Scrum và XP từ những Chiến hào

Chúng tôi đã Scrum như thế nào?

Henrik Kniberg

Viết lời tựa Jeff Sutherland, Mike Cohn

InfoQ *Loạt sách "Enterprise Software Development"*

BẢN TRỰC TUYẾN MIỄN PHÍ

(không được phép in ấn từ bản này)

Nếu bạn thích cuốn sách này, xin hãy hỗ trợ

tác giả và InfoQ bằng cách

đặt mua bản in của cuốn sách tại

http://www.lulu.com/content/899349 **(giá bán 22,95$)**

Cuốn sách được cung cấp tới bạn với sự trợ giúp của

Bản miễn phí được đăng tải trên trang web InfoQ.com, nếu bạn nhận được cuốn sách này từ bất cứ nguồn nào xin hãy hỗ trợ tác giả và nhà xuất bản bằng cách đăng ký với trang web InfoQ.com.

Cuốn sách được cung cấp tại:

http://infoq.com/minibooks/scrum-xp-from-the-trenches

Scrum và XP từ những Chiến hào

Chúng tôi đã Scrum như thế nào?

Tác giả: Henrik Kniberg

Dịch: Nguyễn Việt Khoa (TN), Bùi Thanh Hải, Vũ Thị Hà Thảo, Phạm Anh Đới, Nguyễn Ngọc Tú, Dương Lê Vinh, Nguyễn Bá Trường, Nguyễn Duy Hoàng

Hiệu đính và chú giải: Dương Trọng Tấn

Bản Trực tuyến Miễn phí

Hỗ trợ bằng cách mua bản in tại địa chỉ:

http://infoq.com/minibooks/scrum-xp-from-the-trenches

Bản tiếng Việt được cung cấp bởi **HanoiScrum** tại:

http://hanoiscrum.net/hnscrum/resource

Mọi đóp góp, phản hồi về bản dịch, xin liên hệ hanoiscrum@gmail.com

Trang web của nhà xuất bản C4Media: InfoQ.com

Cuốn sách này thuộc loạt sách về “Enterprise Software Development” trên InfoQ.

Để có thêm thông tin hoặc đặt mua cuốn sách này hoặc các cuốn khác trên InfoQ, mời bạn liên hệ với

books@c4media.com

Người biên tập: Diana Plesa

Thiết kế bìa: Dixie Press

Dàn trang: Dixie Press

Thư viện Congress Cataloguing-in-Publication Data:

ISBN: 978-1-4303-2264-1

Được in tại Hoa Kỳ

Lời cảm ơn

Tôi chỉ có một ngày cuối tuần để soạn bản thảo đầu tiên cho cuốn sách này nhưng chắc chắn đó là một ngày làm việc với cường độ cao! 150% tập trung cho công việc :o)

Xin cảm ơn vợ tôi là Sophia và các con tôi Dave và Jenny đã dành riêng cho tôi ngày cuối tuần này, và xin cảm ơn bố mẹ vợ tôi là Eva và Jorgen đã tới giúp tôi chăm sóc gia đình.

Cũng xin gửi lời cảm ơn tới các bạn đồng nghiệp của tôi tại Crisp ở Stockholm và những người trong nhóm Yahoo Scrumdevelopment đã xem xét và giúp tôi trong việc hoàn thành tài liệu này.

Và cuối cùng, xin cảm ơn tất cả các bạn đọc đã cung cấp cho tôi những thông tin phản hồi hữu ích. Đặc biệt, tôi rất vui mừng khi biết được bài viết của tôi đã tác động nhiều đến các bạn trong việc sử dụng Phương pháp Phát triển Phần mềm Linh hoạt (Agile)!

Mục lục

Lời tựa của Jeff Sutherland

Các Nhóm cần có hiểu biết cơ bản về Scrum. Bạn làm thế nào để tạo và ước tính một Product Backlog? Bạn làm thế nào để triển khai chúng trong Sprint Backlog? Bạn quản lý biểu đồ Burndown và tính toán tốc độ của Nhóm như thế nào? Cuốn sách của Henrik là công cụ khởi nguồn cho các thực hành cơ bản, nó giúp các Nhóm vượt qua trở ngại để thực hành Scrum tốt hơn.

Thực hành tốt Scrum ngày càng trở lên quan trọng với các nhóm muốn được đầu tư dài hạn. Là nhà huấn luyện Agile cho một nhóm đầu tư mạo hiểm, tôi trợ giúp với mục tiêu của họ là chỉ đầu tư vào các công ty sử dụng Agile và thực hành tốt phương pháp này. Senior Partner của nhóm sẽ mời tất cả các công ty trong danh mục đầu tư nếu họ biết được tốc độ của các Nhóm của công ty đó. Họ sẽ khó mà trả lời đúng được câu hỏi đó vào lúc này. Các cơ hội đầu tư trong tương lai sẽ đòi hỏi những nhóm phát triển phần mềm phải nắm được tốc độ sản xuất của mình.

Tạo sao điều này lại quan trọng như vậy? Nếu các Nhóm không biết được tốc độ của mình, Product Owner không thể tạo ra được lộ trình đáng tin cậy cho việc phát hành sản phẩm. Nếu không xác định được thời điểm phát hành tin cậy, công ty có thể sẽ thất bại và các nhà đầu tư sẽ mất trắng khoản tiền của họ.

Các công ty nào dù lớn hay nhỏ, mới hay cũ, được tài trợ hay không được tài trợ đều có thể phải đối mặt với vấn đề này. Tại một cuộc thảo luận gần đây về việc triển khai Scrum của Google ở một hội thảo tổ chức ở London, tôi đã hỏi một trong số 135 người về việc có bao nhiêu người đang triển khai Scrum và đã nhận được 30 phản hồi tích cực. Sau đó tôi hỏi tiếp nếu họ đang triển khai phương pháp phát triển lặp (iterative) theo chuẩn của Nokia. Phát triển lặp là nền tảng trong Tuyên ngôn Phát triển Phần mềm Linh hoạt (Agile Manifesto) – cho phép cung cấp phần mềm chạy tốt nhanh hơn và đều đặn hơn. Sau nhiều năm cải tiến với hàng trăm Nhóm Scrum, Nokia đã đưa ra một số yêu cầu cơ bản đối với phát triển lặp:

- Các phân đoạn phải được đóng khung về thời gian dưới sáu tuần.

- Mã nguồn phải được kiểm thử bởi QA[1] cuối mỗi phân đoạn và phải chạy đúng yêu cầu.

Trong số 30 người nói rằng họ đang triển khai Scrum, chỉ có một nửa trong số đó nói họ đáp ứng được nguyên tắc đầu tiên của Tuyên ngôn Agile với các tiêu chuẩn của Nokia. Tôi hỏi tiếp nếu họ đáp ứng được các chuẩn của Nokia đối với Scrum:

- Một Nhóm Scrum phải có một Product Owner và biết rõ về người đó.
- Product Owner phải tạo ra một Product Backlog với các ước tính do Nhóm đưa ra.
- Nhóm phải có một Biểu đồ Burndown và biết được tốc độ của mình.
- Không ai ngoài Nhóm được phép can thiệp vào công việc của Nhóm trong suốt thời gian triển khai một Sprint.

Trong số 30 người đang triển khai Scrum, chỉ có 3 người qua được bài sát hạch của Nokia đối với một Nhóm Scrum. Chỉ những nhóm này mới nhận được các đầu tư trong tương lai từ các đối tác đầu tư mạo hiểm của tôi.

Giá trị mà cuốn sách của Henrik chính là ở chỗ nếu bạn tuân theo các phương pháp thực hành mà tác giả đưa ra, bạn sẽ có một Product Backlog, các ước tính cho Product Backlog, một Biểu đồ Burndown, và biết được tốc độ của Nhóm cùng với nhiều phương pháp thực hành quan trọng khác dành cho việc triển khai Scrum hiệu quả hơn.

Bạn sẽ qua được các sát hạch của Nokia về Scrum và đáng được cân nhắc để đầu tư. Nếu bạn bắt đầu khởi nghiệp, bạn có thể được rót vốn từ các nhóm đầu tư mạo hiểm. Chính bạn có thể là tương lai của phát triển phần mềm và là người kiến tạo một thế hệ các sản phẩm phần mềm hàng đầu.

Ts. Jeff Sutherland, Đồng tác giả của Scrum

---

[1] QA = Quality Assurance : bộ phận đảm bảo chất lượng. Công việc cụ thể của QA của từng công ty có sự khác nhau tương đối nhiều. Một số nơi QA thiên về đảm bảo quy trình, một số nơi QA thực hiện cả công việc của QC (quality control – kiểm soát chất lượng), có nơi thậm chí QA làm công việc của kiểm thử viên (tester).

Lời tựa của Mike Cohn

Cả Scrum và XP (Extreme Programming) đều đòi hỏi các Nhóm cần phải hoàn thành một phần công việc có thể chuyển giao được cuối mỗi phân đoạn. Các phân đoạn đó được thiết kế ngắn và đóng khung về thời gian. Việc tập trung vào cung cấp các mã nguồn chạy tốt trong một khoảng thời gian ngắn có nghĩa là Scrum và XP không có thời gian dành cho các lý thuyết. Các phương pháp này không theo đuổi việc vẽ ra các mô hình UML hoàn hảo bằng một công cụ nào đó, viết tài liệu đầy đủ về các yêu cầu của sản phẩm, hoặc viết mã nguồn có thể tương thích với tất cả các thay đổi được tưởng tượng ra trong tương lai. Thay vào đó, các Nhóm Scrum và XP tập trung vào hoàn thành các công việc. Những Nhóm đó chấp nhận rằng họ có thể gặp vấn đề trong quá trình triển khai, nhưng họ cũng nhận thấy rằng cách tốt nhất để tìm ra các vấn đề là dừng việc tư duy về phần mềm đó ở cấp độ lý thuyết (phân tích, thiết kế) và chuyển ngay sang việc bắt tay vào xây dựng sản phẩm.

Điều này tương tự với việc tập trung vào thực hiện hơn là chú trọng vào các lý thuyết được đề cập trong cuốn sách này. Rõ ràng, Henrik Kniberg hiểu được điều này ngay từ khi bắt đầu. Anh ấy không đưa ra các mô tả dài dòng về Scrum là cái gì; anh ấy chỉ ra cho chúng ta một số trang web đơn giản về các nội dung đó. Thay vào đó, Henrik bắt đầu ngay lập tức với việc mô tả cách mà Nhóm của anh ấy đã quản lý và làm việc với Product Backlog của họ ra sao. Từ đó anh ấy chuyển qua tất cả các thành phần và thực hành khác để triển khai tốt một dự án với Agile. Không hề lý thuyết. Không tài liệu tham khảo. Không có các chú thích. Không cần các thứ đó. Cuốn sách của Henrik không triết lý về việc tại sao Scrum thực thi được hoặc tại sao bạn muốn thử cái này hoặc cái kia. Cuốn sách chỉ mô tả cách để một nhóm Agile làm việc hiệu quả.

Đó là lý do mà cuốn sách có tiêu đề phụ là "Chúng tôi đã Scrum như thế nào?". Đây có thể không phải là cách mà bạn thực hành Scrum mà chỉ là cách Nhóm của Henrik đã làm. Có thể bạn sẽ hỏi tại sao lại cần quan tâm đến cách triển khai Scrum của một Nhóm nào đó. Câu trả lời là bạn nên quan tâm bởi vì tất cả chúng ta có thể học được cách triển khai Scrum tốt hơn thông qua các câu chuyện của các nhóm khác, đặc biệt là những nhóm đang làm tốt với Scrum. Không có và sẽ không bao giờ có danh sách "Các thực hành tốt nhất trong Scrum" bởi vì Nhóm và bối cảnh của dự án mới là những điều cốt lõi so với các vấn đề khác. Thay vì các cách thực hành tốt nhất, những gì chúng ta cần là biết được những thực hành hiệu quả và bối cảnh để triển khai chúng thành công. Đọc đầy đủ về các nhóm thành công và cách thức mà họ đã làm sẽ giúp bạn có những chuẩn bị tốt khi gặp phải các trở ngại trong quá trình triển khai Scrum và XP.

Henrik cung cấp nhiều các cách thực hành hiệu quả cùng với những bối cảnh cần thiết để trợ giúp chúng ta hiểu tốt hơn về cách thực hành Scrum và XP trong các dự án của mình.

Mike Cohn,

Tác giả cuốn sách "*Agile Estimating and Planning*" (Ước tính và Lập kế hoạch linh hoạt) và "*User Stories Applied for Agile Software Development*" (Ứng dụng User Story trong Phát triển Phần mềm Linh hoạt).

Lời nói đầu

Chào các bạn, Scrum đem lại hiệu quả trong công việc!

Scrum hiệu quả trong công việc! Ít nhất là với chúng tôi (bao gồm khách hàng hiện tại của tôi ở Stockholm, tên của họ tôi không tiện nêu ra ở đây). Hy vọng nó cũng sẽ hiệu quả với công việc của các bạn! Có thể cuốn sách này sẽ trợ giúp bạn trong quá trình triển khai.

Đây là lần đầu tiên tôi thấy một phương pháp phát triển phần mềm (xin lỗi Ken[2], một *framework* mới đúng) thực sự hiệu quả chứ không chỉ là sách vở. Đưa vào là dùng được ngay. Tất cả chúng tôi từ nhà phát triển, kiểm thử đến quản lý đều vui mừng vì điều này. Scrum đã trợ giúp chúng tôi vượt qua những tình huống khó khăn và cho phép chúng tôi duy trì sự tập trung và động lực mặc dù thị trường bất ổn và suy giảm về nhân sự.

Tôi không nên nói rằng tôi đã rất ngạc nhiên nhưng thực sự là như vậy. Sau khi đọc một số cuốn sách có vẻ thú vị về Scrum, nhưng phải là quá hay mới đúng (và tất cả chúng ta đều biết câu nói "khi một điều gì đó có vẻ quá tốt là đúng…"). Vì vậy cũng là hợp lý khi có một chút hoài nghi. Nhưng sau một năm triển khai Scrum tôi đã ấn tượng và sẽ tiếp tục sử dụng Scrum mặc định cho các dự án mới của mình cho tới khi nào có một lý do đủ mạnh để không dùng đến nó nữa.

[2] Ken Schwaber, đồng tác giả của Scrum, cùng với Jeff Sutherland

1
Giới thiệu

Bạn mới bắt đầu áp dụng Scrum trong tổ chức của mình, bạn đã áp dụng Scrum được một vài tháng, bạn mới có những kiến thức cơ bản về Scrum, bạn đã đọc một số quyển sách về nó, hay bạn đã có chứng chỉ Scrum Master. Xin chúc mừng bạn!

Nhưng bạn vẫn có thể gặp bối rối trong một số trường hợp.

Theo Ken Schwaber, Scrum không hẳn là một phương pháp luận, mà là một *khung làm việc* (framework). Điều đó có nghĩa là lý thuyết Scrum không hẳn sẽ nói cho bạn chính xác, cụ thể những việc cần phải làm.

Một tin tốt là tôi sẽ nói cho các bạn chính xác và rất chi tiết về cách mà tôi đã triển khai Scrum. Và, một tin xấu là những mô tả đó *chỉ là cách của tôi* khi áp dụng Scrum. Nó không có nghĩa là bạn phải áp dụng đúng theo những gì tôi làm. Thực thế, tôi cũng sẽ áp dụng theo cách khác trong những hoàn cảnh khác.

Sức mạnh và cốt lõi của Scrum là bắt buộc chúng ta phải thích ứng với từng hoàn cảnh cụ thể.

Phương pháp tiếp cận của chúng tôi là kết quả kinh nghiệm của một năm áp dụng Scrum cho nhóm phát triển khoảng 40 người. Công ty chúng tôi đã từng gặp các tình huống khó khăn trong công việc như phải làm việc ngoài giờ, có vấn đề nghiêm trọng về chất lượng, chữa cháy liên tục, trễ hạn, v.v. Chúng tôi đã quyết định sử dụng Scrum, nhưng không thực sự áp dụng cho hoàn toàn công ty, mà chỉ áp dụng cho cá nhân tôi. Với hầu hết mọi người trong nhóm phát triển lúc bấy giờ, "Scrum" là một thuật ngữ lạ lẫm như thể họ chỉ mới nghe thấy đâu đó ngoài hành lang, nó không có ý nghĩa cho công việc hàng ngày của họ.

Qua một năm áp dụng Scrum với tất cả các cấp trong công ty, chúng tôi đã thử nghiệm với các nhóm lớn nhỏ khác nhau (3 - 12 người), với độ dài Sprint khác nhau (2 - 6 tuần), với những cách định nghĩa “hoàn thành” khác nhau, với các định dạng khác nhau về Product Backlog và Sprint Backlog (Excel, Jira, thẻ chỉ mục), với các chiến thuật kiểm thử khác nhau, các phương thức demo khác nhau cũng như các cách đồng bộ hóa các nhóm Scrum khác nhau, v.v. Chúng tôi cũng đã thử

nghiệm với phương pháp XP - theo các cách khác nhau như: xây dựng liên tục, lập trình theo cặp, phát triển hướng kiểm thử (Test Driven Development - TDD), hay kết hợp chúng với Scrum.

Đó là một quá trình học hỏi liên tục và chắc chắn câu chuyện sẽ không kết thúc ở đây. Tôi tin rằng chúng tôi vẫn sẽ tiếp tục cải tiến hơn nữa (nếu chúng tôi vẫn duy trì được các cuộc Họp Cải tiến Sprint - Sprint Retrospectives) và sẽ đạt được những hiểu biết mới để áp dụng tốt nhất Scrum trong hoàn cảnh của mình.

Khuyến cáo

Tài liệu này không đưa ra một phương pháp chuẩn mực trong việc triển khai Scrum! Chúng tôi chỉ mô tả một cách áp dụng Scrum, và kết quả sẽ được sàng lọc liên tục theo thời gian. Thậm chí, bạn có thể cho rằng mọi điều chúng tôi viết là sai.

Tất cả mọi điều trong tài liệu này phản ánh quan điểm của cá nhân tôi và nó không phải là một tuyên bố chính thức từ Crisp hay từ những khách hàng hiện tại của tôi. Với lý do này, tôi đã tránh đề cập đến bất kỳ sản phẩm hay con người cụ thể.

Lý do tôi viết cuốn sách này

Khi tìm hiểu về Scrum, tôi đã đọc một vài cuốn sách về Scrum và Agile có trên các website và các diễn đàn về Scrum, tôi đã nhận chứng chỉ từ Ken Schwaber, chất vấn ông ấy và đã dành nhiều thời gian để thảo luận với các đồng nghiệp của tôi. Nhưng có lẽ, một trong những nguồn thông tin giá trị nhất lại chính là những va vấp thực tế. Những va vấp thực tế đã chuyển thành những Lý thuyết và Thực hành v.v. và v.v. Bạn thực sự làm gì trong hoàn cảnh đó? Chúng đã giúp tôi xác định (hoặc tránh được) những sai lầm điển hình mà các thành viên mới tiếp cận Scrum đã gặp phải.

Do đó, đây là cơ hội để tôi gửi lại các bạn, câu chuyện về những va vấp của chúng tôi.

Tôi hi vọng rằng cuốn sách này sẽ đem lại cho các bạn những giải pháp hữu ích trong những hoàn cảnh tương tự. Xin hãy khai sáng cho tôi.

Vậy Scrum là gì?

Ồ, xin lỗi. Với trường hợp bạn mới tiếp xúc với Scrum và XP, xin hãy xem qua một số trang web sau[3]:

- http://agilemanifesto.org/

[3] Bạn đọc có thể tiếp cận các tài liệu cơ bản về Agile bằng tiếng Việt tại http://hanoiscrum.net

- http://www.mountaingoatsoftware.com/scrum
- http://www.xprogramming.com/xpmag/whatisxp.htm

Nếu bạn không đủ kiên nhẫn để làm điều đó, xin hãy cố xem qua chúng. Các thuật ngữ tối thiểu của Scrum sẽ được giải thích ở đó và bạn cần phải biết về chúng để đọc những dòng thú vị tiếp theo trong cuốn sách này.

2

Chúng tôi làm Product Backlog như thế nào

Product Backlog là trái tim của Scrum. Tất cả sẽ được bắt đầu từ đây. Về cơ bản, Product Backlog là một danh sách ưu tiên hóa gồm các yêu cầu, các story, hoặc các tính năng của sản phẩm, hay thứ gì đó tương tự như vậy. Đó là những điều mà khách hàng mong muốn và được mô tả bằng các thuật ngữ của khách hàng.

Chúng ta gọi đó những *story* (hay đầy đủ là user story), hoặc đôi khi là các *hạng mục backlog* (backlog items).

Những story đó bao gồm các trường sau:

- **ID** (Số định danh) – một định danh duy nhất, là một con số tự tăng. Điều này nhằm tránh việc nhầm lẫn các story khi chúng ta đổi tên chúng.
- **Name** (Tên) – một cái tên ngắn, có tính mô tả cho story. Ví dụ "Xem lịch sử giao dịch của bạn". Phải đủ rõ ràng để Nhóm Phát Triển và Product Owner hiểu chính xác chúng ta đang nói về vấn đề gì, và cũng đủ rõ ràng để chúng ta phân biệt với các story khác. Thường từ 2 - 10 từ.
- **Importance** (Độ quan trọng) – Mức độ quan trọng của story mà Product Owner đánh giá. Ví dụ 10. Hoặc 150. Cao = quan trọng hơn.
    - Tôi muốn tránh dùng thuật ngữ "độ ưu tiên" (priority) bởi vì độ ưu tiên 1 thường được xem là độ ưu tiên "cao nhất", điều này sẽ trở nên rất tồi tệ nếu như bạn có ý định đặt một story khác có độ ưu tiên cao hơn. Bạn sẽ đặt độ ưu tiên đó như thế nào? Độ ưu tiên 0? Hay độ ưu tiên -1?
- **Initial estimate** (Ước tính ban đầu) – một ước tính ban đầu của đội dự án về khối lượng làm việc cần thiết để thực hiện story này so với các story khác. Đơn vị là điểm (story point) và nó tương ứng với "ý tưởng ngày công" (man-days).
    - Hãy hỏi đội dự án "nếu bạn có thể lấy một số lượng người tùy ý đối với story này (không quá ít và không quá nhiều, thường là 2 người) sau đó vào phòng làm việc với đầy đủ thực phẩm, các bạn làm việc một cách hết sức tập trung, sau bao nhiêu ngày các bạn sẽ có thể bước ra khỏi phòng làm việc với một sản phẩm được coi là

hoàn thành, có thể demo được, đã được test, có thể phát hành được?". Nếu câu trả lời là "với 3 người ở trong phòng làm việc tập trung sẽ mất khoảng 4 ngày" thì ước tính cho story đó là 12 điểm.

- Điều quan trọng không phải để bạn xác định được ước tính chính xác tuyệt đối (ví dụ một story 2 điểm sẽ mất 2 ngày để hoàn thành), điều quan trọng là cần ước tính với độ chính xác tương đối (ví dụ một story 2 điểm sẽ chiếm một nửa khối lượng công việc so với story 4 điểm)

- **How to demo** (Cách để demo) - một bản mô tả ở mức độ cao về cách demo tính năng này trong buổi demo[4] cuối Sprint. Về cơ bản thường là một đặc tả kiểm thử đơn giản. "Làm thế này, sau đó làm thế này, thì cái này sẽ xảy ra".
  - Nếu bạn triển khai TDD [5](Phát triển Hướng Kiểm Thử), mô tả này có thể được sử dụng như một mã giả (pseudo-code) cho mã kiểm thử chấp nhận (acceptance test).

- **Notes** (Các lưu ý) – bất kỳ thông tin nào khác, dùng để phân loại, tham chiếu tới các nguồn tài nguyên khác, v.v. Thường là rất ngắn gọn.

PRODUCT BACKLOG (Ví dụ)					
ID	**Tên**	**Độ quan trọng**	**Ước tính**	**Cách để demo**	**Lưu ý**
1	Gửi tiền vào	30	5	Đăng nhập, mở trang gửi tiền, gửi vào 10€, chuyển sang trang tài khoản của tôi và kiểm tra tài khoản nó đã được tăng thêm 10€.	Cần một biểu đồ tuần tự UML. Chưa cần quan tâm tới việc mã hóa ở thời điểm này.
2	Xem lịch sử giao dịch của	10	8	Đăng nhập, bấm vào "Các giao dịch". Thực hiện việc gửi tiền. Quay lại trang	Sử dụng kỹ thuật phân trang để tránh việc phải truy vấn CSDL lớn. Thiết kế tương

[4] Trong tài liệu này, tác giả dùng từ Demo để nói về buổi Họp Sơ kết (Sprint Review) theo Scrum Guide.

[5] TDD = Test-Driven Development: một chiến thuật phát triển phần mềm được điều hướng bởi kiểm thử. Còn gọi là Test-First, phương pháp này đòi hỏi lập trình viên phải viết ra các test case trước khi lập trình. Đây là một phương pháp thực hành quan trọng trong XP. Xem thêm chương "Chúng tôi kết hợp Scrum và XP như thế nào".

	bạn			"Các giao dịch" và kiểm tra thông tin.	tự trang xem danh sách người dùng.

Chúng tôi đã thử nghiệm với rất nhiều trường khác, nhưng cuối cùng, sáu trường trên là những trường mà chúng ta thường sử dụng nhất qua các Sprint.

Chúng ta có thể làm Product Backlog với một tài liệu Excel đã mở chức năng chia sẻ (nhiều người dùng có thể đồng thời chỉnh sửa tập tin). Chính thức thì Product Owner sở hữu tài liệu này, nhưng chúng ta muốn những người khác trong Nhóm Phát Triển cũng biết về nó. Có nhiều khi một nhà phát triển muốn mở tài liệu này ra để làm rõ hơn một điều gì đó hoặc thay đổi một ước tính nào đó.

Cũng vì thế, chúng ta không đặt tài liệu này trong một kho kiểm soát phiên bản (version control repository); thay vào đó chúng ta đặt nó vào một ổ đĩa chia sẻ. Đây là cách đơn giản nhất để cho phép nhiều người sửa đồng thời mà tài liệu không bị khóa hay bị xung đột.

Hầu hết những tài liệu khác có thể để ở kho kiểm soát phiên bản.

Các trường khác

Đôi khi chúng ta sử dụng thêm các trường khác trong Product Backlog, mục đích là làm cho Product Owner cảm thấy thuận tiện hơn khi sắp xếp độ ưu tiên.

- **Track** (Nhánh) – một sự phân loại cho story, ví dụ "back office" hoặc "optimization". Theo cách đó, Product Owner có thể dễ dàng lọc ra tất cả các hạng mục "optimization" và đặt mức ưu tiên thấp, v.v..
- **Components** (thành phần) – Thường là các ô kiểm (checkbox) trong tài liệu Excel, ví dụ "database, server, client". Đây là nơi mà Nhóm Phát Triển hoặc Product Owner có thể xác định các thành phần kỹ thuật nào sẽ tham gia vào quá trình triển khai story. Điều này hữu ích khi bạn có nhiều nhóm Scrum, ví dụ một nhóm "back office" và một nhóm client, và bạn muốn các nhóm dễ dàng chọn các story để thực hiện.
- **Requestor** (người yêu cầu) – có thể Product Owner muốn biết khách hàng hoặc người liên quan nào yêu cầu hạng mục này, để có thể gửi các phản hồi trong quá trình thực hiện.
- **Bug tracking ID** (mã bug) – nếu bạn có một hệ thống theo dõi lỗi riêng, như chúng tôi sử dụng Jira, sẽ rất hữu ích để theo dõi các lỗi liên quan tới một story.

Làm thế nào chúng ta giữ Product Backlog ở mức nghiệp vụ [6]

Nếu Product Ownder biết về kỹ thuật, anh ta có thể thêm các story như "Đặt chỉ mục cho bảng Event". Tại sao anh ta làm việc này? Mục tiêu thực sự có thể là một điều gì đó chẳng hạn như "tăng tốc độ tìm kiếm các event trên form dành cho back-office".

Điều đó có nghĩa rằng các chỉ mục không phải là nút cổ chai dẫn tới form tìm kiếm bị chậm. Nó có thể là một thứ gì đó hoàn toàn khác. Nhóm Phát Triển sẽ là người phải tìm cách giải quyết vấn đề, vì thế Product Owner nên tập trung vào các mục tiêu nghiệp vụ hơn là kỹ thuật.

Khi tôi nhìn thấy các story mang quá nhiều tính chất kỹ thuật, tôi thường hỏi Product Owner một chuỗi các câu hỏi "tại sao" cho tới khi chúng tôi tìm ra mục tiêu ẩn chứa trong đó. Sau đó chúng tôi phân tích lại story theo hướng của các mục tiêu ẩn bên dưới nó (đó là "tăng tốc độ tìm kiếm các event trong biểu mẫu tìm kiếm của back office"). Mô tả kỹ thuật ban đầu kết thúc như một ghi chú ("Đặt chỉ mục cho bảng Event có thể giải quyết vấn đề này").

[6] Không giống như đặc tả hệ thống (System Requirement Specification), các user story là cách thể hiện yêu cầu người dùng ở dạng cô đọng và đơn giản nhất, dùng ngôn ngữ người dùng để thể hiện. Việc này có tác dụng lớn cho việc thảo luận và tìm hiểu nhu cầu của người dùng theo cách chân thực nhất: thực sự người dùng muốn gì? chức năng này để giải quyết nghiệp vụ nào? Do vậy, việc giữ Product Backlog ở mức nghiệp vụ là rất quan trọng.

3

Chúng tôi chuẩn bị Họp Kế hoạch Sprint như thế nào

OK, chúng ta sẽ nắm bắt được ngày lên kế hoạch cho Sprint một cách nhanh chóng thôi. Một bài học chúng tôi học đi học lại đó là:

Bài học: Hãy chắc chắn Product Backlog đã định hình trước khi tiến hành Họp Kế hoạch Sprint.

Điều đó có nghĩa là gì ? Đó là tất cả các story được xác định một cách đầy đủ hay chưa? Đó là việc ước lượng (estimate) đã chính xác chưa ? Đó là thứ tự ưu tiên đã được xác định chưa? Nếu câu trả lời là: chưa, chưa và chưa! Có nghĩa là:

- Product Backlog chưa được coi là xong và Product Backlog cần phải được thực hiện cho xong đã! (chẳng lẽ bạn phải tưởng tượng ra nó sao?)
- Có một Product Backlog và một Product Owner (cho từng sản phẩm).
- Tất cả các công việc sẽ có đánh giá về tầm quan trọng của chúng, các đánh giá này là khác nhau.
    - Thực tế, sẽ không có vấn để gì nếu các công việc có độ quan trọng thấp hơn có cùng giá trị, do vậy dù thế nào các công việc này sẽ không được nâng mức độ lên trong suốt thời gian lên kế hoạch Sprint.
    - Bất kỳ một story nào mà Product Owner tin có một khả năng nào đó được đưa vào trong Sprint tiếp theo thì nên có duy nhất một mức độ quan trọng.
    - Đánh giá độ quan trọng chỉ dùng để bố trí các công việc cho phù hợp. Do đó nếu công việc A có độ quan trọng là 20 và công việc B có độ quan trọng là 100, có nghĩa là B quan trọng hơn A. Nó không có nghĩa là B quan trọng hơn A gấp 5 lần. Nếu B có mức đánh giá quan trọng là 21 thì nó sẽ vẫn có nghĩa như khi nó có độ quan trọng là 100!
    - Sẽ rất có ích nếu ta để một khoảng giãn cách khi đánh giá mức độ quan trọng giữa các công việc khác nhau, chẳng hạn trong trường hợp công việc C đưa thêm vào có độ quan trọng hơn A nhưng nhỏ hơn B. Tất nhiên bạn sẽ có thể sử dụng độ quan trọng cho C là 20.5, nhưng điều đó không hay lắm, thay vì thế chúng ta nên để một khoảng giãn cách giữa các công việc.

- Product Owner phải *hiểu rõ* từng story (thông thường anh ta chính là tác giả, nhưng trong một số trường hợp người khác bổ sung yêu cầu mà Product Owner có thể xác định độ ưu tiên). Anh ta không cần biết chính xác chúng được triển khai ra sao, nhưng anh ta sẽ phải hiểu tại sao có story đó.

Lưu ý: Những người không phải là Product Owner có thể bổ sung story vào Product Backlog. Nhưng họ không được gán độ quan trọng cho công việc đó, quyền này hoàn toàn chỉ dành cho Product Owner. Product Owner không được bổ sung thêm các ước lượng (estimate), quyền này là thuộc hoàn toàn của Nhóm Phát Triển.

Một số hướng khác chúng tôi đã thử và đánh giá:

- Sử dụng Jira [7](hệ thống giám sát lỗi chúng tôi dùng) để quản lý Product Backlog. Tuy nhiên, hầu hết Product Owner đều cảm thấy khó thích ứng với hệ thống này. Excel là một cách hay và dễ dàng để thao tác trực tiếp. Bạn có thể dễ dàng đặt màu, sắp xếp lại các công việc, bổ sung thêm cột mới, các lưu ý, có thể xuất và nhập dữ liệu, v.v..
- Sử dụng các công cụ hỗ trợ quy trình agile như VersionOne, ScrumWorks, XPlanner, v.v. Chúng tôi đã không quan tâm nhiều tới việc đánh giá các công cụ này nhưng có thể chúng tôi sẽ thực hiện việc đó trong tương lai.

[7] Jira (www.atlassian.com/software/jira) là một ứng dụng web dùng cho các nhóm phát triển phần mềm với công dụng chính là quản lí và theo dõi lỗi.

4

Chúng tôi họp Kế hoạch Sprint ra sao

Lập kế hoạch cho Sprint là buổi họp rất quan trọng, đây là sự kiện quan trọng nhất trong Scrum (theo quan điểm của tôi về vấn đề này). Buổi họp lập kế hoạch cho Sprint được thực hiện không tốt có thể làm ảnh hưởng rất lớn tới toàn bộ Sprint.

Mục đích của buổi họp kế hoạch là cung cấp thông tin đầy đủ cho Nhóm [8]giúp họ có thể có đầy đủ thông tin làm việc trong vài tuần, để Product Owner tin tưởng nhóm có thể hoàn thành các công việc đó..

OK, điều này quả thực còn mơ hồ. Kết quả cụ thể của buổi họp lập kế hoạch Sprint là:

- Xác định mục tiêu của Sprint.
- Danh sách thành viên của Nhóm (mức độ cam kết của họ, nếu không phải là 100%).
- Sprint Backlog (= danh sách các công việc thực hiện trong Sprint).
- Xác định được ngày demo Sprint (Sơ kết Sprint).
- Xác định thời gian và địa điểm cho buổi Họp Scrum hằng ngày (Daily Scrum).

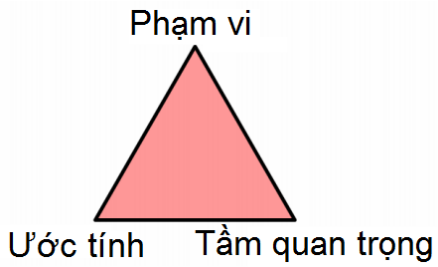
Tại sao Product Owner phải có mặt

Đôi khi Product Owner không sẵn sàng bỏ thời gian để họp lập kế hoạch Sprint với Nhóm. "Các anh em, tôi đã đưa ra đây những gì tôi muốn. Tôi không có thời gian để tham dự buổi họp lập kế hoạch của các bạn". Đó là một vấn đề tương đối nghiêm trọng.

Lý do tại sao toàn bộ Nhóm và Product Owner phải có mặt tại buổi họp lập kế hoạch Sprint, bởi vì mỗi tính năng của sản phẩm chứa ba tham số mà có mối quan hệ phụ thuộc lẫn nhau.

---

[8] Trong tài liệu này, tác giả dùng Nhóm (The Team) để chỉ Nhóm Phát triển. Đây là cách dùng từ cũ. Từ tháng 10-2011, Ken Schwaber và Jeff Sutherland thống nhất cách dùng từ khác trong bản cập nhật của Scrum Guide (http://www.scrum.org/scrumguides/). Theo đó, có ba vai trò trong một Nhóm Scrum, bao gồm Product Owner, Scrum Master và Nhóm Phát triển (Development Team).



Phạm vi và tầm quan trọng được xác định bởi Product Owner. Ước tính (estimate) do Nhóm thực hiện. Trong buổi họp lập kế hoạch Sprint, ba tham số này liên tục được điều chỉnh bằng đối thoại trực tiếp giữa Nhóm và Product Owner.

Thường các Product Owner bắt đầu buổi họp bằng cách tóm tắt mục đích mong muốn của anh ta đối với Sprint và các tính năng quan trọng nhất của sản phẩm. Tiếp theo, Nhóm sẽ thực hiện ước lượng thời gian cho mỗi story, bắt đầu với story quan trọng nhất. Khi thực hiện việc này, đội sẽ đưa ra các câu hỏi quan trọng về phạm vi – "Tính năng 'delete user' có bao gồm việc hủy bỏ các giao dịch đang chờ đối với user đó hay không ?" Trong một vài trường hợp câu trả lời sẽ làm cả Nhóm ngạc nhiên, đòi hỏi Nhóm thay đổi ước tính của mình.

Trong một vài trường hợp, ước tính thời gian cho story sẽ không theo mong muốn của Product Owner. Điều này có thể cần yêu cầu anh ta thay đổi mức độ quan trọng của story. Hoặc thay đổi phạm vi của story, điều này sẽ dẫn tới Nhóm phải ước tính lại, v.v và v.v.

Hình thức cộng tác trực tiếp là nền tảng đối với Scrum, và thực tế là nền tảng của tất cả phương pháp phát triển phần mềm linh hoạt.

Bạn sẽ làm gì nếu Product Owner vẫn nhất định không dành thời gian tham gia buổi họp lập kế hoạch Sprint? Tôi thường thử một trong những chiến lược sau, lần lượt là:

- Cố gắng thuyết phục Product Owner hiểu tại sao sự tham gia trực tiếp của anh ấy là không thể thiếu và hy vọng anh ta thay đổi suy nghĩ.
- Thử để một ai đó trong Nhóm tình nguyện đóng vai như là người đại diện Product Owner trong suốt buổi họp. Nói với Product Owner rằng "Vì anh không thể tham dự buổi họp, chúng tôi sẽ để Jeff đại diện cho anh ở đây. Anh sẽ ủy nhiệm hoàn toàn cho Jeff việc thay đổi thứ tự ưu tiên, phạm vi của story trong suốt cuộc họp. Tôi mong anh hãy truyền đạt mong muốn của anh càng nhiều càng tốt trước khi cuộc họp diễn ra. Nếu anh không thích Jeff đại diện, anh có thể gợi ý cho tôi một ai khác, miễn là người này sẽ có thể tham gia cuộc họp với chúng tôi từ đầu tới cuối."
- Cố gắng thuyết phục nhà quản lý bổ nhiệm một Product Owner mới.

- Hoãn thực hiện Sprint cho tới khi Product Owner sắp xếp được thời gian để tham dự buổi họp. Trong khi đó, từ chối cam kết về bất cứ việc bàn giao sản phẩm nào. Để cho Nhóm làm bất cứ thứ gì mà họ cảm thấy quan trọng nhất trong mỗi ngày trì hoãn đó.

Tại sao chất lượng là thứ không thể thỏa hiệp

Trong hình tam giác phía trên tôi đã cố ý tránh tham số thứ 4 là *chất lượng*. Tôi gắng phân biệt giữa *chất lượng bên trong* và *chất lượng bên ngoài*.

- Chất lượng bên ngoài là những gì người sử dụng hệ thống nhận được. Một giao diện người dùng chậm và không trực quan là một ví dụ về chất lượng bên ngoài kém.
- Chất lượng bên trong nói đến những vấn đề mà thường không nhìn thấy từ phía người sử dụng, nhưng nó có ảnh hưởng sâu sắc đến khả năng bảo trì hệ thống. Những thứ như tính nhất quán trong các thiết kế hệ thống, các trường hợp cần kiểm thử, mã nguồn dễ đọc, cải tiến mã nguồn, v.v..

Nhìn chung mà nói, một hệ thống với chất lượng bên trong cao có thể vẫn có chất lượng bên ngoài thấp. Nhưng một hệ thống với chất lượng bên trong thấp sẽ hiếm khi có được chất lượng bên ngoài cao. Rất khó để xây dựng một cái gì tốt đẹp dựa trên một nền tảng yếu kém.

Tôi coi chất lượng bên ngoài như một phần của mục tiêu chất lượng. Trong một vài trường hợp, khi cảm thấy nghiệp vụ đã hoàn hảo chúng ta có thể cho phát hành (release) một phiên bản của hệ thống mà giao diện người dùng vừa chậm, vừa xấu, sau đó phát hành một phiên bản tốt hơn sau này. Tôi đưa sự đánh giá đó cho Product Owner, vì anh ta là người chịu trách nhiệm xác định phạm vi.

Tuy nhiên, chất lượng bên trong không đưa ra để thảo luận. Đó là trách nhiệm của Nhóm để đảm bảo chất lượng hệ thống trong mọi tình huống và điều này không thể thỏa hiệp một cách đơn giản được. Không bao giờ.

(Mà, OK, hầu như là không bao giờ)

Vậy chúng ta nói về sự khác biệt giữa các vấn đề về chất lượng bên trong và bên ngoài như thế nào?

Trong trường hợp Product Owner nói rằng “OK các chàng trai, tôi tôn trọng ước định thời gian của các bạn là 6 điểm story (story point), nhưng tôi chắc chắn các bạn có một vài thay đổi nhanh để thực hiện việc này với một nửa thời gian nếu các bạn đặt toàn tâm toàn ý vào nó.”

A ha! Anh ta đang thử sử dụng chất lượng bên trong như một tham số. Làm sao bạn biết? Bởi vì anh ta muốn chúng tôi giảm ước tính story point mà không "trả chi phí" cho phần giảm bớt đó. Từ "thay đổi nhanh" kích hoạt một cảnh báo trong đầu bạn …

Và tại sao chúng tôi không cho phép điều này?

Với kinh nghiệm của mình tôi thấy việc hy sinh chất lượng bên trong luôn luôn là một ý tưởng rất tệ hại. Việc tiết kiệm thời gian sẽ trả giá đắt về chi phí trong cả ngắn hạn và dài hạn. Khi nền tảng mã nguồn được phép bắt đầu với chất lượng tồi thì rất khó để làm nó chất lượng trở lại sau này.

Thay vào đó tôi cố hướng cuộc thảo luận tập trung vào mục tiêu cần thực hiện. "Nếu với Product Owner việc cần cho ra đời tính năng sớm là quan trọng, chúng ta có thể giảm bớt mục tiêu đi để thực hiện nó nhanh chóng hơn? Có lẽ chúng ta có thể đơn giản hóa việc quản lý lỗi và thực hiện tính năng 'quản lý lỗi nâng cao' trong một story khác trong tương lai ? Hoặc chúng ta giảm độ ưu tiên của một số tính năng khác trong sản phẩm để có thể tập trung vào tính năng quan trọng hiện tại?"

Khi Product Owner hiểu được rằng chất lượng bên trong không thể thương lượng được, thay vào đó anh ta thường nhanh chóng hướng tới thảo luận về các tham số khác.

Các cuộc họp lập kế hoạch Sprint kéo dài …

Điều khó nhất trong cuộc họp lập kế hoạch Sprint là:

- Mọi người không nghĩ họ sẽ mất quá nhiều thời gian cho cuộc họp.
- … thực tế lại như vậy !

Mọi thứ trong Scrum chỉ được phép nằm trong một khung thời gian (time-box). Tôi thích buổi họp tuân theo quy tắc nhất quán và đơn giản như thế. Chúng tôi cố gắng tuân thủ quy tắc đó.

