**Kiến trúc sư và kiến trúc ngày nay**

Vào thời điểm bắt đầu của thời đại điện toán, chi phí phần cứng phần lớn chiếm ưu thế hơn chi phí phần mềm. Vài thập kỷ sau, chúng tôi nhìn quanh và thấy tình huống này hoàn toàn khác biệt. Ngành công nghiệp đã có tiến bộ đáng kinh ngạc, và chi phí phần cứng đã giảm đáng kể. Chi phí phát triển phần mềm, mặt khác, đã tăng lên đáng kể, chủ yếu là do sự phức tạp ngày càng tăng của phát triển phần mềm doanh nghiệp tùy chỉnh.

Tình huống này đã tạo ra sự cần thiết cho một tập các giới luật để hướng dẫn các kỹ sư trong việc thiết kế các hệ thống như vậy. Chiếm từ ngành xây dựng, thuật ngữ kiến trúc đã trở thành cách phổ biến để mô tả nghệ thuật lập kế hoạch, thiết kế và triển khai các hệ thống phần mềm nhiều. Khi hai chúng tôi là thanh thiếu niên, bộ truyện tranh "Love Is" (http://www.loveiscartoon.com) đang ở đỉnh điểm phổ biến. Mỗi phần có một phần trí tuệ thiếu niên về tình yêu. Một bộ phim hoạt hình nói rằng: "Tình yêu là điều cần thiết chứ không phải là một thứ xa xỉ." Đó chính là vấn đề với kiến trúc phần mềm.

Trong chương đầu tiên này, chúng tôi cố gắng chia sẻ tầm nhìn kiến trúc của chúng tôi và cách bạn tính nó ra và sau đó thực hiện nó. Khi làm như vậy, chúng ta sẽ liên lạc về vai trò mà các kiến trúc sư trong quá trình này và những sự kiện cơ bản mà chúng ta đã thấy khi đưa ra quyết định về cơ chế của các dự án phần mềm. Không cần phải nói, kinh nghiệm của chúng ta chỉ là của chúng ta. Mặc dù chúng tôi đã làm việc trên một số dự án thú vị với các kích cỡ khác nhau, nhưng kinh nghiệm của chúng tôi vẫn còn hạn chế với những gì chúng tôi đã nhìn thấy và những sai lầm chúng tôi đã thực hiện. Tuy nhiên, chúng tôi muốn suy nghĩ về cuốn sách này như một cái cớ để bạn bắt đầu một cuộc trò chuyện dài và tập trung hơn trong nhóm của bạn về cách bạn thực sự xây dựng những thứ hoạt động.

Lưu ý: Mặc dù một số định nghĩa bạn tìm thấy trong cuốn sách này đến từ các tiêu chuẩn quốc tế, một số khác phản ánh quan điểm, kinh nghiệm và cảm xúc cá nhân của bạn. Chúng tôi sẽ giới thiệu những thực tiễn tốt nhất về kiến trúc, nhưng chúng tôi cũng sẽ nhận xét từ kinh nghiệm của riêng mình. Chúng tôi hy vọng sự kết hợp này sẽ giúp bạn liên kết các tiêu chuẩn quốc tế, khô khan, với những gì bạn sẽ thực hiện trong thế giới thực. Chúng tôi hy vọng sự kết hợp này sẽ giúp bạn liên kết các tiêu chuẩn quốc tế, khô khan, với những gì bạn sẽ thực hiện trong thế giới thực.

**Kiến trúc phần mềm là gì?**

Một trong những tác giả của cuốn sách này, trong quá khứ, thường xuyên tương tác với một studio kiến trúc. Một ngày, một câu hỏi xuất hiện để thảo luận: kiến trúc là gì? Nó là một nghệ thuật? Hay nó chỉ là xây dựng cho một khách hàng?

Trong phần mềm, thuật ngữ kiến trúc chính xác đề cập đến việc xây dựng một hệ thống cho một khách hàng. Đó là nó: không hơn không kém.

Điều này nói, việc tìm kiếm một mô tả chi tiết hơn về những gì liên quan đến việc xây dựng một hệ thống cho khách hàng là phần khó nhất. Chúng ta thường tự hỏi liệu có thực sự có một điểm trong nó. Một triệu lần chúng ta tự hỏi mình câu hỏi này, và một triệu lần chúng ta cùng một câu trả lời. Thật khó, nhưng hãy thử xem. Tồi tệ nhất, nó kết thúc là một phiên thảo luận, và động não là không bao giờ là một điều xấu.

Vì vậy, hãy cố gắng làm rõ "kiến trúc phần mềm" là gì, hoặc, ít nhất, những gì chúng tôi dự định sẽ là:

Lưu ý Các thuật ngữ kiến trúc đi song song với thuật ngữ kiến trúc sư. Kiến trúc sư là một vai trò, và chúng ta sẽ thảo luận ở phần sau về những trách nhiệm chúng ta liên kết với vai trò của một kiến trúc sư. Tuy nhiên, thuật ngữ kiến trúc sư vẫn làm cho người ta nghĩ đến một nghề nghiệp. Thật không may, sự chuyên nghiệp đằng sau thuật ngữ không phải là phổ quát. Tiêu đề đó được đánh giá bằng các tính từ như doanh nghiệp, giải pháp, an ninh, …. Trừ khi chúng tôi chỉ định một cách khác, chúng tôi muốn nói đến một loại hình chuyên nghiệp đôi khi được mô tả như một kiến trúc sư phần mềm hoặc kiến trúc sư giải pháp

**Áp dụng các nguyên tắc kiến trúc vào phần mềm**

Nhiều người trong chúng tôi đã lớn lên như các chuyên gia phần mềm đầu tiên nghĩ rằng phần mềm có liên quan nào đó với thế giới xây dựng, và sau đó chiến đấu với khái niệm đó. Sự tương tự này lần đầu tiên được sử dụng trong ngành công nghiệp phần mềm để thể hiện sự cần thiết phải lên kế hoạch và thiết kế trước khi xây dựng chương trình máy tính. Tuy nhiên, có một sự khác biệt căn bản giữa thiết kế và xây dựng các cấu trúc có thể sinh sống và thiết kế và xây dựng các hệ thống phần mềm có ích.

Phần mềm tồn tại để tự động hoá quy trình kinh doanh và hành động của con người; Các hiện vật kiến trúc dân dụng tồn tại như một dịch vụ cho cộng đồng. Thông thường, các dịch vụ kiến trúc dân dụng được lên kế hoạch và thực hiện với những ý định tốt nhất và hữu ích cho số người lớn nhất. Mục tiêu của họ không chỉ để đáp ứng một số ít các bên liên quan. Ngoài ra, các hiện vật kiến trúc dân dụng tốn kém và phức tạp đủ để không ai nghĩ đến việc biến đổi các dự án đã được phê duyệt. Hoặc, ít nhất, các biến thể xảy ra chỉ thỉnh thoảng.

Những điều khác biệt trong phần mềm

Phần mềm thường được tạo ra cho một nhóm nhỏ các bên liên quan; Một số người trong số họ phải trả tiền từ túi của họ để có được một cái gì đó giúp tổ chức của họ hoạt động tốt hơn. Yêu cầu, do đó, được liên tục bổ sung, thêm, tháo gỡ, và thay thế lại. Cách tiếp cận này đối với phần mềm đòi hỏi sự nhanh nhẹn và không phù hợp với ý tưởng về một thiết kế trả trước lớn như bạn có trong kiến trúc dân dụng.

Tóm lại, ngày nay, sự trùng hợp về kiến trúc giữa xây dựng và phần mềm không gần như cách đây vài thập kỷ. Tuy nhiên, nhiều từ điển vẫn liệt kê một sự khiếm khuyết về thuật ngữ của thuật ngữ "kiến trúc". Và kiến trúc phần mềm được mô tả như là "thành phần, tích hợp và tương tác của các thành phần trong một hệ thống máy tính". Đó chắc chắn là một sự khôn ngoan mà mọi người sẽ đồng ý. Tuy nhiên, theo chúng tôi, nó khá chung chung và trừu tượng.

Do đó, nó không hoạt động.

Chúng tôi nghĩ rằng các chuyên gia về phần mềm nên đồng ý về một lời giải thích chi tiết hơn, chia rẽ sự phân biệt đó thành những phần nhỏ hơn và đưa chúng vào trong bối cảnh.

**Xử lý kiến trúc theo quan điểm chuẩn**

Nhiều người dường như quên rằng có một định nghĩa chuẩn cho kiến trúc phần mềm tồn tại. Chính xác hơn, nó đã được khoảng từ tháng 9 năm 2000 khi Viện tiêu chuẩn quốc gia Hoa Kỳ / Viện Kỹ thuật Điện và Điện tử (ANSI / IEEE) chuẩn 1471, "Thực hành được đề xuất cho việc mô tả kiến trúc của các hệ thống phần mềm". Trong những năm qua, tài liệu này cũng đã trở thành Tiêu chuẩn của Tổ chức Tiêu chuẩn Quốc tế về Tiêu chuẩn / Tiêu chuẩn Điện toán Quốc tế (ISO / IEC) 42010. Những người quan tâm đến việc đọc toàn bộ tiêu chuẩn có thể liên kết đến những điều sau URL: <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45991>.

Kiến trúc phần mềm liên quan đến việc tổ chức bất kỳ hệ thống thâm dụng phần mềm nào để giải quyết vấn đề và đạt được sứ mệnh của mình với sự tôn trọng đầy đủ các mối quan tâm của các bên liên quan. Trong tổng hợp cực đoan, đây là định nghĩa về thuật ngữ của thuật ngữ kiến trúc phần mềm bạn nhận được từ các giấy tờ tiêu chuẩn quốc tế.

Các bên liên quan được xác định như tất cả các cá nhân quan tâm hoặc quan tâm đến việc xây dựng hệ thống. Danh sách này bao gồm các nhà xây dựng hệ thống (kiến trúc sư, nhà phát triển, người kiểm tra) cũng như người thâu tóm, người dùng cuối, các nhà phân tích, kiểm toán viên, và các nhà cung cấp thông tin trưởng (CIO).

Mối quan tâm được đặt ra như bất kỳ lợi ích nào mà các bên liên quan có thể có về hệ thống và ảnh hưởng của chúng đối với hệ thống, dù là phát triển, công nghệ, hoạt động, tổ chức, kinh tế, hay pháp lý.

