hình vòng đời nguồn mở có các đặc điểm chung với cả mô hình mã và sửa chữa và mô hình tạo mẫu nhanh. Trong cả ba mô hình vòng đời, một phiên bản làm việc ban đầu được tạo ra. Trong trường hợp của mô hình quay nhanh, phiên bản ban đầu này bị loại bỏ, và sản phẩm mục tiêu sau đó được chỉ định và thiết kế trước khi được mã hóa. Trong cả mô hình vòng đời mã và sửa lỗi và mã nguồn mở, phiên bản ban đầu được làm lại cho đến khi nó trở thành sản phẩm mục tiêu. Theo đó, trong một dự án mã nguồn mở, thường không có thông số kỹ thuật hoặc thiết kế.

Ghi nhớ tầm quan trọng to lớn của việc có các trích dẫn và thiết kế cụ thể, làm thế nào mà một số dự án mã nguồn mở lại thành công như vậy? Trong thế giới mã nguồn đóng, một số chuyên gia phần mềm có kỹ năng cao hơn và một số ít kỹ năng hơn (xem Phần 9.2). Thách thức của việc sản xuất phần mềm mã nguồn mở đã thu hút một số chuyên gia phần mềm giỏi nhất. Nói cách khác, một dự án nguồn mở có thể thành công, mặc dù thiếu các trích dẫn hoặc thiết kế cụ thể, nếu kỹ năng của các cá nhân làm việc trong dự án đó xuất sắc đến mức họ có thể hoạt động hiệu quả mà không cần trích dẫn hoặc thiết kế cụ thể.

Mô hình vòng đời nguồn mở bị hạn chế về khả năng áp dụng. Một mặt, mô hình mã nguồn mở đã được sử dụng cực kỳ thành công cho một số dự án phần mềm cơ sở hạ tầng, chẳng hạn như hệ điều hành (Linux, OpenBSD, Mach, Darwin), trình duyệt web (Firefox, Netscape), trình biên dịch (gcc), máy chủ Web (Apache), hoặc hệ thống quản lý cơ sở dữ liệu (MySQL). Mặt khác, khó có thể hình dung sự phát triển mã nguồn mở của một sản phẩm phần mềm chỉ được sử dụng trong một tổ chức thương mại. Chìa khóa để phát triển phần mềm nguồn mở là các thành viên của cả nhóm cốt lõi và ngoại vi đều là người sử dụng phần mềm đang được phát triển. Do đó, mô hình vòng đời nguồn mở không thể áp dụng trừ khi sản phẩm mục tiêu được nhiều người dùng xem là hữu ích đối với họ.

Tại thời điểm viết bài, có khoảng 350.000 dự án mã nguồn mở tại SourceForge.net và FreshMeat.net. Khoảng một nửa trong số họ thậm chí chưa bao giờ thu hút một nhóm làm việc trong dự án. Trong số những người đã bắt đầu công việc, ưu thế vượt trội chưa bao giờ được hoàn thành và không bao giờ có khả năng tiến bộ hơn nữa. Nhưng khi mô hình mã nguồn mở hoạt động, nó đôi khi thành công ngoài sức tưởng tượng. Các sản phẩm mã nguồn mở được liệt kê trong ngoặc đơn ở đoạn trước được sử dụng rộng rãi; hầu hết chúng được sử dụng thường xuyên bởi hàng triệu người dùng.

Các giải thích về sự thành công của mô hình vòng đời nguồn mở được trình bày trong Chương 4 trong bối cảnh các khía cạnh tổ chức nhóm của các dự án phần mềm nguồn mở. 58

2.9.5 Quy trình nhanh

Lập trình cực đoan [Beck, 2000] là một cách tiếp cận mới gây tranh cãi đối với việc phát triển phần mềm dựa trên mô hình lặp đi lặp lại và tăng dần. Bước đầu tiên là nhóm phát triển phần mềm xác định các tính năng (câu chuyện) khác nhau mà khách hàng muốn sản phẩm hỗ trợ. Đối với mỗi tính năng như vậy, nhóm sẽ thông báo cho khách hàng thời gian thực hiện tính năng đó và chi phí sẽ là bao nhiêu. Bước đầu tiên này tương ứng với các yêu cầu và quy trình phân tích của mô hình lặp lại và tăng dần (Hình 2.4).

Khách hàng chọn các tính năng được đưa vào mỗi bản dựng kế tiếp bằng cách sử dụng phân tích chi phí - lợi ích (Phần 5.2), nghĩa là, trên cơ sở thời lượng và ước tính chi phí do nhóm phát triển cung cấp cũng như lợi ích tiềm năng của tính năng cho doanh nghiệp của mình. Bản dựng được đề xuất được chia thành các phần nhỏ hơn được gọi là các nhiệm vụ. Một lập trình viên đầu tiên vẽ các trường hợp kiểm thử cho một nhiệm vụ; đây được gọi là phát triển theo hướng thử nghiệm (TDD). Hai lập trình viên làm việc cùng nhau trên một máy tính (lập trình cặp) [Williams, Kessler, Cunningham và Jeffries, 2000], thực hiện nhiệm vụ và đảm bảo rằng tất cả các trường hợp thử nghiệm hoạt động chính xác. Hai lập trình viên luân phiên đánh máy sau mỗi 15 hoặc 20 phút; lập trình viên không gõ cẩn thận sẽ kiểm tra mã khi đối tác của họ nhập. Nhiệm vụ sau đó được tích hợp vào phiên bản hiện tại của sản phẩm. Tốt nhất, việc triển khai và tích hợp một nhiệm vụ không nên mất quá vài giờ. Nói chung, một số cặp sẽ triển khai các tác vụ song song, do đó, việc tích hợp về cơ bản là liên tục. Các thành viên trong nhóm thay đổi đối tác mã hóa hàng ngày, nếu có thể; học hỏi từ các thành viên khác trong nhóm sẽ nâng cao trình độ kỹ năng của mọi người. Các trường hợp thử nghiệm TDD được sử dụng cho nhiệm vụ được giữ lại và sử dụng trong tất cả các thử nghiệm tích hợp hơn nữa.

