**Trên thực tế, vấn đề thuật toán không phát sinh khi bắt đầu một dự án lớn. Thay vào đó, chúng thường đột nhiên xuất hiện như những bài toán con làm các lập trình viên phải đau đầu, thậm chí còn có thể khiến chương trình không thể hoàn thiện.**

—Steven S. Skiena, *The Algorithm Design Manual*

### **Bối cảnh**

Bạn đang sống trong thế giới của các thời hạn chặt chẽ và các dự án phần mềm phức tạp sử dụng vô số công cụ khác nhau. Nhà tuyển dụng của bạn không có khả năng sử dụng đủ các chuyên gia để lấp đầy mọi vai trò. Bạn chỉ học vừa đủ về bất kỳ công cụ nào để hoàn thành công việc ngày hôm nay. Bạn chọn một vài hướng dẫn về ngôn ngữ hoặc thư viện mà bạn đang làm việc cùng hàng ngày. Bạn đưa ra quyết định mà không dành thời gian để hiểu vấn đề, và sao chép những ví dụ lặt vặt được cung cấp cùng với các công cụ. Điều này có hiệu quả trong chừng mực mà bạn vẫn còn có thể nắm bắt bất cứ công việc nào. Bạn có khả năng tìm hiểu sâu vào một công nghệ mới và đưa ra một giải pháp rất nhanh chóng. Bạn chỉ cần học vài phần của một công nghệ mà bạn cần để có thể nắm bắt chút ít cách hệ thống làm việc, và bạn giao phó vào các thành viên khác trong nhóm để tìm hiểu các phần còn lại. Ví dụ, bạn có thể đang là một nhà phát triển Java phía máy chủ, và do đó bạn có ít hoặc không có kiến ​​thức về cách giao diện người dùng được xây dựng.

### **Vấn đề**

Bạn tiếp tục gặp khó khăn trong việc bảo trì mã đã viết vì có một sự thật rằng các hướng dẫn bạn làm theo không thể giải quyết được tất cả các vấn đề mới phát sinh. Bạn nhận ra rằng kiến thức hời hợt của mình về cả nghìn thứ công cụ vẫn chẳng giúp được bạn khỏi lúng túng khi gặp phải một lỗi khó thấy hay cần phải làm gì đó đòi hỏi những kiến thức chuyên sâu. Mọi người thường nói rằng sơ yếu lý lịch của bạn là gian dối vì bạn không phân biệt nổi giữa vài tuần mở rộng dịch vụ web hiện có và kiến thức chuyên sâu về các vấn đề cố hữu trong việc duy trì một hệ thống doanh nghiệp có khả năng tương tác và mở rộng cao. Thậm chí nó tồi tệ đến mức vì kiến thức quá hời hợt đó mà bạn còn không biết mình non nớt đến thế nào cho đến khi một điều gì hay ai đó đưa bạn đến với thử thách.

### **Giải pháp**

Tìm hiểu để đào sâu vào các công cụ, công nghệ và kỹ thuật. Có được chiều sâu kiến ​​thức đến mức bạn biết tại sao mọi thứ lại hoạt động theo cách như vậy. Độ sâu là sự hiểu biết bản chất đến mức thiết kế tổng quan hơn là chỉ tập trung vào chi tiết của thiết kế. Ví dụ, điều này nghĩa là khi hiểu được lý thuyết kiểu (hoặc ít nhất là sự đơn giản hoá của góc phần tư định kiểu trên (<http://c2.com/cgi/wiki>? TypingQuadrant) chứ không chỉ đơn giản là học vẹt những điều bạn đã nghe những người khác nói.

Một trong những đồng nghiệp cũ của chúng tôi, Ravi Mohan, nói riêng với tôi rằng:

Kiến ​​thức về các xử lý đồng bộ (và giới hạn của chúng) sẽ hữu ích hơn là "phân lớp Thread hoặc thực hiện Runnable."