Vì chúng tôi sẽ làm gì khi một buổi họp lập kế hoạch cho Sprint đã được đặt trong mộ khung thời gian sắp tiến gần tới thời điểm kết thúc và không có dấu hiệu cho thấy đã xác định được mục tiêu Sprint hay Sprint Backlog? Chúng ta có cắt ngắn nó không??? Hay chúng ta kéo dài cuộc họp thêm một giờ? Hoặc chúng ta dừng cuộc họp và tiếp tục vào ngày tiếp theo?

Điều này thường xuyên xảy ra, đặc biệt đối với những Nhóm mới. Vậy bạn phải xử lý thế nào? Tôi không biết. Nhưng chúng tôi đã làm gì ? Ồ, được thôi, tôi thường quyết định một cách dứt khoát ngắn gọn. Chấm dứt cuộc họp. Để Sprint đấy. Cá biệt hơn, tôi nói với Nhóm và Product Owner "vậy thôi, cuộc họp này kết thúc trong vòng 10 phút nữa. Chúng ta không thực sự có nhiều kế hoạch cho Sprint này. Chúng ta sẽ thực hiện những gì hiện chúng ta có chứ, hay chúng ta sẽ tổ

chức họp thêm 4 tiếng vào 8 giờ sáng ngày mai nhé?". Các bạn có thể đoán xem họ sẽ trả lời những gì … : :o)

Tôi đã thử kéo dài thêm cuộc họp. Điều đó thường không đạt được điều gì, bởi vì mọi người mệt mỏi. Nếu họ không đưa ra được một kế hoạch tử tế trong 2-8 giờ (hoặc theo khung thời gian bạn đề ra), họ có thể cũng sẽ không làm thêm được gì trong thời gian thêm đó. Lựa chọn tiếp theo thực tế là OK hơn, đó là hẹn một cuộc họp mới vào ngày tiếp theo. Ngoại trừ trường hợp mọi người thiếu kiên nhẫn và muốn tiếp tục Sprint, và không muốn mất thêm thời gian để lập kế hoạch.

Vì vậy tôi cắt bớt thời gian đi. Và tất nhiên, Sprint sẽ vẫn chưa ra đâu vào đâu cả. Tuy nhiên, bù vào đó, Nhóm đã có được một bài học rất có giá trị và buổi họp lập kế hoạch tiếp theo sẽ hiệu quả hơn nhiều. Hơn nữa, mọi người sẽ ít phản kháng hơn khi bạn đề xuất thời gian họp mà trước đó họ nghĩ là quá dài.

Hãy học để tuân thủ khung thời gian của bạn, học để thiết lập độ lớn của khung thời gian sát với thực tế hơn. Điều này áp dụng cho cả việc xấc định độ dài cuộc họp cũng như Sprint.

Lịch trình Họp Kế hoạch Sprint

Có một số kiểu lên lịch họp cơ bản sẽ giúp bạn giảm bớt nguy cơ trong việc chia điểm dừng trong phạm vi khung thời gian.

Đây là ví dụ về một kiểu lên lịch chúng tôi sử dụng.

Họp Kế hoạch Sprint 13:00 – 17:00 (nghỉ giải lao 10 phút sau mỗi giờ họp)

- **13:00 – 13:30**. Product Owner trình bày mục tiêu của Sprint và tóm tắt Product Backlog. Địa điểm triển khai, thời gian và ngày triển khai Sprint được thiết lập.
- **13:30 – 15:00**. Nhóm Phát triển ước tính thời gian, và chia thành các tác vụ cần thiết. Product Owner cập nhật đánh giá mức độ quan trọng cho các hạng mục. Các tác vụ được làm rõ. "Cách thức tiến hành" được điền vào cho tất cả các hạng mục có tầm quan trọng cao.
- **15:00 – 16:00**. Nhóm chọn story để đưa vào Sprint. Thực hiện tính toán tốc độ để đảm bảo tính khả thi.
- **16:00 – 17:00**. Chọn thời gian và địa điểm để họp Scrum hàng ngày (nếu có khác biệt với Sprint trước đó). Chia nhỏ hơn các story vào các đầu việc phải làm (tasks).

Lịch tiến độ không nhất thiết bắt buộc y nguyên như vậy. Scrum Master có thể đưa ra các khung thời gian ngắn hay dài hơn nếu thấy cần thiết trong quá trình họp.

Xác định thời gian cho Sprint

Một trong những kết quả cần đạt được trong cuộc Họp Kế hoạch Sprint là xác định được ngày triển khai Sprint. Điều đó có nghĩa bạn phải quyết định khung thời gian cho Sprint.

Vậy một Sprint kéo dài bao lâu là tốt ?

Sprint ngắn là tốt. Chúng cho phép công ty "linh hoạt", ví dụ, thay đổi định hướng thường xuyên. Sprint ngắn = chu kỳ phản hồi ngắn = chuyển giao thường xuyên hơn = phản hồi khách hàng thường xuyên hơn = tốn kém thời gian đi sai hướng ít hơn = học và cải tiến nhanh hơn, v.v..

Tuy thế, Sprint dài cũng tốt. Nhóm có nhiều thời gian hơn để tạo đà, họ có nhiều thời gian hơn để khắc phục các vấn đề mà vẫn đảm bảo mục tiêu của Sprint, bạn ít bù đầu hơn cho các buổi họp lập kế hoạch Sprint, tổ chức demo, v.v..

Nhìn chung mà nói Product Owner thích các Sprint ngắn còn nhà phát triển thì thích các Sprint dài. Do đó độ dài của Sprint cần phải dung hòa. Chúng tôi đã thử nghiệm rất nhiều về vấn đề này và rút ra độ dài thích hợp cho chúng tôi là: 3 tuần. Hầu hết các Nhóm của chúng tôi (không phải tất cả) làm việc với các Sprint 3 tuần. Nó đủ ngắn để đảm bảo cho chúng tôi sự cộng tác linh hoạt, đủ dài cho Nhóm đạt được mạch làm việc và khắc phục các vấn đề nảy sinh trong Sprint.

Chúng tôi kết luận một điều là: *Thử nghiệm* độ dài khác nhau cho Sprint ban đầu. Đừng lãng phí quá nhiều thời gian cho việc *phân tích*, chỉ chọn độ dài vừa phải và ghi nhận kết quả đối với một hoặc hai Sprint, sau đó thay đổi độ dài.

Tuy nhiên, khi bạn quyết định độ dài mà bạn thấy tốt nhất, hãy *duy trì* nó trong một khoảng thời gian nhất định. Sau vài tháng thử nghiệm, chúng tôi thấy 3 tuần là hợp lý. Do đó, chúng tôi thực hiện các Sprint với thời hạn 3 tuần. Đôi khi cảm giác nó cũng hơi dài, đôi khi lại cảm thấy quá ngắn. Nhưng bằng cách duy trì cùng một độ dài, nó sẽ giúp tạo ra cho Nhóm một nhịp đập (heartbeat) mà mọi người cảm thấy thoải mái tham gia. Không có tranh luận nào về ngày phát hành bởi vì mọi người biết rằng cứ 3 tuần thì có một lần phát hành (release).

Xác định mục tiêu Sprint

Điều này cần làm trong toàn bộ thời gian họp. Tại một số thời điểm trong buổi họp lập kế hoạch tôi hỏi "vậy mục tiêu của Sprint này là gì?" và mọi người ngây lại nhìn chằm chằm vào tôi và Product Owner.

Vì một vài lý do rất khó để đạt được mục tiêu của Sprint. Nhưng tôi đã thấy điều đó thực đáng giá để theo sát mục tiêu đề ra. Thà nhắc tới mục đích tuy phá đám giữa chừng còn hơn là không nhắc lần nào. Mục tiêu có thể là "làm ra nhiều tiền hơn" hoặc "hoàn thành 3 story có mức ưu tiên cao nhất" hoặc "gây chú ý cho CEO" hoặc "làm cho hệ thống đủ tốt để đưa vào các phiên bản beta đang chạy" hoặc "bổ sung các hỗ trợ cơ bản cho văn phòng" hoặc là bất cứ thứ gì. Điều quan trọng là nó sẽ thuộc vấn đề về nghiệp vụ, chứ không phải là khía cạnh kỹ thuật. Điều này tức là nói về những thứ mà những người bên ngoài nhóm có thể hiểu.

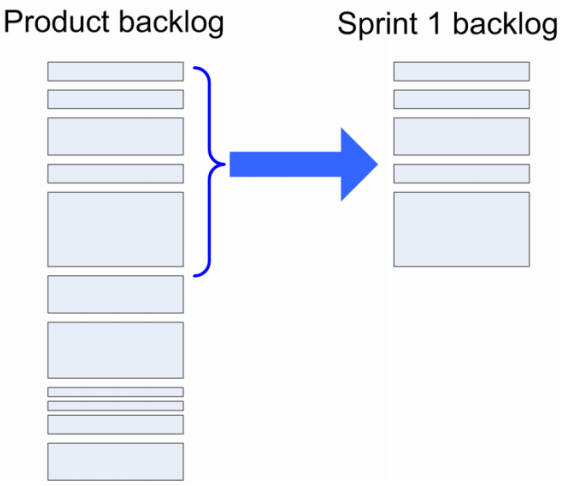
Mục tiêu của Sprint sẽ trả lời câu hỏi cơ bản "*Tại sao* chúng ta thực hiện Sprint này? Tại sao tất cả chúng ta vẫn tiếp tục công việc thay vì đi nghỉ?" Thực tế, một cách để giúp Product Owner làm rõ mục đích của Sprint là đặt ra các câu hỏi như vậy.

Mục tiêu không phải những thứ mà dễ dàng đạt được ngay. "Gây ấn tượng cho CEO" có thể là một mục tiêu tốt, nếu anh ta chưa ấn tượng bởi hệ thống hiện có. Trong trường hợp như vậy mọi người có thể về nhà mà vẫn đạt được mục tiêu của Sprint.

Mục tiêu của Sprint có thể có vẻ rất ngớ ngẩn và được nghĩ ra trong suốt thời gian lập kế hoạch Sprint, nhưng nó thường xuất hiện ở giữa Sprint khi mọi người bắt đầu lẫn lộn về những gì họ sẽ phải làm. Nếu bạn có một vài Nhóm Scrum (như tôi có) làm việc với nhiều sản phẩm khác nhau, sẽ rất hữu ích để liệt kê ra các mục tiêu của Sprint của tất cả các đội trong một trang wiki (hay bất cứ thứ gì) và đặt chúng tại nơi nổi bật để mọi người trong công ty (không chỉ quản lý cấp cao) biết những gì công ty đang thực hiện - và lý do thực hiện chúng!

Nhóm quyết đưa story nào vào Sprint như thế nào?

Một trong những hoạt động chính của buổi Họp kế hoạch Sprint là quyết định story nào sẽ được đưa vào Sprint. Cụ thể hơn, các story gì sẽ được chuyển từ Product Backlog vào Sprint Backlog.



Hãy nhìn vào hình ở phía trên. Mỗi một hình chữ nhật thể hiện 1 story, được sắp xếp theo độ quan trọng. Tính năng quan trọng nhất nằm ở đỉnh danh sách. Kích thước của mỗi hình chữ nhật thể hiện kích thước của tính năng đó (ví dụ: các story được ước tính bằng số điểm). Chiều cao của hình ngoặc nhọn màu xanh thể hiện *tốc độ ước tính* của nhóm, ví dụ, bao nhiêu điểm mà Nhóm tin rằng có thể hoàn thành trong Sprint tiếp theo.

Sprint Backlog ở bên phải là hình ảnh của các story lấy từ Product Backlog. Nó thể hiện danh sách các tính năng mà Nhóm sẽ phải thực hiện trong Sprint này.

Nhóm quyết định có bao nhiêu story đưa vào Sprint. Không có Product Owner hay bất kỳ ai khác can thiệp vào quyết định này.

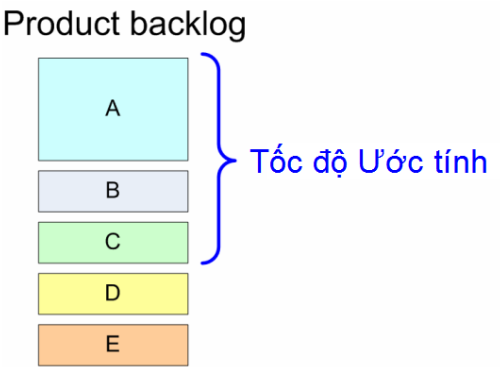
Điều này nảy sinh 2 câu hỏi :

1. Làm thế nào Nhóm quyết định đưa story nào vào Sprint?
2. Product Owner làm thế nào để tác động đến các quyết định của họ?

Tôi sẽ bắt đầu với câu hỏi thứ hai.

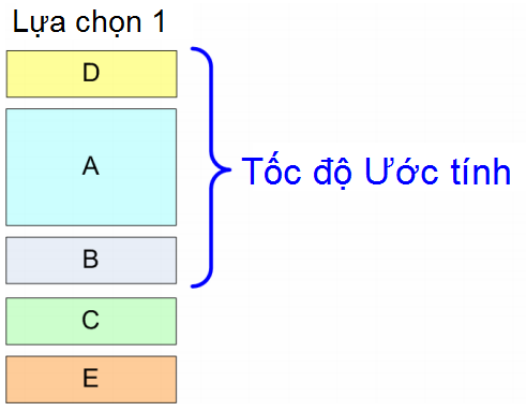
Product Owner tác động thế nào để story nào đó được đưa vào Sprint?

Tôi sẽ kể về các tình huống dưới đây mà tôi gặp phải trong suốt buổi họp lập kế hoạch

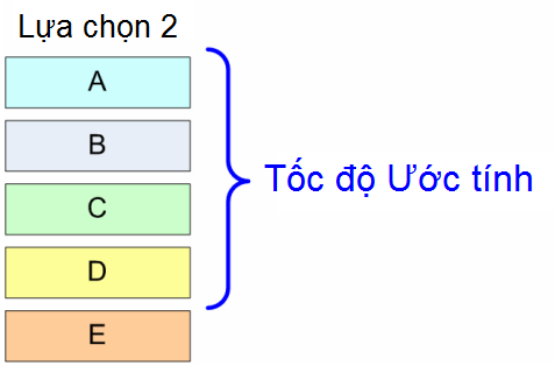


Product Owner không hài lòng D khi thấy story không được đưa vào Sprint. Anh ta sẽ có những lựa chọn gì trong suốt buổi họp lập kế hoạch?

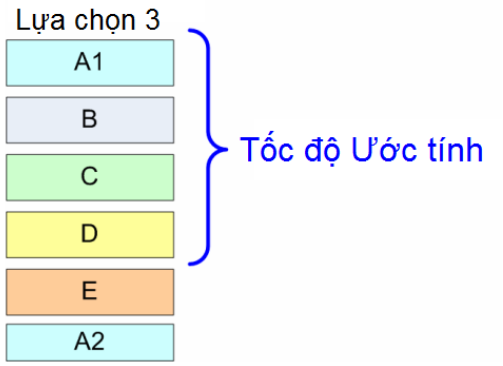
Lựa chọn thứ nhất đó là ưu tiên hóa lại Product Backlog. Nếu anh ta nâng tầm quan trọng của story D lên cao hơn, Nhóm sẽ bắt buộc phải bổ sung nó vào Sprint (trong trường hợp này story C sẽ bị đẩy ra khỏi Sprint Backlog).



Lựa chọn thứ hai là thay đổi phạm vi – giảm phạm vi của story A cho tới khi Nhóm tin là story D sẽ phù hợp với Sprint này hơn.



Lựa chọn thứ ba là chia nhỏ story. Product Owner có thể quyết định rằng có một vài yếu tố của story A không thực sự quan trọng, do đó anh ta chia A thành hai story A1 và A2 với mức độ quan trọng khác nhau.



Như bạn thấy, mặc dù Product Owner thường không thể điều khiển được tốc độ ước tính, nhưng có nhiều cách anh ta có thể tác động để story nào đó được đưa vào Sprint.

Nhóm quyết định đưa story vào Sprint như thế nào

Chúng tôi sử dụng 2 kỹ thuật để làm việc này:

- Trực giác (Gut Feeling)
- Tính toán tốc độ (Velocity)

Ước lượng với kỹ thuật Gut Feel

- **Scrum Master:** "Này các bạn, chúng ta có thể kết thúc story A trong Sprint này được không?" (ám chỉ tính năng quan trọng nhất trong các hạng mục của Product Backlog)
- **Lisa:** "À. Tất nhiên là được. Chúng ta có 3 tuần, và tính năng này cũng không khó lắm"
- **Scrum Master:** "OK, vậy với tính năng B thì thế nào?" (ám chỉ tính năng quan trọng thứ 2)
- **Tom và Lisa cùng trả lời:** "Vẫn không vấn đề gì"
- **Scrum Master:** "OK, vậy sau tính năng A, B thì làm tiếp C được không?"
- **Sam (hướng tới Product Owner):** "Tính năng C bao gồm việc xử lý Advanced Error?"
- **Product Owner:** "Không, cậu có thể bỏ qua việc đó, chỉ triển khai xử lý lỗi cơ bản."
- **Sam:** "nếu vậy thì C cũng ổn."
- **Scrum Master:** "OK, thế còn tính năng D?"
- **Lisa:** "ừm… "
- **Tom:** "Tôi nghĩ là chúng ta có thể."
- **Scrum Master:** "Chắc chắn 90%? hay chỉ 50%?"
- **Lisa và Tom:** "Hơn 90%".
- **Scrum Master:** "OK, vậy ta sẽ làm thêm D. Thế còn tính năng E?"
- **Sam:** "Có lẽ được."

- **Scrum Master:** "90% hay 50%?"
- **Sam:** "Tôi nghĩ khoảng 50%".
- **Lisa:** "Tôi cũng thế"
- **Scrum master:** "OK, vậy ta để lại sau. Chúng ta sẽ hoàn thành các tính năng A, B, C, D. Chúng ta có thể hoàn thành E nếu được, nhưng không ai chắc chắn nên ta sẽ không đưa vào Sprint lần này. Mọi người thấy thế nào?"
- **Mọi người cùng trả lời:** "OK"

Kỹ thuật Gut Feel áp dụng tốt cho những nhóm nhỏ hay những Sprint ngắn.

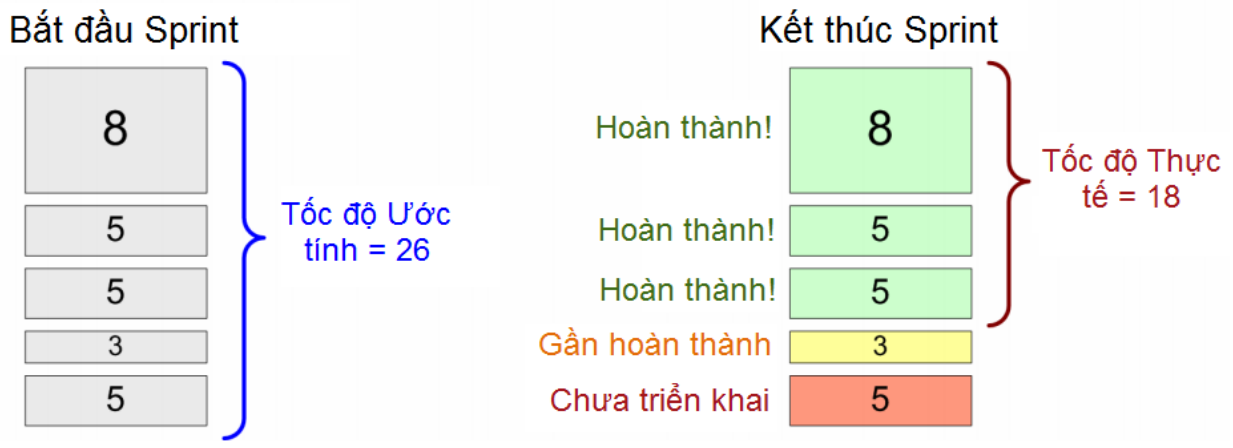
Ước lượng với kỹ thuật tính tốc độ

Kỹ thuật này bao gồm 2 bước:

1. Xác định *Tốc độ ước tính*
2. Tính toán xem bao nhiêu story có thể làm mà không vượt quá tốc độ ước tính.

Tốc độ là thước đo "lượng công việc hoàn thành", ở đó, mỗi hạng mục được đặt một trọng số trong ước lượng ban đầu.

Hình ảnh dưới đây là một ví dụ về *tốc độ ước tính (estimated velocity)* ban đầu của một Sprint và *tốc độ thực tế (actual velocity)* ở cuối Sprint đó. Mỗi hình chữ nhật là một story, con số bên trong ô chữ nhật đó là ước tính ban đầu.



Lưu ý rằng tốc độ thực tế được dựa trên bản ước tính *ban đầu* của mỗi story. Bất kỳ thay đổi nào về thời gian ước tính hoàn thành tính năng trong suốt Sprint sẽ bị lờ đi.

Tôi đã nghe thấy phản hồi rằng: "Điều này hữu ích như thế nào? Tốc độ cao hay thấp có thể phụ thuộc vào rất nhiều yếu tố! Các lập trình viên yếu kém, sự đúng đắn của ước tính ban đầu, sự mở rộng phạm vi, không lập kế hoạch cho những xáo trộn trong suốt Sprint, v.v.!"

Tôi đồng ý rằng các con số này rất khô khan. Nhưng nó vẫn hữu ích, đặc biệt khi so sánh với việc không có gì để xem xét. Nó cung cấp cho bạn một số dữ kiện cứng. "Không kể những lý do trên, ở đây ta chỉ tính đến sự khác nhau tương đối giữa việc chúng ta đã dự kiến bao nhiêu và đã hoàn thành được bao nhiêu trong thực tế".

Điều gì xảy đến với những story "gần hoàn thành" trong Sprint? Tại sao chúng ta không lấy một phần điểm của story đó để đưa vào tốc độ thực tế? Vâng, điều này nhấn mạnh đến thực tế của Scrum (và cả trong Phát triển phần mềm linh hoạt (Agile) lẫn phương pháp sản xuất tinh gọn nói chung (Lean Manufacturing[9]), đó là chỉ nói về những việc đã được thực sự hoàn thành, có thể chuyển giao, đã kết thúc. Giá trị của những thứ mới hoàn thành một nửa là số không (thực tế là bị bác bỏ). Xin hãy đọc cuốn sách "Managing the Design Factory" của Donald Reinertsen hay một trong những cuốn sách của Poppendieck[10] để biết rõ thêm về điều này.

Vậy thì chúng ta cần thông qua thủ thuật phức tạp gì để có tốc độ ước tính?

Một cách rất đơn giản để ước tính tốc độ là xem lại những gì mà nhóm đã triển khai trước đây. Tốc độ của họ trong một vài Sprint gần đây là gì? Sau đó giả định tốc độ bám theo khoảng đó cho Sprint tới.

Kỹ thuật này được biết với cái tên *Thời tiết của ngày hôm qua.* Nó chỉ phù hợp với những nhóm đã thực hiện được một số Sprint (khi đã có được số liệu thông kê) và những Sprint tiếp theo được hoàn thành với phương thức gần giống với những Sprint cũ, với cùng kích thước nhóm trong cùng điều kiện làm việc, v.v.. Điều này tất nhiên không phải lúc nào cũng được như vậy.

---

[9] Lean Manufacturing (hay Lean Production): sản xuất tinh gọn, có xuất xứ trực tiếp từ Hệ thống sản xuất Toyota (Toyota Production System), là một phương pháp được xây dựng dựa trên ý tưởng cơ bản về loại bỏ các dư thừa không cần thiết trong chu trình phát triển sản phẩm, tối ưu hóa luồng sản xuất để tạo ra các giá trị cao nhất cho khách hàng. Lean có ảnh hưởng rộng lớn trong các ngành công nghiệp, rất gần gũi với Agile và có ảnh hưởng lớn trong các hoạt động thực tiễn của những người thực hành Agile. Một trong các phương pháp Agile được chuyển đổi gần như trực tiếp từ Lean Manufacturing sang thế giới phần mềm, là Lean Software Develpoment.

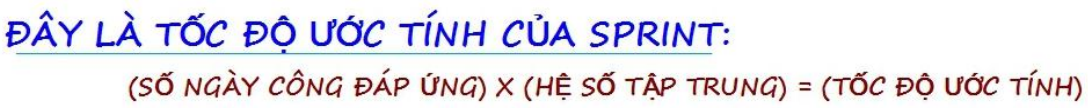
[10] Tom Poppendieck và Mary Poppendieck, các tác giả của loạt sách về Phát triển Phần mềm Tinh gọn (Lean Software Development).

Có một vấn đề phức tạp hơn là phải tính toán được nguồn lực cơ bản. Giả sử chúng tôi đang phải lên kế hoạch cho một Sprint kéo dài 3 tuần (15 ngày làm việc) với một nhóm 4 người. Lisa sẽ có 2 ngày nghỉ. Dave thì chỉ có thể làm việc với 50% thời gian và sẽ có 1 ngày nghỉ. Hãy liệt kê tất cả những điều đó…

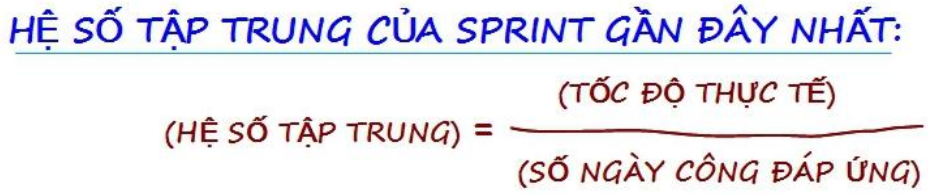
… 50 ngày công sẵn sàng cho Sprint này.

Giá trị trên có phải là tốc độ ước tính? Không! Do đơn vị ước tính của chúng tôi là điểm (story point), nên trong trường hợp này, đó mới chỉ là giá trị "số ngày công lý tưởng". Một ngày công lý tưởng là một ngày làm việc tập trung, thực sự hiệu quả, không bị làm phiền, điều này rất hiếm. Ngoài ra, chúng tôi còn phải tính đến những việc không nằm trong dự kiến nhưng vẫn phát sinh trong Sprint, ví dụ như nhân viên nghỉ việc vì ốm, v.v.

Do vậy, *tốc độ ước tính* của chúng tôi chắc chắn phải nhỏ hơn con số 50. Nhưng nhỏ hơn bao nhiêu? Chúng tôi sẽ dùng đến khái niệm "hệ số tập trung" (focus factor) để xác định điều đó.



"Hệ số tập trung" là một số ước lượng độ tập trung công việc của nhóm. Một hệ số tập trung thấp có nghĩa là nhóm có thể bị phân tâm bởi nhiều việc khác hoặc thời gian ước tính của họ là quá lạc quan.

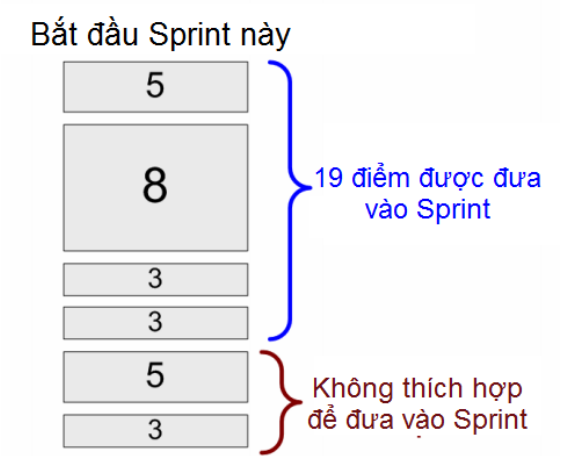


Cách tốt nhất để xác định hệ số tập trung phù hợp là hãy dựa vào Sprint gần nhất (hay tốt hơn là dựa trên mức bình quân của một số Sprint gần nhất).

Tốc độ thực tế là tổng các giá trị ước tính của tất cả các story được hoàn thành trong Sprint gần nhất.

Giả sử Sprint gần nhất có 18 điểm được hoàn thành với nhóm 3 thành viên bao gồm Tom, Lisa và Sam, làm việc trong 3 tuần với tổng số 45 ngày công. Bây giờ chúng tôi cần tính ra tốc độ ước tính cho Sprint tiếp theo. Để giải quyết những thứ phức tạp, nhóm có thêm thành viên mới gia nhập là Dave. Tính cả các kỳ nghỉ và một số thứ khác, chúng tôi có 50 ngày công cho Sprint tiếp theo.

Vì vậy, tốc độ ước tính của chúng tôi với Sprint tới là 20 điểm. Điều đó có nghĩa là nhóm nên thêm các story vào Sprint cho tới khi việc bổ sung đó xấp xỉ 20 điểm.



Trong trường hợp này, nhóm có thể chọn 4 story đứng đầu có tổng số điểm là 19, hoặc chọn 5 story với tổng số điểm là 24. Hãy bảo họ chọn giải pháp 4 story, do nó cho giá trị gần nhất với 20 điểm. Nếu không chắc chắn, hãy chọn ít story hơn.

Do 4 story có tổng số điểm là 19 nên tốc độ ước tính cuối cùng của họ cho Sprint này là 19.

Kỹ thuật "thời tiết ngày hôm qua" khá là dễ áp dụng, nhưng hãy vận dụng nó với chút ít điều chỉnh cho phù hợp. Với trường hợp Sprint gần nhất có kết quả kém vì hầu hết các thành viên trong nhóm phải nghỉ ốm một tuần, khi đó có thể an toàn để bạn giả định rằng tình trạng không may như thế sẽ không lặp lại và bạn có thể ước tính với hệ số tập trung cao hơn cho Sprint tiếp theo. Với trường hợp nhóm vừa cài đặt được hệ thống build liên tục (continuous build system[11]), bạn có thể tăng hệ số tập trung tới một giá trị tốt hơn. Trong trường hợp nhóm có thêm một thành viên mới, chúng ta cần giảm hệ số tập trung vì phải tốn thêm chi phí đào tạo anh ta.v.v..

Mỗi khi có thể, hãy xem xét một vài Sprint gần đây và lấy trung bình cộng để có được những ước lượng thực tế hơn.

Điều gì sẽ đến nếu như nhóm vừa mới được thành lập và bạn không có chút dữ liệu thống kê nào? Hãy xem lại "hệ số tập trung" của các nhóm khác có cùng hoàn cảnh.Làm gì nếu bạn không tìm ra được nhóm nào tương tự? Hãy ước đoán "hệ số tập trung". Một tin tốt là giá trị ước đoán đó chỉ áp dụng ở Sprint đầu tiên. Sau đó, bạn đã có được dữ liệu thống kê và bạn có thể tiếp tục tính toán và cải thiện giá trị các hệ số và tốc độ ước tính.

Thường thì "mặc định" hệ số tập trung sử dụng cho nhóm mới có giá trị vào khoảng 70%, đó là con số mà các nhóm của chúng tôi đạt được qua thời gian.

Chúng ta sẽ sử dụng kỹ thuật ước tính nào?

Tôi đã nói ở trên một số kỹ thuât: Trực giác, tính tốc độ dựa trên "tình hình ngày hôm qua" và tính tốc độ dựa trên số lượng ngày công khả dụng và hệ số tập trung ước tính.

Vậy chúng tôi sử dụng kỹ thuật nào?

Chúng tôi thường phối hợp tất cả các kỹ thuật để tìm được con số tương đối. Chúng không tốn nhiều thời gian.

Chúng tôi xem xét hệ số tập trung và tốc độ thực tế của Sprint gần nhất. Chúng tôi xem xét tổng nguồn lực sẵn sàng cho Sprint này và ước tính hệ số tập trung. Chúng tôi thảo luận về bất kỳ sự khác nhau giữa hai hệ số tập trung này và có những điều chỉnh cần thiết.

Một khi chúng tôi có được danh sách sơ bộ các story được đưa vào Sprint, tôi sử dụng kỹ thuật "Trực giác" để kiểm tra. Khi đó, tôi cũng yêu cầu nhóm tạm thời bỏ qua các con số trên và chỉ

[11] Các hệ tự động build liên tục, còn gọi là các hệ tích hợp liên tục (Continuous Integration), như Maven, Hudson, CruiseControl v.v., là các phần mềm tự động tải về mã nguồn và các tài nguyên cần thiết từ kho lưu trữ mã nguồn, biên dịch và xây dựng các gói thực thi (executables). Nhờ vào quá trình tự động hóa này, Nhóm phát triển không phải biên dịch và build thủ công, tiết kiệm được công sức trong phát triển phần mềm.

nghĩ về việc "cảm thấy" có đưa vào Sprint này được không. Nếu cảm thấy quá nhiều, chúng tôi sẽ loại bỏ một hoặc hai story, và ngược lại.

Cuối ngày làm việc hôm đó, mục tiêu của chúng tôi chỉ đơn giản là xác định những story nào sẽ được đưa vào Sprint. Hệ số tập trung, nguồn lực khả dụng và Tốc độ ước tính chỉ có ý nghĩa để đạt được các mục tiêu đó khi kết thúc Sprint.

Tại sao chúng tôi sử dụng các thẻ chỉ mục (Index Card)

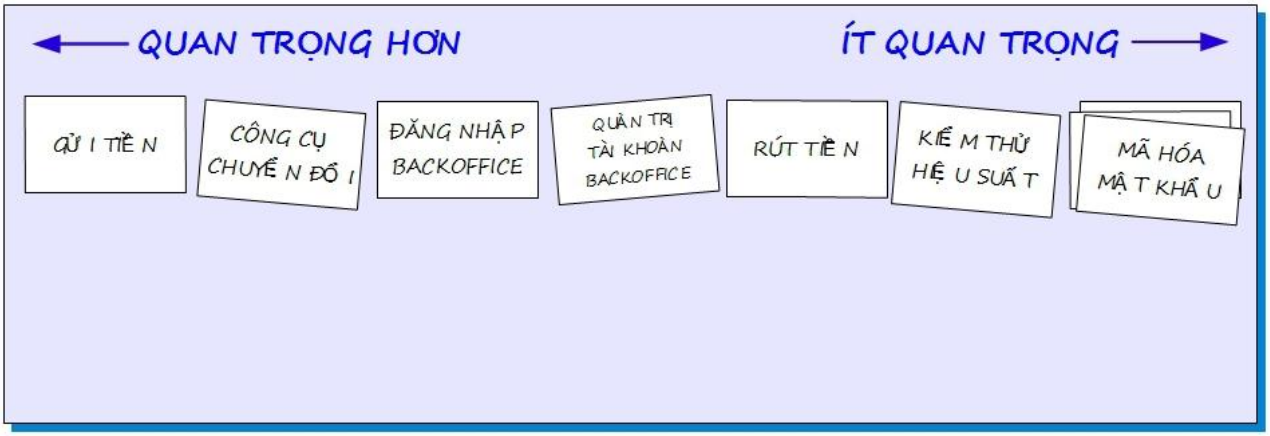
Tại hầu hết cuộc họp lập kế hoạch Sprint, được sử dụng để xử lý các story trong Product Backlog. Ước tính, đặt lại độ ưu tiên, phân loại và chia nhỏ các story, v.v.

Chúng tôi thực hiện điều này như thế nào?

Vâng, mặc định, các nhóm sẽ sử dụng máy chiếu, hiển thị backlog trên Excel, và một thành viên (thường là Product Owner hoặc Scrum Master) sẽ sử dụng bàn phím, duyệt lướt qua mỗi story và mời mọi người thảo luận. Cả nhóm và Product Owner sẽ thảo luận mức độ ưu tiên, thông tin chi tiết và tiến hành cập nhật trực tiếp trên Excel.

Nghe có vẻ rất tốt? Thực ra cũng không hẳn. Thường là rất tệ. Và điều tồi tệ hơn, nhóm thường không nhận ra cho đến khi gần kết thúc cuộc họp, khi đó, nhóm nhận ra rằng họ vẫn chưa thông qua được danh sách các story!

Một giải pháp tốt hơn nhiều là tạo ra các *thẻ chỉ mục* và *dán chúng lên tường* (hoặc một bảng lớn)

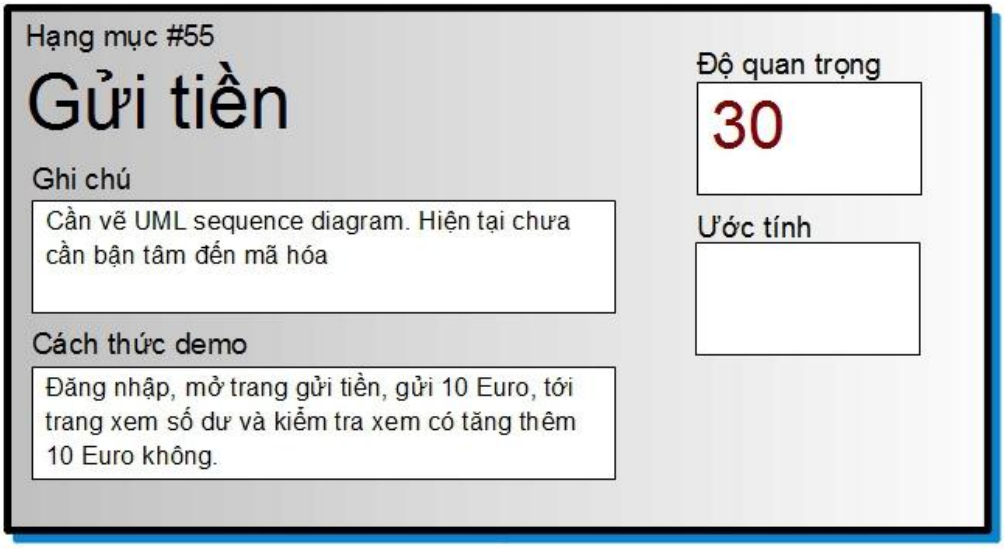


Điều này cho ta cái nhìn rất trực quan nếu so với máy tính và máy chiếu, bởi vì:

- Mọi người có thể đứng gần và đi lại xung quanh đó => họ sẽ nhìn thấy thường xuyên và lưu ý lâu hơn.

- Mọi người đều cảm bản thân mình được tham gia (không chỉ mỗi một anh chàng đang dùng bàn phím).
- Nhiều story có thể được chỉnh sửa đồng thời.
- Việc đánh giá lại mức độ ưu tiên rất đơn giản - chỉ việc di chuyển các thẻ chỉ mục.
- Sau cuộc họp, các chỉ mục có thể được giữ lại trong phòng làm việc của nhóm và sẽ được sử dụng như là bảng công việc trên tường (xem trang 59, "Chúng tôi xây dựng Sprint Backlog như thế nào").

Bạn có thể hoặc là tự tay viết *thẻ chỉ mục* hoặc (như chúng tôi thường làm) sử dụng một chương trình đơn giản để có thể in các thẻ trực tiếp từ Product Backlog.



TB. Chương trình (script) này có trên website của tôi: http://blog.crisp.se/henrikkniberg.

Lưu ý: sau cuộc họp lập kế hoạch Sprint, Scrum Master của chúng tôi thường cập nhật lại Product Backlog trong Excel, với bất cứ thay đổi nào đã được thực hiện trên các thẻ chỉ mục. Vâng, đây chỉ là một chút rắc rối về mặt thủ tục, nhưng chúng tôi thấy rằng hoàn toàn chấp nhận được nếu nhìn lại tính hiệu quả của các *thẻ chỉ mục* trong các cuộc họp lập kế hoạch Sprint.

Một lưu ý về trường "Độ quan trọng". Điều quan trọng này được mô tả trong Product Backlog trên Excel khi in ra. Có nó, các *thẻ chỉ mục* sẽ dễ dàng được sắp xếp theo mức ưu tiên của công việc (thường thì chúng tôi đặt các hạng mục quan trọng hơn ở bên trái, thấp hơn ở bên phải). Tuy nhiên, khi đã dán xong trên bảng, ta có thể không cần để ý đến mức quan trọng của nó nữa và thay vào

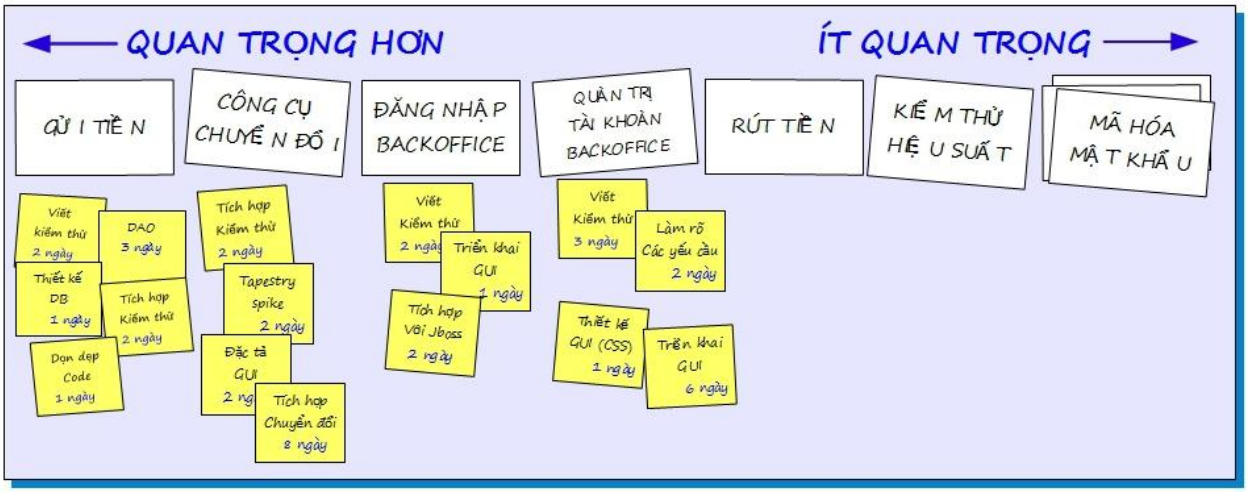
đó, nhìn vào thứ tự sắp xếp trên bảng để biết được độ quan trọng. Nếu Product Owner đã hoán đổi vị trí các thẻ đó, đừng lãng phí thời gian để cập nhật lại thông tin về độ quan trọng trên các tờ giấy.

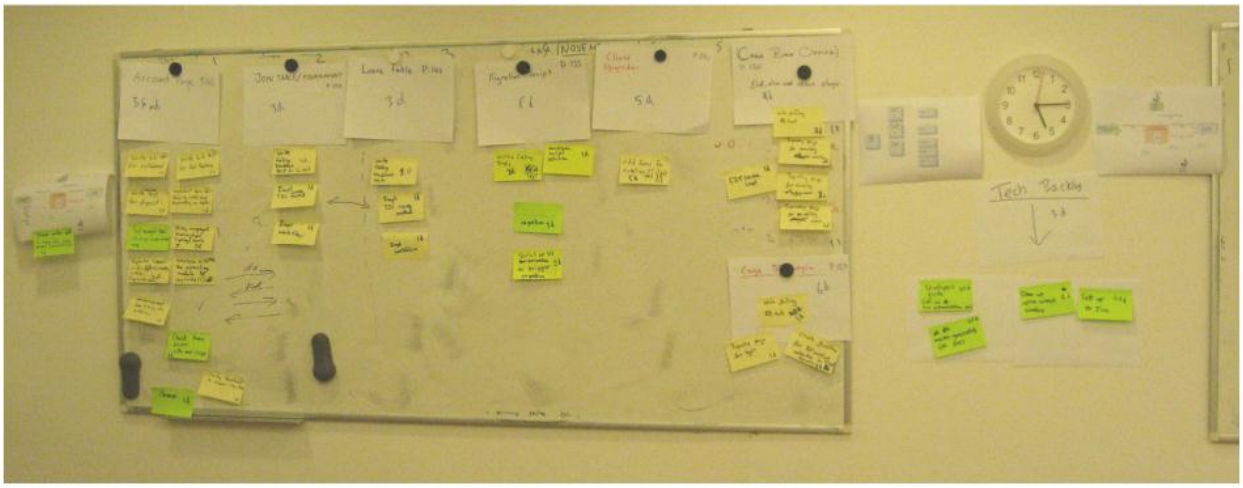
Chỉ cần cập nhật lại *mức* độ quan trọng của trong Product Backlog trên Excel sau cuộc họp.

