CHƯƠNG 3

Sau khi nghiên cứu chương này, bạn sẽ có thể

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

  • Hiểu được các mô hình vòng đời phát triển cây.

  • Đánh giá cao những tác động tiêu cực của biến đổi trên các sản phẩm phần mềm.

  • Sử dụng các mô hình vòng đời lặp đi lặp lại-và-gia tăng.

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

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

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

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

  • So sánh và tương phản của 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. Các Chủ đề của chương này là những gì xảy ra trong thực tế. Như sẽ được giải thích, có những 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 mô tả sơ đồ trong hình 2.1, hệ thống được phát triển từ đầu; biểu thị sự trống rỗng bộ. (Xem Chỉ trong trường hợp bạn muốn biết Box 2.1 nếu bạn muốn biết nguồn gốc của hạn từ đầu.) Trước yêu cầu của khách hàng được xác định, và sau đó phân tích được thực hiện. Khi các đồ tạo tác phân tích đầy đủ, thiết kế được sản xuất. Đây là

tiếp theo là thực hiện của các sản phẩm phần mềm hoàn chỉnh, sau đó được cài đặt

trên máy tính của khách hàng. Tuy nhiên, phát triển phần mềm là khác nhau đáng kể trong thực tế vì hai lý do.

* Đầu tiên, các chuyên gia phần mềm cũng là con người và do đó làm cho những sai lầm.
* Thứ hai, của khách hàng yêu cầu có thể thay đổi trong khi các phần mềm đang được phát triển.

Trong chương này, cả hai vấn đề được thảo luận trong một số sâu, nhưng fi đầu tiên chúng tôi trình bày một trường hợp nghiên cứu nhỏ, dựa trên các trường hợp nghiên cứu trong [Tom và Schach, 2000], để minh họa các vấn đề liên quan.

Just in Case You Wanted to Know Box 2.1

Chỉ trong trường hợp bạn muốn biết Box 2.1

Thuật ngữ từ đầu, có nghĩa là "bắt đầu với không có gì," xuất phát từ thể thao thế kỷ 19

thuật ngữ. Trước khi đường (và đường chạy đua) đã được trải nhựa, các chủng tộc đã được tổ chức vào ngày mở đất. Trong nhiều trường hợp, các dòng bắt đầu là một vết xước trên cát. Một Á hậu người không có lợi thế hay bất lợi đã phải bắt đầu từ dòng đó, đó là, "từ [các] đầu." Các đầu dài có một ý nghĩa thể thao khác nhau hiện nay. Một "đầu tay golf" là một mà chơi golf handicap là số không.

**2.2 Winburg Mini Case Study**

Để giảm ùn tắc giao thông ở trung tâm thành phố Winburg, Indiana, thị trưởng thuyết phục thành phố để thiết lập một hệ thống giao thông công cộng. làn đường xe buýt được thành lập, và hành khách sẽ được khuyến khích "công viên và đi xe"; đó là, đỗ xe ở bãi đỗ xe ở ngoại ô và sau đó đi xe buýt từ đó để làm việc và trở lại với chi phí một đô la cho mỗi chuyến đi. Mỗi xe buýt là phải có một máy tính giá vé chấp nhận hóa đơn chỉ.1 đô la Hành khách chèn một hóa đơn vào khe khi họ lên xe buýt. Cảm biến bên trong giá vé máy quét hóa đơn, và các phần mềm trong máy sử dụng một hình ảnh ghi nhận thuật toán để quyết định xem các hành khách đã thực sự chèn một đồng đô la giá trị vào khe. Điều quan trọng là máy tính giá vé chính xác bởi vì, một khi tin tức được ra rằng bất kỳ mẩu giấy sẽ làm các lừa, thu nhập giá vé sẽ giảm mạnh để có hiệu quả không. Ngược lại, nếu máy tính thường xuyên từ chối hóa đơn đô la giá trị, hành khách sẽ được miễn cưỡng sử dụng xe buýt. Ngoài ra, các máy giá vé phải được nhanh chóng. hành khách sẽ được bình đẳng không muốn sử dụng những chiếc xe buýt nếu máy dành 15 giây đến một quyết định về tính hợp lệ của một đồng đô la hóa đơn đó sẽ mất ngay cả một tương đối nhỏ số lượng hành khách nhiều phút để lên xe buýt. Do đó, các yêu cầu về phần mềm máy giá vé bao gồm thời gian đáp ứng trung bình ít hơn 1 giây và

độ chính xác trung bình ít nhất 98 phần trăm.

Episode 1 Phiên bản đầu tiên của phần mềm được thực hiện.

Episode 2 xét nghiệm cho thấy rằng ràng buộc cần thiết của một thời gian phản ứng trung bình của 1 giây để quyết định về tính hợp lệ của một đồng đô la là không đạt được. Trong thực tế, trên trung bình, phải mất 10 giây để có được một phản ứng. Quản lý cấp cao phát hiện ra nguyên nhân. Có vẻ như rằng, để có được sự chính xác 98 phần trăm yêu cầu, một lập trình viên đã được hướng dẫn bởi quản lý của cô để sử dụng con số tăng gấp đôi độ chính xác cho tất cả các phép tính toán học. Kết quả là, mọi hoạt động phải mất ít nhất hai lần miễn là nó làm với các con số chính xác đơn thông thường. Kết quả là chương trình là chậm hơn nhiều so, dẫn đến thời gian phản ứng dài. Các tính toán sau đó cho thấy rằng, mặc dù những gì người quản lý nói với các lập trình viên, các quy định độ chính xác 98 phần trăm có thể đạt được thậm chí nếu con số chính xác đơn được sử dụng. Các lập trình viên bắt đầu làm việc thay đổi cần thiết để thực hiện.

Episode 3 Trước khi các lập trình viên có thể hoàn thành công việc của mình, kiểm tra hơn nữa của hệ thống cho thấy, ngay cả khi những thay đổi chỉ định để thực hiện được thực hiện, hệ thống vẫn sẽ có một thời gian phản ứng trung bình hơn 4,5 giây, không nơi nào gần các quy định 1 giây. Vấn đề là các thuật toán nhận dạng hình ảnh phức tạp.

May mắn thay, một thuật toán nhanh hơn vừa được phát hiện, vì vậy các phần mềm máy giá vé được thiết kế lại và thực hiện lại bằng cách sử dụng thuật toán mới. Điều này dẫn đến trung bình thời gian phản ứng đạt được thành công.

Episode 4:

Đến nay, dự án chậm tiến độ và cách trên ngân sách là đáng kể. Các thị trưởng, một doanh nhân thành đạt, có ý tưởng sáng của yêu cầu phần mềm nhóm phát triển để cố gắng để tăng độ chính xác của việc ghi nhận hóa đơn đô la là thành phần của hệ thống càng nhiều càng tốt, để bán các gói kết quả cho các công ty máy bán hàng tự động. Để đáp ứng yêu cầu mới này, một thiết kế mới được thông qua đó cải thiện độ chính xác trung bình tới hơn 99,5 phần trăm. Quản lý quyết định cài đặt là phiên bản của phần mềm trong máy giá vé. Tại điểm này, phát triển của phần mềm là hoàn tất. Thành phố này là sau đó có thể bán hệ thống của nó cho hai công ty máy bán hàng tự động nhỏ, Trả chi khoảng một phần ba chi phí

Epilogue

Một vài năm sau đó, các cảm biến bên trong máy giá vé trở nên lỗi thời và cần phải được thay thế bằng một mô hình mới hơn. Quản lý cho thấy lợi ích của sự thay đổi để nâng cấp phần cứng cùng một lúc. Các chuyên gia phần mềm chỉ ra rằng việc thay đổi phần cứng có nghĩa là phần mềm mới cũng cần thiết. họ đề nghị thực hiện lại các phần mềm trong một ngôn ngữ lập trình khác nhau. Tại thời điểm này, dự án là chậm 6 tháng so với kế hoạch và vượt 25 phần trăm trong ngân sách. Tuy nhiên, tất cả mọi người tham gia là tự tin rằng hệ thống mới sẽ được tin cậy hơn và có chất lượng cao hơn, mặc dù "khác biệt nhỏ" trong việc đáp ứng thời gian phản ứng của nó và yêu cầu độ chính xác.

Hình 2.2 mô tả sự phát triển cây mô hình vòng đời của nghiên cứu nhỏ. Các hộp bên trái đại diện cho Tập 1. Như trong hình, hệ thống đã được phát triển từ đầu (). Các yêu cầu (yêu cầu 1), phân tích (phân tích 1), thiết kế (Thiết kế 1), và thực hiện (thực hiện 1) tiếp theo lần lượt. Tiếp theo, như mô tả trước đây, các thử nghiệm của phiên bản đầu tiên của phần mềm đã cho thấy rằng trung bình thời gian đáp ứng của 1 giây không thể đạt được và thực hiện phải được sửa đổi. Việc thực hiện biến đổi xuất hiện trong hình 2.2 như thực hiện 2. Tuy nhiên, thực hiện 2 không bao giờ được hoàn thành. Đó là lý do tại sao các hình chữ nhật biểu diễn Thực hiện 2 được vẽ bằng một đường chấm chấm.

Trong tập 3, các thiết kế đã được thay đổi. Cụ thể, sự công nhận hình ảnh nhanh hơn với

Thuật toán được sử dụng Các thiết kế sửa đổi (Thiết kế 3) dẫn đến một thực hiện sửa đổi (Thực hiện 3). Cuối cùng, trong tập 4, các yêu cầu đã được thay đổi (Yêu cầu 4) để tăng

sự chính xác Điều này dẫn đến chi tiết kỹ thuật sửa đổi (Phân tích 4), sửa đổi thiết kế (thiết kế 4), và thực hiện sửa đổi (thực hiện 4). Trong hình 2.2, các mũi tên biểu thị rắn phát triển và các mũi tên đứt đoạn biểu bảo trì. Ví dụ, khi thiết kế được thay đổi trong tập 3, Thiết kế 3 thay thế thiết kế 1 như thiết kế của Phân tích 1.

Các mô hình phát triển cây là một ví dụ về **một mô hình vòng đời** (hoặc mô hình,cho ngắn), đó là, một loạt các bước được thực hiện trong khi các sản phẩm phần mềm phát triển và duy trì. Một mô hình vòng đời có thể được sử dụng cho các trường hợp nghiên cứu mini là **mô hình thác nước vòng đời** [Royce, 1970]; một phiên bản đơn giản của mô hình thác nước được mô tả trong hình 2.3. mô hình vòng đời này cổ điển có thể được xem như là mô hình tuyến tính ở hình 2.1 với các vòng phản hồi. Sau đó, nếu như sự cố là được tìm thấy trong các thiết kế đã được gây ra bởi một lỗi trong các yêu cầu, sau tan mũi tên hướng lên, các nhà phát triển phần mềm có thể quay lại từ các thiết kế lên đến phân tích và do đó các yêu cầu và thực hiện những sửa chữa cần thiết đó. Sau đó, chúng di chuyển xuống để phân tích, sửa tài liệu đặc tả để phản ánh sự điều chỉnh các yêu cầu, và đến lượt nó, sửa tài liệu thiết kế. thiết kế hoạt động bây giờ có thể tiếp tục mà họ đã bị đình chỉ khi các lỗi được phát hiện. Một lần nữa, các mũi tên rắn biểu thị sự phát triển; các mũi tên đứt, bảo trì.

