NHẬP MÔN
CÔNG NGHỆ PHẦN MỀM
(INTRODUCTION TO SOFTWARE ENGINEERING)

Chương 3. Phương pháp Agile

- Outline
  - 1. Đặt vấn đề
  - 2. Tuyên ngôn của phương pháp Agile
  - 3. Các nguyên lý của phương pháp Agile
  - 4. So sánh Agile và Waterfall
  - 5. Nguyên tắc cơ bản của phương pháp Agile
  - 6. Tính phù hợp của các phương pháp Agile
  - 7. Một số phương pháp Agile phổ biến
    - Scrum
    - Lean development
    - Extreme Programming (XP)

1. Đặt vấn đề

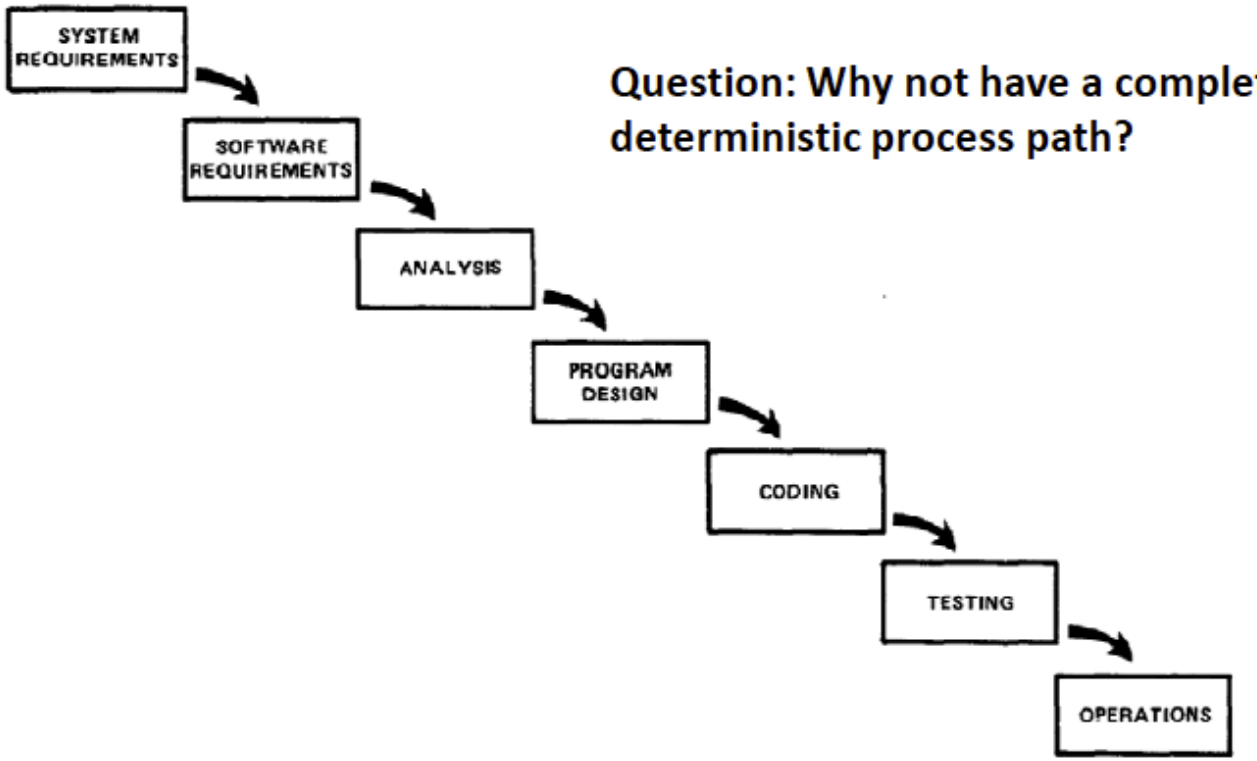
- Tại sao dự án phần mềm lại thất bại?
  - Không xác định được rủi ro
  - Xây dựng sai sản phẩm
  - Bị công nghệ chi phối
- Do đó:
  - Việc áp dụng một quy trình và vòng đời phần mềm tốt sẽ giúp giải quyết vấn đề này.
  - Tuy nhiên, điều đó vẫn không đảm bảo sự thành công. Vì không bao giờ có thể có một quá trình phát triển hoàn toàn hợp lý

Mô hình thác nước – nhắc lại

- Mô hình Thác nước (nổi bật những năm 70)
- Mô hình thác nước là mô hình vòng đời lâu đời nhất; được đề xuất bởi Winston Royce vào năm 1970.
- Mô hình này được gọi là thác nước vì nó thường được vẽ với một chuỗi các hoạt động qua các giai đoạn của vòng đời "xuống dốc" từ trái sang phải:
  - phân tích, yêu cầu, đặc tả, thiết kế, cài đặt, kiểm thử, bảo trì
- Có nhiều phiên bản của mô hình thác nước:
  - các giai đoạn / hoạt động có thể được cấu trúc theo các mức độ chi tiết khác nhau
  - phản hồi có thể linh hoạt hơn hoặc ít hơn

Vòng đời lý tưởng - Thác nước (Nghiêm ngặt) Không có phản hồi

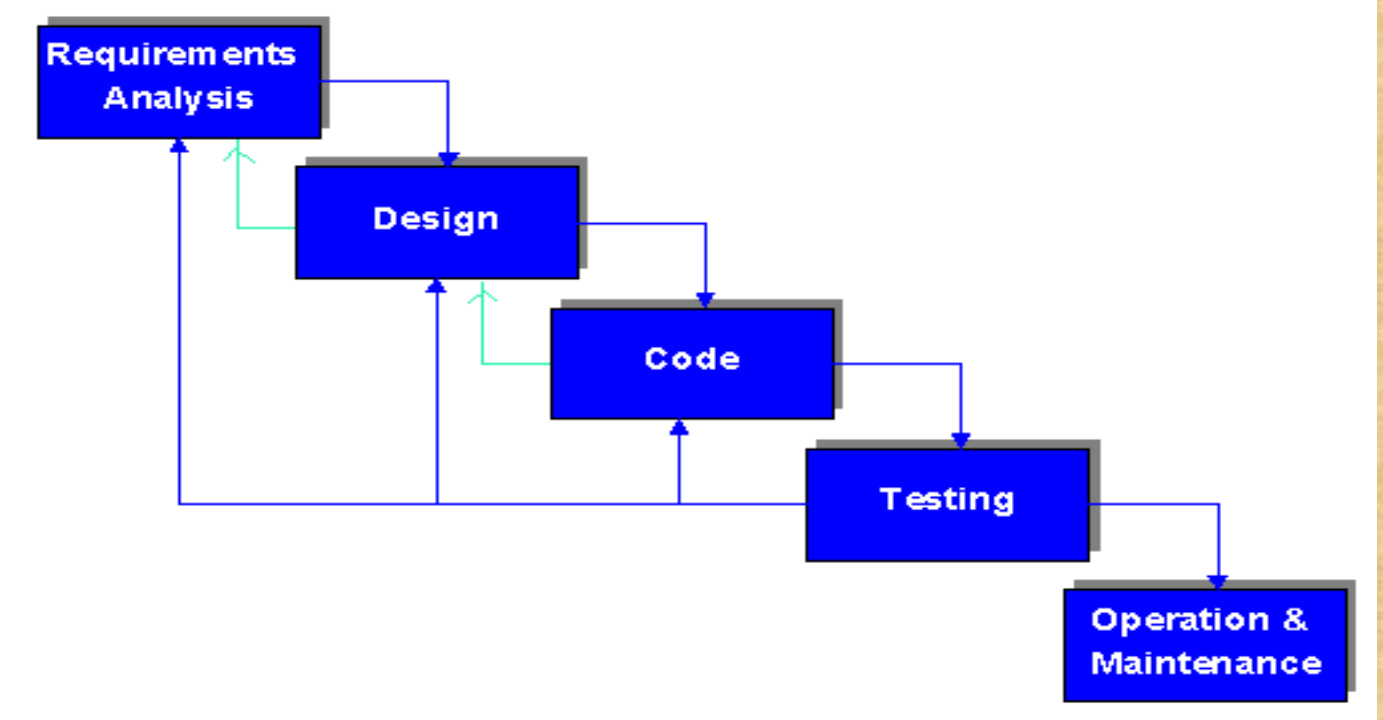
Life Cycle Ideal - (Strict) Waterfall With No Feedback



Question: Why not have a completely deterministic process path?