🗏

Lưu ý Chúng ta biết rằng rất nhiều người sợ hãi khi họ chạy qua các từ viết tắt như ISO, ANSI, và tương tự. Chúng tôi cũng có khuynh hướng bỏ qua những tài liệu tham khảo vì họ, chỉ cần nghe có vẻ nhàm chán và ý thức về những liều lượng lý thuyết mà chúng tôi nhận được ở trường đại học. Dù sao, tiêu chuẩn giấy tờ có đầy đủ thông tin mà chúng tôi chỉ cần đồng ý với. Trong chương này, chúng tôi tham khảo các tiêu chuẩn chủ yếu để nói rằng trong một số lĩnh vực (như xử lý yêu cầu của người sử dụng) rất nhiều đã được thực hiện và nó đã được thực hiện tốt, đủ để tái sử dụng nó.

Sơ đồ của một hệ thống phần mềm thâm canh

Hình 1-1 tóm tắt phần cốt lõi của sơ đồ pháp lý mô tả kiến trúc phần mềm theo các bài báo được công nhận bởi ISO, IEC và IEEE.

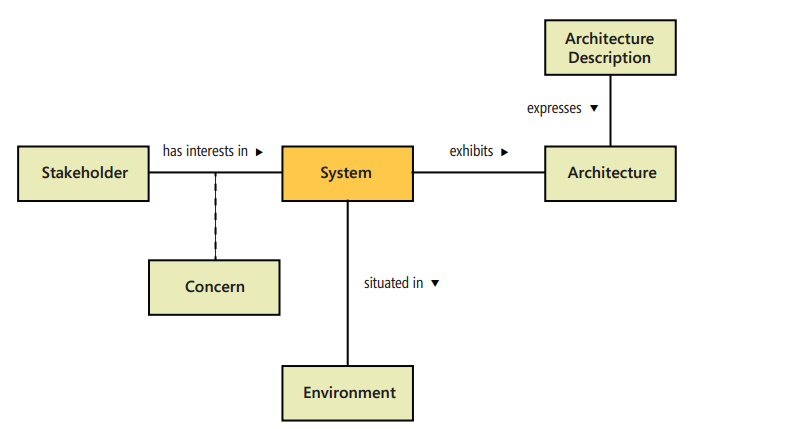


Figure 1 - 1 Sơ đồ tổng kết quan điểm kiến trúc phần mềm từ ISO, IEC, và IEEE.

Một hệ thống sống trong một bối cảnh, và bối cảnh này ảnh hưởng đến việc thiết kế hệ thống bằng cách đưa ra một số quyết định phát triển và hoạt động. Mặc dù các tiêu chuẩn hiện tại hình dung hệ thống như là một thành phần của các thành phần kết nối, kiến trúc cũng thiết lập một số điểm irm khó sửa đổi sau đó. Theo một cách nào đó, việc thể hiện sự phát triển phần mềm về mặt kiến trúc đã tạo ra một số quyết định quan trọng ảnh hưởng đến vòng đời phát triển và cuối cùng là chất lượng của hệ thống kết quả.

**Làm chủ sự pha trộn của kiến trúc phần mềm**

Khi những người mới gia nhập công ty của chúng tôi, chúng tôi cung cấp cho họ - bất kể kỹ năng và vai trò - quan điểm cá nhân của chúng tôi về kiến trúc phần mềm. Chúng tôi là người Ý, và ở Ý chúng tôi nghiêm túc nghiêm túc về cà phê. Khi nói đến cà phê hơi, không có nhiều loại cà phê khác nhau. Phân loại duy nhất được công nhận là giữa cà phê hơi tốt và cà phê hơi không tốt. Nhưng khi chúng tôi nhập Starbucks, giống như mọi người khác, chúng tôi có pha trộn cà phê yêu thích của riêng mình.

Chúng tôi muốn mở rộng sự tương đồng cà phê với kiến trúc phần mềm.

Cũng như cà phê hơi, trong thế giới này chúng ta chỉ thấy kiến trúc tốt và kiến trúc không tốt. Để giữ cho sự tương đồng, chúng tôi chỉ giới hạn ở kiến trúc mà kết quả trong một hệ thống làm việc. Chúng tôi không xem xét ở đây kiến trúc xấu đưa đến kết quả hỏng hệ thống, giống như chúng ta không uống cà phê không tốt!

Khi phục vụ cà phê hơi không tốt, người Ý không phàn nàn, anh ta chỉ cần bước ra khỏi quầy bar và chia sẻ phản hồi thích hợp (và được tôn trọng).

Khi nói đến việc bố trí kiến trúc, giống như chúng ta làm ở Starbucks, chúng ta có tầm nhìn riêng về nó. Tầm nhìn của chúng tôi có thể được mô tả bằng một đồ thị đơn giản với các nút và vòng cung, nhưng chúng tôi cảm thấy nó thể hiện cách chúng tôi nghĩ và cách chúng tôi làm việc (Xem hình 1-2)

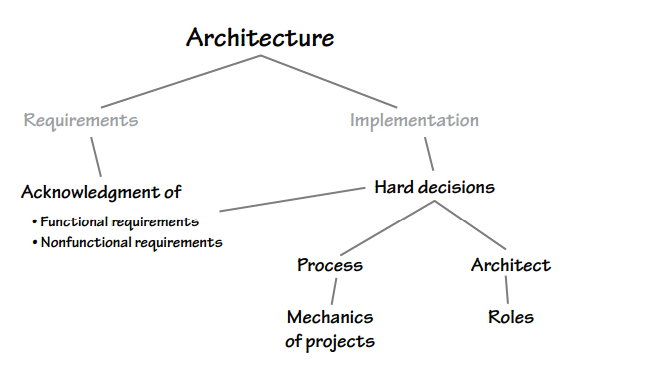


Figure 2Tầm nhìn thiết yếu của chúng ta về kiến trúc phần mềm.

**Yêu cầu xác nhận**

Nhiệm vụ của hệ thống được thể hiện thông qua một bộ yêu cầu. Những yêu cầu này cuối cùng dẫn đến kiến trúc của hệ thống.

Trong các từ ngữ khá trừu tượng, một yêu cầu là một đặc tính của hệ thống có thể là chức năng hoặc phi chức năng. Một yêu cầu chức năng đề cập đến một hành vi mà hệ thống phải cung cấp để làm đầy một kịch bản nhất định. Một yêu cầu phi chức năng đề cập đến một thuộc tính của hệ thống được yêu cầu rõ ràng bởi các bên liên quan.

Có phải sự khôn ngoan của các yêu cầu về chức năng và phi chức năng là điều tiêu chuẩn và được chấp nhận rộng rãi? Trên thực tế, một tiêu chuẩn quốc tế để chính thức hóa các đặc tính chất lượng của hệ thống phần mềm đã tồn tại từ năm 1991.

**Xem nhanh tiêu chuẩn ISO / IEC 9126**

Trên thực tế, không thừa nhận yêu cầu là một trong những nguyên nhân thông thường nhất - thường là nguyên nhân chính dẫn đến thất bại của các dự án phần mềm. Tiêu chuẩn ISO / IEC 9126 tạo ra một tập hợp các đặc tính chất lượng chung được yêu cầu trong các sản phẩm phần mềm. Tiêu chuẩn này xác định sáu gia đình có đặc điểm chất lượng, có khớp nối với 21 đặc tính phụ. Các gia đình chính là tính năng, độ tin cậy, khả năng sử dụng, tính hiệu quả, khả năng bảo trì và tính di động. Bảng 1-1 giải thích chi tiết hơn và liệt kê các đặc điểm chính liên quan đến mỗi bên.

**BẢNG 1-1. Các gia đình có đặc điểm chất lượng theo ISO / IEC 9126**

|  |  |
| --- | --- |
| **Family** | **Diễn giải** |
| Chức năng | Cho biết phần mềm nào đáp ứng được mong đợi. Nó được dựa trên các yêu cầu như Tính phù hợp, tính chính xác, an ninh, khả năng tương tác và tuân thủ các tiêu chuẩn và quy định. |
| Độ tin cậy | Chỉ ra khả năng của phần mềm để duy trì một mức hiệu suất nhất định khi sử dụng trong các điều kiện đặc biệt. Nó dựa trên các yêu cầu như sự trưởng thành, khả năng chịu lỗi và khả năng thu hồi. Sự trưởng thành là khi phần mềm không bị gián đoạn trong trường hợp các lỗi phần mềm bên trong. Khả năng chịu đựng lỗi cho thấy khả năng kiểm soát sự thất bại và duy trì mức độ hành vi nhất định. Khả năng phục hồi cho thấy khả năng phục hồi sau một thất bại. |
| Khả năng sử dụng | Cho biết khả năng của phần mềm được người sử dụng hiểu, sử dụng và hấp dẫn. Nó chỉ ra rằng phần mềm phải tuân theo các tiêu chuẩn và quy định về tính hữu dụng |
| Hiệu quả | Chỉ ra khả năng cung cấp một mức hiệu suất nhất định cả về đáp ứng thích hợp và kịp thời và sử dụng nguồn lực |
| Khả năng bảo trì | Chỉ ra khả năng của phần mềm để hỗ trợ sửa đổi như chỉnh sửa, cải tiến, hoặc thích ứng. Nó dựa trên các yêu cầu như khả năng kiểm tra, tính ổn định, khả năng phân tích và khả năng thay đổi. |
| Khả năng di chuyển | Cho thấy khả năng của phần mềm được chuyển từ nền tảng này sang nền khác và khả năng của nó để cùng tồn tại với các phần mềm khác trong một môi trường chung và chia sẻ các tài nguyên chung. |

Đặc điểm của hai loại: bên ngoài và bên trong. Một đặc điểm bên ngoài là người dùng định hướng và đề cập đến một cái nhìn bên ngoài của hệ thống. Một đặc điểm bên trong là hệ thống định hướng và đề cập đến một cái nhìn nội bộ của hệ thống. Đặc điểm bên ngoài xác định các yêu cầu chức năng; Các đặc tính bên trong xác định những yêu cầu phi chức năng.

Tiêu chuẩn ISO / IEC 25010 mới thay thế ISO / IEC 9126 và đã được ban hành vào tháng 3 năm 2011. ISO 25010 có 8 đặc tính chất lượng sản phẩm và 31 đặc điểm phụ.

**Yêu cầu về chức năng**

Một yêu cầu chức năng đặt ra một chức năng cần thiết trong một hệ thống phần mềm. Một chức năng được mô tả về đầu vào, hành vi và đầu ra. Vấn đề chính của các yêu cầu chức năng là mô tả các hành vi dự kiến. Thật không may, mô tả thường không được rõ ràng và dễ hiểu như nó phải được.

