DEV WORLD
THẾ GIỚI LẬP TRÌNH VIÊN

THẾ GIỚI
NGHỆ NHÂN PHẦN MỀM

VOL.1 - 11.2021

CODEGYM
KIENTHUCLAPTRINH.VN

MỤC LỤC

LỜI NGỎ

Quý bạn đọc thân mến,

DevWorld là ấn phẩm dành riêng cho cộng đồng lập trình viên Việt Nam, với mục đích xây dựng, chia sẻ và phát triển tri thức về lập trình nói riêng và công nghệ nói chung cho cộng đồng chúng ta. Tiền thân của DevWorld là chuỗi ấn phẩm *Tạp chí Lập trình*, đã ra đời lần đầu tiên vào năm 2013 và đã được cộng đồng hưởng ứng đón nhận. Phiên bản DevWorld lần này không chỉ là sự đổi tên, mà còn là sự thích nghi và tái định hướng nội dung và hình thức của các ấn phẩm, để chúng trở nên phù hợp hơn, gần gũi hơn, có chiều sâu hơn và hữu ích hơn.

Với mục đích đó, còn gì tuyệt vời hơn khi chúng ta có ấn phẩm đầu tiên với chủ đề *Thế giới Nghệ nhân Phần mềm* - để tìm hiểu và thảo luận nhiều hơn về nghề nghiệp của mình, về các đóng góp và định hướng phát triển của ngành nghề cũng như của từng cá nhân. Thông qua ấn phẩm này, Ban biên tập kỳ vọng sẽ mang đến cho quý bạn đọc nhiều thông tin hữu ích về ngành nghề, nhiều câu chuyện thú vị về cuộc sống của các lập trình viên, và thông qua đó có thể là nhiều lời khuyên hữu ích - đặc biệt là dành cho các bạn lập trình viên trẻ mới vào nghề.

Ban biên tập cũng mong muốn nhận được sự hưởng ứng của quý bạn đọc, và đồng thời là những phản hồi và đóng góp để cho chuỗi ấn phẩm của chúng ta ngày càng trở nên chất lượng hơn và hữu ích hơn.

Cảm ơn và chúc quý bạn đọc nhiều thành công!

Ban biên tập DevWorld

Manifesto for Software Craftsmanship

Vươn lên tầm cao.

Với tư cách của những Nghệ nhân Phần mềm đầy khao khát, chúng tôi đang nâng tầm ngành phát triển phần mềm chuyên nghiệp bằng cách thực hành và giúp đỡ người khác học tập bí quyết nhà nghề. Qua đó, chúng tôi đã đi đến đề cao:

Không chỉ phần mềm chạy tốt,
mà còn **phần mềm tinh xảo**

Không chỉ phản hồi với thay đổi,
mà còn **tạo giá trị liên tục**

Không chỉ cá nhân và sự tương tác,
mà còn **cộng đồng chuyên nghiệp**

Không chỉ cộng tác với khách hàng,
mà còn **quan hệ đối tác bền chặt**

Trong khi theo đuổi các giá trị ở bên trái, chúng tôi nhận thấy các giá trị ở bên phải là không thể thiếu được.

Ký vào bản Tuyên ngôn

Đọc thêm Metrics

01 PHẦN MỀM CŨNG THỦ CÔNG

Dương Trọng Tấn
CEO Agilead Global

Xưa kia, việc làm phần mềm được coi như là một khoa học nghiêm túc (science), dần dần chuyển sang sản xuất kiểu công nghiệp (software engineering). Nhưng dần dà, người ta thấy nó còn mang tính nghệ thuật, thủ công rất nhiều. Ngày càng nhiều người quan tâm tới phát triển phần mềm theo hướng thủ công này. Thực ra, việc gọi phần mềm là một nghề thủ công (Software Craftsmanship) đã có thâm niên trên dưới tính bằng một hai thập kỉ.

Năm 1999, Andy Hunt xuất bản *The Pragmatic Programmer: From Journeyman to Master* có cái bìa sách đen xì xì cùng với Dave Thomas, mô tả về một "trường phái" lập trình viên kiểu mới. Trong đó, Hunt mô tả một lập trình viên đi lên từ những kẻ ất ơ (Journeman) đến cao thủ (Master) không khác gì một nghệ nhân đi từ học việc đến lành nghề. Mặt khác các tác giả khẳng định, nghề lập trình tuy có những nét tương đồng, nhưng có những điểm không nên thuộc về phạm trù "engineering" theo cách hiểu của các software engineer lúc bấy giờ. Từ sau cuốn sách này, Hunt cho ra đời hẳn một tủ sách có tên "The Pragmatic Programmer" gồm mười cuốn khác nằm trong bộ sách Pragmatic Bookshelf trứ danh. Nhân tiện chú thích về Hunt một tí, bác ấy là đồng tác giả của The Manifesto for Agile Software Development khai sinh ra trường phái mới trong phát triển phần mềm hiện đại, đồng thời ông cũng là sáng lập viên của Agile Alliance. Hunt là một bố già thực sự trong nghề.

Trước đó không lâu (15/5/1998), một giáo sư (Professor Emeritus) ở Đại học Princeton là Freeman Dyson từng có bài biện luận rất đáng chú ý về tính "nghệ" trong việc làm khoa học trong bài viết "*Science as a Craft Industry*". Bài này có đề cập đến một thứ crafts là software.

> *"In spite of the rise of Microsoft and other giant producers, software remains in large part a craft industry. Because of the enormous variety of specialized applications, there will always be room for individuals to write software based on their unique knowledge."*

Năm 2001, Pete McBreen cho chữ "Software Craftsmanship" lần đầu xuất hiện đàng hoàng trong một cuốn sách có cùng tiêu đề ("*Software Craftsmanship: The New Imperative*"). Cuốn này thiên về lí luận, "cãi nhau" với truyền thống là chính, ít hướng dẫn thực hành. Có lẽ do đó mà ít phổ biến.

Năm 2008, Robert Cecil Martin, biệt danh là Uncle Bob xuất bản *Clean Code: A Handbook of Agile Software Craftsmanship*, khẳng định về một thứ trường phái có sách vở, có quy củ đàng hoàng. Ngày nay, lập trình viên khắp nơi trên toàn cầu coi cuốn này không khác gì bảo bối. Thực tế là có những framework lập trình đã đưa "clean code" vào tiêu chuẩn để chuẩn hóa code cho coder. Chú Bob còn đi xa hơn nữa khi cho xuất bản "*The Clean Coder: A Code of Conduct for Professional Programmers*" vào năm 2011, đánh dấu bước hoàn thiện về một "lề luật", "đạo đức nghề nghiệp (code of conduct)" cho cánh lập trình viên hiện đại. Lập trình viên phải là nghệ nhân phần mềm, chị ta phải biết viết mã sạch (clean code).

Cũng năm 2008, triết gia Richard Sennett ở Đại học Yale xuất bản *The Craftsman* cũng liệt kê software vào thế giới nghề thủ công, có chung một số đặc trưng với những công việc nom thì có vẻ chân tay (như làm gốm hay nấu ăn chẳng hạn): có sử dụng đôi bàn tay, truyền nghề kiểu thị phạm, sử dụng và phát minh công cụ đặc thù, làm việc linh hoạt để tạo ra sản phẩm, sinh hoạt như là một cộng đồng (phường hội).

> *"To understand the living presence of Hephaestus, I ask the reader to make a large mental jump. People who participate in "open source" computer software, particularly in the Linux operating system, are craftsmen who embody some of the elements first celebrated in the hymn to Hephaestus, but not others. The Linux technicians also represent as a group Plato's worry, though in a modern form; rather than scorned, this body of craftsmen seem an unusual, indeed marginal, sort of community."*

Đầu năm 2009, bản Tuyên ngôn nghề thủ công phần mềm (*The Manifesto for Software Craftsmanship*) ra đời cùng chữ kí của hàng loạt nghệ nhân có tiếng và vô danh trên toàn cầu, khẳng định một đường lối lập trình kiểu mới.

Các "nghệ nhân" Đới PA (Pham Anh Doi), Tú NN (Tu Nguyen), Khoa NV(Nguyễn Việt Khoa), Hiển NV (Nguyen Hien) và một cơ số người khác nữa đã dịch và truyền bá tinh thần của tuyên ngôn này vào Việt Nam từ những năm 2012 trên Facebook (https://www.facebook.com/SoftwareCraftsmanshipVN/) và một vệt bài dài và rất được đón nhận trên Kiến thức Lập trình (http://kienthuclaptrinh.vn/category/craftsman/tho-lanh-nghe-craftsman/). Hội thảo XPDay hằng năm của Agile Vietnam cũng nỗ lực giương cao ngọn cờ "Thủ công", nhưng có vẻ gió chưa to nên cờ cũng chưa được phần phật lắm.

Từ khi có Tuyên ngôn tới nay, có nhiều phường hội thủ công phần mềm, và những hội thảo của họ cũng xôm tụ không kém những hội thảo khoa học hoành tráng trong giới làm phần mềm. Tìm "Software Craftsmanship" trên Google Scholar cho thấy tình hình quan tâm tới nghề thủ công phần mềm là rất nổi bật. Những nghiên cứu mới từ 2013 trở lại đây không chỉ quan tâm nhiều đến phương diện thực hành của "Software Craftsmanship" với các quy mô và đặc trưng khác nhau, mà còn quan tâm đến cách thức dạy nó trong nhà trường, dùng nó để đổi mới việc dạy software engineering trong các trường đại học.

Bản tuyên ngôn Nghề thủ công Phần mềm ra đời năm 2009.

02 HIỂU ĐÚNG VỀ NGHỀ THỦ CÔNG PHẦN MỀM

David Green
Co-CTO at Paradine
Theo blog.activelylazy.co.uk

Học cách viết phần mềm

Nhiều lập trình viên trưởng thành nhờ tự học. Tôi không biết bạn thế nào, chứ riêng tôi thì đã tự học lập trình từ khi còn trẻ. Tôi yêu lập trình. Tôi thích các thử thách nhằm điều khiển máy tính hoạt động theo ý muốn của mình. Tôi sung sướng sau mỗi lần dành hàng giờ đồng hồ để xử lí các tình huống hóc búa ngỡ như không thể vượt qua. Sự thực là tôi đã bị lôi cuốn và dính chặt vào lập trình.

Sau đó, tôi cũng vào đại học – với mong muốn sẽ được học lập trình một cách bài bản. Trường đại học đã dạy tôi nhiều thứ: logic bậc nhất, lí thuyết hàng đợi, thiết kế trình biên dịch, các hệ thống phân tán. Tất cả đều rất thú vị - về mặt lí thuyết. Nhưng trong số đó chỉ có kiến thức về các hệ thống phân tán là thực sự hữu ích đối với nghề nghiệp của tôi sau này.

Vấn đề là việc viết các phần mềm thương mại không liên quan quá nhiều đến khoa học máy tính. Chắc chắn là việc có kiến thức về độ phức tạp thuật toán là rất tốt, nhưng mà tôi không nhất thiết phải vận dụng kiến thức về độ phức tạp thời gian tuyến tính (*linear time complexity*) và độ phức tạp thời gian đa thức (*polynomial time complexity*) để đánh giá sự nhanh-chậm của chương trình khi có khối lượng lớn dữ liệu đầu vào.

> *Nếu gọi những gì chúng ta đang làm là "khoa học máy tính" thì không khác gì gọi việc nấu ăn là "khoa học của việc sử dụng dao".*

Nếu muốn nấu ăn thì bạn cần phải biết dùng dao. Tuy nhiên, việc nấu ăn thì còn bao gồm nhiều thứ hơn thế. Nấu ăn bao gồm một phần khoa học, một phần nghệ thuật, một phần là sự khéo léo trong ước đoán mùi vị. Tôi cho rằng lập trình cũng tương tự như vậy. Một phần mềm tốt là sản phẩm của khoa học máy tính, các quyết định thiết kế có phần sáng tạo, và sự khéo léo điều chỉnh các dòng mã. Khoa học máy tính thì có thể dạy trong trường học, nhưng những thứ khác thì sao? Làm sao để các lập trình viên mới vào nghề có thể học được các thứ này?

Nghề thủ công

Trong hầu hết các công việc mà cần đến các kiến thức bẩm sinh và các kỹ năng thông qua luyện tập – thì cách tốt nhất đã được chứng minh qua hàng thế kỷ đó là học hỏi từ những người có nhiều kinh nghiệm hơn mình. Đây chính là phương thức hoạt động của nghề thủ công. Người thợ học việc sẽ tìm đến một người thợ cả để học hỏi, sau nhiều năm thì người thợ học việc sẽ học được những kỹ năng này và sẵn sàng để truyền lại cho các thế hệ kế tiếp.

Điều này liệu có xảy ra trong thế giới phần mềm?

Mike Cohn đã có một nhận xét rằng *"không một ai muốn tiếp tục lập trình sau 30 tuổi".* Điều này thật đáng sợ. Nếu tất cả các lập trình viên đều chuyển sang làm huấn luyện viên agile, nhà tư vấn, nhà quản lý hoặc kiến trúc sư hệ thống… thì ai sẽ là người dẫn dắt các thế hệ tiếp theo? Tất cả những kiến thức mà chúng ta đã mất công tìm hiểu rồi sẽ biến mất? Rồi sau đó sẽ lại phải đầu tư công sức để học lại một lần nữa?