Non-strict waterfall model

- Mặc dù mô hình thác nước nhấn mạnh một chuỗi tuyến tính của các pha, trên thực tế, trong thực tế luôn có một lượng lớn sự lặp lại các pha trước đó, một điểm được tạo bởi các mũi tên dẫn ngược lên thác nước



Mô hình thác nước

- Điểm mạnh:
  - Nhấn mạnh việc hoàn thành một giai đoạn trước khi tiếp tục
  - Nhấn mạnh việc lập kế hoạch sớm, đầu vào của khách hàng và thiết kế
  - Nhấn mạnh kiểm tra như một phần không thể thiếu của vòng đời
  - Cung cấp các cổng chất lượng ở mỗi giai đoạn vòng đời
- Điểm yếu:
  - Phụ thuộc vào các yêu cầu được xác định sớm từ đầu
  - Phụ thuộc vào việc tách các yêu cầu khỏi thiết kế
  - Không khả thi trong một số trường hợp đòi hỏi có nhiều thay đổi
  - Nhấn mạnh vào sản phẩm hơn là quy trình

Yêu cầu

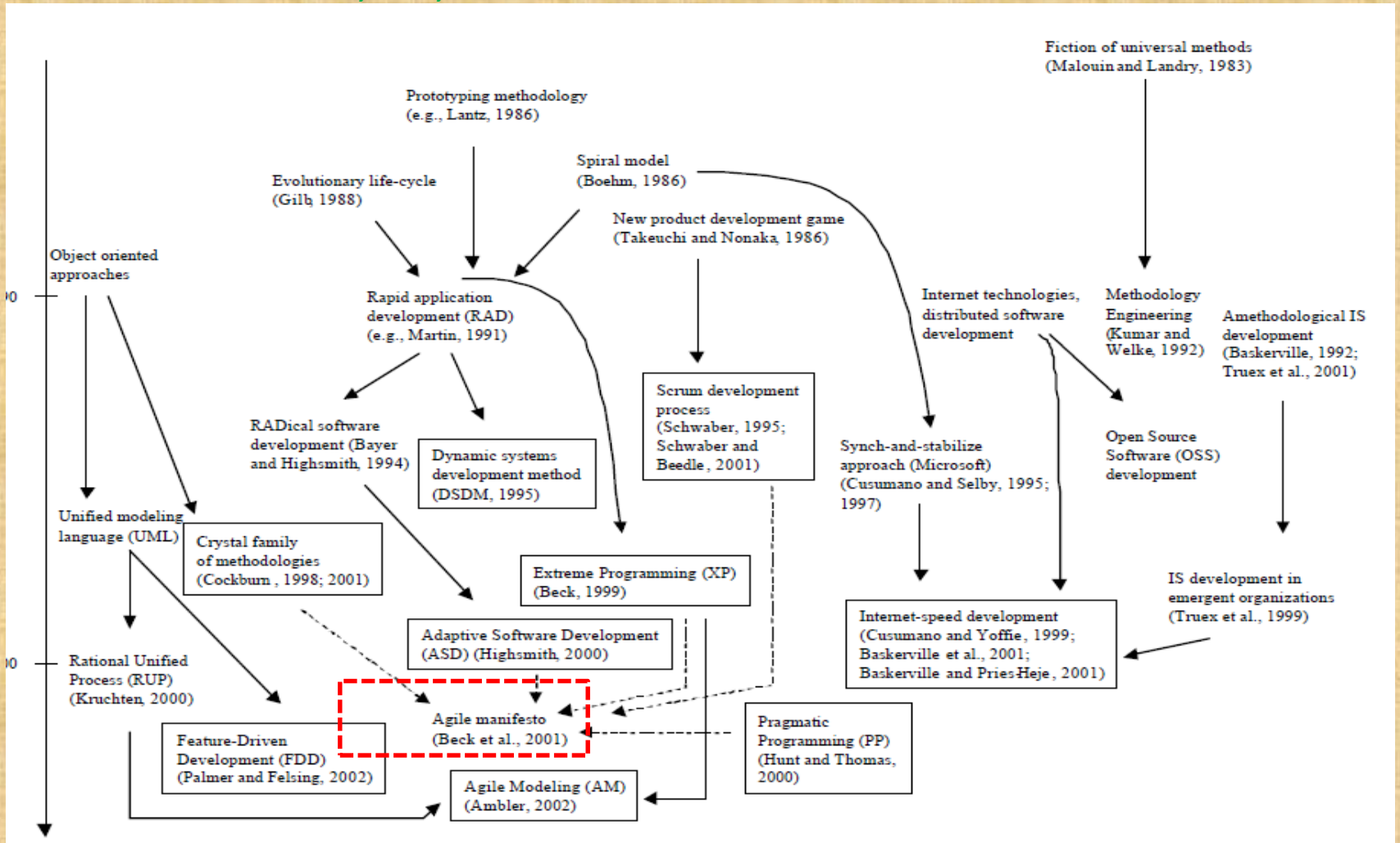
- Mô hình thác nước:
  - chú trọng nhiều vào tài liệu phân tích và thiết kế.
- Phương pháp Agile:
  - kết hợp quá trình tạo mẫu và dựa trên thử nghiệm: nơi có sự phát triển liên tục của lập trình (viết mã) và liên tục xác nhận các yêu cầu của khách hàng

Lịch sử của Agile

- Agile không phải là một bộ công cụ hay một phương pháp duy nhất, mà là một triết lý được đưa ra vào năm 2001.
- Agile thay đổi đáng kể từ cách tiếp cận phát triển phần mềm nặng theo hướng tài liệu - chẳng hạn như mô hình thác nước.

History of Agile: Where Did it Come From?

Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen, and Jussi Ronkainen. 2003. New directions on agile methods: a comparative analysis. In *Proceedings of the 25th International Conference on Software Engineering* (ICSE '03). IEEE Computer Society, Washington, DC, USA, 244-254.

