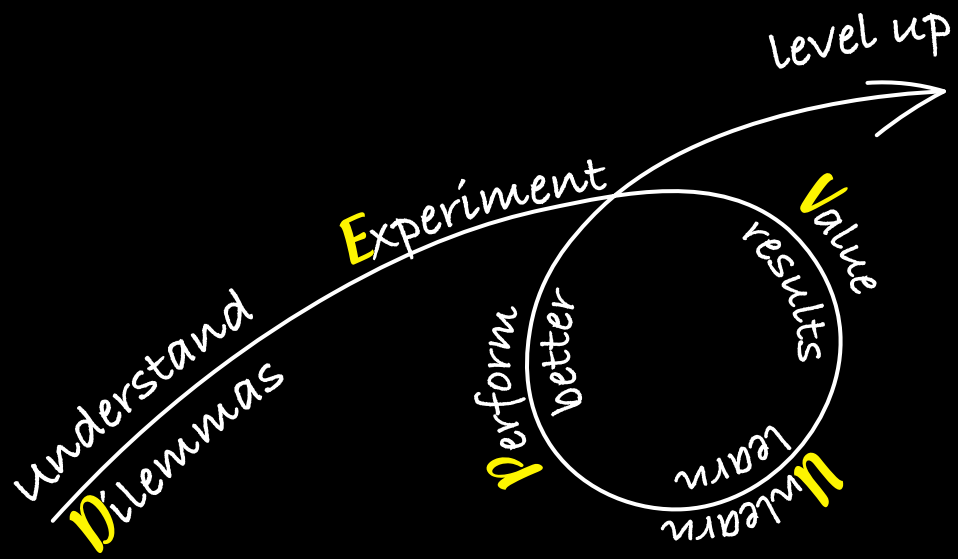
NGUYỄN HIỂN

DevUP

Cuốn sách toàn diện phát triển sự nghiệp của Lập trình viên



spiderum

NHÀ XUẤT BẢN THẾ GIỚI

Nguyễn Hiển

DevUP

NHÀ XUẤT BẢN THẾ GIỚI

Tôi rất hào hứng với những cuốn sách của người Việt viết cho người Việt vì tính thiết thực của những cuốn sách này. Đúng như mong đợi, DevUP là một cuốn sách hay, dành cho những người quan tâm đến việc thúc đẩy bản thân, cũng như các bạn đang chập chững những ngày đầu trên con đường trở thành các lập trình viên xuất sắc. Tôi đặc biệt thích phần Dilemmas (Hiểu những thế lưỡng nan) trong cuốn sách, nó phản ánh gần gũi các góc cạnh của các bạn lập trình viên thường xuyên phải tự vấn. Tôi hi vọng, cũng như tôi, các anh chị làm quản lý khi đọc cuốn sách này sẽ thấu hiểu và cùng cố gắng tạo dựng những môi trường lưỡng nan "cân bằng", qua đó nhận về các giá trị phát triển cho lực lượng then chốt trong thời đại CNTT là sức mạnh này.

Trần Trung Kiên, PMI Certified Project Manager.

* * *

Thật lòng mà nói, đây không phải là một cuốn sách dễ đọc. Nó không phải để đọc một lần rồi thôi - nó đòi hỏi bạn phải tư duy cùng nó, ăn ngủ cùng nó, và chiêm nghiệm cùng nó. Nó được đúc rút từ kinh nghiệm của một người đã nhiều năm lăn lộn trong ngành công nghiệp phần mềm, trực tiếp hiện thực hóa các ý tưởng thành những sản phẩm thương mại, nhiều năm tham gia đào tạo lập trình viên và quản lý các đội ngũ lập trình viên. Nó đến từ những đêm dài debug, những email chửi mắng, những yêu cầu có một không hai của khách hàng, những giải pháp phải đập đi làm lại, những lần tranh thủ ngồi code trong lúc chờ máy bay ở một phương trời xa lạ. Nó đến từ những lần server quá tải, những dòng lệnh điên rồ, những ly cafe thay cơm, và cả những lần trễ hẹn với người thương. Tôi không hy vọng cuốn sách này sẽ mang lại cho bạn câu trả lời trực tiếp, nhưng hy vọng rằng nó sẽ đi theo bạn trong cả quá trình phát triển sự nghiệp lập trình viên.

Khánh DAO, Head of Software development @ Pixelz.

* * *

Lập trình viên đang và sẽ trở thành một nghề phổ biến, không lạ lẫm với xã hội trong tương lai gần. Như những nghề khác, những lập trình viên cần có những cái nhìn đầy đủ và lựa chọn con đường đi phù hợp không chỉ trong 3 hay 5 năm đầu sự nghiệp mà thậm chí cả cuộc đời làm nghề.

Đọc cuốn sách, tôi có cảm giác như những lời của một bậc đàn anh dành cho những thế hệ kế tiếp, ân cần và tận tâm. Một cuốn sách mà mỗi lập trình viên nên đọc ít nhất một lần.

Lưu Hiếu, CEO @ Agilearn & CTO @ Agilead Global.

* * *

Cuốn sách như một cuốn sổ tay bỏ túi với đầy đủ các chỉ dẫn để một lập trình có thể phát triển và thành công. Nó như chứa đựng mọi trăn trở của một lập trình viên trong cuộc sống. Từ những băn khoăn về sự nghiệp, tổ chức, tới các định hướng phát triển chuyên môn, cách thức làm việc, tạo dựng quan hệ, tìm kiếm cơ hội,... Bằng cách sắp xếp thành 5 phần: trăn trở (tiến thoái lưỡng nan - dilemmas), thử nghiệm, đánh giá, học tập, thực thi, lập trình viên dễ dàng tìm thấy cho mình những kiến thức hữu ích để phát triển và hoàn thiện bản thân. Cuốn sách được viết giản dị như lời chia sẻ của một người quản lý lâu năm. Nó xứng đáng để các lập trình viên bỏ túi để bất khi nào cần thì có thể mở ra tham khảo.

Lê Minh Nghĩa, Director of Marketplace Engineering @ TIKI.

* * *

DevUP là một cuốn sách rất hay mang lại cho người đọc cái nhìn đa chiều về các vấn đề thường nhật trong công việc phát triển phần mềm. Tôi với 17 năm kinh nghiệm, trải qua rất nhiều vị trí cũng như đã từng đối mặt với những vấn đề được nêu ra trong cuốn sách, cảm nhận được sự tâm huyết và chia sẻ từ tác giả mong muốn làm cho các vấn đề được rõ ràng và phù hợp trong thế giới thực.

Linh Nguyen, Software Engineer.

* * *

Tôi vẫn thường nhận được những câu hỏi của các bạn trẻ ở giai đoạn bắt đầu sự nghiệp xin lời khuyên về định hướng phát triển nghề nghiệp. Hơn 10 năm trước, tôi cũng có những câu hỏi như các bạn bây giờ nhưng tôi không có những câu trả lời rõ ràng. Phần lớn những câu hỏi đó qua thời gian, qua trải nghiệm đã được tác giả đúc kết trong cuốn sách này. Tôi tin rằng, những câu chuyện nghề, những bài học và những

phân tích mạch lạc sẽ là giúp các bạn trẻ có một cái nhìn và lựa chọn đúng đắn cho con đường sự nghiệp của mình.

Lê Đôn Khuê, Software Engineer @ BeGroup.

* * *

Cuốn sách là những tranh luận và băn khoăn của nhiều thế hệ lập trình viên về lựa chọn nghề nghiệp, công việc và đôi khi là những ý kiến khá gay gắt của cả một số người có uy tín trong cộng đồng. Đó là sự lựa chọn khó khăn chọn làm ở công ty sản phẩm hay gia công, công ty khởi nghiệp hay tập đoàn lớn, chuyển việc nhiều hay trung thành ở một công ty và làm sao để phát triển nghề nghiệp của một lập trình viên.

Anh Hiển là số ít người được trải nghiệm cũng như được quan sát những tình huống như vậy ở nhiều vị trí khác nhau từ lập trình viên, lãnh đạo, nhà tư vấn và huấn luyện viên. Những trải nghiệm đó cộng với nỗi ưu tư cho cộng đồng lập trình viên đã cho chúng ta một cái nhìn rộng, rõ ràng vả cả lộ trình mà hầu hết lập trình viên có thể dùng nó như một la bàn cho sự chọn lựa chọn trong sự nghiệp.

Dù cuốn sách về chủ đề quá đỗi nghiêm túc nhưng giọng văn không đao to búa lớn mà nhẹ nhàng, dễ tiếp nhận.