Các mô hình thác nước chắc chắn có thể được sử dụng để đại diện cho các trường hợp nghiên cứu nhỏ Winburg, nhưng, không giống như các mô hình phát triển cây của hình 2.2, nó có thể không hiển thị thứ tự của các sự kiện. Các mô hình phát triển cây có lợi thế hơn so với mô hình thác nước. Cuối cùng của mỗi tập phim, chúng tôi có **một cơ sở(baseline)**, đó là một bộ hoàn chỉnh các đồ tạo tác (nhớ lại rằng một vật là một thành phần cấu thành của một sản phẩm phần mềm). Có bốn đường cơ sở trong hình 2.2. Họ là

At the end of Episode 1: Requirements 1 , Analysis 1 , Design 1, Implementation 1

At the end of Episode 2: Requirements 1, Analysis 1, Design 1, Implementation 2

At the end of Episode 3: Requirements 1, Analysis 1, Design 3, Implementation 3

At the end of Episode 4: Requirements 4, Analysis 4, Design 4, Implementation 4

Các cơ sở đầu tiên là thiết lập ban đầu của hiện vật; đường cơ sở thứ hai phản ánh sự biến đổi(Nhưng không bao giờ hoàn thành) Thực hiện 2 Episode 2, cùng với sự thay đổi yêu cầu, phân tích và thiết kế của Tập 1. Các cơ sở thứ ba là giống như cơ sở đầu tiên, nhưng với việc thiết kế và thực hiện thay đổi. Các cơ sở thứ tư là bộ hoàn chỉnh các hiện vật mới thể hiện trong hình 2.2. Chúng tôi xem xét lại khái niệm về một cơ sở ở Chương 5 và 16.

**2.3 Lessons of the Winburg Mini Case Study**

Các Winburg nghiên cứu trường hợp nhỏ mô tả sự phát triển của một sản phẩm phần mềm mà đi xiên đối với một số nguyên nhân không liên quan, chẳng hạn như một chiến lược thực hiện kém (không cần thiết bị sử dụng hai chính xác số) và quyết định sử dụng một thuật toán mà là quá chậm. Cuối cùng, dự án đã thành công. Tuy nhiên, câu hỏi rõ ràng là, là phát triển phần mềm thực sự là hỗn loạn trong thực tế? Trong thực tế, các nghiên cứu trường hợp nhỏ là ít chấn thương hơn nhiều, nếu không phải là đa số, các dự án phần mềm. Trong trường hợp nhỏ nghiên cứu Winburg, đã có chỉ có hai phiên bản mới của phần mềm vì lỗi (việc sử dụng không phù hợp của độ chính xác kép số; việc sử dụng một thuật toán mà không thể đáp ứng thời gian đáp ứng yêu cầu) và chỉ có một phiên bản mới vì sự thay đổi được thực hiện bởi các khách hàng (nhu cầu để tăng độ chính xác).

Tại sao rất nhiều thay đổi cho một sản phẩm phần mềm cần thiết? Thứ nhất, như đã nói trước đây, các chuyên gia phần mềm cũng là con người và do đó làm cho những sai lầm. Thứ hai, một sản phẩm phần mềm là một mô hình của thế giới thực, và thế giới thực liên tục thay đổi. Vấn đề này được thảo luận tại chiều dài lớn hơn tại mục 2.4.

**2.4 Teal Tractors Mini Case Study**

Teal Máy kéo, Inc., bán máy kéo ở hầu hết các vùng của Hoa Kỳ. Công ty đã yêu cầu bộ phận phần mềm của mình để 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 bán hàng, hàng tồn kho, và

hoa hồng trả cho các nhân viên kinh doanh, cũng như cung cấp tất cả các kế toán cần thiết

chức năng. Trong khi sản phẩm phần mềm này đang được thực hiện, Teal Máy kéo mua một Công ty máy kéo Canada. Việc quản lý của Teal Máy kéo 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 các phần mềm đã được thay đổi trước khi nó được 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 các loại thuế.

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

USD

Teal Máy kéo là một công ty phát triển nhanh chóng với triển vọng xuất sắc trong tương lai. Các tiếp quản của các công ty máy kéo Canada là một sự phát triển tích cực, một trong đó có thể cũng dẫn đến lợi nhuận lớn hơn trong những năm tới. Tuy nhiên, từ quan điểm của các bộ phận phần mềm, mua bán của các công ty Canada có thể là một thảm họa. Trừ khi các yêu cầu, phân tích và thiết kế đã được thực hiện với mục đích kết hợp các phần mở rộng trong tương lai, các công việc liên quan trong việc thêm các khu vực bán hàng của Canada có thể rất tuyệt vời mà nó có thể có hiệu quả hơn để loại bỏ tất cả mọi thứ được thực hiện cho đến nay và bắt đầu từ đầu. Lý do là thay đổi các sản phẩm ở giai đoạn này là tương tự như cố gắng để sửa chữa một sản phẩm phần mềm cuối trong vòng đời của nó (xem hình 1.6). Mở rộng các phần mềm để xử lý các khía cạnh cụ thể cho các thị trường Canada, cũng như tiền Canada, có thể là khó kể.

Thậm chí nếu các phần mềm cũng đã được nghĩ ra và thiết kế ban đầu thực sự là mở rộng, thiết kế của sản phẩm vá-cùng nhau kết quả không thể được như cố kết như nó đã có thể nó đã được phát triển từ đầu để phục vụ cho cả Hoa Kỳ và Canada. Điều này có thể có những tác động nghiêm trọng để bảo trì trong tương lai.

  Các bộ phận phần mềm của Teal Máy kéo là một nạn nhân của **vấn đề di chuyển, mục tiêu (moving-target problem)**Vững đó là, trong khi phần mềm đang được phát triển, các yêu cầu thay đổi. Nó không quan trọng đó là lý do cho sự thay đổi là nếu không vô cùng đáng giá. Thực tế là sự tiếp quản của công ty Canada cũng có thể gây hại đến chất lượng của phần mềm được phát triển.

Trong một số trường hợp, lý do cho các mục tiêu di động là ít nhẹ. Đôi khi một người quản lý cao cấp quyền lực trong một tổ chức luôn thay đổi của mình hoặc tâm trí của cô liên quan đến các chức năng của một sản phẩm phần mềm được phát triển. Trong trường hợp khác, có **tính năng leo(feature creep)**, một kế nhỏ,bổ sung yêu cầu, gần như tầm thường.Nhưng bất cứ lý do có thể, thay đổi thường xuyên, không có vấn đề như thế nào nhỏ họ có thể có vẻ, có hại cho sức khỏe của một sản phẩm phần mềm. Điều quan trọng là một sản phẩm phần mềm được thiết kế như là một tập hợp các thành phần có thể độc lập càng tốt, do đó, một sự thay đổi đối với một phần của phần mềm không gây ra một lỗi trong một bộ phận dường như không liên quan đến phần có lỗi này, được gọi là **lỗi quy hồi(regression fault).** Khi nhiều thay đổi được thực hiện, hiệu quả là để tạo ra phụ thuộc trong các mã. Cuối cùng, có rất nhiều người phụ thuộc mà hầu như bất kỳ sự thay đổi gây ra một hoặc nhiều lỗi hồi quy. Tại thời điểm đó, điều duy nhất có thể làm là để thiết kế lại toàn bộ sản phẩm phần mềm và thực hiện nó

Thật không may, không có giải pháp được biết đến vấn đề di chuyển mục tiêu. đối với với những thay đổi tích cực cho các yêu cầu, các công ty đang phát triển luôn sẽ thay đổi, và những thay đổi này phải được phản ánh trong các sản phẩm phần mềm nhiệm vụ quan trọng của công ty. Đối với những thay đổi tiêu cực, nếu gọi điện thoại cá nhân cho những thay đổi đó có đủ sức mạnh, không có gì có thể được thực hiện để ngăn chặn những thay đổi được thực hiện, gây thiệt hại cho khả năng bảo trì hơn nữa của các sản phẩm phần mềm

**2.5 Iteration and Incrementation**

Như một hệ quả của cả hai vấn đề di chuyển mục tiêu và sự cần thiết phải sửa chữa những sai lầm không thể tránh khỏi trong khi một sản phẩm phần mềm đang được phát triển, chu kỳ sống của sản phẩm phần mềm thực tế giống như các mô hình phát triển cây của Hình 2.2 hoặc mô hình thác nước của Hình 2.3 , chứ không phải là chuỗi lý tưởng hóa của Hình 2.1. Một hệ quả của thực tế này là nó không có ý nghĩa nhiều để nói về (nói) "giai đoạn phân tích." Thay vào đó, các hoạt động của giai đoạn phân tích được trải ra trong chu kỳ cuộc sống. Tương tự, hình 2.2 cho thấy bốn phiên bản khác nhau của thực, một trong số đó (Thực hiện 2) không bao giờ được hoàn thành bởi vì các vấn đề di chuyển mục tiêu.

Hãy xem xét các phiên bản kế tiếp của một vật, ví dụ, các tài liệu kỹ thuật hoặc một mô-đun mã. Từ quan điểm này, quá trình cơ bản là lặp đi lặp lại. Đó là, chúng tôi sản xuất các phiên bản đầu tiên của hiện vật, sau đó chúng tôi sửa lại nó và sản xuất phiên bản thứ hai, và như vậy. Mục đích của chúng tôi là mỗi phiên bản là gần hơn với mục tiêu của chúng tôi so với người tiền nhiệm của nó và cuối cùng chúng tôi xây dựng một phiên bản đó là thỏa đáng. **Lặp đi lặp lại (iteration)** là một khía cạnh nội tại của công nghệ 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, mà lần đầu tiên đưa ra vào năm 1970, là lặp đi lặp lại (nhưng không gia tăng).

**Các yếu tố nội tại là những yếu tố bên trong có thể kiểm soát, điều chỉnh được và chúng có ảnh hưởng trực tiếp đến hoạt động và kết quả hoạt động**

Một khía cạnh thứ hai của phát triển phần mềm thực tế là hạn chế đối với chúng tôi bởi **Luật Miller(Miller’s Law).** Năm 1956, George Miller, một giáo sư về tâm lý học, cho thấy, tại một thời điểm, con người chúng ta có khả năng tập trung vào chỉ khoảng bảy khối (đơn vị thông tin) [Miller, 1956]. Tuy nhiên, một phần mềm tạo tác điển hình có nhiều hơn bảy khối. Ví dụ, một mã vật có thể có nhiều hơn hẳn so với bảy biến và một văn bản yêu cầu có thể có nhiều hơn bảy yêu cầu. Một cách con người chúng ta xử lý các hạn chế này vào lượng thông tin chúng tôi có thể xử lý tại một thời điểm là sử dụng **tinh tế từng bước(stepwise refinement .).** Đó là, chúng tôi tập trung vào những khía cạnh hiện đang quan trọng nhất và trì hoãn cho đến khi sau những khía cạnh mà hiện nay là ít quan trọng. Nói cách khác, mọi khía cạnh cuối cùng cũng xử lý nhưng theo thứ tự tầm quan trọng hiện nay. Điều này có nghĩa rằng chúng tôi bắt đầu bằng việc xây dựng một artifact mà giải quyết chỉ là một phần nhỏ của những gì chúng tôi đang cố gắng để đạt được. Sau đó, chúng ta xem xét khía cạnh của vấn đề và thêm kết quả phần mới để tạo tác hiện có. Ví dụ, chúng ta có thể xây dựng một tài liệu yêu cầu bằng cách xem xét bảy yêu cầu chúng tôi xem xét quan trọng nhất. Sau đó, chúng tôi sẽ xem xét bảy yêu cầu quan trọng nhất tiếp theo, và như vậy. Đây là quá trình gia tăng. **Incrementation** cũng là một khía cạnh nội tại của công nghệ phần mềm; phát triển phần mềm gia tăng là hơn 45 tuổi [Larman và Basili, 2003].