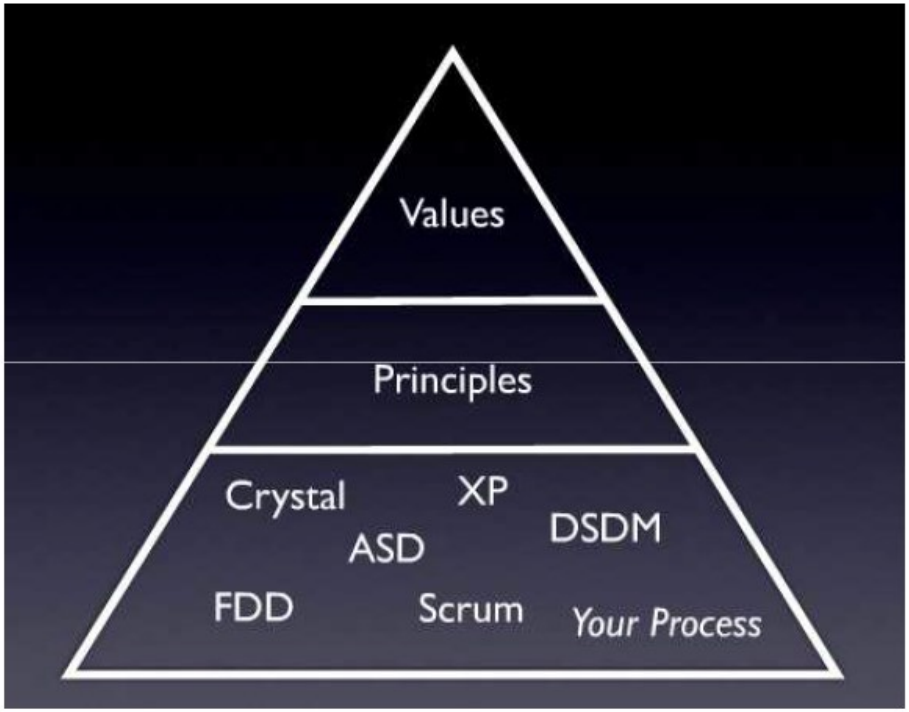


2. Tuyên ngôn phương pháp Agile

- Cách tốt hơn để phát triển phần mềm là làm điều đó và giúp người khác làm điều đó. Thông qua việc này, các giá trị sau sẽ được nhận ra:
  - **Các cá nhân và tương tác** qua **các quy trình và công cụ**
  - **Phần mềm làm việc** dựa trên **tài liệu toàn diện**
  - **Sự hợp tác của khách hàng** trong quá trình **đàm phán hợp đồng**
  - Đáp ứng **sự thay đổi** so với việc **tuân theo kế hoạch**
- Mặc dù các giá trị có trong các mục trên **bên phải**, nhưng các mục **bên trái** được coi trọng hơn.

http://agilemanifesto.org/

Agile: Values, Principles and Methods



3. Agile: 12 nguyên lý

1. Ưu tiên cao nhất của là làm hài lòng khách hàng thông qua việc phân phối sớm và liên tục các phần mềm có giá trị.
2. Hoan nghênh các yêu cầu thay đổi, ngay cả trong giai đoạn muộn của việc phát triển. Các quy trình nhanh khai thác sự thay đổi vì lợi thế cạnh tranh của khách hàng.
3. Cung cấp sản phẩm phần mềm thường xuyên, từ vài tuần đến vài tháng, ưu tiên khoảng thời gian ngắn hơn.
4. Người kinh doanh và nhà phát triển phải làm việc cùng nhau hàng ngày trong suốt dự án.
5. Xây dựng các dự án xung quanh những cá nhân có động lực. Cung cấp cho họ môi trường và sự hỗ trợ mà họ cần, và tin tưởng để họ hoàn thành công việc.
6. Phương pháp hiệu quả nhất để truyền tải thông tin đến và trong nhóm phát triển là trò chuyện trực tiếp.

3. Agile: 12 nguyên lý

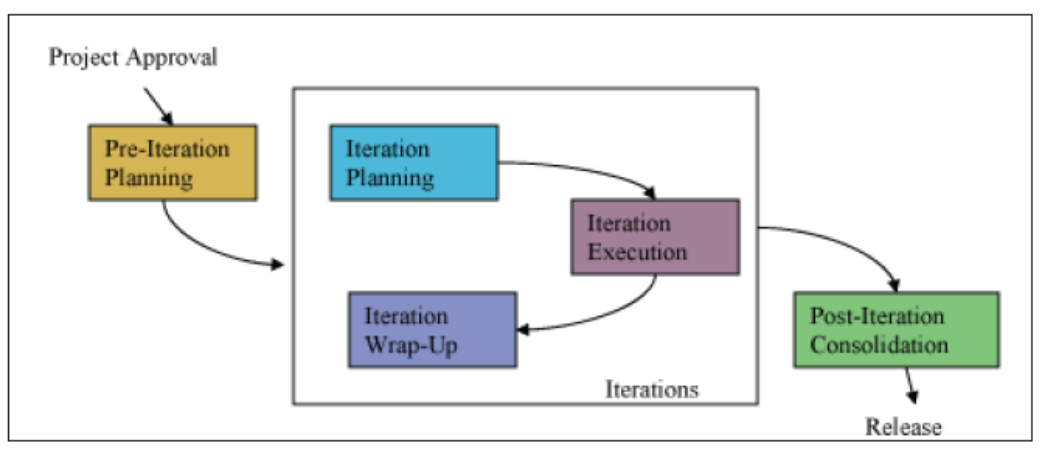
7. Phần mềm làm việc là thước đo chính của sự tiến bộ.
8. Các quy trình Agile thúc đẩy sự phát triển bền vững. Các nhà tài trợ, nhà phát triển và người dùng sẽ có thể duy trì một tốc độ phát triển liên tục.
9. Sự quan tâm liên tục, sự xuất sắc về kỹ thuật và thiết kế tốt giúp tăng cường sự nhanh nhẹn.
10. Sự đơn giản - nghệ thuật tối đa hóa khối lượng công việc chưa hoàn thành - là điều cần thiết.
11. Các kiến trúc, yêu cầu và thiết kế tốt nhất xuất hiện từ các nhóm dự án.
12. Theo định kỳ, các nhóm phản ánh về cách trở nên hiệu quả hơn, sau đó điều chỉnh hoạt động của mình sao cho cho phù hợp.

4. Agile vs Waterfall

- Waterfall có các giai đoạn riêng biệt với các điểm kiểm tra và phân phối, trong khi các phương thức nhanh có các lần lặp trong đó đầu ra của mỗi lần lặp nhanh là mã nguồn có thể được sử dụng để đánh giá và đáp ứng các yêu cầu thay đổi và phát triển của người dùng.
- Waterfall giả định rằng có thể hiểu rõ các yêu cầu ngay từ đầu. Nhưng trong phát triển phần mềm, các bên liên quan thường không biết họ muốn gì và không thể nói rõ các yêu cầu của họ. Với kiểu thác nước, sự phát triển hiếm khi mang lại những gì khách hàng muốn ngay cả khi đó là những gì khách hàng yêu cầu.
- Với Agile nhấn mạnh vào khách hàng và yêu cầu của họ.

Nguyên tắc cơ bản: số lần lặp

- Các phương pháp luận Agile gồm các bước lặp lại.
- Các nhóm nhỏ làm việc cùng với các bên liên quan để xác định các nguyên mẫu nhanh, bằng chứng về các khái niệm hoặc các phương tiện trực quan khác để mô tả vấn đề cần giải quyết. Nhóm xác định các yêu cầu cho việc lặp lại, phát triển mã, xác định và chạy các tập lệnh kiểm tra tích hợp và người dùng xác minh kết quả.
- Việc kiểm định diễn ra sớm hơn nhiều trong quá trình phát triển.



Tính phù hợp của các phương pháp Agile

- Khó xác định về loại dự án phần mềm nào phù hợp nhất cho cách tiếp cận Agile.
- Nhiều tổ chức lớn gặp khó khăn trong việc chuyển từ phương pháp truyền thống sang một phương pháp Agile (linh hoạt).
- Khi Agile có rủi ro:
  - Phát triển quy mô lớn (> 20 nhà phát triển)
  - Phát triển phân tán (các nhóm không nằm chung)
  - Các công ty kiểm soát kỳ quặc
  - Khách hàng hoặc người liên hệ không đáng tin cậy
  - Bắt buộc một quy trình nhanh trong nhóm phát triển
  - Nhà phát triển thiếu kinh nghiệm

Một số phương pháp Agile phổ biến

- Scrum
- Lean Development
- Extreme programming (XP)
- Adaptive Software Development (ASD)
- Agile Modeling
- Crystal Methods
- Dynamic System Development Methodology (DSDM)
- Feature Driven Development