Một số hạn chế đối với lập trình cặp đã được quan sát thấy trong thực tế [Drobka, Noftz, và Raghu, 2004]. Ví dụ, lập trình cặp đòi hỏi khối lượng lớn thời gian không bị gián đoạn và các chuyên gia phần mềm có thể gặp khó khăn trong việc tìm kiếm các khối thời gian từ 3 đến 4 giờ. Ngoài ra, lập trình cặp không phải lúc nào cũng hoạt động tốt với những cá nhân nhút nhát hoặc hống hách, hoặc với hai lập trình viên thiếu kinh nghiệm.

Một số tính năng của lập trình cực đoan (XP) hơi khác so với cách mà phần mềm thường được phát triển:

• Các máy tính của nhóm XP được đặt ở trung tâm của một căn phòng lớn với những ngăn tủ nhỏ.

• Đại diện khách hàng làm việc với nhóm XP mọi lúc.

• Không một cá nhân nào có thể làm thêm giờ trong hai tuần liên tiếp.

• Không có chuyên môn hóa. Thay vào đó, tất cả các thành viên của nhóm XP làm việc dựa trên các yêu cầu, phân tích, thiết kế, mã và thử nghiệm.

• Không có bước thiết kế tổng thể trước khi xây dựng các công trình khác nhau. Thay vào đó, thiết kế được sửa đổi trong khi sản phẩm đang được chế tạo. Thủ tục này được gọi là tái cấu trúc. Bất cứ khi nào một trường hợp thử nghiệm không chạy, mã sẽ được tổ chức lại cho đến khi nhóm được thỏa mãn rằng thiết kế đơn giản, dễ hiểu và chạy tất cả các trường hợp thử nghiệm một cách thỏa đáng. 59

Hai từ viết tắt hiện được kết hợp với lập trình cực đoan là YAGNI (bạn sẽ không cần đến nó) và DTSTTCPW (làm điều đơn giản nhất có thể hoạt động). Nói cách khác, một nguyên tắc của lập trình cực đoan là giảm thiểu số lượng các tính năng; không cần thiết phải xây dựng một sản phẩm làm được nhiều hơn những gì khách hàng thực sự cần.

Lập trình cực đoan là một trong số các mô hình mới được gọi chung là các quy trình nhanh. Mười bảy nhà phát triển phần mềm (sau này được gọi là Liên minh Agile) đã gặp nhau tại một khu nghỉ mát trượt tuyết ở Utah trong hai ngày vào tháng 2 năm 2001 và đưa ra Tuyên ngôn về Phát triển Phần mềm Agile [Beck và cộng sự, 2001]. Nhiều người trong số những người tham gia trước đây đã là tác giả của các phương pháp phát triển phần mềm của riêng họ, bao gồm Lập trình cực đoan [Beck, 2000], Crystal [Cockburn, 2001] và Scrum [Schwaber, 2001]. Do đó, Agile Alliance không quy định một mô hình vòng đời cụ thể, mà đặt ra một nhóm các nguyên tắc cơ bản chung cho các cách tiếp cận phát triển phần mềm của cá nhân họ.

Các quy trình Agile có đặc điểm là ít chú trọng đến phân tích và thiết kế hơn so với hầu hết các mô hình vòng đời hiện đại khác. Việc triển khai bắt đầu sớm hơn nhiều trong vòng đời bởi vì phần mềm làm việc được coi là quan trọng hơn tài liệu chi tiết. Đáp ứng các thay đổi trong yêu cầu là một mục tiêu chính khác của các quy trình nhanh và tầm quan trọng của việc cộng tác với khách hàng cũng vậy.

Một trong những nguyên tắc trong Tuyên ngôn là cung cấp phần mềm hoạt động thường xuyên, lý tưởng là 2 hoặc 3 tuần một lần. Một cách để đạt được điều này là sử dụng hộp thời gian [Jalote, Palit, Kurien, và Peethamber, 2004], đã được sử dụng trong nhiều năm như một kỹ thuật quản lý thời gian. Một khoảng thời gian cụ thể được dành cho một nhiệm vụ và các thành viên trong nhóm sau đó sẽ làm tốt nhất công việc mà họ có thể trong thời gian đó. Trong bối cảnh của các quy trình nhanh, thường là 3 tuần được dành cho mỗi lần lặp lại. Một mặt, nó cho khách hàng hiểu rằng một phiên bản mới với chức năng bổ sung sẽ đến sau 3 tuần một lần. Mặt khác, các nhà phát triển biết rằng họ sẽ có 3 tuần (nhưng không nhiều hơn) để cung cấp một bản lặp mới mà không có sự can thiệp của khách hàng dưới bất kỳ hình thức nào; một khi khách hàng đã chọn tác phẩm để lặp lại, nó không thể được thay đổi hoặc tăng lên. Tuy nhiên, nếu không thể hoàn thành toàn bộ công việc trong hộp thời gian, công việc có thể bị giảm bớt (“descoped”). Nói cách khác, các quy trình nhanh yêu cầu thời gian xác định, không phải tính năng xác định.

Một đặc điểm chung khác của các quy trình nhanh là có một cuộc họp ngắn vào một thời điểm thường xuyên mỗi ngày. Tất cả các thành viên trong nhóm phải tham gia cuộc họp. Làm cho tất cả những người tham gia đứng thành vòng tròn, thay vì ngồi xung quanh bàn, giúp đảm bảo rằng cuộc họp kéo dài không quá 15 phút quy định. Mỗi thành viên trong nhóm lần lượt trả lời năm câu hỏi:

• Tôi đã làm gì kể từ cuộc họp ngày hôm qua?