Phạm Anh Đới, CEO @ Agilead Global.

* * *

Một tập hợp các phân tích và đánh giá thú vị về nghề lập trình của tác giả. Đặc biệt là các câu hỏi khó khăn mà rất nhiều lập trình viên đều hỏi đi hỏi lại nhưng rất nhiều người vẫn chưa thể có các câu trả lời cụ thể như: Tôi nên đi làm công ty lớn hay cho startup? Nên làm sản phẩm hay làm outsource? Nên đi sâu hay đi rộng? Tôi là senior hay junior?

Với các bạn trẻ vừa ra trường nói chung tôi hay giới thiệu quyển: *Nếu tôi biết được khi còn 20*. Giờ đây, đối với các bạn trẻ đã xác định theo hướng lập trình thì tôi sẽ giới thiệu đến quyển sách này.

Son Tran, Software Engineer.

* * *

Tôi ước mình đọc được cuốn sách này từ hơn 10 năm trước, nếu bạn là LTV hoặc muốn làm LTV, cuốn sách này dành cho bạn.

Quyển sách là một tập hợp các chủ đề với nhiều góc nhìn, chia sẻ thú vị của tác giả. Qua từng chương, tôi tìm thấy cho mình câu trả lời cho những vấn đề mà tôi đã từng băn khoăn, hoài nghi trong suốt quá trình làm nghề.

Có một câu nói của tác giả mà tôi rất thích: *"Để đi xa hơn, chúng ta phải chuẩn bị cho con đường dài hơn. Không bước chuẩn bị nào tốt hơn việc quay lại với những nguyên lý cơ bản - những thứ đúng phần lớn theo đơn vị thập kỷ thay vì quá cố gắng bám lấy những thứ khiến chúng ta chóng mặt với tốc độ thử thách theo đơn vị năm hoặc tháng."*

Hi vọng bạn cũng sẽ tìm được nhiều niềm vui, thông tin hữu ích khi thưởng thức cuốn sách này.

Charlie Brown, Former Assistant Manager/
Project Lead @ Samsung R&D Vietnam.

* * *

Bình dị, giản đơn, nhưng chân thành và sâu sắc là cảm xúc mà chúng ta sẽ có được sau khi đọc cuốn sách này. Cái hay của DevUP là nó sử dụng lối dẫn chuyện đơn giản và cấu trúc rất bình dị để giải quyết rất nhiều vấn đề phức tạp của con đường sự nghiệp của lập trình viên, và vì thế nó phù hợp với một lớp khá lớn người đọc, từ những người mới vào nghề cho đến những người thạo nghề, thậm chí là các quản lý.

Những lập trình viên mới vào nghề có thể tìm thấy trong sách nhiều chỉ hướng bổ ích để chọn con đường đi đúng đắn. Những lập trình viên thạo nghề có thể tìm thấy trong sách một triết lý sâu sắc: tiến bộ mỗi ngày. Các quản lý có thể tìm được nhiều lời khuyên và dữ liệu bổ ích để xây dựng và phát triển sự nghiệp cho các lập trình viên trong doanh nghiệp của mình.

Tôi ước gì tác giả có thể dành thêm thời gian và nội dung để giải quyết thật rốt ráo các vấn đề, tuy nhiên đó có thể là khoảng mở để độc giả có thể cùng khám phá tiếp trên con đường tiến bộ mỗi ngày của mình.

Hoàng Phan Bảo Trung @DEHA Software JSC.

* * *

Hiển có một thế giới quan rất đặc biệt. Vừa là một *dev*, vừa là một *nhà sư phạm*, một *mentor*, một *scrummaster*, một *đội trưởng,* một *nhà lãnh đạo*, và còn là một *entrepreneur*. Ở vai nào, Hiển cũng từng trải và tận tâm với nó. Điều đó được thể hiện xuyên suốt trong DevUP, khiến nó trở thành một cuốn sách rất đáng quý cho giới lập trình viên, các nhà quản lý và hơn thế.

Sách của Hiển viết, ngoài việc đáp ứng nhu cầu về kiến thức, còn đáp ứng một nhu cầu tinh thần thực ra là rất cần (và hơi bị thiếu thốn) cho anh chị em trong nghề: nhu cầu được lắng nghe một tiếng nói gần gũi và *authentic* từ một đồng nghiệp đang làm cái nghề rất toàn cầu này, bằng tiếng Việt, từ Việt Nam.

Bùi Anh Dũng (Dave Bui), Nhà sáng lập AhaSlides.com.

* * *

Một điều đáng buồn ở Việt Nam là tôi đã phải gặp rất nhiều các bạn LTV luôn nghĩ rằng chỉ code tới 40 tuổi là phải thăng tiến lên làm quản lý, hoặc cho rằng già không thể làm LTV được. Giờ tôi hy vọng các bạn sẽ được đọc DevUP.

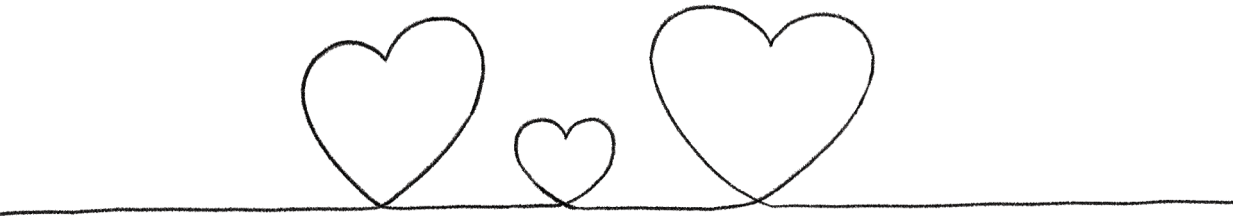
Các bạn có thể rất giỏi lập trình, nhưng lại không tìm ra được thuật toán để phát triển sự nghiệp? DevUP sẽ là quyển sách tốt để bắt đầu.

Trước đây, tôi vẫn phải dành nhiều thời gian tìm kiếm kiến thức để giúp định hướng tốt hơn cho các LTV và giúp các bạn tạo ra giá trị lớn hơn. Bây giờ, tôi mua DevUP.

Hoá ra, cuối cùng LTV Việt Nam cũng có một quyển sách đáng gối đầu giường.

Hãy đọc DevUP để biết cách trở thành 1 LTV có tầm ảnh hưởng thì như thế nào.

Hoàng Minh Châu, Engineering Director @ KiotViet.



Tặng Bean và gia đình.

Bạn thân mến,

Năm 2000, bố tôi tình cờ phải ở nhà một người bạn tại Hà Nội, đó là một cửa hàng dạy vi tính. Ông nghĩ rằng con trai mình sẽ thích những thứ mới mẻ này. Ông lấy hết số tiền còn lại trong nhà, cho tôi theo học 1 tháng và mua cho tôi chiếc máy tính đầu tiên. Như bao người được "đào tạo bài bản" vào thời điểm đó, tôi được học Microsoft Word, Microsoft Excel và Norton Commander; tôi đo lường hiểu biết của mình qua việc đếm những chương trình có thể sử dụng được.

Năm 2001, anh Thơ - một người bạn của anh tôi học Khối chuyên Toán của Đại học Sư phạm về nghỉ hè. Bố mẹ tôi, bằng cách nào đó, đã "đàm phán" để anh ở nhà tôi 1 tháng; đổi lại, anh dạy tôi vi tính. Đó là lúc tôi biết đến Pascal, giải thuật. Tôi biết máy tính có thể làm nhiều hơn việc tạo văn bản, bảng tính hay chơi game; tôi đo lường hiểu biết của mình qua việc đếm những giải thuật có thể học thuộc và cài đặt.

Năm 2002, tôi bắt đầu học tại Khối chuyên Toán Tin, Đại học Khoa học Tự nhiên - nơi tôi học thêm C, C++ và tất nhiên gặp gỡ nhiều ngôi sao lập trình. Tôi đo lường hiểu biết của mình qua những bài toán có thể giải qua những giải thuật. May mắn (hoặc không), tôi biết về OpenGL, Delphi, Windows Forms,... nên bắt đầu viết phần mềm đồ hoạ. Tôi đo lường hiểu biết của mình qua những bài toán có thể giải bằng phần mềm.