Scrum

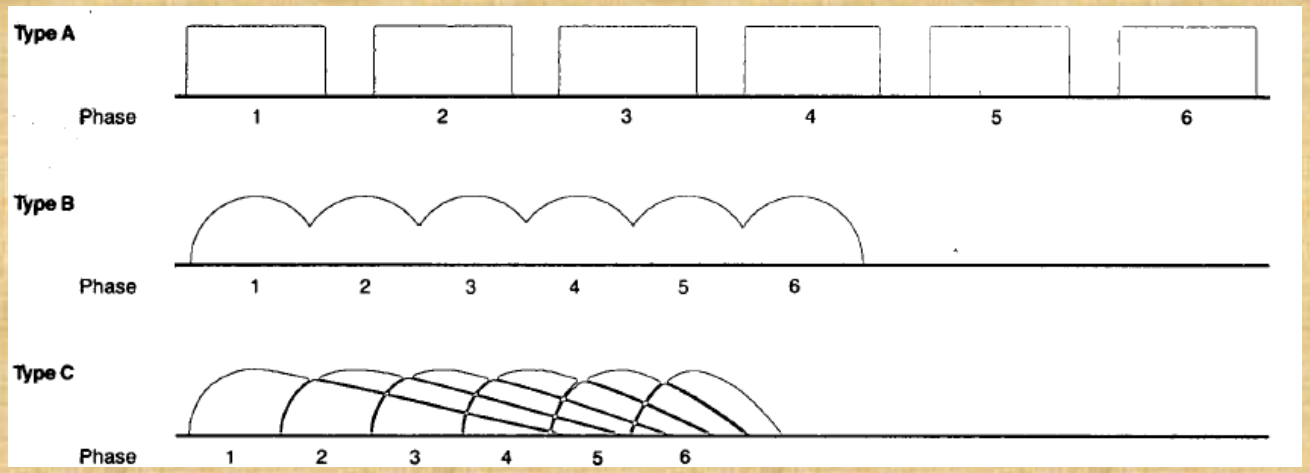
‘Scrum’ (liên quan đến “scrimmage”) là thuật ngữ chỉ một nhóm người chơi tụ tập với nhau để hoàn thành công việc trong môn bóng bầu dục. Trong phát triển phần mềm, công việc là đưa ra một bản phát hành.

Comes from Japan and based from industrial process control theory:

Takeuchi, Hirotaka and Nonaka, Ikujiro: *The New New Product Development Game*, Harvard Business Review, 1986.

“Stop running the relayy race and take up rugby”

"Dừng chạy cuộc đua tiếp sức và chơi bóng bầu dục"



Speed Up development

Scrums cho các vấn đề xấu

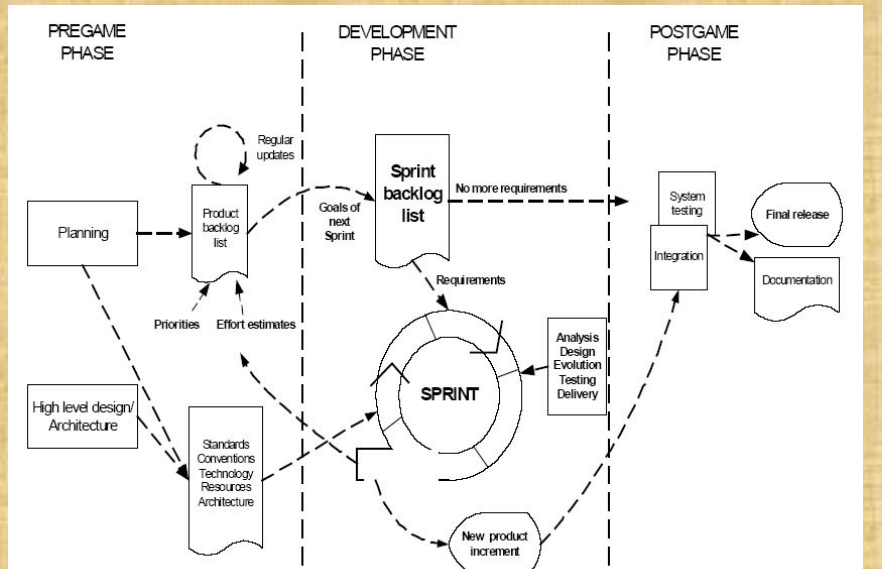
"Cách tiếp cận quản lý dự án thích ứng tốt nhất cho các dự án phần mềm xấu"

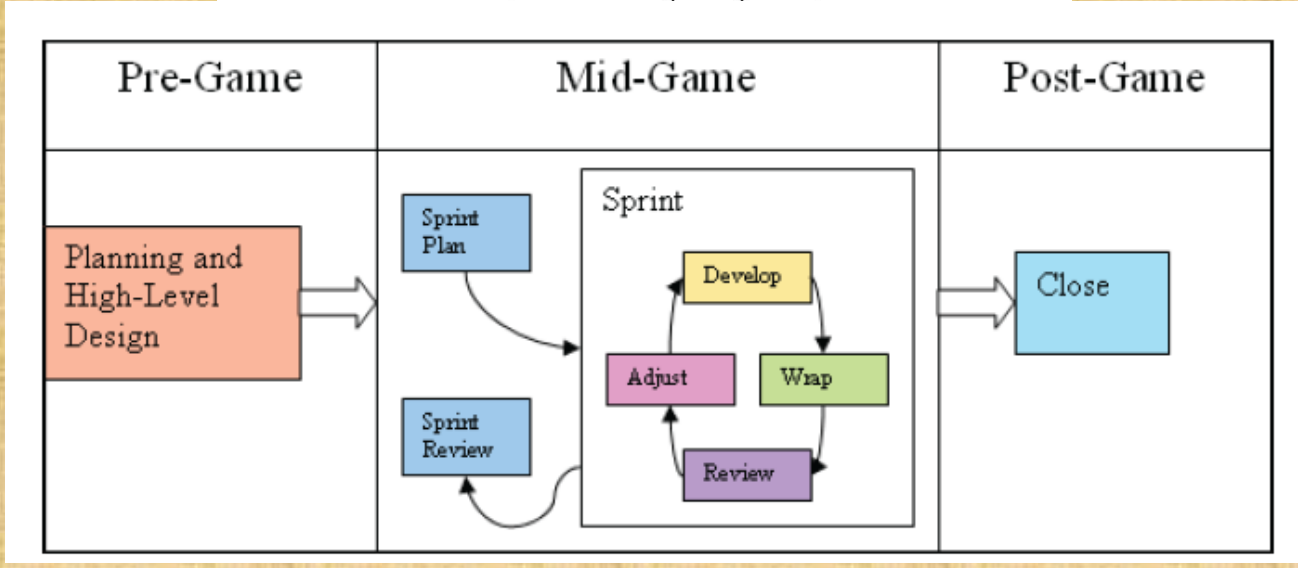
Peter DeGrace and Leslie Hulet Stahl.
***Wicked Problems, Righteous Solutions*.**
1990. Yourdon Press

Nhiều vấn đề hệ thống mà các nhà phát triển phần mềm gặp phải (các đặc điểm xấu):

1. Không có công thức chính xác cho một vấn đề xấu.
2. Những vấn đề xấu không có quy luật dừng.
3. Giải pháp cho các vấn đề xấu không phải đúng-hay-sai mà là tốt-hay-xấu
4. Không có thử nghiệm tức thời và cuối cùng nào về giải pháp cho một vấn đề xấu.
5. Mọi giải pháp được thực hiện cho một vấn đề xấu đều có hậu quả.
6. Các vấn đề xấu không có một tập hợp các giải pháp tiềm năng tốt.
7. Mọi vấn đề xấu xa về cơ bản là duy nhất.
8. Mọi vấn đề xấu có thể được coi là một nguyên nhân của một vấn đề khác.
9. Nguyên nhân của một vấn đề xấu có thể được giải thích theo nhiều cách.
10. Người lập kế hoạch (thiết kế) không có quyền sai.