Trong thực tế, lặp đi lặp lại và incrementation được sử dụng kết hợp với nhau. Đó là, một vật được xây dựng từng mảnh (incrementation), và mỗi số gia đi qua nhiều phiên bản (lặp lại). Những ý tưởng này được minh họa trong hình 2.2, đại diện cho vòng đời cho Winburg nhỏ trường hợp nghiên cứu (mục 2.2 và 2.3). Như thể hiện trong hình đó, không có đơn "yêu cầu giai đoạn" như vậy. Thay vào đó, các yêu cầu của khách hàng được trích lập và phân tích hai lần, đạt các yêu cầu ban đầu (Yêu cầu 1) và yêu cầu sửa đổi (Yêu cầu 4). Tương tự, không có "giai đoạn thực hiện," riêng rẽ mà là bốn tập riêng biệt trong đó các mã được sản xuất và sau đó sửa đổi.

Những ý tưởng này được tổng quát trong hình 2.4, trong đó phản ánh các khái niệm cơ bản nằm dưới **vòng đời mô hình lặp đi lặp lại-và-gia tăng (iterative-and-incremental life-cycle model)**[Jacobson, Booch, và Rumbaugh, 1999]. Hình này cho thấy sự phát triển của một sản phẩm phần mềm trong bốn gia, nhãn Tăng A, Tăng B, Tăng C, và Tăng D. Trục ngang là thời gian, và trục thẳng đứng là người-giờ (một người giờ là lượng làm việc mà một người có thể làm trong 1 giờ), vì vậy các khu vực bóng mờ dưới mỗi đường cong là tổng sức cho tăng đó.

Điều quan trọng là phải hiểu rằng Hình 2.4 mô tả chỉ là một cách có thể mềm sản phẩm một phần có thể được phân rã thành từng bước. Một sản phẩm phần mềm có thể được xây dựng chỉ trong 2 increments, trong khi một phần ba có thể yêu cầu 14. Hơn nữa, con số này không có ý định trở thành một đại diện chính xác chính xác làm thế nào một sản phẩm phần mềm được phát triển. Thay vào đó, nó cho thấy sự nhấn mạnh thay đổi từ lặp đi lặp lại để lặp.

Các giai đoạn tuần tự của Hình 2.1 là cấu trúc nhân tạo. Thay vào đó, như một cách rõ ràng

phản ánh trong hình 2.4, chúng ta phải thừa nhận rằng công việc khác nhau (các hoạt động) được thực hiện trên toàn bộ vòng đời. Có 5 **công việc năm lõi(core workfollow)**, công việc yêu cầu, phân tích công việc, công việc thiết kế, công việc triển khai, quy trình làm việc và kiểm tra, và, như đã nói trong câu trước đó, tất cả năm được thực hiện trong vòng đời của một sản phẩm phần mềm. Tuy nhiên, có những lúc một quy trình làm việc chiếm ưu thế hơn trong bốn cái còn lại

Ví dụ, vào đầu của vòng đời, các nhà phát triển phần mềm trích xuất một thiết lập ban đầu của yêu cầu. Nói cách khác, vào lúc bắt đầu của chu kỳ cuộc sống lặp đi lặp lại-và-gia tăng, các công việc yêu cầu chiếm ưu thế. Những yêu cầu này hiện vật được mở rộng và modifi ed trong phần còn lại của chu kỳ cuộc sống. Trong thời gian đó, bốn công việc khác (phân tích, thiết kế, thực hiện và kiểm tra) chiếm ưu thế. Nói cách khác, các công việc yêu cầu là công việc chủ yếu ở phần đầu của chu kỳ cuộc sống, nhưng tầm quan trọng của nó giảm đi sau đó. Ngược lại, việc triển khai và kiểm tra quy trình công việc chiếm nhiều thời gian của các thành viên trong nhóm phát triển phần mềm hướng tới sự kết thúc của chu kỳ cuộc sống hơn là lúc đầu.

Kế hoạch hoạt động và tài liệu được thực hiện trong suốt chu kỳ cuộc sống lặp đi lặp lại và gia tăng. Hơn nữa, thử nghiệm là một hoạt động quan trọng trong mỗi lần lặp, và đặc biệt là ở phần cuối của mỗi lần lặp. Ngoài ra, các phần mềm như một toàn thể được kiểm tra kỹ lưỡng một khi nó đã được hoàn thành; tại thời điểm đó, thử nghiệm và sau đó thay đổi thực hiện trong ánh sáng của các kết quả của các xét nghiệm khác nhau là hầu như các hoạt động duy nhất của nhóm phần mềm. Đây là refl ected trong workfl kiểm tra ow của Hình 2.4..

Hình 2.4 cho thấy bốn gia. Hãy xem xét Tăng A, được mô tả bởi các cột trên Bên trái. Tại đầu của tăng này, các thành viên yêu cầu nhóm xác định các yêu cầu của khách hàng. Khi hầu hết các yêu cầu đã được xác định, phiên bản đầu tiên của phần của phân tích có thể được bắt đầu. Khi đủ tiến bộ đã được thực hiện với sự 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 quá trình tăng đầu tiên fi này, có lẽ trong các hình thức của một bằng chứng của khái niệm nguyên mẫu để kiểm tra tính khả thi của một phần của các sản phẩm phần mềm được đề nghị. Cuối cùng, như đã đề cập trước đó, lập kế hoạch, kiểm tra, và các hoạt động tài liệu bắt đầu vào ngày thứ nhất và tiếp tục từ đó về sau, cho đến khi các sản phẩm phần mềm cuối cùng được giao cho khách hàng.

Tương tự như vậy, sự tập trung chính trong Tăng B là trên các yêu cầu và quy trình công việc phân tích, và sau đó vào các công việc thiết kế. Sự nhấn mạnh trong Tăng C là đầu tiên trên công việc thiết kế, và sau đó vào các công việc thực hiện và kiểm tra workflow.Finally, trong Tăng D, công việc thực hiện và kiểm tra công việc chiếm ưu thế.

Theo phản ánh trong hình 1.4, khoảng một phần năm tổng số nỗ lực được dành cho các yêu cầu và quy trình công việc phân tích (với nhau), một số khác-thứ năm vào công việc thiết kế, và khoảng ba phần năm với công việc thực hiện. Tổng kích thước tương đối của các khu vực bóng mờ trong hình 2.4 phản ánh những giá trị

Có lặp lại trong mỗi tăng của hình 2.4. Điều này được thể hiện trong hình 2.5, trong đó mô tả ba lần lặp lại trong Tăng B. (Hình 2.5 là một cái nhìn mở rộng của cột thứ hai của hình 2.4.) Như trong hình 2.5, mỗi lần lặp lại liên quan đến tất cả các fi đã làm việc lưu chuyển nhưng lại theo tỷ lệ khác nhau.

Một lần nữa, cần phải nhấn mạnh rằng hình 2.5 không có ý định để cho thấy rằng mỗi tăng liên quan đến chính xác ba lần lặp lại. Số lần lặp lại thay đổi từ tăng thặng dư. Mục đích của hình 2.5 là để cho thấy sự lặp lại trong mỗi tăng và lặp lại rằng tất cả năm công việc (yêu cầu, phân tích, thiết kế, thực hiện và kiểm tra, cùng với việc lập kế hoạch và tài liệu) được thực hiện trong hầu hết các lần lặp lại, mặc dù trong các tỷ lệ khác nhau mỗi thời gian.

Như đã giải thích, hình 2.4 refl các dự các incrementation nội tại cho sự phát triển của mỗi sản phẩm phần mềm. Hình 2.5 hiển thị một cách rõ ràng sự lặp đi lặp lại mà nền tảng incrementation. Cụ thể, Hình 2.5 mô tả ba bước lặp đi lặp lại liên tục, như trái ngược với một incrementation lớn. Chi tiết hơn, lặp lại B.1 bao gồm các yêu cầu, phân tích, thiết kế, thực hiện và quy trình công việc kiểm tra, đại diện bởi các hình chữ nhật nét đứt tận cùng bên trái với các góc được bo tròn. Các lặp tiếp tục cho đến khi các cổ vật của một trong năm công việc là đạt yêu cầu.

Tiếp theo, tất cả năm bộ đồ tạo tác này lặp đi lặp lại trong Iteration B.2. lặp thứ hai này cũng tương tự như trong tự nhiên để là người đầu tiên. Đó là, các yêu cầu hiện vật được cải thiện, do đó gây cải tiến cho các đồ tạo tác phân tích, và như vậy, như được phản ánh trong các lần lặp thứ hai của hình 2.5, và tương tự cho các lần lặp thứ ba. Quá trình lặp đi lặp lại và incrementation bắt đầu vào đầu của Tăng A và tiếp tục cho đến cuối Tăng D. Các sản phẩm phần mềm hoàn thành sau đó được cài đặt

trên máy tính của khách hàng.