Năm 2005, tôi bắt đầu học tại Đại học Công nghệ, Đại học Quốc Gia Hà Nội. Cùng năm, tôi bắt đầu sự nghiệp của một lập trình viên chuyên nghiệp. Do đã có nhiều kiến thức nền tảng, tôi không khó khăn khi duy trì song song việc học và làm; đôi lúc tôi có trốn làm, trốn tiết. C#, Java, Perl, Python, Ruby, PHP, Objective C,... là những ngôn ngữ tôi học tiếp theo. Tôi đo lường hiểu biết của mình qua số lượng ngôn ngữ, nền tảng công nghệ và những hệ thống được xây dựng.

Năm 2009, tôi bắt đầu giảng dạy tại Đại học FPT. Tôi bắt đầu biết đến Agile.

Năm 2013, tôi bắt đầu tại Planday. Tôi thực sự thực hành Agile. Tôi đo lường hiểu biết của mình qua những giải pháp mang lại sự hiệu quả cho người dùng.

Năm 2016, tôi xuất bản Agile Y. Một lần nữa tôi làm công việc đào tạo, huấn luyện, tư vấn như một nghề tay trái. Tôi đo lường hiểu biết của mình qua sự ảnh hưởng tới thành công, thất bại của những cá nhân, tổ chức mình làm việc cùng.

Đó là hành trình 20 năm đầy thú vị. Hành trình thay đổi tư duy. Tất cả xuất phát từ quyết định của bố mẹ tôi vào năm 2000. Tôi xin dành tặng cuốn sách này cho gia đình. Nếu bạn cảm thấy cuốn sách hữu ích, hãy nói lời cảm ơn tới gia đình tôi; đặc biệt là bố mẹ tôi.

Mục lục

VALUATION
Đánh giá

UNLEARN
Học tập

Một lần nào đó trong sự nghiệp, bạn sẽ tự hỏi mình: *"Làm gì tiếp?"*, trả lời rồi bế tắc ở câu hỏi ngay sau: *"Làm thế nào?"*, khiến bạn quay trở lại câu hỏi ban đầu. Đôi khi, bạn sẽ chủ động đi tiếp. Đôi khi, bạn để hoàn cảnh đẩy đưa. Không sao cả, nhiều lập trình viên (LTV) hỏi những câu trên hàng ngày, bế tắc đến mức chán nản không buồn hỏi nữa.

Tôi cũng vậy. Không ít lần. Theo thời gian, tôi cố gắng định hình một cách thức tốt hơn để phát triển sự nghiệp, mô hình hoá và chia sẻ trong cuốn sách này. Hy vọng đây là những gì bạn đang tìm kiếm.

Cuốn sách này không có gì?

Cuốn sách này không có những chỉ dẫn chi tiết về ngôn ngữ, công nghệ, nền tảng,... Thi thoảng, bạn sẽ tìm thấy một số chỉ dấu về ngôn ngữ, công nghệ,... nhưng đó tuyệt nhiên không phải là những gì cuốn sách hướng tới; ngôn ngữ, công nghệ,... chỉ được sử dụng như một ví dụ làm rõ hơn những chỉ dấu lý thuyết.

Cuốn sách này có gì?

Cuốn sách này có những chỉ dẫn mang tính định hướng việc phát triển sự nghiệp lâu dài của LTV. Bởi mang tính định hướng lâu dài, sách có những nguyên lý cơ bản, đơn giản nhưng mang tính nền tảng giúp LTV nâng cao trình độ một cách bền vững theo thời gian.

Ai không nên đọc cuốn sách này?

Những LTV siêu hạng và những LTV *đi-theo-đường-thẳng*.

Tôi từng giành nhiều giải thưởng trong các cuộc thi lập trình chuyên sâu cũng như phát triển phần mềm khi còn là học sinh, sinh viên. Tôi đã xây dựng những sản phẩm tiện dụng và những hệ thống phức tạp trong sự nghiệp dưới nhiều vai trò. Nhưng tôi chưa bao giờ đánh giá mình là một LTV siêu hạng. Vậy nên, tôi chưa thể đưa ra chỉ dẫn để đi đến một vị trí mà tôi chưa từng.

Tương tự, tôi không phải là một LTV được lập trình sẵn sự nghiệp theo một *đường-thẳng* với rõ ràng đích đến và đường đi. Tôi cũng không tin đó là phương thức tốt cho LTV. Vậy nên, tôi không thể viết về nó.

Ai nên đọc cuốn sách này?

Những LTV *bình thường*. Những LTV thích khám phá. Những LTV muốn chủ động phát triển sự nghiệp bền vững thông qua sự chuyên nghiệp và kiểm soát tiến trình phát triển bản thân.

Những nhà quản lý. Những người muốn hiểu LTV cần và muốn gì để hỗ trợ họ phát triển sự nghiệp.

LTV là từ được tôi sử dụng vì đây là một vai trò quan trọng, chiếm tỉ lệ lớn trong ngành Công nghệ thông tin (CNTT); song cuốn sách không chỉ giới hạn cho LTV, bạn có thể tìm thấy những chỉ dẫn này phù hợp với nhiều vai trò khác. Điều đó có nghĩa, nếu bạn muốn phát triển sự nghiệp trong ngành CNTT, cuốn sách này cũng dành cho bạn.

Đọc cuốn sách này như thế nào?

Cuốn sách đề cập tới một mô hình giúp liên tục nâng cao trình độ của LTV, được thiết kế để bạn đọc lại nhiều lần, thậm chí sau mỗi năm bạn vẫn có thể tìm thấy những điều mới mẻ. Sau mỗi phần, có những trang trắng chờ bạn bổ sung thêm những kiến thức, hiểu biết, trải nghiệm,... của bản thân tương ứng với những nội dung trước đó. Do đó, bạn không chỉ là người đọc; cùng tôi, bạn viết cuốn sách cho riêng mình.

Ngôn ngữ sử dụng

Từ *lập trình viên* (LTV) đại diện cho đối tượng độc giả. Trong đa số nội dung, LTV chính là LTV; ngoài ra, LTV là chính bạn dù bạn đang thực hành vai trò khác.

Từ *tổ chức* đại diện cho tập đoàn, doanh nghiệp,...; đôi khi là phòng ban, nhóm,...

Từ *nhà quản lý* đại diện cho vai trò có chức năng quản lý như giám đốc, trưởng phòng, trưởng nhóm,... *Sếp* mang nghĩa tương tự.

Cuốn sách sử dụng tiếng Việt kết hợp tiếng Anh trong các tiêu đề và một số trường hợp giúp bạn dễ dàng tìm kiếm thêm các thông tin trên Internet.

Chúc bạn tìm thấy những điều thú vị và hữu ích trong những trang tiếp theo.

DevUP = Dilemmas + Experiment + Valuation + Unlearn + Performance

Chúng ta cần đồng ý với nhau rằng, trong bất kỳ lĩnh vực chuyên môn nào, phân bổ về trình độ luôn có hình kim tự tháp với những người xuất sắc nhất ở đỉnh tháp. Công nghệ thông tin và nghề lập trình cũng vậy: những LTV xuất sắc nhất ngự trị ở đỉnh tháp; tiếp đến là những LTV khá và trung bình; cuối cùng, là những LTV non nớt và yếu kém ở đáy tháp. Sự coi trọng về nghề nghiệp cũng như tưởng thưởng, qua đó, cũng tăng dần từ đáy tháp lên đỉnh tháp.

James Gosling, LTV quan trọng bậc nhất trong công ty công nghệ hàng đầu, tác giả của Java – một trong những ngôn ngữ lập trình phổ biến nhất thế giới. David Heinemeier-Hansson, LTV đã tạo ra một nền tảng, xây dựng một công ty công nghệ hàng đầu. Linus Torvald, LTV đã sáng tạo ra cả một hệ điều hành đang được sử dụng bởi hàng triệu máy chủ, cũng như sáng tạo ra Git, giao thức được hàng chục triệu LTV sử dụng hàng ngày. Ba LTV huyền thoại này đương nhiên ngự trị ở đỉnh tháp. Phần lớn LTV muốn được như David và Linus nhưng không biết mình đang ở đâu; đa số không thừa nhận mình ở đáy tháp – nơi được coi là đánh dấu sự thất bại trong sự nghiệp