Việc ước tính thời gian sẽ dễ hơn (và chính xác hơn) nếu ta chia các story thành các tác vụ. Trên thực tế, chúng tôi sử dụng thuật ngữ "hoạt động (activity)" vì ở Thụy Điển, từ "tác vụ (task)" có nghĩa là một cái gì đó *hoàn toàn* khác :o)

Điều này cũng cũng tốt và dễ dàng hơn để tiến hành với các *thẻ chỉ mục*. Bạn có thể chia nhóm làm việc thành từng cặp và cùng nhau chia nhỏ từng story một.

Cụ thể, chúng tôi thực hiện điều này bằng cách thêm vào các ghi chú nhỏ đính kèm dưới mỗi story, mỗi ghi chú là một công việc cần thực hiện.





Chúng tôi không cập nhật lại Product Backlog trên Excel để thêm vào những công việc được phân tách ở trên, có hai nguyên nhân:

- Các công việc được tác nhỏ thường không cố định, tức là chúng thường bị thay đổi hoặc được định nghĩa lại trong suốt Sprint. Do đó, sẽ rất khó để tổng hợp vào Product Backlog trên Excel.
- Product Owner không cần phải tham gia vào những công việc ở mức độ chi tiết như vậy.

Chỉ với các thẻ chỉ mục viết các story, những mẫu giấy viết các công việc được tách ra có thể được tái sử dụng trong Sprint Backlog (xem thêm trang 45 "Chúng tôi xây dựng Sprint Backlog như thế nào").

Định nghĩa “Hoàn thành”

Điều quan trọng là Product Owner và nhóm đồng thuận cho một định nghĩa rõ ràng về "hoàn thành". Một story được cho là hoàn thành khi tất cả mã nguồn đã được kiểm thử? Hoặc là chỉ khi được triển khai vào môi trường kiểm thử và được xác nhận bởi một nhóm kiểm thử tích hợp? Mỗi khi có thể, chúng tôi đều sử dụng định nghĩa hoàn thành "sẵn sàng để triển khai sản phẩm", nhưng đôi khi chúng tôi phải sử dụng định nghĩa hoàn thành "đã triển khai trên dịch vụ kiểm thử và sẵn sàng để kiểm thử chấp nhận".

Thời gian đầu, chúng tôi đã sử dụng danh sách kiểm tra chi tiết cho việc này. Nhưng bây giờ, chúng tôi chỉ cần nói "một story được hoàn thành chỉ khi kiểm thử viên trong nhóm Scrum chấp nhận". Điều đó có nghĩa là kiểm thử viên đảm bảo rằng Nhóm Phát triển đã hiểu được yêu cầu của Product Owner, và story “hoàn thành” đủ để vượt qua định nghĩa hoàn thành đã được chấp nhận.

Chúng tôi đã nhận ra rằng các story cần được xử lý theo cách khác nhau. Một story có tên là "Biểu mẫu truy vấn người dùng " cần được xử lý rất khác so với story có tên "Hướng dẫn vận hành". Trong tính năng thứ hai, định nghĩa "hoàn thành" chỉ đơn giản là "được chấp nhận bởi nhóm vận hành". Đó là lý do tại sao cảm giác chung thường tốt hơn so với danh sách kiểm tra mô phạm.

Nếu bạn thường nhầm lẫn về định nghĩa hoàn thành (như chúng tôi đã vướng phải lúc đầu), bạn nên có hẳn một trường "định nghĩa của hoàn thành" trên mỗi story.

Ước tính thời gian sử dụng Bộ bài Lập kế hoạch[12]

Việc ước tính là hoạt động của cả nhóm - mọi thành viên cần tham dự vào việc ước tính tất cả các story. Tại sao vậy?

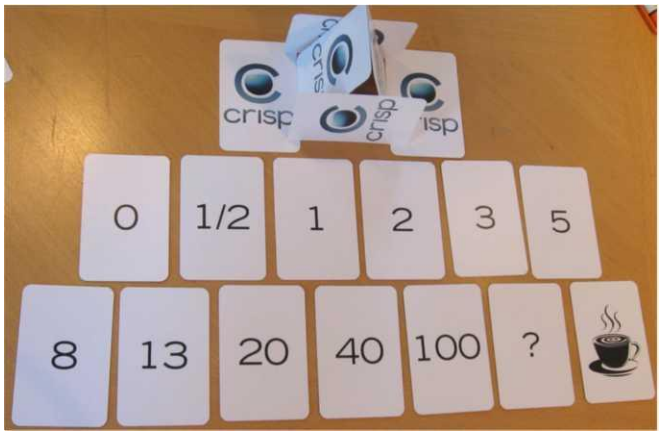
- Trong lúc lập kế hoạch, chúng tôi thường không biết chính xác ai sẽ làm phần việc nào trong story.
- Các story thường đòi hỏi vài người với một vài kỹ năng khác nhau (như thiết kế giao diện, lập trình, kiểm thử, v.v.).
- Để có một ước tính, mỗi thành viên trong nhóm cần hiểu biết story đó làm về cái gì. Với việc yêu cầu mọi người tham gia ước tính mỗi hạng mục, chúng tôi đã đảm bảo mọi thành viên đều hiểu rõ về hạng mục đó. Việc này tăng sự thông suốt giữa các thành viên để giúp đỡ nhau trong lúc thực hiện Sprint. Nó cũng tăng sự hiểu biết về những câu hỏi quan trọng cho những story sẽ làm trong giai đoạn sau.
- Khi yêu cầu mọi người tham gia ước tính một story nào đó, chúng tôi thường nhận thấy những sự đánh giá khác của các thành viên về cùng một story. Điều này tốt cho việc tìm hiểu sâu hơn và thảo luận sớm về nó.

Nếu bạn yêu cầu nhóm cung cấp một ước tính, thường thì thành viên hiểu rõ nhất về story đó sẽ là người đầu tiên nêu ý kiến. Thật không may, điều này sẽ ảnh hưởng rất nhiều đến ước tính của những người khác.

---

[12] Planning Poker, còn gọi là Scrum Poker, là một sáng tạo của Mike Cohn – một lý thuyết gia và là nhà thực hành Agile nổi tiếng. Đó là một bộ bài với các lá bài được đánh số theo dãy Fibonacci (0,1, 2,3,5,8 ..) được dùng trong ước tính tương đối và lập kế hoạch. Xem thêm trên http://hanoiscrum.net về cách dùng, cũng như tải về một bộ bài tiêu chuẩn để in ra và dùng cho nhóm của bạn.

Có một kỹ thuật tuyệt vời để tránh gặp phải vấn đề này - nó được gọi là Planning Poker (theo tôi biết thì được đề xuất bởi Mike Cohn)



Mỗi thành viên trong nhóm sẽ nhận được một bộ bài 13 lá như hình trên. Mỗi khi phải ước tính một story nào đó, mỗi thành viên sẽ chọn một lá bài có ghi giá trị mà mình ước tính (tính theo điểm) và đặt úp xuống trên bàn. Khi tất cả các thành viên trong nhóm đã chọn xong lá bài của mình, các lá bài sẽ được lật lên cùng một lúc. Theo cách này, mỗi thành viên buộc phải đánh giá theo suy nghĩ của chính mình thay vì bị ảnh hưởng bởi ước tính của người khác.

Nếu có sự khác biệt lớn giữa hai giá trị ước tính, nhóm sẽ thảo luận về những khác biệt đó và cố gắng tạo ra một cái nhìn chung về những việc phải làm với story đang xem xét. Họ có thể phải chia ra các công việc nhỏ. Sau đó, nhóm thực hiện ước tính lại. Vòng lặp như vậy lặp đi lặp lại cho đến khi các ước tính tiến gần tới giá trị chung, tức là khi mà tất cả đánh giá cho story đó *xấp xỉ* như nhau.

Một điều quan trọng cần nhắc các thành viên là họ đang ước tính cho tổng khối lượng công việc cần làm cho story đó, chứ không phải là đánh giá chỉ "phần việc của họ". Kiểm thử viên không nên chỉ ước tính cho riêng phần kiểm thử.

Lưu ý rằng các con số có thể không liên tục. Ví dụ như không có giá trị nào nằm giữa 40 và 100. Tại sao vậy?

Điều này tránh cho những cảm giác sai lầm về tính chính xác khi đưa ra những ước tính lớn về thời gian. Nếu một story được ước tính vào khoảng 20 điểm, không quá cần thiết để thảo luận rằng nó lên là 20, 18 hay 21. Tất cả chúng ta đều hiểu rằng đó là một story phức tạp, mất nhiều công sức và khó để đưa ra ước tính chính xác hơn. Do đó, 20 là con số có thể “chơi” được.

Chúng ta có cần ước tính chi tiết hơn không? Chia nhỏ các story thành những story nhỏ hơn và ước tính cho những story nhỏ đó!

Không, bạn không thể làm theo kiểu cộng hai con số 5 và 2 để thành 7. Bạn phải chọn hoặc là 5, hoặc là 8, không có số 7 ở đây.

Một số lá bài đặc biệt cần lưu ý:

- 0 = "story này đã hoàn thành" hay "story này quá đơn giản, chỉ mất vài phút để hoàn thành".
- ? = "Tôi thực sự không hiểu gì về nó. Không có gì cả."
- Biểu tượng cốc cà phê = "Tôi nghĩ là mình quá mệt rồi. Nghỉ ngơi một chút nhé."

Phân loại các story

Điều tồi tệ nhất xảy ra khi nhóm đang thuyết trình đầy tự hào về một tính năng mới tại buổi demo Sprint, và Product Owner cau mày nói "vâng, tốt đấy, nhưng đó *không phải là thứ tôi yêu cầu*!"

Làm sao để đảm bảo những gì Product Owner hiểu cũng là những gì nhóm hiểu về mỗi story? Hoặc mỗi thành viên trong nhóm có cùng sự hiểu biết giống nhau về mỗi story? Vâng, bạn không thể. Tuy nhiên, có một số kỹ thuật đơn giản để xác định những hiểu lầm thô thiển nhất. Kỹ thuật đơn giản nhất là đảm bảo rằng tất cả các trường thông tin cần thiết của mỗi story được viết ra đầy đủ (cụ thể hơn, là với mỗi story có mức độ ưu tiên cao được quan tâm trong Sprint này).

Ví dụ 1:

Nhóm phát triển và Product Owner rất hài lòng với kế hoạch Sprint và chuẩn bị kết thúc cuộc họp. Scrum Master nói: “Đợi một chút, story ‘Thêm mới người dùng’ vẫn chưa có ước tính. Hãy ước tính nó đã!” Sau một vài vòng chơi Poker, nhóm đạt được đồng thuận 20 điểm cho story đó, nhưng Product Owner đứng dậy ngay “Cái gì cơ?!”. Sau vài phút thảo luận nóng, vấn đề được phát hiện ra, nhóm đã hiểu sai phạm vi của story ‘Thêm mới người dùng’, họ nghĩ rằng nó sẽ là ‘một giao diện web đẹp với các chức năng thêm mới, di chuyển, xóa và tìm kiếm người dùng’, trong khi Product Owner lại hiểu ‘thêm mới người dùng đơn giản thực hiện bằng tay thông qua câu lệnh SQL trong cơ sở dữ liệu’. Họ đã phải ước tính lại với 5 điểm cho story này.

Ví dụ 2:

Nhóm Phát triển và Product Owner rất hài lòng với kế hoạch Sprint và chuẩn bị kết thúc cuộc họp. Scrum Master nói: “Đợi một chút, story ‘Thêm mới người dùng’, sẽ được demo như thế nào?”. Một số tiếng xì xào được phát ra và sau một phút, một số thành viên đứng dậy phát biểu “Vâng, đầu tiên chúng ta cần đăng nhập vào website, và sau đó...”, và Product Owner liền ngắt lời “đăng nhập vào website?! Không, không, không, chức năng này không cần thiết chút nào, chỉ cần người quản trị thực hiện vài lệnh SQL đơn giản là được”.

Mô tả "cách thức demo" cho story có thể (và nên) *thật ngắn*. Nếu không, bạn sẽ không thể kết thúc cuộc họp lập kế hoạch Sprint đúng giờ. Đó là mô tả mức đơn giản trong tiếng Anh về việc làm thế nào để thực hiện các kịch bản kiểm thử đặc trưng nhất. "Làm điều này, rồi làm việc kia, sau đó xác minh điều này".

Tôi nhận thấy rằng mô tả đơn giản này thường phát sinh những hiểu lầm lớn về phạm vi của một story. Phát hiện vấn đề sớm cũng tốt, đúng không?

Phân tách các story thành những story nhỏ hơn

Trong ước tính, các story không nên quá nhỏ hoặc quá lớn. Nếu bạn có một loạt các story với 0,5 điểm, bạn có thể đang là nạn nhân của quản lý vi mô. Mặt khác, một story có 40 điểm có nguy cơ cao trở thành "*hoàn thành một phần*", cái đó không đem lại giá trị cho công ty và chỉ gia tăng chi phí quản lý. Hơn nữa, nếu *tốc độ ước tính* của là 70 và có hai tính năng ưu tiên cao nhất đều là 40 điểm thì việc lập kế hoạch sẽ rất khó khăn. Bạn phải lựa chọn hoặc dưới mức chỉ tiêu (chỉ chọn một story) hoặc vượt mức chỉ tiêu (chọn cả hai story).

Tôi nhận thấy rằng luôn có thể phân tách một story lớn thành những story nhỏ hơn. Chỉ cần bảo đảm những story nhỏ đó vẫn có giá trị nghiệp vụ khi chuyển giao.

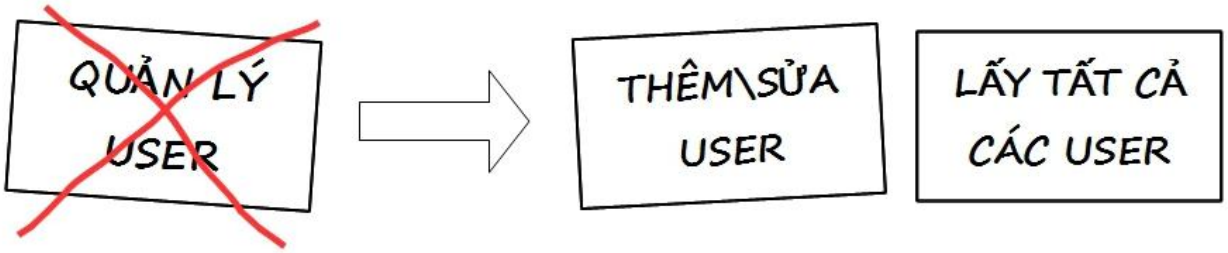
Thường thì chúng tôi cố gắng đưa những story về khoảng 2 - 8 ngày công. Một nhóm điển hình thường có tốc độ vào khoảng 40-60 điểm, điều này cho phép chúng tôi xử lý khoảng 10 story với mỗi Sprint. Đôi khi ít hơn với 5 story và đôi khi nhiều hơn với 15 story. Đó cũng chính là số lượng *thẻ chỉ mục* mà chúng tôi có thể quản lý được.

Phân tách story thành các tác vụ

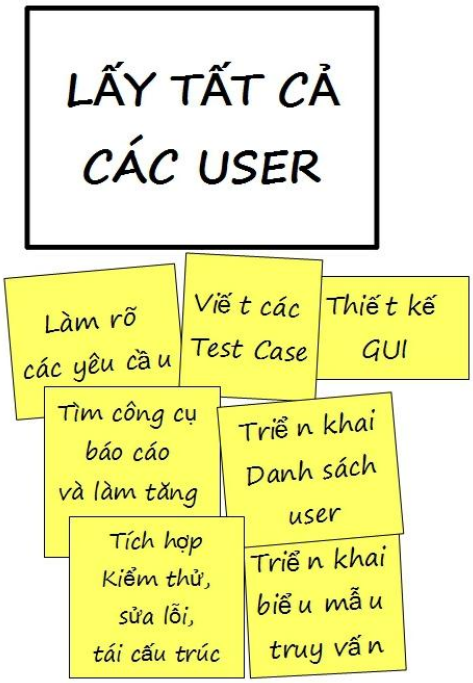
Chờ một chút, điều khác biệt giữa "task" (hạng mục công việc, tác vụ, công việc) và "story" là gì? Câu hỏi này rất quan trọng.

Khá dễ để phân biệt hai khái niệm này. Story là những thành phần của sản phẩm chuyển giao được mà Product Owner quan tâm. Tác vụ không phải là thành phần chuyển giao, hoặc là những thành phần mà Product Owner không quan tâm.

Ví dụ về việc tách một story thành các story nhỏ hơn:



Ví dụ về việc tách một story thành các tác vụ:



Dưới đây là một số ghi nhận thú vị:

- Các Nhóm Scrum mới thường không muốn dành thời gian để phân tách story thành các tác vụ như ví dụ minh họa ở trên. Một số còn cảm thấy hướng tiếp cận này giống như phương pháp thác nước (waterfall).
- Khi Nhóm hiểu rõ về các story sẽ giúp việc phân tách chúng thành các tác vụ dễ dàng hơn.
- Việc phân tách thường làm gia tăng các công việc do đó dẫn đến ước tính thời gian có thể tăng lên, và cũng giúp việc lập kế hoạch Sprint sát với thực tế hơn.

- Phân tách công việc còn giúp cho các buổi Họp Scrum Hằng ngày hiệu quả hơn (Xem nội dung "Chúng tôi tiến hành Họp Scrum Hằng ngày như thế nào" ở trang 48).
- Thậm chí, nếu việc phân tách công việc không chính xác và có thể làm thay đổi công việc khi triển khai, thì những lợi ích ở trên vẫn được chấp nhận.

Vì vậy, chúng tôi cố gắng để lập khung thời gian phù hợp cho Sprint, nhưng nếu vượt quá thời gian cho phép chúng tôi sẽ bỏ chúng ra khỏi Sprint (Xem phần "Chúng ta cần dừng ở đâu" bên dưới).

Lưu ý – chúng tôi triển khai TDD (Phát triển Hướng Kiểm thử) nghĩa là công việc đầu tiên với hầu hết các story là "viết một kiểm thử cho kết quả sai" và công việc cuối cùng là tái cấu trúc (refactor = tăng tính dễ đọc của mã nguồn và loại bỏ mã nguồn lặp).

Xác định thời gian và địa điểm cho buổi Họp Scrum Hằng ngày

Một công việc thường bị bỏ quên trong các buổi Họp Kế hoạch Sprint đó là "xác định thời gian và địa điểm Họp Scrum Hằng ngày". Nếu không thực hiện việc này Sprint của bạn sẽ dễ có một khởi đầu tồi tệ. Buổi Họp Scrum Hằng ngày đầu tiên là buổi khởi động cần thiết, ở đó mọi thành viên sẽ quyết định nơi khởi đầu công việc của mình.

Tôi thường thích họp vào buổi sáng hơn. Nhưng, tôi phải thừa nhận rằng chúng tôi không thực sự cố gắng để thử tiến hành các cuộc Họp Scrum Hằng ngày vào buổi trưa hoặc buổi chiều.

Những nhược điểm của việc Họp Scrum Hằng ngày vào buổi chiều: khi bạn tiến hành công việc vào buổi sáng, bạn phải cố gắng nhớ những gì bạn nói với mọi người vào chiều ngày hôm qua về những công việc mà bạn sẽ làm hôm nay.

Những nhược điểm của việc Họp Scrum Hằng ngày vào buổi sáng: khi bạn bắt đầu công việc vào buổi sáng, bạn phải cố gắng nhớ những gì bạn đã làm hôm qua để bạn còn báo cáo nó.

Theo tôi thì nhược điểm đầu là tồi hơn, vì hầu hết những điều quan trọng lại là những thứ mà bạn *sẽ phải làm* chứ không phải những thứ mà bạn *đã làm*.

Phương thức mặc định của chúng tôi là chọn thời gian sớm nhất được chấp nhận bởi tất cả các thành viên trong Nhóm. Thường là 9:00, 9:30, hoặc 10:00. Điều quan trọng nhất đó là thời điểm mà mọi thành viên trong nhóm đều thực sự chấp nhận.

Chúng ta cần dừng ở đâu

OK, vậy là thời gian đang trôi đi. Tất cả những thứ mà chúng ta muốn làm trong suốt thời gian họp kế hoạch Sprint, chúng ta cần cắt bỏ những gì khi mà thời gian họp đã hết?

À, tôi sử dụng danh sách ưu tiên dưới đây:

Ưu tiên 1: Mục tiêu của Sprint và ngày demo. Đây là điều tối thiểu bạn cần có để bắt đầu một Sprint. Nhóm phải có mục tiêu, ngày kết thúc, và họ cần làm việc đúng với Product Backlog. Vậy thì thật là tệ, vâng, và bạn nên xem xét để tổ chức một cuộc họp lập kế hoạch mới vào ngày mai, nhưng nếu bạn thực sự cần phải bắt đầu Sprint này thì bạn có thể vẫn tiến hành được. Mặc dù vậy, thành thật mà nói, tôi chưa bao giờ thực sự phải bắt đầu Sprint với thông tin ít ỏi như thế.

Ưu tiên 2: Liệt kê ra các story mà Nhóm phải thực hiện trong Sprint này.

Ưu tiên 3: Ước tính cho tất cả các story trong Sprint.

Ưu tiên 4: "Cách thức demo" được điền cho mỗi story trong Sprint.

Ưu tiên 5: Tính toán tốc độ và tài nguyên cho Sprint của bạn thực sự đã được kiểm tra. Bổ sung danh sách các thành viên của Nhóm và những cam kết của họ (nếu không bạn không thể tính toán được tốc độ).

Ưu tiên 6: Xác định thời gian và địa điểm Họp Scrum Hằng ngày. Điều này chỉ mấy một chút thời gian để quyết định, nhưng nếu các bạn đã hết thời gian cho việc đó thì Scrum Master đơn giản có thể quyết định điều này sau buổi họp và gửi email cho mọi người.

Ưu tiên 7: Tách các story thành các tác vụ. Việc phân tách này có thể được thực hiện hằng ngày kết hợp với các buổi Họp Scrum Hằng ngày, nhưng nó sẽ có thể làm gián đoạn tiến độ của Sprint.

Các story kĩ thuật (tech stories)

Đây là một vấn đề phức tạp: các story kĩ thuật. Hoặc các hạng mục phi chức năng hay bất kỳ thứ gì bạn muốn gọi tên.

Tôi đang đề cập đến những thứ cần phải được thực hiện nhưng không có khả năng đem lại, không có liên quan trực tiếp tới bất kỳ một story cụ thể nào, và không có giá trị trực tiếp với Product Owner.

Chúng tôi gọi chúng là "story kĩ thuật" (tech story).

Ví dụ:

- **Cài đặt máy chủ build liên tục**

  Tại sao lại cần phải làm việc này: bởi vì nó tiết kiệm khối lượng lớn thời gian cho các nhà phát triển phần mềm và giảm thiểu phát sinh những vấn đề lớn khi tích hợp ứng dụng ở cuối phân đoạn.

- **Viết mô tả tổng quan thiết kế hệ thống**

  Tại sao lại cần phải làm việc này: Bởi vì các nhà phát triển có thể quên thiết kế tổng thể và dẫn đến viết những mã nguồn không phù hợp. Cần có một tài liệu mô tả "bức tranh lớn" về hệ thống để giúp mọi người tuân thủ cùng một thiết kế.

- **Cải tiến tầng DAO** (Data Access Object)

  Tại sao lại cần phải làm việc này: Bởi vì tầng DAO thực sự lộn xộn làm mất nhiều thời gian của mọi người do nhầm lẫn và gây ra những lỗi không cần thiết. Làm sạch mã nguồn sẽ tiết kiệm thời gian cho nhà phát triển và tăng cường sức mạnh cho hệ thống.

- **Nâng cấp Jira** (phần mềm kiểm soát lỗi)

  Tại sao lại cần phải làm việc này: Phiên bản hiện hành có nhiều lỗi và chạy chậm, nâng cấp sẽ giúp các bạn tiết kiệm thời gian.

Có những story mang ý nghĩa thông thường không? Hoặc chúng là những tác vụ không có kết nối với bất cứ một story cụ thể nào? Ai là người đặt độ ưu tiên cho chúng? Product Owner có nên tham gia vào công việc này?

Chúng tôi đã tiến hành rất nhiều các cách thức khác nhau để quản lý những story kĩ thuật. Chúng tôi đã thử xử lý chúng như những story thuộc phân lớp thứ nhất, giống như bất kỳ những story khác. Điều này là không tốt, khi Product Owner thiết lập ưu tiên trong Product Backlog, nó giống như việc so sánh các quả táo với các quả cam. Trong thực tế, vì mục đích minh bạch, các story "công nghệ" thường nhận được độ ưu tiên thấp với mô-típ kiểu như "vâng thưa các vị, tôi chắc chắn một dịch vụ build liên tục là rất quan trọng, nhưng hãy cứ xây dựng cho chúng tôi các tính năng về quản lý doanh thu trước đã? Sau đó các vị có thể bổ sung những viên kẹo công nghệ của mình cũng vẫn được mà?"

Trong một số trường hợp, Product Owner đúng, nhưng thường thì không. Chúng tôi đã kết luận rằng Product Owner không phải lúc nào cũng đủ điều kiện để tạo ra sự cân bằng. Vì vậy sau đây là những gì chúng tôi làm:

- Cố gắng tránh những story "công nghệ". Cố tìm cách để đưa một story "công nghệ" thành một story thông thường với giá trị nghiệp vụ vừa phải. Với cách đó Product Owner có cơ hội tốt hơn để tạo những sự cân bằng một cách chính xác.
- Nếu chúng tôi không thể chuyển một story "công nghệ" thành một story thông thường, thì xem xét đến việc thực hiện chúng như một hạng mục công việc trong một story khác. Ví dụ

"cải tiến tầng DAO" có thể được xem là một task nằm trong story "chỉnh sửa user", vì nó có liên quan đến tầng DAO.

- Nếu cả hai cách trên đều không thực hiện được, hãy định nghĩa chúng là một story "công nghệ", và đặt riêng chúng trong một danh sách. Hãy để Product Owner trông thấy chúng nhưng không được phép thay đổi. Sử dụng các *"hệ số tập trung"* và *"tốc độ ước tính"* để đàm phán với Product Owner và khéo léo đưa thêm Sprint thời gian để triển khai các story "công nghệ" đó.

Xem xét ví dụ sau (một cuộc đối thoại tương tự điều này đã xảy ra trong một cuộc Họp Kế hoạch Sprint của chúng tôi).

- **Nhóm:** "Chúng tôi có một số công nghệ bên trong cần phải được triển khai. Chúng tôi muốn dùng 10% thời gian để thực hiện chúng, nghĩa là giảm các thành phần trọng tâm từ 75% xuống 65%. Điều đó có được không?"
- **Product Owner:** "Này, không được đâu! Chúng ta không có thời gian cho việc đó!"
- **Nhóm:** "Thôi được, anh hãy xem lại Sprint trước (tất cả mọi người đều quay nhìn vào tốc độ được vẽ nguệch ngoạc trên bảng). Chúng ta đã ước tính tốc độ là 80, và tốc độ thực sự là 30, đúng không?"
- **Product Owner:** "Chính xác! Vì vậy chúng ta không có thời gian cho việc triển khai mấy thứ công nghệ đó! Điều chúng ta cần là các tính năng mới!"
- **Nhóm:** "Được rồi, nguyên nhân dẫn tới tốc độ của chúng ta tồi tệ như vậy là bởi vì chúng ta tốn quá nhiều thời gian để cố gắng tạo ra các bản phát hành thống nhất cho vieecj kiểm thử".
- **Product Owner:** "Đúng rồi, sao nữa?"
- **Nhóm:** "Tốt, tốc độ của chúng ta sẽ tiếp tục tồi tệ như vậy nếu chúng ta không làm gì để cải thiện nó."
- **Product Owner:** "Đồng ý, sao nữa?"
- **Nhóm:** "Vì vậy chúng tôi đề nghị rằng chúng ta sẽ dành khoảng 10% thời gian của Sprint này để cài đặt một máy chủ build liên tục và một số thứ khác trợ giúp công việc tích hợp. Điều này có thể sẽ làm tăng tốc độ lên ít nhất là 20% cho Sprint này và các Sprint sau đó!"
- **Product Owner:** "Ồ thật chứ? Tại sao chúng ta không làm điều này từ Sprint trước?!"

- **Nhóm:** "À… bởi vì anh không muốn chúng tôi làm…"
- **Product Owner:** "Ồ, ừ, tốt thôi, đây là một ý tưởng tốt, triển khai luôn đi!"

Dĩ nhiên, một lựa chọn khác đó là để Product Owner đứng ngoài hoặc đưa cho anh ta một hệ số tập trung không có khả năng thương lượng. Nhưng chẳng có lý do gì để không cố gắng đạt được sự đồng thuận trước tiên.

Nếu Product Owner là một thành viên có thẩm quyền (và chúng tôi may mắn có được điều này) tôi khuyên hãy để anh ấy có cơ hội thể hiện khả năng của mình và hãy để anh ấy thiết lập các ưu tiên tổng thể. Minh bạch là một trong những giá trị cốt lõi của Scrum, đúng vậy không?

Hệ thống Theo dõi lỗi và Product Backlog

Đây là một vấn đề đòi hỏi sự khôn khéo. Excel là một công cụ tốt dành cho Product Backlog. Nhưng bạn vẫn cần một hệ thống theo dõi lỗi (bug tracking system), và Excel không có khả năng thực hiện được điều này. Chúng tôi sử dụng Jira.

Vậy chúng tôi làm thế nào để đưa vấn đề sử dụng Jira vào cuộc họp lập kế hoạch Sprint? Theo tôi chẳng cần làm gì cả, cứ lờ điều đó đi và tập trung vào các story.

Chúng tôi đã thử một vài chiến lược sau:

- Product Owner in ra những hạng mục có độ ưu tiên cao nhất trong Jira, mang chúng tới cuộc Họp Kế hoạch Sprint, và dán lên tường cùng với các story (do đó mặc thể hiện việc so sánh độ ưu tiên của các hạng mục đó với các story khác).
- Product Owner tạo ra các story tham chiếu tới các hạng mục trong Jira. Ví dụ "Chỉnh sửa những lỗi nghiêm trọng xảy ra khi gửi báo cáo tới bộ phận hành chính, Jira-124, Jira-126, và Jira-180".
- Sửa lỗi được coi là việc nằm ngoài Sprint, tức là một Nhóm giữ hệ số tập trung ở mức đủ thấp (ví dụ 50%) để đảm bảo rằng họ có thời gian dành cho việc sửa các lỗi. Sau đó chỉ cần giả định rằng Nhóm sẽ mất một khoảng thời gian nhất định trong mỗi Sprint dành cho việc sửa các lỗi được ghi nhận trong Jira.
- Đưa Product Backlog vào Jira (thay cho Excel). Coi các lỗi như bất kỳ một story nào đó.

Thực sự chúng tôi không đánh giá chiến lược nào là tốt nhất; trong thực tế nó tùy thuộc vào các Nhóm khác nhau và mỗi Sprint khác nhau. Mặc dù vậy tôi thường có thiên hướng đi theo chiến lược thứ nhất. Chiến lược đó vừa hay vừa đơn giản.

Họp Kế hoạch Sprint đã đến hồi kết!

Chà, tôi không bao giờ nghĩ rằng chương này là về các cuộc họp lập kế hoạch cho Sprint lại dài như thế! Tôi đoán rằng nó phản ánh quan điểm của tôi về việc Họp Kế hoạch Sprint là sự kiện quan trọng nhất mà bạn cần làm khi triển khai Scrum. Sử dụng nhiều sự nỗ lực để đạt được những thứ đúng đắn, và sẽ dễ dàng hơn cho các công việc sau này.

Cuộc họp lập kế hoạch Sprint sẽ thành công nếu mọi người (tất cả các thành viên của Nhóm và Product Owner) vui vẻ khi rời khỏi cuộc họp, và mỉm cười khi tỉnh dậy vào mỗi buổi sáng, và hoan hỉ trong buổi Họp Scrum Hằng ngày.

Khi đó, tất nhiên tất cả mọi thứ vẫn có thể đi xuống một cách tồi tệ, nhưng ít nhất các bạn không thể đổ lỗi cho kế hoạch Sprint :o)

5

Chúng tôi truyền thông trong các Sprint như thế nào

Giữ cho toàn bộ công ty biết về những gì đang được làm là rất quan trọng. Nếu không mọi người có thể sẽ phàn nàn, thậm chí tệ hơn là có những nhận định sai lầm về những gì đang diễn ra. Chúng tôi sử dụng một "trang thông tin về Sprint" cho việc này.

Nhóm Jackass, Sprint 15

Mục tiêu Sprint

- Bản Beta sẵn sàng phát hành!

Sprint Backlog (ước tính trong ngoặc tròn)

- Gửi tiền (3)
- Công cụ chuyển đổi (8)
- Đăng nhập dành cho hành chính(5)
- Quản trị người dùng (5)

Tốc độ ước tính: 21

Lịch làm việc

- Thời gian của Sprint: từ 06/11/2006 đến 24/11/2006
- Họp Scrum hàng ngày: 9:30 - 9:45, tại phòng làm việc của Nhóm
- Demo Sprint: 24/11/2006, 13:00, tại quán ăn tự phục vụ

Nhóm

- Jim
- Erica (Scrum Master)
- Tom (75%)
- Eva
- John

Đôi khi chúng tôi cũng bổ sung thêm thông tin về cách thức tốt nhất dành cho việc demo một story.

Sau buổi họp kế hoạch Sprint, Scrum Master tạo trang này, sau đó đưa thông tin này lên wiki, và gửi lại cho toàn bộ công ty sớm nhất có thể.

Tiêu đề: Sprint 15 của Nhóm Jackass đã bắt đầu

Chào mọi người! Nhóm Jackass hiện tại đã bắt đầu làm Sprint 15. Mục tiêu của chúng tôi là demo một bản beta sẵn sàng để phát hành vào ngày 24/11.

Xem chi tiết thông tin của sprint tại:
http://wiki.mycompany.com/jackass/sprint15

Chúng tôi cũng có một trang "bảng điều khiển" trên wiki, nó sẽ liên kết với tất cả các Sprint đang được thực hiện.

Bảng điều khiển của doanh nghiệp

Các Sprint đang được thực hiện

- Nhóm X sprint 15
- Nhóm Y sprint 12
- Nhóm Z sprint 1

Thêm vào đó, Scrum Master sẽ in ra những trang thông tin của Sprint và dán chúng lên tường bên ngoài phòng làm việc của nhóm. Vì vậy bất kỳ ai đi qua đó cũng có thể nhìn thấy thông tin về Sprint đó và biết được nhóm nào đang làm việc gì. Nếu có thêm thông tin về thời gian và địa điểm Họp Scrum Hằng ngày và demo Sprint, anh ta sẽ biết nơi cần đến để biết thêm những thông tin khác.

Khi Sprint gần kết thúc, Scrum Master nhắc nhở mọi người về việc sắp tới ngày demo:

Tiêu đề: Demo Sprint của nhóm Jackass vào ngày mai lúc 13:00 tại quán ăn.

Chào mọi người! Các bạn được mời đến tham dự buổi demo Sprint lúc 13:00 tại quán ăn vào ngày mai (Thứ Sáu). Chúng tôi sẽ chạy thử bản beta sắp được phát hành.

Xem chi tiết thông tin của Sprint tại:
http://wiki.mycompany.com/jackass/sprint15

Làm như vậy, chắc chắn sẽ không có ai nói rằng mình không biết việc gì đang diễn ra.

7

Chúng tôi xây dựng Sprint Backlog như thế nào

Bạn đã làm công việc này khá lâu rồi? Úi chà, tốt đấy.

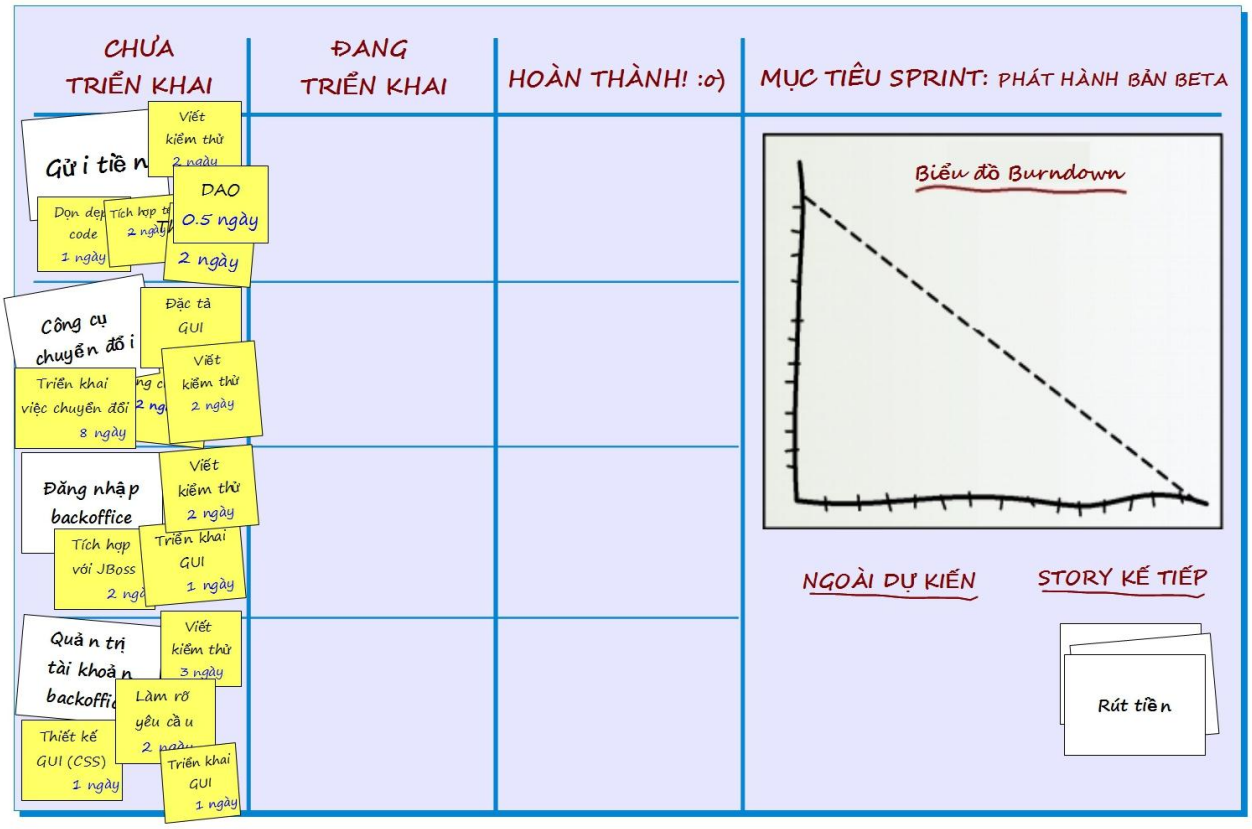
Bây giờ ta đã tiến hành xong buổi họp kế hoạch Sprint và đã kết luận rõ ràng công việc trong Sprint mới, đây là thời điểm để Scrum Master tạo một Sprint Backlog. Việc này cần được hoàn thành sau buổi họp kế hoạch Sprint, nhưng phải xong trước buổi họp Scrum Hằng ngày đầu tiên.

Khuôn mẫu cho Sprint Backlog

Chúng tôi đã thử nghiệm với nhiều định dạng khác nhau cho Sprint Backlog, bao gồm cả Jira, Excel, và bảng công việc treo trên tường. Hầu hết khi mới sử dụng Excel, có rất nhiều mẫu (template) Excel sẵn có cho Sprint Backlog, bao gồm cả biểu đồ Burndown được tự động sinh ra và một số thứ giống như vậy. Tôi có thể nói rất nhiều về cách chúng tôi cải thiện Sprint Backlog trên Excel. Nhưng tôi sẽ không làm như vậy ở đây, thậm chí là chỉ mộtví dụ.

Thay vì việc đó, tôi sẽ mô tả chi tiết những gì chúng tôi đã tìm thấy để có được một định dạng hiệu quả nhất cho Sprint Backlog – đó là một bảng công việc trên tường!

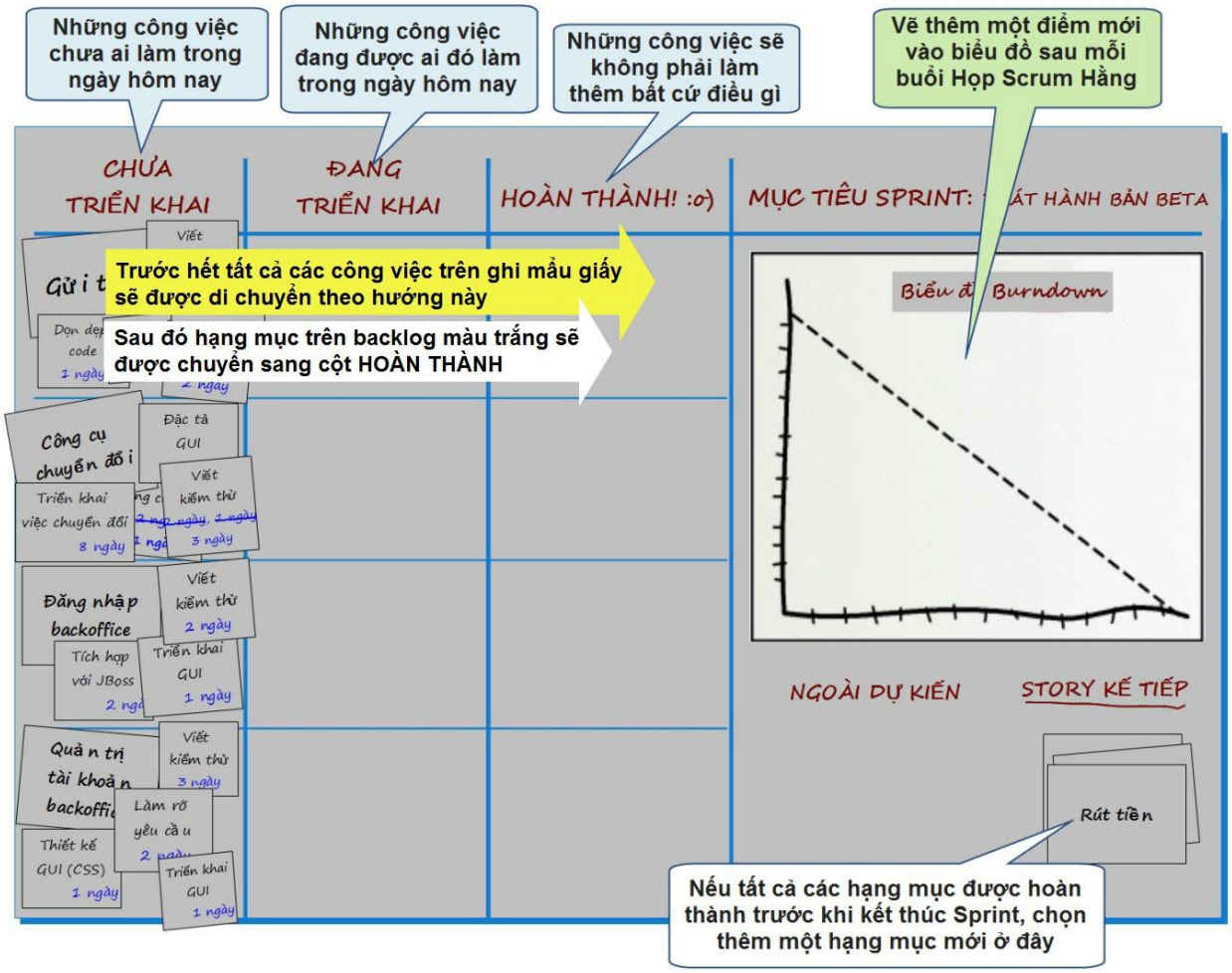
Tìm kiếm một bức tường lớn đang không sử dụng hoặc chứa những thứ chẳng mấy hữu ích như biểu tượng của công ty, biểu đồ cũ hoặc những bức vẽ xấu. Làm sạch bức tường (hãy xin phép để làm điều này nếu bạn không có thẩm quyền). Dán lên đó một tờ giấy có khổ lớn (ít nhất là 2x2 mét, hoặc 3x2 mét cho một Nhóm với nhiều thành viên). Sau đó làm tiếp như sau:



Tất nhiên là bạn có thể sử dụng bảng trắng. Nhưng việc đó hơi lãng phí. Nếu có thể, hãy dành bảng trắng cho thiết kế thô và sử dụng những bức tường màu cho việc treo bảng công việc.