Có lẽ tôi là một người may mắn, vì đã được gặp những người thầy tốt trước đây. Nhờ vào những con người vĩ đại đó mà bây giờ tôi đang làm tốt công việc của mình. Họ đã giúp tôi cải thiện hiểu biết, thúc đẩy tôi đạt đến những tầng tri thức mới, để có thể biết được rằng chúng ta phải dày công làm việc để có những sản phẩm tinh xảo. Nhưng có vẻ đối với nhiều người khác thì không được may mắn như tôi. Tôi đã tham gia vào nhiều nhóm, mà ở đó không có những người thầy thực sự tốt. Tôi nghĩ rằng sẽ có rất nhiều lập trình viên mà chưa bao giờ gặp được một người thầy tốt trong suốt sự nghiệp của mình.

Đó là lí do khiến tôi thích ý tưởng về nghề thủ công phần mềm. Tôi thích ý tưởng rằng tất cả các lập trình viên nên luôn luôn cải tiến sản phẩm của mình. Tôi thích ý tưởng rằng các lập trình viên giàu kinh nghiệm nên truyền lại các thủ thuật, các kinh nghiệm của họ. Nhưng làm thế nào mà chúng ta có thể thúc đẩy ý tưởng này? Làm sao chúng ta có thể kêu gọi các lập trình viên liên tục cải tiến? Những gì chúng ta cần là một bản tuyên ngôn.

Bản tuyên ngôn

Bản tuyên ngôn nghề thủ công phần mềm rất tuyệt vời, nhưng tôi không chắc là nó có tạo cảm hứng cho mình hay không. Chắc chắn là nó có thúc đẩy tôi "vươn lên tầm cao – raise the bar". Nhưng ngoài việc nhắc đến cải tiến liên tục thì nó không nói thêm điều gì nhiều cả.

Để so sánh, tuyên ngôn agile đã rất xuất sắc khi đưa ra được một tầm nhìn tốt. Nó định nghĩa rõ agile là gì, và đồng thời cũng chỉ ra một số thứ không phải là agile. Chẳng hạn như, việc cộng tác với khách hàng là điều tuyệt vời – ai cũng muốn cả. Nhưng thứ gây cản trở cho việc cộng tác với khách hàng thông thường là việc đàm phán hợp đồng. Đây chính là lúc mà ý nghĩa thực sự được thể hiện, nếu tôi muốn trở nên agile hơn thì tôi cần cố gắng để giảm phụ thuộc vào việc đàm phán hợp đồng với khách hàng. Bản tuyên ngôn đã giúp tôi biết bằng cách nào mình có thể trở nên agile hơn – điều này tạo cảm hứng cho tôi.

Nhưng tôi cần làm gì để trở thành một nghệ nhân phần mềm tốt hơn? Có phải là chỉ cần liên tục tạo ra giá trị thì sẽ tạo ra được các sản phẩm tinh xảo?

Nghề thủ công phần mềm không là cái gì?

Nghề thủ công phần mềm không phải là để đạt được một cái danh hiệu cá nhân. Thủ công là một tư duy, là cách mà chúng ta sẽ làm việc. Nó không phải là thứ mà bạn có thể trở thành, mà nó là thứ mà tất cả chúng ta đều mong muốn đạt được.

Nghề thủ công phần mềm không phải là một cái chứng chỉ. Tôi phát ốm với các loại chứng chỉ. Tôi đã gặp rất nhiều lập trình viên xuất sắc mà lại không có cái chứng chỉ nào. Chứng chỉ không đồng nghĩa với năng lực.

Con người

Trong suy nghĩ của tôi, mấu chốt của nghề thủ công phần mềm là con người, chứ không phải là phần mềm. Tôi cho rằng tất cả những lập trình viên tốt thì đều có thể trở thành những lập trình viên xuất sắc, với điều kiện họ có được sự trợ giúp phù hợp. Nhưng ai sẽ là người sẵn sàng trợ giúp? Làm sao mà các lập trình viên tìm được sự trợ giúp này? Tôi nghĩ rằng mấu chốt của nghề thủ công phần mềm đó là xác định được người thợ cả lành nghề, là quá trình mà người lập trình viên học viên tìm được cho mình những người thầy vĩ đại.

Cách duy nhất để lập trình tốt hơn đó là thực hành. Cách tốt nhất để học đó là được người khác chỉ ra những lỗi sai của mình. Điều này có nghĩa là bạn cần một người hướng dẫn (mentor) – một người làm việc cùng bạn hằng ngày. Nghề thủ công phần mềm tức là liên tục nhận được phản hồi.

Nếu phải đưa ra một định nghĩa ngắn gọn về nghề thủ công phần mềm, tôi sẽ nói:

- **Năng lực** *hơn là* chứng chỉ
- **Thực dụng** *hơn là* các quy trình cụ thể
- **Hướng dẫn** *hơn là* đào tạo.

03 THẾ GIỚI CỦA CÁC LẬP TRÌNH VIÊN ĐANG THAY ĐỔI

Nguyễn Hiển
Co-founder & CTO BetterMetrics
Co-founder & CTO zen8labs

Thế giới ngày càng khắc nghiệt...

20 năm, 10 năm trước, chuẩn mực của một LTV là gì? Dù là béo phì (LTV Mỹ) hay gầy còm (LTV Việt) thì cũng đều căng thẳng, đầu tóc rối bời, những cặp kính cận lớn và... xa lánh cộng đồng; họ đặt mình vào những căn phòng với thuốc lá, bia, chất kích thích và chỉ giao tiếp với nhau hoặc với máy tính. Chẳng khó để nhận ra một LTV trong đám đông.

Ngày nay, thật khó để nhận ra một LTV trong đám đông. Thật sự xin lỗi, nhưng có lẽ dấu hiệu duy nhất để nhận ra những người làm nghề lập trình là họ thường mang theo laptop. Vậy thôi. Những LTV ngày nay, quá giống một người bình thường với áo thun, quần jeans, đầu vuốt keo, đôi khi cùng khuyên tai hay quần tụt.

Công bằng mà nói, những LTV ngày nay đang có cuộc sống xã hội phức tạp hơn. 20 năm trước, LTV là những người đi tiên phong trong ngành CNTT với chỉ số IQ bắt buộc phải nằm trong top 10% nhân loại, họ luôn phải giải quyết những bài toán khó đòi hỏi nền tảng khoa học tốt; và họ tạo cho mình 1 đặc quyền: tách ra khỏi đời sống xã hội. Giờ đây, như xu thế của mọi ngành công nghiệp phát triển khác, việc lập trình trở nên đại chúng hơn, và LTV dần mất chất hơn. Họ bị kéo trở lại đời sống xã hội bình thường.

Và góc nhìn của LTV cũng buộc phải thay đổi. Cảm giác của LTV cũng buộc phải thay đổi. 20 năm trước, LTV vui mừng vì viết ra 3 dòng code giúp tiết kiệm được 1KB bộ nhớ sau 1 tuần trăn trở. Giờ đây, LTV vui mừng vì tạo ra sản phẩm giúp ích tới hàng ngàn người dùng. Hệ quy chiếu thay đổi, và thước đo cũng đã thay đổi. LTV thay vì đặt mình trong một không gian riêng, chiêm ngưỡng vẻ đẹp của những đoạn code về mặt khoa học thuần tuý; giờ họ buộc phải gắn mình với đời sống kinh doanh phức tạp hơn: Tiết kiệm 1KB bộ nhớ chẳng có ý nghĩa gì nếu nó không mang lại giá trị cho người dùng. 20 năm trước, những LTV được coi là tệ hại nếu không biết cách tiết kiệm thêm 10KB bộ nhớ. Ngày nay, những LTV được coi là thiếu đạo đức nếu viết ra những dòng code tuy tối ưu nhưng khiến đồng nghiệp khó hiểu. Tất nhiên, ở đâu đó trên thế giới, hệ quy chiếu truyền thống vẫn tồn tại, nhưng nó đang ít dần đi.

Và rồi những LTV truyền thống cảm thấy thất vọng, họ thấy thật nực cười khi phải quan tâm tới những giá trị mang lại cho khách hàng, bực bội khi phải quan tâm tới lợi nhuận của doanh nghiệp. Không, tôi muốn làm thế này, vì nó chạy rất nhanh, vì nó là thử thách, vì tôi muốn giải quyết bài toán này... Họ bị stress khi bị lôi trở lại đời sống xã hội phức tạp, nơi những giá trị kinh doanh là thứ họ chưa bao giờ

muốn biết. Tôi chỉ muốn lập trình thôi, trời ơi – một LTV gào lên và ngay lập tức ông chủ của anh ta sẽ đáp lại: Anh bạn à, một bức ảnh đẹp phải được đo bằng view và like chứ không phải vì nó tuân theo tỉ lệ vàng. Thế đấy.

Nhưng thế giới cũng đang đẹp hơn...

Những LTV chân chính có cảm giác rằng LTV không còn là một nghề cao quý và đáng trân trọng khi mà công việc lập trình được bình dân hoá đến mức #kids_can_ code và luôn bị cuốn theo giá trị kinh doanh của doanh nghiệp. Song không thể phủ nhận rằng, cuộc sống của họ đang tốt dần lên chính nhờ những điều đó. Họ biết thêm nhiều kiến thức, kỹ năng mới, hòa nhập hơn đồng nghiệp và xã hội. Họ có thời gian để tập gym, quan tâm tới thời trang, những show ca nhạc... thay vì chỉ vùi đầu trong những đoạn code và chất kích thích.

Nhưng không có nghĩa là những LTV chân chính muốn mất đi không gian riêng. Họ vẫn muốn và vẫn cần có những không gian để không quan tâm tới giá trị kinh doanh, rời xa những ồn ào hiện tại và quay trở lại niềm vui chỉ với những dòng code. Đấy là lý do hàng loạt những website như topcoder, projecteuler, hackerrank... ra đời cùng hoạt động Coderetreat, Code Kata... Và doanh nghiệp cũng ngày càng chú trọng đến những hoạt động như Hackathon nơi họ cố gắng tạo ra một không gian để nhân viên của mình được làm những LTV chân chính dù chỉ 1 vài lần trong năm.

> "Anh bạn à, một bức ảnh đẹp phải được đo bằng view và like chứ không phải vì nó tuân theo tỉ lệ vàng."

Cũng giống như mọi ngành nghề khác, nhiều LTV không muốn nghề nghiệp của mình bị bình dân hoá; nhưng số nhiều và toàn xã hội lại hưởng lợi khi công việc này trở thành đại chúng với những con người cân bằng giữa công việc và đời sống xã hội. Nhưng cuộc chơi giờ là vậy, những LTV, hãy sống cân bằng, biết cách giao tiếp với đồng nghiệp, quan tâm tới những chỉ số kinh doanh và giá trị mang lại cho khách hàng. Nhưng đừng quên rằng, vẫn còn đó những nơi cho chúng ta không gian để chỉ quan tâm tới kỹ thuật, chỉ sống như một LTV chân chính. Hãy tìm lấy không gian để sống như một LTV chân chính nếu bạn thực sự là một LTV chân chính.

04 LẬP TRÌNH VIÊN: LÀM VÌ ĐAM MÊ HAY VÌ TIỀN?

Nguyễn Hiển
Co-founder & CTO BetterMetrics
Co-founder & CTO zen8labs

Đến đây, có một câu hỏi cho mọi ngành nghề: làm vì đam mê hay vì tiền? Đây không phải là một câu hỏi riêng của nhân sự ngành CNTT, song ngành CNTT có những đặc trưng khiến câu hỏi này "hóc búa" hơn nhiều những ngành nghề khác. Bạn không khó để trả lời rằng anh bạn chạy Grab làm vì tiền hay vì đam mê – đa số người làm những công việc đó vì họ không có lựa chọn khác. Nhưng CNTT nói chung và lập trình nói riêng là câu chuyện khác: không ai lựa chọn làm LTV vì lập trình là lựa chọn duy nhất. Lập trình là nghề vất vả, có sự đòi hỏi và đào thải cao. Lập trình là nghề dễ gây "nghiện". Lập trình là nghề, hiện nay, đang được chi trả cao. Ba yếu tố trên khiến câu hỏi làm vì đam mê hay vì tiền khó trả lời hơn nhiều với LTV. Tôi được hỏi câu này hàng chục lần khi tiếp xúc với các bạn sinh viên hoặc LTV mới vào nghề: theo anh, em nên làm vì đam mê hay vì tiền? Có quá nhiều sách báo và những người nổi tiếng khuyên các bạn theo đuổi đam mê; cũng có quá nhiều người nổi tiếng khác nói điều ngược lại.

Nếu có một lời khuyên rõ ràng cho LTV, tôi đã không đưa câu hỏi này vào phần Hiểu những thế lưỡng nan. Tôi chỉ có thể đưa ra một vài gợi ý.

Thứ nhất, nếu bạn thực sự yêu thích nghề lập trình, hãy yên tâm theo đuổi đam mê. Với nhu cầu lớn từ thị trường hiện nay, LTV có đam mê (và từ đó, dần hình thành kiến thức và kỹ năng tốt) luôn có cơ hội được trả lương cao và rất cao. Đây là điều thuận lợi mà gần như chỉ ngành CNTT có tại thời điểm này. Là một LTV có đam mê với nghề, bạn nên cảm thấy may mắn. Không một nghề nghiệp nào có sự đảm bảo chắc chắn về tiền bạc sẽ đi kèm đam mê nhưng CNTT thì có: LTV có đam mê được đảm bảo sống tốt thậm chí là sung túc, chỉ cần họ thực sự mê (là đắm chìm vào công việc bằng hành động) chứ không chỉ là thích (và không hành động gì). Thứ hai, nếu bạn vẫn đang loay hoay với câu hỏi trên, hãy làm vì tiền. Tôi tin rằng những LTV có đủ đam mê đang được trả công xứng đáng, và họ không màng trả lời câu hỏi này. Tuy vậy, không có gì sai trái nếu bạn chọn nghề lập trình vì tiền. Tư duy làm vì tiền thậm chí cũng tốt vì giúp bạn chuyên nghiệp hơn: bạn lo sợ bị đào thải và vì thế, cập nhật công nghệ thường xuyên; bạn muốn được trả lương cao hơn và vì thế, hoàn thành công việc với chất lượng tốt hơn và nhanh hơn… Nỗi sợ hãi và mong muốn về vật chất nhiều khi là động lực tốt để con người phát triển. Phần lớn người Việt Nam, công dân của một nước nghèo đang phát triển, đang làm việc chăm chỉ vì sự ám ảnh nghèo đói từng đã trải qua. Điều đó không có gì sai trái, thậm chí đáng tự hào.