Trong tài liệu ISO / IEC 9126, các yêu cầu về chức năng được mô tả như là khả năng của phần mềm cung cấp các chức năng đáp ứng nhu cầu đã nêu và ngụ ý khi phần mềm được sử dụng trong điều kiện cụ thể. Kiến trúc phần mềm miêu tả những gì; Thiết kế phần mềm mô tả cách thức. Tuy nhiên, chúng tôi tin rằng chúng ta thường quên về trục thứ ba - khi nào.

Bạn có thể nhớ thảm hoạ Ariane 5 hồi năm 1996. Tóm lại, trong ánh sáng ban đầu, tên lửa Ariane 5 đã rơi 40 giây sau khi cất cánh. Theo bản báo cáo chính thức, chiếc tên lửa bị tự hủy do phản ứng dây chuyền do một sự ngoại lệ không được quản lý trong quá trình chuyển đổi một số. Ngoại lệ không được kiểm soát bong ra trong một cách không kiểm soát để trình bày con số quỹ đạo không phù hợp với mô-đun tự huỷ. Giải thích chính xác là bằng chứng của tên lửa đang bị mất không thể đảo ngược trong không gian, tự hủy diệt là cuối cùng, chỉ là một sự lựa chọn khôn ngoan. Trớ trêu là môđun mà trường hợp ngoại lệ đã không được quản lý không phải là chạy vì nó là không cần thiết, tại thời điểm cất cánh. Vì vậy, các kỹ sư biết rằng những con số không phù hợp có thể nhận được trong quá trình cất cánh, nhưng thay vì bẫy ngoại lệ, họ chỉ giả định rằng nó sẽ không bao giờ là cần thiết vì mô-đun này sẽ không hoạt động khi cất cánh!

Do đó, khi kích thước của yêu cầu là vấn đề

Làm thế nào ứng dụng khách hàng dựa trên Internet của bạn sẽ hoạt động như thế nào trong trường hợp kết nối kém? Nếu nó thất bại? Nên cố gắng chờ đợi và phục hồi? Nên nó được tốt đẹp cho người dùng? Nó có nên sử dụng dữ liệu lưu trữ và chuyển sang chế độ tuyến tính? Không có câu trả lời rõ ràng; Tất cả phụ thuộc vào các yêu cầu thuộc các thuộc tính như độ tin cậy và khả năng thu hồi. Đó là trách nhiệm của bạn như là một kiến trúc sư để tìm câu trả lời-đôi khi hỏi nhiều câu hỏi hơn trong một lần thứ hai và chu đáo hơn. Có những câu trả lời dẫn đến một sự hiểu biết tốt hơn về hệ thống, và sau đó, làm giảm khả năng làm điều sai trái hoặc không mã hóa một tính năng yêu cầu.

Lý tưởng nhất, tất cả các yêu cầu chức năng được tập hợp trước khi bắt đầu phát triển và không bao giờ thay đổi khi phát triển đã bắt đầu. Tuy nhiên nhiều khả năng nhiều yêu cầu chức năng sẽ được khám phá hoặc hiểu đầy đủ chỉ trong giai đoạn phát triển. Điều này tạo ra nhu cầu xem xét lại việc thực hiện và đôi khi là kiến trúc của mô đun, và trong trường hợp khó khăn, thậm chí cả các hệ thống con. Không làm điều này, bắt đầu một hội chứng "Big Ball of Bud" khét tiếng mà chúng ta thảo luận trong chương tiếp theo.

Quan trọng Khi bắt đầu sự nghiệp của chúng tôi, chúng tôi dễ bị nhầm lẫn với vàng bất cứ điều gì xuất hiện trong các cuộc phỏng vấn với khách hàng. Giống như một bộ phim hoạt hình Dilbert hài hước, nó giống như chúng ta tự trách mình vì không thể thỏa mãn được các yêu cầu, cho dù có bao nhiêu, vô tổ chức, mâu thuẫn, và từng phần. (Nhân tiện, kiểm tra <Http://dilbert.com/strips/comic/2001-04-14>.). "Khách hàng luôn luôn đúng" là phương châm của chúng tôi, và các yêu cầu của khách hàng giống như Bảng Luật. Qua nhiều năm, chúng tôi đã học được rằng khách hàng luôn có từ bên trong về bất kỳ quyết định chức năng nào, nhưng trách nhiệm của chúng tôi là phải có nhiều lựa chọn. Và chúng tôi xây dựng một loạt các lựa chọn ngoài việc thừa nhận thành công các yêu cầu nguyên liệu mà chúng tôi nhận được từ các cuộc phỏng vấn

**Yêu cầu phi chức năng**

Một yêu cầu phi chức năng đề cập đến một thuộc tính của hệ thống được yêu cầu rõ ràng bởi các bên liên quan. Thuộc tính điển hình là những thứ như khả năng mở rộng, bảo mật, hoặc khả năng truy cập. Mặc dù các yêu cầu về chức năng chủ yếu là về mã và thực hiện nhưng các yêu cầu phi chức năng chủ yếu là về sự lựa chọn kiến trúc và sau đó là các khía cạnh của hệ thống khó thay đổi sau đó. Các yêu cầu phi chức năng có thể ảnh hưởng đến các công nghệ được sử dụng trong một dự án cũng như các giải pháp kỹ thuật.

Ví dụ, làm thế nào bạn có thể giải quyết nhu cầu về khả năng mở rộng cực đoan?

Lưu ý Chúng tôi vẫn không gặp một khách hàng không cho rằng khả năng mở rộng cực cao của ứng dụng danh sách liên lạc là điều quan trọng cho sự sống còn của công ty và sức khoẻ của danh mục đầu tư của nhà đầu tư. Bất kể những gì đang được nói, và đôi khi được thực hiện, hai xu của chúng tôi là nhu cầu về khả năng mở rộng cực kỳ thực sự là khá hiếm. Trừ khi, tất nhiên, bạn đang thực sự xây dựng Twitter tiếp theo hoặc Facebook!

*Khả năng mở rộng* là khả năng của một hệ thống phần mềm để cung cấp hiệu suất tương tự như số lượng người dùng kết nối phát triển, do đó tạo ra rất nhiều công việc và trafic. Đối với một hệ thống để quy mô tốt, đọc và viết phải xảy ra trong thời gian ngắn nhất có thể. Điều này có thể có nghĩa, ví dụ, sử dụng bộ đệm rộng rãi trong việc đọc và sử dụng viết không đồng bộ.

Tương tự, *an ninh* có thể được giải quyết hiệu quả bằng cách thiết kế hệ thống để mô-đun quan trọng có thể được triển khai vào các máy cô lập. *Khả năng tiếp cận* được giải quyết tốt nhất nếu bạn chọn ASP.NET MVC chứ không phải là ASP.NET Web Forms để thực hiện trang web. Điều này là do ASP.NET MVC cung cấp kiểm soát nhiều hơn HTML được hiển thị.

Nói chung, các yêu cầu phi chức năng phải được công nhận cùng lúc với các yêu cầu chức năng. Thứ hai sản xuất chi tiết kỹ thuật phần mềm; Các cựu giúp với chiến lược thực hiện và truyền cảm hứng cho sự lựa chọn công nghệ. Yêu cầu về chức năng và phi chức năng liên quan đến nhau, và cả hai đều có thể thay đổi liên tục. Tuy nhiên, các yêu cầu phi chức năng làm tăng nhu cầu đưa ra một số quyết định khó.

**Lưu ý**: Một số người bạn đã giúp chúng tôi viết cuốn sách này đã có nhận xét vào thời điểm này. Mặc dù họ có thể đồng ý với phần trước, họ muốn làm rõ rằng trong nhiều trường hợp yêu cầu về chức năng cũng ảnh hưởng đến các lựa chọn kiến trúc và các khía cạnh của hệ thống khó thay đổi sau đó. Chắc chắn rồi!

**Yêu cầu thu thập**

Có hàng tấn sách giải thích làm thế nào để thu thập và viết yêu cầu. Vì vậy, bạn có thể học được rằng, ví dụ, một yêu cầu tốt chỉ giải quyết một vấn đề, không phải là một văn bản mơ hồ, có thể dễ dàng bắt nguồn từ yêu cầu của doanh nghiệp hoặc các bên liên quan, không phải lỗi thời, …

**Lưu ý**: Nếu bạn đang tìm kiếm một cuốn sách giải thích lý thuyết về các yêu cầu về phần mềm, chúng tôi khuyên bạn nên sử dụng các mẫu Yêu cầu Phần mềm của Stephen Withall (Microsoft Press, 2007).

Điều gì xảy ra trong thực tế là những người kinh doanh mô tả những gì họ tin rằng họ muốn và các nhà phát triển xây dựng những gì họ tin rằng những người kinh doanh được mô tả. Bất kể nỗ lực nào bạn đưa ra để thu thập yêu cầu, luôn có những điều bị giữ lại, bỏ qua, lãng quên, hoặc chỉ thể hiện theo cách rõ ràng đối với người khác nhưng không phải với người khác. Trên cơ sở của vấn đề truyền thông này, có một thực tế là những người kinh doanh và phát triển sử dụng những từ ngữ khác nhau.

Chúng tôi là những người tin tưởng mạnh mẽ rằng các kiến trúc sư và nhà phát triển nên nói cùng một ngôn ngữ với những người kinh doanh.

Bất kể như thế nào, nói rằng, một chứng từ thực sự là mã hoá, để thực hiện đầy đủ nó, bạn phải hiểu những gì nó được. Thậm chí quan trọng hơn, bạn cần phải hiểu ý nghĩa của nó. Là kiến trúc sư, bạn không nên mong đợi rằng những người kinh doanh hiểu ngôn ngữ của bạn, được làm bằng những thứ như các bảng cơ sở dữ liệu, dịch vụ và các giao thức. Đó là bạn, những người cần thực hiện bất kỳ nỗ lực cần thiết để biết và hiểu các thực thể mà cư trú trong lĩnh vực kinh doanh.

**Quan trọng** Quan điểm của việc có người kỹ thuật nói cùng ngôn ngữ với người kinh doanh là nền tảng của một khái niệm chính mà chúng tôi giới thiệu trong Chương 5, "Khám phá kiến trúc miền". Khái niệm này là ngôn ngữ phổ biến và nó là một trong những trụ cột của Domain- Driven Design (DDD).

**Cách chúng ta giải quyết các yêu cầu**

Một thực hành đơn giản nhưng hiệu quả để xử lý yêu cầu chức năng là nhóm chúng theo các loại, như được mô tả trong Bảng 1-1. Những gì chúng ta thường làm là tạo một tài liệu Microsoft Office Excel với một tab cho mỗi loại. Tiếp theo, chúng ta sẽ đi qua danh sách các yêu cầu mà chúng tôi thu thập được từ các cuộc phỏng vấn và chỉ cần lập bản đồ cho từng thể loại hiện có.