Chú ý - nếu bạn có nhiều công việc, đừng quên dán chúng bằng các mẩu băng dính, vì nếu không bạn có thể sẽ thấy chúng nằm la liệt trên sàn nhà vào một ngày nào đó.

Sử dụng bảng công việc (taskboard) như thế nào

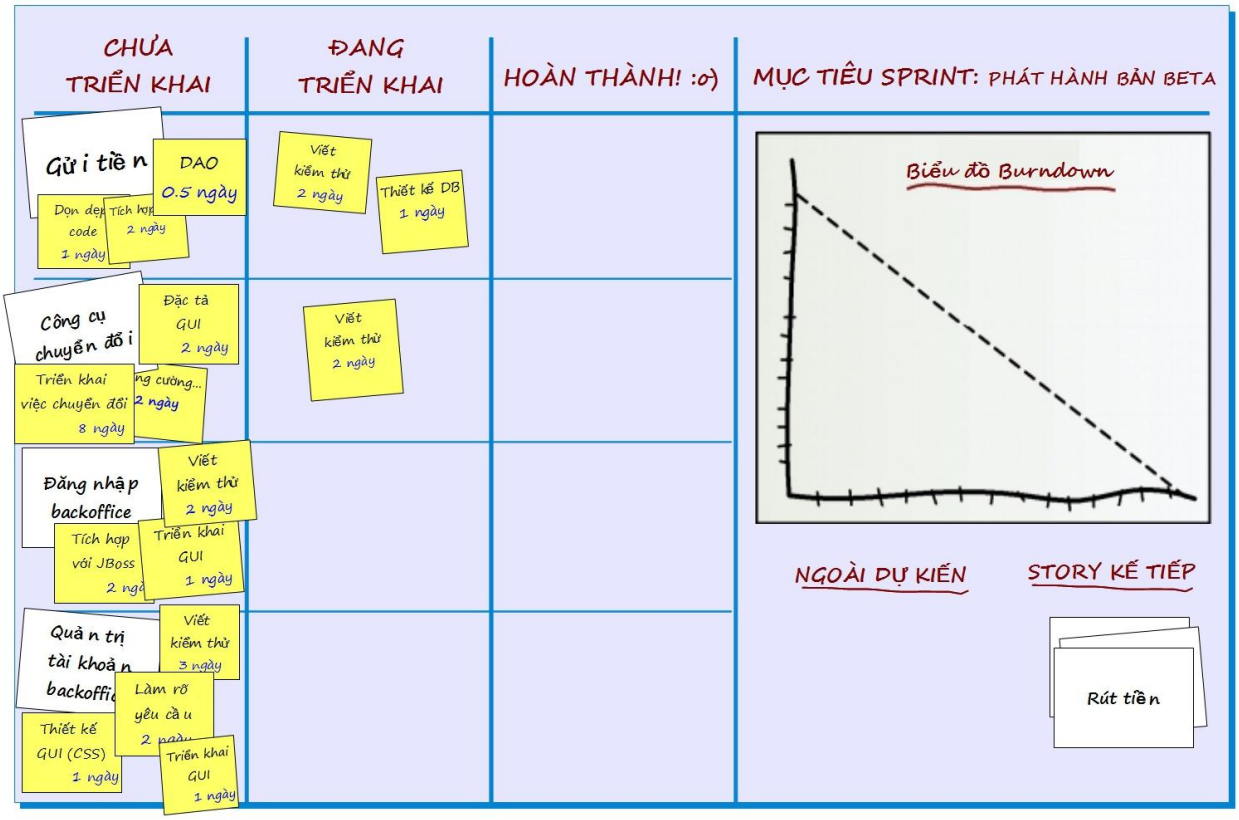


Tất nhiên, bạn có thể thêm tất cả các kiểu cột khác nhau. Ví dụ cột *"Chờ tích hợp kiểm thử"* hoặc *"Đã được hủy bỏ"*. Tuy nhiên trước khi bạn gây thêm rắc rối, hãy suy nghĩ kỹ. Việc thêm những cột này có *thực sự* cần thiết hay không?

Tôi đã phát hiện ra những thứ đơn giản và vô cùng giá trị cho những điều này, vì vậy tôi chỉ thêm vào những cột khi chi phí cho việc *không* làm như vậy là rất lớn.

Ví dụ 1 – sau buổi Họp Scrum Hàng ngày

Sau buổi Họp Scrum Hằng ngày đầu tiên, bảng công việc có thể trông giống như sau:



Như bạn có thể thấy, có ba công việc đã được "triển khai" (checked out), tức là Nhóm làm việc sẽ bắt đầu những mục công việc này trong ngày hôm nay.

Đôi khi, đối với những Nhóm lớn, một công việc có thể bị mắc kẹt ở tình trạng "triển khai" vì không ai trong nhóm nhớ rằng ai đang làm việc đó. Nếu điều này thường xảy ra trong một Nhóm thì cần đưa ra những chính sách như thêm vào tên của người thực hiện công việc vào mỗi công việc đã được chọn để triển khai.

Ví dụ 2 – Sau một vài ngày

Một vài ngày sau bảng công việc có thể trông như thế này:

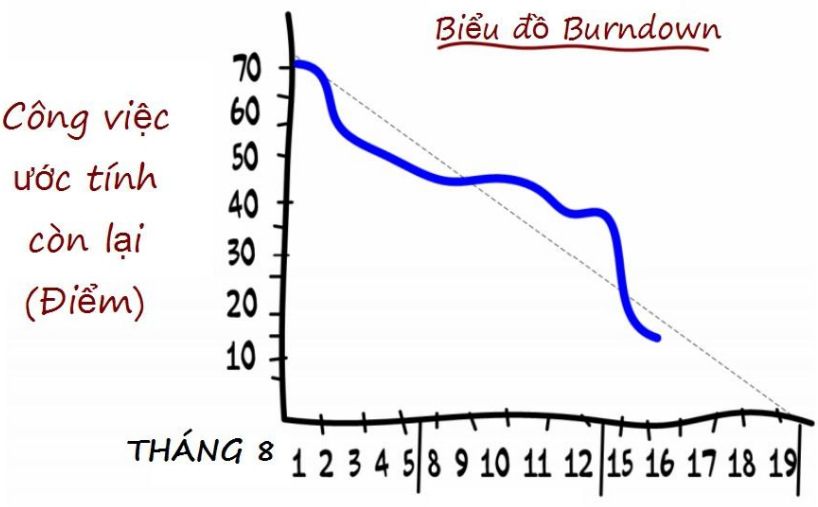
Như bạn có thể thấy chúng tôi đã hoàn thành story "gửi tiền" (tức là mã nguồn đã được đưa vào kho quản lý, đã được kiểm thử, đã được tái cấu trúc v.v.). Công cụ chuyển đổi đã được hoàn thiện một phần, chức năng đăng nhập cho bộ phận hành chính đã bắt đầu được triển khai, và chức năng quản trị người dùng vẫn chưa bắt đầu.

Chúng ta có 3 hạng mục chưa được lên kế hoạch, như bạn thấy ở phía dưới bên phải. Điều này cần phải nhớ khi bạn tiến hành Họp Cải tiến Sprint.

Dưới đây là một ví dụ thực về một Sprint Backlog khi sắp kết thúc Sprint. Nó trông lộn xộn hơn các quy trình Sprint, nhưng điều đó chấp nhận được vì thời gian tồn tại của nó không lâu. Khi một Sprint mới được triển khai chúng ta dọn dẹp và làm mới lại chúng để chuẩn bị cho Sprint Backlog mới.

Cách làm việc với Biểu đồ Burndown

Hãy phóng to Biểu đồ Burndown:



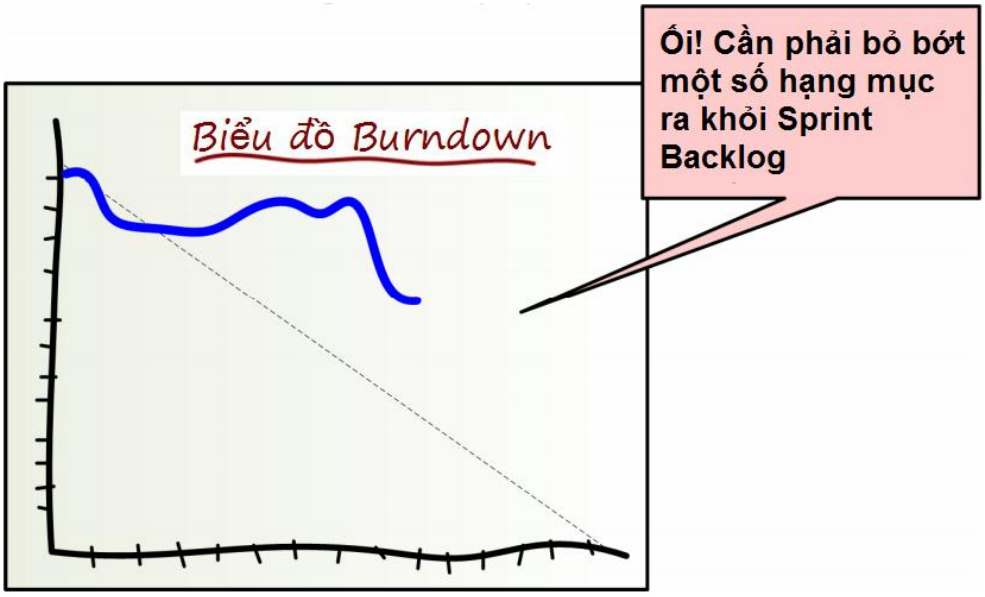
Biểu đồ này cho thấy:

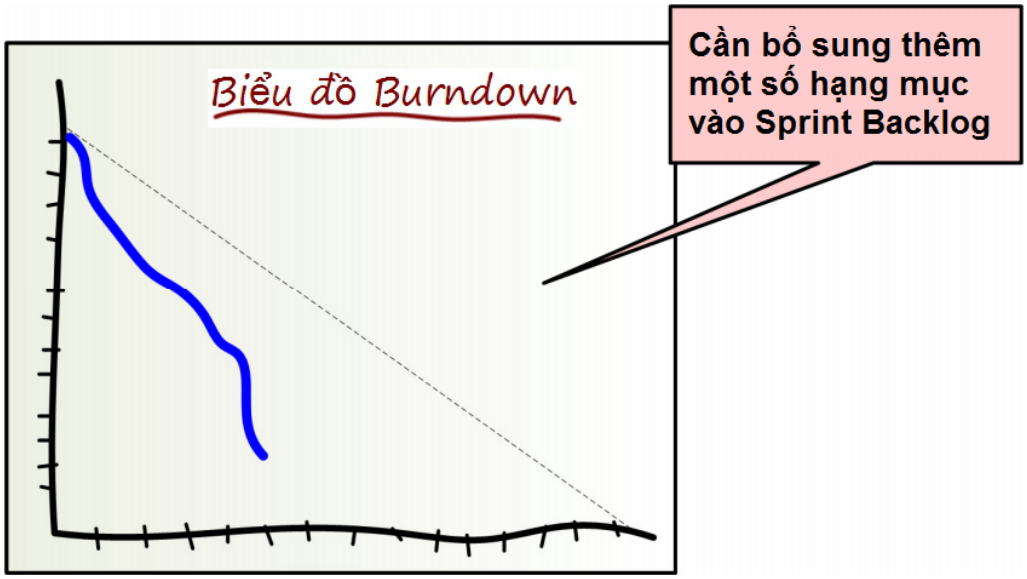
- Trong ngày đầu tiên của Sprint, ngày 1/8, Nhóm đã ước tính có khoảng 70 điểm (story point) cần phải thực hiện. Điều này có ảnh hưởng đến tốc độ ước tính của toàn bộ Sprint.
- Vào ngày 16/8 Nhóm ước tính có khoảng 15 điểm còn lại cần phải làm. Đường thẳng với nét đứt nằm ở trên cho thấy họ đang trong tầm kiểm soát, tức là với tốc độ như vậy họ sẽ hoàn thành tất cả các công việc khi kết thúc Sprint.

Chúng ta bỏ qua ngày nghỉ cuối tuần trên trục X vì công việc hiếm khi được làm vào cuối tuần. Chúng ta có thể thêm vào các ngày cuối tuần nhưng điều này sẽ dễ gây hiểu lầm khi xem biểu đồ, do đó phần ngày nghỉ cuối tuần sẽ được thêm vào các "gạch thẳng" trông giống như một ký hiệu cảnh báo (warning sign).

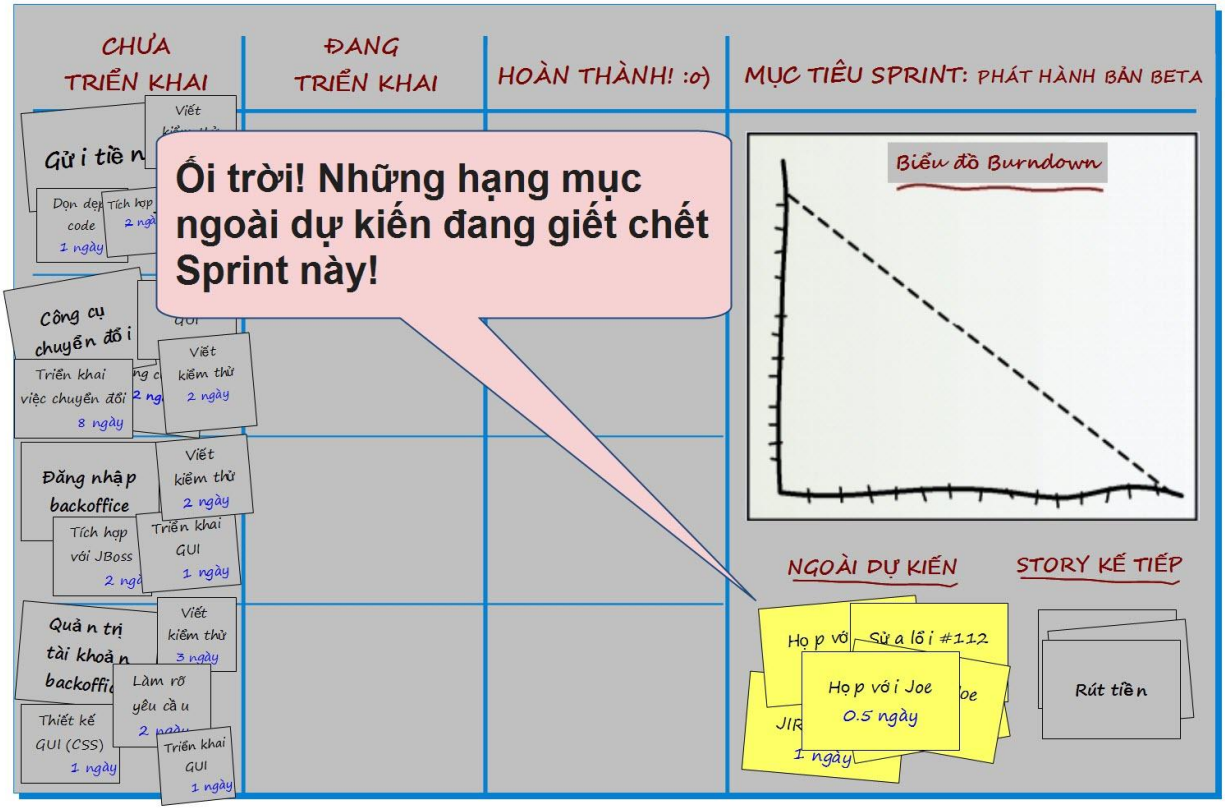
Những dấu hiệu cảnh báo trên bảng công việc

Nhìn lướt nhanh bảng công việc sẽ giúp bất kỳ ai có thể thấy được Sprint đang diễn ra như thế nào. Scrum Master có trách nhiệm đảm bảo rằng Nhóm hành động dựa trên những dấu hiệu cảnh báo giống như sau:




Biểu đồ Burndown
Cần bổ sung thêm một số hạng mục vào Sprint Backlog


Ối trời! Nhóm đang làm những thứ không đúng với thứ tự và bỏ qua các hạng mục có mức ưu tiên hàng đầu!
CHƯA TRIỂN KHAI
MỤC TIÊU SPRINT: PHÁT HÀNH BẢN BETA
Biểu đồ Burndown
Viết kiểm thử 2 ngày
Dọn dẹp code 1 ngày
DAO 0.5 ngày
Thiết kế DB 2 ngày
Đặc tả GUI
Viết kiểm thử 2 ngày
Tăng cường... 2 ngày
Đăng nhập backoffice
Triển khai GUI 1 ngày
Tích hợp với JBoss 2 ngày
Viết kiểm thử 2 ngày
Quản trị tài khoản backoffice
Viết kiểm thử 3 ngày
Triển khai GUI 1 ngày
Thiết kế GUI (CSS) 1 ngày
2 ngày
NGOÀI DỰ KIẾN
STORY KẾ TIẾP
Sửa lỗi rò rỉ bộ nhớ (JIRA 125) 2 ngày
Hỗ trợ bán hàng 3 ngày
Viết whitepaper 4 ngày
Rút tiền



Này, thế còn về khả năng theo dõi thì sao?!

Việc theo dõi (traceable[13]) tốt nhất tôi có thể gợi ý trong phần này là chụp một kiểu ảnh số của bảng công việc hàng ngày. Nếu đó là việc mà bạn phải làm. Tôi thỉnh thoảng làm như vậy nhưng không bao giờ cần phải đào bới lại những bức ảnh đó.

Nếu việc theo dõi là quan trọng với bạn, thì có thể giải pháp sử dụng bảng công việc không dành cho bạn.

[13] Traceability là khả năng theo dõi và lưu vết sự tiến triển, thay đổi trạng thái của công việc hoặc bản thân bảng công việc. Một bảng công việc bằng Excel hoặc các phần mềm quản lí dự án thường có khả năng lưu vết tốt hơn là bảng công việc. Việc này thường có ích khi chúng ta thanh tra lại cách làm, quy trình, đặc biệt là khi có sai hỏng; các vết lưu được sẽ cung cấp bằng chứng để ra quyết định sau đó.

Nhưng tôi thực sự đề nghị bạn hãy thử ước tính giá trị thực tế của việc lưu vết chi tiết mỗi Sprint. Một khi Sprint được hoàn thành và mã nguồn chạy tốt đã được gửi đi và tài liệu đã được triển khai, có ai đó thực sự quan tâm bao nhiêu story đã được hoàn thành trong ngày thứ 5 của Sprint hay không? Có ai thực sự quan tâm tới thời gian được ước tính cho việc "viết một kiểm thử sai cho story 'Gửi tiền'" không?

Ước tính ngày và giờ

Hầu hết trong các cuốn sách và bài viết về Scrum bạn sẽ tìm thấy công việc (task) được ước tính theo giờ, chứ không phải theo ngày. Chúng tôi đã từng làm như vậy. Công thức chung của chúng tôi là: 1 ngày làm việc (man-day) hiệu quả = 6 giờ làm việc (man-hours) hiệu quả.

Bây giờ chúng tôi không làm như vậy, ít nhất là với tất cả các Nhóm của chúng tôi, do một số nguyên nhân dưới đây:

- Ước tính giờ làm việc là quá nhỏ, việc này có xu hướng tạo nên rất nhiều công việc 1-2 giờ và do đó sẽ trở thành quản lý vi mô (micro-manage).
- Từ đó suy ra rằng tất cả mọi người đang nghĩ đến ngày làm việc theo bất kỳ cách nào, nó sẽ được nhân lên 6 trước khi viết thành giờ làm việc. "Hừm, công việc này cần một ngày để làm. Ô tôi phải viết ra số giờ, tôi sẽ viết 6 giờ và sau đó..".
- Hai đơn vị tính khác nhau gây ra nhầm lẫn. "Nên ước tính bằng ngày làm việc hay giờ làm việc đây?"

Vì vậy bây giờ chúng tôi dùng ngày làm việc làm đơn vị thời gian cơ bản để ước tính (dù chúng tôi gọi nó là điểm (story points)). Giá trị thấp nhất của chúng tôi là 0.5, tức là bất kỳ công việc nào nhỏ hơn 0.5 đều được hủy bỏ hoặc kết hợp lại với một công việc khác, hoặc để ước tính là 0.5 (không có hại lớn cho gì cho việc ước tính đó). Hay và đơn giản.

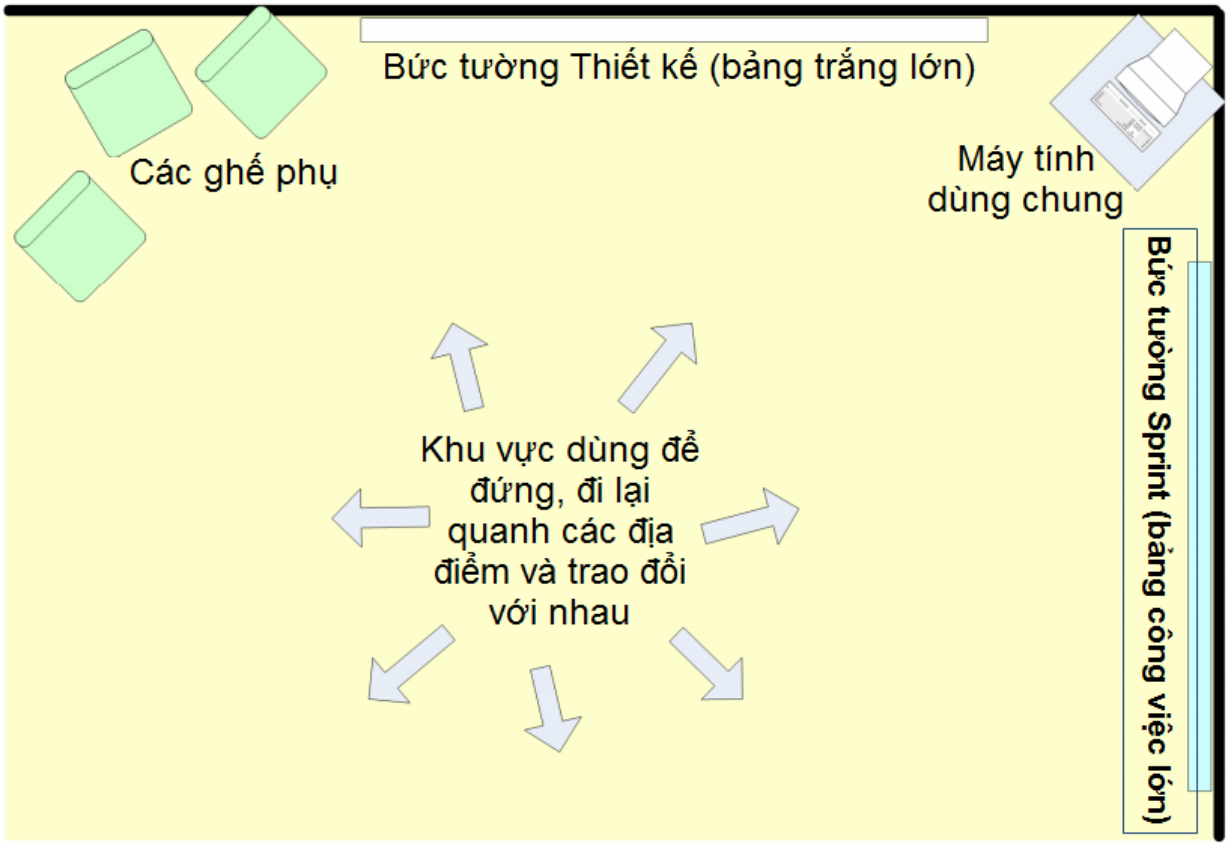
7

Chúng tôi bố trí phòng làm việc như thế nào

Góc thiết kế

Tôi đã nhận thấy những cuộc thảo luận về thiết kế thường thú vị và giá trị nhất khi diễn ra một cách tự nhiên phía trước bảng công việc (taskboard).

Vì lý do này, chúng tôi cố gắng sắp xếp không gian này như "góc thiết kế" hoàn hảo



Điều này thực sự rất có ích. Không có cách nào tốt hơn để quan sát tổng quan hệ thống hơn là đứng tại góc thiết kế và có thể nhìn sang cả hai phía tường, sau đó nhìn vào máy tính và thử bản

build mới nhất của hệ thống (nếu bạn may mắn có các bản build liên tục, xem trang 95, *"Chúng tôi kết hợp Scrum với XP ra sao"*).

"Bức tường thiết kế" chỉ là một bảng trắng lớn chứa các mảnh giấy và các bản in của các tài liệu thiết kế quan trọng nhất (sơ đồ tuần tự, các bản mẫu thiết kế giao diện, mô hình miền, v.v.).

Hình trên: một cuộc Họp Scrum Hằng ngày đang diễn ra ở một góc như vậy.

Hừm... biểu đồ Burndown trông khá đẹp và đi xuống thẳng một cách đáng ngờ đúng không nhỉ. Nhưng Nhóm khẳng định rằng đó là sự thật :o)

Đặt chỗ cho Nhóm ngồi cùng nhau!

Khi nói đến cách bố trí bàn và chỗ ngồi có một điều không cần tranh cãi.

Đặt chỗ cho Nhóm ngồi cùng nhau!

Để làm rõ một chút những gì tôi nói là

Đặt chỗ cho Nhóm ngồi cùng nhau!

Mọi người không thích phải di chuyển. Ít nhất ở nơi tôi làm việc là như vậy. Họ không muốn phải nhấc tất cả các đồ dùng của họ, gỡ máy tính, di chuyển tất cả giấy tờ sang một chiếc bàn mới và sắp xếp mọi thứ lại từ đầu. Khoảng cách càng ngắn thì sự ngại ngần càng giảm. "Thôi nào ông chủ, sao lại phải di chuyển thêm 5 mét nữa làm gì?"

Tuy nhiên, khi xây dựng các nhóm Scrum hiệu quả, không có giải pháp nào thay thế cả. Chỉ có cách sắp xếp cho họ ngồi cùng nhau! Thậm chí bạn phải xâm phạm vào sự riêng tư của mỗi cá nhân, chuyển tất cả các thiết bị của họ ra và tẩy các vết bẩn cà phê cũ của họ. Nếu không có không gian cho Nhóm thì hãy tạo ra không gian. Ở đâu đó. Thậm chí nếu bạn phải đặt Nhóm ở tầng hầm. Di chuyển các bàn xung quanh, hối lộ quản lý tòa nhà, làm bất cứ điều gì có thể. Chỉ cần Nhóm được ngồi cùng với nhau!

Khi Nhóm của bạn làm việc cùng nhau thì thành quả sẽ có ngay lập tức. Chỉ sau một Sprint Nhóm sẽ nhất trí rằng ý tưởng để cả Nhóm cùng nhau là ý tưởng tốt (tôi rút ra từ kinh nghiệm cá nhân về điều này, không có gì để nói nếu Nhóm của bạn không quá bảo thủ để thừa nhận).

Bây giờ cần hiểu "cùng nhau" nghĩa là gì? Các bàn làm việc được sắp xếp ra sao? Vâng, tôi không có bất kỳ một ý kiến kiên quyết nào về việc bố trí bàn làm việc tối ưu. Và cho dù tôi có làm, tôi cho rằng hầu hết các Nhóm không vui vẻ về khả năng quyết định chính xác cách để bố trí bàn làm việc của họ. Thường thì phải tính tới các ràng buộc khác nữa – các nhóm lân cận, cửa nhà vệ sinh, máy móc lớn giữa phòng, v.v.

"Cùng nhau" nghĩa là :

- **Có thể nghe rõ:** Bất cứ ai trong Nhóm có thể nói với người khác mà không cần phải nói to hoặc rời khỏi bàn của anh ta.
- **Khả năng quan sát:** Bất cứ ai trong Nhóm có đều có thể nhìn thấy những người khác. Mọi người đều có thể nhìn thấy bảng công việc (taskboard). Không nhất thiết là phải đủ gần để có thể *đọc* được những thứ viết trên bảng mà chỉ c*ần nhìn th*ấy là được.
- **Riêng biệt:** Nếu toàn bộ Nhóm của bạn đột nhiên đứng lên và tham gia vào một cuộc thảo luận tự phát và trực tiếp, không ai ngoài nhóm của bạn ở gần đó bị quấy rầy. Và ngược lại.

"Riêng biệt" không có nghĩa là Nhóm phải hoàn toàn tách biệt. Trong căn phòng có thể phải đủ để Nhóm của bạn có được không gian riêng và tường bao xung quanh phòng đủ lớn để lọc các tạp âm từ các nhân tố ngoài Nhóm.

Và chuyện gì sẽ xảy ra nếu bạn có một Nhóm phân tán? Vâng, vậy không may cho bạn rồi. Sử dụng càng nhiều các kỹ thuật hỗ trợ có thể để giảm thiểu ảnh hưởng – hội thảo truyền hình, webcam, các công cụ chia sẻ màn hình, v.v.

Bố trí cho Product Owner một khu vực

Product Owner sẽ đủ gần để Nhóm có thể đi tới và hỏi anh ta vài điều, và anh ta có thể đi tới bảng công việc. Nhưng anh ta sẽ không được ngồi cùng Nhóm. Tại sao? Bởi vì có thể anh ta sẽ không kiềm chế được bản thân và can thiệp vào chi tiết, và Nhóm sẽ không "kết dính" một cách đúng đắn (tức là đạt được sự gắn kết, tự quản, trạng thái siêu năng suất).

Thành thật mà nói, đây là việc mà tôi suy diễn. Tôi chưa thực sự nhìn thấy trường hợp nào mà Product Owner ngồi cùng với Nhóm, do đó tôi không có kinh nghiệm thực tế để nói rằng đó là ý kiến tồi. Đó chỉ là trực cảm và được nghe nói từ các Scrum Master khác.

Bố trí cho những người quản lý và người huấn luyện ở một khu vực

Với tôi điều này hơi khó viết do tôi vừa là quản lý vừa là người huấn luyện …

Công việc của tôi làm việc càng gần với Nhóm càng tốt. Tôi thiết lập các Nhóm, di chuyển giữa các Nhóm, lập trình cặp với mọi người, huấn luyện các Scrum Master, tổ chức các buổi Họp Kế hoạch Sprint, v.v. Nhìn lại, mọi người nghĩ đây là **điều tốt**, do tôi có kinh nghiệm với phương pháp phát triển phần mềm linh hoạt.

Nhưng, sau này, tôi cũng là giám đốc phát triển, vai trò là giám đốc quản lý chức năng (gia nhập Darth Vader music). Điều này có nghĩa là với việc tham gia một Nhóm sẽ tự động làm cho Nhóm trở nên ít tự quản lý hơn. "Quỷ tha ma bắt, ông chủ ở đây, ông ý có thể có rất nhiều ý kiến về những điều chúng ta sẽ thực hiện và ai sẽ làm gì. Tôi sẽ để ông ấy nói về vấn đề này."

Quan điểm của tôi ở đây là, nếu bạn là một nhà huấn luyện Scrum (và có lẽ cũng là người quản lý), hãy tham gia càng chặt chẽ càng tốt. Nhưng chỉ trong thời gian có hạn thôi, sau đó ra ngoài và để nhóm gắn kết và tự quản lý. Kiểm tra Nhóm trong chốc lát (không quá thường xuyên) bằng cách tham gia các buổi sơ kết Sprint và nhìn vào bảng công việc, lắng nghe vào các buổi họp hằng ngày. Nếu bạn thấy có lĩnh vực nào cần cải tiến, gặp Scrum Master và hướng dẫn cho anh ta. *Không* phải là trước mặt Nhóm. Một ý hay khác là tham gia các buổi Họp Cải tiến Sprint (xem trang 81, *"Chúng tôi Cải tiến Sprint như thế nào"*), nếu Nhóm của bạn tin rằng sự có mặt của bạn không đủ để làm họ im lặng.

Đối với các Nhóm Scrum thực hiện tốt chức năng, đảm bảo rằng họ có mọi thứ họ cần, sau đó hãy để họ nằm ngoài tầm kiểm soát (ngoại trừ trong các buổi Sơ kết Sprint).

8

Chúng tôi họp Scrum Hằng ngày như thế nào

Các buổi Họp Scrum Hằng ngày được chúng tôi nói đến khá nhiều trong cuốn sách. Các buổi họp bắt đầu đúng giờ, hàng ngày tại cùng một địa điểm. Thời kỳ bắt đầu chúng tôi vào một căn phòng riêng biệt để thực hiện lập kế hoạch Sprint (những ngày đó chúng tôi sử dụng Sprint Backlog điện tử), tuy nhiên hiện tại chúng tôi thực hiện họp Scrum Hằng ngày phía trước bảng công việc trong phòng làm việc của Nhóm. Không có gì phải bàn cãi về điều này cả.

Chúng tôi thông thường đứng họp, nhờ thế sẽ giảm được nguy cơ cuộc họp kéo dài quá 15 phút.

Chúng tôi cập nhật bảng công việc như thế nào

Chúng tôi thường cập nhật bảng công việc trong buổi họp Scrum Hằng ngày. Khi mỗi thành viên mô tả về những gì anh ta đã làm hôm trước và sẽ làm hôm nay, anh ta sẽ di chuyển miếng giấy ghi công việc tới các vị trí tương ứng trên bảng công việc. Nếu anh ta đề cập tới một công việc không nằm trong kế hoạch, anh ta đưa thêm một mẩu giấy mô tả công việc đó. Anh ta cập nhật ước tính thời gian của anh ta để hoàn thành, anh ta viết ước tính thời gian mới lên trên miếng giấy ghi công việc đó và gạch bỏ ước tính cũ. Đôi khi Scrum Master thực hiện động tác thay đổi vị trí các miếng giấy ghi công việc trong khi mọi người nói.

Một số Nhóm có quy ước mỗi người sẽ cập nhật bảng công việc *trước* mỗi buổi họp. Công việc đó cũng tốt. Chỉ cần quyết định quy ước đó và thực hiện theo nó là được.

Cho dù Sprint Backlog của bạn định dạng kiểu gì thì hãy cố gắng để toàn Nhóm cùng tham gia vào việc cập nhật nó hằng ngày. Chúng tôi đã thực hiện các Sprint ở đó Scrum Master là người nắm

giữ chính Sprint Backlog, phải để mắt đến nó hằng ngày, và đề nghị mọi người cập nhật các ước tính thời gian còn lại của họ lên đó. Nhược điểm của nó là:

- Scrum Master mất khá nhiều thời gian vào việc quản lý Sprint Backlog, thay vì việc hỗ trợ Nhóm và loại bỏ những trở ngại trong công việc.
- Các thành viên Nhóm không chú ý tới trạng thái của Sprint, vì vậy dẫn tới Sprin Backlog không phải là thứ họ quan tâm. Việc thiếu vắng sự phản hồi này làm giảm tính linh hoạt và sự tập trung của Nhóm.

Nếu Sprint Backlog được thiết kế tốt sẽ giúp mỗi thành viên cập nhật công việc của mình một cách dễ dàng.

Ngay sau buổi họp Scrum Hằng ngày, một người nào đó sẽ tổng kết lại tất cả các ước tính thời gian (bỏ qua các công việc ở cột "hoàn thành") và cập nhật một điểm mới trên biểu đồ Sprint Burndown.

Ứng xử với những người đến muộn

Một vài Nhóm có một thùng đựng tiền xu và phiếu. Khi bạn đến muộn, thậm chí chỉ muộn một phút, bạn phải đặt vào thùng một lượng theo quy định. Không có gì để thắc mắc cả. Nếu bạn có gọi trước cuộc họp và nói bạn sẽ muộn thì bạn vẫn sẽ phải nộp như thế.

Bạn chỉ không bị phạt nếu bạn có lý do chính đáng chẳng hạn bạn có lịch hẹn khám bác sĩ hoặc bạn sẽ tổ chức đám cưới của mình hay đại loại cái gì đó tương tự.

Tiền trong thùng được sử dụng cho các sự kiện chung. Chẳng hạn mua bánh hamburger khi chúng tôi chơi game vào các buổi tối :o)

Việc này cũng thú vị lắm. Nhưng nó chỉ cần thiết đối với những Nhóm mà thành viên thường hay đến muộn. Một số Nhóm không cần áp dụng hình thức như vậy.

Ứng xử với những câu trả lời "Tôi không biết làm gì ngày hôm nay"

Không phải là không có người nói "Hôm qua tôi đã làm bla bla bla, nhưng hôm nay tôi chưa biết phải làm gì" (này, đó là một mẩu thông tin đưa vào để cho nó có vấn đề). Bây giờ làm gì?

Tôi sẽ kể về Joe và Lisa là những người không biết hôm nay làm gì.

Nếu tôi là Scrum Master, tôi chỉ tiến đến và để cậu bên cạnh nói, nhưng tạo sự chú ý cho những người không có gì để làm. Sau khi mọi người đã nói các xong công việc của họ, tôi duyệt bảng công việc với toàn Nhóm, từ trên xuống dưới, và kiểm tra xem mọi thứ đã đồng bộ chưa để mọi

người biết mỗi công việc có ý nghĩa gì, v.v.. Tôi mời mọi người bổ sung thêm các miếng giấy ghi các công việc mới. Sau đó, tôi hướng trở lại những người không biết phải là gì nói "bây giờ chúng ta đã duyệt qua bảng công việc, bạn có ý tưởng gì về những thứ bạn có thể làm hôm nay chưa?" Hy vọng họ sẽ thấy.

Nếu không, tôi xem có cơ hội nào để tiến hàng lập trình cặp ở thời điểm này không. Sau đó với Niklas tiến hành làm phần giao diện (GUI) cho chức năng quản trị người dùng vào hôm nay. Trong trường hợp này tôi gợi ý một cách lịch sự rằng Joe hoặc Lisa có thể lập trình cặp với Niklas công việc đó. Đấy là việc tôi thường làm.

Và nếu các thức đó không giải quyết được, thì đây là chút mánh khóe:

> **Scrum Master:** "OK, ai muốn demo bản phát hành betacho chúng ta ?" (giả sử đó là mục tiêu của Sprint)
>
> **Nhóm:** Im lặng trong bối rối
>
> **Scrum Master:** "Chúng ta không thực hiện việc này?"
>
> **Nhóm:** "ừm .. không"
>
> **Scrum Master:** "Khỉ thật" Tạo sao không ? Thế còn gì để làm đây?"
>
> **Nhóm:** "Vâng chúng tôi thậm chí chưa có server kiểm thử để chạy và kịch bản build đang bị lỗi."
>
> **Scrum Master:** "Aha." (bổ sung thêm hai đầu việc lên bảng công việc). "Joe và Lisa, các bạn có cách nào để trợ giúp chúng ta ngày hôm nay không?"
>
> **Joe:** "Ừm… Tôi sẽ thử tìm một vài server kiểm thử ở đâu đó".
>
> **Lisa:** "… và tôi sẽ cố gắng sửa lỗi kịch bản build".

Nếu bạn may mắn, một ai đó sẽ trình bày bản phát hành beta mà bạn yêu cầu. Tuyệt vời! Bạn đã đạt được mục tiêu Sprint của bạn. Nhưng bạn sẽ phải làm gì nếu bạn đang ở giữa Sprint? Dễ thôi. Chúc mừng Nhóm vì công việc đang hoàn thành tốt, lấy một hoặc hai tính năng từ khu vực "Những phần kế tiếp" nằm ở bên phải phía dưới bảng công việc của bạn, đặt chúng vào cột "chưa kiểm tra" nằm ở phía bên trái. Sau đó thực hiện lại buổi Họp Scrum Hằng ngày. Thông báo cho Product Owner là bạn đã bổ sung một vài công việc vào Sprint.

Nhưng sẽ ra sao nếu Nhóm chưa đạt được mục tiêu Sprint, Joe và Lisa vẫn từ chối để chọn ra một việc nào đó có ích để làm. Tôi thường xem xét sử dụng một trong những chiến lược dưới đây (đây không phải dùng là chiến lược hay, nhưng đây là kế sách cuối cùng):

- **Làm cho hổ thẹn:** "Vâng, nếu các bạn không có ý tưởng gì để hỗ trợ Nhóm, tôi khuyên bạn về nhà, hay đọc sách hoặc làm gì đó. Hoặc ngồi quanh đây cho đến khi có người cần sự trợ giúp của bạn."
- **Sử dụng cách truyền thống:** Đơn giản giao cho họ một nhiệm vụ.
- **Áp lực ngang hàng:** Nói rằng "Joe và Lisa, các bạn hãy thoải mái sử dụng thời gian rảnh của mình, tất cả chúng tôi sẽ chỉ đứng ở đây cho đến khi các bạn chọn được việc gì đó để hỗ trợ chúng ta hoàn thành công việc".
- **Thân phận nô lệ:** Nói rằng "Tốt thôi, hôm nay các bạn có thể giúp Nhóm gián tiếp bằng việc trở thành quản gia. Lấy cà phê, mát-xa cho mọi người, đi đổ rác, nấu bữa trưa cho chúng tôi, và bất cứ cái gì mà chúng tôi đề nghị trong cả ngày hôm nay nhé". Bạn có thể ngạc nhiên bởi Joe và Lisa sẽ nhanh chóng chọn một việc gì đó về kỹ thuật để làm :o)

Nếu một ai đó thường bắt bạn phải đi quá xa để giải quyết, thì có lẽ bạn nên gạt người đó sang bên cạnh và tiến hành một vài huấn luyện nghiêm khắc. Nếu vấn đề không giải quyết được, bạn cần xem xét xem người này có quan trọng với Nhóm của bạn hay không.

Nếu anh ta *không quá quan trọng*, hãy thử loại anh ta ra khỏi Nhóm của bạn.

Nếu anh ta *quan trọng*, cố tạo cặp cho anh ta với một người nào đó có thể đóng vai trò như "người chăm sóc" anh ta. Joe có thể là một lập trình viên và nhà kiến trúc lão luyện, chỉ có điều anh ta thực sự không thích người khác nói anh ta phải làm gì. Tốt thôi. Giao cho Niklas nhiệm vụ là người chăm sóc thường trực của Joe. Hoặc chính bạn làm nhiệm vụ đó. Nếu Joe quan trọng đối với Nhóm của bạn thì điều này rất đáng để làm. Chúng tôi đã gặp các trường hợp như vậy và những gợi ý ở trên ít nhiều có tác dụng.

9

Chúng tôi Sơ kết Sprint ra sao

Sprint Demo (hoặc Sơ kết Sprint như mọi người thường gọi) là một sự kiện quan trọng của Scrum nhưng thường có xu hướng bị mọi người đánh giá thấp.

"Ồ, thực sự là chúng tôi phải tiến hành demo hay sao? Thực sự có chẳng có nhiều điều vui vẻ để mà trình diễn cả!"

"Chúng tôi không có thời gian để chuẩn bị cho một buổi demo gì gì... đó!" "Tôi không có thời gian tham dự các buổi demo của nhóm khác!"

Lý do chúng tôi nhấn mạnh đến việc các Sprint cần kết thúc với một buổi demo

Một buổi Sơ kết Sprint thành công, mặc dù có thể không gây ấn tượng mạnh, nhưng lại có ảnh hưởng sâu sắc.