Tôi từng tổ chức một buổi họp nhóm, để từng thành viên chia sẻ thẳng thắn lý do họ gắn bó với công ty và làm việc trong nhóm. Có những LTV nói rằng, họ yêu thích công việc và sản phẩm này, họ học được rất nhiều, họ chấp nhận mức lương thấp hơn nhiều lời đề nghị từ những công ty khác. Có những LTV thể hiện rõ quan điểm rằng họ cần tiền để có khoản tích lũy cho mục đích xa hơn. Tôi đánh giá rất cao những LTV thẳng thắn và rõ ràng. Sau cùng, họ đều nhận được thành quả xứng đáng khi có hành xử chuyên nghiệp để đồng bộ được mục tiêu của cá nhân với mục tiêu của tổ chức.

Đáng ngại nhất là những LTV không rõ ràng trong câu trả lời cho câu hỏi làm vì đam mê hay vì tiền?. Họ không có đam mê và cũng không có nhu cầu có thêm tiền vì đang được xã hội trả công ở mức cao để có một cuộc sống "đủ tốt". Họ ở đáy của kim tự tháp mà tôi đã đề cập, và cũng không có nhu cầu di chuyển dần lên đỉnh tháp. Không chỉ ảnh hưởng tới bản thân, họ cũng đẩy tổ chức của mình vào thế khó xử.

05 LẬP TRÌNH VIÊN VÀ CÂU CHUYỆN PHÁT HÀNH SẢN PHẨM

Nguyễn Khắc Nhật
CEO CodeGym

Mười mấy năm trước

Mười mấy năm trước, các phần mềm được phát hành chậm hơn rất nhiều, và khối lượng của các đợt phát hành cũng lớn hơn rất nhiều so với bây giờ.

Chẳng hạn, thời của những hệ điều hành như Windows XP, Windows Vista thì phải mất đến một vài năm chúng ta mới nhận được một phiên bản cập nhật. Và mỗi bản cập nhật đó cũng chứa đựng rất nhiều thay đổi, kể cả về tính năng lẫn giao diện và bảo mật. Hoặc, các phần mềm phổ thông khác như trình duyệt IE, Firefox, các phần mềm nghe nhạc cũng rất lâu mới được cập nhật, thông thường là tính bằng đơn vị nửa năm hoặc một năm.

Năm	Số lần phát hành
2001	2 phiên bản
2002	1 phiên bản
2003	1 phiên bản
2004	1 phiên bản
2005	1 phiên bản

Bảng 1: *Số lần phát hành rất thưa thớt của trình duyệt IE trước đây*

Có nhiều nguyên nhân để việc phát hành phần mềm ở giai đoạn này diễn ra chậm như vậy, một phần là do tư duy về phát triển sản phẩm truyền thống, một phần khác là do các kỹ thuật và công cụ phát triển phần mềm chưa cho phép các nhóm có thể phát hành sản phẩm một cách thường xuyên, và tất nhiên cũng không thể không kể đến nguyên nhân do trình độ của lập trình viên chưa đáp ứng được.

Ở trong quá khứ, để phát triển mới hoặc nâng cấp một phần mềm, các nhóm phát triển thường dành nhiều thời gian để lập kế hoạch rất chi tiết, sau đó lập trình và rồi kiểm thử lần lượt cẩn thận trước khi tung ra cho người dùng cuối. Với tư duy phát triển sản phẩm như vậy cho nên vòng đời của việc phát triển sản phẩm thường kéo dài.

Các kênh để phát hành sản phẩm trước đây cũng rất khác so với bây giờ, khi mà hầu hết các phần mềm đều được tải về và cài đặt thủ công lên máy của người dùng, khó khăn và bất tiện hơn rất nhiều so với việc phân phối phần mềm dưới dạng dịch vụ như ngày nay.

Mười mấy năm trước, hãy thử tưởng tượng tình huống một phần mềm mới được cập nhật, hằng trăm nghìn, thậm chí là hằng triệu người dùng cần lên website của nhà sản xuất, tải về và tự cài đặt thủ công. Mất rất nhiều thời gian và nỗ lực, sai sót và rủi ro. Chẳng thế mà trong giai đoạn đó, vai trò của những người hỗ trợ kỹ thuật là rất quan trọng, họ là các anh hùng trong mắt những người dùng bình thường, bởi vì họ có khả năng cài đặt và cấu hình các sản phẩm rất bình dân.

Mười mấy năm trước, các kỹ thuật lập trình như kiểm thử tự động, tích hợp liên tục vẫn chưa được sử dụng phổ biến và chưa hiệu quả như bây giờ, do đó rất nhiều thao tác cần phải thực hiện thủ công, mất rất nhiều thời gian và công sức. Do đó thời gian phát triển bị kéo dài cũng là chuyện dễ hiểu.

Dăm năm trước

Dăm năm trước, các phần mềm bắt đầu được phát triển và cập nhật thường xuyên hơn, các bản phát hành cũng thường là nhỏ hơn, đôi khi chỉ là để sửa một vài lỗi nhỏ, hoặc là cải thiện một chút về trải nghiệm người dùng. Việc phát hành sản phẩm theo từng tháng, từng tuần hoặc thậm chí là từng ngày là chuyện không còn xa lạ gì.

Dẫn đầu cho xu hướng phát hành thường xuyên thì phải kể đến trình duyệt Chrome, mặc dù ra muộn hơn so với các trình duyệt như IE, Fire- fox hay Opera, nhưng cách phát hành liên tục của Chrome đã khiến cho các trình duyệt khác không theo kịp, và chẳng mấy chốc bị biến thành các gã đối thủ chậm chạp, lỗi thời. Chẳng hạn, trong năm 2010, Chrome có 6 phiên bản phát hành, trong năm 2011 có 8 phiên bản, năm 2012 có 12 phiên bản, năm 2013 có 23 phiên bản... và cứ như vậy.

Năm	***Số lần phát hành***
2010	6 phiên bản
2011	8 phiên bản
2012	12 phiên bản
2013	23 phiên bản
2014	24 phiên bản
2015	24 phiên bản

Bảng 2: *Số lần phát hành rất dày đặc của trình duyệt Chrome*

Đặc biệt, với sự thịnh hành của các thiết bị điện thoại thông minh thì tần suất phát hành của các phần mềm trên đó còn trở nên đều đặn hơn bao giờ hết. Các phần mềm trên điện thoại di động, bao gồm cả ứng dụng lẫn trò chơi, đều được cập nhật rất thường xuyên, có thể theo cả đơn vị ngày hoặc giờ.

Đạt được trình độ phát hành nhanh như vậy thì phải kể đến sự dịch chuyển trong tư duy phát triển sản phẩm, sự hỗ trợ về mặt kỹ thuật và trình độ của các nhóm phát triển phần mềm ngày càng tốt lên.

Tư duy phát triển phần mềm ngày càng linh hoạt hơn, hướng đến việc phục vụ người dùng hơn, lắng nghe người dùng hơn, do đó các nhóm đều cố gắng nhanh chóng tung ra sản phẩm và thử nghiệm các tính năng để thu thập được phản hồi của người dùng. Do đó, việc rút ngắn thời gian phát triển và phát hành là yêu cầu bắt buộc đối với các sản phẩm.

Sự phát triển của các công nghệ, công cụ và nền tảng cũng là một yếu tốt rất quan trọng giúp cho các nhóm có thể tăng tốc trong việc phát triển và phát hành sản phẩm. Chẳng hạn, với sự hỗ trợ của các công cụ kiểm thử tự động, giờ đây thay vì mất hằng tuần, hằng tháng để kiểm thử các sản phẩm thì chúng ta chỉ mất vài phút để có thể chạy một loạt các kiểm thử.

Các công cụ tích hợp liên tục và chuyển giao liên tục cũng giúp cho quá trình ổn định sản phẩm nhanh hơn, quá trình cài đặt và cập nhật sản phẩm hoàn toàn được tự động hoá, giúp giảm thiểu sai sót và tiết kiệm được rất nhiều thời gian. Chẳng hạn, với việc sử dụng các công cụ như Docker kết hợp với quy trình chuyển giao liên tục mà một phần mềm có thể được phát hành và cập nhật trên hằng trăm, thậm chí là hằng nghìn máy một cách rất dễ dàng trong khoảng thời gian tính bằng phút.

Bây giờ

Xu hướng phát hành phần mềm dưới dạng dịch vụ (SaaS - Software As A Service) vẫn tiếp tục thể hiện các lợi ích của mình và đang rất phổ biến. Các công cụ và kỹ thuật tự động hoá cho gần như tất cả các công đoạn trong phát triển sản phẩm ngày càng trở nên thông minh hơn, hoàn thiện hơn, giúp cho lập trình viên tập trung sức lực vào xây dựng nghiệp vụ cho phần mềm, thay vì phải mất công sức vào những thao tác kỹ thuật lặp đi lặp lại.

Các hệ thống phần mềm ngày càng trở nên khổng lồ hơn, phức tạp hơn nhưng đồng thời lại yêu cầu sự linh hoạt, dễ thay đổi, dễ mở rộng hơn, do đó quy trình phát triển phần mềm và trình độ của các lập trình viên cũng phải phát triển hơn để đáp ứng được nhu cầu này.

Trước đây, lập trình viên cố gắng để phần mềm "chạy được" là đã coi như hoàn thành nhiệm vụ rồi, nhưng bây giờ thì yêu cầu phải "chạy tốt", rồi thì "dễ mở rộng", "dễ thay đổi", "dễ kết nối". Trước đây, các nhóm phần mềm thường có quy mô nhỏ, việc cộng tác diễn ra đơn giản giữa các thành viên gần gũi nhau, giờ đây, các nhóm phần mềm có thể lên đến vài trăm thậm chí vài nghìn thành viên, phân bố ở rất nhiều nơi trên thế giới, việc cộng tác diễn ra rất chặt chẽ và khắt khe với các tiêu chuẩn cao về mặt chất lượng, do đó trình độ của lập trình viên cũng phải đáp ứng tương xứng.

06 DevOps

Giải pháp phát hành phần mềm nhanh chóng

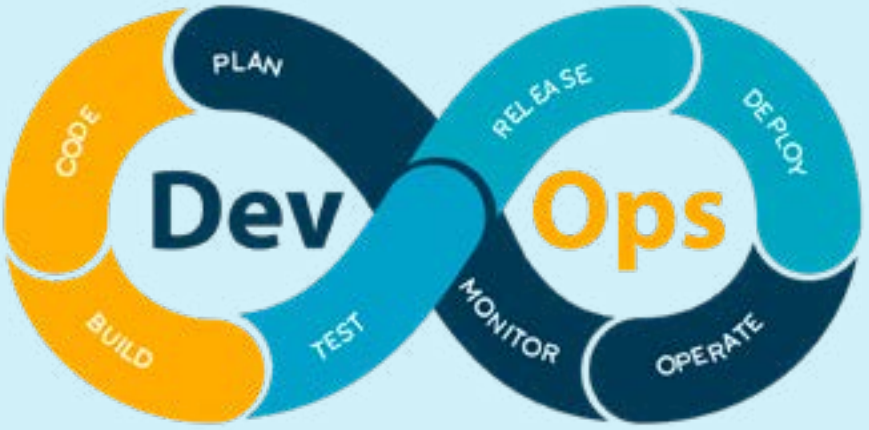
Nhanh chóng phát hành một sản phẩm mới hoặc tính năng mới ra thị trường là nhiệm vụ đầy thử thách với mọi công ty trên thế giới. Việc hóc búa nhất là làm sao để các nhóm riêng biệt: phát triển, QA và vận hành IT làm việc cùng nhau để hoàn thành công việc và phát hành sản phẩm nhanh nhất có thể.

Các quy trình và kỹ thuật đã trải qua một quá trình tiến hóa để giải quyết thử thách này. Một thập kỷ trước không ai biết đến từ DevOps, nhưng vào năm 2009, một phương pháp đã tổng hợp các quy trình để phối hợp và giao tiếp giữa nhóm phát triển, QA và vận hành IT giúp rút ngắn đáng kể thời gian đưa sản phẩm ra thị trường bắt đầu phổ biến với tên gọi DevOps.

Phương pháp DevOps là một tập hợp các kĩ thuật được thiết kế để giải quyết những vấn đề rời rạc giữa nhóm phát triển, QA và vận hành thông qua hợp tác và giao tiếp hiệu quả, kết hợp chặt chẽ quy trình tích hợp liên tục với triển khai tự động. Giúp tạo ra môi trường để phát triển, QA và phát hành phần mềm ra thị trường nhanh chóng, ổn định.