Khi hoàn tất, chúng tôi đọc lại và dừng lại trên tất cả các tab không có sản phẩm hoặc chỉ có một hoặc hai yêu cầu. Nếu không có yêu cầu được tìm thấy, nói, theo thuộc tính của tính di động, chúng tôi dừng lại và tự hỏi nếu chúng ta biết đủ về nó và nếu chúng ta hỏi đủ câu hỏi. Không có yêu cầu về khả năng di chuyển rõ ràng có thể có nghĩa là tính di động không phải là yêu cầu, hoặc nó có thể có nghĩa là chúng ta không biết về nó.

Đặc biệt, tính di động liên quan đến khả năng của phần mềm được sử dụng trong các môi trường khác nhau. Nếu không có yêu cầu như vậy, trong một kết thúc dịch vụ ngay lập tức sẽ đặt câu hỏi liệu phần cuối phía sau chỉ có thể sử dụng được từ khách hàng web hay liệu sự hỗ trợ cho iOS và Android là điều cần thiết

**Kiến trúc là gì và những gì không**

Khi bạn nghĩ về việc tạo ra hoặc xây dựng kiến trúc của một hệ thống phần mềm, bạn irst cố gắng xác định một bộ sưu tập có thể của các thành phần tương tác đó, tất cả cùng nhau, hoàn thành yêu cầu sứ mệnh. Trong tiêu chuẩn quốc tế, không có đề cập đến bất kỳ phương pháp nào bạn có thể sử dụng để phân hủy hệ thống thành nhiều phần chi tiết hơn. Hãy nói rằng trong bước đầu tiên bạn có được một kiến trúc khái niệm và một số quan điểm khác nhau của nó. Trong bước thứ hai, bạn cần phải có được gần gũi hơn với một kiến trúc vật lý và chức năng. Cách bạn nhận được ở đó là chủ quan, mặc dù cách tiếp cận từ trên xuống có vẻ là một chiến lược hợp lý. Bạn phân hủy các thành phần thành các phần nhỏ hơn và nhỏ hơn, và từ đó bạn bắt đầu xây dựng.

Việc thực hiện quá trình phân tích thực tế phụ thuộc vào phương pháp được lựa chọn cho dự án - bạn càng nhanh nhẹn, quá trình phân tích sẽ lặp đi lặp lại và diễn đạt theo các bước nhỏ hơn và thường xuyên hơn. Đầu ra của quá trình phân tích là một tập hợp các đặc điểm cho nhóm phát triển. Ngoài ra, nội dung và định dạng của các chi tiết kỹ thuật phụ thuộc vào phương pháp luận. Bạn càng nhanh nhẹn, sự tự do và sự độc lập càng tốt cho các nhà phát triển khi triển khai kiến trúc.

**Xác định ranh giới giữa kiến trúc và thực hiện**

Các thành phần cấu thành mà bạn xác định trong khi phá vỡ hệ thống đại diện cho các chức năng logic được thực hiện theo một cách nào đó. Việc thiết kế các thành phần, giao diện, trách nhiệm của chúng và hành vi của chúng là một phần không thể thiếu của kiến trúc. Tuy nhiên, có một biên giới tách biệt kiến trúc khỏi thực thi.

Biên giới này rất quan trọng để xác định bởi vì, trong một phạm vi rộng, nó giúp xác định vai trò của một nhóm phát triển. Đặc biệt, nó đánh dấu ranh giới giữa kiến trúc sư và nhà phát triển. Trong những năm qua, chúng tôi đã biết rằng các kiến trúc sư và nhà phát triển không khác nhau, ví dụ, các loại trái cây, như táo và cam. Họ là cùng một loại trái cây. Tuy nhiên, nếu họ là táo, chúng giống như táo đỏ và táo xanh - hương vị khác biệt, nhưng không phải là một loại trái cây khác nhau. Và hương vị cũng không ngon miệng.

Biên giới giữa kiến trúc và thực hiện được công nhận khi bạn đạt đến một hộp đen về hành vi. Một hộp hành vi màu đen chỉ là một phần chức năng có thể dễ dàng thay thế hoặc tái cấu trúc mà không có sự hồi quy đáng kể và không ảnh hưởng đến phần còn lại của kiến trúc. Những gì phía trên hộp hành vi màu đen có thể có ý nghĩa kiến trúc và có thể yêu cầu bạn phải đưa ra quyết định khó thay đổi.

Vậy định nghĩa của chúng ta về kiến trúc tốt là gì? Đó là kiến trúc, trong đó tất cả các quyết định khó thay đổi đều đúng.

Khoa học của các quyết định khó

Có nhiều khía cạnh và tính năng của một hệ thống phần mềm rất khó (chỉ khó, không phải là không thể) để thay đổi khi bạn bước vào quá trình phát triển. Và có những khía cạnh và tính năng có thể thay đổi bất cứ lúc nào mà không cần nỗ lực rất nhiều và không có tác động đáng kể lên hệ thống.

Trong cuốn sách của ông Mẫu Kiến trúc Ứng dụng Doanh nghiệp (Addison-Wesley, 2002), Martin Fowler đã đặt nó khá đơn giản:

*Nếu bạn thấy rằng cái gì đó dễ thay đổi hơn bạn tưởng, thì nó không còn là kiến trúc. Trong kiến trúc kết thúc đun sôi xuống những thứ quan trọng - bất kể đó là gì.*

Để tóm tắt lại, chúng tôi nghĩ rằng dưới ô của thuật ngữ kiến trúc giảm mọi thứ bạn phải nghiêm túc ở giai đoạn khá sớm của dự án. Kiến trúc là cuối cùng về việc xác định các quyết định quan trọng để thực hiện chính xác càng sớm càng tốt trong dự án, nhưng mong muốn bạn có thể làm cho họ càng muộn càng tốt

**Quyết định khó khăn ở khắp mọi nơi**

Khi chúng ta nói về các quyết định kiến trúc cứng, chúng ta không nhất thiết phải đề cập đến các quyết định về các điểm thiết kế có thể rất tốn kém và tốn kém để thay đổi sau này. Các quyết định khó thay đổi ở mọi nơi và nằm trong phạm vi từ định nghĩa các lớp khái niệm đến chữ ký của một hàm tạo.

Để minh hoạ điểm này, chúng ta hãy đi qua một số ví dụ về các điểm kiến trúc có thể chạy vào giới hạn ngân sách và thời hạn nếu bạn phải chạm vào chúng trong quá trình thực hiện dự án

Ví dụ đầu tiên là thay đổi tổ chức logic kinh doanh. Có một số cách tiếp cận để thiết kế logic kinh doanh: Kịch bản giao dịch, Mô đun bảng, Mô hình miền, Mô hình Tên miền với sự tách biệt giữa các lệnh và truy vấn, và Sự kiện Sourcing, chỉ để đặt tên cho một vài. Một khi bạn đã chọn tham gia, chẳng hạn như Bảng Module (có nghĩa là về cơ bản là bạn sẽ đặt logic trong các thành phần kho lưu trữ được xây dựng xung quanh các bảng cơ sở dữ liệu như bạn có thể đã làm trong nhiều năm qua DataSet và DataTables), di chuyển sang mô hình miền hầu như không thể làm được gì trong buổi chiều. Sự thay đổi hướng này đòi hỏi những thay đổi không thay đổi trong lớp truy cập dữ liệu và trong lớp ứng dụng (dịch vụ), và có lẽ cũng trong lớp trình bày.

Một ví dụ thứ hai là chuyển sang một thư viện khác nhau cho cùng một nhiệm vụ. Giả sử bạn phát triển một số chức năng xung quanh một thư viện nhất định. Một ngày nọ, khách hàng bật lên và cho bạn biết rằng một chính sách công ty mới ngăn cản bộ phận CNTT mua các sản phẩm từ một nhà cung cấp cụ thể. Bây giờ bạn có một yêu cầu phi hoạt động mới, bất ngờ để giải quyết, nhưng ở những gì chi phí? Trong trường hợp tốt nhất, bạn có thể có được một công cụ tương tự từ một nhà cung cấp được ủy quyền hoặc, có lẽ, bạn có thể tự xây dựng một công cụ tương tự. Ngoài ra, bạn có thể xem xét đưa ra một sự thay đổi triệt để vào kiến trúc làm cho thư viện không cần thiết

**Lưu ý** Nếu bạn tham dự một số hội nghị mà Dino đã nói trong vài năm qua, có thể bạn đã từng nghe về trải nghiệm này rồi. Tóm lại, công ty của Dino đã đặt giá thầu để xây dựng một ứng dụng di động nhạy cảm với thời gian cho bốn nền tảng khác nhau: iOS, Android, Windows Phone và BlackBerry. Giá thầu được thực hiện trên giả định rằng PhoneGap sẽ là một lựa chọn khả thi. Thật không may, một mẫu thử nghiệm được xây dựng với PhoneGap đã bị khách hàng từ chối, đưa đội trở lại hình vuông và bốn tuần nữa để hoàn thành công việc.

Nhóm nghiên cứu đã bị bỏ lại với một trong hai lựa chọn khó chịu: thực hiện công việc bốn lần trong một khoảng thời gian hoặc sáng chế một điều gì mới. Nhóm đã quản lý để sử dụng lại một số đánh dấu PhoneGap trên trang web dành cho điện thoại di động và kết hợp các lượt xem web trong các khung của bốn ứng dụng gốc cơ bản và tĩnh và chỉ dựa trên đồ họa. Chức năng-khôn ngoan, nó làm việc tuyệt vời, và thậm chí đã có thời gian để tồn tại một ngớ ngẩn từ chối của ứng dụng của Apple. Tuy nhiên, căng thẳng-khôn ngoan, đó là một sự đau đớn trong phần kết thúc phía sau.

Ví dụ về quyết định khó khăn thứ ba và có thể là bất thường đang thay đổi một sửa đổi của một thành viên lớp. Nói chung, khi bạn sử dụng các sửa đổi kín và ảo, bạn sẽ nhận một trách nhiệm không nhỏ. Trong C #, mặc định mỗi lớp được mở và mỗi phương thức trên một lớp không phải là ảo. Trong Java, thay vào đó, mọi thứ trở nên khác biệt với các phương thức, tất cả ảo đều theo mặc định. Từ góc độ thiết kế, các lớp kín là thích hợp hơn. Trong thực tế, khi một lớp được đóng dấu từ đầu bạn biết nó - và bạn tạo mã cho phù hợp. Nếu có gì xảy ra sau đó để biện minh cho sự thừa kế của lớp đó, bạn có thể thay đổi nó để không bị lộ ra mà không vi phạm những thay đổi và không ảnh hưởng đến khả năng tương thích. Gần như có thể nói cho các phương pháp ảo và khả năng hiển thị của các lớp học và các thành viên lớp học, vốn luôn là tư nhân theo mặc định. Phương pháp ngược lại không hoạt động trơn tru. Bạn thường không thể đóng dấu một lớp hoặc đánh dấu một phương thức ảo là không ảo mà không có khả năng phá vỡ một số mã hiện có.