Các lĩnh vực mà bạn có kiến ​​thức vững vàng sẽ nuôi dưỡng sự tự tin và dẫn đường đưa lối khi bạn quyết định áp dụng mẫu Quét Nhà, bởi vì chúng chỉ ra những nơi bạn có thể chứng tỏ giá trị sớm trong khoảng thời gian làm việc với một nhóm mới. Quan trọng hơn, kiến ​​thức sâu rộng này sẽ là một cái gì đó để bạn có thể coi như một điểm tựa vững vàng và cung cấp cho bạn sức mạnh để chinh phục các lĩnh vực khác. Bạn luôn có thể tự nói với bản thân rằng: "Nếu mình làm chủ EJBs thì sau đó có thể tự tin xử lý các siêu lớp."

Một ưu điểm khác của việc đào sâu vào công nghệ là bạn thực sự có thể giải thích điều gì đang diễn ra bên dưới bề mặt của các hệ thống bạn đang làm việc. Trong các cuộc phỏng vấn, sự hiểu biết này sẽ phân biệt bạn với những ứng viên không thể mô tả phần mềm họ đã tham gia xây dựng một cách có ý nghĩa bởi vì tất cả những gì họ hiểu chỉ là một phần nhỏ. Một khi bạn đã là một thành viên trong nhóm, việc áp dụng mô hình này sẽ giúp tách biệt những người đang tạo ra đống đổ nát ngẫu nhiên (cuốn *Pragmatic Programmers* gọi đó là "lập trình may rủi", trong khi Steve McConnell gọi đó là "kỹ thuật phần mềm mê tín") với những người đang xây dựng nhà thờ.

Vậy làm thế nào để chúng ta phát hiện ra những người xây dựng nhà thờ? Họ là những người trong nhóm của bạn, người cuối cùng gỡ lỗi, biên dịch, lật ngược và đọc tài liệu đặc tả, RFC hoặc đưa ra tiêu chuẩn cho các công nghệ bạn sử dụng. Những người làm việc này đã thay đổi quan điểm và xây dựng sự hiểu biết thực tiễn về các công cụ hỗ trợ họ.

Sự thay đổi trong quan điểm bao gồm việc muốn theo dõi một vấn đề sâu xuống các tầng của hệ thống và sẵn sàng dành thời gian để có được kiến ​​thức bao quát mọi vấn đề. Ví dụ, chuyển đổi từ một chiếc máy tính đơn nhân sang đa nhân có thể làm thay đổi hành vi các bài kiểm thử song trong Java. Một số người chỉ nhún vai và chấp nhận rằng các bài kiểm thử của họ sẽ hoạt động một cách khó hiểu. Nhưng có những người lại theo dõi các vấn đề đến tận cấp độ CPU thông qua các thư viện thực thi song song, Mô hình Bộ nhớ Java, và các đặc điểm kỹ thuật của phần cứng vật lý.

Các công cụ bạn cần làm quen bao gồm trình gỡ lỗi (như GDB, PDB và RDB), cho phép bạn xem chương trình đang chạy; các trình gỡ lỗi mức băng thông (như Wireshark), cho phép bạn xem lưu lượng mạng; và sẵn sàng để đọc các tài liệu đặc tả. Đọc được đặc tả kỹ thuật cũng như đọc mã nghĩa là không gì có thể che được mắt bạn. Nó cung cấp cho bạn khả năng đặt những câu hỏi khó về các thư viện bạn sử dụng, và nếu bạn không thích những câu trả lời nhận được, bạn có khả năng thực hiện lại chúng hoặc chuyển sang cách giải quyết theo tiêu chuẩn khác.

Một trong những cách để sử dụng mẫu này là thu thập thông tin từ các nguồn chính thống. Điều này có nghĩa là lần tới khi có người nói chuyện với bạn về *Representation State Transfer*, hay còn gọi tắt là REST, bạn nên xem đó như là một cái cớ để đọc khái niệm này trong luận án tiến sĩ của Roy Fielding. Hãy xem xét viết bài đăng lên blog để làm rõ hoặc chia sẻ những gì bạn đã học và khuyến khích người khác đọc tài liệu gốc.