Phương pháp Truyền thống vs DevOps

Theo phương pháp thác nước truyền thống, nhà phát triển viết mã cho các yêu cầu trên môi trường local. Khi phần mềm được build, đội QA kiểm thử phần mềm trên một môi trường tương tự như môi trường production. Cuối cùng, khi đã đáp ứng các yêu cầu, phần mềm được phát hành cho bên vận hành. Quá trình từ khi thu thập yêu cầu đến khi triển khai sản phẩm vào vận hành mất nhiều thời gian. Do hai bên phát triển và vận hành làm việc độc lập, khả năng cao là sản phẩm cuối cùng sẽ mất thêm nhiều thời gian nữa để đưa vào vận hành. Ngoài ra, sản phẩm có thể không chạy đúng như mong đợi hay gặp những trục trặc khác.



Khi đó, quy trình DevOps có một phương pháp tốt hơn để giải quyết những vấn đề trên. DevOps nhấn mạnh vào việc phối hợp và giao tiếp giữa nhóm phát triển, nhóm QA và nhóm vận hành để thực hiện việc phát triển liên tục, tích hợp liên tục, chuyển giao liên tục và giám sát các quy trình liên tục bằng cách dùng các công cụ kết nối các nhóm giúp đẩy nhanh tốc độ phát hành. Nhờ đó công ty nhanh chóng thích ứng với những thay đổi từ yêu cầu kinh doanh.

Quan hệ giữa Agile và DevOps

DevOps ra đời một phần dựa trên khả năng phát hành sản phẩm nhanh khi công ty áp dụng Agile. Nhưng có thêm những quy trình khiến DevOps khác biệt với Agile.

Các nguyên lý của Agile chỉ áp dụng cho quá trình phát triển và QA, Agile tin tưởng vào việc tạo ra các nhóm nhỏ phát triển và phát hành phần mềm chạy tốt trong một khoảng thời gian ngắn, được gọi là Sprint. Nhóm chỉ tập trung vào Sprint và không giao tiếp với bên vận hành.

Còn DevOps lại chú trọng hơn vào việc phối hợp suôn sẻ giữa các nhóm phát triển, QA và vận hành trong suốt chu trình phát triển. Nhóm Vận hành liên tục tham gia thảo luận cùng nhóm phát triển về các mục tiêu của dự án, lộ trình phát hành và các yêu cầu kinh doanh khác. Từ ngày đầu, nhóm vận hành nên đưa ra các yêu cầu liên quan đến vận hành cho nhóm phát triển, sau đó kiểm tra lại chúng. Việc giám sát dự án liên tục cùng với giao tiếp hiệu quả và thường xuyên giúp công ty có thể nhanh chóng phát hành.

Bạn sẽ nhận được gì từ DevOps?

Hãy xem xét vài lợi ích có thể nhận được từ quy trình và văn hóa DevOps.

1. Lợi ích đầu tiên ta sẽ nhận được là DevOps giúp giảm đáng kể thời gian đưa sản phẩm ra thị trường nhờ kết nối giữa các nhóm làm việc và làm theo qui trình phát triển liên tục.

2. Nhờ các nhóm đồng bộ tốt hơn, các thành viên có được tầm nhìn rõ ràng về các công việc đang làm, do đó họ có thể thấy các vấn đề hoặc các khó khăn trước khi chúng thực sự xảy ra. Nhờ vậy thành viên nhóm có kế hoạch tốt hơn để vượt qua các vấn đề đó.

3. Nhờ sự minh bạch trong quy trình, những người phát triển sản phẩm sẽ cảm thấy mình là chủ sản phẩm. Họ thực sự sở hữu mã từ khi bắt đầu đến lúc vận hành.

4. Việc triển khai tự động sản phẩm bằng các công cụ tự động hóa trong nhiều môi trường cho phép bạn nhanh chóng thấy được các vấn đề liên quan đến môi trường. Với những phương pháp khác thì thường tốn rất nhiều thời gian để phát hiện được các vấn đề này.

Những công cụ cho DevOps

Vì DevOps là sự cộng tác của Phát triển, QA và Vận hành, không thể có một công cụ duy nhất đáp ứng được tất cả các nhu cầu. Vì vậy cần nhiều công cụ để thực hiện thành công mỗi giai đoạn.

Hãy xem thử một vài công cụ cho mỗi giai đoạn ở dưới đây.

- Monitoring: Nagios, NewRelic, Graphite ...
- Virtualization và Containerization: Vagrant, VMware,
- Công cụ để build, test and deployment: Jenkins, Maven, Ant, Travis, Bamboo, Teamcity...
- Quản lý cấu hình (configuration management): Puppet, Chef, Ubuntu Juju, Ansible, cfengine ...
- Orchestration: Zookeeper, Noah ...
- Cloud services: Azure, Openstack, Rackspace ...

Ngoài ra còn có các công cụ cho việc merge code, kiểm soát phiên bản,... giúp áp dụng hiệu quả các qui trình DevOps.

Các hiểu nhầm về DevOps

Dù DevOps đã ra đời hàng thập kỷ, nhưng nó vẫn rất mới mẻ với nhiều người. Do đó, vẫn còn nhiều hiểu nhầm.

Vài người nghĩ rằng triển khai DevOps cho phép nhà phát triển làm luôn công việc của bên vận hành. Điều này hoàn toàn không đúng. DevOps nhấn mạnh vào việc cộng tác từ cả hai nhóm, do vậy những nhà phát triển có các kỹ năng vận hành sẽ có lợi thế trong chu kỳ kinh doanh nhanh chóng hiện nay.

Cũng có rất nhiều người tin rằng có thể triển khai DevOps bằng cách dùng một bộ các công cụ cho tất cả phần việc. Điều đó không đúng; dùng vài công cụ không giúp ta đạt được DevOps, mà ta chỉ có thể đạt được các giá trị cốt lõi của nó thông qua thực sự triển khai quy trình DevOps và khôn ngoan chọn dùng các công cụ.

(Theo AgileBreackfast)

07 TỰ ĐỘNG HOÁ HOẠT ĐỘNG KIỂM THỬ - NHU CẦU CỦA PHÁT TRIỂN PHẦN MỀM HIỆN ĐẠI

Nguyễn Khắc Nhật
CEO CodeGym

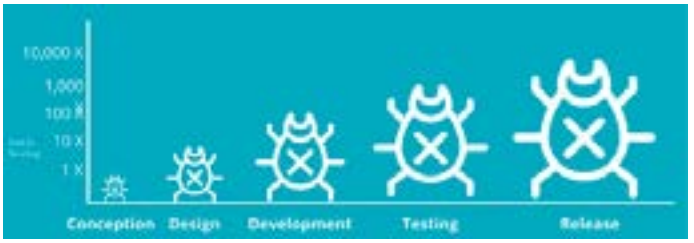
Lỗi trong các sản phẩm dẫn đến thiệt hại vô cùng lớn cho các nhà sản xuất và cho xã hội, không riêng gì trong ngành phần mềm. Hẳn chúng ta dễ dàng tìm thấy các thông tin liên quan đến lỗi trên các dòng xe hơi của các hãng như Toyota, Ford, Honda, Mitsubishi... dẫn đến việc phải triệu hồi và khắc phục hàng triệu chiếc xe. Hoặc sự cố pin gây cháy nổ của dòng điện thoại Galaxy Note 7 của Samsung khi vừa mới ra mắt, mà theo ước tính có thể gây thiệt hại lên đến 17 tỉ USD cho hãng. Hay như lỗi phần mềm của dòng máy bay Boing 737 MAX dẫn đến các tai nạn kinh hoàng ở Indonesia và Ethiopia, không chỉ gây thiệt hại về kinh tế mà còn để lại những hậu quả không thể khắc phục được - hàng trăm người đã thiệt mạng do các sự cố này.

Hình 1: Sự cố của Boing 737 MAX

Bởi những hậu quả vô cùng lớn như vậy, cho nên nhiệm vụ đảm bảo chất lượng luôn là ưu tiêu hàng đầu của các nhóm phát triển sản phẩm, trong đó hoạt động kiểm thử được coi là một chốt chặn quan trọng để phát hiện và xử lí các lỗi tiềm ẩn. Cũng bởi vì thế mà trong ngành phần mềm nói riêng, chúng ta luôn cố gắng để đưa ra các phương pháp, kỹ thuật và công cụ hiệu quả nhất để phát hiện và xử lí lỗi càng sớm càng tốt.

Có một nguyên tắc mang tính chất hiển nhiên trong phát triển sản phẩm: Các lỗi phát hiện càng muộn thì chi phí sửa chữa càng cao. Chẳng hạn, đối với các lỗi được phát hiện trong giai đoạn thiết kế thì có thể chỉ mất một ít thời gian để điều chỉnh, nếu lỗi đó không được phát hiện và chuyển sang giai đoạn sản xuất thì chi phí sẽ tăng lên nhiều, và cứ như thế, nếu đến khi sản phẩm đã được phát hành mà lại xuất hiện lỗi thì chi phí sẽ cực kỳ lớn. Ngoài những ví dụ như đã được nhắc đến ở phần đầu bài viết, hãy thử tưởng tượng một hệ thống lớn như của Facebook hoặc Google mà bị dừng hoạt động trong dăm phút thì thiệt hại sẽ lên đến chừng nào – chắc chắc sẽ là một con số vô cùng khổng lồ. Do nguyên tắc này, chúng ta luôn cố gắng để các hoạt động kiểm thử được diễn ra càng sớm càng tốt.

Trước đây, kiểm thử gần như là công đoạn sau



Hình 2: Lỗi phát hiện càng muộn thì chi phí sửa chữa càng cao

cùng của quá trình phát triển. Giờ đây, chúng ta muốn hoạt động này được diễn ra sớm hơn, song song hoặc thậm chí là trước cả việc viết mã. Cũng bởi vì thế, một số thao tác kiểm thử đã được chuyển sang cho lập trình viên, thay vì gán riêng cho các tester như trước. Ngày nay, việc lập trình viên thực hiện kiểm thử đơn vị (unit testing) hoặc kiểm thử tích hợp (integration testing) đã trở nên phổ biến.

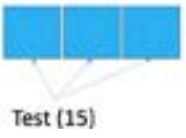
Kiểm thử là công việc lặp đi lặp lại

Giả sử quá trình phát triển các tính năng của một phần mềm và kiểm thử nó được diễn ra như minh họa sau đây.

Bước 1: Chức năng đầu tiên, với 5 kịch bản kiểm thử

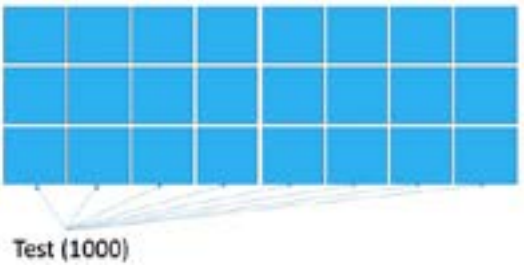
Bước 2: Thêm một tính năng nữa, và chúng ta cần thực hiện 10 kịch bản kiểm thử, bao gồm cả 5 kịch bản kiểm thử của chức năng trước đó, chứ không chỉ là các kịch bản của chức năng mới.

Bước 3: Thêm một tính năng nữa, và chúng ta cần thực hiện 15 kịch bản kiểm thử.



...sau một thời gian, chức năng càng được thêm nhiều vào, chúng ta cần thực hiện 100 kịch bản kiểm thử mỗi khi phát triển một tính năng mới (hoặc là chỉnh sửa một tính năng nào đó)

...và con số cứ thế tăng lên đến hàng nghìn kịch bản kiểm thử



Với một quá trình như thế, điều gì sẽ diễn ra nếu chúng ta phải thực hiện kiểm thử một cách thủ công?

Thứ nhất, chúng ta dễ dàng nhận thấy rằng đây là các thao tác gây hao tốn rất nhiều công sức. Sau khi chỉnh sửa hoặc thêm một chức năng bất kỳ, chúng ta lại phải kiểm thử hàng loạt các kịch bản đã làm trước đó.

Thứ hai, việc phát hành phần mềm sẽ bị kéo dài rất lâu chỉ bởi vì những thao tác kiểm thử này rất mất thời gian. Chúng ta sẽ khó để có thể phát hành phần mềm theo tần suất tuần, ngày hoặc thậm chí là giờ được.

Do vậy, tự động hoá hoạt động kiểm thử là một giải pháp giúp chúng ta rút ngắn được thời gian kiểm thử và giảm thiểu được chi phí nỗ lực rất nhiều so với trước kia.

Hãy thử tính toán, một thao tác kiểm thử thủ công mất vài chục phút thì bây giờ có thể được thực hiện với thời lượng chục giây. Một chuỗi kiểm thử thủ công mất vài giờ thì bây giờ có thể được thực hiện với thời lượng dăm phút. Hơn nữa, sức máy thì không biết mệt, trong khi sức người thì rất không ổn định. Như vậy, với hàng nghìn, thậm chí là chục nghìn kịch bản kiểm thử thì chúng ta đã tiết kiệm được bao nhiêu công sức và thời gian?

Vậy các lập trình viên và kiểm thử viên cần làm gì để bắt đầu với Kiểm thử tự động? Trước tiên, cần tìm hiểu để thay đổi tư duy về phát triển phần mềm theo hướng hiện đại và linh hoạt. Sau đó, tìm hiểu về một số phương pháp triển khai kiểm thử tự động. Cuối cùng, học cách sử dụng một số phần mềm, nền tảng hoặc công cụ hỗ trợ kiểm thử tự động mà chúng ta dễ dàng tìm thấy được, cho dù chúng ta đang sử dụng bất cứ một ngôn ngữ lập trình nào hoặc nền tảng nào.