**Bối cảnh làm cho quyết định khó khăn**

Khoảng bốn năm trước, trong một cuộc hội thảo, chúng tôi nghe một người nói nổi bật đã nói lên một sự thật đúng lúc đó. Ông tuyên bố rằng điện thoại di động và điện toán đám mây là nhức đầu lớn nhất đối với CTOs khi họ cố gắng xác định xem phải làm gì với cả hai.

Bộ nhớ nhỏ này giới thiệu một loại quyết liệt cứng rắn về kiến trúc. Nó xảy ra khi bạn biết rằng một phần công nghệ nhất định, hoặc có lẽ là một mô hình, có thể hữu ích nhưng bạn không có bằng chứng cụ thể cho điều đó. Bạn nhìn xung quanh để nghiên cứu trường hợp và mong muốn tìm thấy chúng, hoặc chỉ là bất kỳ loại văn học có thể giúp bạn thống trị trong một cái gì đó hoặc loại trừ cái gì.

Các quyết định về thực tiễn tốt nhất được biết đến là dễ thực hiện vì có nhiều nghiên cứu trường hợp. Tuy nhiên, việc đưa ra các quyết định về các công nghệ mới, như NoSQL, thực sự là khó khăn vì các nghiên cứu tình huống về các ứng dụng trong thực tế rất ít và thậm chí còn mang tính thiên vị.

**Quá trình kiến trúc**

Được xử lý bởi kiến trúc sư trưởng, yêu cầu được thông báo cho nhóm phát triển và sau đó được triển khai. Tất cả mọi người sẽ đồng ý rằng kiến trúc tốt và thiết kế tốt tốc độ lên sự phát triển thực tế của mô-đun. Nhưng làm thế nào bạn sẽ thực sự đi về xây dựng các ứng dụng?

Về nguyên tắc, xây dựng phần mềm là nỗ lực chung của ba nhóm người: quản lý dự án, phát triển, và kiểm tra chất lượng đảm bảo (QA). Sự phân chia nhiệm vụ gần như hoàn hảo về mặt lý thuyết. Quản lý dự án chịu trách nhiệm về các đặc tính sản phẩm, định nghĩa về các tiêu chuẩn chất lượng, và tiến độ. Phát triển đảm bảo rằng các thuật toán lý tưởng được sử dụng và mã được viết theo các mẫu phù hợp nhất. Cuối cùng, nhóm QA nhấn mạnh hành vi của ứng dụng, với mục đích tuyên bố vi phạm nó để nó có thể được cập nhật và mạnh hơn.

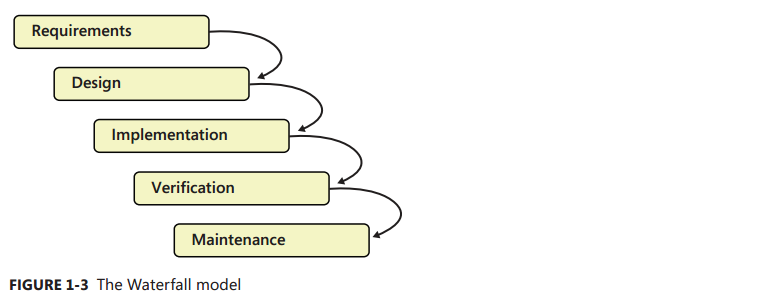
Ba đội phải làm việc cùng nhau tại bất kỳ thời điểm nào trong tiến trình, mặc dù một số việc quản lý dự án cần được thực hiện trước khi bắt đầu phát triển và thử nghiệm không thể bắt đầu cho đến khi một số mã được xây dựng. Toàn bộ vấn đề phát triển phần mềm là xác định các quy tắc tương tác cho các nhóm và xác định cách họ tương tác. Phương pháp phát triển phần mềm cố gắng thực hiện việc này.

Phương pháp luận chủ yếu là hai loại chính: Thác nước và Agile. Các dự án dựa trên thác nước có xu hướng tiến hành tuần tự qua một số giai đoạn, lên đến cực điểm với việc phát hành phần mềm. Các dự án dựa trên Agile có xu hướng xoay vòng và lặp lại các tác vụ nhiều lần cho đến khi ứng dụng đã sẵn sàng hoặc, đôi khi, cho đến khi thời hạn trôi qua. Việc chọn phương pháp luận, theo cách nào đó, sẽ xác định loại quy trình kiến trúc.

**Kiến trúc Upfront**

Kiến trúc trả trước là khi mọi thứ phải được đặt trong đá trước khi bạn bắt đầu. Rất nhiều công việc thiết kế được thực hiện trước khi bắt đầu viết mã. Mã hóa chủ yếu được xem như là bản dịch thực tế của những ý tưởng tốt lành vào biên soạn các hướng dẫn.

Mô hình thác nước bắt đầu từ những năm 1970. Đây là một mô hình trong đó phát triển phần mềm tiến hành từ giai đoạn này sang giai đoạn tiếp theo theo cách thuần túy. Về cơ bản, bạn chuyển sang bước N +1 chỉ khi bước N hoàn thành 100 phần trăm và tất cả đều hoàn hảo với nó. Hình 1-3 cho thấy một mẫu của mô hình thác nước



Trong thế giới thực, một số chồng chéo giữa các giai đoạn là có thể chấp nhận được, nhưng vẫn nhấn mạnh vào việc lập kế hoạch, tạo ngân sách và sắp xếp lịch trình công việc cùng một lúc. Cách tiếp cận này rất phù hợp với khách hàng vì nó mang lại cho họ một số điều chắc chắn rõ ràng. Thực tế là khác nhau, tuy nhiên. Nó không phải là không phổ biến mà các dự án kết thúc chạy qua ngân sách và thời hạn và đôi khi thậm chí không cung cấp cho các yêu cầu dự kiến. Kết quả nghèo nàn này phần lớn là do nỗ lực thiết kế lớn được yêu cầu lên phía trước, mà chỉ cần không nhận được cùng với sự năng động thực tế của phần mềm hiện đại.

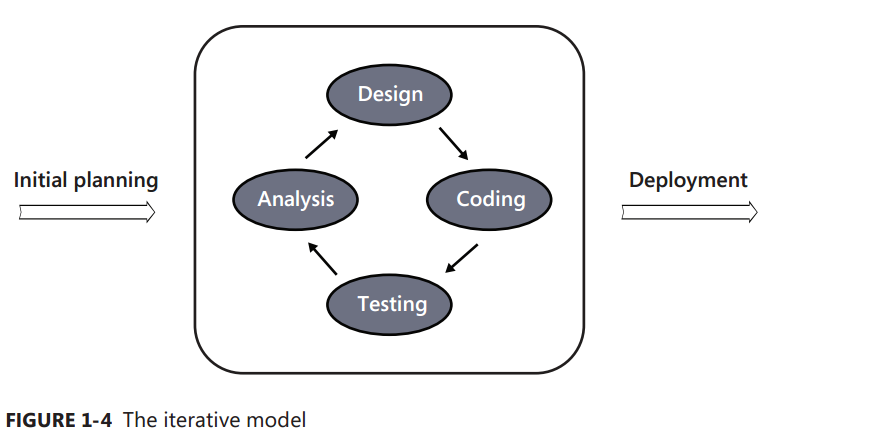
Vấn đề gốc của thiết kế phần mềm là các yêu cầu thay đổi nhanh chóng theo thời gian đến mức một số yêu cầu có thể khác nhau do thời gian phát triển kết thúc. Không khách hàng sẵn sàng trả tiền cho phần mềm mà họ biết sẽ không đáp ứng được tất cả các yêu cầu. Đồng thời, khách hàng không thích ở tất cả để chịu chi phí biến đổi có thể phát triển khi những thay đổi và bổ sung được áp dụng cho hiện vật.

Kết quả là, mô hình Thác nước được coi là một điều của quá khứ, và bạn có thể đổ lỗi cho sự sụp đổ của nó trên toàn bộ ý tưởng rằng phát triển phần mềm là một hình thức kỹ thuật.

**Kiến trúc mới nổi**

Hôm nay của quá trình kiến trúc là bất kỳ đội nào nên bắt đầu phát triển càng sớm càng tốt để có được phản hồi sớm và cải tiến phần mềm dựa trên mã thực. Điều này có nghĩa là chuyển động mọi thứ nhanh chóng, chấp nhận và thậm chí chấp nhận thay đổi, cung cấp một số giá trị sớm và chào đón phản hồi.

Kiến trúc mới nổi là một quá trình kết quả từ việc xây dựng gia tăng của phần mềm. Sau khi khởi động ban đầu, dự án đi qua một loạt các phép lặp bao gồm thiết kế, mã hóa và thử nghiệm. Mỗi lần lặp lại tạo ra một phiên bản có thể phân phối nhưng không đầy đủ của hệ thống. Mỗi lần lặp lại, nhóm nghiên cứu đi vào thay đổi thiết kế và thêm các chức năng mới cho đến khi hoàn thành đầy đủ các yêu cầu cụ thể. Hình 1-4 cung cấp một cái nhìn đồ họa về quá trình lặp.



Sự phát triển lặp lại tạo thành nền tảng của các phương pháp Agile. Thuật ngữ Agile được lựa chọn một cách có chủ ý để tượng trưng cho sự phản đối rõ ràng đối với các phương pháp nặng như mô hình thác nước. Các nguyên tắc đằng sau các phương pháp Agile được liệt kê trong "Manifesto Agile", được xuất bản lần đầu tiên vào năm 2001 và bạn có thể ind tại <http://agilemanifesto.org>.