• Tôi đang làm gì hôm nay?

• Những vấn đề nào đang ngăn cản tôi đạt được điều này?

• Chúng ta đã quên những gì?

• Tôi đã học được gì mà tôi muốn chia sẻ với nhóm?

Mục đích của cuộc họp đứng lên là nêu vấn đề chứ không phải giải quyết chúng; các giải pháp được tìm thấy tại các cuộc họp tiếp theo, tốt nhất là được tổ chức trực tiếp sau cuộc họp đứng. Giống như hộp thời gian, các cuộc họp đứng là một kỹ thuật quản lý thành công hiện được sử dụng trong bối cảnh của các quy trình nhanh. Cả hai lần lặp lại theo hộp thời gian và cuộc họp độc lập đều là những trường hợp của hai nguyên tắc cơ bản làm nền tảng cho tất cả các phương pháp linh hoạt: giao tiếp và đáp ứng nhu cầu của khách hàng càng nhanh càng tốt. 60

Các quy trình Agile đã được sử dụng thành công trên một số dự án quy mô nhỏ. Tuy nhiên, các quy trình nhanh vẫn chưa được sử dụng đủ rộng rãi để xác định xem liệu cách tiếp cận này có thực hiện được lời hứa ban đầu của nó hay không. Hơn nữa, ngay cả khi các quy trình nhanh trở nên tốt cho các sản phẩm phần mềm quy mô nhỏ, điều đó không nhất thiết có nghĩa là chúng có thể được sử dụng cho các sản phẩm phần mềm quy mô vừa hoặc lớn, như bây giờ sẽ được giải thích.

Để đánh giá cao lý do tại sao nhiều chuyên gia phần mềm tỏ ra nghi ngờ về các quy trình nhanh trong bối cảnh của các sản phẩm phần mềm quy mô vừa và đặc biệt là [Reifer, Maurer, và Erdogmus, 2003], hãy xem xét sự tương tự sau đây của Grady Booch [2000]. Bất cứ ai cũng có thể ghép thành công một vài tấm ván để xây một chuồng chó, nhưng sẽ thật ngu ngốc nếu xây một ngôi nhà ba phòng ngủ mà không có kế hoạch chi tiết. Ngoài ra, các kỹ năng về hệ thống ống nước, hệ thống dây điện và roofi là cần thiết để xây dựng một ngôi nhà ba phòng ngủ, và việc kiểm tra là điều cần thiết. (Có nghĩa là, có thể xây dựng các sản phẩm phần mềm quy mô nhỏ không nhất thiết có nghĩa là một người có kỹ năng xây dựng các sản phẩm phần mềm quy mô trung bình.) Hơn nữa, thực tế là một tòa nhà chọc trời cao bằng 1000 căn nhà không có nghĩa là một người có thể xây dựng một tòa nhà chọc trời bằng cách chồng 1000 chuồng chó lên nhau. Nói cách khác, việc xây dựng các sản phẩm phần mềm quy mô lớn thậm chí còn đòi hỏi những kỹ năng chuyên biệt và phức tạp hơn những kỹ năng cần thiết để kết hợp các sản phẩm phần mềm quy mô nhỏ với nhau.

Yếu tố quyết định quan trọng trong việc quyết định liệu các quy trình nhanh có thực sự là một bước đột phá lớn trong kỹ thuật phần mềm hay không sẽ là chi phí bảo trì sau giao hàng trong tương lai (Phần 1.3.2). Có nghĩa là, nếu việc sử dụng các quy trình nhanh dẫn đến giảm chi phí bảo trì sau giao hàng, thì XP và các quy trình nhanh khác sẽ được áp dụng rộng rãi. Mặt khác, tái cấu trúc là một thành phần nội tại của các quy trình nhanh. Như đã giải thích trước đây, sản phẩm không được thiết kế tổng thể; thay vào đó, thiết kế được phát triển từng bước và mã được tổ chức lại bất cứ khi nào thiết kế hiện tại không đạt yêu cầu vì bất kỳ lý do gì. Việc tái cấu trúc này sau đó sẽ tiếp tục trong quá trình bảo trì sau giao hàng. Nếu thiết kế của một sản phẩm khi nó vượt qua kiểm tra chấp nhận được mở ra và có thể hoạt động được, thì việc bảo trì hoàn hảo sẽ dễ dàng đạt được với chi phí thấp. Tuy nhiên, nếu thiết kế phải được tái cấu trúc bất cứ khi nào thêm chức năng bổ sung, thì chi phí bảo trì sau giao hàng của sản phẩm đó sẽ cao đến mức không thể chấp nhận được. Do tính mới của cách tiếp cận, về cơ bản vẫn không có dữ liệu về việc bảo trì phần mềm được phát triển bằng các quy trình nhanh. Tuy nhiên, dữ liệu bảo trì sơ bộ chỉ ra rằng việc tái cấu trúc có thể tiêu tốn một tỷ lệ lớn trong tổng chi phí [Li và Alshayeb, 2002].

Các thử nghiệm đã chỉ ra rằng một số tính năng nhất định của các quy trình nhanh có thể hoạt động tốt. Ví dụ, Williams, Kessler, Cunningham và Jeffries [2000] đã chỉ ra rằng lập trình cặp dẫn đến việc phát triển mã chất lượng cao hơn trong thời gian ngắn hơn, với sự hài lòng trong công việc cao hơn. Tuy nhiên, một thử nghiệm mở rộng để đánh giá việc lập trình theo cặp trong bối cảnh bảo trì phần mềm được mô tả trong Phần 4.6 [Arisholm, Gallis, Dybå, và Sjøberg, 2007] đã đưa ra kết luận tương tự như phân tích 15 nghiên cứu đã được công bố so sánh hiệu quả của từng cá nhân và cặp lập trình [Dybå et al., 2007]: Nó phụ thuộc vào cả chuyên môn của lập trình viên và độ phức tạp của sản phẩm phần mềm và các nhiệm vụ cần giải quyết.