08 CÁC NGUYÊN LÝ MÃ SẠCH: HÃY TRỞ THÀNH MỘT LẬP TRÌNH VIÊN TỐT HƠN

Rakesh Shekhawat
Author at SimpleProgrammer
Phan Văn Luân dịch

Khi nhắc đến mã sạch, hay tái cấu trúc mã nguồn bạn có từng suy nghĩ:

> "Mã của tôi đang hoạt động tốt, trang web tôi xây dựng trông rất tuyệt và khách hàng của tôi rất vui. Vậy tại sao tôi vẫn quan tâm đến việc viết mã sạch?"

Nếu điều này giống suy nghĩ của bạn ngay lúc này, thì hãy đọc tiếp.

Cách đây ít lâu, tôi đang thảo luận với một trong những người bạn của mình, Kabir. Kabir là một lập trình viên có kinh nghiệm. Anh ấy đang làm việc trong một dự án phức tạp, và anh ấy đang thảo luận một vấn đề với tôi. Khi tôi yêu cầu xem mã nguồn cho vấn đề đó, anh ấy nói, có vẻ tự hào, "Tôi đã xây dựng dự án này nên chúng tôi là những người duy nhất có thể hiểu mã nguồn của nó."

Tôi đã khá kinh hoàng. Tôi hỏi anh ta có phải anh ta cố tình viết mã bẩn không.

"Khách hàng đã không cho tôi đủ thời gian," bạn tôi nói với tôi. "Họ luôn vội vàng và thúc giục việc bàn giao sản phẩm, vì vậy tôi không có thời gian để nghĩ về việc viết mã sạch".

Đây hầu như luôn là lý do tôi nghe thấy khi hỏi về mã bẩn. Một số lập trình viên viết mã bẩn bởi vì họ dự định phát hành phiên bản đầu tiên và sau đó làm việc để làm cho nó sạch sẽ. Nhưng thường, điều đó sẽ không bao giờ xảy ra; không có khách hàng nào cho bạn thời gian để làm sạch mã. Khi phiên bản đầu tiên được phát hành, họ sẽ đẩy bạn lên phiên bản thứ hai. Vì vậy, hãy tạo thói quen viết mã sạch nhất có thể từ dòng mã đầu tiên.

Tôi luôn biết rằng việc sử dụng các nguyên tắc mã sạch mang lại nhiều lợi ích và bài viết này sẽ cho bạn biết lý do tại sao.

Công việc của người quản lý dự án, trưởng phòng kinh doanh hoặc khách hàng là hoàn thành dự án trong thời gian tối thiểu để họ có thể kiểm soát chi phí của dự án. Nhưng sản xuất chất lượng, mã sạch là nhiệm vụ của bạn với tư cách là lập trình viên.

Viết mã sạch không phải là một nhiệm vụ lớn hoặc tốn thời gian, nhưng biến nó thành thói quen của bạn và cam kết thực hiện nó, sẽ giúp bạn thăng tiến trong sự nghiệp và cải thiện khả năng quản lý thời gian của chính mình.

Mã sạch luôn trông giống như nó được viết bởi một người có tâm. Giống như Martin Fowler – một chuyên gia hàng đầu về phát triển phần mềm từng phát biểu:

"Bất kỳ kẻ ngốc nào cũng có thể viết mã mà máy tính có thể hiểu được. Các lập trình viên giỏi viết mã mà con người có thể hiểu được."

Có thể bạn đã đọc đến đây vì hai lý do: Thứ nhất, bạn là một lập trình viên. Thứ hai, bạn muốn trở thành một lập trình viên giỏi hơn. Tốt. Chúng tôi cần những lập trình viên giỏi hơn.

Hãy tiếp tục theo dõi để tìm hiểu tại sao mã sạch lại quan trọng và bạn sẽ trở thành một lập trình viên giỏi hơn.

Tại sao chúng ta nên cố gắng làm cho mã sạch?

Mã sạch có thể đọc được và dễ hiểu bởi mọi người, cho dù người đọc là tác giả của nó hay một lập trình viên mới.

Viết mã sạch là một tư duy cần thiết. Cần thực hành để viết mã có cấu trúc và rõ ràng, và bạn sẽ học cách làm điều đó theo thời gian. Nhưng bạn cần bắt đầu với tư duy viết theo cách này. Và bạn sẽ quen với việc xem xét và sửa đổi mã của mình để mã sạch nhất có thể.

Không ai là hoàn hảo, và bạn cũng vậy. Bạn sẽ luôn tìm thấy một số cơ hội để cải thiện hoặc tái cấu trúc lại mã khi bạn quay lại để xem lại mã của mình sau vài ngày hoặc vài tuần.

Vì vậy, hãy bắt đầu viết mã rõ ràng nhất có thể từ dòng mã đầu tiên để sau này bạn có thể làm việc nhiều hơn về cải thiện hiệu suất và logic.

Lợi ích của mã sạch

"Tại sao tôi nên quan tâm đến việc viết mã sạch?" bạn vẫn có thể tự hỏi mình.

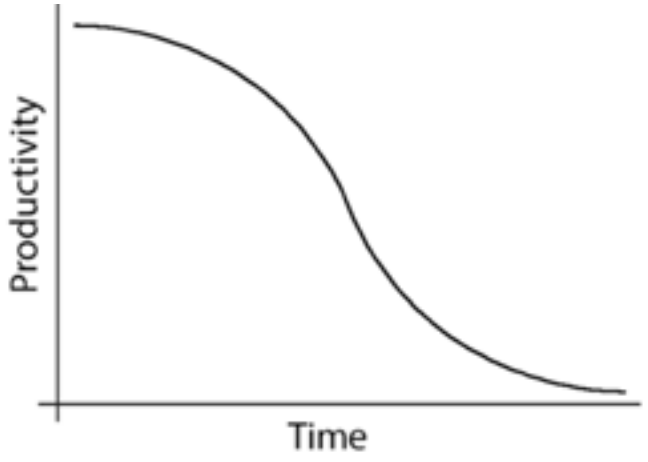
Có nhiều lý do để có được tư duy mã sạch mà tôi đã mô tả ở trên. Một số lý do quan trọng nhất là:

1. Sử dụng thời gian của bạn tốt hơn

Người hưởng lợi đầu tiên của mã sạch là lập trình viên.

Nếu bạn đang thực hiện một dự án trong nhiều tháng, rất dễ quên những điều bạn đã làm trong mã nguồn cũ, đặc biệt là khi khách hàng của bạn quay lại với các thay đổi. Các dòng mã rõ ràng giúp bạn thực hiện các thay đổi dễ dàng hơn.

Năng suất lập trình có mối quan hệ trái chiều với thời gian.



2. Tham gia dễ dàng hơn cho các thành viên mới

Sử dụng các nguyên tắc mã sạch sẽ giúp các lập trình viên mới dễ tiếp cận với mã nguồn đang phát triển hơn. Không cần tài liệu để hiểu mã nguồn; lập trình viên mới có thể trực tiếp tham gia vào dự án. Điều này cũng tiết kiệm thời gian đào tạo lập trình viên mới cũng như thời gian để lập trình viên mới điều chỉnh theo dự án.

3. Debug dễ dàng hơn

Cho dù bạn viết mã bẩn hay mã sạch, bug là không thể tránh khỏi. Nhưng mã sạch sẽ giúp bạn gỡ lỗi nhanh hơn, bất kể bạn có bao nhiêu kinh nghiệm hoặc chuyên môn. Và không có gì lạ khi đồng nghiệp hoặc người quản lý của bạn giúp bạn giải quyết vấn đề. Nếu bạn đã viết mã sạch, không vấn đề gì: Họ có thể nhảy vào và giúp bạn một cách dễ dàng hơn. Nhưng nếu người quản lý của bạn phải xử lý mã bẩn của bạn, thì bạn có thể sẽ giống như Kabir, bạn của tôi.

4. Bảo trì hiệu quả hơn

> "Tất nhiên mã xấu có thể được viết lại. Nhưng nó rất tốn kém. "
>
> *--- Robert C. Martin --*

Bảo trì không đề cập đến việc sửa lỗi. Khi phát triển bất kỳ dự án nào, nó sẽ cần các tính năng mới hoặc các thay đổi đối với các tính năng hiện có. Khi đó mã sạch sẽ giúp chúng ta dễ dàng bảo trì, hay nâng cấp tính năng mới hơn.

Ba điểm đầu tiên giải thích cách mã sạch có thể tiết kiệm thời gian của lập trình viên. Và, tiết kiệm một ít thời gian mỗi ngày sẽ có tác động kép đến thời gian hoàn thành và chi phí của phần mềm. Điều đó tốt cho công ty của bạn.

Vậy làm thế nào để chúng ta có thể cải thiện chất lượng mã của mình? Hãy cùng theo dõi tiếp nhé.

Cách viết mã sạch

> "Bạn nên đặt tên cho một biến bằng cách sử dụng cùng một cách mà bạn đặt tên cho đứa con đầu lòng."
>
> *-- Robert C. Martin, Clean Code: A Hand- book of Agile Software Craftsmanship --*

Một lập trình viên là một tác giả, nhưng họ có thể mắc sai lầm trong việc xác định đối tượng. Đối tượng của một lập trình viên là các lập trình viên khác, không phải máy tính. Nếu máy tính là đối tượng, thì

bạn có thể viết mã bằng ngôn ngữ máy.

Vì vậy, để dễ hiểu, bạn nên sử dụng danh pháp có ý nghĩa cho các biến, hàm và lớp. Và làm cho nó dễ đọc hơn bằng cách sử dụng lùi đầu dòng, phương pháp rút gọn và câu lệnh ngắn, nếu thích hợp:

• Sử dụng tên dễ phát âm cho các biến và phương thức. Không sử dụng chữ viết tắt trong tên biến và phương thức. Sử dụng tên biến ở dạng đầy đủ để có thể dễ dàng phát âm và mọi người có thể hiểu được.

Dirty code examples	Clean code examples
public $notiSms; public $addCmc;	public $notifySms public $addComment;

foreach ($people as $x) {
echo $x->name;
}
	foreach ($people as $person)
	{
	echo $person->name;
	}

$user->createUser(); createUser method does not make sense as it is written in user class.	$user->create(); Remove redundancy

• Sử dụng tên để thể hiện đúng mục đích. Mục đích của biến phải dễ hiểu đối với người đọc tên của biến. Viết tên như bạn sẽ nói.

Dirty code examples	Clean code examples
protected $d; // elapsed time in days	protected $elapsedTimeInDays; protected $daysSinceCreation; protected $daysSinceModification; protected $fileAgeInDays;

if ('paid' === $application->status) {
//process paid application
}
	if ($application->isPaid()) {
	//process paid application
	}

• Đừng đổi mới; hãy cứ đơn giản.Cho thấy sự đổi mới trong logic, không phải trong việc đặt tên biến hoặc phương thức. Có một cái tên đơn giản khiến mọi người dễ hiểu.

Dirty code example	Clean code example
$order->letItGo();	$order->delete();

• Hãy kiên định. Sử dụng một từ cho các chức năng tương tự. Không sử dụng "get" trong một lớp và "fetch" trong lớp khác.

• Đừng ngần ngại sử dụng các thuật ngữ kỹ thuật trong tên. Hãy tiếp tục, sử dụng thuật ngữ kỹ thuật. Đồng nghiệp của bạn sẽ hiểu nó. Ví dụ: "jobQueue" tốt hơn "job".

• Sử dụng một động từ làm từ đầu tiên trong phương thức và sử dụng một danh từ chỉ lớp. Sử dụng camelCase cho tên biến và hàm. Tên lớp bắt đầu bằng từ viết hoa.

Dirty code examples	Clean code examples
public function priceIncrement()	public function increasePrice()
Public $lengthValidateSubDomain	Public $validateLengthOfSubdomain;
class calculationIncentive	class Incentive

• Sử dụng các quy ước đặt tên nhất quán. Luôn sử dụng chữ hoa và phân tách các từ bằng dấu gạch dưới.

Dirty code example	Clean code example
define('APIKEY', '123456');	define('API_KEY', '123456');

• Làm cho các hàm rõ ràng. Giữ một hàm càng ngắn càng tốt. Độ dài lý tưởng của một function là tối đa 15 dòng. Đôi khi nó có thể kéo dài hơn, nhưng về mặt khái niệm thì mã phải sạch để hiểu.

• Tham số của hàm nên nhỏ hơn hoặc bằng ba. (Nếu các tham số lớn hơn ba, thì bạn phải suy nghĩ để cấu trúc lại hàm thành một lớp.)

• Một lớp nên làm một việc. Nếu nó dành cho người dùng, thì tất cả các phương pháp phải được viết hoàn toàn cho trải nghiệm người dùng.

• Hạn chế comment vô tội vạ. Nếu bạn phải thêm comment để giải thích mã của mình, điều đó có nghĩa là bạn cần phải cấu trúc lại mã của mình. Chỉ comment nếu điều đó là bắt buộc về mặt pháp lý hoặc nếu bạn cần ghi chú về tương lai hoặc lịch sử của chương trình.

Dirty code example	Clean code example
// Check to see if the employee is eligible for full benefits if ($employee->flags && self::HOURLY_FLAG && $employee->age > 65)	if ($employee->isEligibleForFullBenefits())

• Sử dụng Git đánh version cho ứng dụng. Đôi khi, các tính năng thay đổi và các phương thức cần được viết lại. Thông thường, chúng tôi nhận xét mã cũ vì sợ rằng khách hàng sẽ thay đổi và yêu cầu phiên bản cũ hơn. Nhưng nếu bạn sử dụng hệ thống kiểm soát phiên bản Git, hệ thống này sẽ lưu trữ tất cả các phiên bản, vì vậy bạn không cần phải giữ mã bẩn. Xóa nó và làm cho mã của bạn sạch sẽ hơn.