**2.6 Winburg Mini Case Study Revisited**

Hình 2.6 cho thấy các mô hình phát triển cây của Winburg trường hợp nhỏ nghiên cứu (Hình 2.2) chồng trên mô hình lặp đi lặp lại-và-gia tăng (các công việc kiểm tra không được hiển thị vì các mô hình phát triển cây giả thử nghiệm liên tục, giải thích trong Phần 1.7). Hình ánh sáng 2,6 chuồng thêm về bản chất của incrementation:

* Tăng A tương ứng với tập 1, Tăng B tương ứng với tập 2, và như vậy.
* Từ quan điểm của mô hình lặp đi lặp lại-và-gia tăng, hai trong số những gia không bao gồm tất cả bốn quy trình công việc. Chi tiết hơn, Tăng B (Tập 2) bao gồm chỉ các công việc thực hiện, và Tăng C (tập 3) chỉ bao gồm các công việc thiết kế và các công việc thực hiện. Các mô hình lặp đi lặp lại và gia tăng không đòi hỏi rằng mọi công việc được thực hiện trong mỗi tăng.
* Hơn nữa, trong hình 2.4 hầu hết các công việc yêu cầu được thực hiện trong Tăng A và B Tăng, trong khi ở hình 2.6 nó được thực hiện trong Tăng A và D. Tăng Ngoài ra, trong hình 2.4 hầu hết các phân tích được thực hiện trong Tăng B, trong khi ở hình 2.6 các công việc phân tích được thực hiện trong Tăng A và Tăng D. Điều này chỉ ra rằng không phải hình 2.4 cũng không Hình 2.6 thể hiện cách mỗi sản phẩm phần mềm được tích hợp. Thay vào đó, mỗi con số cho thấy cách mà một sản phẩm phần mềm đặc biệt được xây dựng, làm nổi bật cơ bản lặp đi lặp lại và incrementation.
* Kích thước nhỏ và chấm dứt đột ngột của công việc thực hiện trong Tăng B (Tập 2) của Hình 2.6 cho thấy rằng thực hiện 2 không hoàn thành. Việc các dự mảnh refl xám phần của công việc này thực hiện mà không được thực hiện.
* Ba mũi tên đứt đoạn của sự phát triển cây mô hình cho thấy, mỗi tăng tạo thành duy trì tăng trước đó. Trong ví dụ này, increments thứ hai và thứ ba là các trường hợp bảo trì sửa chữa. Điều đó là, mỗi increment sửa lỗi trong tăng trước đó. Như trước đây giải thích, Tăng B (Tập 2) sửa chữa các công việc thực hiện bởi thay thế các biến đôi chính xác với các biến chính xác đơn thông thường. Tăng C (tập 3) sửa chữa các công việc thiết kế bằng cách sử dụng một hình ảnh nhanh hơn công nhận thuật toán, do đó cho phép các yêu cầu thời gian đáp ứng là đã gặp. thay đổi tương ứng sau đó phải được thực hiện để thực hiện công việc. Cuối cùng, trong Tăng D (tập 4) các yêu cầu được thay đổi để định cải thiện độ chính xác tổng thể, một thể hiện của bảo trì hoàn bị. tương ứng thay đổi sau đó được thực hiện cho các công việc phân tích, thiết kế quy trình làm việc,và công việc thực hiện

**2.7 Rủi ro và các khía cạnh khác của Iteration và incrementation**

Một cách khác để nhìn vào sự lặp lại và incrementation là toàn bộ dự án được chia thành tiểu dự án nhỏ hơn (hoặc gia tăng). Mỗi dự án nhỏ mở rộng các đồ tạo tác yêu cầu, phân tích, thiết kế, triển khai thực hiện và thử nghiệm. Cuối cùng, các bộ kết quả hiện vật cấu thành sản phẩm phần mềm hoàn chỉnh.

Trong thực tế, mỗi dự án nhỏ bao gồm hơn chỉ mở rộng các hiện vật. Nó là điều cần thiết để kiểm tra xem mỗi vật là đúng (các công việc thử nghiệm) và thực hiện bất kỳ thay đổi cần thiết để các hiện vật có liên quan. Quá trình kiểm tra và sửa đổi, họ kiểm tra lại và đang thay đổi, và như vậy, rõ ràng là lặp đi lặp lại trong tự nhiên. Nó tiếp tục cho đến khi các thành viên của nhóm phát triển là hài lòng với tất cả các hiện vật của các dự án nhỏ hiện tại (hoặc tăng). Khi điều đó xảy ra, họ tiến tới mục tiếp theo.

So sánh hình 2.3 (mô hình thác nước) với hình 2.5 (quan điểm của các lần lặp lại trong Tăng B) cho thấy, mỗi lần lặp có thể được xem như là một mô hình thác nhỏ nhưng đầy đủ. Đó là, trong mỗi lần lặp các thành viên của nhóm phát triển đi qua các yêu cầu cổ điển, phân tích, thiết kế, và giai đoạn thực hiện trên một phần cụ thể của sản phẩm phần mềm. Từ quan điểm này, các mô hình lặp đi lặp lại-và-gia tăng các hình 2.4 và 2.5 có thể được xem như là một loạt liên tiếp của mô hình thác nước.

Các mô hình lặp đi lặp lại-và-gia tăng có nhiều ưu điểm:

1. Nhiều cơ hội được cung cấp để kiểm tra rằng sản phẩm phần mềm là chính xác.Mỗi lần lặp kết hợp các công việc kiểm tra, như vậy mỗi lần lặp là một cơ hội khác để rà soát tất cả các hiện vật phát triển đến thời điểm này. Các lỗi sau đó được phát hiện và điều chỉnh, cao hơn là chi phí, như thể hiện trong hình 1.6. Không giống như các mô hình thác nước cổ điển, từng nhiều lần lặp lại của mô hình lặp đi lặp lại-và-gia tăng cung cấp một cơ hội tốt để fi lỗi thứ và sửa chữa chúng, do đó tiết kiệm tiền.
2. Sự vững mạnh của kiến ​​trúc cơ bản có thể được xác định tương đối sớm trong chu kỳ cuộc sống. **Kiến trúc (architecture)** của một sản phẩm phần mềm bao gồm các thành phần khác nhau hiện vật và làm thế nào họ phù hợp với nhau. Một tương tự là kiến ​​trúc của một nhà thờ, mà có thể được mô tả như Romanesque, Gothic, Baroque hay, trong số khác khả năng. Tương tự như vậy, kiến ​​trúc của một sản phẩm phần mềm có thể được mô tả như hướng đối tượng (Chương 7), ống dẫn và bộ lọc (UNIX hay Linux thành phần), hoặc client-server (với một máy chủ trung tâm cung cấp các tập tin lưu trữ cho một mạng lưới khách hàng máy tính). Kiến trúc của một sản phẩm phần mềm được phát triển bằng cách sử dụng lặp đi lặp lại và gia tăng mô hình phải có tài sản mà nó có thể được mở rộng liên tục (Và, nếu cần thiết, dễ dàng thay đổi) để kết hợp mục tiếp theo. Có khả năng để xử lý phần mở rộng và thay đổi như vậy mà không rơi xuống ngoài được gọi là mạnh mẽ. **Vững mạnh (Robustness)** là một chất quan trọng trong quá trình phát triển của một sản phẩm phần mềm; nó là quan trọng trong quá trình bảo trì sau giao hàng. Vì vậy, nếu một sản phẩm phần mềm là để kéo dài qua thông thường 12, 15, hoặc năm nữa bảo trì postdelivery, kiến ​​trúc cơ bản có phải là mạnh mẽ. Khi một mô hình lặp đi lặp lại-và-gia tăng được sử dụng, nó sớm trở nên rõ ràng hay không kiến ​​trúc là mạnh mẽ. Nếu trong quá trình kết hợp (nói) tăng thứ ba, nó là rõ ràng rằng các phần mềm được phát triển để ngày phải được tổ chức lại đáng kể và phần lớn thực hiện lại, sau đó nó là rõ ràng rằng kiến ​​trúc là không đủ mạnh. Các khách hàng phải quyết định xem có nên từ bỏ dự án hoặc bắt đầu lại từ đầu. Một khả năng khác là để thiết kế lại kiến trúc để được mạnh mẽ hơn, và sau đó tái sử dụng càng nhiều các hiện vật hiện tại càng tốt trước khi tiến tới mục tiếp theo. Một lý do tại sao một mạnh mẽ kiến trúc là rất quan trọng là vấn đề di chuyển, mục tiêu (mục 2.4). Đó là tất cả nhưng chắc chắn rằng các yêu cầu của khách hàng sẽ thay đổi, hoặc là vì sự tăng trưởng trong vòng tổ chức của khách hàng hoặc do khách hàng luôn thay đổi tâm trí của mình như với những gì các phần mềm mục tiêu đã làm. Mạnh mẽ hơn kiến ​​trúc, càng có nhiều đàn hồi để thay đổi phần mềm sẽ được. Nó không phải là có thể thiết kế một kiến ​​trúc có thể đối phó với quá nhiều thay đổi mạnh mẽ. Nhưng, nếu những thay đổi cần thiết một cách hợp lý trong phạm vi, một kiến ​​trúc mạnh mẽ nên có khả năng kết hợp những thay đổi mà không cần phải được cơ cấu lại đáng kể.
3. Các mô hình lặp đi lặp lại-và-gia tăng cho phép chúng tôi để giảm thiểu rủi ro sớm. Rủi ro (Risks ) là luôn luôn tham gia phát triển phần mềm và bảo trì. Trong trường hợp nghiên Winburg mini, ví dụ, các thuật toán nhận dạng ảnh ban đầu là không đủ nhanh; có một nguy cơ luôn hiện diện rằng một sản phẩm phần mềm hoàn thành sẽ không gặp khó khăn thời gian của mình. Phát triển các sản phẩm phần mềm cho phép chúng tôi từng bước để giảm thiểu những rủi ro này sớm trong vòng đời. Ví dụ, giả sử một mạng lưới khu vực địa phương mới (LAN) đang được phát triển và có mối quan tâm rằng phần cứng mạng hiện tại chưa đủ cho các sản phẩm phần mềm mới. Sau đó, các fi đầu tiên một hoặc hai lần lặp được hướng vào xây dựng những phần của phần mềm giao diện với các phần cứng mạng. Nếu nó quay ra rằng, trái với lo ngại của các nhà phát triển, mạng có những khả năng cần thiết, các nhà phát triển có thể tham gia thảo luận với các dự án, confi dent rằng nguy cơ này đã được giảm thiểu. Mặt khác, nếu các mạng thực sự không thể đối phó với lưu lượng bổ sung rằng các mạng LAN mới tạo ra, điều này được báo cáo cho khách hàng sớm trong chu kỳ cuộc sống, khi chỉ có một tỷ lệ nhỏ của ngân sách đã được chi tiêu. Các khách hàng bây giờ có thể quyết định hủy bỏ dự án, mở rộng khả năng của mạng lưới hiện có, mua một hệ thống mới và mạnh mẽ hơn, hoặc có hành động khác.
4. Chúng tôi luôn luôn có một phiên bản làm việc 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 vào cuối của dự án là có một phiên bản làm việc của các sản phẩm phần mềm. Ngược lại, khi các mô hình vòng đời lặp đi lặp lại-và-gia tăng được sử dụng, ở phần cuối của mỗi lần lặp, có một phiên bản làm việc của một phần của các sản phẩm phần mềm tiêu chung. Các khách hàng và người sử dụng dự định có thể thử nghiệm với phiên bản đó và xác định những gì cần thay đổi nhằm đảm bảo việc thực hiện hoàn thành trong tương lai đáp ứng nhu cầu của họ. Những thay đổi này có thể được thực hiện cho một số gia tiếp theo, và sau đó các khách hàng và người sử dụng có thể xác định những thay đổi khác là cần thiết. Một biến thể này là để cung cấp các phiên bản phần của các sản phẩm phần mềm, không chỉ để thử nghiệm nhưng cũng để mịn sự ra đời của các sản phẩm phần mềm mới trong tổ chức khách hàng. Thay đổi là hầu như luôn luôn được coi là một mối đe dọa. Tất cả các quá thường xuyên, người dùng lo sợ rằng sự ra đời của một mới sản phẩm phần mềm trong môi trường làm việc sẽ dẫn đến họ mất việc với máy tính. Tuy nhiên, giới thiệu một sản phẩm phần mềm có thể dần dần có hai lợi ích mà. Đầu tiên, nỗi sợ hãi hiểu bị thay thế bởi một máy tính được giảm bớt. Thứ hai, nó thường dễ dàng hơn để tìm hiểu những tính năng của một sản phẩm phần mềm phức tạp nếu chức năng được giới thiệu từng bước trong khoảng thời gian tháng, chứ không phải như một toàn thể.
5. Có bằng chứng thực nghiệm rằng chu kỳ cuộc sống lặp đi lặp lại-và-gia tăng hoạt động. Biểu đồ hình tròn trong hình 1.1 cho thấy các kết quả được báo cáo của Tập đoàn Standish vào các dự án hoàn thành năm 2006 [Rubenstein, 2007]. Trong thực tế, báo cáo này (còn gọi là CHAOS Report-thấy Chỉ trong trường hợp bạn muốn biết Box 2.2) được sản xuất mỗi 2 năm. Hình 2,7 cho thấy kết quả cho năm 1994 đến năm 2006. Tỷ lệ sản phẩm thành công tăng đều đặn từ 16 phần trăm năm 1994 xuống còn 34 phần trăm trong năm 2002, nhưng sau đó giảm xuống 29 phần trăm trong năm 2004. Trong cả năm 2002 [Softwaremag.com, 2004] và 2004 [Hayes, 2004] báo cáo, một trong những yếu tố liên quan đến dự án thành công là việc sử dụng một quá trình lặp đi lặp lại. (Các lý do đưa ra cho việc giảm tỷ lệ phần trăm của các dự án thành công trong năm 2004 bao gồm: các dự án lớn hơn vào năm 2002, sử dụng các mô hình thác nước, thiếu sự tham gia của người sử dụng, và thiếu sự hỗ trợ từ các giám đốc điều hành cao cấp [Hayes, 2004]). Sau đó , tỷ lệ phần trăm của các dự án thành công đã tăng trở lại trong nghiên cứu năm 2006 đến 35 phần trăm. Chủ tịch của Standish Group, Jim Johnson, do sự gia tăng này đến ba yếu tố: quản lý dự án tốt hơn, các cơ sở hạ tầng đang nổi lên Web, và (lại) phát triển lặp [Rubenstein, 2007].