Tuyên ngôn về Phát triển Phần mềm Agile về cơ bản tuyên bố rằng các quy trình nhanh vượt trội hơn các quy trình kỷ luật hơn như Quy trình hợp nhất (Chương 3). Những người hoài nghi trả lời rằng những người ủng hộ các quy trình nhanh không nhiều hơn tin tặc. Tuy nhiên, có một nền tảng trung gian. Hai cách tiếp cận không phải là không tương thích; có thể kết hợp các tính năng đã được chứng minh của các quy trình nhanh trong khuôn khổ các quy trình có kỷ luật. Sự tích hợp hai cách tiếp cận này được mô tả trong các cuốn sách như sách của Boehm và Turner [2003].

Tóm lại, các quy trình nhanh dường như là một cách tiếp cận hữu ích để xây dựng các sản phẩm phần mềm quy mô nhỏ khi các yêu cầu của khách hàng còn mơ hồ. Ngoài ra, một số tính năng của quy trình nhanh có thể được sử dụng hiệu quả trong bối cảnh của các mô hình vòng đời khác. 61

2.9.6 Đồng bộ hóa và ổn định mô hình vòng đời

Microsoft, Inc., là nhà sản xuất phần mềm COTS lớn nhất thế giới. Phần lớn các gói của nó được xây dựng bằng cách sử dụng phiên bản của mô hình lặp đi lặp lại và tăng dần được gọi là mô hình vòng đời đồng bộ hóa và ổn định [Cusumano và Selby, 1997].

Giai đoạn phân tích yêu cầu được thực hiện bằng cách phỏng vấn nhiều khách hàng tiềm năng cho gói và trích xuất danh sách các tính năng ưu tiên cao nhất cho khách hàng. Một tài liệu cation cụ thể hiện đã được tạo ra. Tiếp theo, tác phẩm được chia thành ba hoặc bốn lần xây dựng. Bản dựng đầu tiên bao gồm các tính năng quan trọng nhất, bản dựng thứ hai bao gồm các tính năng quan trọng nhất tiếp theo, v.v. Mỗi công trình được thực hiện bởi một số nhóm nhỏ làm việc song song. Vào cuối mỗi ngày, tất cả các nhóm đồng bộ hóa; có nghĩa là, họ đặt các thành phần đã hoàn thành một phần lại với nhau và kiểm tra và gỡ lỗi sản phẩm kết quả. Ổn định được thực hiện ở cuối mỗi bản dựng. Bất kỳ lỗi nào còn lại đã được phát hiện cho đến nay đều đã được xử lý xong và hiện chúng đã đóng băng bản dựng; nghĩa là, sẽ không có thay đổi nào khác đối với các thông số kỹ thuật.

Bước đồng bộ hóa lặp đi lặp lại đảm bảo rằng các thành phần khác nhau luôn hoạt động cùng nhau. Một lợi thế khác của việc thực hiện thường xuyên sản phẩm được xây dựng một phần này là các nhà phát triển có được cái nhìn sâu sắc sớm về hoạt động của sản phẩm và có thể sửa đổi các yêu cầu nếu cần thiết trong quá trình xây dựng. Mô hình vòng đời có thể được sử dụng ngay cả khi cation đặc trưng ban đầu chưa hoàn chỉnh. Mô hình đồng bộ hóa và ổn định được xem xét thêm trong Phần 4.5, nơi các chi tiết tổ chức nhóm được thảo luận.

Mô hình xoắn ốc vẫn tồn tại lâu dài vì nó kết hợp các khía cạnh của tất cả các mô hình khác được mô tả trong Phần 2.9.

2.9.7 Mô hình vòng đời xoắn ốc

Như đã nêu trong Phần 2.5, một yếu tố rủi ro luôn liên quan đến sự phát triển của phần mềm. Ví dụ, nhân viên chủ chốt có thể từ chức trước khi sản phẩm được ghi chép đầy đủ. Nhà sản xuất phần cứng mà sản phẩm phụ thuộc rất nhiều có thể bị phá sản. Quá nhiều, hoặc quá ít, có thể được đầu tư vào việc kiểm tra và đảm bảo chất lượng. Sau khi chi hàng trăm nghìn đô la để phát triển một sản phẩm phần mềm lớn, những đột phá về công nghệ có thể khiến toàn bộ sản phẩm trở nên vô giá trị. Một tổ chức có thể nghiên cứu và phát triển một hệ thống quản lý cơ sở dữ liệu, nhưng trước khi sản phẩm có thể được đưa ra thị trường, một đối thủ cạnh tranh sẽ công bố một gói tương đương về chức năng, có giá thấp hơn. Các thành phần của một sản phẩm có thể không kết hợp với nhau khi thực hiện tích hợp. Vì những lý do rõ ràng, các nhà phát triển phần mềm cố gắng giảm thiểu những rủi ro như vậy nếu có thể.

Một cách để giảm thiểu một số loại rủi ro là xây dựng một nguyên mẫu. Như được mô tả trong Phần 2.9.3, một cách tiếp cận để giảm nguy cơ sản phẩm được giao sẽ không đáp ứng nhu cầu thực sự của khách hàng là tạo một mẫu thử nhanh trong giai đoạn yêu cầu. Trong các giai đoạn tiếp theo, các loại nguyên mẫu khác có thể phù hợp. Ví dụ, một công ty điện thoại có thể nghĩ ra một thuật toán mới, có vẻ hiệu quả cao để định tuyến cuộc gọi qua mạng đường dài. Nếu sản phẩm được triển khai nhưng không hoạt động như mong đợi, công ty điện thoại sẽ lãng phí chi phí phát triển sản phẩm. Ngoài ra, những khách hàng tức giận hoặc bất tiện có thể đưa doanh nghiệp của họ đi nơi khác. Có thể tránh được kết quả này bằng cách xây dựng một nguyên mẫu bằng chứng khái niệm để chỉ xử lý việc định tuyến các cuộc gọi và thử nghiệm nó trên một trình mô phỏng. Bằng cách này, hệ thống thực tế không bị xáo trộn; và đối với chi phí chỉ triển khai thuật toán định tuyến, công ty điện thoại có thể xác định xem liệu có đáng để phát triển toàn bộ bộ điều khiển mạng kết hợp thuật toán mới hay không. 62