• Tránh làm việc với một mảng lớn. Tránh tạo một mảng cho một tập dữ liệu lớn; thay vào đó, hãy sử dụng một lớp. Điều đó làm cho nó dễ đọc hơn, chưa kể rằng nó tạo ra một sự an toàn bổ sung cho ứng

dụng của bạn.

• Không lặp lại mã. Mỗi khi bạn viết một phương thức, hãy tự hỏi bản thân xem liệu điều gì đó tương tự đã được xây dựng chưa. Kiểm tra thư viện mã hoặc tài liệu khác.

• Đừng "hardcode". Xác định hằng số hoặc sử dụng các biến thay vì fix cứng các giá trị. Việc sử dụng biến sẽ không chỉ làm cho nó có thể đọc được mà còn giúp bạn dễ dàng thay đổi nếu nó đang được sử dụng ở nhiều nơi.

• Làm cho câu lệnh có thể đọc được. Để làm cho câu lệnh có thể đọc được, hãy giữ dòng ngắn gọn để bạn không cần phải cuộn theo chiều ngang để đọc hết dòng.

Một số mẹo khác cho mã sạch hơn

• Tự xem lại mã của bạn. Xem lại mã của bạn một lần sau một thời gian. Tôi chắc chắn rằng bạn sẽ tìm thấy một cái gì đó mới để cải thiện mỗi khi bạn xem lại nó.

• Xem lại mã của bạn với đồng nghiệp. Xem lại mã của đồng nghiệp của bạn và yêu cầu họ xem lại mã của bạn. Đừng ngần ngại xem xét các đề xuất, các phản hồi của đồng nghiệp rất tốt với bạn

• Sử dụng coding convention. Nếu bạn đang viết cho PHP, hãy sử dụng hướng dẫn về kiểu viết mã của PSR-2.

• Sử dụng TDD, bạn nên sử dụng phương pháp TDD và viết các bài kiểm tra đơn vị để tăng chất lượng mã nguồn...

Tôi mong rằng qua bài viết này, bạn sẽ nghiệm lại nhiều điều và khi đọc lại mã nguồn cũ, tôi tin bạn sẽ thốt ra: "mình đã code cái quái gì vậy". Thực hành Clean Code không phải chuyện ngày một ngày hai, hãy luyện tập từng ngày với những thay đổi nhỏ nhất. "Những thay đổi nhỏ, sẽ làm nên thành công to".

09 CODE REFACTORING
TÁI CẤU TRÚC MÃ NGUỒN

Martin Fowler
Chief Scientist at ThoughtWorks

Tái cấu trúc là một quá trình cơ học, hình thức và trong nhiều trường hợp rất đơn giản để làm việc với mã của hệ thống đã tồn tại để chúng trở nên "tốt hơn". Khái niệm "tốt hơn" là một khái niệm mang tính chủ quan, và không đồng nghĩa với việc làm cho ứng dụng chạy nhanh hơn mà thường được hiểu là theo các kỹ thuật hướng đối tượng, tăng an toàn kiểu dữ liệu, dễ đọc, dễ bảo trì và mở rộng.

Hiệu quả của tái cấu trúc

Sản xuất phần mềm sẽ không hiệu quả nếu như bạn không thể theo kịp thay đổi của thế giới. Nếu như chúng ta chỉ sản xuất ra các phần mềm trong một vài ngày thì đơn giản hơn rất nhiều. Nhưng trong thế giới này chúng ta có rất nhiều đối thủ cạnh tranh. Nên nếu bạn không tái cấu trúc phần mềm của mình, khi đối thủ có một số tính năng hữu ích mới mà bạn không cập nhật thì sản phẩm của bạn nhanh chóng bị lạc hậu. Bởi thế là một lập trình viên bạn phải đón nhận và hành động một cách thích hợp với những thay đổi. Và khi thực hiện tái cấu trúc mã là bạn đang làm điều đó.

Cải thiện thiết kế

Nếu không áp dụng tái cấu trúc khi phát triển ứng dụng, thì thiết kế sẽ ngày càng tồi đi. Vì khi phát triển ứng dụng thì ta sẽ ưu tiên cho các mục tiêu ngắn hạn (đặc biệt khi áp dụng các quy trình phát triển linh hoạt), nên mã ngày càng mất đi cấu trúc. Một trong những tên của vấn đề này gọi là technical debt (nợ kỹ thuật). Khi xảy ra vấn đề thì rất khó để quản lý và dễ bị tổn thương. Thế nên việc áp dụng tái cấu trúc sẽ giúp cho mã giữ được thiết kế tốt hơn là ưu điểm quan trọng.

Mã dễ đọc hơn

Khi lập trình là chúng ta đang giao tiếp với máy tính để yêu cầu chúng làm điều mình muốn. Nhưng còn có người khác tham gia vào quá trình này là các lập trình viên khác hay chính chúng ta trong tương lai. Chúng ta biết khi lập trình thường sẽ có người phải đọc để kiểm tra xem có vấn đề với mã đó không hoặc để mở rộng hệ thống.

Nhưng có một vấn đề là khi làm việc, lập trình viên thường không nghĩ tới những người đó trong tương lai. Vậy thì trong trường hợp này tái cấu trúc đóng vai quan trọng là giúp cải thiện thiết kế của hệ thống, từ đó cũng giúp đọc mã dễ hơn.

Lợi ích hệ quả

Từ những lợi ích cơ bản ở trên ta có thêm các lợi ích khác: do hệ thống hiện thời có một thiết kế tốt hơn và mã dễ hiểu hơn, từ đó thì việc mở rộng hệ thống dễ dàng hơn, khó bị tổn thương hơn, nên tốc độ phát triển hệ thống luôn được duy trì; mã và thiết kế dễ đọc hơn, từ đó giúp tìm ra lỗi dễ dàng hơn; vì những mục tiêu ngắn hạn lập trình viên có thể chấp nhận một lỗ hổng nào đó về công nghệ hay thiết kế mà hiện thời không gây ảnh hưởng gì tới hệ thống, nhưng khi hệ thống lớn dần thì những lỗ hổng này được tích tụ và làm cho hệ thống dễ bị tổn thương, thế nên việc tái cấu trúc giúp nhanh chóng sửa những lỗ hổng này.

Thời điểm thực hiện tái cấu trúc

Khi thêm một chức năng mới

Khi thêm một chức năng mới, ta phải đọc lại mã để hiểu. Như vậy nếu lúc này ta thực hiện việc tái cấu trúc, mã sẽ dễ hiểu hơn, cộng với đó ta cũng dễ dàng hiểu mã hơn vì mình là người đã đọc và thực hiện việc tái cấu trúc. Một lý do khác là khi thực hiện việc tái cấu trúc vào thời điểm này thì thiết kế của hệ thống sẽ tốt hơn, từ đó việc mở rộng cũng dễ dàng hơn.

Khi sửa lỗi

Khi sửa lỗi ta cũng phải đọc mã, và như vậy việc tái cấu trúc làm mã dễ đọc hơn, có cấu trúc rõ ràng hơn từ đó dễ dàng phát hiện lỗi là điều cần thiết. Và bởi thế nếu bạn được gán là người sửa một lỗi nào đó thì bạn cũng thường được gán là người phải tái cấu trúc mã.

Khi rà soát mã

Nhiều tổ chức thực hiện việc rà soát mã (code review). Rà soát mã giúp cho các lập trình viên giỏi truyền lại cho các lập trình viên ít kinh nghiệm hơn, giúp cho mọi người viết mã rõ ràng hơn. Mã có thể là rất rõ ràng với tác giả, nhưng với người khác thì có thể không, bởi thế rà soát mã sẽ làm cho nhiều người đọc mã hơn. Có nhiều người đọc mã thì mã phải dễ đọc hơn và có nhiều ý tưởng hơn được trao đổi giữa các thành viên trong nhóm hơn. Lần đầu bạn đọc bạn bắt đầu hiểu mã. Lần tiếp theo bạn sẽ có nhiều ý tưởng hơn để tái cấu trúc mã, từ đó bạn có thể thực hiện việc tái cấu trúc.

Các "mã bẩn" thường gặp

Khái niệm mã bẩn (code smell) là mã có thể sinh vấn đề một cách lâu dài, sẽ giúp ta phát hiện mã cần phải tái cấu trúc.

Mã lặp

Là những đoạn mã xuất hiện nhiều hơn một lần trong một hoặc nhiều ứng dụng của một chủ thể. Đó là hệ quả của các hành động: sao chép mã; các chức năng tương tự được viết bởi các lập trình viên khác nhau. Hệ quả là mã trở nên dài hơn, khó hiểu hơn và khó bảo trì hơn.

Hàm dài

Hàm dài là quá phức tạp, có lượng mã lớn. Nên khó để hiểu, triển khai, bảo trì và tái sử dụng. Nên việc tách thành các hàm nhỏ hơn là điều cần thiết.

Lớp lớn

Lớp lớn là lớp chứa quá nhiều thuộc tính và chức năng, thường là của nhiều lớp khác.

Hàm có nhiều tham số đầu vào

Khi một hàm có nhiều tham số đầu vào sẽ gây khó khăn để đọc, dùng và thay đổi. Với các ngôn ngữ lập trình hướng đối tượng ta có thể nhóm các tham số có liên quan vào một đối tượng để giảm số lượng tham số đầu vào.

Tính năng không phải của lớp

Là hiện tượng một phương thức không nên thuộc một lớp, nhưng do phương thức muốn sử dụng các dữ liệu của lớp đó nên lập trình viên đã gán phương thức cho lớp. Ta cần trả phương thức đó về đúng đối tượng.

Lớp có quan hệ quá gần gũi

Đó là hiện tượng hai lớp có thể truy xuất vào các thuộc tính riêng tư của nhau một cách không cần thiết. Điều đó dẫn đến là chính các lớp đó hay lớp con của chúng có thể thay đổi các thuộc tính của lớp con lại một cách "vô thức".

Lớp quá nhỏ

Việc tạo, bảo trì và hiểu một lớp tốn tài nguyên. Vậy nếu lớp đó quá nhỏ thì ta nên xóa bỏ lớp đó đi.

Lệnh switch

Vấn đề của lệnh switch chủ yếu là vấn đề lặp mã. Bạn thường gặp những lệnh switch để phân bổ các chức năng ở nhiều nơi khác nhau trong cùng một ứng dụng. Và mỗi khi bạn thêm một tính năng mới, bạn phải tìm ở tất cả các lệnh này để thay đổi.

Từ chối kế thừa

Điều này xảy ra khi một lớp được thừa kế dữ liệu cũng như các tính năng của lớp cha, nhưng lại không cần phải dùng tới chúng.

Định danh quá dài hoặc quá ngắn

Các định danh cần mô tả đủ ý nghĩa để mã dễ đọc hơn và tránh gây hiểu nhầm. Bởi thế các định danh quá ngắn thường không mô tả hết ý nghĩa và gây ra nhầm lẫn. Nhưng các định danh quá dài cũng có vấn đề tương tự.

Dùng quá nhiều giá trị

Nếu trong mã có nhiều giá trị thì khi cần phải thay đổi các giá trị đó - vì một lý do nào đó - thì bạn cần phải thay đổi ở nhiều nơi. Không chỉ thế mà bạn còn không nhớ những giá trị đó có ý nghĩa gì, nên làm cho mã trở nên khó đọc hơn. Trong trường hợp này bạn có thể thay bằng cách đặt tên cho các hằng.

10 CODERETREAT: CÁC PHIÊN LUYỆN TẬP SÂU

Coderetreat là sự kiện diễn ra trong một ngày, các thành viên tham gia vào hoạt động thực hành chuyên sâu, tập trung vào những kỹ năng căn bản của phát triển và thiết kế phần mềm.

Có rất nhiều thảo luận xoay quanh việc chúng ta đã không thực sự thực hành một cách chú tâm. Trong những lĩnh vực có yếu tố sáng tạo, người ta cần rèn luyện thường xuyên. Nhưng hầu hết các lập trình viên lại được rèn luyện chủ yếu từ thực tế công việc. Vì vậy họ đã không học tập tốt theo cách thường thấy với khả năng vốn có như khi họ rời xa những áp lực công việc. Ở Coderetreat các nhà phát triển tập trung vào những nguyên lý căn bản của thiết kế hướng đối tượng, giúp họ cải thiện kỹ năng lập trình để giảm thiểu chi phí thay đổi trong tương lai. Trong Coderetreat mọi người tham gia có thể dùng ngôn ngữ, môi trường phát triển, khung làm việc tùy ý.

Ý tưởng về Coderetreat khởi xướng khoảng năm 2009 tại hội thảo CodeMash ở Sandusky Ohio. Code Retreat lần đầu tiên được tổ chức tại Ann Arbor, Michigan. Với một vài nỗ lực, giờ đây Coderetreat đã có định dạng như chúng ta thấy hôm nay: đơn giản, hiệu quả và có tính lan tỏa. Giờ đây hoạt động này đã trở thành thông lệ với nhiều người vào mỗi dịp cuối tuần.

Ảnh: Các thành viên tham gia một phiên Coderetreat tại Hà Nội.