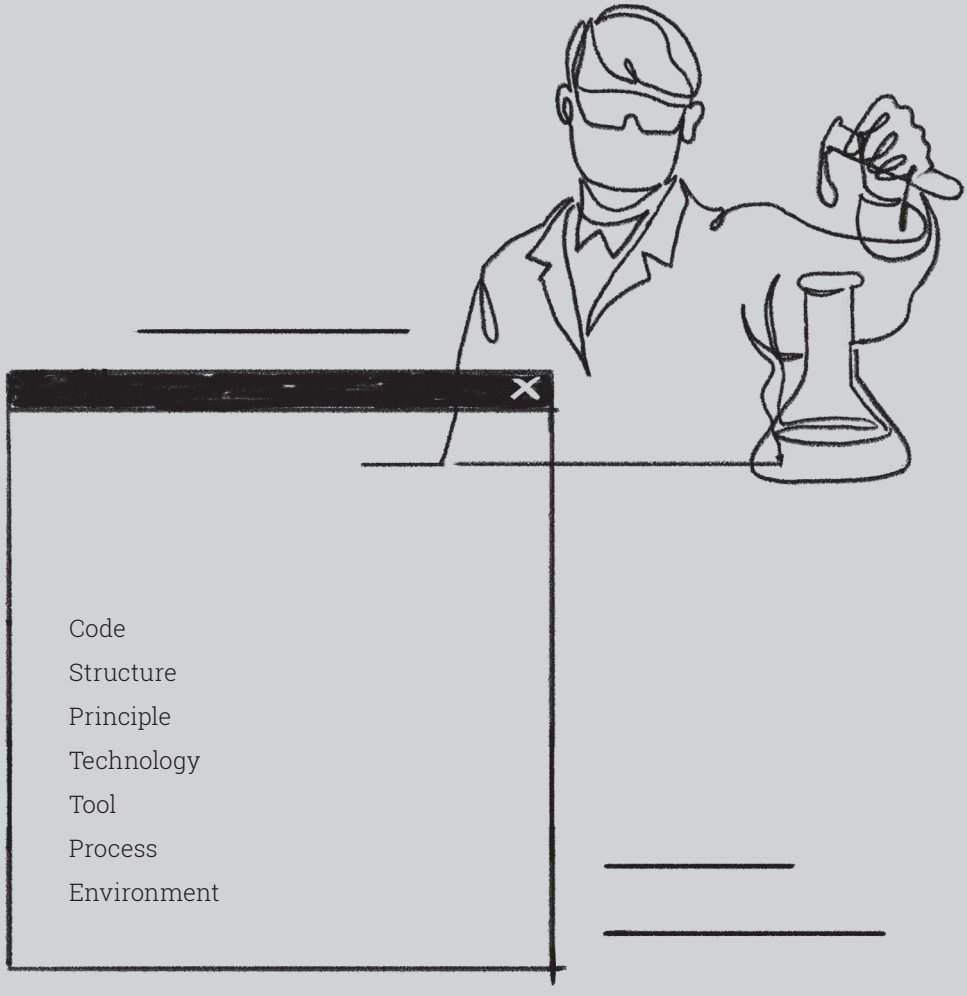
Không sao cả, việc của phần lớn LTV không phải ngay lập tức nhảy lên đỉnh tháp, sáng tạo ra một ngôn ngữ, một nền tảng. Việc của phần lớn LTV, trước hết, là giữ mình không bị tụt sâu xuống đáy tháp; sau đó, từng bước nhỏ đi lên phía trên.

"Trước khi trở thành người hùng, bạn phải sống sót đã."

Đến đây, chắc bạn đã hiểu quan điểm của tôi. Bạn có thể dừng lại nếu thấy cuốn sách này không phù hợp với mình. Vì xuyên suốt những nội dung về sau, sẽ chỉ là những chỉ dẫn cho những LTV tốt và tốt hơn từng ngày; không có chỉ dẫn cho những LTV siêu hạng hoặc những LTV muốn nhanh chóng nhảy cóc tới đỉnh tháp.

Experiment

Thử nghiệm



Code
Structure
Principle
Technology
Tool
Process
Environment

Lập trình là việc khoa học, LTV ưa thích sự rõ ràng của khoa học: đúng và sai - cũng giống như bit chỉ có thể là 0 hoặc 1. Song phần mềm được viết ra để phục vụ nhu cầu cuộc sống. Mà cuộc sống không chỉ có trắng và đen, đúng và sai; điều đó khiến cách thức làm phần mềm cũng như kiến trúc hệ thống hay việc viết code gặp rủi ro lớn nếu chỉ dựa trên việc đúng - sai một cách rõ ràng.

Đúng và sai luôn rõ ràng trong một hệ thống cô lập. Cuộc sống và phần mềm, do tính phức tạp vốn có, không (thể) là một hệ thống cô lập. Tôi hy vọng phần *Dilemmas: Hiểu những thế lưỡng nan* phần nào giúp bạn hiểu được việc xác định tính đúng sai của một ý tưởng/quyết định mà tách biệt ý tưởng/quyết định đó ra khỏi hoàn cảnh là việc không hợp lý. Tuy vậy, chờ tới khi hoàn cảnh xảy ra mới đưa ra những ý tưởng cũng không phải một hành động khôn ngoan. Điều một LTV cần làm, là chuẩn bị kiến thức và kỹ năng để sẵn sàng với những tình huống khác nhau. Cách tốt nhất là liên tục có những *thử nghiệm*.

Code
Mã nguồn

LTV luôn cần bắt đầu từ việc căn bản nhất: code - thử nghiệm với việc code và những giải pháp code khác nhau.

Tương tự, đã bao giờ bạn viết một đoạn code như sau?

```
class Monkey {
  // ...
  double getSpeed() {
    double speed = 0;

    if (type == ASIAN) {
      speed = getBaseSpeed();
    }
    if (type == AMERICAN) {
      speed = getBaseSpeed() - getLoadFactor() * numberOfBananas;
    }
    if (type == EUROPEAN) {
      speed = (isHungry) ? 0 : getBaseSpeed();
    }

    return speed;
  }
}
```

Và bạn biết rằng mình có thể viết nó tốt hơn? *(giảm thiểu số lần kiểm tra điều kiện, và đảm bảo early exit)*.

```
class Monkey {
  // ...
  double getSpeed() {
    if (type == ASIAN) {
      return getBaseSpeed();
    } else if (type == AMERICAN) {
       return getBaseSpeed() - getLoadFactor() * numberOfBananas;
    } else if (type == EUROPEAN) {
       return (isHungry) ? 0 : getBaseSpeed();
  }
}
```

Hoặc tốt hơn nữa? *(dễ đọc hơn)*

```
class Monkey {
  // ...
  double getSpeed() {
    switch (type) {
      case ASIAN:
        return getBaseSpeed();
      case AMERICAN:
       return getBaseSpeed() - getLoadFactor() * numberOfBananas;
      case EUROPEAN:
        return (isHungry) ? 0 : getBaseSpeed();
      default:
        throw new RuntimeException("Should be unreachable");
    }
  }
}
```

Và có thể hơn nữa không? *(anti-if pattern và đảm bảo tính đóng-mở)*

```
abstract class Monkey {
  // ...
  abstract double getSpeed();
}

class AsianMonkey extends Monkey {
  double getSpeed() {
    return getBaseSpeed();
  }
}
class AmericanMonkey extends Monkey {
  double getSpeed() {
    return getBaseSpeed() - getLoadFactor() * numberOfBananas;
  }
}
class EuropeanMonkey extends Monkey {
  double getSpeed() {
    return (isHungry) ? 0 : getBaseSpeed();
  }
}

// ...
speed = monkey.getSpeed();
```

Để thử nghiệm với những giải pháp code khác nhau, tôi khuyến nghị bạn tìm đến những chương trình Code Retreat hoặc Coding Dojo - nơi bạn thực hành với những bài toán nhỏ nhưng đòi hỏi sức sáng tạo không ngừng. Một trong những ví dụ là:

- Lập trình không sử dụng vòng lặp như for, while, do,...
- Lập trình không sử dụng câu lệnh điều kiện như if, switch, case,...
- Lập trình không sử dụng cấu trúc danh sách (array, list, map,...)
- ...

Thật ngớ ngẩn khi không sử dụng những gì được cung cấp bởi ngôn ngữ lập trình? Ý tưởng đằng sau những thử thách này, là khi bị giới hạn bởi các yếu tố, bạn sẽ tìm ra những điều thú vị đằng sau những gì mình có. Bạn có biết rằng đa số các chương trình chạy chậm do có quá nhiều câu lệnh điều kiện; và bạn có thể giải quyết bằng polymorphism?

Hãy tham khảo và thực hành:

- *https://www.coderetreat.org*
- *https://cyber-dojo.org*

Nếu bạn lập trình theo OOP, tôi đặc biệt khuyến nghị bạn thuộc nằm lòng trang web *https://refactoring.com* của huyền thoại Martin Fowler.