Scrums phases





Main scrum concepts

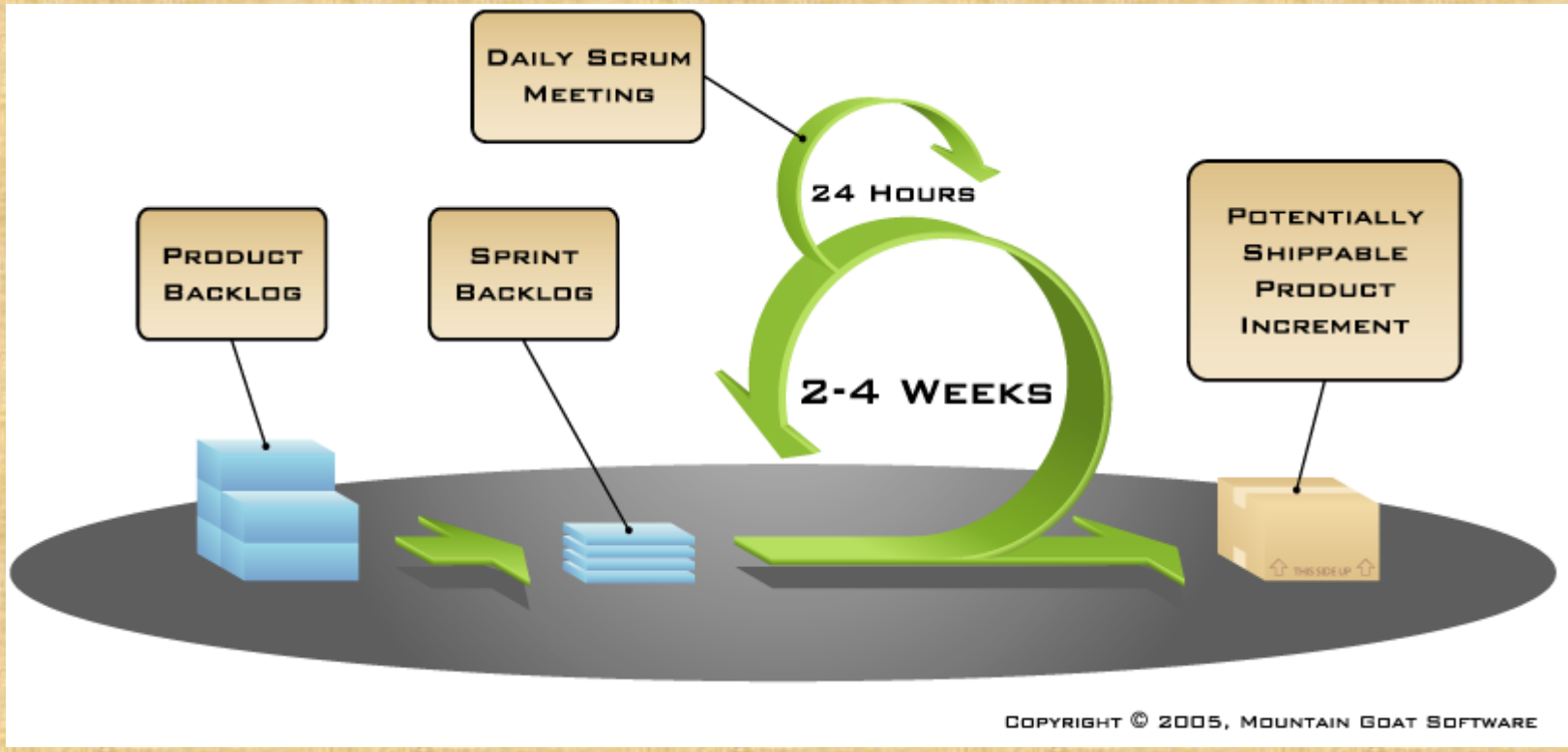
Burndown chart. Biểu đồ này, được cập nhật mỗi ngày, cho thấy công việc còn lại trong sprint. Biểu đồ burndown được sử dụng để theo dõi tiến trình spint và quyết định khi nào các mục phải được loại bỏ khỏi sprint backlog và hoãn lại sprint tiếp theo.

Product backlog. Product backlog là danh sách đầy đủ các yêu cầu — bao gồm lỗi, yêu cầu nâng cao, cải tiến khả năng sử dụng và hiệu suất — hiện không có trong bản phát hành sản phẩm.

ScrumMaster. ScrumMaster là người chịu trách nhiệm quản lý dự án Scrum.

Sprint backlog. Sprint backlog là danh sách các hạng mục tồn đọng được chỉ định cho một sprint, nhưng chưa hoàn thành. Trong thực tế phổ biến, không có hạng mục tồn đọng nào của sprint phải mất hơn hai ngày để hoàn thành. Sprint backlog giúp nhóm dự đoán mức độ nỗ lực cần thiết để hoàn thành một sprint.

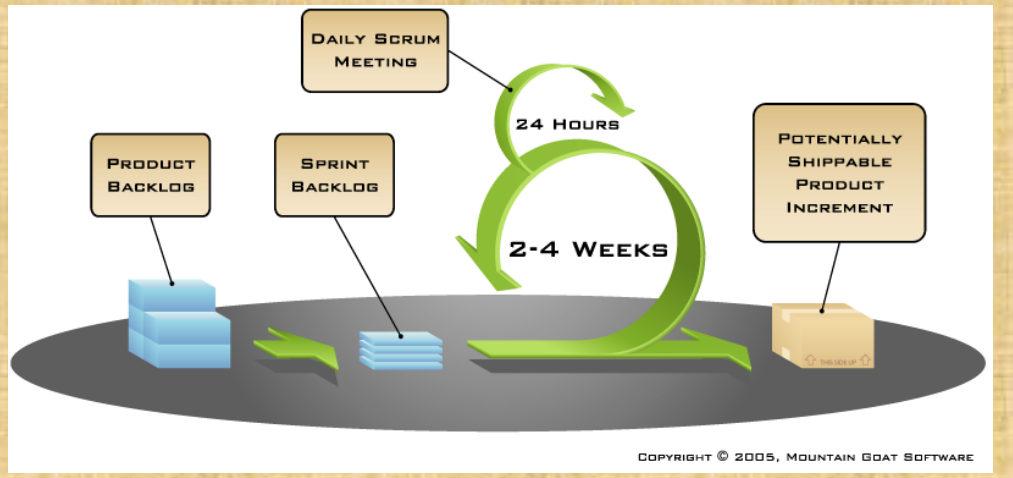
Scrum Meetings



The backlog is key: it is populated during the planning phase of a release and defines the scope of the release

Việc tồn đọng là khóa: nó được điền vào trong pha lập kế hoạch của lần phát hành sản phẩm và xác định phạm vi của bản phát hành

Scrum Meetings



Quá trình phát triển được chia thành một loạt các lần lặp ngắn được gọi là spint. Trước mỗi sprint, các thành viên trong nhóm xác định các hạng mục tồn đọng cho sprint. Khi kết thúc sprint, nhóm sẽ xem xét sprint để nêu rõ các bài học kinh nghiệm và kiểm tra tiến độ.

Trong thời gian một spint, nhóm có một cuộc họp hàng ngày. Mỗi thành viên trong nhóm mô tả công việc sẽ được thực hiện trong ngày hôm đó, tiến độ từ ngày hôm trước. Để giữ cho các cuộc họp ngắn gọn, cuộc họp phải được tiến hành với tất cả mọi người (họp đứng trong cùng một phòng).

Khi đủ các backlog đã được thực hiện để người dùng cuối tin rằng bản phát hành đáng được đưa vào sản xuất, kết thúc quá trình phát triển. Sau đó, nhóm thực hiện kiểm tra tích hợp, đào tạo và tài liệu khi cần thiết để phát hành sản phẩm.