CHUẨN BỊ CHO MỘT SỰ KIỆN CODERETREAT

Tạo sự kiện và quảng bá

Mặc dù điều then chốt là thứ mà người tham dự mang về sau sự kiện. Nhưng số lượng và đặc biệt là sự đa dạng của các thành viên tham gia là yếu tố quan trọng. Số lượng và sự đa dạng của các thành viên tham gia quyết định rất lớn tới chất lượng trải nghiệm. Do đó việc tạo sự kiện và quảng bá rất quan trọng. Nếu bạn định tổ chức chỉ cho công ty mình thì cũng nên mời người ở các doanh nghiệp khác, bạn bè, v.v.

Trong thư mời bạn nên thông báo rõ những thứ mà người tham gia phải chuẩn bị như máy tính, ngôn ngữ, cài đặt môi trường phát triển, khung kiểm thử, server mà họ có thể dùng trong Coderetreat.

Địa điểm và hậu cần

Tùy vào số lượng người tham gia mà cần một địa điểm rộng hay hẹp, nhưng địa điểm cần đảm bảo việc di chuyển dễ dàng cho các các thành viên.

Bạn cần chuẩn bị đủ ổ cắm điện cho lượng người tương ứng. Tuy nhiên Coderetreat thường áp dụng lập trình cặp, nên bạn cần chuẩn bị số ổ cắm bằng một nửa số người tham dự.

Phòng cần có một nơi để thực hiện hoạt động cải tiến và chia sẻ. Các ý kiến sẽ được cố định lên vị trí đó. Đó có thể là một bảng dính được bằng nam châm. Tuy nhiên bạn cũng có thể làm là dán một tờ giấy lớn lên tường và dùng băng dính hai mặt hoặc sticky note.

Các quy tắc cũng nên được in và dán ở vị trí mà mọi người tham gia luôn luôn nhìn thấy chúng.

Bạn cũng cần chuẩn bị giấy, bút và sticky cho tất cả mọi người để thực hiện quá trình cải tiến.

Thức ăn, đồ uống

Trong quá trình làm việc các thành viên cần có đồ uống. Tùy vào sở thích bạn có thể có nước lọc, cafe (dễ nhất là cafe hòa tan và nước nóng), trà, nước ngọt. Nếu có điều kiện bạn nên có thêm chút bánh ngọt và hoa quả.

Đồ uống và thức ăn nên được để ở nơi tương đối tách rời khỏi nơi diễn ra Coderetreat bởi mùi của chúng có thể gây phân tán hoặc ảnh hưởng tới mọi người.

Bữa trưa đối với hoạt động Coderetreat là rất quan trọng. Sau buổi sáng làm việc mọi người sẽ đói, nên để cho hoạt động buổi chiều hiệu quả thì bữa trưa không nên là bữa ăn nhẹ (ví dụ pizza). Chú ý tới sở thích của mọi người, có thể một số người phải ăn kiêng. Nên chọn đồ ăn ở một nhà hàng địa phương. Bạn nên có một nhà tài trợ cho đồ ăn riêng.

Người hỗ trợ (Facilitator)

Coderetreat là hoạt động thực hành có chủ ý, nên người tham gia cần có người hướng dẫn (facilitator) các hoạt động luyện tập. Theo lời khuyên thì để trở thành người hỗ trợ bạn nên tham gia ít nhất 2 lần codetreat.

CODERETREAT CÓ CẤU TRÚC NHƯ THẾ NÀO?

Đăng ký, ăn sáng, cafe

Phần này có thể kéo dài từ 30 phút tới 45 phút Thời gian này có mục đích để mọi người giao lưu với nhau, nghỉ ngơi, chờ những người tới muộn, và chuẩn bị những thứ cần thiết như chuẩn bị máy tính.

Giới thiệu Coderetreat và thử thách

- Giới thiệu về lịch sử của Coderetreat
- Giới những giá trị định hướng và định dạng của Coderetreat.
- Giới thiệu về bài toán Conway's Game of Life

Các phiên làm việc

Phiên bản của Global Day of Coderetreat có 6 phiên làm việc, tuy nhiên bạn có thể điều chỉnh. Thực tế chúng tôi đã tổ chức chỉ có 5 phiên như ở Summer Coderetreat 2013 và cả Global Day of Coderetreat 2012. Mỗi phiên gồm 2 phần: 45 phút lập trình và 15 phút cho cải tiến và giải lao.

Ở mỗi phiên lập trình, các cặp sẽ tự chọn cho mình quy tắc muốn áp dụng. Ở phiên đầu tiên hầu hết mọi người không nên chọn quy tắc nào để làm quen với bài toán. Nhưng ở các phiên sau bạn có thể gợi ý quy tắc cho các nhóm. Sau đó họ sẽ lập trình tuân thủ quy tắc đó. Trong khi các thành viên lập trình, người hỗ trợ nên đi quanh phòng và quan sát cách các nhóm cộng tác, cách thực hiện các quy tắc, cách mã nguồn được viết và hãy thỏa mái hỏi họ về điều đang làm, khi hợp lý hãy hỏi họ về cách tiếp cận như:

- Tại sao bạn lại đặt tên hàm là getLiveCells?
- Tại sao bạn không viết kiểm thử?
- Tại sao bạn lại tái cấu trúc khi kiểm thử thất bại?

Khi bạn thấy ai đó đang có vấn đề về việc làm việc cặp, người hỗ trợ hãy cố gắng giúp họ cặp với người có thể mang lại cho họ một trải nghiệm tốt hơn.

Sau mỗi phiên làm việc bạn phải đảm bảo là tất cả mọi người đã XÓA MÃ NGUỒN.

Sau đó sẽ đến phần cải tiến và giải lao. Ở phần này mỗi nhóm nên viết ra một số ý mà họ nghĩ về phiên làm việc vừa qua. Bạn cũng có thể gợi ý các câu hỏi như:

- Bạn đã học được gì?
- Bạn thích gì?
- Bạn không thích gì?
- Có gì tốt?
- Có gì không tốt?

Do thời gian có hạn, bạn nên chia thành các nhóm và chọn số câu hỏi hạn chế để mọi người viết vào giấy, sau đó mọi người trong nhóm chia sẻ với nhau và cả nhóm chọn ra một số lượng ý nhất định (3-5) để chia sẻ với toàn bộ Coderetreat ở bảng.

Tổng kết

Mỗi người sẽ trả lời ba câu hỏi ra giấy:

- Bạn học được gì hôm nay?
- Điều gì làm bạn bất ngờ?
- Bạn sẽ làm gì khác trong công việc?

Sau mỗi người sẽ trình bày nhanh với toàn bộ Coderetreat về những câu trả lời của mình. Bạn nên chú ý tới lượng người mình có để căn thời gian cho phù hợp và dán lên bảng.

Nếu Coderetreat của bạn lớn, thì nên chia thành các nhóm (khoảng 10 người). Mọi người tự trả lời các câu hỏi đó, sau đó chia sẻ với nhóm và đại diện của nhóm sẽ chia sẻ với toàn bộ Coderetreat.

Lấy ý kiến đánh giá như thế nào?

Sau cùng bạn nên lấy ý kiến đánh giá để rút kinh nghiệm. Bạn nên in mẫu lấy ý kiến để mỗi người điền giúp bạn hay trực tuyến để dễ dàng xử lý. Chúng tôi tạo ra một biểu mẫu của Google và gửi cho mọi người.

MỘT SỐ QUY TẮC Ở CODERETREAT

Người tham gia nên chọn một hoặc nhiều trong những quy tắc để áp dụng trong mỗi phiên làm việc. Facilitator nên quan sát để hướng dẫn người tham gia thực hiện đúng quy tắc. Ở phần mở đầu khi giới thiệu về các quy tắc người hỗ trợ nên giới thiệu về các quy tắc này. Trong các phiên làm việc, người hỗ trợ cũng nên giải thích cho các thành viên.

Câu hỏi đặt ra là những quy tắc này để làm gì? Tại sao chúng ta nên tự ràng buộc mình? Chúng ta có rất nhiều cách và công cụ để giải quyết một vấn đề. Khi đặt ra một giới hạn tức là đang hạn chế các công cụ và cách thức để giải quyết cùng vấn đề, chúng ta sẽ phải vận dụng tốt hơn những kỹ năng, công cụ và cách thức khác. Từ đó chúng ta sẽ rèn luyện mình tốt hơn. Đó là một thực hành có chủ ý.

(Theo kienthuclaptrinh.vn, codegym.vn và hanoiscrum.net)

11 KHÔNG CẢM XÚC, LẬP TRÌNH VIÊN KHÓ TIẾN XA

Phỏng vấn anh Nguyễn Thanh Tùng
CEO - Hamsa Technologies

"Anh đang ở tuổi "tam thập nhi lập", hay nói cách khác là vẫn còn đang ương. Cũng chính vì còn ương, nên anh luôn tự nhắc nhở mình phải luôn rèn luyện, luôn phát triển bản thân, để một ngày nào đó sẽ… chín."

Đó là những lời đầu tiên anh Nguyễn Thanh Tùng bộc bạch trong buổi trò chuyện cùng chúng tôi.

Anh Nguyễn Thanh Tùng hiện đang là CEO Hamsa Technologies, một trong những công ty hàng đầu Việt Nam trong lĩnh vực cung cấp các giải pháp thương mại điện tử cho thị trường toàn cầu. Trước đây, anh vốn là cử nhân Ngôn ngữ học. Và có lẽ vì cái duyên "ngôn ngữ" đó nên anh bén tiếp duyên sang các ngôn ngữ lập trình từ HTML, CSS, Javascript đến PHP, C#, Java,… rồi tới các "ngôn ngữ" của tiếp thị, bán hàng, tài chính, kế toán,… cho đến ngôn ngữ của nhân sự.

Một trong những sở thích đặc biệt của anh là làm việc với con người. Chính bởi cái sở thích có phần kỳ quặc đó, mà phần lớn thời gian hàng ngày anh dành để cộng tác, hỗ trợ các nhóm trong công ty, để họ biết xác định được mục tiêu và làm thế nào để đạt được mục tiêu. Có lẽ vì vậy, trong buổi tâm sự chuyện nghề IT cùng CodeGym, bên cạnh những lời khuyên hữu ích cho các bạn trẻ, anh Tùng cũng nhấn mạnh vào vấn đề trên khi cho rằng: "Muốn đi xa, lập trình viên phải học cách làm việc với con người."

Sau đây là nội dung cuộc trao đổi ngắn của chúng tôi với anh Nguyễn Thanh Tùng.

Hỏi: Dịch Covdid-19 đã tác động thế nào đến thị trường CNTT Việt Nam?

Dịch Covid-19 có ảnh hưởng vừa tích cực, vừa tiêu cực tới thị trường công nghệ thông tin. Theo quan sát của anh, nhìn chung thì ảnh hưởng tích cực nhiều hơn là tiêu cực, đó là tín hiệu tốt cho ngành.

Ở góc độ dự án, nhiều dự án trọng điểm trong và ngoài nước bị tạm dừng hoặc dừng hẳn, điều này ảnh hưởng không nhỏ tới các doanh nghiệp. Mặt khác, các dòng dự án, sản phẩm phục vụ chuyển đổi số liên quan đến nhu cầu thiết thực như các giải pháp thương mại điện tử, giải pháp họp hành, giải pháp đào tạo, giải pháp cộng tác nhóm,… lại có cơ hội phát triển rất mạnh.

Ở góc độ thị trường lao động, do tác động tích cực của các dự án, sản phẩm, các doanh nghiệp gia tăng đáng kể trong nhu cầu tuyển dụng. Các bạn có kinh nghiệm cũng có xu hướng chọn phương án an toàn hơn trong lúc tình hình dịch khó lường.

Động thái thị trường và tâm lý này đã làm gia tăng độ thiếu hụt trong thị trường nhân lực, vốn đã thiếu hụt, nay càng trầm trọng hơn. Nhu cầu nhân sự tăng, cũng tạo điều kiện cho các bạn Intern và Fresher có nhiều "cửa" trường đời hơn, thậm chí ngay khi vừa mới chân ướt chân ráo bước qua cánh cửa trường học, mà không cần qua bất kì môi trường đào tạo chính quy nào.

Đây là thời điểm thuận lợi cho các bạn các ngành chịu ảnh hưởng của Covid có cơ hội chuyển dịch sang lĩnh vực công nghệ thông tin, vốn không chỉ tốt thời dịch, mà còn là xu hướng phát triển lâu dài và bền vững.

Hỏi: Những bạn trẻ trái ngành, trái nghề sẽ đối mặt với những cơ hội, thách thức nào khi theo ngành CNTT?

Như anh vừa nói ở trên, hiện tại có rất nhiều cơ hội cho các bạn, vì thị trường công nghệ thông tin được dự báo là vẫn tiếp tục thiếu hàng trăm nghìn nhân sự trong những năm tới.

Chỉ cần bạn làm được việc thôi, thì chắc chắn bạn đã có trong tay một nghề đáng mơ ước rồi.

Tuy nhiên, cơ hội cũng đi liền với thách thức.

Như nhiều người vẫn hay nói "dân công nghệ mà", công nghệ là phải thích ứng rất nhanh với các ngôn ngữ mới, công nghệ mới. Và đây là điểm bất lợi với những bạn trái ngành, vì nền tảng công nghệ của các bạn chưa đủ chắc.

Có thể các bạn sẽ học rất nhanh để làm được một việc, nhưng khi yêu cầu dự án thay đổi, thì các bạn sẽ rất vất vả. Nếu không kịp thích ứng, thì công việc đáng mơ ước đang sở hữu trong tay sẽ bị dậm chân tại chỗ, và không loại trừ khả năng bị đào thải trong khoảng 10 đến 15 năm nữa.