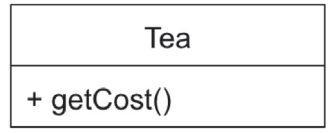
Structure
Cấu trúc

Tiếp theo là structure. Bạn cần thử nghiệm với những cách cấu trúc chương trình khác nhau. Tôi khuyến nghị bạn thử nghiệm sử dụng và không sử dụng các design pattern trong cùng một tình huống và những tình huống khác nhau.

Hãy thử giải bài toán sau: *Xây dựng module tính tiền* cho cửa hàng trà sữa. Bài toán tính tiền cho ly trà sữa sẽ phức tạp khi khách hàng gọi:

cho một ly trà sữa, thêm topping. Và giá thành cũng cần thay đổi cho phù hợp với lựa chọn bổ sung: topping.

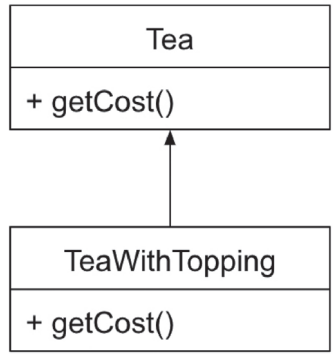
Với OOP rất cơ bản, chúng ta có thể cấu trúc module chỉ với 1 class:



Trong đó, mọi việc tính toán giá thành được xử lý trong hàm getCost(), như:

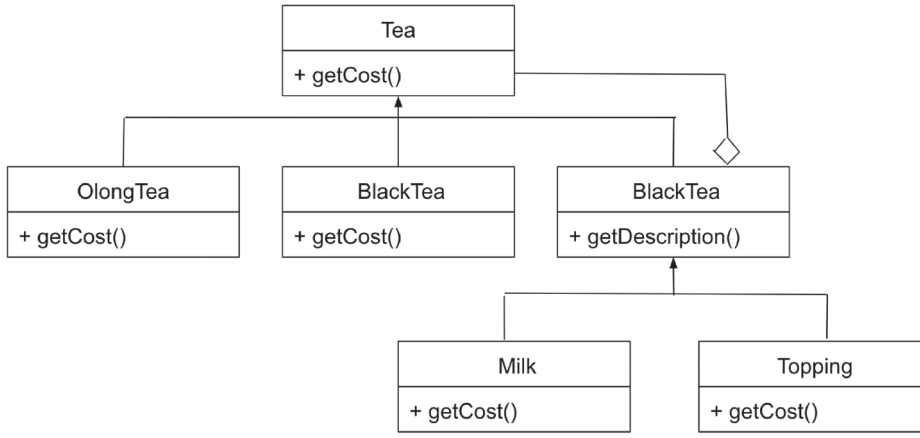
```
float getCost {
  float cost = baseCost;
  if (hasTopping)
    cost += toppingCost;
  return cost;
}
```

Nhưng chúng ta cần tính tới những trường hợp mở rộng tốt hơn, module có thể được cấu trúc như sau:



Tuyệt vời. Bài toán sẽ trở nên thú vị hơn nếu cửa hàng có 4 loại trà sữa: *hồng trà, thanh trà, ô long, hoa cúc* và 10 loại lựa chọn bổ sung (add-on): *sữa, kem, topping*,... Vậy chúng ta sẽ có bao nhiêu sub-class? Trong trường hợp cửa hàng chỉ cho phép khách hàng lựa chọn tối đa 2 add-on cho 1 loại trà, module có trên 300 sub-class (đúng không nhỉ?). Sẽ ra sao nếu cửa hàng có 10 loại trà và 10 add-on? Và cho phép khách hàng chọn thoải mái số lượng add-on?

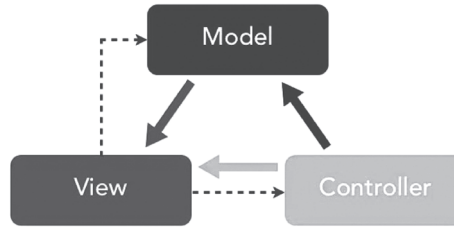
Hãy sử dụng design pattern Decorator như sau:



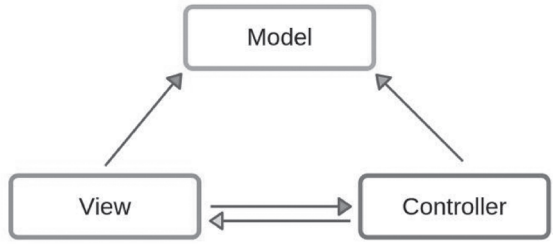
Như vậy chúng ta còn bao nhiêu class và sub-class?

Tôi mời bạn thực hiện một thử nghiệm với một pattern siêu quen thuộc: MVC. Khi nhắc tới MVC, hầu hết các LTV đều tự tin mình hiểu rất rõ về pattern này, gồm 3 thành phần: *Model, View, Controller*. Như vậy mọi chương trình cấu trúc theo MVC đều giống nhau? Thật ra, MVC chỉ là một pattern về thiết kế, việc cài đặt cụ thể phụ thuộc vào từng platform hay nền tảng công nghệ. Từ đó, có rất nhiều loại MVC dựa theo cách kết nối giữa các thành phần M-V-C hay mục đích cụ thể của từng thành phần.

ASP.NET sử dụng MVC theo cách này:



Và iOS sử dụng MVC theo cách này:



Mỗi nền tảng công nghệ có quy chuẩn riêng về MVC như thế nào là đúng đắn và hợp lý. Song bạn đã bao giờ thử nghiệm việc cài đặt không theo quy chuẩn? Như áp dụng MVC của iOS vào ASP.NET và ngược lại? Tất nhiên, bạn sẽ thấy việc này thật ngớ ngẩn. Song hãy thử nghiệm để hiểu một LTV đang thực hành MVC với ASP.NET tự tin thế nào (và nguyền rủa ra sao) khi anh ấy chuyển sang lập trình iOS.

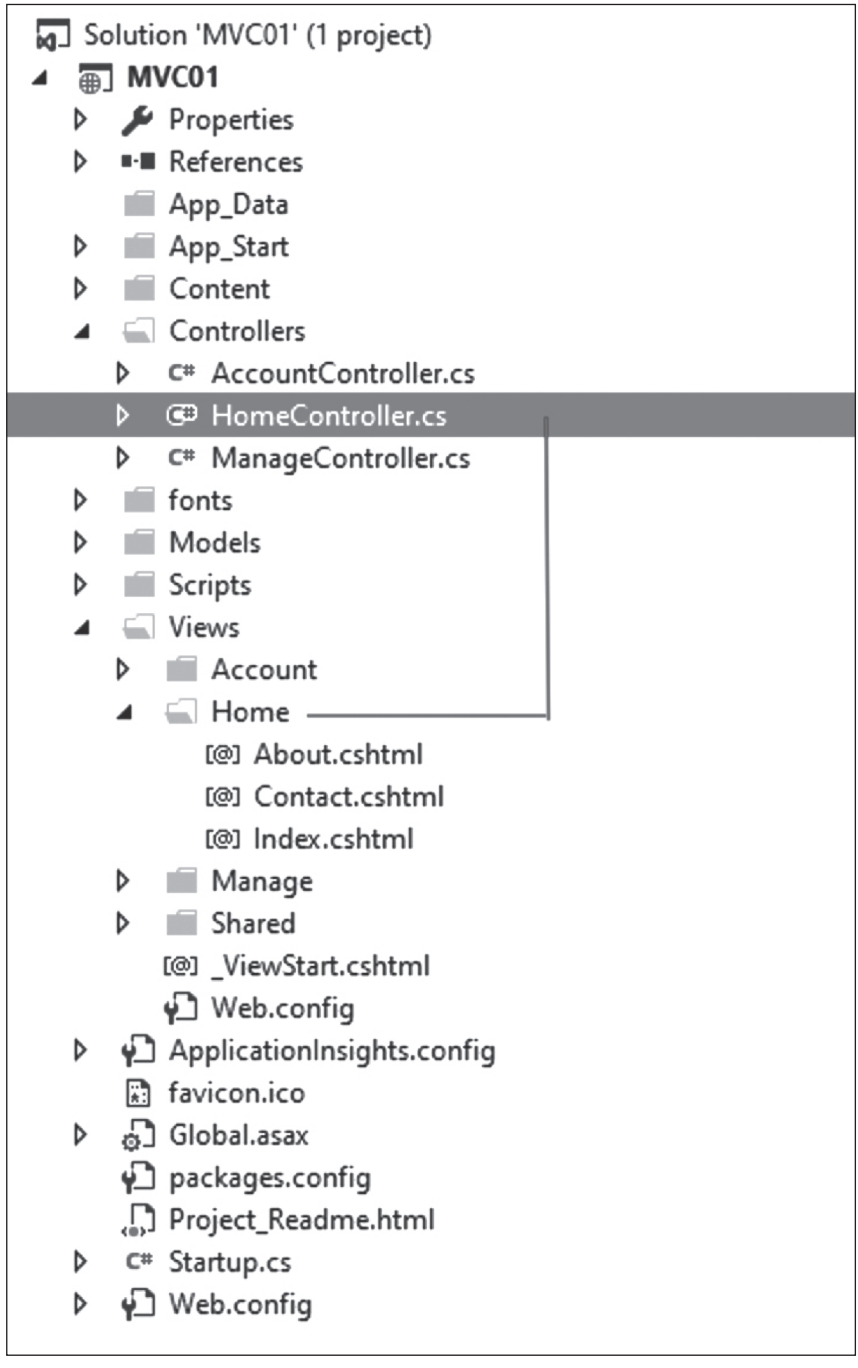
Một thử nghiệm khác bạn có thể thực hiện là với từng thành phần M-V-C. Trong thực tế, việc sử dụng 1 Model để thao tác giữa các thành phần khác là giải pháp không thực sự tốt do sự không phù hợp về cấu trúc. Nhu cầu này tương đối dễ hiểu vì cấu trúc dữ liệu lưu trữ ở database thường dưới dạng table, trong khi dữ liệu để xử lý logic và hiển thị thường dưới dạng object. Nhiều LTV ưa chuộng sử dụng nhiều loại Model khác nhau trong cùng một dự án như:

- View Model: model thể hiện ở phía người dùng

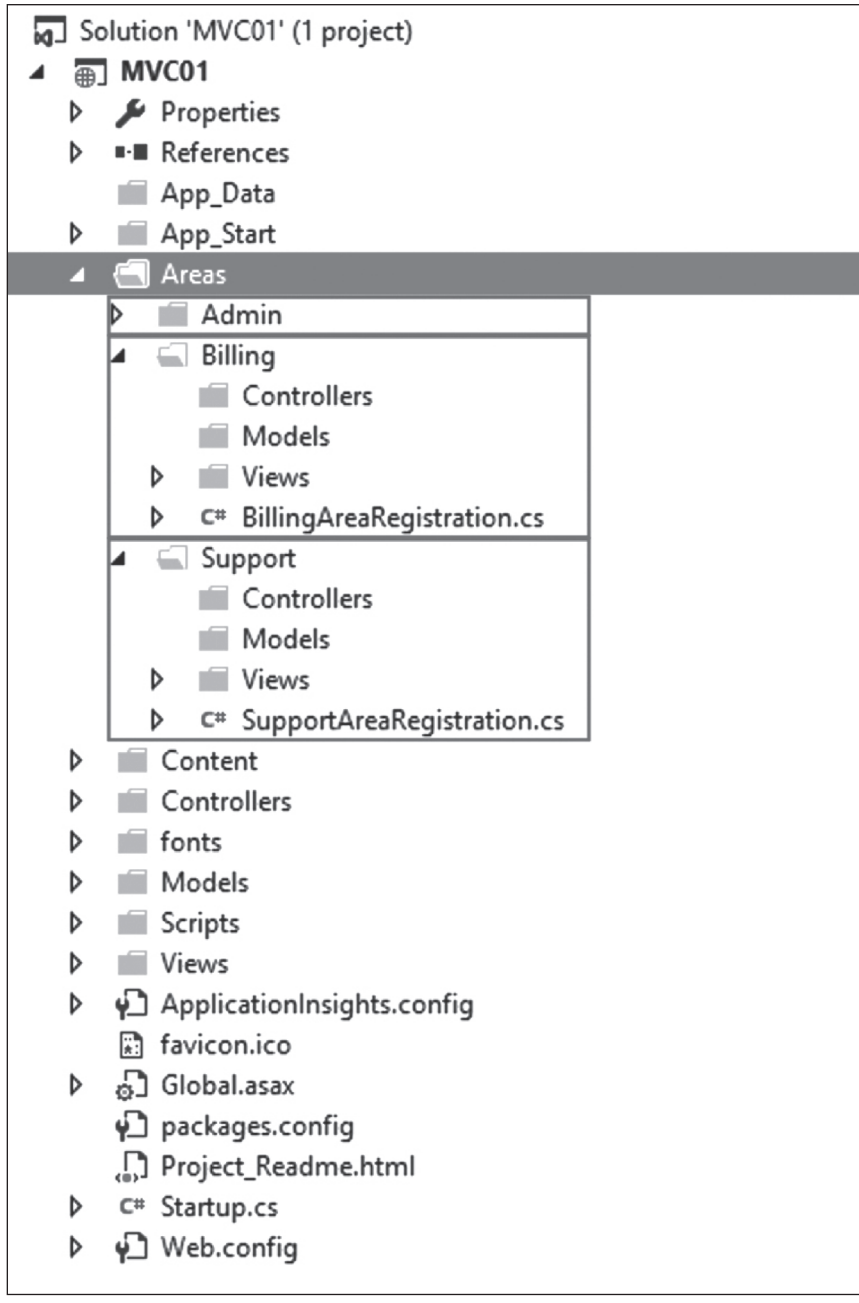
- Binding Model, Mapping Model: model được truyền qua lại giữa các tầng hoặc các thành phần

- Storage Model: model sử dụng cho việc lưu trữ trong file hoặc cơ sở dữ liệu

Và thậm chí, cùng sử dụng MVC, bạn cũng có thể cấu trúc code theo những phương pháp khác nhau. Bạn có thể tổ chức theo các thành phần Controllers, Models và Views, mỗi thành phần tương ứng với một folder. Bên trong mỗi folder là sub-folder hoặc file source code tương ứng với mỗi module.


Solution 'MVC01' (1 project)
MVC01
Properties
References
App_Data
App_Start
Content
Controllers
AccountController.cs
HomeController.cs
ManageController.cs
fonts
Models
Scripts
Views
Account
Home
About.cshtml
Contact.cshtml
Index.cshtml
Manage
Shared
_ViewStart.cshtml
Web.config
ApplicationInsights.config
favicon.ico
Global.asax
packages.config
Project_Readme.html
Startup.cs
Web.config

Hay tổ chức code theo từng module: mỗi module tương ứng với một folder chứa các sub-folder tương ứng với 3 thành phần Controllers, Models và Views.



MVC như thế nào và được cấu trúc như thế nào thì tốt, tôi xin dành câu trả lời cho bạn sau những thử nghiệm của chính bạn. Và MVC cũng chỉ là một trong số rất nhỏ những structure hay pattern bạn có thể thử nghiệm và thử nghiệm theo những cách khác nhau. Tôi khuyến nghị bạn thử nghiệm với những pattern khác gần với MVC như MVP hay MVVM để thấy những ưu và nhược điểm của từng pattern.

Và tất nhiên, hãy thử sức với những pattern và anti-pattern khác.

Một trong những hạn chế của việc thử nghiệm những structure mới là LTV buộc phải làm việc với các hệ thống legacy. Tôi khuyến nghị bạn tìm hiểu và thử nghiệm những tư tưởng trong cuốn sách *Working Effectively with Legacy Code* của Michael Feathers.

Principle
Nguyên lý

Tương tự, hãy thử nghiệm với việc có và không sử dụng những nguyên lý hay sử dụng theo những cách thức khác nhau. Thật ra, nguyên lý mang tính chất định hướng nên không gian để bạn thử nghiệm là rất nhiều.

Đừng cho rằng nguyên lý là những điều quá cao siêu, thử nghiệm nguyên lý có thể đến từ những điều rất nhỏ như những dòng code. Đã bao giờ bạn viết những dòng code như:

```
String getWeekday(int day) {
  switch (day) {
    case 1:
      return “Monday”;
    case 2:
      return “Tuesday”;
    case 3:
      return “Wednesday”;
    case 4:
      return “Thursday”;
    case 5:
      return “Friday”;
    case 6:
      return “Saturday”;
    case 7:
      return “Sunday”;
      default:
      throw new InvalidOperationException(“day must be in range (1,7)”);
  }
}
```

DRY nói gì? Don't Repeat Yourself. Ở đây chúng ta có sự lặp lại tới 7 lần. Vậy có cách nào loại bỏ việc lặp lại này không?