Khi một dự án Agile bắt đầu, chỉ có một vài yêu cầu có thể được thực hiện đầy đủ, nhưng bạn biết vì một thực tế là nhiều người sẽ hiển thị hoặc cần phải được làm rõ hơn giữa thời điểm này và kết thúc của dự án. Với một tư duy Agile, đây không phải là vấn đề. Phát triển được khớp nối trong lặp lại. Khi bắt đầu lặp lại, bạn thương lượng với khách hàng tập con của các yêu cầu hiện tại cần được thực hiện. Trong quá trình lặp lại, bạn tập trung vào một yêu cầu duy nhất tại một thời điểm và thực hiện nó. Khi kết thúc lặp đi lặp lại, bạn cung cấp một phần mềm làm việc. Nó có thể là không đầy đủ, nhưng nó hoạt động. Tiếp theo bạn đi với một lần lặp lại tập trung vào một bộ yêu cầu khác. Nếu có gì đó thay đổi trong thời gian chờ đợi hoặc đã được chứng minh là sai, thì việc sắp xếp lại theo thứ tự. Và quá trình này tiếp tục cho đến khi không có gì thêm nữa. Chiều dài lặp lại được tính theo tuần - thường là hai tuần. Trong một từ, một quy trình nhanh nhẹn là nhanh nhẹn đủ để phản ứng với những thay đổi. Và những thay đổi trong kinh doanh là nguyên tắc, chứ không phải ngoại lệ.

Phương pháp Agile là một thuật ngữ chăn. Khi bạn tham khảo một phương pháp Agile, bạn không nói chuyện rất chính xác. Bạn có nghĩa là phương pháp luận, thực sự?

Các phương pháp Agile phổ biến nhất cho phát triển phần mềm là Extreme Programming (XP). Trong XP, các giai đoạn được thực hiện trong quá trình lặp lại rất ngắn, mất hai tuần để chấm dứt. Mã và thiết kế tiến hành song song. Để biết thêm thông tin về XP, hãy truy cập <http://www.extremeprogramming.org>.

Scrum là một phương pháp luận Agile phổ biến, nhưng nó nhằm mục đích quản lý các dự án hơn là chỉ phát triển mã. Scrum không phải là quy định cho bất kỳ mô hình phát triển phần mềm, nhưng nó hoạt động rất tốt với XP như là một phương pháp để phát triển mã. Để biết thêm thông tin về Scrum, hãy xem Quản lý dự án Agile với Scrum của Ken Schwaber (Microsoft Press, 2004).

**Quan trọng** Kiến trúc Agile quan trọng đôi khi được trình bày như một mâu thuẫn hình dung pháp, như nói rằng nếu bạn nhanh nhẹn, bạn không phân tích kiến trúc, bạn chỉ cần bắt đầu viết mã, tập trung vào tính năng và hoàn toàn bỏ qua mô hình hóa. Thành thật mà nói, chúng tôi cho rằng mặc dù Agile không loại trừ bất kỳ hành vi như vậy, tầm nhìn này chủ yếu là huyền thoại. Theo kinh nghiệm của chúng tôi, Agile hầu hết chỉ có nghĩa là ngược lại và kiến trúc và các mối quan tâm về mô hình được giải quyết trong toàn bộ chu trình sống phát triển.

**Kiến trúc mới nổi với phân tích trả trước**

Nó có thể được đề cập đến một tùy chọn thứ ba mà ngồi ở đâu đó giữa Waterfall và Agile. Đây là một quá trình bao gồm rất nhiều phân tích trả trước sơ bộ và sau đó bắt đầu với một phương pháp luận Agile cổ điển. Một số gọi đây là cách tiếp cận nhanh Agile; Một số cũng giới thiệu nó như là một điều không sprint.

Nói chung chúng ta thấy rằng tất cả các phương pháp luận phát triển phần mềm chia sẻ một vài thuộc tính chung: một số giai đoạn để đi qua, một số lặp để tạo ra phần mềm, và một khoảng thời gian điển hình cho một lần lặp lại. Tất cả các giai đoạn thực hiện theo tuần tự, và luôn có ít nhất một lần lặp kết thúc bằng việc phân phối phần mềm. Sự khác nhau giữa các phương pháp luận là tất cả theo thứ tự trong đó giai đoạn được nhập, số lượng lặp lặp lại yêu cầu, và thời lượng lặp lại đơn. Sau khi chấp nhận tiền đề này, bước đầu tiên để áp dụng các phương pháp Agile nhỏ hơn nhiều so với lúc đầu bạn nghĩ.

**Ai là kiến trúc sư?**

Như bạn đã thấy, kiến trúc chủ yếu là về các quyết định khó thay đổi. Và ai đó phải đưa ra những quyết định này.

Thiết kế của kiến trúc dựa trên phân tích các yêu cầu. Phân tích xác định hệ thống dự kiến sẽ làm gì; Kiến trúc xác định làm thế nào để làm điều đó. Và ai đó phải kiểm tra cái gì để xác định những sai lầm.

Kiến trúc sư là chuyên nghiệp buộc các yêu cầu và yêu cầu kỹ thuật. Nhưng trách nhiệm của kiến trúc sư là gì? Và các kỹ năng cần thiết là gì?

**Trách nhiệm của một kiến trúc sư**

Theo tiêu chuẩn ISO / IEC 42010, một kiến trúc sư là người, đội, hoặc tổ chức chịu trách nhiệm về kiến trúc của hệ thống. Kiến trúc sư tương tác với các nhà phân tích và người quản lý dự án, đánh giá và gợi ý các lựa chọn cho hệ thống, và điều phối một nhóm các nhà phát triển.

Kiến trúc sư tham gia vào tất cả các giai đoạn của quá trình phát triển, bao gồm phân tích các yêu cầu và thiết kế của kiến trúc, thực hiện, thử nghiệm, tích hợp và triển khai.

Hãy mở rộng các trách nhiệm chính của kiến trúc sư: thừa nhận các yêu cầu, phá vỡ hệ thống xuống các hệ thống con nhỏ hơn, xác định và đánh giá các công nghệ, và xây dựng các quy trình cụ thể.

**Thừa nhận yêu cầu**

Trong một dự án phần mềm, một vài điều xảy ra trước khi kiến trúc sư tham gia. Hàng loạt nhà phân tích, quản trị viên CNTT, và giám đốc điều hành gặp, thảo luận, đánh giá, và đàm phán. Một khi nhu cầu về một hệ thống mới hoặc cập nhật được đánh giá và ngân sách được tìm thấy, các nhà phân tích bắt đầu gợi ý các yêu cầu, thường dựa trên kiến thức của họ về các quy trình kinh doanh, công ty, bối cảnh và phản hồi từ người dùng cuối.

Khi danh sách các yêu cầu đã sẵn sàng, trong nhiều trường hợp, người quản lý dự án gặp gỡ kiến trúc sư và đưa ra gói, nói nhiều hơn hoặc ít hơn, "Đây là điều mà chúng tôi (chúng tôi nghĩ) muốn; Bây giờ bạn xây dựng nó. "

Kiến trúc sư thừa nhận các yêu cầu và nỗ lực để chúng được chấp nhận và thực hiện trong thiết kế.

**Quan trọng** Một khoảnh khắc trước đây, chúng tôi đã đề cập đến một vai trò khác: người quản lý dự án. Người quản lý dự án là một vai trò có thể khác biệt trong các công ty khác nhau. Chúng tôi nghĩ nó là vai trò chịu trách nhiệm quyết định phương pháp luận, lập kế hoạch công việc, theo dõi, báo cáo và phục vụ như một liên kết có giá trị giữa những người kỹ thuật và doanh nhân. Vai trò này có thể được thực hiện bởi cùng một cá nhân đóng vai trò kiến trúc sư. Khi điều này xảy ra - và nó không phải là yêu cầu bất thường - thấp trực tiếp từ các chuyên gia về tên miền cho nhóm phát triển. Nếu những người khác ở giữa, nguy cơ chúng ta thấy là kiến thức về miền bị ô nhiễm, giống như trò chơi của trẻ em nơi mà một đứa trẻ thì thầm một cụm từ vào tai của đứa trẻ. Còn thứ hai nói chuyện với đứa trẻ thứ ba và cứ tiếp tục, cho đến khi cảm giác của cụm từ ban đầu bị mất hoàn toàn. Điều quan trọng là các yêu cầu được thể hiện bằng Ngôn ngữ phổ biến thấp từ các chuyên gia về miền đến nhóm phát triển mà không có người môi giới hoặc chỉ trong một lớp thông qua.

**Phá vỡ hệ thống**

Dựa vào các yêu cầu, kiến trúc sư thể hiện hệ thống tổng thể như là một thành phần của các hệ thống con nhỏ và các thành phần hoạt động trong các quy trình. Khi làm như vậy, kiến trúc sư hình dung các lớp logic, các dịch vụ hoặc cả hai. Sau đó, dựa trên bối cảnh, kiến trúc sư quyết định về giao diện của lớp, mối quan hệ của chúng với các lớp khác và mức độ định hướng dịch vụ mà hệ thống yêu cầu.

**Lưu ý** Ở giai đoạn này, kiến trúc sư đánh giá các mẫu kiến trúc khác nhau. Phân lớp là một sự lựa chọn phổ biến và là sự lựa chọn phổ biến nhất mà chúng ta đang theo đuổi trong cuốn sách này. Sự sắp xếp lớp đòi hỏi sự phân bố theo chiều dọc của các chức năng. Phân vùng là một phương pháp tiếp cận khác, ở đó tất cả các phần đều ở cùng mức hợp lý và phân tán quanh một số thực thể được chia sẻ - chẳng hạn như mô hình đối tượng hoặc cơ sở dữ liệu. Kiến trúc hướng dịch vụ (SOA) và Kiến trúc lục giác (HA) là những mô hình có xu hướng có các thành phần (các dịch vụ trong SOA, các bộ điều hợp trong HA) và tương tác ở cùng một mức logic. Micro-service vụ là một mô hình kiến trúc gần đây tập trung vào ý tưởng về các thành phần chuyên biệt và bị cô lập.

Thiết kế tổng thể sẽ phù hợp với các mục tiêu và yêu cầu của doanh nghiệp. Đặc biệt, thiết kế tổng thể sẽ được điều khiển bởi các yêu cầu; Nó sẽ không dẫn các yêu cầu.

Kết quả kiến trúc được lấy cảm hứng từ các hướng dẫn chung, chẳng hạn như giảm thiểu sự ghép nối giữa các thành phần, cung cấp mức độ gắn kết cao nhất có thể trong các thành phần và cho mỗi thành phần một bộ trách nhiệm rõ ràng.

Kết quả kiến trúc cũng được thúc đẩy bởi các yêu cầu phi chức năng, chẳng hạn như bảo mật, khả năng mở rộng và công nghệ được phép hoặc bị từ chối. Tất cả các khía cạnh này đưa ra các trở ngại khác và, ở một mức độ nào đó, phân cách không gian nơi kiến trúc sư có thể tìm kiếm các giải pháp.