Just in Case You Wanted to Know Box 2.2

Các CHAOS hạng là một từ viết tắt. Đối với một số lý do không rõ, Tập đoàn Standish giữ bí mật từ viết tắt đầu. Các nhà nước [Standish, 2003]:

Chỉ có một vài người ở Standish Group, và bất kỳ một trong 360 người nhận và lưu các T-shirt, chúng tôi đã đưa ra sau khi họ hoàn thành cuộc khảo sát đầu tiên fi năm 1994, biết những gì các chữ CHAOS đại diện.

**2.8 Managing Iteration and Incrementation**

Ngay từ cái nhìn đầu tiên, mô hình lặp đi lặp lại-và-gia tăng các hình 2.4 và 2.5 trông hoàn toàn hỗn loạn. Thay vì sự tiến triển có trật tự từ yêu cầu để thực hiện các mô hình thác nước (Hình 2.3), có vẻ như các nhà phát triển làm bất cứ điều gì họ thích, có lẽ một số mã hóa vào buổi sáng, một giờ hoặc hai trong thiết kế sau khi ăn trưa, và sau đó nửa giờ chỉ định trước khi về nhà. Đó không phải là trường hợp. Ngược lại, các mô hình lặp đi lặp lại-và-gia tăng là như regimented như mô hình thác nước, bởi vì như trước đây chỉ ra, việc phát triển một sản phẩm phần mềm bằng cách sử dụng mô hình lặp đi lặp lại-và-gia tăng là không có gì nhiều hơn hoặc ít hơn so với việc phát triển một hàng loạt các sản phẩm phần mềm nhỏ, tất cả đều sử dụng mô hình thác nước.

Cụ thể hơn, như thể hiện trong hình 2.3, phát triển một sản phẩm phần mềm sử dụng mô hình thác nước có nghĩa là liên tục thực hiện các yêu cầu, phân tích, thiết kế, và giai đoạn thực hiện (theo thứ tự) vào các sản phẩm phần mềm như một toàn thể. Nếu một vấn đề đang gặp phải, các vòng phản hồi của Hình 2.3 (mũi tên đứt đoạn) được theo sau; đó là, lặp lại (bảo trì) được thực hiện. Tuy nhiên, nếu các sản phẩm phần mềm tương tự được phát triển bằng cách sử dụng mô hình lặp đi lặp lại-và-gia tăng, các sản phẩm phần mềm được coi là một tập hợp các gia số. Đối với mỗi tăng lần lượt, yêu cầu, phân tích, thiết kế, và các giai đoạn thực hiện (theo thứ tự) được liên tục thực hiện trên thặng dư đó cho đến khi nó là rõ ràng rằng không có sự lặp lại thêm là cần thiết. Nói cách khác, toàn bộ dự án được chia thành một loạt các dự án thác nước mini. Trong mỗi dự án nhỏ, lặp được thực hiện khi cần thiết, như thể hiện trong hình 2.5. Vì vậy, lý do đoạn trước đã nói rằng mô hình lặp đi lặp lại-và-gia tăng là như regimented như mô hình thác nước là vì các mô hình lặp đi lặp lại-và-gia tăng là mô hình thác nước, áp dụng liên tục.

**2.9 Other Life-Cycle Models**

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

**2.9.1 Code-and-Fix Life-Cycle Model**

Đáng tiếc là rất nhiều sản phẩm được phát triển dựa trên những gì có thể được gọi là **mô hình vòng đời mã và sửa chữa (code-and-fix life-cycle model .).** Sản phẩm được thực hiện mà không yêu cầu hoặc thông số kỹ thuật, hoặc bất kỳ nỗ lực thiết kế. Thay vào đó, các nhà phát triển chỉ đơn giản là ném đang cùng nhau và làm lại nó nhiều lần nếu cần thiết để đáp ứng các khách hàng. Cách tiếp cận này được hiển thị trong hình 2.8, trong đó hiển thị rõ ràng sự vắng mặt của các yêu cầu, thông số kỹ thuật và thiết kế. Mặc dù phương pháp này có thể hoạt động tốt trên trình ngắn bài tập dài 100 hoặc 200 dòng, các mô hình mã và sửa chữa là hoàn toàn không đạt yêu cầu cho các sản phẩm của bất kỳ kích thước hợp lý. Hình 1.6 cho thấy rằng chi phí của việc thay đổi một sản phẩm phần mềm là tương đối nhỏ nếu thay đổi được thực hiện trong các yêu cầu, phân tích, hoặc giai đoạn thiết kế, nhưng phát triển không thể chấp nhận lớn nếu thay đổi được thực hiện sau khi các sản phẩm đã được mã hóa, hay tệ hơn, nếu nó có đã được giao hàng và cài đặt trên máy tính của khách hàng. Do đó, chi phí của các phương pháp mã và sửa chữa thực sự là lớn hơn nhiều so với chi phí của một ed đúng specifi và thiết kế tỉ mỉ sản phẩm. Ngoài ra, duy trì một sản phẩm có thể được sùng bái rất diffi không có đặc điểm kỹ thuật hoặc thiết kế tài liệu, và các cơ hội của một lỗi xảy ra hồi quy là lớn hơn đáng kể. Thay vì các phương pháp tiếp cận x mã và-fi, điều quan trọng là, trước khi phát triển một sản phẩm bắt đầu, một mô hình vòng đời thích hợp được lựa chọn.

Đáng tiếc, tất cả quá nhiều dự án sử dụng các mô hình mã và sửa chữa. Vấn đề này đặc biệt nghiêm trọng tại các tổ chức đo lường sự tiến bộ chỉ về dòng mã, vì vậy các thành viên của nhóm phát triển phần mềm được áp lực vào tung ra nhiều dòng mã càng tốt, bắt đầu từ ngày đầu của dự án. Mô hình mã và sửa chữa là cách dễ nhất để phát triển phần mềm và cách tốt nhất

Một phiên bản đơn giản của mô hình thác nước đã được trình bày tại mục 2.2. Bây giờ chúng ta xem xét rằng mô hình chi tiết hơn.

**2.9.2 Waterfall Life-Cycle Model**

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

Một điểm quan trọng liên quan đến mô hình thác nước là không có giai đoạn hoàn tất cho đến khi tài liệu cho giai đoạn đó đã được hoàn thành và sản phẩm của giai đoạn đó đã được chấp thuận bởi các nhóm đảm bảo chất lượng phần mềm (SQA). Điều này mang sang sửa đổi nếu các sản phẩm của một giai đoạn trước đó đã được thay đổi như một hệ quả của sau một vòng phản hồi, mà giai đoạn trước đó được coi là hoàn thành chỉ khi các tài liệu cho các giai đoạn đã được sửa đổi và sửa đổi đã được kiểm tra nhóm SQA. Vốn có trong mọi giai đoạn của mô hình thác nước là thử nghiệm. Thử nghiệm không phải là một giai đoạn riêng biệt để chỉ được thực hiện sau khi sản phẩm đã được xây dựng, cũng không phải là chỉ được thực hiện vào cuối mỗi giai đoạn. Thay vào đó, như đã nêu tại mục 1.7, các xét nghiệm cần tiến hành liên tục trong suốt quá trình phần mềm. Đặc biệt, khi bảo trì, nó là cần thiết để đảm bảo không chỉ là phiên bản sửa đổi của sản phẩm vẫn làm những gì các phiên bản trước đó đã làm và vẫn còn hiện nó một cách chính xác (kiểm tra hồi quy), nhưng nó cũng satisfi es bất kỳ yêu cầu mới đặt ra bởi khách hàng.

Mô hình thác nước có nhiều ưu điểm, trong đó có thi hành kỷ luật cách tiếp cận các quy định tài liệu được cung cấp tại mỗi giai đoạn và yêu cầu tất cả các sản phẩm của từng giai đoạn (bao gồm cả tài liệu) được kiểm tra tỉ mỉ bởi SQA. Tuy nhiên, thực tế là mô hình thác nước là tài liệu hướng cũng có thể là một điểm yếu. Để xem này, xem xét hai kịch bản có phần kỳ lạ sau

Đầu tiên, Joe và Jane Johnson quyết định xây dựng một ngôi nhà. Họ tham khảo ý kiến với một kiến trúc sư. Thay vì hiển thị chúng phác thảo, kế hoạch, và có lẽ một mô hình quy mô, kiến trúc sư cung cấp cho họ một 20 trang đơn cách nhau tài liệu đánh máy mô tả ngôi nhà trong điều kiện kỹ thuật cao. Mặc dù cả Joe và Jane không có kinh nghiệm kiến trúc trước và hầu như không hiểu các tài liệu, họ nhiệt tình ký tên và nói, "Đi ngay phía trước, xây dựng nhà!"

kịch bản khác là như sau: Đánh dấu Marberry mua bộ quần áo của mình bằng cách đặt hàng qua thư. Thay thế của mailing anh hình ảnh của bộ quần áo của họ và các mẫu vải có sẵn, công ty sẽ gửi Đánh dấu một mô tả bằng văn bản của việc cắt giảm và vải của sản phẩm của họ. Đánh dấu sau đó ra lệnh cho một bộ đồ chỉ trên cơ sở của một mô tả bằng văn bản.