- Nhóm có được niềm tin vào thành quả của họ. Họ *cảm thấy những điều tốt đẹp*.
- Những người khác biết được những gì mà nhóm đang thực hiện.
- Buổi demo còn thu nhận được các phản hồi quan trọng từ những người liên quan.
- Các buổi demo là (hoặc nên là) một sự kiện mang tính cộng đồng để các nhóm khác nhau có thể trao đổi và thảo luận về công việc của họ. Đây chính là giá trị mang lại.
- Tiến hành demo sẽ tạo áp lực để nhóm tiến tới *hoàn chỉnh thực sự một số thứ* và phát hành nó (ngay cả khi nó chỉ là môi trường kiểm thử). Không có buổi demo, chỉ chúng ta mới biết rằng 99% công việc đã hoàn thành. Với các buổi demo chúng ta có thể chỉ có một số hạng mục được hoàn thành, nhưng chúng là những hạng mục đã *thực sự hoàn thành*, điều này (trong trường hợp của chúng tôi) còn tốt hơn có cả đống những thứ mới chỉ *sắp hoàn thành* và sẽ làm ô nhiễm Sprint tiếp theo.

Nếu một nhóm ít nhiều bị áp lực để tiến hành demo Sprint, thậm chí khi họ không có nhiều thứ thực sự hoàn chỉnh, buổi demo sẽ gặp phải khó khăn. Nhóm sẽ lúng túng và dễ sơ sẩy trong khi

demo và những tràng pháo tay chỉ là nửa vời. Mọi người sẽ cảm thấy một chút tiếc nuối với nhóm, một số có thể sẽ tức tối vì họ đã lãng phí thời gian tham dự một buổi demo tệ hại.

Quả là một sự đau đớn. Nhưng nó có hiệu quả như một liều "thuốc đắng dã tật". *Sprint tới,* Nhóm sẽ thực sự cố gắng để có được những công việc được gọi là *hoàn thành*! Họ sẽ cảm thấy rằng "tốt thôi, có thể chúng ta sẽ chỉ trình diễn 2 thứ vào Sprint tới thay vì 5, nhưng những thứ chết tiệt này sẽ thực sự CHẠY NGON!". Nhóm biết rằng họ sẽ phải làm một buổi demo mà không xảy ra vấn đề gì, đây là dấu hiệu làm tăng cơ hội có một điều gì đó hữu ích để demo. Tôi đã từng thấy điều này xảy ra một vài lần.

Những mục cần kiểm tra cho buổi Sơ kết Sprint

- Hãy chắc chắn rằng bạn trình bày rõ được mục tiêu của Sprint. Nếu có ai đó tham dự buổi demo mà không hiểu về bất cứ điều gì liên quan đến sản phẩm của bạn, hãy dành một chút thời gian để mô tả về nó.
- Đừng tốn quá nhiều thời gian chuẩn bị cho buổi demo, đặc biệt là để có một bản thuyết trình hào nhoáng. Cắt bỏ những nội dung vớ vẩn và chỉ tập trung trình bày các mã nguồn thực sự chạy tốt.
- Giữ nhịp độ cao, tức là tập trung những nội dung chuẩn bị của bạn vào việc tạo ra bản demo có tốc độ cao chứ không phải đẹp.
- Giữ cho nội dung demo ở mức hướng vào các nghiệp vụ, bỏ qua các chi tiết kỹ thuật. Tập trung vào "những gì chúng tôi đã làm" hơn là "cách mà chúng tôi đã làm".
- Nếu có thể, hãy để cho khách tham dự sử dụng thử sản phẩm.
- Đừng trình bày việc sửa một loạt các lỗi nhỏ và các tính năng tầm thường. Đề cập đến chúng nhưng không demo, vì điều đó thường tốn thời gian và giảm trọng tâm đối với những thứ quan trọng hơn.

Giải quyết với những thứ "chưa thể demo"

Thành viên Nhóm: "Tôi sẽ không trình bày tính năng này, bởi vì nó không thể demo được. Chức năng đó yêu cầu 'Tăng khả năng đáp ứng của hệ thống để có thể phục vụ đồng thời 10,000 người dùng'. Tôi có thể không bỏ công sức để có được 10,000 người dùng đồng thời cho việc demo có được không?"

Scrum Master: "Anh đã hoàn thành tính năng này chưa?"

Thành viên Nhóm: “Vâng, tất nhiên rồi”.

Scrum Master: “Làm sao anh biết được điều này?”

Thành viên Nhóm: “Tôi đã cài đặt hệ thống vào trong một môi trường kiểm thử hiệu suất, khởi động với 8 máy chủ và tạo những yêu cầu đồng thời để thử nghiệm hệ thống”.

Scrum Master: “Nhưng anh có cái gì để xác định rằng hệ thống sẽ đáp ứng được 10,000 người dùng”.

Thành viên Nhóm: “Có. Các máy kiểm thử hơi tồi, chúng không thể đáp ứng được 50,000 yêu cầu đồng thời trong suốt quá trình kiểm thử”.

Scrum Master: “Làm sao anh biết được điều này?”

Thành viên Nhóm (đã bắt đầu nản): “Vâng, tôi có báo cáo đó! Anh có thể tự xem, nó cho thấy cách thiết lập việc kiểm thử và cách gửi đi nhiều yêu cầu!”

Scrum Master: “Ồ, tuyệt vời! Vậy thì khi nào demo anh chỉ cần trình bày báo cáo này và thảo luận nó với người tham dự. Điều này tốt hơn là chẳng làm gì đúng không?”.

Thành viên Nhóm: “Ồ, như vậy là đủ sao? Nhưng nó hơi xấu, cần phải chỉnh sửa cho đẹp hơn”.

Scrum Master: “OK, nhưng đừng tốn quá nhiều thời gian cho việc đó. Nó không cần phải đẹp, chỉ cần đầy đủ thông tin”.

10
Chúng tôi Cải tiến Sprint như thế nào

Tại sao chúng tôi nhấn mạnh rằng tất cả các nhóm phải tiến hành cải tiến

Điều quan trọng nhất của cải tiến là *đảm bảo mọi thứ đều diễn ra suôn sẻ*.

Với một vài lí do, các nhóm dường như không sẵn sàng với việc cải tiến. Nếu không có sự thúc giục nhẹ nhàng thì hầu hết các nhóm của chúng tôi thường bỏ qua việc cải tiến và thay vào đó là chuyển sang tiến hành Sprint tiếp theo. Đây có thể là một thứ văn hóa tồn tại ở Thụy Điển, nhưng không chắc chắn lắm.

Tuy nhiên, hầu hết mọi người đều đồng ý rằng cải tiến là thật sự hữu ích. Trong thực tế, tôi cho rằng cải tiến là sự kiện quan trọng thứ hai trong Scrum (sự kiện quan trọng nhất là Họp Kế hoạch Sprint) bởi vì đó là *cơ hội tốt nhất để cải tiến*.

Tất nhiên, bạn không cần một buổi đến Họp Cải tiến Sprint để có được ý tưởng tốt, bạn có thể làm việc đó ngay tại bồn tắm nhà mình! Nhưng nhóm sẽ chấp nhận ý tưởng của bạn? Có thể, nhưng cả nhóm sẽ chấp nhận ý tưởng nhanh hơn nếu nó đến "từ nhóm", tức là, nó được đưa ra trong buổi Họp Cải tiến khi mọi người chấp nhận đóng góp và thảo luận về các ý tưởng đó.

Nếu không có cải tiến, bạn sẽ nhận thấy rằng nhóm sẽ tiếp tục gặp phải những lỗi giống nhau.

Chúng tôi tổ chức các buổi Họp Cải tiến như thế nào

Dưới đây là cách thức phổ biến, có thể thay đổi một chút, nhưng thông thường, chúng tôi sẽ làm như sau:

- Chúng tôi phân bổ từ 1-3 giờ phụ thuộc vào thời gian thảo luận dự kiến.
- Thành phần: Product Owner, toàn bộ Nhóm, và chính tôi.
- Chúng tôi chuyển tới một phòng kín, có ghế sô-pha ấm cúng ở góc phòng, có sân thượng hoặc một số nơi tương tự. Miễn là chúng ta có thể thảo luận mà không bị quấy rầy.
- Chúng tôi thường không thể họp cải tiến trong phòng của Nhóm Phát triển, vì mọi người có xu hướng không tập trung.
- Chỉ định ai đó làm thư ký.

- Với sự hỗ trợ từ Nhóm Phát triển, Scrum Master cho thấy Sprint Backlog và tổng hợp tóm tắt về Sprint. Những sự kiện quan trọng và các quyết định, v.v.
- Chúng tôi làm theo hình thức "vòng tròn". Mỗi người sẽ có cơ hội để nói, mà không bị ngắt giữa chừng, những gì họ cho là tốt, những gì họ nghĩ là có thể tốt hơn và những nghĩ họ muốn làm khác đi trong Sprint tiếp theo.
- Chúng tôi sẽ nhìn vào ước tính so với tốc độ thực tế. Nếu có sự khác biệt rõ ràng, chúng tôi sẽ cố gắng phân tích nguyên nhân.
- Sau khoảng thời gian thảo luận trên, Scrum Master cố gắng tóm tắt các đề xuất cụ thể về những gì chúng tôi có thể làm tốt hơn trong Sprint tiếp theo

Những cải tiến của chúng tôi nhìn chung không quá cầu kì. Chủ đề cơ bản luôn luôn giống nhau kiểu như "cái gì chúng ta có thể làm tốt hơn trong Sprint tiếp theo".

Đây là một ví dụ trên bảng về một cải tiến gần đây:

Ý nghĩa của ba cột trên bảng như sau:

- **Tốt**: nếu chúng tôi có thể làm lại một Sprint tương tự như vậy, chúng tôi có thể làm những công việc tương tự theo cách này.
- **Có thể làm tốt hơn**: nếu chúng tôi có thể làm lại Sprint tương tự, chúng tôi có thể làm những điều đó theo cách khác.
- **Cải tiến**: ý tưởng chắc chắn về cách mà chúng tôi có thể cải tiến trong tương lai.

Cột 1 và cột 2 là nhìn về quá khứ, trong khi cột 3 hướng tới tương lai.

Sau khi cả đội suy nghĩ về những ý kiến trên các mẫu giấy đó, họ sẽ sử dụng hình thức "bỏ phiếu" để xác định những cải tiến sẽ tập trung trong Sprint tiếp theo. Mỗi thành viên của nhóm nhận được 3 viên nam châm (dùng cho bảng từ) và đặt viên nam châm đó vào bất cứ cái tiến nào mà họ ưu tiên trong Sprint tiếp theo. Mỗi thành viên của nhóm có thể phân phối nam châm theo ý thích của họ, thậm chí có thể đặt 3 viên nam châm vào cùng 1 vấn đề.

Dựa vào cơ sở này, họ lựa chọn 5 cải tiến qui trình để tập trung và sẽ tuân theo nó cho tới buổi Họp Cải tiến tiếp theo.

Điều quan trọng là không được quá tham vọng. Chỉ tập trung vào một vài cải tiến với mỗi Sprint.

Lan truyền những bài học tới các nhóm khác

Thông tin được đề cập trong suốt quá trình Họp Cải tiến Sprint thường rất giá trị. Nhóm có khoảng thời gian tập trung khó khăn bởi vì nhà quản lý kinh doanh kìm giữ lập trình viên chỉ ở vai trò "chuyên gia công nghệ" trong các cuộc họp kinh doanh? Đây là thông tin quan trọng. Có lẽ các nhóm khác cũng có vấn đề tương tự? Chúng ta nên đào tạo về quản lý sản phẩm nhiều hơn là về sản phẩm, vì vậy họ có thể hỗ trợ nhiều hơn cho việc kinh doanh của họ?

Mỗi cải tiến Sprint không chỉ cho thấy cách mà một nhóm có thể thực hiện công việc tốt hơn trong Sprint tiếp theo, mà nó còn có ý nghĩa rộng hơn thế.

Chiến lược của chúng tôi để xử lý vấn đề này là rất đơn giản. Một người (trong trường hợp này là tôi) tham dự tất cả các buổi Họp Cải tiến Sprint và hành động giống như một cầu nối tri thức. Khá đơn giản.

Một giải pháp khác, có thể mỗi nhóm Scrum đưa ra một báo cáo cải tiến Sprint. Chúng tôi đã thử điều này nhưng nhận thấy rằng không có nhiều người đọc báo cáo này, và thậm chí chẳng mấy ai hành động theo chúng. Bởi vậy chúng tôi tiến hành một cách đơn giản để thay thế.

Quy tắc quan trọng cho người đóng vai trò "cầu nối tri thức":

- Anh ta là một người biết lắng nghe.

- Nếu buổi Họp Cải tiến quá trầm, anh ta chuẩn bị hỏi những câu hỏi đơn giảm với mục đích kích thích thảo luận trong nhóm. Ví dụ "nếu bạn có thể quay lại thời gian và làm lại Spint tương tự từ ngày đầu tiên, bạn có muốn làm gì khác không?"
- Anh ta sẵn sàng dành thời gian để xem xét tất cả các cải tiến của tất cả các nhóm.
- Anh ta là có vị trí trong cơ quan, bởi vậy anh ta có thể hành động cho những đề xuất cải tiến ngoài tầm kiểu soát của nhóm.

Công việc này có vẻ tốt, nhưng có thể có cách tiếp cận tốt hơn. Trong trường hợp đó xin hãy khai sáng cho tôi.

Thay đổi hay không thay đổi

Giả sử nhóm nhận định rằng "chúng ta giao tiếp quá ít, bởi vậy chúng ta tiếp tục bước theo chân người của khác và cứ làm rối tung các thiết kế của người khác lên."

Bạn cần làm điều gì trong trường hợp này? tổ chức họp thiết kế hằng ngày? Giới thiệu những công cụ mới giúp truyền thông dễ dàng hơn? Viết thêm các trang wiki? Vâng, có thể. Nhưng sau đó lại lặp lại, có thể không.

Chúng ta có thể thấy rằng, trong nhiều trường hợp, chỉ xác định rõ vấn đề là đủ để giải quyết trong Sprint tiếp theo. Đặc biệt, nếu bạn đưa những thông tin cải tiến Sprint lên tường trong phòng của nhóm (chúng tôi luôn luôn quên điều đó, thật đáng xấu hổ!). Mọi thay đổi bạn đều tính đến chi phí, trước khi tiến hành thay đổi, cân nhắc tất cả và hy vọng rằng sẽ không tự nhiên xuất hiện vấn đề gì (hoặc nếu có thì sẽ rất nhỏ).

Ví dụ đề cập ở trên ("chúng tôi giao tiếp quá ít...") là những ví dụ điển hình rằng có thể giải pháp tốt nhất là chúng ta không làm gì cả.

Nếu bạn định đổi mới mỗi khi có bất cứ ai đó than phiền về một vấn đề gì đó, mọi người có thể không sẵn sàng tiết lộ những lỗi nhỏ, điều đó cũng có thể rất đáng quan ngại.

Ví dụ về những vấn đề có thể xảy ra trong quá trình cải tiến

Đây là một vài ví dụ điển hình có thể xảy ra trong suốt cuộc Họp Kế hoạch Sprint, và những hành động điển hình cần thực hiện.

"Lẽ ra chúng ta đã phải dành nhiều thời gian hơn để chia nhỏ story thành những hạng mục nhỏ hơn và các tác vụ”

Đây là vấn đề khá phổ biến. Mỗi ngày tại cuộc Họp Scrum Hằng ngày, thành viên trong nhóm xem xét bản thân và nói “Tôi thực sự không biết phải làm gì ngày hôm nay”. Vì vậy, sau mỗi buổi họp hằng ngày bạn dành thời gian để tìm kiếm những công việc cụ thể. Thông thường sẽ hiệu quả hơn nếu làm những việc này trước.

Hành động điển hình: không. Một nhóm có thể sẽ sắp xếp những thứ đó trong buổi Họp Kế hoạch Sprint tiếp theo của mình. Nếu việc này lặp đi lặp lại, hãy tăng khung thời gian của buổi Họp Kế hoạch Sprint.

"Quá nhiều phiền nhiễu bên ngoài"

Hành động điển hình:

- yêu cầu nhóm giảm hệ tố tập trung của họ trong Sprint tiếp theo, để họ có thể thực hiện kế hoạch một các khả quan hơn.
- yêu cầu nhóm ghi nhận tốt hơn những phiền nhiễu trong Sprint tiếp theo. Ai gây ra và mất bao lâu để giải quyết. Điều này sẽ dễ dàng hơn để giải quyết những vấn đề sau này.
- yêu cầu nhóm cố gắng đưa những phiền nhiễu này tới Scrum Master hoặc là Product Owner.
- yêu cầu nhóm chỉ định một người là “thủ môn”, tất cả các phiền nhiễu này đều chuyển tới anh ta, để những người còn lại của nhóm có thể tập trung. Có thể là Scrum Master hoặc luân phiên.

"Chúng tôi không đạt được cam kết và chỉ hoàn thành được một nửa công việc”

Hành động điển hình: không. Nhóm có thể sẽ không gặp vấn đề này trong Sprint tiếp theo. Hoặc ít nhất là không quá tồi tệ.

"Văn phòng của chúng tôi là quá ồn ào và lộn xộn"

Hành động điển hình:

- cố gắng tạo ra môi trường tốt hơn, hoặc là chuyển nhóm tới một nơi khác. Thuê một phòng khách sạn chẳng hạn. Hoặc bất cứ điều gì bạn nghĩ ra được. (Xem trang 70, “Chúng tôi bố trí phòng làm việc của nhóm như thế nào”)

- Nếu không thể, hãy nói với nhóm để giảm hệ số tập trung trong Sprint tiếp theo, và nói rõ điều này là do những ồn ào và lộn xộn của môi trường. Hy vọng rằng điều này sẽ là lý do để Product Owner làm phiền quản lý cấp trên.

May mắn là là tôi không bao giờ phải đe dọa di chuyển nhóm tới nơi khác. Nhưng tôi sẽ làm nếu phải như vậy :o)

11
Quãng nghỉ giữa các Sprint

Trong thực tế cuộc sống, bạn không thể triển khai chạy nước rút[14] liên tục. Bạn cần phải có thời gian nghỉ ngơi giữa các lần chạy. Nếu bạn cứ liên tục chạy nước rút, bạn sẽ nhận được kết quả không khác gì việc đi bộ.

Những điều trên tương tự với Scrum hay phát triển phần mềm nói chung. Các Sprint diễn ra với cường độ khá dồn dập. Với vai trò là nhà phát triển bạn không bao giờ có được sự nghỉ ngơi thực sự, hằng ngày bạn phải đứng trong một cuộc họp đáng ghét và nói với mọi người những gì bạn đã làm được trong ngày hôm qua. Ít khi có khuynh hướng để nói rằng "tôi sử dụng hầu hết thời gian gác chân lên bàn và duyệt blog và nhấm nháp tách cà phê cappuccino".

Không chỉ đơn thuần là thời gian để thư giãn, có những lý do chính đáng để cần có những quãng nghỉ giữa các Sprint. Sau buổi Sơ kết Sprint và Cải tiến, cả Nhóm và Product Owner sẽ có nhiều thông tin và ý tưởng để suy xét và thấu hiểu. Nếu ngay lập tức họ bỏ qua và bắt đầu lập kế hoạch luôn cho Sprint tiếp theo, kết quả là chẳng ai có cơ hội để suy xét bất cứ thông tin nào hoặc học được bài học gì, Product Owner sẽ không có thời gian để điều chỉnh các ưu tiên của mình sau buổi demo, v.v.

Tồi:

Thứ 2

09h–10h: Sơ kết Sprint 1
10h–11h: Họp Cải tiến Sprint 1

13h–16h: Họp Kế hoạch Sprint 2

Chúng tôi cố gắng đưa ra một số hình thức nghỉ ngơi trước khi bắt đầu một Sprint mới (cụ thể hơn là giai đoạn sau khi tiến hành Họp Cải tiến Sprint và trước khi Họp Kế hoạch Sprint tiếp theo). Mặc dù chúng tôi không phải lúc nào cũng thành công với việc này.

---

[14] Sprint trong tiếng anh có nghĩa là "chạy nước rút", đây là một thuật ngữ trong môn bóng bầu dục (rugby.

Ít nhất, chúng tôi cũng cố gắng để đảm bảo rằng việc họp cải tiến Sprint và lập kế hoạch cho Sprint tiếp theo không tổ chức trong cùng một ngày. Mọi người ít ra cần phải có một đêm ngon giấc trước khi bắt tay vào một Sprint mới.

Tốt hơn:

Thứ 2	Thứ 3
09h–10h: Sơ kết Sprint 1 10h–11h: Họp Cải tiến Sprint 1	09h–13h: Họp Kế hoạch Sprint 2

Tốt hơn nữa:

Thứ 6	Thứ 7	Chủ nhật	Thứ 2
09h–10h: Sơ kết Sprint 1 10h–11h: Họp Cải tiến Sprint 1			09h–13h: Họp Kế hoạch Sprint 2

Có một cách để làm được điều này đó là "những ngày thí nghiệm" (lab days) (hoặc bất cứ tên gọi gì mà bạn thích). Đó là những ngày mà các nhà phát triển được phép làm bất cứ điều gì họ muốn (OK, tôi thừa nhận là đã lấy cảm hứng này từ Google). Ví dụ, đọc và tìm hiểu các công cụ và API mới nhất, học để thi chứng chỉ, thảo luận linh tinh với đồng nghiệp, viết mã nguồn cho một dự án yêu thích nào đó, v.v.

Mục tiêu của chúng tôi là phải có một "lab day" giữa các Sprint. Bằng cách đó, bạn tự nhiên sẽ có được quãng nghỉ giữa các Sprint, và bạn sẽ giúp Nhóm phát triển có cơ hội thực tế để cập nhật kiến thức thường xuyên. Hơn nữa, việc này sẽ làm tăng tính hấp dẫn cho công việc.

Tốt nhất?

Thứ 5	Thứ 6	Thứ 7	Chủ nhật	Thứ 2
09h–10h: Sơ kết Sprint 1 10h–11h: Họp Cải tiến Sprint 1	LAB DAY			09h–13h: Họp Kế hoạch Sprint 2

Hiện tại chúng tôi thực hiện "lab days" một tháng một lần. Ngày này được ấn định vào Thứ Sáu đầu tiên trong tháng. Tại sao không để ngày đó giữa các Sprint? Vâng, bởi vì tôi cảm thấy một điều quan trọng đó là toàn bộ công ty đều có "lab day" vào cùng một thời điểm. Nếu không một số người sẽ có xu hướng không thực hiện nó một cách nghiêm túc. Và bởi vì chúng tôi (cho đến nay)

không có sự liên kết các Sprint giữa các sản phẩm khác nhau, tôi phải chọn một "lab day" độc lập với Sprint để thay thế.

Chúng tôi có thể một ngày nào đó thử đồng bộ các Sprint của tất cả các sản phẩm (tức là các Sprint của các sản phẩm và Nhóm khác nhau cùng có ngày bắt đầu và kết thúc giống nhau). Trong trường hợp đó chúng tôi sẽ xác định một "lab day" giữa các Sprint.

12

Chúng tôi lập Kế hoạch Phát hành và định giá các hợp đồng như thế nào

Đôi khi tại một thời điểm nào đó chúng ta cần phải lập kế hoạch xa hơn kế hoạch của một Sprint. Thông thường kết hợp với việc định giá hợp đồng là lúc chúng ta phải hoạch định trước, hoặc nếu không khi một vài dấu hiệu rủi ro xuất hiện có thể khiến chúng không thể bàn giao sản phẩm đúng hẹn.

Thường thì, lập kế hoạch phát hành giúp chúng ta cố gắng trả lời được câu hỏi "*khi nào*, hoặc *muộn nhất,* chúng ta sẽ có thể chuyển giao được phiên bản 1.0 của hệ thống mới này"

Nếu bạn *thực sự* muốn học về cách lập kế hoạch phát hành, tôi gợi ý bạn hãy bỏ qua chương này và thay vào đó bạn mua cuốn "Ước tính và Lập kế hoạch linh hoạt" (Agile Estimating and Planning) của Mike Cohn. Tôi thực sự ước rằng mình được đọc cuốn sách đó sớm hơn (Tôi đã đọc nó *sau khi* chúng tôi phải tìm kiếm những thứ mà chúng tôi không có…). Phương án lập kế hoạch phát hành của tôi hơi có vẻ "đơn giản hóa" một chút nhưng sẽ đó là một điểm khởi đầu tốt.

Xác định ngưỡng chấp nhận của bạn

Ngoài ra, bên cạnh Product Backlog, Product Owner thường đưa ra một danh sách các *ngưỡng chấp nhận*, đó là một nhóm đơn giản về những mức quan trọng trong Product Backlog có ý nghĩa thực sự đối với các điều khoản của hợp đồng.

Dưới đây là một ví dụ về các quy định về ngưỡng chấp nhận:

- Tất cả các hạng mục có độ quan trọng >=100 phải được hoàn thành trong phiên bản 1.0, hoặc nếu không chúng ta sẽ bị trừng phạt cho tới khi chấm dứt.
- Tất cả các hạng mục có độ quan trọng 50-99 cần được đưa vào phiên bản 1.0, nhưng chúng ta *có thể* có cách để triển khai chúng trong một bản phát hành nhanh ngay sau đó.
- Những hạng mục có độ quan trọng 25 – 49 là cần thiết, nhưng có thể hoàn thành chúng trong bản phát hành 1.1 sau đó.

- Các hạng mục có độ quan trọng <25 là những hạng mục đầu cơ và có thể sẽ không bao giờ cần triển khai chúng.

Và đây là một ví dụ về Product Backlog, sử dụng màu sắc để thiết lập các quy ước trên.

Độ quan trọng	Tên
130	chuối
120	táo
115	cam
110	ổi
100	lê
95	nho khô
80	đậu phộng
70	bánh rán
60	hành
40	bưởi
35	đu đủ
10	việt quất
10	đào

Màu đỏ = phải được đưa vào phiên bản 1.0 (từ "chuối" đến "lê")

Màu vàng = nên có trong bản 1.0 (từ "nho khô" đến "hành tây")

Màu xanh = có thể hoàn thành muộn hơn (từ "bưởi" đến "đào")

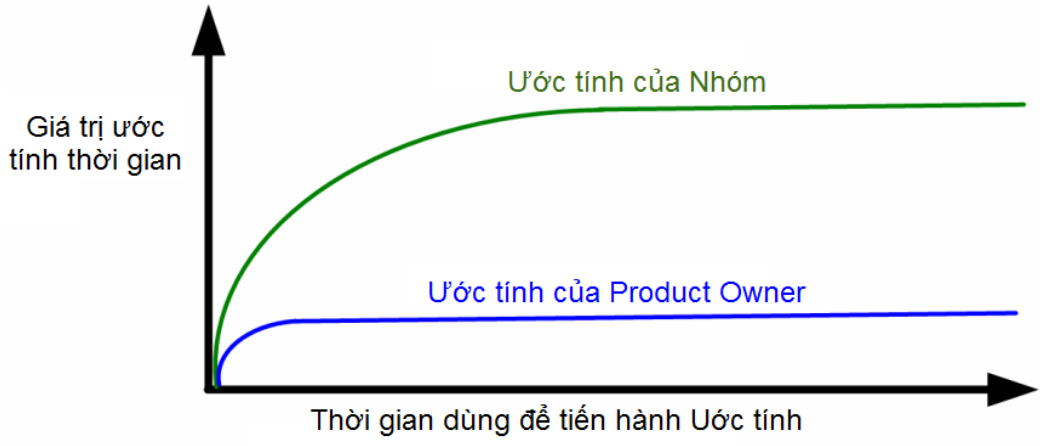
Vì vậy nếu chúng tôi hoàn thành được mọi thứ từ "chuối" đến "hành" đúng thời hạn là chúng tôi an toàn. Nếu thời gian thời gian triển khai ngắn chúng tôi *có thể* tiến hành bằng cách bỏ đi "nho khô", "đậu phộng", "bánh rán" và "hành tây". Tất cả mọi thứ có độ quan trọng thấp hơn "hành" là phần gia tăng thêm.

Ước tính thời gian cho những hạng mục quan trọng nhất

Để Product Owner có thể thực hiện được việc lập kế hoạch phát hành, ít nhất tất cả các story có trong hợp đồng đều đã được ước tính. Cũng giống như khi lập kế hoạch Sprint, đây là nỗ lực hợp tác giữa Product Owner và Nhóm – Nhóm thực hiện ước tính, Product Owner mô tả các hạng mục và trả lời các câu hỏi.

Ước tính thời gian là có *giá trị* nếu nó sát với thực tế thực hiện, ít có giá trị nếu ngược lại, chỉ tiêu là 30%, và hoàn toàn không có giá trị gì nếu nó không có bất cứ liên hệ gì với thực tế.

Đây là những thu nhận của tôi về giá trị của một ước tính thời gian quan hệ với người thực hiện ước tính và thời gian họ sử dụng để tiến hành việc này.



Tất cả những điều này chỉ là cách nói dài dòng về:

- Hãy để *Nhóm* thực hiện việc ước tính.
- Không để họ mất quá nhiều thời gian cho việc này.
- Đảm bảo rằng họ hiểu rằng các ước tính thời gian là *các ước tính thô,* chứ không phải là *các cam kết.*

Thông thường Product Owner tập hợp toàn bộ nhóm trong một căn phòng, cung cấp một số đồ ăn nhẹ, và nói với họ rằng mục tiêu của cuộc họp này là để ước-tính-thời-gian của 20 story (hoặc bất kỳ một số lượng nào) hàng đầu trong Product Backlog. Anh ta duyệt qua một lượt tất cả các story đó, và sau đó để nhóm bắt đầu làm việc. Product Owner ngồi lại trong phòng để trả lời các câu hỏi và làm rõ phạm vi của mỗi hạng mục khi cần thiết. Giống như khi tiến hành lập kế hoạch Sprint, trường "Cách để demo" (trong Product Backlog) là cách rất hữu ích để giảm thiểu những rủi ro khi không hiểu hạng mục đó.

Cuộc họp phải tuân thủ khung-thời-gian, nếu không nhóm sẽ có xu hướng tốn quá nhiều thời gian mà chỉ ước tính được một vài story.

Nếu Product Owner muốn có thêm thời gian cho việc này anh ta đơn giản chỉ cần đặt thêm một lịch họp sau đó. Nhóm phải chắc chắn về ảnh hưởng của những cuộc họp đó với Sprint hiện tại của họ được hiện diện rõ ràng cho Product Owner, vì vậy anh ta hiểu rằng công việc ước-tính-thời-gian của nhóm không diễn ra một cách tự do được.

Độ quan trọng	Tên	Ước tính
130	chuối	12
120	táo	9
115	cam	20
110	ổi	8
100	lê	20
95	nho khô	12
80	đậu phộng	10
70	bánh rán	8
60	hành	10
40	bưởi	14
35	đu đủ	4
10	việt quất	
10	đào	

Dưới đây là ví dụ về việc ước tính thời gian sẽ như thế nào sau khi kết thúc phiên họp (ước tính theo điểm – story point):

Tốc độ Ước tính

OK, vậy là bây giờ chúng tôi đã có một số những ước tính thời gian thô cho các story quan trọng nhất.

Bước tiếp theo là ước tính tốc độ trung bình của chúng tôi cho mỗi Sprint.

Điều này có nghĩa là chúng tôi cần quyết định trên các hệ số tập trung. Hãy xem hình 24 “Cách nhóm quyết định story nào được đưa vào Sprint”.

Hệ số tập trung cơ bản là “Nhóm sử dụng bao nhiêu thời gian để tập trung cho những story đã cam kết hiện nay”. Sẽ không bao giờ là 100% bởi vì Nhóm còn phải dành thời gian để thực hiện những việc không có trong kế hoạch, thực hiện những bước chuyển về ngữ cảnh, trợ giúp các nhóm khác, gửi nhận email, sửa lỗi xảy ra với máy tính, tranh luận về chính trị trong nhà ăn, v.v.

Phải nói rằng chúng tôi xác định hệ số tập trung cho nhóm là 50% (khá thấp, chúng tôi thường du di trong khoảng 70%). Và chúng tôi sử dụng thời gian 3 tuần (15 ngày) cho Sprint của mình với nhóm gồm 6 thành viên.

Mỗi Sprint như vậy tương đương với 90 ngày công (man-day), nhưng có thể chỉ dùng 45 ngày công dành cho các story (do hệ số tập trung của nhóm là 50%).

Vì vậy tốc độ ước tính của chúng tôi là 45 điểm (story point).

Nếu mỗi story có ước tính thời gian là 5 ngày (thực tế thì có thể không phải) khi đó nhóm sẽ thực hiện được khoảng 9 story mỗi Sprint.

Đặt vào cùng trong một kế hoạch phát hành

Bây giờ chúng ta đã có các ước tính thời gian và tốc độ ước tính (45), chúng ta có thể dễ dàng phân tách các hạng mục trong Product Backlog vào các Sprint:

Độ quan trọng	Tên	Ước tính
		Sprint 1
130	chuối	12
120	táo	9
115	cam	20
		Sprint 2
110	ổi	8
100	lê	20
95	nho khô	12
		Sprint 3
80	đậu phộng	10
70	bánh rán	8
60	hành	10
40	bưởi	14
		Sprint 4
35	đu đủ	4
10	việt quất	

Mỗi Sprint bao gồm càng nhiều story càng tốt mà không vượt quá tốc độ ước tính là 45 điểm.

Lúc này chúng ta có thể thấy rằng chúng ta có thể cần đến 3 Sprint để hoàn thành tất cả các hạng mục "phải có" và "cần có".

3 Sprint = 9 tuần lịch = 2 tháng lịch. Bây giờ thì đó là thời hạn mà chúng ta sẽ hứa hẹn với khách hàng? Phụ thuộc hoàn toàn vào bản chất của hợp đồng; phạm vi được phép, v.v. Chúng ta thường thêm vào một số vùng đệm để phòng trừ trường hợp ước tính thời gian không tốt, các vấn đề và các tính năng đột xuất, v.v.

Mọi thứ sẽ tốt đẹp khi chúng ta có thể demo được một số tính năng sử dụng cho khách hàng 3 tuần một lần và mời họ thay đổi các yêu cầu khi mà chúng ta vẫn có thể (dĩ nhiên là phụ thuộc vào hợp đồng ra sao).

Hiệu chỉnh kế hoạch phát hành để thích nghi

Thực tế sẽ không tự nó thích nghi với kế hoạch, vì vậy phải có cách nào đó để giải quyết.

Sau mỗi Sprint chúng ta xem xét lại tốc độ thực tế trong Sprint đó. Nếu tốc độ thực tế rất chênh lệch so với tốc độ ước tính, chúng ta điều chỉnh lại tốc độ ước tính cho các Sprint trong tương lai và cập nhật lại kế hoạch phát hành. Nếu điều này đẩy chúng ta vào một rắc rối, Product Owner có thể phải tiến hành đàm phán với khách hàng hoặc kiểm tra xem anh ta có thể có cách nào để giảm bớt phạm vi mà không phá hỏng hợp đồng. Hoặc có thể anh ta và nhóm đưa ra một số cách để tăng tốc độ hoặc giảm bớt hệ số tập trung bằng cách bỏ bớt một số trở ngại nghiêm trọng phát sinh trong quá trình triển khai Sprint.

Product Owner có thể liên lạc với khách hàng và nói rằng "xin chào, chúng tôi đang chậm tiến độ một chút nhưng tôi tin rằng chúng tôi có thể đảm bảo về thời hạn nếu chỉ cần bỏ đi tính năng 'nhúng Pacman' đó là cái tốn rất nhiều thời gian để xây dựng. Chúng tôi có thể thêm vào tính năng đó ở bản phát hành ngay sau bản phát hành đầu tiên 3 tuần nếu anh muốn".

Có thể chẳng có tin tức tốt đẹp nào từ khách hàng, nhưng ít nhất chúng ta đang trung thực và cung cấp sớm cho khách hàng lựa chọn – hoặc là chúng ta cung cấp những tính năng quan trọng nhất đúng hẹn hoặc mọi thứ nhưng sẽ muộn hơn. Thường thì đó không phải là một lựa chọn khó khăn :o)

13

Chúng tôi kết hợp Scrum với XP ra sao

Việc nói rằng Scrum và XP (eXtreme Programming) có thể kết hợp với nhau hiệu quả thực sự không còn là một vấn đề gây tranh cãi. Hầu hết các thông tin tôi thấy từ mạng đều hỗ trợ cho giả thiết này, vì vậy tôi không muốn lãng phí thời gian để tranh luận lý do tại sao.

Vâng, tôi sẽ đề cập đến một điều. Scrum chú trọng vào các vấn đề quản lý và tổ chức trong khi XP lại quan tâm chủ yếu vào đến các phương thức thực hành lập trình trong thực tế. Đó là lý do tại sao chúng có thể kết hợp tốt được với nhau – chúng giải quyết các "bài toán" khác nhau và bổ sung cho nhau.

Tôi xin phép được đưa thêm vào những bằng chứng thực nghiệm đã có về sự kết hợp có hiệu quả giữa Scrum và XP!

Tôi sẽ làm nổi bật một số phương pháp thực hành XP có giá trị hơn và cách áp dụng chúng trong công việc hằng ngày của chúng tôi. Không phải tất cả các nhóm của chúng tôi đều được quản lý để áp dụng tất cả các phương pháp thực hành này, nhưng xét một cách tổng quan chúng tôi đã trải nghiệm với hầu hết những khía cạnh trong việc kết hợp giữa XP và Scrum. Một số phương pháp thực hành trong XP được trực tiếp đề cập bởi Scrum và có thể xem như có sự chồng lấp lẫn nhau, ví dụ như "Toàn bộ nhóm" (Whole Team), "Ngồi cùng nhau" (Sit Together), "Các story" (Stories), và "Trò chơi Lập kế hoạch" (Planning game). Trong những trường hợp này chúng tôi coi như đó là Scrum.

Lập trình cặp (Pair programming)

Gần đây chúng tôi bắt đầu triển khai hoạt động này cho một trong số các nhóm thuộc công ty. Việc làm này thực sự là khá hiệu quả. Hầu hết các nhóm khác vẫn không triển khai được nhiều về lập trình cặp, nhưng thực sự hiện nay đã có những cố gắng triển khai ở một nhóm cho một vài Sprint, tôi đang có hứng thú để thử tiến hành huấn luyện phương pháp này cho nhiều nhóm hơn.

Một số kết luận của tôi đến thời điểm này về lập trình cặp:

- Lập trình cặp giúp tăng cường chất lượng cho mã nguồn.
- Lập trình cặp làm tăng khả năng tập trung của nhóm (ví dụ khi một anh chàng đứng sau bạn nói rằng "này, những thứ này có thực sự cần thiết cho Sprint này không?").

- Một điều đáng ngạc nhiên đó là nhiều nhà phát triển phản đối quyết liệt lập trình cặp là do họ thực sự không thử triển khai chúng, và họ nhanh chóng thích nó khi họ thử triển khai một lần.
- Lập trình cặp rất mệt mỏi và không nên thực hành nó suốt cả ngày.
- Đổi cặp thường xuyên sẽ hiệu quả hơn.
- Lập trình cặp làm tăng khả năng truyền bá kiến thức trong nhóm. Với tốc độ đáng ngạc nhiên.
- Một số người thấy không thoải mái khi lập trình cặp. Đừng gạt ra ngoài một lập trình viên tuyệt vời chỉ bởi vì anh ta không cảm thấy thoải mái với lập trình cặp.
- Rà soát mã nguồn (code review) là một lựa chọn tạm chấp nhận được để thay thế lập trình cặp.
- "Hoa tiêu" (navigator) (người không sử dụng đến bàn phím) tốt nhất nên có một máy tính riêng. Máy tính này không dùng cho việc phát triển, nhưng dùng cho một số việc nhỏ khi cần thiết, ví dụ như mở tài liệu khi "lái xe" (driver) (người sử dụng bàn phím) gặp phải khó khăn, v.v.
- Không nên ép buộc lập trình cặp với mọi người. Khuyến khích và cung cấp các công cụ chuẩn nhưng hãy để họ trải nghiệm theo cách riêng của mỗi người.

Phát triển Hướng-Kiểm-thử (Test-driven development)

Amen! Đối với tôi, đây là thứ quan trọng hơn cả Scrum và XP. Bạn có thể lấy đi căn nhà, ti vi và chú cún của tôi, nhưng đừng cố gắng ngăn tôi triển khai TDD (Test-Driven Development)! Nếu bạn không thích TDD xin đừng bắt tôi phải theo bạn, bởi vì tôi sẽ cố lén lút triển khai bằng cách này hay cách khác :o)

Đây là một tổng kết ngắn gọn về TDD:

Phát triển Hướng-Kiểm-thử có nghĩa là bạn viết kiểm thử tự động trước, sau đó bạn viết mã nguồn đủ để đạt được kiểm thử đó, tiếp theo bạn cải tiến mã nguồn ban đầu để tăng khả năng đọc hiểu và loại bỏ mã nguồn bị trùng lặp. Tiếp tục lặp lại quá trình đó.

Một số nhận xét về Phát triển Hướng-Kiểm-thử.

- TDD *không dễ triển khai*. Phải mất thời gian để các lập trình *làm quen với nó*. Trong thực tế, nhiều trường hợp bạn dạy, huấn luyện và demo bao nhiêu không thực sự là vấn đề - trong nhiều trường hợp chỉ có cách để cho lập trình viên *biết được TDD* là yêu cầu anh ta lập trình cặp với một ai đó giỏi về TDD. Tuy nhiên, một khi lập trình viên *biết TDD,* anh ta sẽ thường thích thú với nó và sẽ không bao giờ muốn làm việc với bất kỳ cách nào khác.
- TDD có ảnh hưởng tích cực và sâu sắc đối với thiết kế hệ thống.
- Phải tốn thời gian mới có thể triển khai tốt được TDD và sử dụng nó hiệu quả cho một sản phẩm mới, đặc biệt là các kiểm thử tích hợp, nhưng một điều *chắc chắn* là nó mang lại hiệu quả.
- Hãy chắc chắn rằng bạn đầu tư thời gian cần thiết để làm cho việc viết kiểm thử trở nên *dễ dàng*. Điều này có nghĩa là nhận được những công cụ chuẩn, huấn luyện mọi người, cung cấp cho nhóm các lớp tiện ích chuẩn hoặc lớp căn bản, v.v.

Chúng tôi sử dụng những công cụ dưới đây cho Phát triển Hướng-Kiểm-thử:

- jUnit\httpUnit\jWebUnit. Chúng tôi cũng đang xem xét TestNG và Selenium.
- HSQLDB được dùng như là một Cơ sở dữ liệu nhúng trong-bộ-nhớ dành cho việc kiểm thử.
- Jetty là một web container (môi trường thực thi ứng dụng web) nhúng trong-bộ-nhớ dành cho mục đích kiểm thử.
- Cobertura dùng để cung cấp các số liệu về test coverage[15].
- Khung làm việc Spring dùng để kết nối các loại cơ cấu kiểm thử khác nhau (với những đối tượng mô phỏng, không có mô phỏng, với cơ sở dữ liệu mở rộng, với cơ sở dữ liệu trong bộ nhớ, v.v.).

Trong những sản phẩm phức tạp nhất (xét theo quan điểm TDD), chúng tôi có những kiểm thử chấp nhận hộp-đen tự động hóa (automated black-box acceptance tests). Những kiểm thử đó khởi động toàn bộ hệ thống trong bộ nhớ, bao gồm các cơ sở dữ liệu và các dịch vụ web, và truy xuất hệ thống chỉ sử dụng những giao diện công cộng của chúng (ví dụ như HTTP).

---

[15] Còn gọi là code coverage, là độ đo mức độ được kiểm thử của mã nguồn. Đây là một trong những tiếp cận hệ thống đầu tiên và cơ bản về kiểm thử phần mềm. Mức độ code coverage càng lớn, nhóm có thể càng tự tin hơn về chất lượng của mã nguồn.