Một nguyên mẫu thử nghiệm khái niệm không phải là một nguyên mẫu nhanh chóng được xây dựng để chắc chắn rằng các yêu cầu đã được xác định chính xác, như được mô tả trong Phần 2.9.3. Thay vào đó, nó giống như một nguyên mẫu kỹ thuật, tức là một mô hình quy mô được xây dựng để kiểm tra tính khả thi của việc xây dựng. Nếu nhóm phát triển lo ngại liệu một phần cụ thể của sản phẩm phần mềm được đề xuất có thể được xây dựng hay không, thì một nguyên mẫu thử nghiệm bằng chứng về khái niệm sẽ được xây dựng. Ví dụ, các nhà phát triển có thể lo lắng liệu một phép tính cụ thể có thể được thực hiện đủ nhanh hay không. Trong trường hợp đó, họ xây dựng một nguyên mẫu để kiểm tra thời gian của phép tính đó. Hoặc họ có thể lo lắng rằng phông chữ họ định sử dụng cho tất cả các màn hình sẽ quá nhỏ để người dùng bình thường có thể đọc mà không bị mỏi mắt. Trong trường hợp này, họ xây dựng một nguyên mẫu để hiển thị một số màn hình khác nhau và xác định bằng thử nghiệm xem liệu người dùng có thấy phông chữ nhỏ một cách khó chịu hay không.

Ý tưởng giảm thiểu rủi ro thông qua việc sử dụng các nguyên mẫu và các phương tiện khác là ý tưởng nằm trong mô hình vòng đời xoắn ốc [Boehm, 1988]. Một cách đơn giản hơn để xem xét mô hình vòng đời này là mô hình thác nước với mỗi giai đoạn được phân tích rủi ro trước đó, như thể hiện trong Hình 2.12. Trước khi bắt đầu mỗi giai đoạn, hãy cố gắng giảm thiểu (kiểm soát) rủi ro. Nếu không thể giảm thiểu tất cả các rủi ro đáng kể trong giai đoạn đó, thì dự án sẽ bị chấm dứt ngay lập tức. 63

Các nguyên mẫu có thể được sử dụng một cách hiệu quả để cung cấp thông tin về một số loại rủi ro nhất định. Ví dụ, các ràng buộc về thời gian nói chung có thể được kiểm tra bằng cách xây dựng một nguyên mẫu và đo lường xem nguyên mẫu có thể đạt được hiệu suất cần thiết hay không. Nếu nguyên mẫu là một biểu diễn chức năng chính xác của các tính năng liên quan của sản phẩm, thì các phép đo được thực hiện trên nguyên mẫu sẽ cung cấp cho các nhà phát triển một ý tưởng tốt về việc liệu có thể đạt được các hạn chế về thời gian hay không.

Các lĩnh vực rủi ro khác ít khả thi hơn đối với việc tạo mẫu, ví dụ, rủi ro mà nhân viên phần mềm cần thiết để xây dựng sản phẩm không thể được thuê hoặc nhân viên chủ chốt có thể từ chức trước khi dự án hoàn thành. Một rủi ro tiềm ẩn khác là một nhóm cụ thể có thể không đủ năng lực để phát triển một sản phẩm quy mô lớn cụ thể. Một nhà thầu thành công xây dựng các ngôi nhà cho một gia đình có lẽ sẽ không thể xây dựng một khu phức hợp cao cấp. Theo cách tương tự, có những khác biệt cơ bản giữa phần mềm quy mô nhỏ và quy mô lớn, và việc tạo mẫu ít được sử dụng. Rủi ro này không thể được giảm thiểu bằng cách thử nghiệm hiệu suất của nhóm trên một nguyên mẫu nhỏ hơn nhiều, trong đó các vấn đề về tổ chức nhóm đặc biệt đối với phần mềm quy mô lớn không thể phát sinh. Một lĩnh vực rủi ro khác mà việc tạo mẫu không thể được sử dụng là đánh giá các lời hứa giao hàng của một nhà cung cấp phần cứng. Một chiến lược mà nhà phát triển có thể áp dụng là xác định xem các khách hàng trước đây của nhà cung cấp đã được đối xử tốt như thế nào, nhưng hiệu suất trong quá khứ không có nghĩa là một yếu tố dự đoán nhất định về hiệu suất trong tương lai. Điều khoản phạt trong hợp đồng giao hàng là một cách để cố gắng đảm bảo rằng phần cứng thiết yếu được giao đúng thời hạn, nhưng điều gì sẽ xảy ra nếu nhà cung cấp từ chối ký một thỏa thuận bao gồm điều khoản như vậy? Ngay cả với điều khoản phạt, việc giao hàng muộn có thể xảy ra và cuối cùng dẫn đến hành động pháp lý có thể kéo dài trong nhiều năm. Trong thời gian chờ đợi, nhà phát triển phần mềm có thể đã phá sản vì không phân phối được phần cứng đã hứa gây ra tình trạng không phân phối được phần mềm đã hứa. Nói tóm lại, trong khi việc tạo mẫu giúp giảm thiểu rủi ro trong một số lĩnh vực, thì trong các lĩnh vực khác, nó tốt nhất là một câu trả lời một phần, và trong những lĩnh vực khác, nó hoàn toàn không phải là câu trả lời.