Đừng vội tin lời của ai đó đang trích dẫn cuốn sách được nhắc đến trong một bài báo có đề cập tới trang Wikipedia có liên kết với tài liệu IETF Request for Comment chính thống. Để thực sự hiểu bất cứ ý tưởng nào, bạn cần phải dựng lại bối cảnh mà nó đã được thể hiện lần đầu tiên. Điều này cho phép bạn xác minh được bản chất của ý tưởng đang lan truyền qua tất cả những người trung gian đó.

Tìm ra những người đầu tiên đưa ra ý tưởng và hiểu những vấn đề họ đã cố gắng để giải quyết. Hoàn cảnh này thường bị thiếu sót trong các bản dịch cũng như khi ý tưởng được truyền đi. Đôi khi bạn thấy rằng những ý tưởng mới dường như đã bị bỏ rơi từ lâu trước đó, thường vì những lý do chính đáng - nhưng mọi người đã quên nó vì bối cảnh ban đầu đã bị mất. Lại có những lần bạn sẽ thấy rằng nguồn gốc của một ý tưởng là một hướng dẫn tốt hơn nhiều so với chuỗi những người trích dẫn lẫn nhau trong nhiều năm. Bất kể điều gì xảy ra, việc theo dõi nguồn gốc các ý tưởng bạn thấy hữu ích là một bài tập quan trọng, một thói quen tốt sẽ giúp ích cho bạn trong thời gian dài khi cố gắng học hỏi những kiến thức mới.

Khi đọc một hướng dẫn, bạn không nên tìm kiếm mã để sao chép mà nên đặt hiểu biết mới vào một cấu trúc tư duy. Mục tiêu của bạn phải là hiểu được bản chất của khái niệm cho dù nó là một ví dụ đặc biệt của cái gì đó khác. Hãy tự hỏi liệu có các khái niệm khoa học máy tính cơ bản nào đằng sau những gì bạn đang học, và những gì đã được cân nhắc để đi tới triển khai như khi bạn đang sử dụng hiện. Được trang bị với kiến thức sâu sắc hơn này, bạn sẽ có thể vượt qua hướng dẫn ban đầu khi gặp rắc rối.

Ví dụ, người ta thường gặp rắc rối với biểu thức chính quy bởi vì họ chỉ có được một sự hiểu biết hời hợt. Bạn có thể cảm thấy ổn trong nhiều năm, thậm chí hàng thập kỷ, mà không cần thực sự hiểu sự khác biệt giữa một Ôtômát hữu hạn đơn định và một Ôtômát hữu hạn không đơn định. Một ngày nào đó, wiki của bạn ngừng hoạt động. Hóa ra là nếu máy xử lý biểu thức chính quy của bạn sử dụng đệ quy, thì trên một số đầu vào yêu cầu quay lui nó sẽ chạy trong một thời gian rất lâu và cuối cùng ném ra một StackOverflowException. Ade đã khám phá ra điều này một cách khó khăn, nhưng may mắn là nó đã xảy ra trong khi triển khai trên wiki vọc chứ không phải trong môi trường sản phẩm thực tế.

Với sự tập trung vào sự hiểu biết về công nghệ và công cụ một cách sâu sắc, bạn cần phải cẩn thận để không trở thành một chuyên gia hẹp. Mục đích ở đây là thu thập nhiều kiến thức chuyên môn cần thiết để giải quyết bất kỳ vấn đề nào, nhưng không làm mất tầm nhìn tương đối của bạn về tầm quan trọng của nhiều khía cạnh khác nhau trong phát triển phần mềm.

Một điều bạn sẽ học được từ việc cố gắng áp dụng mẫu này là việc tiếp thu hiểu biết sâu sắc thật sự rất khó khăn. Đây là lý do tại sao hầu hết hiểu biết của mọi người về khoa học máy tính để hỗ trợ quá trình phát triển phần mềm chỉ là một dặm rộng và một inch dày. Dựa vào kiến thức của người khác về những điều cơ bản thường là dễ dàng và có lợi hơn so với việc dành thêm nỗ lực để chinh phục được kiến thức cho bản thân. Bạn có thể tự nhủ rằng bạn luôn có thể học nó khi nào bạn cần; tuy nhiên, khi ngày đó đến bạn sẽ cần dùng đến tất cả mọi thứ trong một cuối tuần, nhưng lại phải mất một tháng để học tất cả các điều kiện tiên quyết.