Scrum Meetings

Cuộc họp đứng hàng ngày (còn được gọi là "cuộc họp hàng ngày", "cuộc họp hàng ngày", "điểm danh buổi sáng", v.v.), để giữ cuộc họp ngắn, được mô tả như sau:

- Cả nhóm họp hàng ngày để cập nhật tình trạng nhanh chóng.
- Tuy nhiên việc này không thực sự đảm bảo giúp phân biệt một phương án hiệu quả với việc lãng phí thời gian.
- Đối với những người có kinh nghiệm, với việc đứng lên theo bản năng, khi gặp sự cố họ sẽ biết phải điều chỉnh những gì để khắc phục tình hình.
- Đối với những người mới bắt đầu, khi mọi thứ gặp trục trặc, ít có khả năng họ sẽ tìm ra giải pháp (những gì phải làm) và có nhiều khả năng là từ bỏ hoàn việc nếu không được hỗ trợ.
- Điều này sẽ mang lại giá trị đáng kể cho cả đội.

Scrum Meetings

Mục đích cho daily stand-up là Good Start, Improvement, Focus, Team, Status (GIFTS)

Có một số mục tiêu cho cuộc họp đứng hàng ngày:

- Để giúp khởi đầu ngày mới tốt đẹp
- Để hỗ trợ cải thiện
- Để củng cố sự tập trung
- Để củng cố tinh thần đồng đội
- Để thông báo những gì đang xảy ra

The purpose is not to meet... it is to improve.
-- Joe Ely, "More on Daily Start-Up Meetings"

Scrum Meetings

Ai tham dự?
Mọi người và các đại diện từ các lĩnh vực khác nhau muốn biết và/hoặc đóng góp vào dự án. Trạng thái giao tiếp trong nhiều cuộc họp cần nhiều sự nỗ lực.

Do đó. Thay thế một số hoặc tất cả các cuộc họp và báo cáo bằng cuộc họp hàng ngày. Bất kỳ ai trực tiếp tham gia hoặc muốn biết về hoạt động hàng ngày của dự án đều nên tham gia cuộc họp đứng hàng ngày.

Nhưng. Những người không liên quan trực tiếp có thể làm gián đoạn cuộc họp nếu họ không rõ về hành vi được mong đợi. Điều này có thể được giải quyết bằng cách thông báo trước cho những người tham gia và quan sát viên mới về các chỉ tiêu dự kiến.

Scrum Meetings

Ở đâu và khi nào và như thế nào?

- **Gặp gỡ nơi công việc xảy ra**
- **Cùng một nơi, cùng một lúc**
- **Vào đầu ngày? … hay không?**
- **Đứng gần nhau**
- **Chuẩn bị trước**
- **Mười lăm phút hoặc ít hơn… Thực hiện giải quyết vấn đề ngoại tuyến**
- **Khuyến khích quyền tự chủ (xoay người điều hành?)**

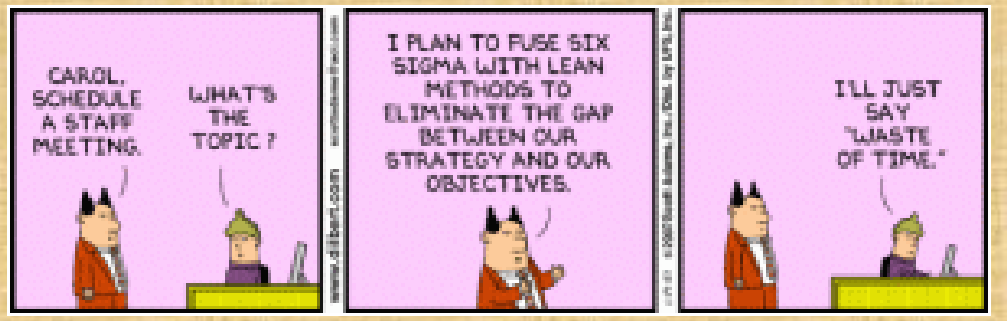
Scrum

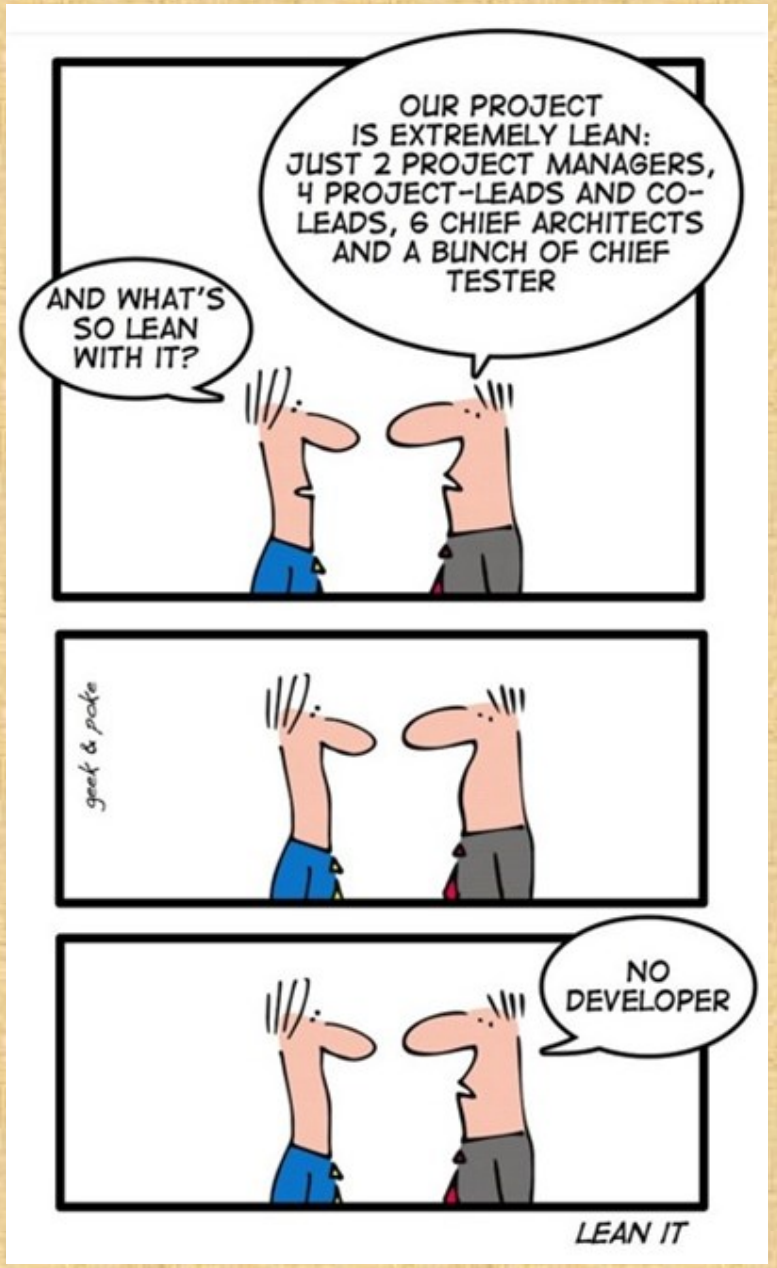
Pros	Cons
Scrum sử dụng Sprint nhỏ để chia hệ thống thành các thành phần nhỏ hơn một cách hiệu quả và phân chia cho các nhóm.	Scrum chỉ hoạt động khi ban quản lý “tin tưởng các nhà phát triển sử dụng phán đoán của riêng mình” để hoàn thành nhiệm vụ. Các thuộc tính chính của scrum là khả năng kiểm soát nhẹ nhàng và tinh tế. Vì vậy, nếu đội ngũ phát triển còn trẻ và chưa trưởng thành, Scrum trở rất rủi ro.
Một hoạt động chính trong scrum là “cuộc họp scrum hàng ngày”, giúp các thành viên trong nhóm đưa ra bằng chứng về việc hoàn thành nhiệm vụ và cho phép cải tiến liên tục, do đó cho phép áp dụng kỹ thuật từ dưới lên một cách nhanh chóng	Scrum được thiết kế lý tưởng cho công ty có “các phương pháp nhanh hiện có”. Do đó, một công ty phải có một số kiến thức làm việc về các phương pháp nhanh trước khi sử dụng Scrum.