Điều này tạo ra chu trình Phát triển – Build - Kiểm thử diễn ra nhanh vô cùng. Đồng thời cũng hoạt động như một mạng lưới an toàn, đem lại sự tự tin cho các nhà phát triển đủ để họ thường xuyên cải tiến, có nghĩa là thiết kế vẫn rõ ràng và đơn giản ngay cả khi hệ thống lớn lên.

TDD với mã nguồn mới

Chúng tôi thực hành TDD cho tất cả các dự án phát triển mới, thậm chí nếu điều này có nghĩa là tốn kém thời gian ngay lần đầu thiết lập dự án (vì chúng tôi cần nhiều công cụ và hỗ trợ hơn dành cho việc sử dụng kiểm thử, v.v.). Không cần nhiều trí tuệ cho điều đó, trong khi lại thu được những lợi ích tuyệt vời thì thực sự chẳng có lý do gì để không triển khai TDD.

TDD với mã nguồn cũ

TDD không dễ, nhưng cố gắng triển khai nó với những mã nguồn không sử dụng TDD ngay từ đầu… thì *thực sự là khó khăn*! Tai sao vậy? Vâng, thực tế, tôi có thể viết nhiều hơn về chủ đề này vậy nên tôi nghĩ mình sẽ dừng ở đây. Tôi sẽ để dành nó cho cuốn sách tiếp theo của tôi là "TDD từ những Chiến hào" :o)

Chúng tôi đã mất khá nhiều thời gian để cố gắng tự động hóa quá trình kiểm thử tích hợp ở một trong số những hệ thống phức tạp của chúng tôi, nền tảng mã nguồn được xây dựng vội vàng và trong tình trạng có những sai sót nghiêm trọng và hoàn toàn không có bất cứ một kiểm thử nào.

Đối với mọi bản phát hành của hệ thống chúng tôi có một nhóm chuyên gia kiểm thử, họ sẽ thực hiện toàn bộ một nhóm các kiểm thử hồi quy và các kiểm thử hiệu năng phức tạp. Các kiểm thử hồi quy (regression tests) hầu như được thực hiện thủ công. Điều này làm chậm đáng kể chu trình phát triển và phát hành của chúng tôi. Mục tiêu của chúng tôi là tự động hóa các kiểm thử đó. Tuy nhiên, sau khi vò đầu bứt tai một vài tháng, chúng tôi thực sự không đạt được nhiều như mong đợi.

Sau đó chúng tôi chuyển sang tiếp cận theo hướng khác. Chúng tôi thừa nhận một sự thật rằng chúng tôi đang mắc kẹt với việc kiểm thử hồi quy thủ công, và thay vào đó chúng tôi bắt đầu tự hỏi "Làm thế nào để việc xử lý kiểm thử thủ công tốn ít thời gian hơn?" Đây là một hệ thống chơi game, và chúng tôi nhận ra rằng nhóm kiểm thử đã mất nhiều thời gian tiến hành thiết lập những thứ ít có giá trị, như đi quanh các phòng hành chính để thiết lập các giải đấu dành cho mục đích kiểm thử, hoặc chờ đợi cho tới khi giải đấu được bắt đầu. Vì vậy chúng tôi đã tạo ra những tiện ích cho điều đó. Nhỏ gọn, dễ dàng truy xuất thông qua các biểu tượng tắt và kịch bản để làm tất cả các công việc khó khăn và giúp nhóm kiểm thử tập trung vào thử nghiệm thực tế.

Nỗ lực đó đã thực sự được đền đáp! Trong thực tế, điều đó có thể là những gì mà chúng tôi phải hoàn thành từ khi bắt đầu triển khai dự án. Chúng tôi đã quá háo hức với việc tự động hóa kiểm

thử mà quên mất rằng cần phải làm điều đó từng bước một, trong đó bước đầu tiên là xây dựng những thứ giúp cho việc kiểm thử *thủ công* hiệu quả hơn.

Bài học rút ra: Nếu bạn mắc kẹt với việc phải tiến hành kiểm thử hồi quy thủ công, và muốn tự động hóa công việc này, xin hãy đừng làm (trừ khi nó thực sự dễ dàng). Thay vào đó, hãy xây dựng những thứ giúp công việc thủ công đó được thực hiện dễ dàng hơn. *Sau đó* mới xem xét đến việc tự động hóa kiểm thử trong thực tế.

Thiết kế tiến hóa (Incremental Design)

Thiết kế tiến hóa có nghĩa là giữ cho việc thiết kế đơn giản khi mới bắt đầu và tiếp tục cải thiện nó, hơn là cố gắng để có những tất cả những điều đúng đắn ngay từ đầu và sau đó không còn quan tâm đến nữa.

Hiện tại, chúng tôi đang làm khá tốt việc này, tức là chúng tôi sử dụng khoảng thời gian hợp lý để cải tiến và cải thiện những thiết kế đã có, và chúng tôi hiếm khi bỏ thời gian để tiến hành thiết kế lớn ngay từ đầu. Đôi khi, tất nhiên là chúng tôi siết chặt chuyện này, ví dụ bằng việc cho phép một thiết kế yếu để "thúc" mạnh việc cải tiến nó trở thành một dự án lớn. Tuy nhiên tất cả chúng tôi đều khá hài lòng.

Cải tiến thiết kế liên tục là kết quả tất yếu của việc thực hiện TDD.

Tích hợp liên tục (Continuous Integration)

Hầu hết các sản phẩm của chúng tôi đều được thiết lập tích hợp liên tục khá phức tạp dựa trên Maven và QuickBuild. Điều này cực kỳ có giá trị và tiết kiệm thời gian. Đây là giải pháp cơ bản cho vấn đề "này, nhưng nó chạy tốt trên máy *của tôi* mà". Máy chủ build liên tục (continuous build server) của chúng tôi hoạt động như một "thẩm phán" hoặc điểm tham chiếu để xác định chất lượng của tất cả các mã nguồn nền tảng. Mỗi khi một ai đó cập nhật vào hệ thống quản lý phiên bản, dịch vụ build liên tục sẽ hoạt động, build mọi thứ từ đầu trên máy chủ chia sẻ, và thực thi tất cả các kiểm thử. Nếu bất cứ lỗi nào xảy ra, nó sẽ gửi email cảnh báo tới các thành viên trong nhóm về việc build không thành công, thêm vào đó là các thông tin chính xác những thay đổi trong mã nguồn dẫn đến thất bại, liên kết tới báo cáo kiểm thử, v.v.

Hằng đêm máy chủ build liên tục sẽ build lại sản phẩm từ đầu và đẩy lên cổng điện tử tài liệu nội bộ của chúng tôi các tệp đã biên dịch (ví dụ ear, war, v.v.), tài liệu, các báo cáo kiểm thử, báo cáo phạm vi kiểm thử, báo cáo thành phần phụ thuộc, v.v. Một số sản phẩm cũng sẽ được triển khai tự động vào môi trường kiểm thử.

Mất rất nhiều công sức để thiết lập một hệ thống tích hợp liên tục, nhưng chúng giá trị đến từng phút giây.

Sở hữu mã nguồn tập thể

Chúng tôi khuyến khích sở hữu mã nguồn tập thể nhưng không phải nhóm nào cũng tuân thủ được điều này. Chúng tôi đã nhận thấy rằng lập trình cặp với sự luân chuyển thường xuyên các cặp sẽ tự động dẫn tới việc sở hữu mã nguồn tập thể ở mức độ cao. Các nhóm có sự sở hữu mã nguồn tập thể ở mức độ cao đã được chứng minh là rất mạnh, ví dụ Sprint của họ không bao giờ thất bại chỉ vì lý do một vài người quan trọng bị ốm.

Không gian làm việc giàu thông tin

Tất cả các nhóm sử dụng bảng trắng và các bức tường trống và làm cho việc sử dụng chúng tốt hơn. Trong hầu hết các căn phòng, bạn sẽ tìm thấy những bức tường được dán đầy những thông tin về sản phẩm và dự án. Vấn đề lớn nhất là có những thứ không còn giá trị vẫn ở trên các bức tường đó, cho nên chúng tôi có thể đưa ra thêm một vai trò là "quản gia" (housekeeper) trong mỗi nhóm.

Chúng tôi khuyến khích sử dụng bảng công việc (taskboard), nhưng không phải nhóm nào cũng làm được điều này. Xem trang 70, *"Chúng tôi bố trí phòng làm việc như thế nào"*.

Tiêu chuẩn mã nguồn

Gần đây chúng tôi đã bắt đầu định nghĩa chuẩn cho viết mã nguồn. Điều này rất hữu ích, ước gì chúng tôi triển khai được sớm hơn. Hầu như không mất thời gian cho việc này, chỉ cần bắt đầu đơn giản và hãy để nó phát triển dần lên. Chỉ viết những thứ không rõ ràng với mọi người và liên kết chúng tới những thứ đã có bất kỳ khi nào có thể.

Hầu hết các lập trình viên đều có phong cách riêng trong viết mã nguồn. Những chi tiết nhỏ như cách họ xử lý các ngoại lệ, cách họ chú thích mã nguồn, khi họ trả về null, v.v. Trong một số trường hợp những khác biệt đó không là vấn đề, trong một số trường hợp khác nó lại có thể dẫn tới việc thiết kế hệ thống thiếu nhất quán nghiêm trọng và mã nguồn khó hiểu. Chuẩn viết mã nguồn rất hữu ích ở đây, miễn là bạn tập trung vào những thứ có vấn đề.

Sau đây là một vài ví dụ về chuẩn viết mã nguồn của chúng tôi:

- Bạn có thể phá bỏ bất cứ quy tắc gì, nhưng phải đảm bảo có lý do chính đáng và phải viết tài liệu mô tả.
- Mặc định sử dụng quy ước viết mã nguồn của Sun: http://iava.sun.com/docs/codeconv/html/CodeConvTOC.doc.html
- Không bao giờ, bắt các ngoại lệ mà không ghi lại dấu vết hoặc đẩy tiếp đi. Dùng phương thức `log.debug()` là được, chỉ cần không để mất dấu vết.

- Sử dụng dependency injection dựa trên setter để tách biệt các lớp với nhau (tất nhiên trừ khi không muốn tách biệt).
- Tránh dùng từ viết tắt. Nên dùng những từ viết tắt phổ biến, ví dụ như DAO.
- Các phương thức trả về `Collection` (tập hợp) hoặc mảng không nên "return null". Nên trả về tập hợp hoặc mảng rỗng thay vì null.

Làm việc với tốc độ\năng lượng bền vững

Nhiều cuốn sách về phát triển phần mềm linh hoạt cảnh báo rằng làm việc ngoài giờ (overtime) sẽ làm giảm năng suất trong phát triển phần mềm.

Sau một số thử nghiệm không mong muốn về vấn đề này tôi chỉ có thể nói rằng tôi hết sức đồng ý với khuyến cáo đó!

Khoảng một năm về trước, một nhóm của chúng tôi (nhóm lớn nhất) đã làm việc ngoài giờ rất nhiều. Chất lượng của mã nguồn mà họ làm ra rất tệ và họ phải sử dụng hầu hết thời gian để chữa cháy. Nhóm kiểm thử (cũng làm việc ngoài giờ) không có cơ hội để thực hiện bất kỳ một phương án bảo đảm chất lượng nào một cách đúng đắn. Khách hàng của chúng tôi tức giận và các tờ báo lá cải thì xông vào "ăn tươi nuốt sống" chúng tôi.

Sau một vài tháng chúng tôi đã quản lý để giảm giờ làm việc của mọi người xuống mức hợp lý. Mọi người làm việc với số giờ làm việc bình thường (trừ một số thời điểm trong dự án). Và, thật ngạc nhiên, năng suất và chất lượng được cải thiện đáng kể.

Tất nhiên, giảm giờ làm việc không có nghĩa đó là *khía cạnh duy nhất* dẫn đến sự cải thiện này, nhưng tất cả chúng tôi đều tin chắc nó chiếm phần lớn trong đó.

14
Chúng tôi kiểm thử như thế nào

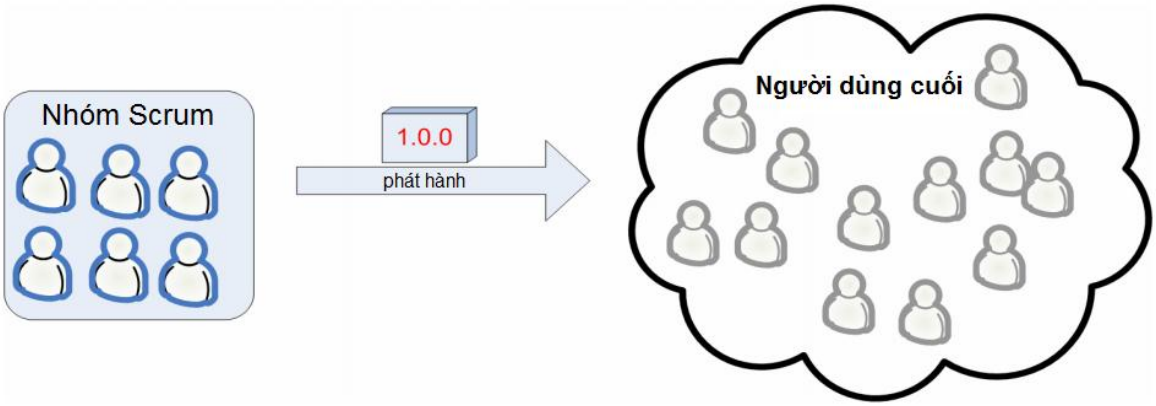
Đây là phần khó khăn nhất. Tôi không chắc chắn rằng nó là phần khó nhất của Scrum, hoặc chỉ là phần khó nhất trong phát triển phần mềm nói chung.

Kiểm thử có thể là công đoạn khác nhau nhất giữa các tổ chức. Phụ thuộc vào việc bạn có bao nhiêu kiểm thử viên, mức độtự động hóa kiểm thử mà bạn đang triển khai, hệ thống của bạn thuộc loại gì (chỉ là dịch vụ web? hoặc bạn có thường chuyển giao phần mềm đóng gói không?), thời gian của các chu kỳ phát hành, phần mềm quan trọng tới mức nào (dịch vụ blog đơn giản hay hệ thống điều khiển chuyến bay), v.v.

Chúng tôi đã thử nghiệm khá nhiều với các cách thức để kiểm thử trong Scrum. Tôi sẽ cố gắng mô tả những gì mà chúng tôi đã thực hiện và những gì mà chúng tôi học được từ đó.

Bạn có lẽ sẽ không thể bỏ được giai đoạn kiểm thử chấp nhận

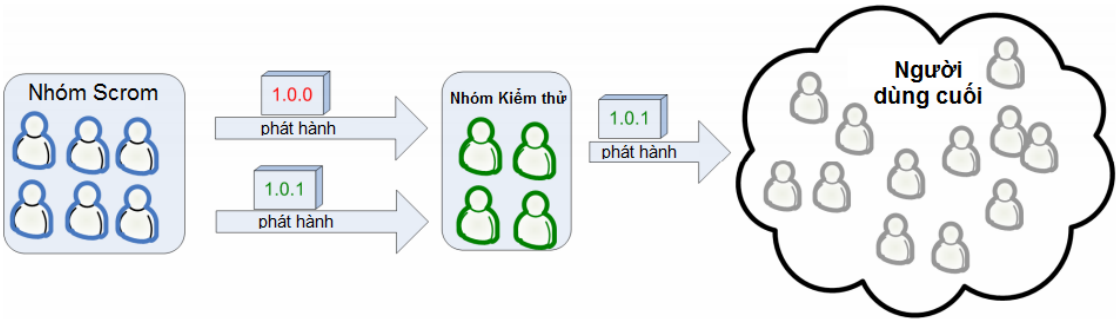
Trong thế giới lý tưởng của Scrum, kết quả của mỗi Sprint là một phiên bản có khả năng triển khai của hệ thống bạn đang phát triển. Vậy thì chỉ cần triển khai nó, đúng không?



Sai rồi.

Kinh nghiệm của chúng tôi cho thấy rằng cái đó thường không thực hiện được. Sẽ có những lỗi rất hiểm ác. Nếu chất lượng có bất kỳ giá trị gì đối với bạn, thì một số kiểm thử chấp nhận thủ công là

cần thiết phải có. Đó là khi các nhà kiểm thử chuyên nghiệp không có trong Nhóm để đáp ứng hệ thống với những loại kiểm thử mà Nhóm Scrum không thể nghĩ tới, hoặc không có thời gian để tiến hành, hoặc không có thiết bị phần cứng để triển khai các kiểm thử đó. Các nhân viên kiểm thử truy xuất hệ thống theo cách thức giống như những người dùng thực sự, điều đó có nghĩa là họ phải hoàn thành thủ công công việc đó (giả sử hệ thống của bạn dành cho người dùng thực).



Nhóm kiểm thử sẽ tìm kiếm lỗi, Nhóm Scrum sẽ phải làm các bản phát hành vá lỗi, và sớm hay muộn hơn (hy vọng là sớm hơn) bạn sẽ có thể phải phát hành phiên bản vá lỗi 1.0.1 tới người dùng, thay cho phiên bản không ổn định 1.0.0.

Khi tôi nói rằng "giai đoạn kiểm thử chấp nhận", tôi đang đề cập tới toàn bộ thời gian triển khai kiểm thử, sửa lỗi, và phát hành lại cho tới khi có một phiên bản đủ tốt để phát hành sản phẩm.

Giảm thiểu giai đoạn kiểm thử chấp nhận

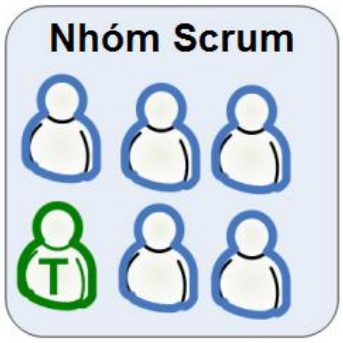
Giai đoạn kiểm thử chấp nhận gây ra các thiệt hại. Điều này tạo ra cảm giác chưa thực sự linh hoạt (un-agile). Mặc dù chúng ta không thể bỏ qua công đoạn này, chúng ta có thể (và tiến hành) cố gắng để giảm thiểu nó. Cụ thể hơn, đó là giảm thiểu lượng thời gian cần thiết dành cho triển khai kiểm thử chấp nhận. Điều này được thực hiện bởi:

- Tối đa hóa chất lượng mã nguồn do Nhóm Scrum cung cấp.
- Tối đa hóa hiệu quả của việc hướng dẫn công tác kiểm thử (tức là tìm kiếm các nhân viênkiểm thử tốt nhất, cung cấp cho họ công cụ tốt nhất, đảm bảo rằng việc báo cáo có thể được tự động để tránh lãng phí thời gian).

Vậy thì chúng tôi đã làm như thế nào để tối đa chất lượng mã nguồn do Nhóm Scrum cung cấp? Vâng, có rất nhiều cách. Sau đây là hai cách mà chúng tôi thấy rất hiệu quả:

- Đưa các nhà kiểm thử vào Nhóm Scrum
- Giảm bớt công việc với mỗi Sprint

Tăng cường chất lượng bằng cách đưa các nhà kiểm thử vào Nhóm Scrum



Vâng, tôi nghe thấy hai phản đối sau:

- "Nhưng một điều rõ ràng! Nhóm Scrum được hỗ trợ để có thể liên chức năng!"
- "Nhóm Scrum được hỗ trợ để giảm thiểu các vai trò! Chúng ta không thể có một người chỉ có vai trò là kiểm thử viên!"

Hãy để tôi làm rõ điều này: Những gì mà tôi nói về "kiểm thử viên" trong trường hợp này là "một người có kỹ năng căn bản về kiểm thử", chứ không phải là "một người chỉ có vai trò kiểm thử".

Các nhà phát triển thường khá tồi trong việc kiểm thử. *Đặc biệt* là khi các nhà phát triển kiểm thử chính mã nguồn của mình.

Nhân viên kiểm thử là "người phát tín hiệu dừng"

Hơn nữa, để "được" là một thành viên của nhóm, kiểm thử viên có những nhiệm vụ quan trọng. Anh ta là người phát tín hiệu dừng hệ thống. Không có gì được cho là "hoàn thành" trong Sprint cho tới khi anh ta bảo rằng nó đã hoàn thành. Tôi đã phát hiện ra rằng các nhà phát triển thường nói một số thứ đã hoàn thành trong khi thực tế không phải vậy. Thậm chí, ngay cả khi bạn có một định nghĩa "hoàn thành" rất rõ ràng (điều này bạn nên xem ở trang 32 "Định nghĩa 'Hoàn thành'"), các nhà phát triển thường hay quên nó. Các lập trình viên của chúng tôi là những người thiếu kiên nhẫn và muốn chuyển ngay tới hạng mục kế tiếp.

Vậy thì, Mr. T (nhân viên kiểm thử của chúng tôi) làm thế nào để biết được những thứ đó đã hoàn thành? Được rồi, trước hết, anh ấy nên (bất ngờ) *kiểm thử* nó. Trong nhiều trường hợp nó cho thấy rằng những công việc mà một nhà phát triển coi là "hoàn thành" có thể thậm chí còn *không thể kiểm thử* được! Bởi vì, nó đã không được đưa vào, hoặc không được triển khai trên dịch vụ kiểm thử, hoặc có thể chưa được khởi chạy, hoặc bất kỳ điều gì khác. Khi T kiểm thử tính năng đó, anh ấy rà soát thông qua danh sách kiểm tra "hoàn thành" (nếu bạn có) với nhà phát triển. Ví dụ, nếu định nghĩa các nhiệm vụ "hoàn thành" cần có ghi chú phát hành, khi đó T sẽ kiểm tra những ghi

chú đó xem có không. Nếu có một số đặc tả quy củ hơn cho tính năng này (hiếm gặp với chúng tôi) thì T kiểm tra xem nó đã tốt chưa, v.v.

Một hệ quả tốt phát sinh từ việc này đó là Nhóm hiện nay đã có một người rất thành thạo và phù hợp trong việc tổ chức các buổi demo Sprint.

Kiểm thử viên sẽ làm gì khi mà không có gì để kiểm thử?

Câu hỏi này được đặt ra. Mr. T: "Này, Scrum Master, chẳng có gì để kiểm thử vào lúc này cả, vậy tôi nên làm gì đây?". Có thể phải mất cả tuần để nhóm hoàn thành story đầu tiên, vậy kiểm thử viên nên làm gì trong suốt quãng thời gian đó?

Được thôi, trước hết, anh ta nên *chuẩn bị cho các kiểm thử*. Đó là, viết các đặc tả kiểm thử, chuẩn bị môi trường kiểm thử, v.v. Do đó, khi một nhà phát triển có những thứ sẵn sàng để kiểm thử, sẽ không phải chờ đợi nữa, T chỉ việc tiến hành ngay việc kiểm thử.

Nếu nhóm đang triển khai TDD thì mọi người dành thời gian để viết mã kiểm thử trước đó 1 ngày. Kiểm thử viên nên lập trình cặp với nhà phát triển để viết các mã kiểm thử đó. Nếu kiểm thử viên không thể lập trình anh ta vẫn nên cặp với nhà phát triển, anh ta chỉ làm nhiệm vụ điều hướng và hãy để các nhà phát triển viết mã nguồn. Một kiểm thử viên tốt thường đưa ra những dạng kiểm thử tốt hơn so với nhà phát triển giỏi, vì vậy họ sẽ bổ trợ cho nhau. Nếu nhóm không triển khai TDD, hoặc nếu không có đủ test-case để lấp đầy thời gian kiểm thử, đơn giản, anh ta nên tiến hành bất cứ điều gì có thể để trợ giúp nhóm đạt được Mục tiêu Sprint. Cũng giống như bất kỳ thành viên nào trong nhóm. Nếu kiểm thử viên có thể lập trình thì điều này thật tuyệt vời. Nếu không, nhóm của bạn phải xác định tất cả các công việc không liên quan tới lập trình (gọi là các việc phi lập trình) và cần thiết phải hoàn thành trong Sprint.

Khi tiến hành tách các story thành các đầu việc trong buổi họp lập kế hoạch Sprint, nhóm có xu hướng tập trung vào *các công việc liên quan tới lập trình.* Tuy nhiên, thường thì có rất nhiều các công việc *phi lập trình* cần phải hoàn thành trong Sprint. Nếu bạn dành thời gian để *xác định những công việc phi lập trình* trong buổi lập kế hoạch Sprint, sẽ tạo cơ hội để T có thể đóng góp khá nhiều, thậm chí kể cả anh ấy không thể lập trình và không có công việc kiểm thử để làm ngay.

Ví dụ về những công việc phi lập trình thường cần phải hoàn thành trong một Sprint:

- Thiết lập môi trường kiểm thử.
- Làm rõ các yêu cầu.
- Thảo luận chi tiết về các hoạt động triển khai.

- Viết các tài liệu triển khai (ghi chú phát hành, RFC, hoặc bất cứ điều gì mà tổ chức của bạn làm).
- Liên hệ với các nguồn lực bên ngoài (ví dụ: các nhà thiết kế giao diện người dùng (GUI)).
- Cải tiến các kịch bản build.
- Phân tách các story chi tiết hơn thành những tác vụ.
- Ghi nhận những câu hỏi quan trọng từ nhà phát triển và tìm kiếm câu trả lời cho họ.

Xét theo chiều ngược lại, chúng tôi phải làm gì nếu T trở thành "nút cổ chai"? Chẳng hạn chúng tôi đang ở ngày làm việc cuối cùng của Sprint và đột nhiên có nhiều việc hoàn thành và T không có cơ hội để kiểm thử tất cả. Chúng tôi phải làm gì? Được thôi, chúng tôi có thể đưa mọi thành viên trong nhóm thành phụ tá của T. Anh ấy quyết định những thứ anh ta cần tự làm, và phó thác các kiểm thử còn lại cho các thành viên khác trong nhóm. Đó là những gì mà chúng ta nói về các nhóm liên chức năng!

Vậy là, T có một vai trò đặc biệt trong nhóm, nhưng anh ấy vẫn được phép làm các công việc khác, và các thành viên khác trong nhóm vẫn được phép làm các công việc của anh ấy.

Tăng chất lượng bằng cách giảm thiểu công việc trong Sprint

Điều này khiến ta quay trở lại cuộc Họp Kế hoạch Sprint. Đơn giản chỉ cần đặt ra, không tham làm quá nhiều story trong một Sprint! Nếu bạn có các vấn đề về chất lượng, hoặc chu trình kiểm thử chấp nhận kéo dài, hãy giảm bớt công việc trong Sprint! Điều này sẽ tự động dẫn tới chất lượng cao hơn, chu trình kiểm thử chấp nhận ngắn hơn, ít lỗi phía người dùng, và năng suất cao hơn trong thời gian dài khi mà nhóm có thể tập trung toàn thời gian vào những thứ mới hơn là sửa chữa những thứ đã làm để tránh phá hỏng thứ khác.

Điều này thường luôn có chi phí thấp hơn để xây dựng ít hơn, nhưng đạt được sự ổn định, thay vì xây dựng nhiều hơn và phát hoảng với công việc sửa chữa.

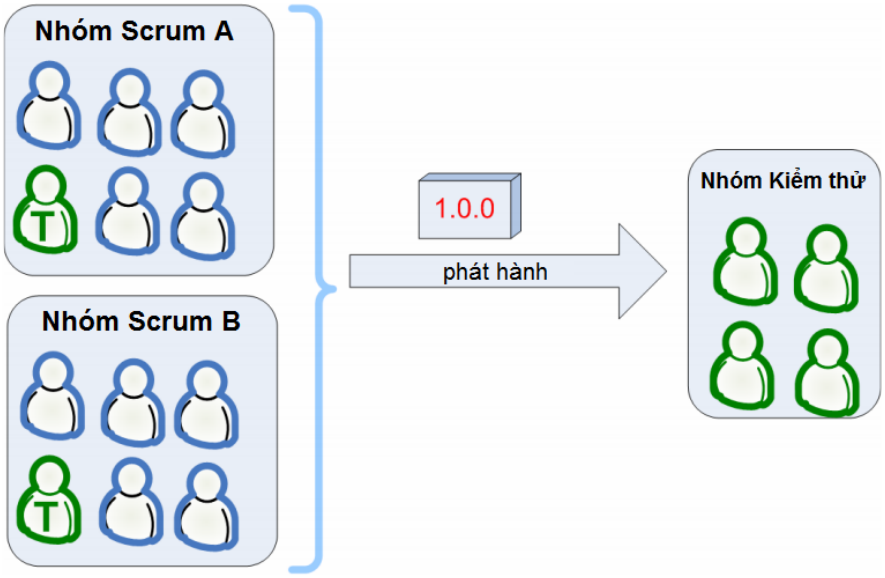
Kiểm thử chấp nhận có nên là một phần của Sprint không?

Chúng tôi do dự rất nhiều ở đây. Một số nhóm của chúng tôi bổ sung thêm kiểm thử chấp nhận vào Sprint. Tuy nhiên, hầu hết các nhóm của chúng tôi không làm như vậy bởi hai lý do sau:

- Sprint có một khung thời gian. Kiểm thử chấp nhận (sử dụng định nghĩa của tôi nó bao gồm gỡ lỗi và phát hành lại) là rất khó để xác định khung thời gian (time-box). Chuyện gì nếu bạn đã hết thời gian mà vẫn còn những lỗi nghiêm trọng? Bạn sẽ phát hành sản phẩm mà vẫn còn lỗi nghiêm trọng đó? Bạn sẽ đợi đến Sprint tiếp theo? Trong hầu hết các trường

hợp cả hai giải pháp trên đều không thể chấp nhận được. Vì vậy chúng tôi để kiểm thử chấp nhận thủ công bên ngoài Sprint.

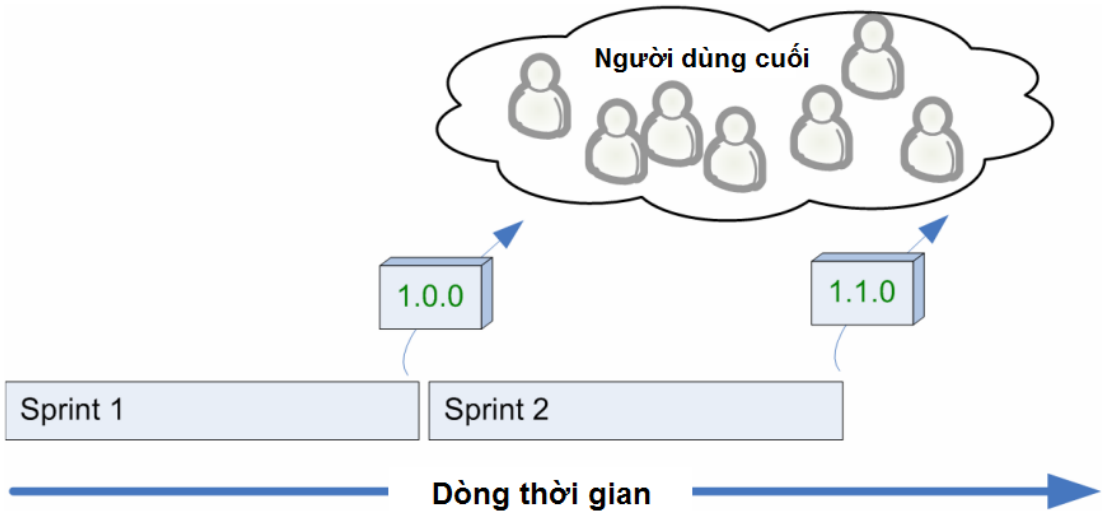
- Nếu bạn có nhiều Nhóm Scrum cùng triển khai một sản phẩm, kiểm thử chấp nhận thủ công phải được hoàn thành dựa trên sự kết hợp kết quả của tất cả các nhóm. Nếu các nhóm đã tiến hành kiểm thử thủ công trong Sprint, bạn có thể vẫn cần một nhóm kiểm thử bản phát hành cuối cùng, cái sẽ được build tích hợp với công việc của nhóm khác.



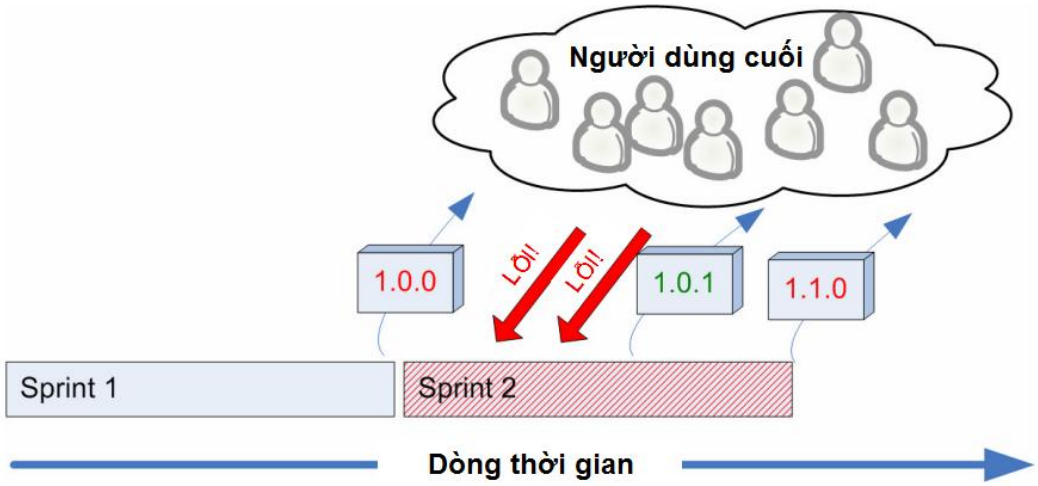
Điều này không có nghĩa đây là giải pháp hoàn hảo nhưng nó đủ tốt đối với chúng tôi trong hầu hết các trường hợp.

Chu trình Sprint với chu trình kiểm thử chấp nhận

Trong thế giới Scrum hoàn hảo, bạn không cần đến kiểm thử chấp nhận khi Nhóm Scrum phát hành một phiên bản sẵn sàng của sản phẩm sau mỗi Sprint.



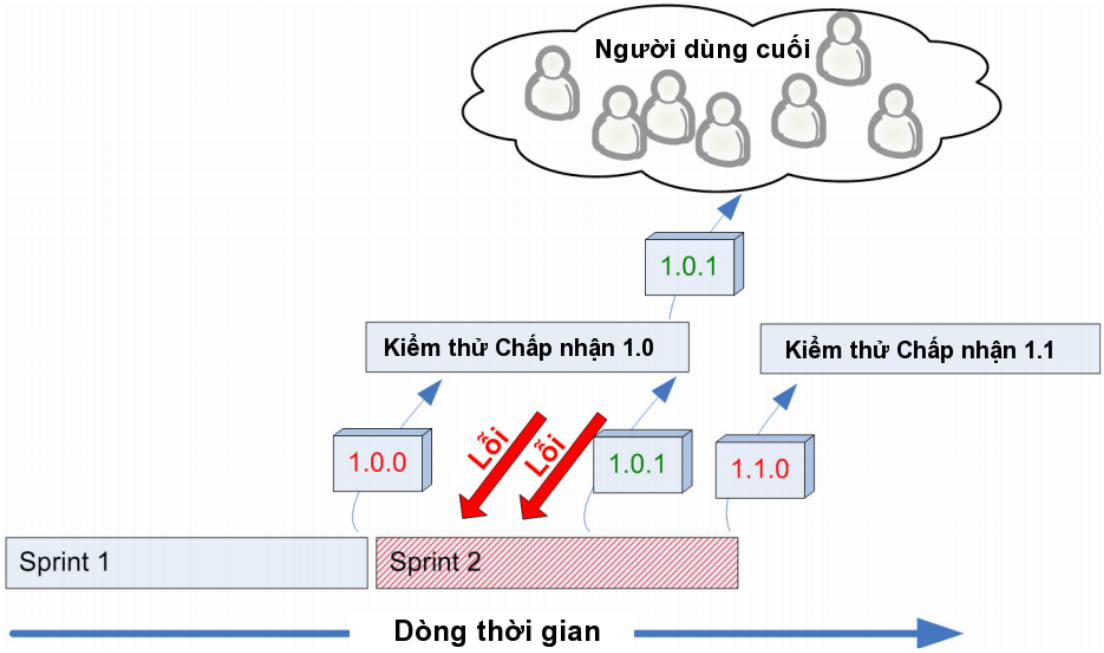
Tốt thôi, đây mới là hình ảnh thực tế hơn:



Sau Sprint 1, phiên bản có lỗi 1.0.0 được phát hành. Suốt Sprint 2, các báo cáo về lỗi bắt đầu đổ dồn về và nhóm phải dành hầu hết thời gian để sửa lỗi và bị buộc phải phát hành một bản vá lỗi 1.0.1 giữa Sprint. Sau đó cuối Sprint 2 họ phát hành phiên bản mới 1.1.0, tất nhiên là thậm chí ít lỗi hơn khi đó họ tốn ít thời gian hơn để chỉnh sửa chúng trong thời gian này, do các rối loạn từ bản phát hành trước đó. V.v. và v.v.

Các đường chéo đỏ trong Sprint 2 thể hiện các xáo trộn.

Không quá đẹp mắt, phải không? Tốt thôi, điều đáng buồn là vấn đề đó vẫn còn ngay cả khi bạn có một nhóm kiểm thử chấp nhận. Chỉ có điểm khác biệt đó là hầu hết các báo cáo lỗi sẽ đến từ đội kiểm thử thay vì từ sự bực bội của người dùng. Đó là một khác biệt rất lớn từ góc độ kinh doanh, nhưng với các nhà phát triển thì chẳng có gì rốt cuộc vẫn vậy. Ngoại trừ trường hợp các kiểm thử viên kém tích cực hơn so với người dùng cuối. Thường là vậy.



Chúng tôi không tìm ra bất cứ giải pháp đơn giản nào đối với vấn đề này. Chúng tôi đã thử nghiệm với nhiều mô hình khác nhau.

Trước tiên, một lần nữa, cần tối đa hóa chất lượng của mã nguồn do Nhóm Scrum phát hành. Chi phí đối với việc tìm kiếm và sửa lỗi sớm, trong Sprint, sẽ thấp hơn nhiều so với việc tìm kiếm và sửa lỗi sau đó.

Nhưng trong thực tế vẫn còn những báo cáo về lỗi đến sau khi Sprint hoàn tất, ngay cả khi chúng ta có thể giảm thiểu số lượng. Chúng tôi giải quyết vấn đề này ra sao?

Hướng tiếp cận 1: “Không tiến hành xây dựng các thứ mới cho tới khi những thứ đã làm vẫn còn chưa hoàn thành”

Bạn nghe câu này có thú vị? Bạn cũng đã có cảm thấy lờ mờ sự kích động hay không?

Chúng tôi đã tiến với việc áp dụng hướng tiếp cận này một vài lần, và đã rút ra các mô hình phù hợp với cách thức mà chúng tôi làm điều đó. Tuy nhiên, chúng tôi luôn thay đổi tư tưởng của mình khi chúng tôi nhận ra những nhược điểm. Chúng tôi phải bổ sung một giai đoạn phát hành không đóng khung thời gian giữa các Sprint, trong khoảng thời gian đó chúng tôi chỉ kiểm thử và sửa lỗi cho tới khi chúng tôi có thể tạo ra một bản phát hành sản phẩm.

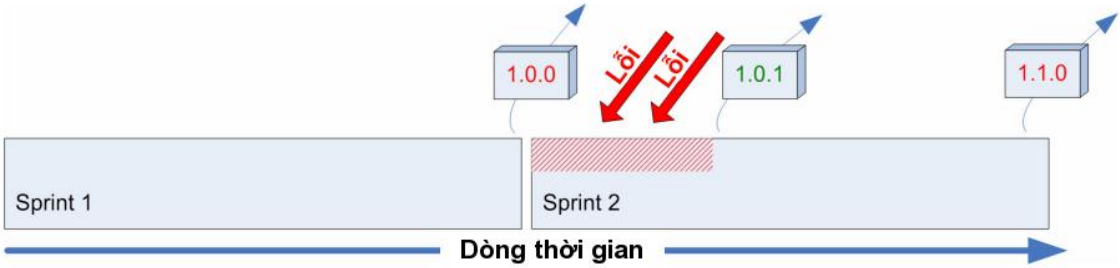
Chúng tôi không thích chú thích về việc có các giai đoạn phát hành không đóng khung thời gian giữa các Sprint, chủ yếu là bởi vì nó sẽ phá vỡ quy tắc giữ nhịp đập Sprint. Chúng tôi có thể không nói rằng "sau 3 tuần chúng tôi lại bắt đầu một Sprint mới". Bên cạnh đó, điều này không hoàn toàn giải quyết được vấn đề. Thậm chí nếu chúng tôi có giai đoạn phát hành, sẽ có các báo cáo lỗi khẩn cấp liên tục gửi tới, và chúng tôi phải chuẩn bị để đối phó với chúng.

Hướng tiếp cận 2: "OK tiến hành xây dựng thứ mới, nhưng ưu tiên hoàn thành những thứ đã triển khai trước"

Đây là hướng tiếp cận ưa thích của chúng tôi. Ít nhất là trong lúc này.

Về cơ bản, khi chúng tôi hoàn thành một Sprint chúng tôi chuyển ngay sang Sprint tiếp theo. Nhưng chúng tôi vẫn phòng trước phải sử dụng một số thời gian trong Sprint tiếp theo để sửa các lỗi của Sprint trước. Nếu Sprint tiếp theo đó bị thiệt hại nghiêm trọng do chúng tôi phải sử dụng quá nhiều thời gian cho việc sửa các lỗi của Sprint trước, chúng tôi đánh giá nguyên nhân xảy ra điều này và cách thức để chúng tôi có thể cải tiến chất lượng. Chúng tôi đảm bảo Sprint đủ dài để có đủ thời gian cần thiết cho việc sửa các lỗi từ Sprint trước.

Dần dần, trải qua nhiều tháng, lượng thời gian dành cho sửa lỗi từ Sprint trước sẽ giảm đi. Thêm vào đó, chúng tôi có thể có một vài người tham gia cùng khi xảy ra các lỗi, vì vậy *toàn bộ* nhóm không cần phải băn khoăn mỗi khi như vậy. Bây giờ, chúng tôi đã đạt được mức độ chấp nhận cao hơn.



Trong suốt buổi họp lập kế hoạch Sprint, chúng tôi thiết lập hệ số tập trung đủ thấp để dành thời gian dự phòng cho việc sửa chữa lỗi từ Sprint trước. Cùng với thời gian, nhóm sẽ có những tiến bộ đáng kể trong việc ước tính thời gian cho việc này. Đo đạc về tốc độ trợ giúp rất nhiều (xem trang 30, *"Nhóm quyết định đưa story nào vào Sprint như thế nào?"*).

Hướng tiếp cận tồi – "tập trung vào việc xây dựng tính năng mới"

Điều này có nghĩa là "tập trung xây dựng những thứ mới *hơn là hoàn thành những thứ đã làm trước đó*". Ai sẽ muốn làm điều này? Chúng tôi đã gặp phải những sai lầm này khá thường xuyên thuở ban đầu, và tôi chắc chắn rằng nhiều công ty khác cũng đã vướng phải chuyện như vậy. Đó là căn bệnh liên quan tới áp lực. Nhiều nhà quản lý không thực sự hiểu điều đó, khi tất cả mã nguồn đã hoàn tất, bạn thường vẫn còn ở khá xa mới tới được thời điểm phát hành sản phẩm. Ít nhất là với các hệ thống phức tạp. Vì vậy, nhà quản lý (hoặc Product Owner) yêu cầu nhóm tiếp tục làm thêm những thứ mới trong khi những phần còn lại của mã nguồn gần như đã sẵn sàng phát hành trước đó ngày càng trở lên nặng nề hơn, làm mọi thứ chậm lại.