Trước đó hai kịch bản là rất khó. Tuy nhiên, họ tiêu biểu cho chính xác các phần mềm cách thường được xây dựng bằng cách sử dụng mô hình thác nước. Quá trình bắt đầu với các thông số kỹ thuật. Nói chung, các tài liệu đặc tả là dài, chi, và khá thẳng thắn nhàm chán để đọc. Các khách hàng thường thiếu kinh nghiệm trong việc đọc thông số kỹ thuật phần mềm và khó khăn này là phức tạp bởi thực tế rằng các tài liệu kỹ thuật thường được viết bằng một phong cách mà các khách hàng là không quen thuộc. Sự khó khăn thậm chí còn tồi tệ hơn khi các thông số kỹ thuật được viết bằng một ngôn ngữ đặc tả hình thức như Z [Spivey, 1992] (mục 12.9). Tuy nhiên, khách hàng tiến hành ký tắt vào các tài liệu kỹ thuật, cho dù hiểu đúng hay không. Trong nhiều cách có rất ít sự khác biệt giữa Joe và Jane Johnson ký hợp đồng để có một ngôi nhà được xây dựng từ một mô tả bằng văn bản rằng họ chỉ có một phần hiểu và khách hàng phê duyệt một sản phẩm phần mềm được mô tả trong điều khoản của một tài liệu đặc tả rằng họ chỉ có một phần hiểu.

Mark Marbury và mail đặt hàng qua thư của ông có vẻ kỳ lạ trong cùng cực, nhưng đó là chính xác những gì sẽ xảy ra khi các mô hình thác nước được sử dụng trong phát triển phần mềm. Lần đầu tiên mà khách hàng nhìn thấy một sản phẩm làm việc là chỉ sau khi toàn bộ sản phẩm đã được mã hóa. Một ngạc nhiên nhỏ các nhà phát triển phần mềm sống trong sợ hãi của câu, "Tôi biết điều này là những gì tôi yêu cầu, nhưng nó không phải là thực sự những gì tôi muốn."

Điều gì đã đi sai? Có một sự khác biệt đáng kể giữa cách một khách hàng hiểu một sản phẩm như mô tả của các tài liệu kỹ thuật và các sản phẩm thực tế. Các thông số kỹ thuật chỉ tồn tại trên giấy; khách hàng do đó có thể không thực sự hiểu những gì bản thân sản phẩm sẽ như thế nào. Các mô hình thác nước, phụ thuộc như nó làm như vậy chủ yếu vào thông số kỹ thuật bằng văn bản, có thể dẫn đến việc xây dựng các sản phẩm mà chỉ đơn giản là không đáp ứng được nhu cầu thực sự của khách hàng.

Công bằng mà nói nó sẽ được chỉ ra rằng, chỉ như một kiến trúc sư có thể giúp khách hàng hiểu rõ những gì sẽ được xây dựng bằng cách cung cấp mô hình quy mô, bản phác thảo, và các kế hoạch, vì vậy các kỹ sư phần mềm có thể sử dụng kỹ thuật đồ họa, như sơ đồ luồng dữ liệu (Mục 12.3 ) hoặc sơ đồ UML (Chương 17) để giao tiếp với khách hàng. Vấn đề là những trợ đồ họa không mô tả cách thức sản phẩm hoàn thành sẽ làm việc. Ví dụ, có một sự khác biệt đáng kể giữa một sơ đồ (mô tả sơ đồ của một sản phẩm) và các sản phẩm làm việc riêng của mình. Trong cuốn sách này, hai giải pháp được đưa ra để giải quyết những vấn đề mà các tài liệu đặc tả thường không mô tả một sản phẩm trong một cách mà cho phép khách hàng để xác định xem các sản phẩm đề xuất đáp ứng nhu cầu của mình. Các giải pháp hướng đối tượng được mô tả trong Chương 11 và 13. Những giải pháp cổ điển là mô hình nhanh chóng-mẫu, mô tả trong Phần 2.9.3.

**2.9.3 Rapid-Prototyping Life-Cycle Model**

Một **nguyên mẫu nhanh chóng (Rapid-Prototyping)** là một mô hình làm việc mà có chức năng tương đương với một tập hợp con của sản phẩm. Ví dụ, nếu các sản phẩm mục tiêu là để xử lý các khoản nợ phải trả, các khoản phải thu, và kho bãi, sau đó các mẫu thử nghiệm nhanh chóng có thể bao gồm một sản phẩm mà thực hiện việc xử lý để thu thập dữ liệu màn hình và in các báo cáo nhưng không có fi le cập nhật hoặc lỗi xử lý . Một nguyên mẫu nhanh chóng cho một sản phẩm mục tiêu đó là để xác định nồng độ của một enzym trong một giải pháp có thể thực hiện các tính toán và hiển thị các câu trả lời, nhưng mà không làm bất kỳ xác nhận và hợp lý kiểm tra các dữ liệu đầu vào.

Bước đầu tiên trong **mô hình vòng đời nhanh tạo mẫu (Rapid-Prototyping Life-Cycle Model)** mô tả trong hình 2.10 là để xây dựng một nguyên mẫu nhanh chóng và để cho các khách hàng và người sử dụng trong tương lai tương tác và trải nghiệm với các mẫu thử nghiệm nhanh chóng. Một khi khách hàng hài lòng rằng các mẫu thử nghiệm nhanh thực hiện hầu hết những gì là cần thiết, các nhà phát triển có thể xây dựng các tài liệu đặc tả với một số bảo đảm rằng sản phẩm đáp ứng nhu cầu thực tế của khách hàng

Sau khi sản xuất nguyên mẫu nhanh chóng, quá trình phần mềm tiếp tục như trong Hình 2.10. Một thế mạnh lớn của mô hình nhanh tạo mẫu là sự phát triển các sản phẩm chủ yếu là tuyến tính, tiến hành trên nguyên mẫu nhanh chóng để sản phẩm được chuyển đến; các vòng phản hồi của các mô hình thác nước (Hình 2.9) ít có khả năng được cần thiết trong mô hình nhanh chóng-prototyping. Có một số lý do cho việc này. Đầu tiên, các thành viên của nhóm phát triển sử dụng nguyên mẫu nhanh chóng để xây dựng các tài liệu đặc tả. Bởi vì các nguyên mẫu nhanh chóng làm việc đã được xác nhận qua sự tương tác với khách hàng, đó là lý do để cho rằng các kết quả tài liệu đặc tả sẽ được chính xác. Thứ hai, xem xét việc thiết kế. Mặc dù nguyên mẫu nhanh chóng đã (khá đúng) được lắp ráp vội vã, đội ngũ thiết kế có thể đạt được cái nhìn sâu sắc từ nó, lúc tồi tệ nhất nó sẽ là "làm thế nào để không làm điều đó" nhiều. Một lần nữa, các vòng phản hồi của các mô hình thác nước ít có khả năng được cần thiết ở đây.

Thực hiện ra tiếp theo. Trong mô hình thác nước, thực hiện việc thiết kế đôi khi dẫn đến thiết kế lỗi đưa ra ánh sáng. Trong mô hình nhanh chóng-mẫu, thực tế là một phiên bản làm việc sơ bộ của các sản phẩm phần mềm đã được xây dựng có xu hướng giảm bớt sự cần thiết phải sửa chữa các thiết kế trong hoặc sau khi thực hiện. Các mẫu đã đưa ra một số hiểu biết để đội ngũ thiết kế, mặc dù nó có thể chỉ phản ánh một phần chức năng của sản phẩm mục tiêu hoàn tất.

Một khi sản phẩm đã được chấp nhận bởi khách hàng và cài đặt, bảo trì sau giao hàng bắt đầu. Tùy thuộc vào công việc bảo trì cụ thể mà đã được thực hiện, chu kỳ được trở lại với một trong hai tại các yêu cầu phân tích, thiết kế, hoặc giai đoạn thực hiện.

Một khía cạnh quan trọng của một nguyên mẫu nhanh chóng được thể hiện trong lời nhanh chóng. Các nhà phát triển nên nỗ lực để xây dựng nguyên mẫu nhanh càng nhanh càng tốt để đẩy nhanh quá trình phát triển phần mềm. Sau khi tất cả, việc sử dụng duy nhất của nguyên mẫu nhanh chóng là để xác định những nhu cầu thực sự của khách hàng được; một khi điều này đã được xác định, việc thực hiện nguyên mẫu nhanh chóng bị loại bỏ, nhưng các bài học kinh nghiệm được giữ lại và được sử dụng trong giai đoạn phát triển tiếp theo. Vì lý do này, các cấu trúc bên trong của nguyên mẫu nhanh là không có liên quan. Điều quan trọng là các nguyên mẫu được xây dựng nhanh chóng và biến 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 Open-Source Life-Cycle Model**

Hầu như tất cả các dự án **phần mềm mã nguồn mở (open-source software)** thành công đi qua hai giai đoạn chính thức.Đầu tiên, một cá nhân có ý tưởng cho một chương trình, chẳng hạn như là một hệ điều hành (Linux), một trình duyệt Net (Firefox), hoặc một máy chủ Web (Apache). Anh ta hoặc cô xây dựng một phiên bản ban đầu, sau đó được làm sẵn có cho phân phối miễn phí cho bất cứ ai muốn có một bản ngày nay, điều này được thực hiện thông qua mạng Internet, ở các trang web như SourceForge.net và FreshMeat.net. Nếu ai đó tải một bản sao của phiên bản ban đầu và nghĩ rằng chương trình fulfi LLS một nhu cầu, họ sẽ bắt đầu sử dụng chương trình đó.

Nếu có đủ quan tâm đến các chương trình, dự án di chuyển dần vào thức giai đoạn hai. Người dùng trở thành người đồng phát triển, trong đó một số người dùng báo cáo lỗi và những người khác đề nghị cách fi xing những người khuyết tật. Một số người dùng đã đưa ra ý tưởng cho việc mở rộng chương trình, và những người khác thực hiện những ý tưởng. Khi chương trình mở rộng về tính năng, nhưng cổng người dùng khác các chương trình để nó có thể chạy trên các hệ điều hành kết hợp thêm / phần cứng. Một khía cạnh quan trọng 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.

Bây giờ nhìn kỹ hơn vào ba hoạt động của giai đoạn chính thức thứ hai:

1. Báo cáo và hiệu chỉnh các lỗi là bảo trì sửa chữa.
2. Thêm chức năng bổ sung là bảo trì hoàn bị.
3. Porting các chương trình vào một môi trường mới là bảo trì thích ứng.