Lean development (phát triển tinh gọn)

J Paul Gibson: Agile Methods

October 2011





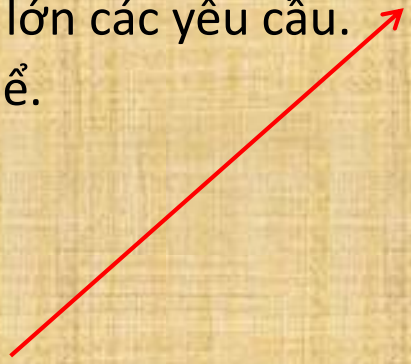
Lean development

Lean software development là bản dịch các nguyên tắc và thực hành sản xuất tinh gọn sang lĩnh vực phát triển phần mềm. Được điều chỉnh từ Hệ thống sản xuất Toyota, một tiểu văn hóa ủng hộ tinh gọn đang xuất hiện trong cộng đồng Agile:

is a translation of lean manufacturing principles and practices to the software development domain. Adapted from the Toyota Production System, a pro-lean subculture is emerging from within the Agile community:

Lean Software Development – key ideas

- Cuộc sống sẽ dễ dàng hơn rất nhiều nếu khách hàng ngừng thay đổi suy nghĩ của họ.
- Hãy để khách hàng trì hoãn quyết định của họ về chính xác những gì họ muốn càng lâu càng tốt và khi họ yêu cầu điều gì đó, hãy đưa nó cho họ nhanh đến mức họ không có thời gian để thay đổi ý định.
- Các thiết kế tuyệt vời đến từ các nhà thiết kế vĩ đại và các nhà thiết kế vĩ đại hiểu rằng các thiết kế xuất hiện khi họ phát triển sự hiểu biết ngày càng cao về vấn đề, chứ không phải thu thập số lượng lớn các yêu cầu.
- Cung cấp hệ thống làm việc nhanh nhất có thể.



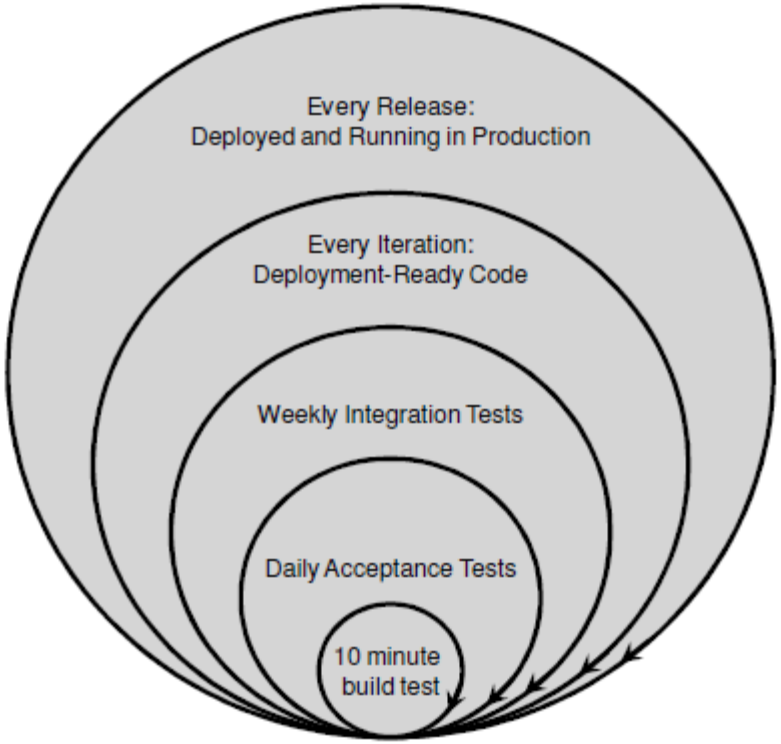
QUESTION: Do you agree with this?

Lean Software Development

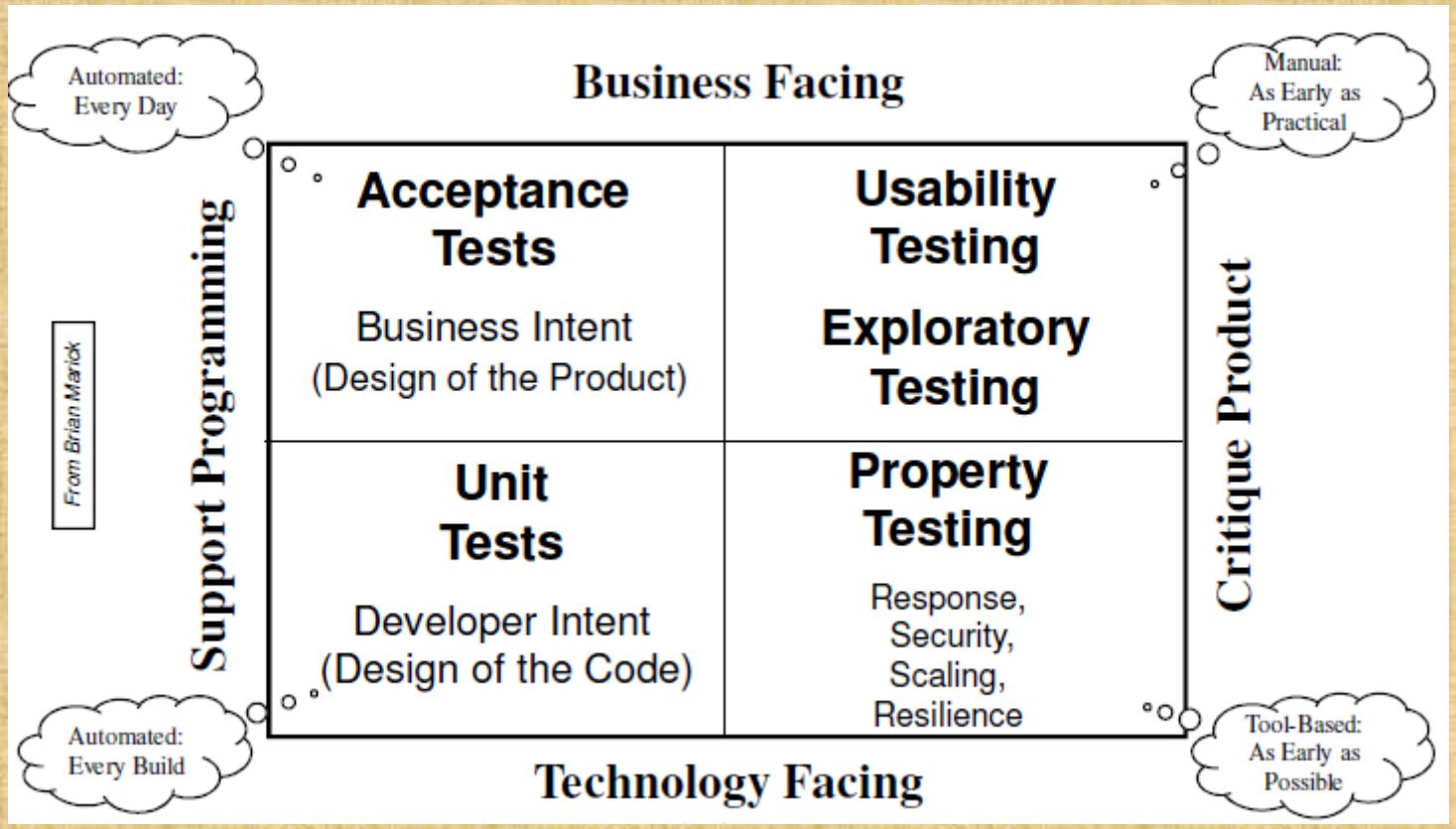
1. **Eliminate waste.** Bất kỳ hoạt động nào không "tự trả giá" trong nỗ lực giảm thiểu ở những nơi khác trong hệ thống và cần được loại bỏ.
2. **Amplify learning.** Các nhà phát triển luôn cần học các phương pháp mới để tạo ra một hệ thống mạnh mẽ nhất.
3. **Decide as late as possible.** Lợi ích của việc đưa ra quyết định vào phút cuối là tránh đưa ra quyết định sai lầm sớm và sau đó phải sửa chữa nó sau này.
4. **Deliver as fast as possible.** Nguyên tắc này là nếu bạn cung cấp rất nhanh, nó sẽ làm giảm cơ hội thay đổi yêu cầu. Điều này có thể phải trả giá rất lớn nếu sự thay đổi đến muộn trong quá trình phát triển.
5. **Empower the team**. Để mọi người chịu trách nhiệm, có động lực và gắn bó với nhau như một đội, họ cần phải chịu trách nhiệm về kết quả và được ủy quyền để biến nó thành hiện thực.
6. **Build integrity in:** Điều quan trọng là hệ thống duy trì tính toàn vẹn trong suốt chu trình phát triển. Điều đó có nghĩa là phải kiểm tra tích hợp, kiểm thử đơn vị và kiểm thử chung, đặc biệt là từ khách hàng.
7. **See the whole.** Đừng chia nhỏ hệ thống thành nhiều phần mà hãy giữ nguyên toàn bộ hệ thống.