Đừng chạy nhanh hơn mắt xích chậm nhất trong dây chuyền của bạn

Giả sử rằng kiểm thử chấp nhận là mắt xích chậm nhất của bạn. Bạn có quá ít kiểm thử viên, hoặc giai đoạn kiểm thử chấp nhận kéo dài bởi vì chất lượng nghèo nàn của mã nguồn.

Giả sử rằng nhóm kiểm thử chấp nhận của bạn có thể kiểm thử nhiều nhất 3 tính năng/tuần (không, chúng tôi không sử dụng "tính năng/tuần" khi đo đạc; tôi chỉ sử dụng nó cho ví dụ này). Và giả sử rằng các nhà phát triển của bạn có thể thực hiện được 6 tính năng/tuần.

Điều này sẽ hấp dẫn các nhà quản lý hoặc Product Owner (hoặc có thể thậm chí cả nhóm) để đặt lịch phát triển 6 tính năng mới/tuần.

Xin đừng! Thực tế sẽ "túm" được bạn bằng cách này hay cách khác, và nó sẽ làm tổn thương bạn.

Thay vào đó, hãy đặt lịch làm 3 tính năng/tuần và sử dụng thời gian còn lại để giảm thiểu các nút cổ chai kiểm thử. Ví dụ:

- Có một vài nhà phát triển làm việc như là các kiểm thử viên (ồ, họ sẽ yêu quý bạn vì điều này…).
- Triển khai các công cụ và kịch bản giúp tiến hành kiểm thử dễ dàng hơn.
- Thêm nhiều hơn các mã nguồn kiểm thử tự động.
- Tăng độ dài của Sprint và có thêm kiểm thử chấp nhận trong Sprint.
- Định nghĩa một số Sprint như là "Sprint kiểm thử" ở đó toàn bộ nhóm làm việc như một nhóm kiểm thử chấp nhận.
- Tuyển thêm cáckiểm thử viên(thậm chí nếu điều này có nghĩa là giảm bớt nhà phát triển)

Chúng tôi đã thử tất cả các giải pháp trên (trừ giải pháp cuối cùng). Giải pháp dài hạn tốt nhất tất nhiên là 2 và 3, tức là các công cụ và kịch bản tốt hơn và tự động kiểm thử.

Các buổi Họp Cải tiến là một diễn đàn tốt để xác định những mắt xích chậm nhất trong dây chuyền của mình.

Quay lại với thực tế

Tôi có thể cho bạn thấy ấn tượng về việc chúng tôi có các nhà kiểm thử trong tất cả các Nhóm Scrum, việc chúng tôi có một nhóm kiểm thử chấp nhận rất lớn dành cho mỗi sản phẩm, việc chúng tôi phát hành sau mỗi Sprint, v.v. và v.v.

Vâng, nhưng chúng tôi không làm như vậy.

Đôi khi chúng tôi cũng đã có ý thức để làm những thứ này, và chúng tôi đã nhìn thấy những ảnh hưởng tích cực từ đó.

Nhưng chúng tôi vẫn còn xa mới có được quy trình đảm bảo chất lượng chấp nhận được, và chúng tôi vẫn phải học hỏi rất nhiều.

15

Chúng tôi quản lí nhiều nhóm Scrum như thế nào

Có nhiều thứ khó khăn hơn khi bạn có nhiều Nhóm Scrum cùng phát triển một sản phẩm. Đó là vấn đề phổ quát chứ không thực sự không phải là chuyện riêng của Scrum. Nhiều nhà phát triển = nhiều rắc rối.

Chúng tôi đã thử nghiệm (như thường lệ) với vấn đề này. Có lúc nhóm chúng tôi lên đến với xấp xỉ 40 người làm việc với cùng một sản phẩm.

Các câu hỏi chính đặt ra:

- Sẽ tạo ra bao nhiêu Nhóm
- Bố trí mọi người vào nhóm như thế nào

Bao nhiêu Nhóm?

Nếu làm việc với nhiều Nhóm Scrum khó khăn như thế, thì tại sao chúng ta lại bận tâm? Tại sao không để mọi người chỉ trong cùng một Nhóm thôi?

Nhóm Scrum lớn nhất chúng tôi có nằm trong khoảng 11 người. Nhóm vẫn hoạt động được, nhưng không tốt lắm. Buổi họp Scrum Hằng ngày sẽ kéo dài quá 15 phút. Các thành viên Nhóm không biết những thành viên khác trong Nhóm đang làm gì, do đó sẽ dẫn đến nhầm lẫn. Rất khó để Scrum Master giữ cho tất cả các thành viên bám sát mục tiêu, và khó để bố trí thời gian giải quyết tất cả những trở ngại mà mọi người gặp phải.

Một giải pháp thay thế là chia thành hai Nhóm. Nhưng cái đó có tốt hơn không? Không nhất thiết phải như vậy.

Nếu Nhóm đã có kinh nghiệm và quen với Scrum, có một cách lô-gíc chia lộ trình (roadmap) thành hai phần riêng biệt, cả hai phần này không liên đới tới cùng mã nguồn, ý tôi muốn nói đấy là một ý tưởng tốt để chia nhóm. Nếu không được, tôi sẽ xét tới việc gộp thành một nhóm, mặc dù có nhiều nhược điểm khi nhóm lớn.

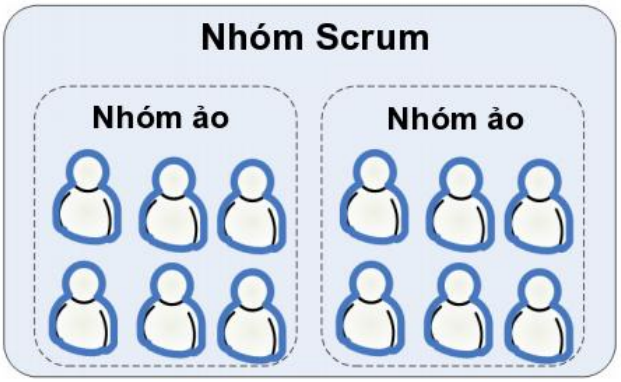
Kinh nghiệm của tôi là nên có một vài nhóm lớn thì sẽ tốt hơn là có nhiều nhóm nhỏ mà phải giữ công việc liên đới nhau. Chỉ chia thành các nhóm nhỏ khi chúng không liên đới công việc với nhau!

Nhóm ảo

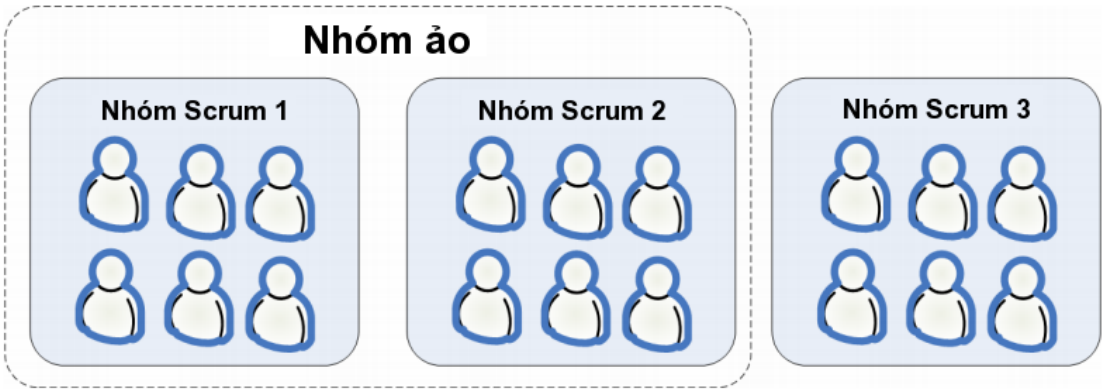
Làm thế nào bạn biết mình đã quyết định đúng hay sai khi xem xét "nhóm lớn" với "nhiều nhóm"?

Nếu bạn giữ cho đôi mắt và đôi tai mình luôn mở bạn có thể nhận ra hình thức "nhóm ảo".

Ví dụ 1: Bạn lựa chọn có một nhóm lớn. Nhưng khi bạn bắt đầu quan sát mọi người nói chuyện với nhau trong Sprint, bạn nhận thấy Nhóm có phân chia thành hai nhóm nhỏ hiệu quả.



Ví dụ 2: Bạn lựa chọn có ba nhóm nhỏ hơn. Nhưng khi bạn quan sát mọi người nói chuyện với nhau trong suốt Sprint, bạn nhận thấy nhóm 1 và nhóm 2 nói chuyện với nhau trong suốt thời gian, trong khi nhóm 3 làm việc tách biệt.



Điều đó có nghĩa là gì? Điều đó có nghĩa chiến lược chia nhóm của các bạn đã sai chăng? Đúng, nếu các nhóm ảo dường **như** cứ như vậy suốt. Còn sai, nếu nhóm ảo chỉ tạm thời như vậy.

Nhìn lại ví dụ 1. Nếu hai nhóm ảo có xu hướng thay đổi một lần trong một thời gian (ví dụ, mọi người chuyển qua lại giữa các nhóm ảo) thì có thể bạn đã quyết định đúng khi để họ ở một Nhóm

Scrum duy nhất. Nếu hai nhóm ảo vẫn cứ như vậy trong suốt toàn bộ thời gian của Sprint, có thể bạn sẽ phải nghĩ đến việc chia họ thành hai nhóm Scrum thực sự ở Sprint tiếp theo.

Bây giờ nhìn lại ví dụ 2. Nếu nhóm 1 và nhóm 2 (không có nhóm 3) trao đổi với nhau trong suốt cả Sprint, bạn có thể muốn kết hợp nhóm 1 và 2 thành một nhóm trong Sprint tiếp theo. Nếu nhóm 1 và 2 trao đổi với nhau nhiều trong suốt nửa đầu Sprint và sau đó nhóm 1 và nhóm 3 trao đổi với nhau trong suốt nửa cuối của Sprint, thì bạn nên xem xét việc kết hợp cả ba nhóm đó thành một, hoặc vẫn giữ họ như vậy ở ba nhóm riêng biệt. Mang câu hỏi đó ra bàn trong buổi Họp Cải tiến Sprint và để cho các nhóm tự quyết định.

Việc chia nhóm là một trong những phần việc thực sự khó của Scrum. Đừng nghĩ quá nhiều hoặc tối ưu hóa những thứ quá khó. Về mặt kinh nghiệm, hãy chịu khó quan sát các nhóm ảo, chắc chắn bạn dành thời gian để thảo luận về các hình thức này trong các buổi họp cải tiến. Không sớm thì muộn bạn sẽ tìm ra giải pháp đúng cho tình huống cụ thể của bạn. Điều quan trọng là các nhóm phải cảm thấy thoải mái và không dẫm chân lên công việc của các nhóm khác quá thường xuyên.

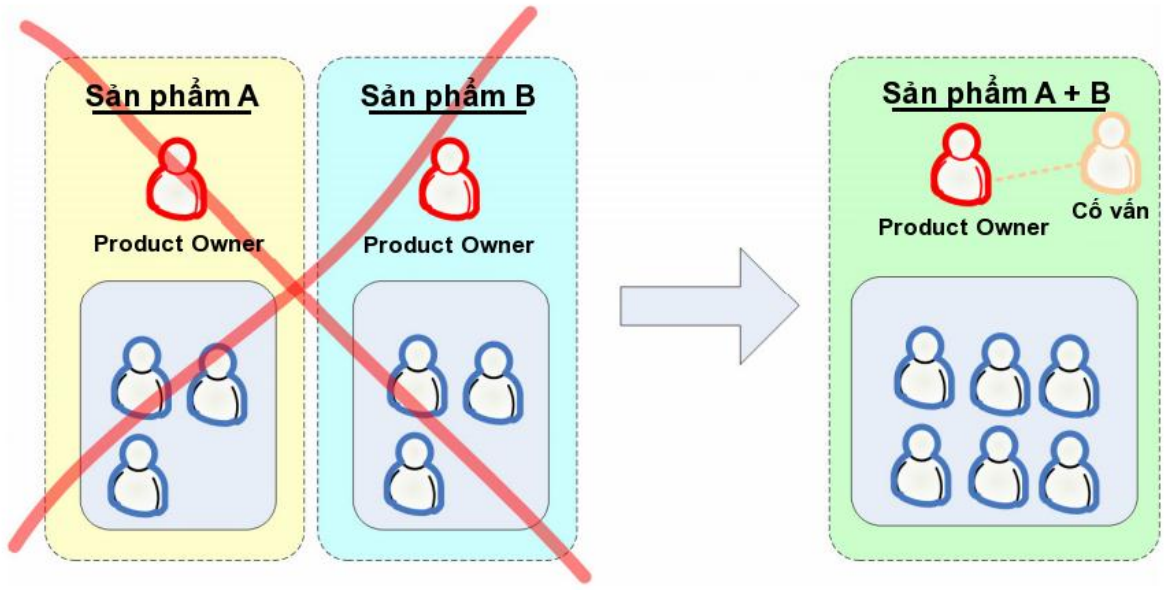
Kích cỡ nhóm tối ưu

Hầu hết các cuốn sách tôi đọc cho rằng kích cỡ "tối ưu" của nhóm nằm đâu đó trong khoảng từ 5 đến 9 người.

Từ những gì tôi thấy tới bây giờ tôi chỉ có thể nói là tôi đồng ý. Mặc dù tôi muốn nói là từ 3 đến 8 người. Thực tế, tôi tin rằng sẽ phải đón nhận một vài thất bại thì nhóm mới rút ra được kết luận về kích cỡ đó.

Giả sử bạn có một nhóm Scrum 10 người. Hãy xem xét việc loại ra khỏi nhóm hai người yếu nhất. Đáng tiếc, tôi đã nói vậy đúng không?

Giả thiết bạn có hai sản phẩm khác nhau, với một nhóm 3 người phát triển một sản phẩm, và cả hai nhóm tiến triển quá chậm. Một ý tưởng tốt là có thể nếu gộp họ lại thành một nhóm 6 người chịu trách nhiệm phát triển cả hai sản phẩm. Trong tình huống đó hãy để một trong hai Product Owner ra đi (hoặc để anh ý giữ vai trò cố vấn hay đại loại một cái gì đó).

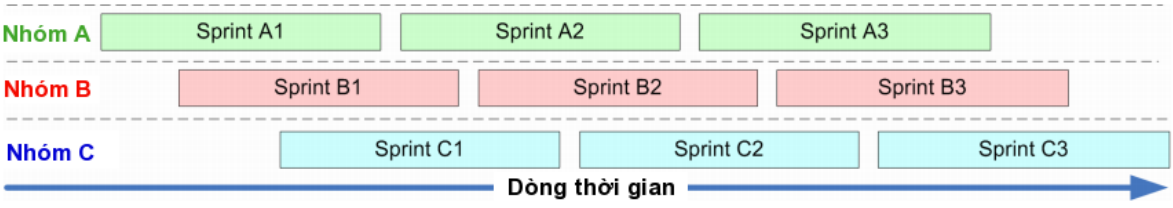


Giả thiết bạn có một nhóm Scrum 12 người, bởi vì cơ sở mã nguồn ở trong trạng thái lộn xộn, không có cách nào để 2 nhóm riêng biệt làm việc trên sản phẩm đó một cách độc lập. Hãy dành những nỗ lực nghiêm túc vào việc sửa cơ sở mã nguồn (thay vì xây dựng thêm các tính năng mới) cho tới thời điểm bạn có thể chia tách nhóm. Việc đầu tư công sức này sẽ có thể sẽ phải trả giá kha khá đấy.

Đồng bộ hay không đồng bộ các Sprint?

Giả sử bạn có ba nhóm Scrum cùng phát triển một sản phẩm. Các Sprint của họ có nên được đồng bộ hay không, ví dụ bắt đầu và kết thúc cùng nhau? Hay nên để chúng gối nhau?

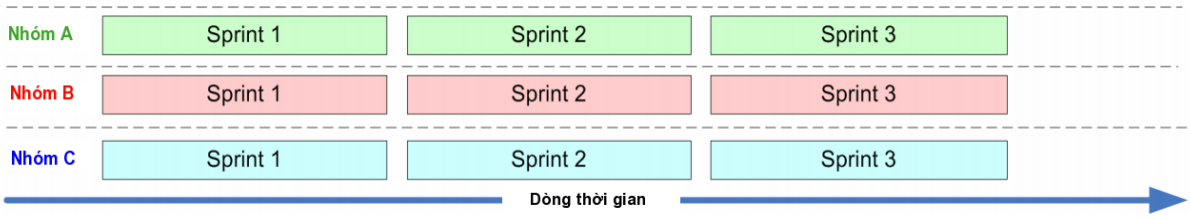
Hướng tiếp cận đầu tiên của chúng tôi là để các Sprint gối nhau (tương ứng thời gian).



Điều này nghe có vẻ tốt đẹp. Tại bất cứ thời điểm nào sẽ có một Sprint đang diễn ra và đi đến hồi kết, và có một Sprint mới bắt đầu. Khối lượng công việc của Product Owner sẽ được trải đều theo thời gian. Liên tục có các bản phát hành của hệ thống. Tuần nào cũng demo. Quá ngon lành!.

Vâng, tôi biết, nhưng thực sự điều đó đã rất thuyết phục vào thời điểm đó!

Chúng tôi đã chỉ bắt đầu làm theo cách này cho đến một ngày tôi có cơ hội để nói chuyện với Ken Schwaber (trong một lần học lấychứng chỉ Scrum). Ông ấy đã chỉ ra rằng đây là một ý tưởng *tồi*, tốt hơn nhiều là hãy đồng bộ các Sprint. Tôi không nhớ chính xác lý do ông ấy đưa ra là gì, nhưng sau một vài thảo luận tôi đã bị thuyết phục.

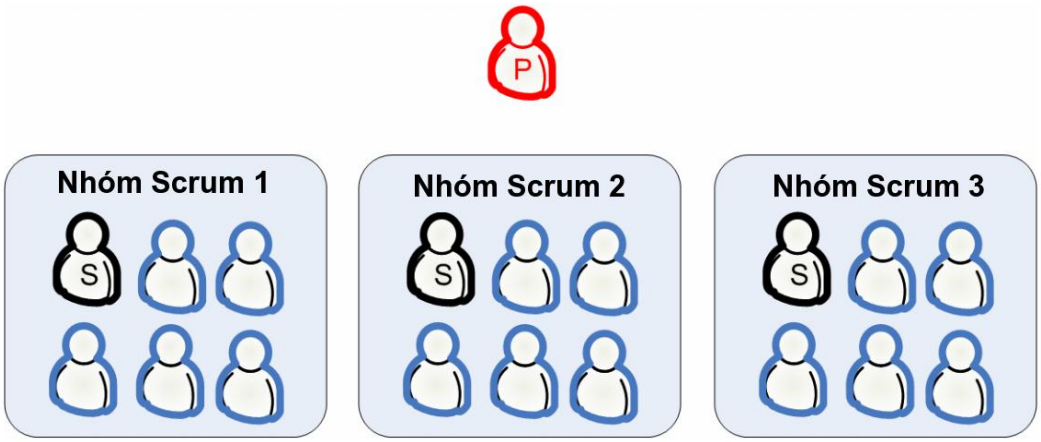


Đây là giải pháp chúng tôi đã từng sử dụng, và chưa từng hối tiếc về nó. Tôi sẽ không bao giờ biết nếu chiến lược để Sprint gối nhau có thất bại hay không, nhưng tôi nghĩ vậy. Ưu điểm của các Sprint được đồng bộ hóa là :

- Có thời điểm tự nhiên để sắp xếp lại các nhóm – giữa các Sprint! Với các Sprint gối đầu, không có cách nào để sắp xếp lại các nhóm mà không ảnh hưởng đến ít nhất một nhóm ở giữa Sprint.
- Tất cả các nhóm có thể làm việc hướng tới cùng mục tiêu trong Sprint và thực hiện các buổi họp lập kế hoạch cho Sprint cùng nhau, điều đó dẫn tới việc cộng tác tốt hơn giữa các nhóm.
- Ít công sức quản trị hơn, ví dụ: ít các cuộc họp lập kế hoạch Sprint hơn, ít buổi demo Sprint và phát hành hơn.

Tại sao chúng tôi đưa thêm vai trò “lãnh đạo nhóm”

Giả sử chúng tôi có một sản phẩm được phát triển bởi ba nhóm.



Biểu tượng đỏ với ký hiệu P ở hình trên là Product Owner. Các biểu tượng đen với ký hiệu S là Scrum Master. Các hình còn lại tương ứng là các thành viên Nhóm đáng kính.

Với phân bổ như vậy, ai là người quyết định ai sẽ nằm ở nhóm nào? Product Owner? Ba Scrum Master cùng nhau quyết? Hay mọi người tự chọn nhóm cho mình? Nhưng vậy chuyện gì sẽ xảy ra nếu mọi người muốn cùng ở nhóm 1 (bởi vì Scrum Master nhóm 1 có vẻ hay hơn)?

Nếu sau này có những lý do thấy rằng thực sự không thể có nhiều hơn hai nhóm cùng làm việc trên cùng cơ sở mã nguồn này, khi đó chúng tôi cần chuyển thành hai nhóm 9 người thay vì ba nhóm 6 người. Điều đó có nghĩa còn 2 Scrum Master. Vì vậy, một trong số 3 Scrum Master hiện tại sẽ từ bỏ vai trò của mình?

Trong nhiều công ty đây sẽ là những vấn đề khá nhạy cảm.

Hãy để cho Product Owner quyết định việc phân bổ và thuyên chuyển mọi người. Nhưng đó có phải thực sự là việc làm đúng trách nhiệm của Product Owner không? Product Owner chuyên về một lĩnh vực, người sẽ nói với nhóm định hướng mà họ nên theo. Anh ta sẽ không thực sự phải dính dáng gì đến những chi tiết ngóc ngách, đặc biệt vì anh ta chỉ là "gà" (nếu bạn đã từng nghe hình ảnh ẩn dụ về gà và lợn, nếu không hãy tìm trên Google cụm từ "chickens and pigs").

Chúng tôi đã giải quyết vấn đề này bằng cách đưa thêm một vai trò là "lãnh đạo nhóm". Điều này tương ứng với cái mà bạn có thể gọi là "Scrum Master của Scrum" hay "ông chủ" hay "Scrum Master chủ chốt", v.v. Anh ta không phải lãnh đạo bất cứ nhóm nào, nhưng anh ta chịu trách nhiệm về các vấn đề liên nhóm chẳng hạn ai sẽ là Scrum Master của nhóm, cách mọi người sẽ được chia vào các nhóm, v.v.

Chúng tôi đã có thời điểm khó khăn để đi tới thống nhất về cái tên hợp lý cho vai trò này. "Lãnh đạo nhóm" là tên hợp lý nhất mà chúng tôi có thể gọi.

Giải pháp này đã phát huy hiệu quả tốt đối với chúng tôi và tôi có thể khuyến nghị bạn nên thực hiện điều này (bất kể bạn quyết định gọi vai trò này là gì).

Bố trí những thành viên vào các nhóm như thế nào

Có hai chiến lược phổ biến cho việc bố trí thành viên vào các nhóm, khi bạn có nhiều nhóm phát triển cùng một sản phẩm.

- Hãy để một người được chỉ định thực hiện việc bố trí, ví dụ "lãnh đạo nhóm" mà tôi đã đề cập ở trên, Product Owner, hay giám đốc chức năng (nếu anh ta tham gia đầy đủ để có thể đưa ra các quyết định tốt ở đây).
- Hãy để các nhóm tự chia bằng cách nào đấy.

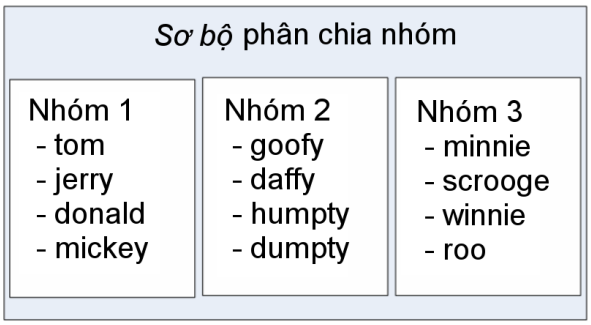
Chúng tôi đã thử nghiệm bằng cả ba cách. Ba cách? Đúng thế. Chiến lược 1, chiến lược 2 và kết hợp cả hai.

Chúng tôi đã nhận thấy việc kết hợp cả hai là tốt nhất.

Trước buổi họp lập kế hoạch Sprint, lãnh đạo nhóm triệu tập cuộc họp phân bổ thành viên nhóm cùng với Product Owner và tất cả các Scrum Master. Chúng tôi nói về cuối Sprint và quyết định nếu bất kỳ việc tái bố trí nào được đảm bảo. Có lẽ chúng tôi muốn kết hợp hai nhóm, hoặc đưa một vài người từ nhóm này sang nhóm khác. Chúng tôi quyết định dựa vào một vài tiêu chí và ghi nó ra gọi là *đề xuất phân bổ nhóm*, cái đó chúng tôi mang tới buổi họp lập kế hoạch Sprint.

Điều đầu tiên chúng tôi thực hiện trong buổi họp lập kế hoạch Sprint là đi từ các công việc có độ ưu tiên cao trong Product Backlog. Lãnh đạo nhóm sau đó sẽ nói một vài điều như:

"Chào mọi người, chúng tôi gợi ý việc phân bổ nhóm cho Sprint tiếp theo như sau".



“Như các bạn thấy, điều này có nghĩa là sẽ giảm từ 4 nhóm xuống còn 3 nhóm. Chúng tôi đã có danh sách các thành viên của từng nhóm. Các bạn vui lòng tập hợp lại và chọn cho mình một phần bức tường.”

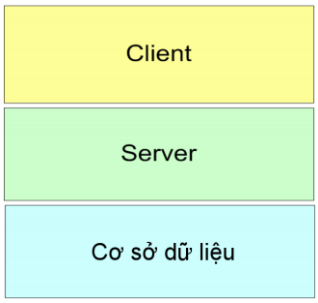
(lãnh đạo nhóm đợi trong khi mọi người đi lòng vòng trong phòng, sau một hỗi sẽ có 3 nhóm, mỗi nhóm đứng cạnh một khoảng tường trống).

“Bây giờ việc phân chia nhóm này là *sơ bộ!* Nó chỉ là điểm khởi đầu, để tiết kiệm thời gian. Trong quá trình lập kế hoạch Sprint bạn có thể thoải mái đi sang một nhóm khác, chia nhóm của bạn thành hai, kết hợp với nhóm khác, hay bất cứ điều gì. Sử dụng lý trí của các bạn dựa vào các thứ tự ưu tiên của Product Owner.”

Đây là những công việc tốt nhất mà chúng tôi đã tìm ra. Ban đầu nên điều khiển ở một cấp độ tập trung nhất định, tiếp theo về sau là tối ưu hóa phi tập trung ở một cấp độ nhất định.

Nhóm chuyên biệt – có hay không ?

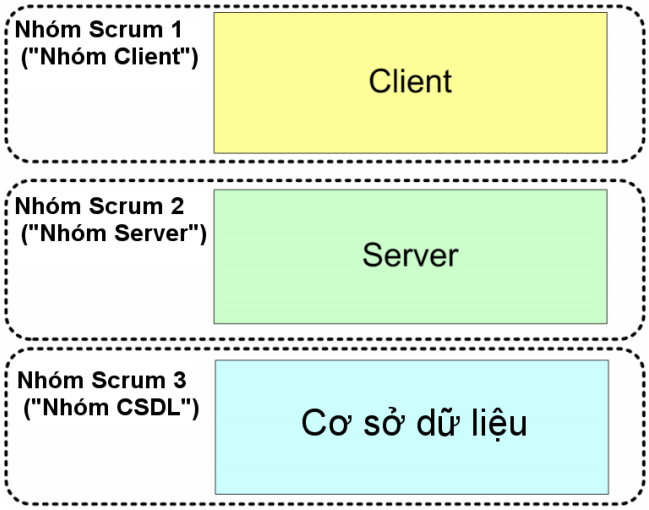
Giả sử công nghệ của bạn gồm ba thành phần chính :



Và giả thiết bạn có 15 người phát triển sản phẩm này, vì vậy bạn thực sự không muốn thực hiện với duy nhất một Nhóm Scrum. Vậy bạn sẽ tạo ra các nhóm như thế nào?

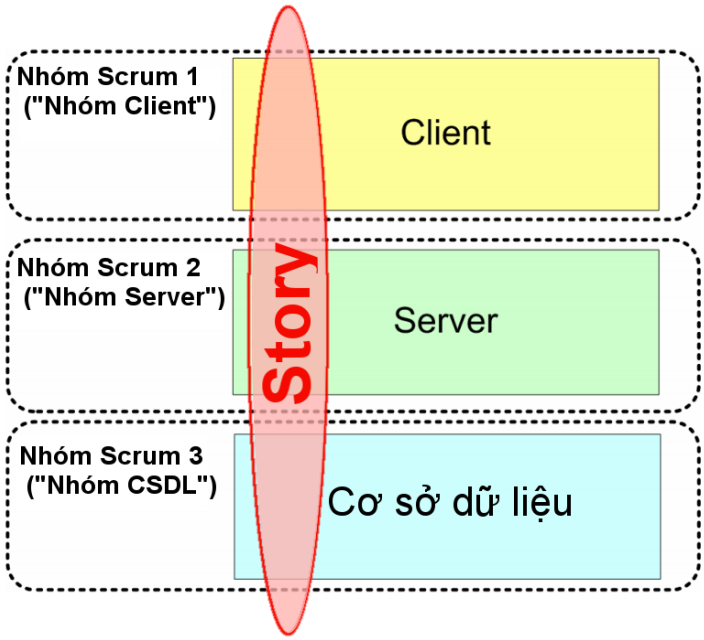
Hướng 1: các nhóm chuyên biệt về từng thành phần

Một hướng là tạo ra các nhóm chuyên biệt từng thành phần như một “nhóm Client”, một “nhóm Server”, và một “nhóm Cơ sở dữ liệu”.



Đây là cách chúng tôi đã bắt đầu. Không hiệu quả lắm, ít nhất là không tốt khi mà hầu hết các story liên quan tới nhiều thành phần.

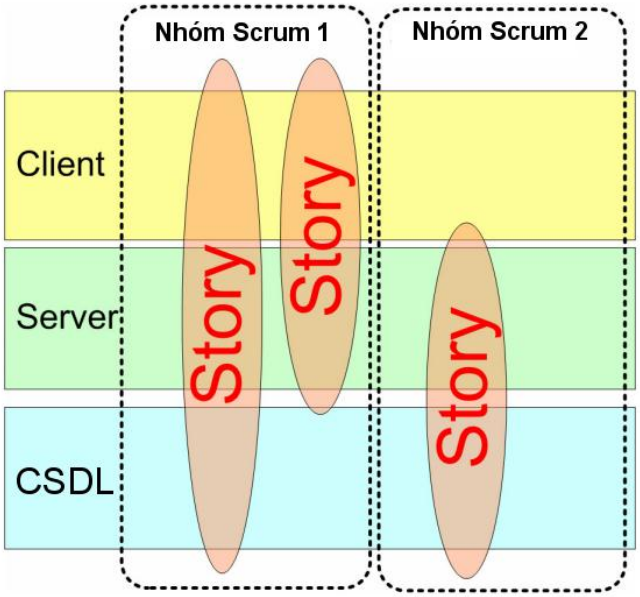
Ví dụ giả sử chúng tôi có một story có tên "bảng thông báo nơi người dùng có thể gửi thông điệp cho nhau". Chức năng bảng thông báo này sẽ liên quan tới việc cập nhật giao diện người dùng đối với khách hàng, thêm lô-gíc đối với server, và bổ sung một số bảng trong cơ sở dữ liệu.



Có nghĩa là tất cả 3 nhóm – Nhóm Client, Nhóm Server, và Nhóm Cơ sở dữ liệu – phải cộng tác để hoàn thành story này. Không như mong đợi.

Hướng 2: Các nhóm liên thành phần.

Hướng thứ hai là tạo các Nhóm liên thành phần, tức là các nhóm không gắn với bất kỳ thành phần cụ thể nào.



Nếu nhiều story của bạn liên quan tới nhiều thành phần, kiểu chiến lược chia nhóm này sẽ làm việc tốt hơn. Mỗi nhóm có thể triển khai toàn bộ story bao gồm các phần client, các phần server, và các phần cơ sở dữ liệu. Do đó các nhóm có thể làm việc độc lập với nhau hơn, đó là một thứ tốt đẹp.

Một trong những điều đầu tiên chúng tôi đã làm khi bước đầu triển khai Scrum là giải tán các nhóm chuyên biệt về từng thành phần đang tồn tại (hướng 1) và tạo ra các nhóm liên thành phần với nhau để thay thế (hướng 2). Điều này giảm bớt một số trường hợp kiểu "chúng tôi không thể hoàn thành công việc này bởi vì chúng tôi đang đợi các vị ở nhóm server thực hiện phần việc của họ."

Tuy nhiên, đôi khi chúng tôi tạm thời xây dựng các nhóm chuyên biệt về từng thành phần khi có nhu cầu thực sự cần thiết.

Có tái sắp xếp lại các nhóm giữa Sprint hay không?

Mỗi Sprint thường khác nhau khá nhiều, phụ thuộc vào các loại story có độ ưu tiên hàng đầu tại một thời điểm cụ thể. Kết quả là, việc thiết lập nhóm sao cho tối ưu có thể khác nhau đối với mỗi Sprint.

Trong thực tế, hầu hết tất cả các Sprint chúng tôi đều nhận thấy một số thứ đại khái như "Sprint này thực sự là một Sprint không bình thường bởi vì (bla la bla) …". Sau một thời gian chúng tôi đã đưa ra khái niệm Sprint "bình thường". Không có các Sprint bình thường. Giống như vậy không có các gia đình "bình thường" hay con người "bình thường".

Một ý tưởng hay đó là một Sprint có thể có một nhóm chuyên về phần khách hàng, gồm tất cả mọi người hiểu rõ về cơ sở mã nguồn khách hàng. Sprint tiếp theo có thể có ý hay để có hai nhóm liên thành phần và chia những người chuyên về phần khách hàng giữa những nhóm đó.

Một trong những khía cạnh quan trọng của Scrum là "chất kết dính nhóm", tức là nếu một nhóm làm việc cùng nhau qua nhiều Sprint họ sẽ thường trở nên *rất gắn kết*. Họ sẽ học cách để tham gia vào luồng làm việc của nhóm và đạt mức độ năng suất đáng kinh ngạc. Nhưng cũng phải qua vài Sprint thì mới đạt được điều đó. Nếu bạn liên tục thay đổi các nhóm qua lại, bạn sẽ không bao giờ thực sự đạt được độ gắn kết cao của nhóm.

Vì vậy nếu bạn thực sự muốn tái sắp xếp lại các nhóm, hãy chắc chắn bạn đã xét đến các hậu quả. Đây có phải là một sự thay đổi ngắn hạn hay dài hạn? Nếu nó là thay đổi ngắn hạn hãy xét xem bỏ qua nó. Nếu nó là thay đổi dài hạn, bạn hãy tiến hành nó.

Có một ngoại lệ khi lần đầu tiên bạn triển khai Scrum với nhóm lớn. Trong trường hợp này có lẽ có chút thử nghiệm với việc chia nhóm nhỏ cho tới khi bạn nhận ra một vài thứ mà mọi người thoải mái với nó. Cần phải chắc chắn là mọi người hiểu rằng họ không phải quá lo lắng về những sai sót trong một vài lần đầu, miễn là bạn tiếp tục cải tiến.

Các nhân viên bán thời gian

Tôi chỉ có thể công nhận rằng những điều mà các cuốn sách Scrum nói – việc có các thành viên làm việc bán thời gian (part-time member) trong nhóm Scrum nhìn chung không phải là ý tưởng hay.

Giả sử bạn đưa Joe như một thành viên làm việc bán thời gian vào Nhóm Scrum của bạn. Đầu tiên hãy suy nghĩ cẩn thận. Bạn có thực sự cần Joe trong Nhóm của bạn? Bạn chắc chắn bạn không thể đề nghị Joe làm toàn thời gian? Các cam kết khác của anh ta là gì? Có ai khác có thể tiếp quản các cam kết của Joe và để Joe đảm nhận vai trờ hỗ trợ, thụ động hơn đối với cam kết đó? Joe có thể

tham gia nhóm của bạn toàn thời gian từ Sprint tiếp theo không và trong thời gian làm việc các trách nhiệm khác của anh ấy có thể chuyển giao cho ai khác hay không?

Đôi khi chẳng có cách nào được đưa ra cả. Bạn rất cần Joe bởi vì anh ấy là DBA [16]duy nhất trong công ty, nhưng thật tệ là những nhóm khác cũng cần anh ấy, vì thế anh ấy sẽ không bao giờ dành toàn thời gian cho nhóm của bạn, và công ty không thể thuê thêm DBA. Tốt thôi. Đó là lý do hợp lý để anh ấy làm việc bán thời gian (đây là tình huống chính xác đã xảy ra với chúng tôi). Nhưng hãy chắc chắn bạn thực sự tính toán điều này mỗi khi thực hiện.

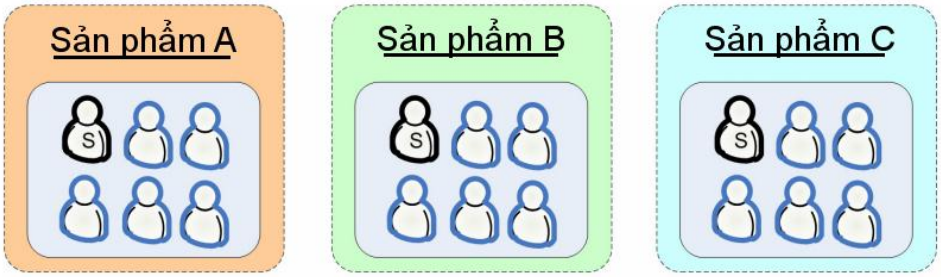
Nhìn chung tôi thích một nhóm có 3 người làm việc toàn thời gian hơn nhóm có 8 người làm việc bán thời gian.

Nếu bạn có một người sẽ chia thời gian của anh ấy cho nhiều nhóm, giống như trường hợp DBA đề cập ở trên, một ý hay là vẫn để anh ta làm việc chính cho một nhóm. Tìm ra nhóm nào cần anh ta nhất và coi nhóm đó là "đội nhà" của anh ấy. Khi không ai khác kéo anh ấy ra, anh ấy sẽ tham ra các buổi Họp Scrum Hằng ngày, các buổi Họp Kế hoạch Sprint, các buổi họp Cải tiến Sprint, v.v. của nhóm đó.

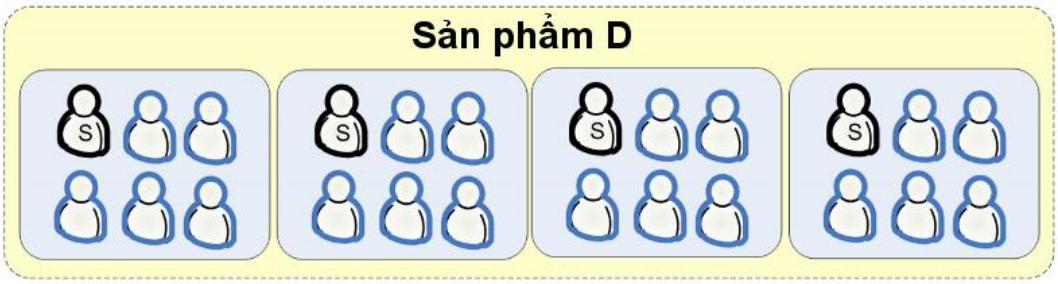
Chúng tôi thực hiện họp Scrum-của-các-Scrum như thế nào

Scrum-của-các-Scrum cơ bản là buổi họp thường kỳ nơi tất cả các Scrum Master tập hợp để trao đổi.

Tại một thời điểm chúng tôi có bốn sản phẩm, ở đó chỉ có ba sản phẩm có một Nhóm Scrum riêng, và sản phẩm cuối cùng có 25 người được chia vào vài Nhóm Scrum. Ví dụ như sau:



[16] DBA: Database Administrator = chuyên viên quản trị cơ sở dữ liệu. Đối với một số công ty chuyên gia về DBA thường không dồi dào, do độ phức tạp của công nghệ hoặc do chi phí. Do vậy, DBA thường là nguồn lực chia sẻ cho cả công ty.



Điều này có nghĩa chúng tôi có hai cấp độ Scrum-của-các-Scrum. Chúng tôi có một Scrum-của-các-Scrum ở "mức sản phẩm" bao gồm tất cả các nhóm thuộc Sản phẩm D, và một Scrum-của-các-Scrum với "mức công ty" gồm tất cả các sản phẩm.

Họp Scrum-của-các-Scrum mức sản phẩm

Buổi họp này rất quan trọng. Chúng tôi thực hiện nó một lần một tuần, thỉnh thoảng thì thường xuyên hơn thế. Chúng tôi tiến hành thảo luận các vấn đề liên quan, các vấn đề cân bằng nhóm, các chuẩn bị cho buổi họp kế hoạch Sprint tiếp theo, v.v. Chúng tôi đã dành 30 phút để thực hiện việc này nhưng thường xuyên bị quá giờ. Một cách khác thay thế là sẽ có buổi Scrum-của-các-Scrum hằng ngày nhưng chúng tôi chưa bao giờ thử điều này.

Chương trình buổi họp Scrum-của-các-Scrum của chúng tôi là

1. Bàn tròn, tất cả mọi người mô tả những gì mà nhóm của họ đã hoàn thành tuần trước, những gì họ lên kế hoạch hoàn thành cho tuần này và họ gặp những khó khăn, trở ngại gì.
2. Bất cứ vấn đề gì khác có tính chất liên nhóm cần phải được đưa ra, ví dụ các vấn đề tích hợp.

Chương trình làm việc cụ thể của buổi Scrum-của-các-Scrum không thực sự quan trọng với tôi, điều quan trọng là bạn có các buổi họp Scrum-của-các-Scrum thường kỳ.

Họp Scrum-của-các-Scrum mức công ty

Chúng tôi gọi cuộc họp này là "Nhịp đập" (The Pulse). Chúng tôi đã thực hiện cuộc họp này với các dạng khác nhau, với nhiều người tham gia khác nhau. Gần đây chúng tôi đã hoàn toàn bỏ khái niệm này và thay thế nó với cuộc họp toàn thể hằng tuần (tất cả mọi người liên quan đến việc phát triển sản phẩm). Buổi họp diễn ra trong 15 phút.

Gì cơ? 15 phút? Toàn thể? Tất cả thành viên của tất cả các nhóm của tất cả các sản phẩm? Điều đó có thực hiện nổi không?

Đúng thế, bạn (hoặc bất cứ ai tổ chức cuộc họp) có thể làm được nếu nghiêm chỉnh tuân thủ quy tắc giữ cho cuộc họp ngắn gọn.

Định dạng của cuộc họp này là:

1. Các tin tức và các cập nhật từ trưởng bộ phận phát triển. Ví dụ, thông tin về các sự kiện sắp tới.
2. Trình bày lần lượt theo vòng. Một người từ mỗi nhóm sản phẩm báo cáo những gì họ đã hoàn thành tuần trước, những gì họ lên kế hoạch hoàn thành tuần này và các khó khăn. Một vài người khác cũng tham gia báo cáo (lãnh đạo CM, lãnh đạo QA, v.v.).
3. Bất kỳ ai khác cũng được tự do bổ sung thông tin và đưa ra các câu hỏi.