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

Có một số khác biệt quan trọng giữa nguồn đóng và các mô hình vòng đời phần mềm mã nguồn mở:

* phần mềm nguồn đóng được duy trì và thử nghiệm bởi các nhóm nhân viên của tổ chức sở hữu phần mềm. Người dùng đôi khi gửi báo cáo lỗi. Tuy nhiên, đây là những hạn chế để **báo cáo thất bại (failure reports)** (báo cáo về hành vi quan sát không chính xác); người dùng không có quyền truy cập vào mã nguồn, vì vậy họ không thể nộp báo cáo lỗi (fault reports) (báo cáo miêu tả nơi mã nguồn là không chính xác và làm thế nào để sửa nó).
  + Ngược lại, phần mềm mã nguồn mở thường được duy trì bởi các tình nguyện viên không lương. Người dùng được khuyến khích nộp báo cáo lỗi. Mặc dù tất cả người dùng có quyền truy cập vào mã nguồn, chỉ là thiểu số có độ nghiêng và thời gian, cũng như các kỹ năng cần thiết, kiểm tra nội dung mã nguồn và gửi báo cáo lỗi ( "sửa chữa"); do đó hầu hết các báo cáo lỗi đều báo cáo thất bại. thường có một **nhóm cốt lõi (core group)** của bảo trì chuyên dụng, người chịu trách nhiệm cho việc quản lý các dự án mã nguồn mở. Một số thành viên của **nhóm thiết bị ngoại vi (peripheral group)**, đó là, người sử dụng không phải là thành viên của nhóm nòng cốt, chọn nộp báo cáo lỗi theo thời gian. Các thành viên của nhóm nòng cốt là trách nhiệm đảm bảo rằng các khiếm khuyết này được sửa chữa. Cụ thể hơn, khi một báo cáo lỗi được gửi, một thành viên nhóm nòng cốt kiểm tra rằng việc sửa chữa thực sự giải quyết vấn đề và sửa đổi mã nguồn một cách thích hợp. Khi báo cáo thất bại được gửi, một thành viên của nhóm nòng cốt sẽ hoặc là cá nhân xác định fi x hoặc giao nhiệm vụ đó cho tình nguyện viên khác, thường là một thành viên của nhóm ngoại vi đang háo hức tham gia nhiều hơn vào các dự án mã nguồn mở. Một lần nữa, sức mạnh để cài đặt các fi x trong phần mềm được giới hạn cho các thành viên của nhóm nòng cốt.
* Phiên bản mới của phần mềm nguồn đóng thường được phát hành khoảng một năm một lần. Mỗi phiên bản mới được kiểm tra cẩn thận bởi các nhóm đảm bảo chất lượng phần mềm trước khi phát hành; một loạt các trường hợp thử nghiệm đang chạy.

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

So sánh con số 2.8, 2.10, và 2.11, chúng ta thấy rằng các mô hình vòng đời nguồn mở có đặc điểm chung với cả x mã mô hình-và-fi và các mô hình nhanh chóng-prototyping. Trong cả ba mô hình vòng đời, một phiên bản làm việc ban đầu được sản xuất. Trong trường hợp của các mô hình nhanh chóng-mẫu, phiên bản đầu tiên này được loại bỏ, và sau đó các sản phẩm mục tiêu được chỉ định và thiết kế trước khi được mã hóa. Trong cả hai mã-và-fi x và các mô hình vòng đời 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, có nói chung là không có cation specifi hoặc thiết kế.

Mang trong tâm trí tầm quan trọng lớn của việc có thông số kỹ thuật và thiết kế, làm thế nào có một số dự án mã nguồn mở thành công như vậy? Trong thế giới mã nguồn đóng, một số chuyên gia phần mềm có kỹ năng hơn và một số ít có tay nghề cao (xem phần 9.2). Thách thức của việc sản xuất phần mềm mã nguồn mở đã thu hút được một số các chuyên gia phần mềm fi tổ. Nói cách khác, một dự án mã nguồn mở có thể thành công, mặc dù thiếu thông số kỹ thuật hoặc thiết kế, nếu các kỹ năng của các cá nhân làm việc trên dự án mà rất tuyệt vời mà họ có thể hoạt động hiệu quả mà không cation specifi hoặc thiết kế.

Các mô hình vòng đời nguồn mở bị hạn chế trong ứng dụng của nó. Một mặt,các mô hình nguồn mở đã được sử dụng cực kỳ thành công cho các dự án phần mềm cơ sở hạ tầng nhất định, chẳng hạn như hệ điều hành (Linux, OpenBSD, Mach, Darwin), các 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, rất khó để thụ thai phát triển mã nguồn mở của một sản phẩm phần mềm được sử dụng chỉ trong một tổ chức thương mại. Một chìa khóa để mở nguồn phát triển phần mềm là các thành viên của cả hai nhóm nòng cốt và ngoại vi là những người sử dụng các phần mềm được phát triển. Do đó, các mô hình vòng đời nguồn mở được áp dụng trừ khi sản phẩm mục tiêu được xem bởi một loạt các người dùng là hữu ích cho họ.

Tại thời điểm viết bài, có khoảng 350.000 dự án mã nguồn mở tại SourceForge.net và FreshMeat.net. Khoảng một nửa số họ đã không bao giờ thậm chí còn thu hút được một đội ngũ làm việc trên dự án. Trong số những người mà công việc đã bắt đầu, ưu thế áp đảo đã không bao giờ được hoàn thành và không có khả năng bao giờ tiến xa hơn nữa. Nhưng khi các mô hình nguồn mở đã làm việc, nó đôi khi là vô cùng thành công. Các sản phẩm mã nguồn mở được liệt kê trong ngoặc đơn trong đoạn trước đó được sử dụng rộng rãi; hầu hết trong số họ được sử dụng một cách thường xuyên bởi hàng triệu người sử dụng.

Giải thích cho sự thành công của mô hình vòng đời nguồn mở được thể hiện trong Chương 4 trong bối cảnh đội khía cạnh tổ chức của phần mềm mã nguồn mở dự án.

**2.9.5 Agile Processes**

**lập trình cực (Extreme programming)** [Beck, 2000] là một cách tiếp cận mới gây tranh cãi để phát triển phần mềm dựa trên các mô hình lặp đi lặp lại-và-gia tăng. Bước gốc đầu tiên mà nhóm phát triển phần mềm xác định các tính năng khác nhau (tầng) khách hàng muốn sản phẩm để hỗ trợ. Đối với mỗi tính năng như vậy, nhóm nghiên cứu thông báo cho khách hàng bao lâu để thực hiện rằng tính năng và bao nhiêu nó sẽ chi phí. bước đầu tiên kinh này tương ứng với các yêu cầu và quy trình công việc phân tích các mô hình lặp đi lặp lại-và-gia tăng (Hình 2.4).

Các khách hàng lựa chọn các tính năng được bao gồm trong mỗi build tiếp sử dụng về mặt chi phí phân tích lợi ích (mục 5.2), có nghĩa là, trên cơ sở thời gian và dự toán chi phí được cung cấp bởi đội ngũ phát triển cũng như những lợi ích tiềm năng của các tính năng kinh doanh của mình. Các đề xuất xây dựng được chia thành các phần nhỏ hơn gọi là **nhiệm vụ(tasks)**. Một lập trình viên đầu tiên đưa ra những trường hợp thử nghiệm cho một nhiệm vụ; này được gọi là **phát triển thử nghiệm điều khiển (test-driven development) (TDD)**. Hai người lập trình làm việc cùng nhau trên một máy tính **( pair programming ) (cặp lập trình)** [Williams, Kessler, Cunningham, và Jeffries, 2000], thực hiện nhiệm vụ và đảm bảo rằng tất cả các trường hợp kiểm tra hoạt động chính xác. Hai người lập trình luân phiên đánh máy mỗi 15 hoặc 20 phút; người lập trình không phải là đánh máy cẩn thận kiểm tra các mã trong khi nó đang được nhập vào bởi đối tác của mình. Nhiệm vụ sau đó được tích hợp vào phiên bản hiện tại của sản phẩm. Lý tưởng nhất, triển khai và tích hợp một công việc nên không có nhiều hơn một vài giờ. Nói chung, một số cặp sẽ thi hành nhiệm vụ song song, do đó hội nhập là cơ bản liên tục. thành viên trong nhóm thay đổi mã hóa các đối tác hàng ngày, nếu có thể; học hỏi từ các thành viên khác trong nhóm làm tăng mức độ kỹ năng của tất cả mọi người. Các trường hợp kiểm tra TDD sử dụng cho các nhiệm vụ được giữ lại và sử dụng trong tất cả các thử nghiệm hội nhập sâu hơn.

Một số nhược điểm để ghép các chương trình đã được quan sát trong thực tế [Drobka, Noftz, và Raghu, 2004]. Ví dụ, cặp lập trình đòi hỏi một khối lượng lớn thời gian bị gián đoạn, và các chuyên gia phần mềm có thể có gặp khó diffi trong fi nding 3 đến khối 4 giờ thời gian. Ngoài ra, cặp lập trình không phải lúc nào cũng làm việc với các cá nhân hoặc nhút nhát hách, hoặc với hai lập trình viên thiếu kinh nghiệm.

Một số tính năng của chương trình cực đoan (XP) là hơi khác nhau từ cách thức mà phần mềm này thường phát triển:

* Các máy tính của đội XP được thiết lập ở trung tâm của một căn phòng lớn lót bằng buồng vệ nhỏ
* Một đại diện khách hàng làm việc với các đội XP ở tất cả các lần
* Không có cá nhân có thể làm việc thêm giờ trong hai tuần kế tiếp.
* Không có chuyên môn. Thay vào đó, tất cả các thành viên của nhóm làm việc XP vào yêu cầu, phân tích, thiết kế, mã, và thử nghiệm.
* Không có bước thiết kế tổng thể trước khi xây dựng khác nhau được xây dựng. Thay vào đó, thiết kế được thay đổi trong khi các sản phẩm đang được xây dựng. Thủ tục này được gọi là **tái cấu trúc(refactoring)**. Bất cứ khi nào một trường hợp thử nghiệm sẽ không chạy, các mã được tổ chức lại cho đến khi nhóm nghiên cứu là hài lòng rằng các thiết kế đơn giản, dễ hiểu, và chạy tất cả các trường hợp thử nghiệm một cách thỏa đáng.