Cuối cùng, kiến trúc sư cũng chiến lược về các nhà phát triển cá nhân hoặc các đội phát triển, với mỗi thành phần dựa trên cách bố trí của hệ thống.

**Quan trọng** Không có sự thật tuyệt đối trong kiến trúc phần mềm. Và không có quy tắc toán học (hoặc các mã xây dựng như trong kỹ thuật kết cấu) để giúp đưa ra lựa chọn. Công ty X có thể tìm thấy kiến trúc A thành công trong cùng một thời điểm công ty Y đang di chuyển ra khỏi nó để nắm lấy kiến trúc B. Thực tế tốt đẹp là cả hai có thể là hoàn toàn đúng. Bối cảnh là vua, và như vậy là ruột cảm giác.

**Xác định và đánh giá các công nghệ**

Sau khi thừa nhận các yêu cầu và thiết kế các lớp của hệ thống, bước tiếp theo của kiến trúc sư đòi hỏi phải lập bản đồ các thành phần logic vào công nghệ và sản phẩm.

Kiến trúc sư thường biết chi phí và lợi ích của sản phẩm và công nghệ có thể liên quan đến nội dung của dự án. Kiến trúc sư đề xuất việc sử dụng bất kỳ công nghệ và sản phẩm nào mà ông cho là sẽ mang lại hiệu quả và chi phí cho dự án.

Kiến trúc sư không chọn công nghệ; Dựa trên kỹ năng của mình, kiến trúc sư chỉ đưa ra các đề xuất.

Kiến trúc sư có thể gợi ý sử dụng, ví dụ, Microsoft SQL Server 2014 vì các chỉ mục lưu trữ cột mới hoặc có thể lựa chọn ứng dụng web Single-Page được hỗ trợ bởi một kết thúc ASP.NET Web API. Tương tự, kiến trúc sư có thể nêu ra điểm sử dụng một kho tài liệu NoSQL địa phương thay vì một số lưu trữ bảng dựa trên đám mây.

Ai sẽ đưa ra quyết định nội bộ về công nghệ và sản phẩm nào sẽ được sử dụng?

Thông thường, nó là người quản lý dự án hoặc bất cứ ai quản lý ngân sách. Kiến nghị của kiến trúc sư có thể được chấp nhận hoặc từ chối. Nếu một gợi ý bị từ chối, việc sử dụng hoặc không sử dụng một sản phẩm hoặc công nghệ nhất định chỉ trở thành một yêu cầu phi chức năng mới đạt được và điều này có thể ảnh hưởng đến, thậm chí là cả về kiến trúc.

**Xây dựng chi tiết kỹ thuật**

Kiến trúc sư chịu trách nhiệm cuối cùng cho sự phát triển của hệ thống và điều phối công việc của một nhóm các nhà phát triển. Kỹ thuật cụ thể là phương tiện mà kiến trúc sư truyền đạt quyết định kiến trúc cho các nhà phát triển.

Thông số kỹ thuật có thể được thực hiện dưới nhiều hình thức khác nhau: bản phác thảo UML, tài liệu Microsoft Word, sơ đồ Microsoft Visio, hoặc thậm chí là các nguyên mẫu làm việc.

Truyền thông là chìa khóa cho một kiến trúc sư. Truyền thông xảy ra giữa kiến trúc sư và nhà phát triển, và điều này cũng xảy ra giữa các kiến trúc sư và các nhà quản lý dự án và các nhà phân tích, nếu không phải là người dùng. Một thuộc tính tuyệt vời của một kiến trúc sư là rõ nét về ngôn ngữ.

Sự tương tác giữa các kiến trúc sư và nhà phát triển sẽ thay đổi tùy thuộc vào phương pháp được chọn. Và sự tham gia của các nhà quản lý, các nhà phân tích và người sử dụng cũng phụ thuộc vào mức độ nhanh nhẹn mà bạn chấp nhận.

**Vai trò của kiến trúc sư**

Kiến trúc bao hàm một vai trò được gọi chung là "kiến trúc sư". Theo ISO / IEC, không có nhiều loại kiến trúc sư. Kiến trúc sư là kiến trúc sư. Giai đoạn.

Tuy nhiên, nếu bạn nhìn xung quanh (và kiểm tra sơ yếu lý lịch), bạn sẽ thấy khá nhiều kiến trúc sư. Cuối cùng, đó là một từ quá mức, và nó thực sự có ý nghĩa rất khác nhau tùy thuộc vào bối cảnh, công ty, hoặc thậm chí cả đất nước.

**Bạn có biết bao nhiêu loại kiến trúc sư?**

Tại Hoa Kỳ, một kiến trúc sư doanh nghiệp (EA) hầu như không liên quan gì đến việc phát triển ứng dụng bởi vì người trong vai trò đó tập trung 90 phần trăm thời gian vào chiến lược, dàn xếp và cơ sở hạ tầng kinh doanh liên quan đến CNTT. Trong một tổng hợp cực đoan, EA là một hình thức kết hôn kinh doanh và CNTT. Một ứng cử viên cho vai trò này dự kiến sẽ biết rất ít về các đối tượng kiến trúc phần mềm và có lẽ không có gì về những thứ như kiến trúc lớp hoặc DDD.

Ở nhiều công ty, vai trò chịu trách nhiệm về các quyết định khó khăn và chịu trách nhiệm đề xuất các phương pháp tiếp cận và công nghệ thậm chí còn không được coi là "kiến trúc sư" nhưng lại được đặt tên cho nhà phát triển chính hoặc một thứ tương tự.

Vì vậy bạn có thể tìm các nhãn hiệu như kiến trúc sư doanh nghiệp, kiến trúc sư cơ sở hạ tầng (IA), kiến trúc sư chuyên về công nghệ (TSA) và thậm chí là kiến trúc sư giải pháp (SA). Tất cả những sự phân biệt này là sai lầm vì chúng cố gắng đột nhập vào những phần mà cuối cùng là một vai trò nguyên tử, nhưng phức tạp. Theo chúng tôi, nó tạo ra phân loại không cần thiết và đặt nền móng cho bối cảnh khó hiểu, người-không-gì.

Trong cuốn sách này, chúng tôi sử dụng định nghĩa của ISO / IEC về kiến trúc sư, đó là "người, nhóm hoặc tổ chức chịu trách nhiệm về kiến trúc của hệ thống". Khi lập bản đồ khái niệm này tới phạm vi công ty rộng nhất, Chúng tôi gọi Kiến trúc sư là một kiến trúc sư phần mềm (hoặc giải pháp) hoặc thậm chí là một nhà phát triển chính.

**Vai trò Kiến trúc sư**

Bạn có thể đã nhận thấy nó đã có, nhưng nếu bạn đi đến Microsoft TechEd, bạn thấy rằng bài Kiến trúc hầu như không có bài về các vấn đề thực sự liên quan đến phát triển phần mềm và kiến ​​trúc. Vì lý do này, tất cả các phiên họp DDD của Gazillion mà chúng tôi đã đệ trình trong nhiều năm qua đã thường xuyên bị từ chối.

Trong số các nhân viên sự kiện của Microsoft TechEd, kiến ​​trúc sư thường có ý nghĩa quan trọng đối với kiến ​​trúc doanh nghiệp. Và tất cả các phiên làm việc DDD tại Microsoft TechEd đều nằm trong mục Phát triển! Không có nhiều kiến ​​trúc sư trong cùng một nhóm dự án. Tương tự như vậy, nếu không muốn, kiến ​​trúc sư khác có những kỹ năng hơi khác nhau. Tuy nhiên, như chúng ta dự định trong cuốn sách này, và bất kể các tiêu đề về mặt pháp lý, kiến ​​trúc sư có tiếp xúc với mã. Họ thiết kế hệ thống nhưng sau đó làm việc chặt chẽ với các nhà phát triển để đảm bảo thực hiện đúng.

Như chúng ta đã thấy, kiến ​​trúc sư là một trong những nhà phát triển giỏi và có kinh nghiệm nhất. Chúng tôi không tin rằng có giá trị trong việc có kiến ​​trúc sư chỉ nói chuyện bằng UML và Visio và để lại bất kỳ chi tiết triển khai nào cho các nhà phát triển. Ít nhất, chúng tôi đã không bao giờ tìm thấy nó dễ dàng để làm việc với những người này khi chúng tôi đã vượt qua đường dẫn với họ.

**Những quan niệm sai lầm phổ biến về kiến trúc sư**

Hầu hết là do ý nghĩa khác nhau của thuật ngữ kiến trúc, một tập hợp các quan niệm sai lầm đã phát triển vượt ra khỏi khối lượng của sự khinh thị và cách giải thích cá nhân. Hãy đi qua một vài trong số họ và, chúng tôi hy vọng, làm rõ một số ít.

**Kiến trúc sư là một nhà phân tích**

Đây là một tuyên bố sai. Kiến trúc sư chỉ đơn giản là không phải là nhà phân tích.

Đôi khi, một kiến trúc sư có thể giúp các nhà phân tích trong quá trình gợi ý để làm rõ các yêu cầu tối nghĩa hoặc giải quyết các yêu cầu lạ lẫm và ưa thích thêm vào danh sách chỉ để "hoàn thiện". Đôi khi, kiến trúc sư có thể tham gia vào các cuộc họp với các bên liên quan. Nhưng đó là nó.

Nói chung, một nhà phân tích là một người là một chuyên gia về miền. Kiến trúc sư không hẳn là chuyên gia. Một nhà phân tích chia sẻ với kiến trúc sư về những phát hiện của ông về cách hệ thống hoạt động và hệ thống nên làm gì.

Quan niệm sai phổ biến này có thể bắt nguồn từ ý nghĩa không chính xác của từ nhà phân tích . Nếu từ chỉ đơn giản cho thấy ai đó thực hiện một số phân tích trên một hệ thống, thật khó để phủ nhận sự tương đồng giữa các kiến trúc sư và các nhà phân tích. Cách đây 30 năm, thuật ngữ hệ thống phân tích đã được sử dụng để chỉ một chuyên gia có khả năng đưa ra các cân nhắc thiết kế cho một hệ thống. Tuy nhiên, vào thời điểm đó, phần mềm không có liên quan như hiện nay; Nó chỉ là một phần nhỏ (của) một hệ thống cơ bản dựa trên phần cứng.

Ngày nay, vai trò của một nhà phân tích và kiến trúc sư thường được công nhận là khác nhau. Và hầu như không bao giờ một kiến trúc sư đóng vai trò của một nhà phân tích.