Mô hình xoắn ốc đầy đủ được thể hiện trong Hình 2.13. Kích thước hướng tâm đại diện cho chi phí tích lũy cho đến nay và kích thước góc thể hiện tiến trình thông qua hình xoắn ốc. Mỗi chu kỳ của vòng xoắn tương ứng với một pha. Một giai đoạn bắt đầu (ở góc phần tư trên cùng bên trái) bằng cách xác định các mục tiêu của giai đoạn đó, các lựa chọn thay thế để đạt được các mục tiêu đó và các ràng buộc áp đặt cho các lựa chọn thay thế đó. Quá trình này dẫn đến một chiến lược để đạt được các mục tiêu đó. Tiếp theo, chiến lược đó được phân tích từ quan điểm rủi ro. Các nỗ lực được thực hiện để giảm thiểu mọi rủi ro tiềm ẩn, trong một số trường hợp bằng cách xây dựng một nguyên mẫu. Nếu một số rủi ro không thể được giảm thiểu, dự án có thể bị chấm dứt ngay lập tức; Tuy nhiên, trong một số trường hợp, có thể đưa ra quyết định tiếp tục dự án nhưng ở quy mô nhỏ hơn đáng kể. Nếu tất cả các rủi ro được giảm thiểu thành công, bước phát triển tiếp theo sẽ được bắt đầu (góc phần tư phía dưới bên phải). Góc phần tư này của mô hình xoắn ốc tương ứng với mô hình thác nước cổ điển. Cuối cùng, kết quả của giai đoạn đó được đánh giá và giai đoạn tiếp theo được lập kế hoạch.

Mô hình xoắn ốc đã được sử dụng thành công để phát triển nhiều loại sản phẩm. Trong một tập hợp 25 dự án trong đó mô hình xoắn ốc được sử dụng cùng với các phương tiện tăng năng suất khác, năng suất của mọi dự án tăng ít nhất 50% so với các mức năng suất trước đó và 100% trong hầu hết các dự án [Boehm, 1988 ]. Để có thể quyết định xem mô hình xoắn ốc có nên được sử dụng cho một dự án nhất định hay không, điểm mạnh và điểm yếu của mô hình xoắn ốc bây giờ được đánh giá.

Mô hình xoắn ốc có một số điểm mạnh. Việc nhấn mạnh vào các giải pháp thay thế và ràng buộc hỗ trợ việc tái sử dụng phần mềm hiện có (Phần 8.1) và kết hợp chất lượng phần mềm như một mục tiêu cụ thể. Ngoài ra, một vấn đề phổ biến trong phát triển phần mềm là xác định thời điểm các sản phẩm của một giai đoạn cụ thể đã được kiểm tra kỹ lưỡng. Dành quá nhiều thời gian cho việc thử nghiệm là một sự lãng phí tiền bạc và việc giao sản phẩm có thể bị chậm trễ quá mức. Ngược lại, nếu quá ít thử nghiệm được thực hiện, thì phần mềm được phân phối có thể chứa các lỗi còn sót lại, dẫn đến hậu quả khó chịu cho các nhà phát triển. Mô hình xoắn ốc trả lời câu hỏi này về những rủi ro có thể xảy ra do không thực hiện đủ thử nghiệm hoặc thử nghiệm quá nhiều. Có lẽ quan trọng nhất, trong cấu trúc của mô hình xoắn ốc, việc bảo trì sau giao hàng chỉ đơn giản là một chu trình khác của vòng xoắn; về cơ bản không có sự phân biệt giữa duy trì và phát triển sau giao hàng. Do đó, vấn đề mà việc bảo trì sau giao hàng đôi khi bị các chuyên gia phần mềm thiếu hiểu biết đánh giá sai không phát sinh, bởi vì bảo trì sau giao hàng được xử lý giống như cách phát triển. 64+65

Có những hạn chế về khả năng áp dụng của mô hình xoắn ốc. Đặc biệt, ở dạng hiện tại, mô hình này chỉ dành riêng cho việc phát triển nội bộ phần mềm quy mô lớn [Boehm, 1988]. Hãy xem xét một dự án nội bộ, tức là một dự án trong đó các nhà phát triển và khách hàng là thành viên của cùng một tổ chức. Nếu phân tích rủi ro dẫn đến kết luận rằng dự án nên được chấm dứt, thì nhân viên phần mềm nội bộ có thể được chỉ định lại cho một dự án khác. Tuy nhiên, khi một hợp đồng đã được ký kết giữa một tổ chức phát triển và một khách hàng bên ngoài, nỗ lực của một trong hai bên nhằm chấm dứt hợp đồng đó có thể dẫn đến một vụ kiện vi phạm hợp đồng. Do đó, trong trường hợp phần mềm hợp đồng, tất cả các phân tích rủi ro phải được thực hiện bởi cả khách hàng và nhà phát triển trước khi hợp đồng được ký kết, không phải như trong mô hình xoắn ốc.

Hạn chế thứ hai đối với mô hình xoắn ốc liên quan đến quy mô của dự án. Đặc biệt, mô hình xoắn ốc chỉ có thể áp dụng cho phần mềm quy mô lớn. Sẽ không có ý nghĩa gì khi thực hiện phân tích rủi ro nếu chi phí thực hiện phân tích rủi ro có thể so sánh với chi phí của toàn bộ dự án hoặc nếu việc thực hiện phân tích rủi ro sẽ ảnh hưởng đáng kể đến tiềm năng lợi nhuận. Thay vào đó, trước tiên các nhà phát triển nên quyết định mức độ rủi ro và sau đó thực hiện phân tích rủi ro bao nhiêu, nếu có.

Một điểm mạnh chính của mô hình xoắn ốc là nó được định hướng bởi rủi ro, nhưng đây cũng có thể là một điểm yếu.