Hỏi: Anh có định hướng hay lời khuyên nào dành cho các bạn trẻ đang theo ngành CNTT không ạ?

Xã hội ngày nay có rất nhiều biến động, nên việc đầu tiên là cần chuẩn bị cho mình một tinh thần "học tập suốt đời" nếu không muốn bị loại ra khỏi cuộc chơi. Nếu bạn luôn cầu thị, luôn học hỏi, luôn đổi mới thì không những không bị loại ra khỏi cuộc chơi, mà bạn hoàn toàn có cơ hội để làm chủ chính cuộc chơi đó, góp phần tích cực vào sự nghiệp phát triển chung của quốc gia.

Ngoại ngữ cũng là một phần quan trọng của nghề lập trình. Tối thiểu là tiếng Anh. Ngoài ra, bạn có thể xem xét thêm tiếng Nhật, tiếng Hàn, tiếng Trung, tiếng Pháp,…

Ngoài học hỏi về chuyên môn, một cách đi đường dài nữa không thể thiếu, đó là học cách làm người. Có rất nhiều cơ hội hấp dẫn trong lĩnh vực CNTT. Nhưng càng phát triển lên, bạn sẽ càng phải làm việc nhiều với con người. Trong khi đó, chúng ta có thể đã có một thời gian dài làm việc với… máy, rất có thể, chúng ta vô tình quên đi những thứ căn bản như giao tiếp, cảm xúc, tôn trọng, trung thực,… Thiếu đi những thứ đó, chúng ta khó có thể đi xa được.

ĐÔI NÉT VỀ HAMSA TECHNOLOGIES

Được thành lập từ năm 2014, đến nay, Hamsa đã mang giải pháp của mình đến hơn 1000 khách hàng, chủ yếu là các nước Âu Mỹ và châu Á Thái Bình Dương. Để làm được điều đó, Hamsa không ngừng nuôi dưỡng và theo đuổi các giá trị, mà sau nhiều năm, nó đã trở thành giá trị cốt lõi của công ty. Đó là:

1. Tôn trọng sự khác biệt
2. Hướng tới sự minh bạch
3. Mưu cầu sự học hỏi và trưởng thành
4. Nghiêm túc và cẩn trọng trong mọi việc
5. Chào đón và dẫn dắt sự thay đổi
6. Thiết lập và duy trì các tiêu chuẩn cao trong công việc
7. Định hướng kết quả
8. Quan tâm đến đồng nghiệp, khách hàng và đối tác.

Hamsa luôn chào đón các bạn cùng chung giá trị và niềm tin, để đồng hành và lan toả những nét văn hoá này tới những người xung quanh, kiến tạo một cộng đồng hạnh phúc và thịnh vượng.

12 CODER: MỘT NGHỀ CHÂN CHÍNH, MỘT NGHỀ CẦN VINH DANH

GIỚI THIỆU SÁCH "THE CLEAN CODER: A CODE OF CONDUCT FOR PROFESSIONAL PROGRAMMERS"

Dương Trọng Tấn
CEO Agilead Global

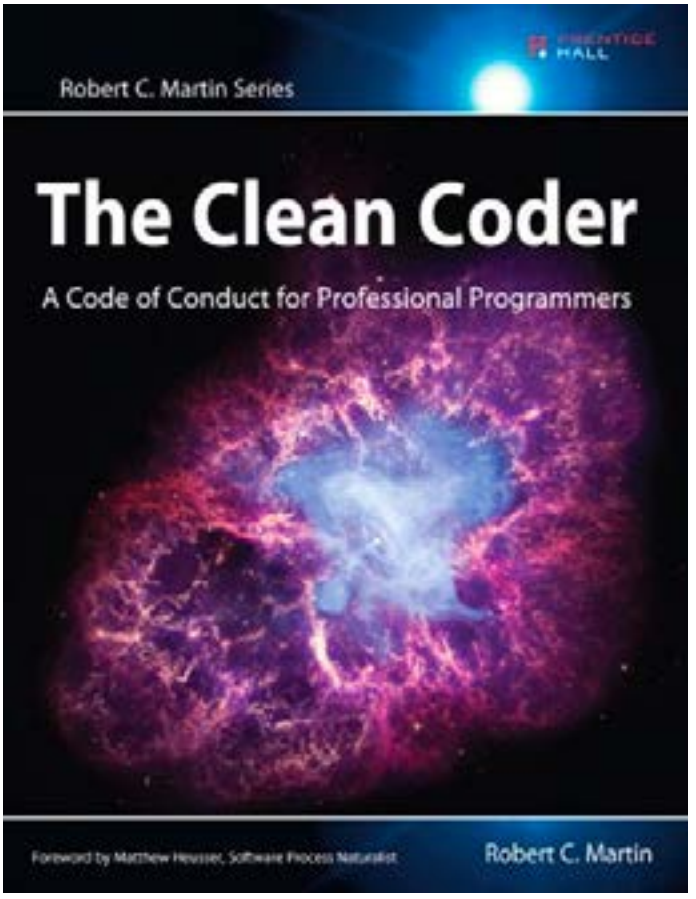
Ở Việt Nam, chưa ai viết sách về Lập trình viên (gọi bình dân là coder) và ca ngợi nghề lập trình thì phải? Có chăng chỉ là một vài trang ca ngợi qua quýt, vài bài ca cẩm hời hợt, cùng lắm là một bộ phim nhàn nhạt có tên "*Lập trình trái tim*" vốn chả dính dáng gì tới nghề Lập trình. Dĩ nhiên ở Mĩ thì khác hẳn: phim làm về các lập trình viên rất rất thú (như phim về Bill Gates hay Mark Zuckerberg chẳng hạn), sách viết về lập trình viên, lập trình viên tự viết, hay viết cho lập trình viên thì rất nhiều. Thử lấy ví dụ về vụ FlappyBird ầm ĩ gần đây thì thấy sự khác nhau rất rõ: trong khi báo chí trong nước chỉ biết xào đi xáo lại vài mẩu tin cũ từ tạp chí nước ngoài, thì hàng loạt hãng tin có tiếng như CNN, Forbes đưa tin, phỏng vấn trực tiếp; thậm chí tờ báo rất oách Rolling Stones thuê hẳn một cây viết có tiếng David Kushner viết thật chi tết về một chân dung coder thứ thiệt đầy đam mê và sức trẻ. Một coder học tập, lớn lên và thành danh ở chính quê hương Hà Đông của mình, nhưng lại được vinh danh ở những chỗ trang trọng nhất trên những diễn đàn nổi tiếng nhất của miền đất hứa của dân Lập trình.

Cho đến giờ, coder ở Việt Nam phần nhiều chỉ coi mình là những "*cu-li*" của thời đại số, coi nghề lập trình chỉ như chỗ ghé chân thoảng qua trong sự nghiệp. Đấy quả là một não trạng đầy u ám và không nên duy trì lâu trên một quốc gia muốn lấy CNTT làm "bàn đạp" để tiến ra thế giới. Có thể những Nguyễn Hà Đông sẽ giúp cải thiện phần nào tình hình, nhưng những cánh én ấy chỉ mang lại những cảm hứng, chứ để định hình cả một nghề nghiệp tử tế thì còn nhiều việc phải làm và phải chờ một thời gian nữa

Chú Bob ("Uncle Bob", tên "đường phố" của Robert C. Martin, một tay tổ trong giới lập trình, tác giả của hàng loạt tập sách thuộc loại gối đầu giường cho coder, đồng tác giả của Tuyên ngôn Agile (*Manifesto for Agile Software Development*) trứ danh – một văn bản định hình lại văn hóa lập trình hơn một thập kỉ qua) là người nhiệt thành đến cực đoan trong việc thúc đẩy một quan điểm "nâng tầm" nghề lập trình, đòi hỏi cả coder lẫn những người khác phải nhìn nhận lập trình như một nghề chuyên nghiệp đáng

trân trọng, và phải đầu tư nghiêm túc, từ chuyên môn tới trách nhiệm và đạo đức. Cuốn sách mới nhất của Chú Bob, cuốn "*Clean Coder: A Code Of Conduct for Professional Programmers*" hiển hiện rất rõ như là một "kinh điển" mà một coder cần phải mang theo, từ lúc còn tập tọe viết "Hello World" cho tới lúc đã 10 năm kinh nghiệm lập trình. Với cuốn sách này, Chú Bob tiếp tục thể hiện mình là một lãnh đạo tư tưởng (thought leader) xứng đáng của giới viết mã.

Chương 1 bắt đầu như thể để nối liền tựa sách: Professionalism. Chú Bob bắt đầu thảo luận trước hết về đạo đức nghề nghiệp, những điều suy nghĩ nghiêm túc về trách nhiệm của một coder. Trong tiểu mục có lẽ là quan trọng nhất của chương, lấy tựa kiểu như một tín điều trong Kinh thánh "First, Do No Harm", Chú Bob chỉ rõ các quy tắc và yêu cầu cụ thể đối với một coder thứ thiệt, như: không để lại bug, viết code phải sạch mã đến mức QA không thể tìm thấy gì, coder phải biết rõ là code mình chạy tốt chứ không cần nhờ đến tester hay ai khác "kiểm thử" giúp. Đó là những quan điểm nhất quán được Chú Bob đặt như là một trong những nền tảng đạo đức quan trọng nhất của nghề viết mã. Hai chương tiếp theo tiếp tục làm rõ những tình huống cần nói Không và Có, như là cụ thể hóa các ranh giới đạo đức nghề nghiệp mà một coder cần phải vạch ra. Năm chương 4, 5, 6, 7, 8 bàn kĩ về các khía cạnh kĩ thuật mà một coder phải thành thục hằng ngày: viết mã, kiểm thử, đảm bảo chất lượng, và luyện tập nâng cao tay nghề liên tục. Thông điệp của những chương này hẳn là rất rõ: coder cần đảm bảo tay nghề luôn vững và tăng trưởng liên tục, trong khi luôn tạo ra những đoạn mã chất lượng cao và sạch lỗi (clean code), những phần mềm có chất lượng tự thân và không làm phiền QA. Chương 9 là một chương hết sức thú vị về quản lí thời gian, từ chuyện họp hành, tới những điều nhỏ nhặt nhưng gần gũi và không kém phần quan trọng như chuyện uống cà phê, chuyện tập trung khi làm việc hay nghỉ ngơi như thế nào. Nếu được góp ý với Chú Bob, chắc là tôi thích gọi chương này cùng với chương bàn về áp lực (Ch11: Pressure) là "Quản lí năng lượng" (managing energy) thay vì tách ra và gọi riêng như tác giả đã dùng. Những điều này cực kì quan trọng đối với một coder, để thoát ra khỏi cái định kiến về một giới thường hay ngồi xó văn phòng, đầu to mắt cận da bọc xương ít giao tiếp xã hội. Trong quan điểm của Chú Bob, coder hoàn toàn là những người bình thường như bao người khác. Sau một chương hơi kĩ về kĩ thuật ước tính (Estimation), các chương tiếp theo tác giả bàn quanh chuyện cộng tác và làm việc trong nhóm; có thể coi như những dẫn giải kĩ lưỡng về nguyên lí đã từng được viết cô đặc trong Tuyên



Bìa cuốn sách "The Clean Coder: A Code Of Conduct for Professional Programmers "

ngôn Agile: "*Individuals and interations over process and tools*", và "*Customer collaboration over contract negotiation*", sự khác biệt ở đây là tác giả chỉ rất rõ cái "How" cho những độc giả còn chưa quen Agile Software Developent (Phát triển Phần mềm Linh hoạt). Chú Bob kết thúc phần chính của quyển sách của mình với từ khóa quan trọng nhất cho một trào lưu hậu-Agile: Software Craftsmanship (Nghề Thủ công Phần mềm) với các chủ điểm ngắn gọn về kèm cặp (mentoring), học việc (apprenticeship) và một nghệ nhân phần mềm (craftsman). Phần phụ lục có khi lại là phần đắc địa với nhiều người nhất khi Chú Bob liệt kê đầy đủ "dao kéo" (tools) cho một coder thứ thiệt như ông đã nhắc tới trong suốt cả cuốn sách.

Được viết với một văn phong giản dị và dễ hiểu, những từ khóa không mấy hấp dẫn như "trách nhiệm", "đạo đức", "những điều răn" … được Chú Bob dẫn nhập một cách có lí có tình và thật nhiều hướng dẫn thi triển cụ thể. Cộng thêm những suy tư cá nhân của riêng một người có thâm niên gõ bàn phím được dàn đầy trong các trang sách, với các coder chân chính "*The Clean Coder*" xứng đáng là cuốn sách cần đọc lòng và ngẫm nghĩ thường xuyên.

Cảm ơn quý độc giả và hẹn gặp lại mọi người trong ấn phẩm tiếp theo với chủ đề:

Thế giới Front-end.

Mọi ý kiến góp ý xin vui lòng gửi về *devworld.magazine@gmail.com*

Các thành viên Ban biên tập:
Nguyễn Khắc Nhật
Vũ Thị Kiều Anh
Nguyễn Khánh Tùng
Nguyễn Bình Sơn
Nguyễn Hữu Anh Khoa
Lê Thị Châu
Trịnh Minh Thảo

Cảm ơn sự đóng góp của các tác giả trong ấn phẩm này:
Dương Trọng Tấn
Nguyễn Văn Hiển
Phan Văn Luân
David Green
Rakesh Shekhawat
Martin Fowler
và nhiều cá nhân khác

Bản quyền ảnh sử dụng trong ấn phẩm:
freepik.com

VOL.1 - 11.2021