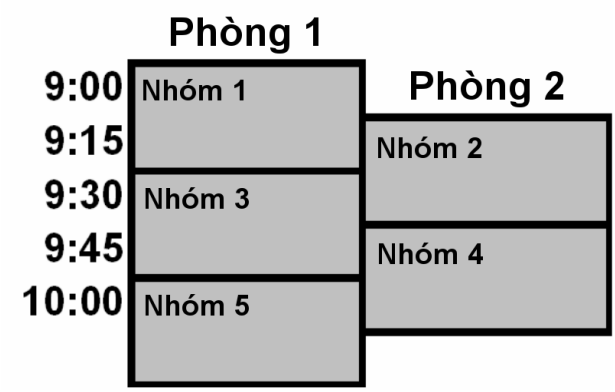
Đây là diễn đàn để tóm tắt thông tin, chứ không phải để thảo luận hay chỉ trích. Bằng cách duy trì như vậy, 15 phút thường là đủ để làm việc. Đôi khi chúng tôi vượt quá thời gian quy định, nhưng hiếm khi nào tổng thời gian vượt quá 30 phút. Nếu các thảo luận thú vị được nêu ra, tôi tạm dừng họ lại và mời những người quan tâm ở lại sau buổi họp và tiếp tục thảo luận.

Tại sao chúng tôi thực hiện cuộc họp toàn thể như thế? Bởi vì chúng tôi nhận thấy rằng các buổi Scrum-của-các-Scrum toàn thể chủ yếu là về việc báo cáo. Chúng tôi hiếm khi có các thảo luận thực sự trong nhóm này. Thêm vào đó, nhiều người khác bên ngoài nhóm bị đói về các loại thông tin kiểu này. Cơ bản, các nhóm muốn biết những gì các nhóm khác đang làm. Vì vậy chúng tôi thấy rằng nếu chúng tôi tiếp tục họp và mất thời gian thông báo cho từng người về những gì nhóm khác đang làm, vậy thì tại sao lại không để tất cả mọi người tham gia.

Đan xen Họp Scrum Hằng ngày

Nếu bạn có nhiều Nhóm Scrum cùng triển khai một sản phẩm, và tất cả họ thực hiện họp Scrum Hằng ngày cùng một thời gian, bạn gặp phải vấn đề. Product Owner (và những người tò mò như tôi) một ngày chỉ có thể tham gia Họp Scrum Hằng ngày của một nhóm.

Vì vậy chúng tôi đề nghị các nhóm tránh có các buổi Họp Scrum Hằng ngày cùng một thời điểm.



Mẫu lịch họp ở trên là khi chúng tôi có các buổi Họp Scrum Hằng ngày tại phòng riêng, chứ không phải trong phòng làm việc của nhóm. Các cuộc họp thường là 15 phút nhưng mỗi nhóm có một khoảng là 30 phút trong phòng để phòng khi vượt quá thời gian.

Điều này *vô cùng hữu ích* vì hai lý do.

1. Mọi người thích Product Owner và bản thân tôi có thể tham gia tất cả các buổi Họp Scrum Hằng ngày vào một buổi sáng. Không có cách nào hay hơn để có hình ảnh chính xác về những gì đang diễn với Sprint và các nguy cơ chính là gì.

2. Các nhóm có thể tham gia các buổi Họp Scrum Hằng ngày của các nhóm khác. Không xảy ra thường xuyên, trong một khoảng thời gian nào đó hai nhóm làm việc trên cùng một lĩnh vực, vì vậy vài thành viên nhảy sang các cuộc Họp Scrum Hằng ngày của nhóm khác để đồng bộ hóa công việc.

Nhược điểm là ít tự do hơn đối với nhóm – họ không thể chọn bất cứ thời điểm nào họ thích để Họp Scrum Hằng ngày. Dù vậy đây thực sự không phải là vấn đề đối với chúng tôi.

Các Nhóm chữa cháy

Chúng tôi gặp tình huống tại đó một sản phẩm lớn không thể phù hợp với Scrum bởi vì nó mất rất nhiều thời gian chữa cháy, tức là phải sửa lỗi trên các hệ thống phát hành vội. Đây quả thực là một vòng luẩn quẩn, họ quá bận với công việc chữa cháy nên họ không có thời gian chủ động thực hiện việc phòng cháy (tức là cải tiến thiết kế, tự động hóa kiểm thử, tạo các công cụ theo dõi, các công cụ cảnh báo, v.v.).

Chúng tôi đã giả quyết vấn đề này bằng cách tạo ra một nhóm làm nhiệm vụ chữa cháy, và một Nhóm Scrum.

Công việc của Nhóm Scrum (với phúc lành của Product Owner) cố gắng để ổn định hệ thống và phòng cháy một cách hiệu quả.

Nhóm chữa cháy (chúng tôi gọi họ là "hỗ trợ" thực sự) có hai việc.

1. Chữa cháy
2. Bảo vệ Nhóm Scrum khỏi tất cả các nhiễu loạn, bao gồn những thứ như loại bỏ các yêu cầu tùy hứng mà không xuất phát từ đâu cả.

Nhóm chữa cháy được đặt gần cửa nhất, Nhóm Scrum được đặt phía sau căn phòng. Vì vậy Nhóm chữa cháy có thể bảo vệ vật lý thực sự Nhóm Scrum khỏi các nhiễu loạn như sự hăm hở của nhân viên kinh doanh hay các khách hàng đang nóng giận.

Các nhà phát triển cao cấp được đặt cả ở hai nhóm, do đó một nhóm không bị quá phụ thuộc vào năng lực cốt lõi từ các nhóm khác.

Đây cơ bản là một sự nỗ lực để giải quyết vấn đề khởi sự với Scrum. Làm thế nào chúng tôi bắt đầu thực hiện Scrum nếu nhóm không có cơ hội lập kế hoạch công việc của họ nhiều hơn một ngày tại một thời điểm? Như đã nói, chiến lược của chúng tôi là chia thành các nhóm.

Điều này mang lại hiệu quả khá tốt. Do Nhóm Scrum được đưa vào phòng làm việc một cách chủ động, cuối cùng họ có thể ổn định được hệ thống. Trong khi ấy nhóm chữa cháy hoàn toàn bỏ được khái niệm về khả năng lập kế hoạch trước, họ hoàn toàn làm việc một cách phản xạ, chỉ sửa bất cứ các vấn đề bất thình lình xuất hiện tiếp theo.

Tất nhiên, Nhóm Scrum không hoàn toàn là thoát được các nhiễu loạn. Thường nhóm chữa cháy phải liên hệ với những người quan trọng trong Nhóm Scrum hoặc tệ hơn là toàn nhóm.

Dù sao, sau đôi ba tháng hệ thống đủ độ ổn định để chúng ta có thể hủy bỏ nhóm chữa cháy và tạo thêm Nhóm Scrum để thay thế. Những người chữa cháy khá vui vẻ tới công viên đập bỏ "mũ bảo hiểm" và tham gia Nhóm Scrum.

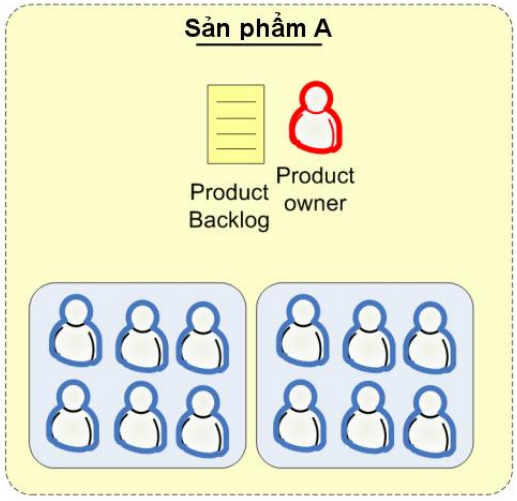
Chia nhỏ Product Backlog nên hay không nên?

Giả sử bạn có một sản phẩm và hai Nhóm Scrum. Bạn nên có bao nhiêu Product Backlog? Bao nhiêu Product Owner? Chúng tôi đã thử nghiệm ba mô hình. Lựa chọn mô hình hiệu quả dựa vào các kết quả của buổi họp kế hoạch Sprint.

Chiến lược 1: Một Product Owner, một Product Backlog

Đây là mô hình "Chỉ có một". Mô hình này là mô hình ưa thích của chúng tôi.

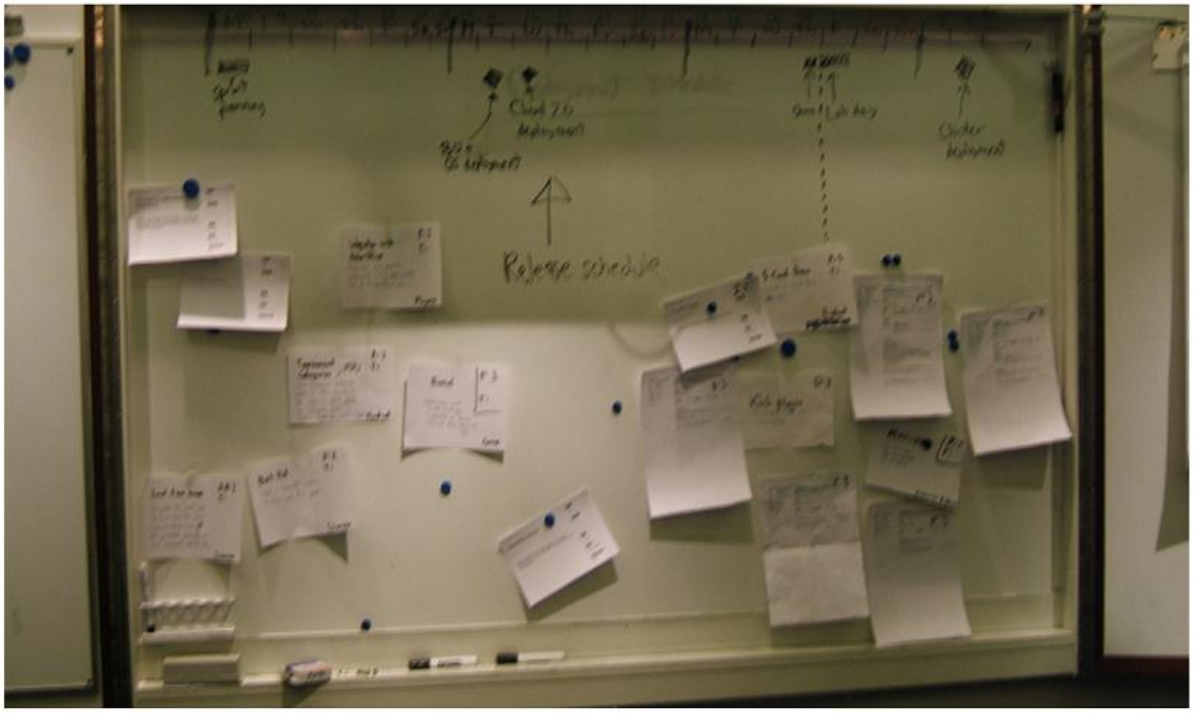
Điểm thú vị của mô hình này là bạn có thể các nhóm tự xây dựng dựa trên các hạng mục hiện tại được Product Owner đặt ưu tiên cao. Product Owner có thể tập trung vào những thứ anh ta cần, và để nhóm quyết định cách phân chia công việc.



Cụ thể hơn, đây là cách mà buổi họp kế hoạch Sprint diễn ra.

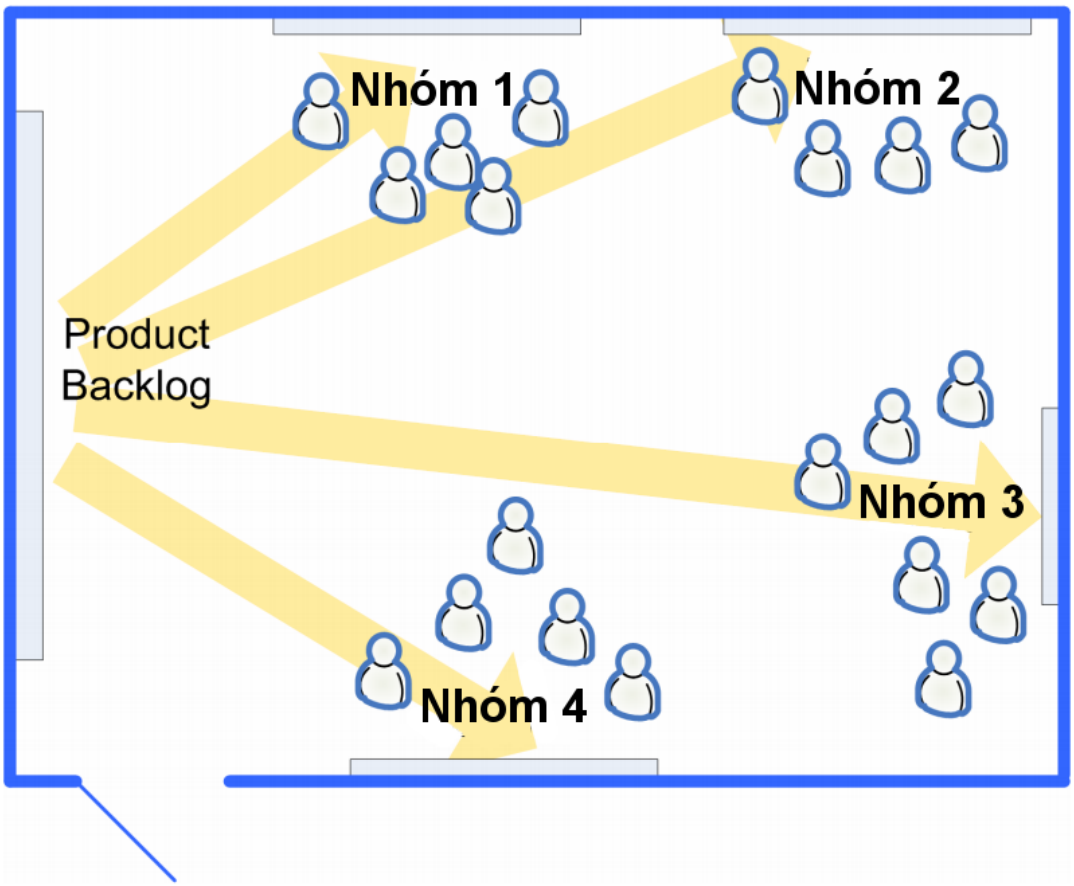
Cuộc họp kế hoạch Sprint tổ chức ở trung tâm hội nghị ở bên ngoài.

Trước buổi họp, Product Owner chọn một bức tường để làm "bức tường Product Backlog" và đặt các story lên đó (các thẻ chỉ mục), sắp xếp theo độ ưu tiên. Anh ra cứ đặt các story như thế cho đến khi bức tường được phủ kín, thường thì nhiều công việc sẽ nhiều hơn để làm trong một Sprint.



Mỗi Nhóm Scrum chọn một khoảng tường trống cho nhóm mình và ghi tên nhóm của mình lên đó. Cái đó gọi là "bức tường nhóm". Mỗi nhóm sau đó nhặt các story từ bức tường Product Backlog, bắt đầu từ các story có độ ưu tiên hàng đầu và kéo các thẻ chỉ mục vào bức tường của nhóm mình.

Dưới đây là hình ảnh minh họa, với mũi tên màu vàng ký hiệu cho luồng các thẻ chỉ mục chứa story từ bức tường Product Backlog tới bức tường của nhóm.



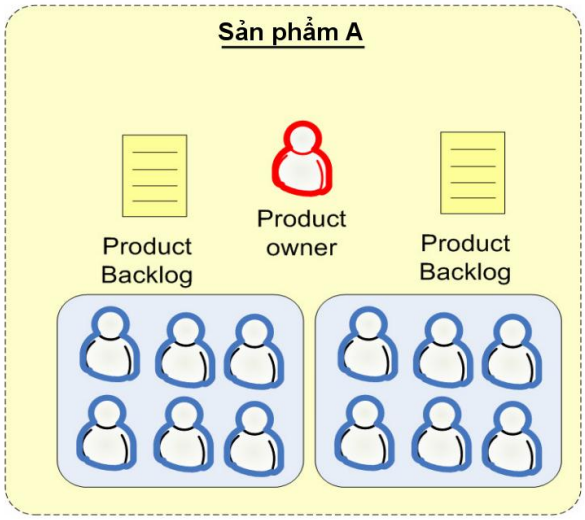
Khi buổi họp diễn ra, Product Owner và các nhóm thương thảo với nhau về các thẻ chỉ mục, di chuyển nó giữa các nhóm, di chuyển chúng lên và xuống để thay đổi độ ưu tiên, chia chúng thành các công việc nhỏ hơn v.v. Sau một giờ hoặc hơn, mỗi nhóm có phiên bản dự thảo đầu tiên của Sprint Backlog trên tường của nhóm. Sau đó các nhóm làm việc độc lập, thực hiện việc ước định thời gian và chia thành các nhiệm vụ.

Việc này rất hỗn loạn và lộn xộn, nhưng cũng khá hiệu quả, vui vẻ. Khi hết thời gian, tất cả các nhóm thường có đủ thông tin để bắt đầu Sprint của họ.

Chiến lược 2: Một Product Owner, nhiều Product Backlog

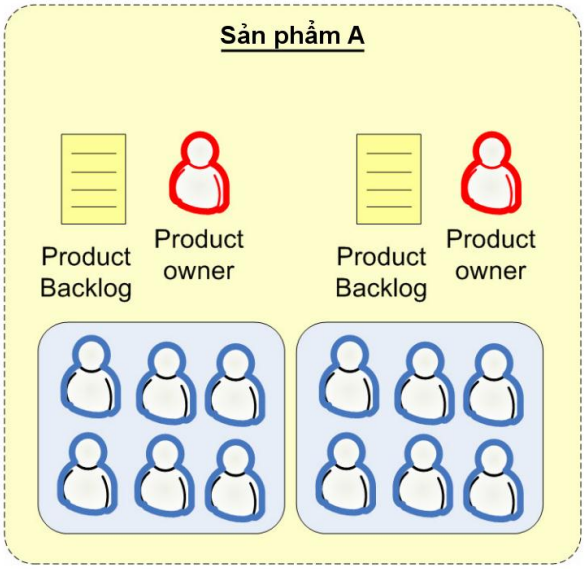
Trong chiến lược này, Product Owner duy trì nhiều Product Backlog, mỗi Product Backlog cho từng nhóm. Chúng tôi thực sự chưa thử qua cách tiếp cận này, nhưng chúng tôi đã tiến rất gần tới nó. Về cơ bản đây là kế hoạch dự phòng trong trường hợp cách tiếp cận thứ nhất thất bại.

Nhược điểm của chiến lược này là Product Owner sẽ phải phân bổ các story cho các nhóm, công việc có thể được làm tốt hơn bởi các nhóm.



Chiến lược 3: Nhiều Product Owner, mỗi người sở hữu một Product Backlog

Chiến lược này giống như chiến lược thứ 2, một Product Backlog cho một nhóm, nhưng cũng có một Product Owner riêng cho từng nhóm!.



Chúng tôi không thực hiện cách này, và có lẽ không bao giờ chúng tôi sẽ thực hiện như vậy.

Nếu có hai Product Backlog liên quan tới cùng cơ sở mã nguồn, điều này có thể gây lên các xung đột nghiêm trọng về lợi ích giữa hai Product Owner.

Nếu có hai Product Backlog liên quan hai cơ sở mã nguồn riêng biệt, điều này cơ bản giống như việc chia toàn bộ sản phẩm thành các sản phẩm con và thực hiện chúng một cách độc lập. Điều này có nghĩa chúng tôi quay trở lại tình huống một nhóm một sản phẩm, đây là cách hay và dễ dàng.

Phân nhánh mã nguồn

Với nhiều nhóm làm việc trên cùng một cơ sở mã nguồn, chúng tôi chắc chắn phải đối phó với việc phân nhánh mã nguồn trong hệ thống SCM [17](Quản lý cấu hình phần mềm – software configuration management). Có nhiều sách và bài viết nói về cách nhiều người làm việc với cùng cơ sở mã nguồn, vì vậy tôi không đề cập chi tiết ở đây. Tôi không có điều gì mới hay có ý tưởng cách mạng nào để bổ sung vào các cách đã được mô tả đó. Tuy nhiên tôi sẽ tóm tắt một vài bài học quan trọng nhất mà các nhóm của chúng tôi đã học được cho tới nay.

- Nghiêm chỉnh bám theo luồng chính với trạng thái nhất quán. Điều này có nghĩa, ít nhất, mọi thứ sẽ biên dịch và các kiểm thử đơn vị sẽ đạt. Có khả năng tạo ra các bản phát hành chạy tốt tại bất cứ thời điểm nào. Tốt hơn là xây dựng hệ thống build liên tục và tự triển khai lên một môi trường kiểm thử mỗi đêm.

- Đánh dấu mỗi lần phát hành. Bất cứ khi nào bạn phát hành để kiểm thử chấp nhận, chắc chắn có một thẻ đánh dấu vào luồng công việc chính để xác định chính xác cái gì đã được phát hành. Điều đó có nghĩa bạn có thể trở lại và tạo các nhánh bảo trì vào bất cứ thời điểm nào trong tương lai.

- Tạo các nhánh mới chỉ khi cần thiết, một quy tắc tốt về việc này là chia nhánh các dòng mã nguồn mới chỉ khi bạn không thể sử dụng các dòng mã nguồn đang có mà không vi phạm chính sách về dòng mã nguồn. Khi gặp vấn đề, đừng chia nhánh. Tại sao? Bởi vì mỗi hành động chia nhánh sẽ mất công quản lý và tăng độ phức tạp.

- Sử dụng các cách chia nhánh chủ yếu để tách biệt các vòng đời khác nhau. Bạn có thể hoặc không thể quyết định có việc mỗi Nhóm Scrum viết mã theo các dòng mã của chính họ. Nhưng nếu bạn kết hợp các bản sửa lỗi ngắn hạn với các thay đổi dài hạn trên cùng dòng mã nguồn, bạn sẽ thấy rất khó để phát hành các bản sửa lỗi ngắn hạn!

---

[17] SCM = Software Configuration Management: các phần mềm cơ bản hỗ trợ nhóm phát triển quản lí được sự thay đổi trong phần mềm. Đôi khi SCM được gọi không đúng với cái tên là Hệ quản lí phiên bản (Revision Control System). Các SCM phổ biến hiện nay là SVN, Git, CVS v.v.

- Thường xuyên đồng bộ. Nếu bạn đang làm việc trên cách nhánh, đồng bộ với dòng chính bất cứ khi nào bạn có một bản build. Hàng ngày khi bạn làm việc, hãy đồng bộ từ dòng chính về nhánh của bạn, để nhánh của bạn được cập nhật ứng với các thay đổi của các nhóm khác. Nếu điều này làm bạn mệt mỏi thì hãy chấp nhận thức tế rằng sẽ còn những điều tồi tệ hơn đang đón chờ bạn ở phía trước.

Họp Cải tiến nhiều nhóm

Chúng tôi thực hiện cải tiến Sprint khi có nhiều nhóm cùng triển khai một sản phẩm như thế nào?

Ngay sau khi sơ kết Sprint xong, sau những tràng vỗ tay, mỗi nhóm đi vào một phòng riêng của mình hoặc một nơi nào đó thoải mái ngoài văn phòng. Họ thực hiện các đánh giá tôi mô tả khá nhiều ở trang 81, “Chúng tôi Cải tiến Sprint như thế nào”.

Trong suốt buổi Họp Kế hoạch Sprint (buổi họp mà tất cả các Nhóm có mặt, do chúng tôi thực hiện các Sprint đã được đồng bộ hóa với mỗi sản phẩm), điều đầu tiên chúng tôi làm là để mỗi nhóm có một người phát biểu tóm tắt lại các điểm quan trọng trong các cải tiến của họ. Mất khoảng 5 phút cho mỗi nhóm. Sau đó chúng tôi mở cuộc thảo luận trong khoảng từ 10 đến 20 phút. Tiếp theo chúng tôi giải lao. Rồi chúng tôi tiếp tục bắt đầu kế hoạch Sprint thực tế.

Chúng tôi đã không thử bất kỳ cách nào khác cho nhiều nhóm, công việc này như thế là đủ tốt rồi. Nhược điểm lớn nhất là không có thời gian nghỉ ngơi sau phần cải tiến Sprint, trước khi phần họp kế hoạch Sprint (xem trang 87 “Quãng nghỉ giữa các Sprint”).

Đối với các sản phẩm chỉ có một nhóm, chúng tôi không thực hiện bất cứ tóm tắt cải tiến nào trong suốt buổi họp kế hoạch Sprint. Không cần thiết phải làm việc đó, vì mọi người đã tham gia buổi họp cải tiến thực tế rồi.

16

Chúng tôi quản lí các nhóm phân tán như thế nào

Chuyện gì sẽ xảy ra khi các thành viên trong nhóm làm việc ở những khu vực địa lý khác nhau? Phần lớn các "điều kì diệu" của Scrum và XP dựa trên sự hợp tác chặt chẽ của các thành viên ở cùng một nơi làm việc như lập trình cặp và họp mặt với nhau hằng ngày.

Chúng tôi có một số nhóm phân tán về mặt địa lý, và chúng tôi cũng có những thành viên trong nhóm làm việc tại nhà.

Chiến lược của chúng tôi cho vấn đề này là khá đơn giản. Chúng tôi sử dụng mọi thủ thuật mà chúng tôi có thể để tối đa **độ rộng băng thông giao tiếp** giữa những thành viên của các nhóm phân tán. Tôi không chỉ muốn nói về băng thông giao tiếp như Mbit/giây (mặc dù tất nhiên đó là điều khá quan trọng). Tôi muốn nói đến băng thông giao tiếp theo nghĩa rộng hơn:

- Khả năng lập trình cặp với nhau.
- Khả năng gặp mặt trực tiếp trong các buổi Họp Scrum Hằng ngày.
- Khả năng thảo luận trực tiếp với nhau bất kỳ lúc nào.
- Khả năng phù hợp về con người và xã hội.
- Khả năng triển khai các cuộc họp tức thời với toàn bộ nhóm.
- Khả năng có những nhìn nhận giống nhau với Sprint Backlog, biểu đồ Burndown của Sprint, Product Backlog và các nguồn thông tin khác.

Một số biện pháp mà chúng tôi đã triển khai (hoặc đang triển khai, chúng tôi chưa thực hiện tất cả những biện pháp đó):

- Webcam và tai nghe cho tất cả các trạm làm việc.
- Các phòng hội thảo "từ xa" với webcam, micro hội thảo, các máy tính luôn sẵn sàng, phần mềm chia sẻ màn hình, v.v..

- "Các cửa sổ từ xa" Các màn hình lớn tại nơi làm việc, cho phép hiển thị thường xuyên các khu vực làm việc khác. Thiết lập một dạng cửa sổ ảo giữa hai phòng ban. Bạn có thể đứng ở đó và vẫy tay ra hiệu. Bạn có thể thấy ai đang ngồi ở bàn làm việc và những ai đang nói chuyện với nhau. Điều này là để tạo ra cảm giác "này, chúng ta đang ở đây cùng với nhau".
- Những chương trình thuyên chuyển, để nhân viên ở chỗ này có thể đến chỗ khác theo định kì.

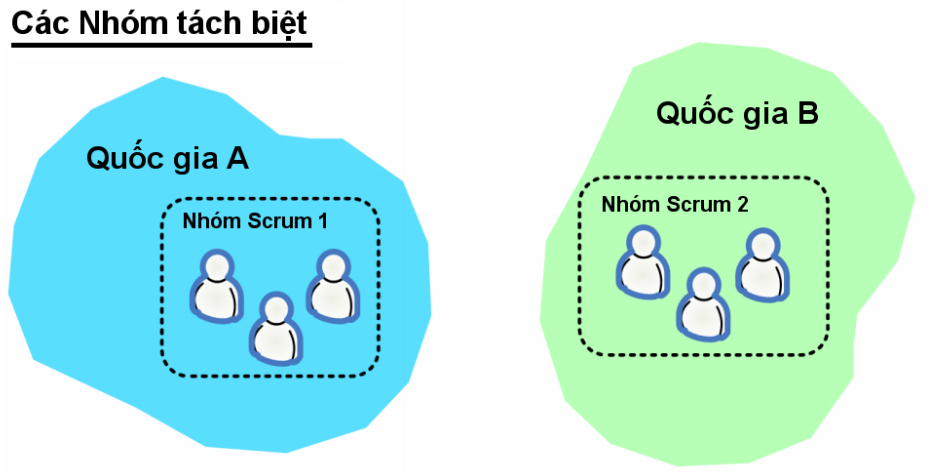
Sử dụng những kỹ thuật này và nhiều hơn nữa chúng tôi đang bắt đầu một cách chậm chạp nhưng chắc chắn để có được cách thức tiến hành các buổi Họp Kế hoạch, Sơ kết , Cải tiến Sprint, Họp Scrum Hằng ngày, v.v., với các thành viên của nhóm làm việc phân tán.

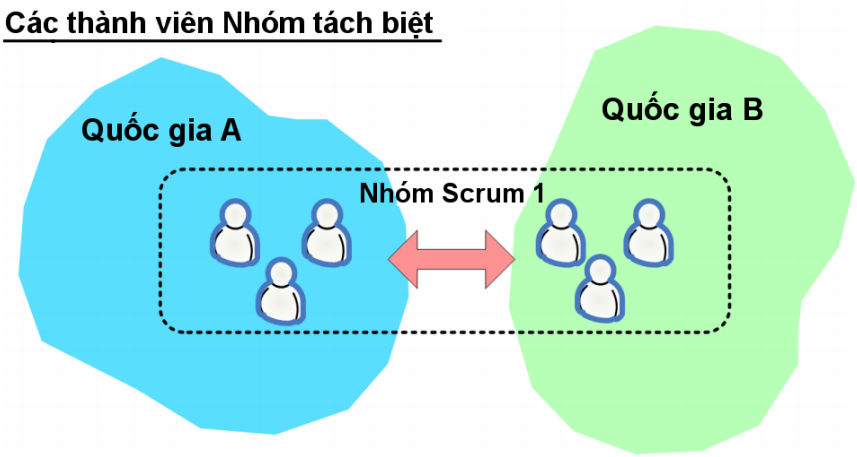
Như thường lệ đó tất cả đều là thử nghiệm. Thanh tra => thích nghi => thanh tra => thích nghi => thanh tra => thích nghi => thanh tra => thích nghi => thanh tra => thích nghi…

Làm việc ở nước ngoài

Chúng tôi có một vài nhóm ở nước ngoài và đã có kinh nghiệm với cách xử lý sao cho việc sử dụng Scrum là hiệu quả.

Có hai chiến lược chính sau đây: các nhóm tách biệt hoặc các thành viên nhóm tách biệt.





Chiến lược đầu tiên, các nhóm tách biệt, là một lựa chọn hấp dẫn. Tuy nhiên, chúng tôi đã bắt đầu với chiến lược thứ hai, các thành viên nhóm tách biệt. Có một số lý do cho sự chọn lựa này:

1. Chúng tôi muốn các thành viên nhóm có được sự hiểu biết rõ về nhau.
2. Chúng tôi muốn hạ tầng truyền thông tuyệt vời giữa hai địa điểm, và muốn cung cấp cho các nhóm một động lực mạnh mẽ để thiết lập điều này.
3. Ban đầu, nhóm ở nước ngoài thường quá nhỏ để tạo thành một Nhóm Scrum hiệu quả.
4. Chúng tôi muốn có một khoảng thời gian tập trung cao độ để chia sẻ kiến thức trước khi các nhóm ngước ngoài độc lập sẽ là một lựa chọn khả thi.

Sau một thời gian chúng tôi có thể thuận lợi để chuyển sang chiến lược “các nhóm tách biệt”.

Các thành viên làm việc ở nhà

Đôi khi ở nhà làm việc có thể thực sự tốt. Đôi khi bạn có thể hoàn thành nhiều chương trình hơn trong một ngày làm việc ở nhà so với cả tuần làm việc ở cơ quan. Ít nhất là nếu bạn không có đứa nhóc nào :o)

Tuy nhiên, một nguyên tắc cơ bản trong Scrum đó là cả nhóm nên được sắp xếp làm việc cùng nhau. Chúng tôi phải làm gì?

Về cơ bản chúng tôi để điều này cho nhóm quyết định khi nào và như thế nào là chấp nhận được để ngồi ở nhà làm việc. Một số thành viên nhóm thường thay phiên nhau làm việc ở nhà. Tuy nhiên, chúng tôi khuyến khích các nhóm làm việc cùng với nhau “hầu hết” mọi thời điểm.

Khi các thành viên nhóm làm việc ở nhà họ tham gia Họp Scrum Hằng ngày bằng cách sử dụng hội thoại qua Skype (đôi khi dùng cả video). Họ luôn luôn làm việc trực tuyến qua hệ thống tin nhắn tức thời. Không tốt bằng việc làm cùng nhau ở một phòng nhưng cũng tạm ổn.

Một lần chúng tôi đã thử quan niệm các ngày Thứ Tư được chỉ định như là *ngày tập trung*. Điều đó về cơ bản có nghĩa là "nếu bạn muốn là việc ở nhà, được thôi, nhưng hãy làm điều đó vào các ngày Thứ Tư." Và tra soát lại với nhóm của bạn" Điều này diễn ra khá tốt với nhóm mà chúng tôi đã thử nghiệm. Thường thì hầu hết cả nhóm ở nhà vào các ngày Thứ Tư và họ hoàn thành được nhiều thứ, trong khi vẫn cộng tác tốt với nhau. Vì chỉ là một ngày, các thành viên nhóm không bị mất đi sự đồng bộ trong nhóm của mình. Mặc dù vậy, vì một số lý do điều này không bao giờ bắt gặp với những nhóm khác.

Việc toàn bộ mọi người làm việc ở nhà không thực sự là vấn đề đối với chúng tôi.

17
Danh mục kiểm tra của Scrum Master

Trong chương cuối này, tôi sẽ nói với các bạn về *"Danh mục kiểm tra của Scrum Master"* của chúng tôi, trong đó liệt kê hầu hết những công việc quản lý hành chính của các Scrum Master. Những thứ này rất dễ quên. Chúng tôi tránh những điều chung chung kiểu như "gỡ bỏ những trở ngại cho nhóm".

Mở đầu Sprint

- Sau cuộc họp lập kế hoạch Sprint, cần tạo trang thông tin của Sprint.
    - Thêm một liên kết tới trang của bạn từ bảng điều khiển trên Wiki.
    - In trang thông tin đó và dán nó trên tường, nơi mà các thành viên thường qua lại.
- Gửi email tới mọi người thông báo về việc bắt đầu một Sprint mới. Trong đó, nội dung bao gồm mục tiêu Sprint và liên kết tới trang thông tin của Sprint.
- Cập nhật tài liệu thống kê dữ liệu Sprint. Thêm nội dung như *Tốc độ ước tính*, kích thước nhóm, thời gian triển khai của Sprint, v.v.

Hoạt động hàng ngày

- Đảm bảo Họp Scrum Hằng hàng được bắt đầu và kết thúc đúng giờ.
- Đảm bảo việc thêm\bớt các story vào Sprint Backlog khi cần thiết để giữ cho Sprint diễn ra theo đúng lịch trình.
    - Đảm bảo rằng Product Owner sẽ được thông báo về những thay đổi đó.
- Đảm bảo Sprint Backlog và Biểu đồ Burndown được Nhóm cập nhật hằng ngày.
- Đảm bảo các vấn đề\trở ngại sẽ được giải quyết hoặc báo cáo lên Product Owner hoặc là Trưởng bộ phận phát triển.

Kết thúc Sprint

- Tổ chức một buổi Sơ kết Sprint mở.
- Mọi thành viên cần được thông báo về buổi Sơ kết Sprint trước từ một đến hai ngày.
- Thực hiện Họp Cải tiến Sprint với mọi thành viên trong nhóm và Product Owner. Mời thêm trường bộ phận phát triển để trợ giúp việc đúng rút kinh nghiệm.

Cập nhật lại tài liệu thống kê dữ liệu Sprint. Bổ sung tốc độ thực tế và những điểm quan trọng rút ra từ cuộc Họp Cải tiến Sprint.

18
Lời chào tạm biệt

Phù! Tôi chưa bao giờ nghĩ sẽ viết được nhiều như thế này.

Hi vọng cuối sách này sẽ gợi cho bạn một số ý tưởng hữu ích, cho dù bạn là anh chàng mới biết đến Scrum hay đã là một chiến binh dạn dày kinh nghiệm .

Scrum cần được tùy biến khi áp dụng vào hoàn cảnh khác nhau. Do đó, thật khó để tranh luận về một cách thực hành tốt nhất theo một mức nào đó. Tuy nhiên, tôi vẫn rất muốn nghe phản hồi của bạn về Scrum. Hãy cho tôi biết sự khác biệt về phương pháp tiếp cận của bạn. Điều này giúp cho tôi những ý tưởng để cải thiện tốt hơn!

Đừng ngần ngại liên hệ với tôi theo địa chỉ: henrik.kniberg@crisp.se.

Tôi cũng thường dùng địa chỉ: scrumdevelopment@yahoogroups.com.

Nếu bạn thích cuốn sách này, bạn có thể tìm thêm những thông tin được cập nhật thường xuyên trên blog của tôi. Tôi hi vọng rằng sẽ có thể viết thêm nhiều đề tài về Java và Phương pháp phát triển phần mềm linh hoạt (Agile).

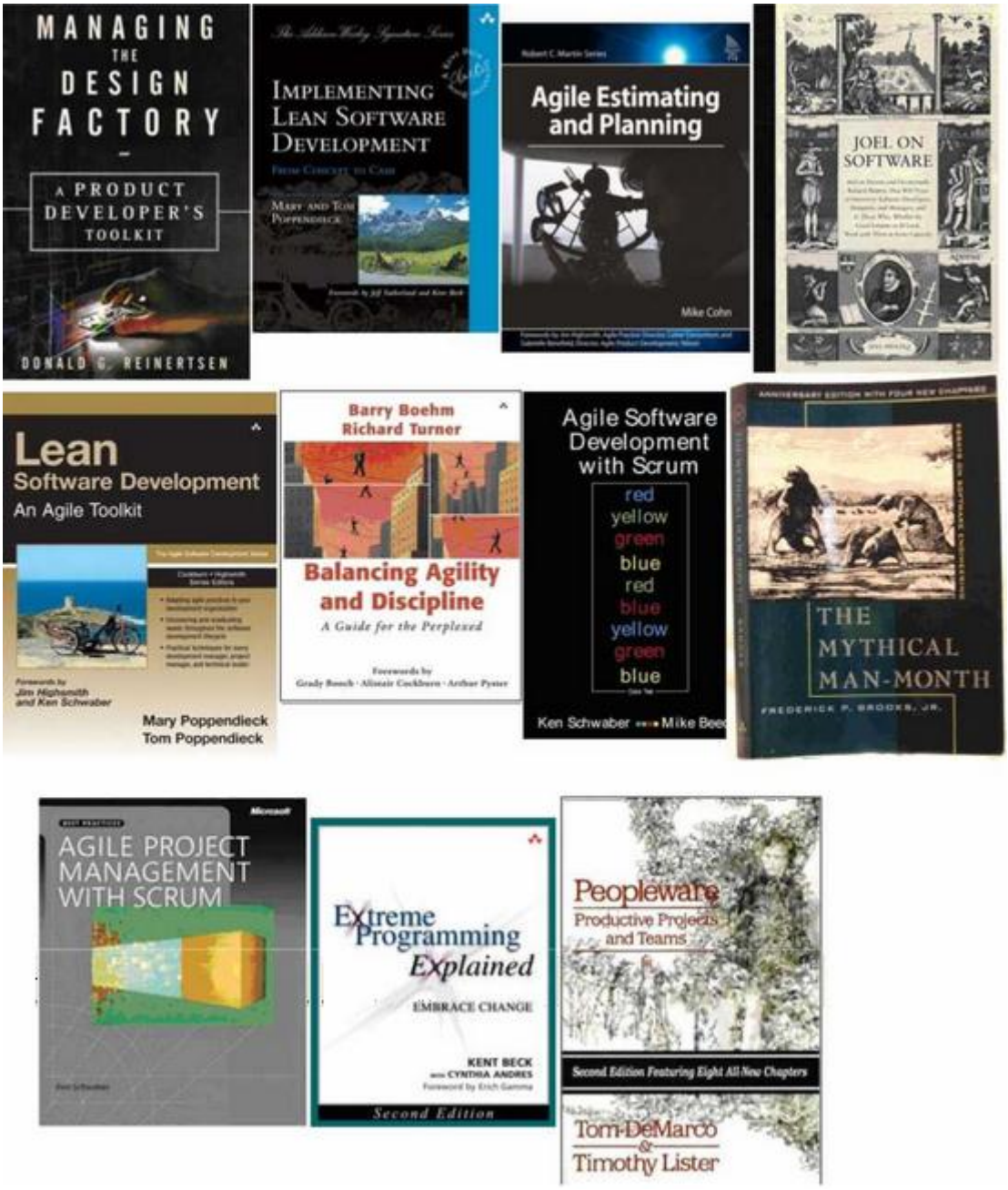
http://blog.crisp.se/henrikkniberg/

Ồ, và đừng quên...

Công việc trên có đúng không nhỉ?

Tài liệu nên đọc

Có một số cuốn sách đã cung cấp cho tôi nhiều cảm hứng và ý tưởng. Các bạn nên tìm đọc!



Về tác giả

Henrik Kniberg (henrik.kniberg@crisp.se) là chuyên gia tư vấn thuộc công ty Crisp tại Stockholm (www.crisp.se), chuyên gia về Java và Phát triển phần mềm linh hoạt (Agile).

Ngay từ lúc những cuốn sách đầu tiên về phương pháp XP và tuyên ngôn agile xuất hiện, Henrik đã nghiền ngẫm các nguyên lý Agile và thử nghiên cứu cách áp dụng hiệu quả vào trong các loại tổ chức khác nhau. Với vai trò là người đồng sáng lập và Giám đốc Kỹ thuật của công ty Goyada giai đoạn 1998-2003, ông đã có nhiều cơ hội trải nghiệm phong phú với phát triển hướng-kiểm-thử (Test-Driven Development) và các kỹ thuật thực hành linh hoạt khác trong quá trình xây dựng và quản lý một đội phát triển với 30 thành viên.

Cuối năm 2005, Henrik chuyển sang làm Trưởng bộ phận phát triển của một công ty ở Thụy Điển, chuyên lĩnh vực Game. Công ty này đã rơi vào cuộc khủng hoảng về vấn đề công nghệ và tổ chức. Với công cụ Scrum và XP, Henrik đã giúp công ty vượt qua cuộc khủng hoảng đó, giúp công ty áp dụng agile và sản xuất tinh gọn ở mọi cấp độ tổ chức trong công ty.

Một ngày thứ Sáu, tháng 11 năm 2006, Henrik đang ở nhà, nằm sốt trên giường và đã quyết định ghi lại một số ghi chú cho bản thân về những gì ông đã học được trong một năm qua. Tuy nhiên, sau khi ông bắt đầu viết, ông đã không thể dừng lại và sau ba ngày đánh máy điên cuồng cùng các bản vẽ, những ghi chép ban đầu đã phát triển thành một bài viết 80 trang với tên gọi "Scrum và XP từ những Chiến hào". Đây chính là cuốn sách mà chúng ta đang đọc.

Henrik có cách tiếp cận toàn diện và đã từng giữ nhiều vai trò khác nhau như nhà quản lý, nhà phát triển, Scrum Master, giáo viên và huấn luyện viên. Ông có đam mê giúp các công ty xây dựng những phần mềm tuyệt vời, đội ngũ phát triển xuất sắc và muốn tham gia vào bất cứ vai trò gì cần thiết.

Henrik lớn lên ở Tokyo và hiện đang sống ở Stockholm với người vợ Sophia và hai người con. Vào lúc rảnh rỗi, ông cũng là một nhạc sĩ, sáng tác âm nhạc, chơi Bass và phối nhạc trong một ban nhạc địa phương. Bạn có thể xem thông tin thêm trên website của ông: http://www.crisp.se/henrik.kniberg