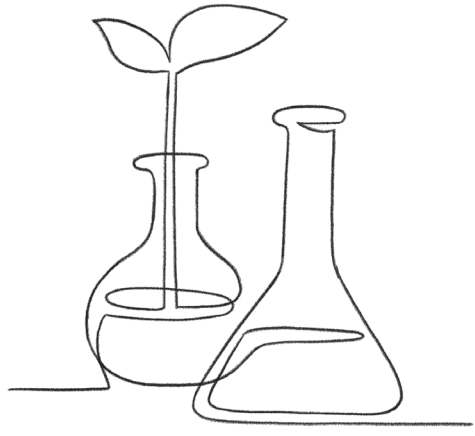
```
String getWeekday(int day) {
  if (day < 1 || day > 7)
    throw new InvalidOperationException("day must be in range (1,7)");

  String[] days = {
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday",
    "Sunday"
  }

  return days[day - 1];
}
```

Không quá khó, đúng không? Giờ hãy đánh giá xem còn sự lặp lại nào không? Mảng days sẽ được lặp lại việc khởi tạo mỗi khi hàm *getWeekday()* được thực thi, nếu hàm này liên tục được sử dụng, chúng ta làm gì để giảm sự lặp lại này? Global variable hoặc static function chính là câu trả lời.

Hoặc thử nghiệm với *KISS (Keep It Simple, Stupid)*. Bạn đã có nhiều giải pháp cho module tính tiền cho ly trà sữa ở trên, đâu là giải pháp tốt nhất cho module *tính tiền cho ly trà sữa tại máy bán hàng tự động?* Có lẽ là giải pháp đầu tiên hoặc thứ hai với không design pattern nào được sử dụng vì máy bán hàng thường không cho phép khách hàng có quá nhiều lựa chọn bổ sung. Đâu là giải pháp tốt nhất cho module *tính tiền ly trà sữa tại quầy tự phục vụ?*



Technology
Công nghệ

Đừng giới hạn bản thân, hãy thử nghiệm với những ngôn ngữ, nền tảng công nghệ khác nhau.

Nếu bạn đang thực hành những ngôn ngữ strong-type, hãy thử những ngôn ngữ scripting.

Nếu bạn đang lập trình mobile trên nền tảng native, hãy thử hybrid hay cross-platform.

Nếu bạn đang sử dụng những thành phần backend phức tạp, hãy thử serverless.

Nếu bạn đang deploy hệ thống với nhiều thư viện phức tạp, hãy thử container.

Nếu bạn đang sử dụng hạ tầng với server vật lý, hãy thử cloud.

Nếu bạn đang sử dụng các dịch vụ cloud của AWS, hãy thử Azure.

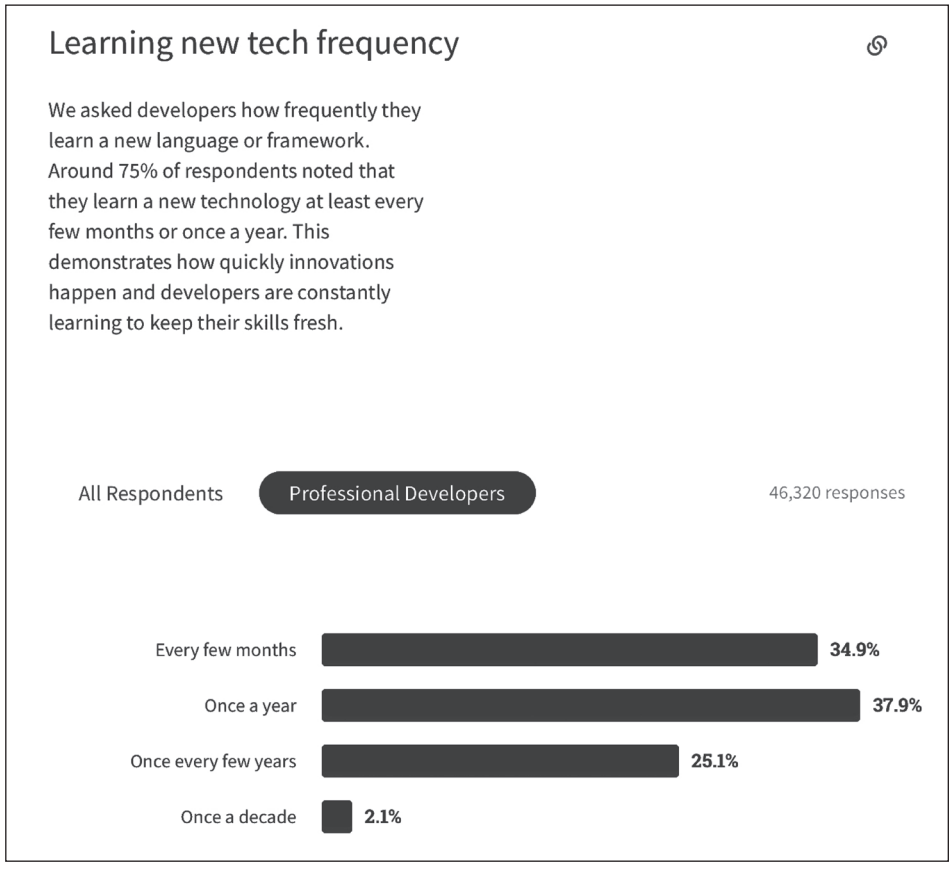
Có quá nhiều thứ bạn có thể thử nghiệm mà tôi không thể liệt kê hết. Công nghệ là thứ thay đổi rất nhanh; đôi khi, một công nghệ mới ra đời, bạn chưa kịp thử nghiệm thì chúng đã bị lãng quên. Điều này dễ gây ra tâm lý chờ đợi của LTV và cố gắng bám chặt những công nghệ lâu đời vì tin rằng những công nghệ khác sẽ trôi qua rất nhanh. Hãy luôn cố gắng thử nghiệm những công nghệ mới, bởi ít nhất hai lý do:

Thứ nhất, tạo cho bạn thói quen thử nghiệm và hiểu rõ xu hướng công nghệ. Dù một công nghệ có qua đi, dựa trên những kết quả thử nghiệm của bản thân, bạn cũng hiểu vì sao nó không được đón nhận. Kinh nghiệm này mài sắc cảm nhận về công nghệ, giúp bạn có đánh giá tốt hơn về những công nghệ mới. Nếu bạn đã trải qua thời kỳ loạn các hệ điều hành cho điện thoại di động và từng thử nghiệm với Bada, bạn sẽ hiểu rằng Tizen không bao giờ có thể trở thành một hệ điều hành phổ biến và có lẽ sẽ dần biến mất trên điện thoại di động; nhưng với sự hậu thuẫn của Samsung và sự chậm chân của Google

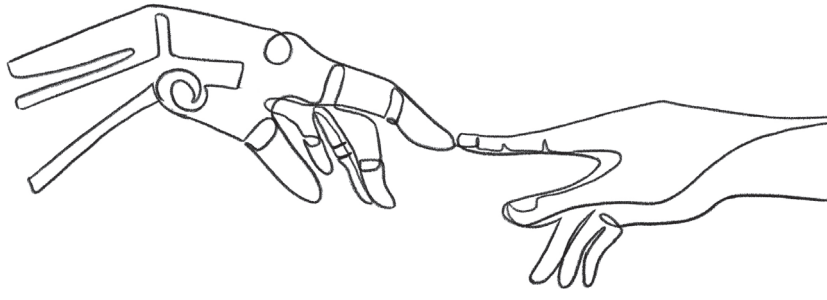
trong mảng thiết bị gia đình, Tizen có cơ hội nhỏ để phát triển trong thị trường smart TV.

Thứ hai, mở ra những cơ hội mới về nghề nghiệp. Công nghệ mới có thể giúp bạn tìm ra hướng đi mới; thậm chí, giúp bạn tránh được thất bại trong nghề nghiệp. Tôi đã gặp nhiều LTV của Nokia, họ không tiếp nhận iOS và Android vì chúng quá mới; chỉ 4 năm sau, họ rất khó kiếm việc với những kỹ năng hiện có và buộc phải học iOS hay Android. So với những LTV mobile khác, họ đi sau 4 năm.

Đừng dừng lại, khảo sát của Stackoverflow vào năm 2020 cho thấy hơn ⅔ số LTV thực hành công nghệ mới hàng năm hoặc chỉ sau một vài tháng. Bạn chắc chắn không muốn là người đứng ngoài cuộc chơi.