Trừ khi các nhà phát triển phần mềm có kỹ năng xác định chính xác các rủi ro có thể xảy ra và phân tích rủi ro một cách chính xác, có một nguy cơ thực sự mà nhóm có thể tin rằng tất cả đều ổn vào thời điểm mà trên thực tế, dự án đang đi đến thảm họa. Chỉ khi các thành viên của nhóm phát triển

các nhà phân tích rủi ro có năng lực nên ban lãnh đạo quyết định sử dụng mô hình xoắn ốc.

Tuy nhiên, nhìn chung, điểm yếu chính của mô hình xoắn ốc, cũng như mô hình thác nước và mô hình tạo mẫu nhanh, là nó giả định rằng phần mềm được phát triển theo các giai đoạn rời rạc. Tuy nhiên, trong thực tế, phát triển phần mềm là lặp đi lặp lại và tăng dần, như được phản ánh trong mô hình cây tiến hóa (Phần 2.2) hoặc mô hình lặp và tăng dần (Phần 2.5).

2.10 So sánh các mô hình vòng đời

Chín mô hình vòng đời phần mềm khác nhau đã được kiểm tra với sự chú ý đặc biệt đến một số điểm mạnh và điểm yếu của chúng. Nên tránh mô hình mã và sửa (Phần 2.9.1). Mô hình thác nước (Mục 2.9.2) là một đại lượng đã biết. Điểm mạnh của nó đã được hiểu rõ, và điểm yếu của nó cũng vậy. Mô hình tạo mẫu nhanh (Mục 2.9.3) được phát triển như một phản ứng đối với một điểm yếu được nhận thức cụ thể trong mô hình thác nước, cụ thể là sản phẩm được giao có thể không phải là thứ khách hàng thực sự cần. Tuy nhiên, vẫn chưa có đủ bằng chứng cho thấy cách tiếp cận này vượt trội hơn so với mô hình thác nước ở các khía cạnh khác. Mô hình vòng đời nguồn mở đã cực kỳ thành công trong một số ít trường hợp khi được sử dụng để xây dựng phần mềm cơ sở hạ tầng (Phần 2.9.4). Các quy trình Agile (Phần 2.9.5) là một tập hợp các phương pháp tiếp cận mới gây tranh cãi, cho đến nay, dường như hoạt động, nhưng chỉ đối với phần mềm quy mô nhỏ. Mô hình đồng bộ hóa và ổn định (Phần 2.9.6) đã được Microsoft sử dụng rất thành công, nhưng vẫn chưa có bằng chứng về sự thành công có thể so sánh được trong các nền văn hóa doanh nghiệp khác. Tuy nhiên, một giải pháp thay thế khác là sử dụng mô hình xoắn ốc (Phần 2.9.7), nhưng chỉ khi các nhà phát triển được đào tạo đầy đủ về phân tích rủi ro và giải quyết rủi ro. Mô hình cây tiến hóa (Phần 2.2) và mô hình lặp lại và tăng dần (Phần 2.5) gần nhất với cách mà phần mềm được sản xuất trong thế giới thực. So sánh tổng thể xuất hiện trong Hình 2.14.

Mỗi tổ chức phát triển phần mềm nên quyết định mô hình vòng đời phù hợp với tổ chức đó, ban quản lý, nhân viên và quy trình phần mềm của tổ chức đó và nên thay đổi mô hình vòng đời tùy thuộc vào các tính năng của sản phẩm cụ thể hiện đang được phát triển. Một mô hình như vậy kết hợp các khía cạnh thích hợp của các mô hình vòng đời khác nhau, sử dụng điểm mạnh và giảm thiểu điểm yếu của chúng. 66

Đánh giá chương

Có sự khác biệt rõ rệt giữa cách mà phần mềm được phát triển trên lý thuyết (Phần 2.1) và cách nó được phát triển trong thực tế. Nghiên cứu trường hợp nhỏ Winburg được sử dụng để giới thiệu mô hình cây tiến hóa (Phần 2.2). Các bài học của nghiên cứu điển hình nhỏ này, đặc biệt là các yêu cầu thay đổi, được trình bày trong Phần 2.3. Thay đổi được thảo luận chi tiết hơn trong Phần 2.4, nơi bài toán mục tiêu di chuyển được trình bày bằng cách sử dụng nghiên cứu điển hình nhỏ về Máy kéo Teal. Trong Phần 2.5, tầm quan trọng của việc lặp lại và tăng dần trong kỹ thuật phần mềm thế giới thực được nhấn mạnh, và mô hình lặp lại và tích lũy được trình bày. Sau đó, nghiên cứu trường hợp nhỏ của Winburg được kiểm tra lại trong Phần 2.6 để minh họa sự tương đương của mô hình tự nhiên và mô hình lặp lại và tăng dần. Trong Phần 2.7, các điểm mạnh của mô hình lặp đi lặp lại và mô hình cộng dồn được trình bày, đặc biệt là nó cho phép chúng ta giải quyết các rủi ro sớm. Quản lý mô hình lặp lại và tăng dần được thảo luận trong Phần 2.8. Một số mô hình vòng đời khác nhau hiện đã được mô tả, bao gồm mô hình vòng đời mã và sửa (Phần 2.9.1), mô hình vòng đời thác nước (Phần 2.9.2), mô hình vòng đời tạo mẫu nhanh ( Phần 2.9.3), mô hình vòng đời nguồn mở (Phần 2.9.4), các quy trình nhanh (Phần 2.9.5), mô hình vòng đời đồng bộ hóa và ổn định (Phần 2.9.6) và vòng đời xoắn ốc mô hình (Mục 2.9.7). Trong Phần 2.10, các mô hình vòng đời này được so sánh và đưa ra các đề xuất liên quan đến việc lựa chọn mô hình vòng đời cho một dự án cụ thể. 67

Để đọc thêm

Mô hình thác nước lần đầu tiên được đưa ra vào [Royce, 1970]. Một phân tích về mô hình thác nước được đưa ra trong chương đầu tiên của [Royce, 1998].