Một hậu quả khác của việc chỉ sở hữu kiến ​​thức bề nổi là bạn không bao giờ nhận ra rằng vấn đề bạn đang cố gắng giải quyết có một giải pháp nổi tiếng hoặc thực tế là điều không thể (trong trường hợp này có thể có rất nhiều bài báo khoa học về lý do tại sao không thể và làm thế nào để xác định và dàn giải lại vấn đề đó). Nếu chỉ lướt qua bề mặt, thì bạn sẽ không thể khám phá ra những điều chưa biết, và nếu không hiểu bản chất những điều bạn biết, bạn không thể khám phá những điều mới mẻ.Thông thường, quá trình tra cứu qua tất cả các lớp của một vấn đề sẽ tiết lộ một khái niệm cơ bản từ khoa học máy tính. Trong khi công việc của các nhà khoa học máy tính dường như không thực tế, những ai có thể áp dụng các lý thuyết tiên tiến nhất cho các vấn đề ngoài đời thực sẽ làm được những việc trong như ma thuật với người khác. Thay đổi đơn giản trong sự lựa chọn thuật toán hoặc cấu trúc dữ liệu có thể khiến thời gian xử lý một khối từ chỗ mất hàng tháng trở nên ít hơn thời gian người dùng rời tay khỏi cú nhấp chuột. Một người chỉ biết về Lists, Sets và HashMaps dường như không nhận ra rằng anh ta cần tới Trie để giải quyết vấn đề của mình. Thay vào đó, anh ta sẽ có chỉ giả định rằng vấn đề như là so trùng tiền tố dài nhất là không thể thực hiện được, và sẽ từ bỏ hoặc hỏi xem tính năng này có thể được loại bớt không.

Nếu áp dụng mẫu này thường xuyên, bạn sẽ trở thành một trong những người thực sự thấu hiểu công cụ của mọi người hoạt động như thế nào. Bạn sẽ không còn chỉ phụ thuộc vào những dòng mã và phép thuật của người khác để làm việc khó nữa. Cần lưu ý rằng sự hiểu biết này sẽ tách biệt bạn khỏi phần lớn các lập trình viên mà bạn đang làm việc cùng, và sẽ giúp bạn có cơ sở lựa chọn hợp lý cho những công việc xứng tầm hơn. Do đó, bạn sẽ có nhiều khả năng thất bại đau đớn hoặc thành công một cách ngoạn mục. Ngoài ra, không được cho phép sự hiểu biết làm bạn kiêu ngạo. Thay vào đó, hãy tiếp tục tìm ra cơ hội để Là Người Kém Nhất. Hãy thử thách bản thân để thu thập các công cụ hữu ích từ những khối cơ bản này, thay vì chỉ dựa vào khả năng sẵn của bạn để tách rời mọi thứ.

### **Hành động**

Tìm và đọc các đặc tả RFC 2616, mô tả HTTP1.1, và RFC 707, mô tả kỹ thuật trong Công nghệ Triệu gọi Thủ tục Từ xa vào tháng 1 năm 1976. Trang bị kiến thức sâu sắc hơn về HTTP, hãy thử triển khai một máy khách và một máy chủ cho RFC 707. Khi bạn cảm thấy đã hiểu rõ về sự đánh đổi của các nhà biên tập RFC 707, hãy ứng dụng với một mã nguồn mở hiện đại để thực hiện cùng ý tưởng đó, như khung Apache Thrift được phát triển bởi Facebook chẳng hạn. Sau đó, từ góc nhìn quan điểm của bạn, hãy viết bài đăng trên blog mô tả sự phát triển tri thức về các phương thức gọi thủ tục của chúng ta và các hệ thống phân tán trong suốt ba thập kỷ qua.

Bây giờ, hãy tìm đọc các bài viết của Steve Vinoski về RPC. Liệu bạn có thấy nghi ngờ về chiều sâu hiểu biết của mình? Viết một bài đăng lên blog về những nghi ngờ và mức độ hiểu biết hiện tại của bạn.