Hai từ viết tắt nay gắn với lập trình cực đoan là YAGNI (bạn không gonna cần nó) và DTSTTCPW (làm điều đơn giản mà có thể làm việc). Nói cách khác, một nguyên tắc lập trình cực là để giảm thiểu số lượng các tính năng; không có nhu cầu xây dựng một sản phẩm mà không có nhiều hơn những gì khách hàng thực sự cần.

lập trình Extreme (cực)là một trong một số mô hình mới được gọi chung là **quy trình nhanh nhẹn.(agile processes .)** các nhà phát triển phần mềm Seventeen (sau này được đặt tên là Liên minh Agile) gặp nhau tại một khu nghỉ mát trượt tuyết ở Utah cho hai ngày vào tháng Hai năm 2001 và sản xuất các Tuyên ngôn Agile Phát triển phần mềm [Beck et al., 2001]. Nhiều người trong số những người tham gia trước đó đã là tác giả của phương pháp phát triển phần mềm riêng của họ, bao gồm Extreme Programming [Beck, 2000], Crystal [Cockburn, 2001], và Scrum [Schwaber, 2001]. Do đó, Liên minh Agile đã không quy định một mô hình vòng đời cụ thể, mà là đặt ra một nhóm các nguyên tắc cơ bản đó đã được phổ biến đến các cách tiếp cận cá nhân của họ để phát triển phần mềm.

quy trình Agile được đặc trưng bởi ít hơn đáng kể sự nhấn mạnh về phân tích và thiết kế hơn trong gần như tất cả các mô hình vòng đời hiện đại khác. Thực hiện bắt đầu sớm hơn nhiều trong chu kỳ cuộc sống bởi vì phần mềm làm việc được coi là quan trọng hơn so với tài liệu hướng dẫn chi tiết. Đáp ứng với những thay đổi trong yêu cầu là một mục tiêu chính của quy trình nhanh nhẹn, và do đó là tầm quan trọng của hợp tác với khách hàng

Một trong những nguyên tắc trong Tuyên Ngôn là để cung cấp phần mềm làm việc thường xuyên, tốt nhất là mỗi 2 hoặc 3 tuần. Một cách để đạt được điều này là sử dụng **timeboxing** [Jalote, Palit, Kurien, và Peethamber, 2004], đã được sử dụng trong nhiều năm như là một kỹ thuật quản lý thời gian. Một số lượng thời gian cụ thể được trích lập cho một nhiệm vụ, và các thành viên trong nhóm sau đó làm công việc tốt nhất mà họ có thể trong thời gian đó. Trong bối cảnh của quá trình nhanh nhẹn, thông thường là 3 tuần được dành cho mỗi lần lặp. Một mặt, nó mang lại cho các khách hàng dence confi để biết rằng một phiên bản mới với chức năng bổ sung sẽ đến mỗi 3 tuần. Mặt khác, các nhà phát triển biết rằng họ sẽ có 3 tuần (nhưng không nhiều) để cung cấp một sự lặp lại mới mà không cần sự can thiệp của khách hàng các loại; một khi khách hàng đã chọn công việc cho một sự lặp lại, nó không thể thay đổi hoặc tăng lên. Tuy nhiên, nếu không thể hoàn thành nhiệm vụ toàn bộ trong timebox, tác phẩm có thể được giảm ( "khinh"). Nói cách khác, các quy trình nhanh nhu cầu cố định thời gian, tính năng này không cố định

Một đặc điểm chung của các quy trình nhanh nhẹn là phải có một cuộc họp ngắn tại một thời gian đều đặn mỗi ngày. Tất cả các thành viên trong nhóm phải tham dự cuộc họp. Làm tất cả những người tham gia đứng trong một vòng tròn, thay vì ngồi quanh một chiếc bàn, giúp đảm bảo rằng cuộc họp kéo dài không quá các quy định 15 phút. Mỗi thành viên trong nhóm lần lượt trả lời 5 câu hỏi:

* Tôi đã làm gì kể từ cuộc họp ngày hôm qua?
* Tôi đang làm việc gì ngày hôm nay?
* Vấn đề gì đang ngăn cản tôi đạt được điều này?
* Chúng ta quên mất những gì?
* Tôi đã học được gì mà tôi muốn chia sẻ với các đội ?

Mục đích của **cuộc họp đứng lên (stand-up meeting)** là nâng cao vấn đề, không giải quyết cho họ; các giải pháp được tìm thấy tại các cuộc họp tiếp theo, tốt nhất được tổ chức ngay sau cuộc họp đứng lên. Giống như timeboxing, các cuộc họp đứng lên là một kỹ thuật quản lý thành công bây giờ sử dụng trong bối cảnh của quá trình nhanh nhẹn. Cả hai lần lặp timeboxed và các cuộc họp đứng lên là trường hợp của hai nguyên tắc cơ bản làm nền tảng cho tất cả các phương pháp nhanh nhẹn: giao tiếp và đáp ứng nhu cầu của khách hàng càng nhanh càng tốt.

**2.9.6 Synchronize-and-Stabilize Life-Cycle Model**

Microsoft, Inc, là nhà sản xuất lớn nhất thế giới của phần mềm COTS. Đa số các gói của nó được xây dựng sử dụng một phiên bản của mô hình lặp đi lặp lại-và-gia tăng đã được gọi là **ổn định và đồng bộ mô hình vòng đời (synchronize-and-stabilize life-cycle model)** [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 giải nén một danh sách các tính năng ưu tiên cao nhất cho khách hàng. Một tài liệu đặc tả hiện đang được soạn thảo. Tiếp theo, công việc được chia thành ba hay bốn xây dựng. Việc xây dựng đầu tiên bao gồm các tính năng quan trọng nhất, việc xây dựng thứ hai bao gồm các tính năng quan trọng nhất tiếp theo, và như vậy. Từng xây dựng được thực hiện bởi một số nhóm nhỏ làm việc song song. Vào cuối mỗi ngày, tất cả các đội **đồng bộ (synchronize)** ; đó là, họ đặt các thành phần hoàn thành phần với nhau và kiểm tra và gỡ lỗi các sản phẩm tạo ra. **Ổn định (Stabilization)** được thực hiện vào cuối mỗi xây dựng. Bất kỳ lỗi còn lại đã được phát hiện cho đến nay là cố định, và bây giờ họ **đóng băng (freeze)** xây dựng; đó là, không có những thay đổi khác sẽ được thực hiện cho các thông số kỹ thuật

Các bước đồng bộ hóa lặp đi lặp lại đảm bảo rằng các thành phần khác nhau luôn luôn làm việc với nhau. Một ưu điểm khác của thực thường xuyên này của sản phẩm xây dựng một phần là các nhà phát triển có được cái nhìn sâu sắc vào đầu vào hoạt động của các sản phẩm và có thể thay đổi yêu cầu nếu cần thiết trong quá trình xây dựng. Các mô hình vòng đời có thể được sử dụng ngay cả khi đặc tả ban đầu là không đầy đủ. Việc đồng bộ hóa-và-ổn định mô hình được xem xét tiếp tại mục 4.5, nơi mà đội bóng chi tiết tổ chức được thảo luận.

Mô hình xoắn ốc đã được để lại để kéo dài bởi vì nó kết hợp các khía cạnh của tất cả các khác mô hình được mô tả trong mục 2.9.

**2.9.7 Spiral Life-Cycle Model**

Như đã nêu tại mục 2.5, một yếu tố rủi ro luôn luôn là tham gia vào sự phát triển của phần mềm. Ví dụ, nhân sự chủ chốt có thể từ chức trước khi sản phẩm đã được ghi nhận đầy đủ. Các nhà sản xuất phần cứng trên sản phẩm đó được giới phê bình phụ thuộc có thể bị phá sản. Quá nhiều hoặc quá ít, có thể được đầu tư vào thử nghiệm và đảm bảo chất lượng. Sau khi chi tiêu hàng trăm ngàn đô la vào việc phát triển một sản phẩm phần mềm lớn, đột phá công nghệ có thể làm cho toàn bộ sản phẩm 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 bán trên thị trường, một giá thấp hơn, gói chức năng tương đương được công bố bởi một đối thủ cạnh tranh. Các thành phần của một sản phẩm có thể không fi t nhau khi hội nhập được thực hiện. Vì những lý do hiển nhiên, các nhà phát triển phần mềm cố gắng giảm thiểu những rủi ro như bất cứ nơi nào có thể.

Một cách để giảm thiểu các loại rủi ro nhất định là xây dựng một nguyên mẫu. Như mô tả trong Phần 2.9.3, một cách tiếp cận để giảm nguy cơ rằng các sản phẩm chuyển giao sẽ không đáp ứng các khách hàng thực sự cần là xây dựng một nguyên mẫu nhanh chóng trong giai đoạn yêu cầu. Trong giai đoạn tiếp theo, các loại khác của nguyên mẫu có thể thích hợp. Ví dụ, một công ty điện thoại có thể đưa ra một thuật toán dường như có hiệu quả cao mới cho việc định tuyến các cuộc gọi thông qua mạng lưới đường dài. Nếu sản phẩm được thực hiện nhưng không làm việc như mong đợi, các công ty điện thoại sẽ phải lãng phí chi phí phát triển sản phẩm. Ngoài ra, khách hàng tức giận hoặc bất tiện có thể đưa doanh nghiệp của họ ở nơi khác. Kết quả này có thể tránh được bằng cách xây dựng một bằng chứng của **khái niệm nguyên mẫu (proof-of-concept prototype)** để xử lý chỉ định tuyến các cuộc gọi và thử nghiệm nó trên một mô phỏng. Bằng cách này, các hệ thống thực tế không bị xáo trộn; và cho các chi phí thực hiện chỉ là thuật toán định tuyến, các công ty điện thoại có thể xác định xem nó là đáng giá để phát triển toàn bộ một bộ điều khiển mạng lưới kết hợp với các thuật toán mới.

**Một nguyên mẫu proof-of-concept** không phải là một nguyên mẫu nhanh chóng xây dựng để chắc chắn rằng các yêu cầu đã được xác định một cách chính xác, như mô tả trong mục 2.9.3. Thay vào đó, nó giống như một mẫu thử nghiệm kỹ thuật, đó là một mô hình quy mô xây dựng để kiểm tra tính khả thi của công trình. Nếu nhóm phát triển là quan tâm đến việc một phần riêng biệt của các sản phẩm phần mềm được đề nghị có thể được xây dựng, một bằng chứng của khái niệm nguyên mẫu được xây dựng. Ví dụ, các nhà phát triển có thể quan tâm đến việc một tính toán cụ thể có thể được thực hiện một cách nhanh chóng đủ. Trong trường hợp đó, họ xây dựng một nguyên mẫu để kiểm tra thời gian chỉ tính toán đó. Hoặc họ có thể lo lắng rằng các font họ có ý định sử dụng cho tất cả các màn hình sẽ là quá nhỏ cho người dùng trung bình để

đọc mà không mỏi mắt. Trong trường hợp này, họ xây dựng một nguyên mẫu để hiển thị một màu nâu đen của màn hình khác nhau và xác định bằng thực nghiệm cho dù người sử dụng tìmcác phông chữ khó chịu nhỏ.

Ý tưởng về giảm thiểu rủi ro thông qua việc sử dụng các nguyên mẫu và các phương tiện khác là ý tưởng cơ bản của cuộc sống-chu kỳ **mô hình xoắn ốc (spiral life-cycle model)** [Boehm, 1988]. Một cách đơn giản nhìn vào mô hình vòng đời này là như là một mô hình thác nước với từng giai đoạn trước bởi phân tích rủi ro, như thể hiện trong hình 2.12. Trước khi bắt đầu mỗi giai đoạn, một nỗ lực được thực hiện để **giảm thiểu (mitigate)** (điều khiển) các rủi ro. Nếu nó là không thể để giảm thiểu tất cả những rủi ro đáng kể ở giai đoạn đó, sau đó dự án ngay lập tức chấm dứt.

Nguyên mẫu có thể được sử dụng một cách hiệu quả để cung cấp thông tin về các lớp rủi ro nhất định. Ví dụ, hạn chế thời gian nói chung có thể được kiểm tra bằng cách xây dựng một nguyên mẫu và đo lường xem các mẫu thử nghiệm có thể đạt được hiệu quả cần thiết. Nếu mẫu thử nghiệm là một đại diện chức năng chính xác của các tính năng phù hợp của sản phẩm, sau đó đo được thực hiện trên nguyên mẫu nên cung cấp cho các nhà phát triển một ý tưởng tốt về việc liệu hạn chế thời gian có thể đạt được.