Mô hình đồng bộ hóa và ổn định được nêu trong [Cusumano và Selby, 1997] và được mô tả chi tiết trong [Cusumano và Selby, 1995]. Mô hình xoắn ốc được giải thích trong [Boehm, 1988], và ứng dụng của nó cho Hệ thống Năng suất Phần mềm TRW xuất hiện trong [Boehm và cộng sự, 1984].

Lập trình cực đoan được mô tả trong [Beck, 2000]; tái cấu trúc là chủ đề của [Fowler và cộng sự, 1999]. Tuyên ngôn về Phát triển Phần mềm Agile có thể được tìm thấy tại [Beck và cộng sự, 2001]. Sách đã được xuất bản về nhiều phương pháp nhanh nhẹn, bao gồm [Cockburn, 2001] và [Schwaber, 2001]. Các phương pháp Agile được ủng hộ trong [Highsmith và Cockburn, 2001], [Boehm, 2002], [DeMarco và Boehm, 2002], và [Boehm và Turner, 2003], trong khi trường hợp chống lại các phương pháp nhanh được trình bày trong [Stephens và Rosenberg, 2003]. Tái cấu trúc được khảo sát trong [Mens và Tourwe, 2004]. Việc sử dụng XP trong bốn dự án quan trọng được mô tả trong [Drobka, Noftz và Raghu, 2004]. Các vấn đề có thể nảy sinh khi giới thiệu các quy trình nhanh trong một tổ chức hiện đang sử dụng các phương pháp luận truyền thống được thảo luận trong [Nerur, Mahapatra và Mangalaraj, 2005] và [Boehm và Turner, 2005].

Một số bài báo về lập trình cực đoan xuất hiện trong số tháng 5 năm 2003 của IEEE Software, bao gồm [Murru, Deias, và Mugheddu, 2003] và [Rasmusson, 2003], cả hai đều mô tả các dự án thành công được phát triển bằng cách sử dụng lập trình cực đoan. Số tháng 6 năm 2003 của IEEE Computer có một số bài báo về các quy trình nhanh. Số tháng 5 đến tháng 6 năm 2005 của IEEE Software có bốn bài báo về các quy trình nhanh, đặc biệt là [Ceschi, Sillitti, Succi, và De Panfi lis, 2005] và [Karlström và Runeson, 2005]. Mức độ mà các phương pháp nhanh được sử dụng trong ngành công nghiệp phần mềm được phân tích trong [Hansson, Dittrich, Gustafsson, và Zarnak, 2006]. Một cuộc khảo sát về các yếu tố thành công quan trọng trong các sản phẩm phần mềm nhanh được trình bày trong [Chow và Cao, 2008]. Các phương pháp hỗ trợ chuyển đổi sang các phương pháp nhanh được đưa ra trong [Qumer và Henderson-Sellers, 2008]. Tái cấu trúc đặt ra các vấn đề đối với các công cụ quản lý kiểm duyệt phần mềm; một giải pháp được đưa ra trong [Dig, Manzoor, Johnson, and Nguyen, 2008]. Kiểm thử nhanh một sản phẩm phần mềm quy mô lớn được mô tả trong [Talby, Keren, Hazzan và Dubinsky, 2006]. Hiệu quả của phát triển theo hướng thử nghiệm được thảo luận trong [Erdogmus, Morisio và Torchiano, 2005]. Số tháng 5 đến tháng 6 năm 2007 của IEEE Software có nhiều bài viết về phát triển theo hướng thử nghiệm, bao gồm [Martin, 2007].

Phân tích rủi ro được mô tả trong [Ropponen và Lyttinen, 2000], [Longstaff, Chittister, Pethia, và Haimes, 2000], và [Scott và Vessey, 2002]. Quản lý rủi ro trong phát triển phần mềm ra nước ngoài được trình bày trong [Sakthivel, 2007] và [Iacovou và Nakatsu, 2008]. Quản lý rủi ro khi phần mềm được phát triển bằng cách sử dụng các thành phần COTS được mô tả trong [Li và cộng sự, 2008]. Một mô hình lặp lại và gia tăng chính được mô tả chi tiết trong [Jacobson, Booch, và Rumbaugh, 1999]. Tuy nhiên, nhiều mô hình lặp đi lặp lại và gia tăng khác đã được đưa ra trong 30 năm qua, như được kể lại trong [Larman và Basili, 2003]. Việc sử dụng mô hình gia tăng để xây dựng hệ thống điều khiển đường hàng không được thảo luận trong [Goth, 2000]. Một cách tiếp cận lặp đi lặp lại để tái thiết kế các hệ thống kế thừa được đưa ra trong [Bianchi, Caivano, Marengo và Visaggio, 2003]. Một công cụ để hỗ trợ phát triển phần mềm gia tăng trong khi đảm bảo rằng các tạo tác phát triển nhất quán được mô tả trong [Reiss, 2006].

Nhiều mô hình vòng đời khác đã được đưa ra. Ví dụ, Rajlich và Bennett [2000] mô tả một mô hình vòng đời theo định hướng bảo trì. Số tháng 7 đến tháng 8 năm 2000 của IEEE Software có nhiều bài báo về các mô hình vòng đời phần mềm, bao gồm [Williams, Kessler, Cunningham và Jeffries, 2000] mô tả một thử nghiệm về lập trình cặp, một thành phần của các phương pháp nhanh.

Rajlich [2006] còn đi xa hơn và gợi ý rằng nhiều chủ đề của chương này đã đưa chúng ta đến một mô hình mới cho kỹ thuật phần mềm.

Các thủ tục của Hội thảo Quy trình Phần mềm Quốc tế là một nguồn thông tin hữu ích về các mô hình vòng đời. [ISO / IEC 12207, 1995] là tiêu chuẩn được chấp nhận rộng rãi cho các quy trình vòng đời phần mềm. 68