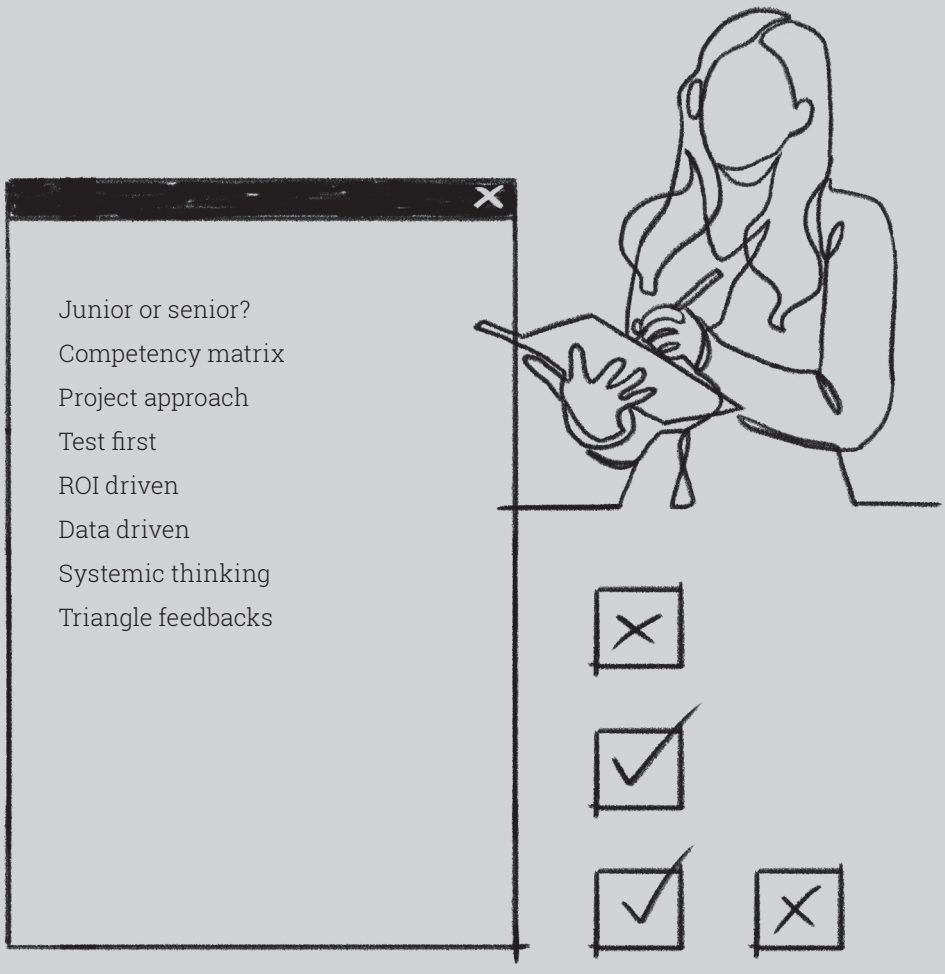


Nhưng nếu bạn thử nghiệm mọi công nghệ mới, chỉ riêng trong phạm vi hẹp của ngành, chính bạn cũng bị kiệt sức bởi số lượng công nghệ mới được giới thiệu là quá nhiều. Quá nửa số công nghệ đó sẽ bị lãng quên chỉ trong 2 - 3 năm. Như tôi đã nói ở lý do thứ nhất, bạn vẫn cần thử nghiệm rất nhiều đến khi đủ kinh nghiệm để nhận diện những công nghệ thực sự tiềm năng. Sau cùng, lượng phải đủ nhiều thì chất mới được chuyển hoá.



Valuation

Đánh giá



Đánh giá luôn là một phần quan trọng trong quá trình phát triển của LTV. Bởi đánh giá và kết quả của đánh giá giúp LTV biết mình đang ở đâu và nên đi tiếp theo hướng nào. Tôi thích dùng từ *định giá (valuation)* hơn là *đánh giá (evaluation)* với ý nghĩa LTV cần nhìn mình như một chủ thể hoàn chỉnh hơn là chỉ đánh giá trên từng khía cạnh nhỏ.

Junior or senior?
Non nớt hay chuyên gia?

Nói về việc đánh giá một LTV, có một câu hỏi muôn thuở: *"Tôi là junior hay senior?"* LTV luôn muốn nhận mình là senior developer, và họ tìm mọi cách để lý giải điều này:

- Tôi có nhiều năm kinh nghiệm, tôi phải là senior.

- Tôi ít năm kinh nghiệm nhưng tôi khá hơn nhiều những người có nhiều năm kinh nghiệm, tôi phải là senior.

- Công ty cũ gọi tôi là senior developer, tôi phải là senior.

Sau cùng, ngoài việc từ *senior* nghe "sang" hơn, quyền lợi của LTV cũng tăng theo. Do đó, đa số LTV luôn mong muốn, bằng cách nào đó, lý giải mình là một *senior developer*.

Cách đây vài năm, tôi có một bài viết về *không ở đâu như Việt Nam, LTV được coi là senior developer chỉ với hơn 2 năm kinh nghiệm*. Nhận xét này được tôi đưa ra khi quan sát thấy nhiều tin tuyển dụng trên Vietnamworks ghi "tuyển dụng senior developer" nhưng trong yêu cầu chỉ cần "hơn 2 năm kinh nghiệm". Bài viết của tôi, khi đó, gây phản ứng đa chiều: những LTV "già cỗi" cho rằng quan điểm của tôi là đúng; những LTV "trẻ và máu" cho rằng tôi đánh giá LTV bằng những đòi hỏi quá cao.

Sau cùng, thế nào là senior developer? Quay trở lại định nghĩa về senior của một nghề nghiệp: trong thời đại công nghiệp, senior được đánh giá qua số năm kinh nghiệm trong nghề; điều này chính xác vì

với những kỹ năng đơn giản và ít biến đổi theo thời gian, kinh nghiệm hơn đồng nghĩa với làm nhanh hơn, chính xác hơn và tốt hơn. Song, lập trình được xếp vào loại công việc có tốc độ biến đổi kỹ năng cao; LTV không thể làm việc cả đời chỉ với một tập các kỹ năng cố định. Do đó, kinh nghiệm hơn không đồng nghĩa với nhanh hơn, chính xác hơn và tốt hơn. Vậy nên, nhiều năm kinh nghiệm không đồng nghĩa với senior. Tôi đã phỏng vấn nhiều LTV đứng đằng sau những sản phẩm rất lớn, nhưng thực sự công việc của họ trong 10 năm chỉ là CRUD dữ liệu; ngày nay với model này và ngày mai với model khác. Họ không biết phải cấu trúc chương trình như thế nào khi phần xử lý logic phức tạp hơn một chút. Mọi người thường nói vui là "một năm kinh nghiệm được lặp lại 10 lần".

Ở chiều ngược lại, quá ít kinh nghiệm cũng không thể coi là senior. Trừ những trường hợp rất xuất sắc, để được coi là senior của một ngành, nhân sự cần khoảng 10.000 giờ thực hành có chủ đích, tương đương với 5 năm làm việc. Hãy nhớ là *có chủ đích*, tức là bạn phải có định hướng để kỹ năng của mình rộng hoặc sâu (hoặc cả hai) vượt bậc hàng ngày. Dù sao, cũng cần đủ lượng thì chất mới được biến đổi - nguyên lý cơ bản của triết học là như vậy. Nhiều LTV mới vào nghề nhưng chăm học, lý thuyết vững, họ tự nhận là senior vì có thể đưa ra những giải pháp đúng đắn về lý thuyết và hợp thời về công nghệ; nhưng thật tiếc, họ chưa bao giờ trải nghiệm nỗi đau của việc áp dụng công nghệ mới khi tổ chức chưa sẵn sàng. Họ cũng chưa bao giờ được trải qua sự suy tàn của một công nghệ từng được coi là của-tương-lai để có quyết định thấu đáo.

Suy cho cùng, sự khác nhau giữa LTV junior và senior là sự trải nghiệm và thực hành có chủ đích sau một khoảng thời gian đủ dài. LTV chỉ có thể thực hành có chủ đích nếu họ định hình rõ và liên tục thực hành những *thử nghiệm* để xử lý những *vùng xám*. Kết quả là, LTV bước dần lên đỉnh tháp. Cuối cùng thì số năm kinh nghiệm, junior hay senior không quan trọng bằng việc bạn đang đứng ở đâu và đang hướng về đỉnh hay đáy tháp?