**Chú ý** Với vai trò không phải lúc nào cũng tách biệt, đặc biệt là trong các công ty nhỏ, điều đó có thể xảy ra với cùng một người như một nhà phân tích và kiến trúc sư. Điều này đơn giản chỉ có nghĩa là có một người trong công ty biết rõ về công việc kinh doanh và xử lý đủ để đưa ra các yêu cầu về chức năng và dịch chúng thành những đặc điểm riêng cho các nhà phát triển. Vai trò và trách nhiệm vẫn còn khác biệt, nhưng kỹ năng riêng biệt cho mỗi người được tìm thấy trong cùng một cá nhân.

**Kiến trúc sư là người quản lý dự án**

Đây có phải là một tuyên bố sai? Nó phụ thuộc.

Kiến trúc sư chịu trách nhiệm về kiến trúc của hệ thống, và người đó phối hợp và hướng dẫn sự phát triển của hệ thống. Người quản lý dự án đại diện cho các bên liên quan và quản lý dự án bằng cách lựa chọn, trước hết, một phương pháp. Người quản lý dự án sau đó chịu trách nhiệm đảm bảo rằng dự án tuân thủ kiến trúc trong khi tiến hành trong giới hạn của thời gian và ngân sách.

Nếu chúng ta nhìn vào vai trò của kiến trúc sư và vai trò của người quản lý dự án, chúng ta thấy rằng chúng khác biệt. Giai đoạn.

Tuy nhiên, không có gì bất thường khi một diễn viên đóng vai hai vai diễn. Giống như trong rạp hát, điều này hầu như không xảy ra trong các công ty lớn, nhưng nó xảy ra khá thường xuyên ở các công ty nhỏ.

Tóm lại, nếu bạn muốn trở thành một kiến trúc sư phần mềm khi lớn lên, bạn không nhất thiết phải phát triển kỹ năng quản lý dự án. Nếu bạn có kỹ năng cho cả hai vai trò, tuy nhiên, bạn có thể thử để có được trả gấp đôi.

**Kiến trúc sư không bao giờ viết bất kỳ mã**

Đây là một cuộc tranh luận vô tận: Nếu kiến trúc sư viết mã? Về cơ bản có hai trường phái tư tưởng.

Một trường nghĩ rằng các kiến trúc sư sống trên tầng lửng trên, có thể ở một tầng áp mái. Các kiến trúc sư sau đó bước xuống sàn nhà phát triển chỉ trong một khoảng thời gian để minh họa, sử dụng sơ đồ, những gì họ nghĩ về hệ thống. Sau đó, họ lên thang máy, thu thập đồ vật của họ, và đi chơi golf. Khi trên khóa học, họ tắt điện thoại di động của họ và tập trung vào trò chơi. Khi hoàn thành, nếu họ lỡ một hay hai cuộc gọi, họ gọi lại và giải thích cho các nhà phát triển giả biết những gì đã được rõ ràng trong sơ đồ mà không ai trong nhóm nhà phát triển có thể hiểu được. Theo trường phái tư duy này, các kiến trúc sư chưa bao giờ bao giờ bẩn tay với câu lệnh C # đơn giản nhất. C #? Oh không, ngôn ngữ mới nhất mà họ đã tiếp xúc có thể là Pascal trong khi ở trường đại học và Visual Basic ở nhà.

Một trường phái tư tưởng khác cũng nghĩ rằng mỗi kiến trúc sư đều là một nhà phát triển. Để đi theo phép ẩn dụ này, chúng ta có thể nói rằng Kiến trúc sư lớp kế thừa từ Nhà phát triển lớp học và thêm một số phương pháp mới (kỹ năng) trong khi ghi đè (chuyên) một số khác. Trở thành kiến trúc sư là sự tiến hóa tự nhiên trong sự nghiệp của một số nhà phát triển. Sự khác biệt cơ bản giữa kiến trúc sư và nhà phát triển là kinh nghiệm và giáo dục. Bạn có kinh nghiệm theo thời gian trong công việc; Bạn nhận được giáo dục từ nghiên cứu sách tốt và lấy các lớp học phải. Ngoài ra, một kiến trúc sư có khả năng tập trung tầm nhìn của mình về hệ thống từ một trình độ cao hơn một nhà phát triển trung bình. Hơn nữa, kiến trúc sư có kỹ năng xử lý khách hàng tốt.

Kiến trúc sư có thể không viết nhiều mã sản xuất. Nhưng cô ấy viết rất nhiều mã; Cô ấy luyện tập với mã hàng ngày; Cô biết về ngôn ngữ lập trình, kỹ thuật mã hóa, thư viện, sản phẩm, công cụ, CTP, và cô sử dụng phiên bản mới nhất của Visual Studio. Trong một số lĩnh vực lập trình, một kiến trúc sư biết nhiều hơn nhiều nhà phát triển. Kiến trúc sư có thể viết các công cụ và tiện ích để giúp các nhà phát triển sản xuất hiệu quả hơn. Và, thường xuyên hơn bạn có thể nghĩ ở irst, kiến trúc sư chỉ là một thành viên của nhóm phát triển. Ví dụ, một mã kiến trúc sư sản xuất văn bản là một sự xuất hiện hoàn toàn bình thường trong một ngữ cảnh Agile. Nó cũng là một sự xuất hiện bình thường trong các công ty nhỏ bất kể phương pháp luận. Đồng thời, việc có một kiến trúc sư viết mã sản xuất có thể hoàn toàn kỳ quặc khi thấy trong một số kịch bản của công ty lớn, đặc biệt nếu sử dụng phương pháp luận truyền thống và phi Agile.  
Còn hai chúng tôi thì sao? Chúng tôi thuộc trường nào? Vâng, Andrea là kiến trúc sư hơn Dino vì ông làm việc trên tầng năm. Dino, mặt khác, là gần gũi hơn với sự phát triển bởi vì ông đã có một vài cuốn sách ASP.NET kỹ thuật cao trong hồ sơ của mình, và quan trọng hơn, các công trình trên tầng hai. Chúng tôi không chơi golf. Dino chơi quần vợt thường xuyên, trong khi Andrea thích squash tốt hơn. Chúng tôi chỉ bị từ chối tiếp cận trường học đầu tiên của tư tưởng.

**Lưu ý** Trong không có lĩnh vực kỹ thuật khác là sự phân biệt giữa những người-thiết kế và những người-xây dựng là không được chấp nhận như nó là trong phần mềm. Sự phân biệt tồn tại chủ yếu thông qua việc lập luận chứ không phải là từ việc công nhận các kỹ năng công.

Sự so sánh kinh điển với kiến ​​trúc dân dụng. Người chơi gạch có kỹ năng độc đáo mà các kỹ sư thiếu. Tuy nhiên, không có thợ xây nào ước mơ thiết kế hoặc tính toán đơn giản chỉ vì họ thiếu kỹ năng tự quyết định. Họ làm công việc của mình tốt nhất có thể, tận dụng lợi thế của công việc xây dựng ủy thác cho họ.

Trong phần mềm, tình hình là khác nhau vì các kiến ​​trúc sư và nhà phát triển có nguồn gốc chung. Người phát triển có tay nghề cao hơn, anh càng cảm thấy khuyến khích để thảo luận về các lựa chọn thiết kế - và thường có lý do. Càng nhiều kiến ​​trúc sư mất liên lạc với chương trình hàng ngày, càng mất nhiều sự tôn trọng của các nhà phát triển khác. Điều này tạo ra một loại vòng tròn luẩn quẩn, mà ma thuật trở nên tốt hơn khi bạn chuyển sang một phương pháp Agile.

**Tóm lược**

Kiến trúc là một điều cần thiết và không phải là một sang trọng cho phần mềm hiện đại. Giả sử bao giờ có một thời gian trong đó kiến trúc là không bắt buộc, ngày nay không còn nữa. Và sự khác biệt là tất cả trong phức tạp yêu cầu của phần mềm hiện đại.

Đó là nhận thức chung rằng phần mềm có thể được so sánh với kỹ thuật dân dụng; Tuy nhiên, khi các kỹ sư xây dựng thiết lập để xây dựng, nói, một cây cầu, một cây cầu được xây dựng. Ngoài ra, cây cầu hầu như luôn hoạt động chính xác và được xây dựng khá gần với ngân sách ban đầu. Bạn khó có thể nói như nhau cho nhiều dự án phần mềm. Khi nói đến phần mềm, đôi khi rất không chắc chắn về những gì các bên liên quan cuối cùng sẽ có được từ cam kết của họ. Thay vào đó, chắc chắn rằng ngân sách ban đầu có thể bị vượt quá và bất kỳ sản phẩm nào cũng có thể khác nhau ở một mức độ nào đó, từ sự mong đợi.

Tại sao nó như vậy với phần mềm?

Nói chung, chúng tôi tin rằng phát triển phần mềm khó có thể phải tuân theo các quy tắc cố định và cứng nhắc như với các nguyên tắc khác, như kỹ thuật xây dựng. Phát triển phần mềm không phải là kỹ thuật thuần túy bởi vì nó bị ô nhiễm bởi các thiết kế dồi dào, sự sáng tạo đồng nhất và thậm chí tâm lý học. Ngoài ra, phần mềm có một tính năng rất năng động vì nó là tương đối chậm xây dựng nhưng cần phải ở lại đồng bộ với các yêu cầu kinh doanh liên tục thay đổi. Các hoạt động phần mềm thay đổi rất nhanh chóng đến nỗi, bất cứ lúc nào, thật khó nắm bắt và lưu trữ trạng thái hiện tại của nghệ thuật.

Trong chương này, chúng tôi tập trung vào kiến trúc phần mềm và vai trò kiến trúc sư trong một nỗ lực để nắm bắt được bản chất của cả hai. Trong chương tiếp theo, chúng tôi tập trung vào những gì các kiến trúc sư thực sự làm khi họ cố gắng áp dụng kiến trúc vào miền vấn đề. Trong chương tiếp theo, chúng ta thảo luận nhiều hơn về cơ chế của các dự án phần mềm và điều đáng sợ làm cho các dự án thất bại - quả cầu bùn lớn.

Hoàn thành với một nụ cười

Dưới đây là một sự xoắn hài hước về một số chủ đề được thảo luận trong chương này:

* Thêm nhân lực vào một dự án phần mềm cuối năm làm cho nó sau này. (Fred Brooks, "The Mythical Man-Month", Addison-Wesley, 1995)
* Không bao giờ được xây dựng theo kế hoạch hoặc trong ngân sách. (<Http://www.murphys-laws.com>)
* Thất bại không phải là một lựa chọn; Nó được bao gồm trong phần mềm. (Http://www.murphys-laws.com)