Lean software development: focus on testing

- ✓Every few minutes
  - × Build & run unit tests
  - × ***STOP*** if the tests don't pass
- ✓Every day
  - × Run acceptance tests
  - × ***STOP*** if the tests don't pass
- ✓Every week
  - × Run production testes
  - × ***STOP*** if the tests don't pass
- ✓Every iteration
  - × Deployment-ready code
- ✓Every Release
  - × Deploy and run in production



Lean software development: focus on testing



Lean Software Development

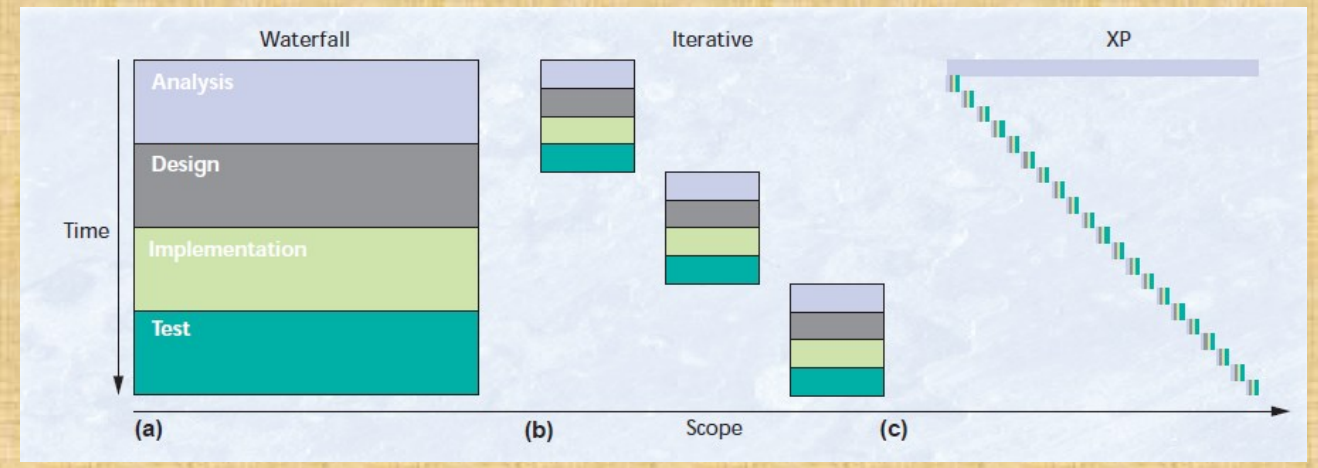
Pros	Cons
Tư duy toàn bộ hệ thống, mặc dù khó đối với hệ thống phức tạp, giúp đảm bảo tính nhất quán và toàn vẹn của hệ thống. Giảm thời gian tích hợp kể từ khi được phát triển dưới dạng đơn vị số ít.	Với hệ thống lớn hoặc hệ thống phức tạp, cách duy nhất để các nhà phát triển hình dung cấu trúc của nó là thông qua việc phân vùng hệ thống. Nhưng LSD đề xuất điều ngược lại, điều này có thể khó thực hiện.
Nhóm làm việc thiết kế các quy trình của riêng mình, đưa ra các cam kết riêng, thu thập thông tin cần thiết để đạt được mục tiêu và tự lập chính sách để đạt được các mốc quan trọng của mình.	Quyết định càng muộn càng tốt có thể ảnh hưởng xấu đến lịch trình. Điều này có thể làm tổn hại đến sự phát triển song song và làm tăng thời gian thực hiện.

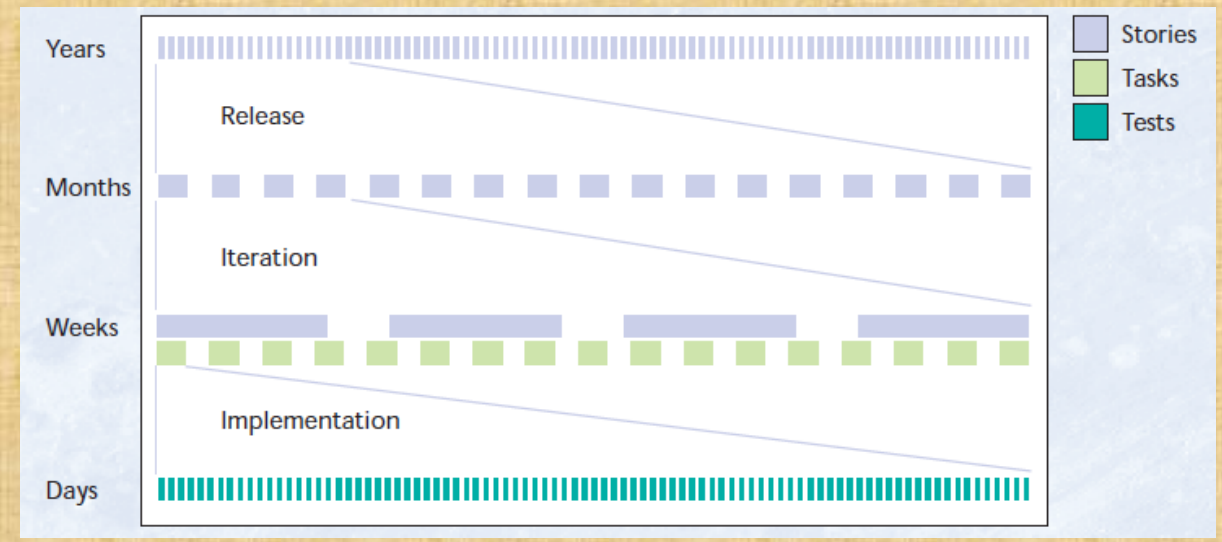
Extreme Programming (XP)

XP is not this *extreme*!

Extreme Programming

From: ***Embracing Change with Extreme Programming***, Beck, 1999





Extreme Programming (XP)

***Embracing Change with Extreme Programming*, Beck, 1999**

Không thể lập trình cho đến khi biết mình đang lập trình gì.

XP coi khoảng thời gian trước khi hệ thống đi vào sản xuất lần đầu tiên là một bất thường nguy hiểm trong vòng đời của dự án và cần được khắc phục càng nhanh càng tốt. Tuy nhiên, mọi dự án đều phải bắt đầu từ đâu đó.

Kết hợp phân tích tổng thể dưới dạng các câu chuyện là số lượng của một trường hợp sử dụng. Mỗi câu chuyện phải theo định hướng kinh doanh, có thể kiểm tra và ước tính được.

Một tháng là một khoảng thời gian dài để đưa ra những câu chuyện cho một dự án.

Extreme Programming (XP)

***Embracing Change with Extreme Programming*, Beck, 1999**

Làm thế nào để xác định thời điểm nên lập trình?

Khách hàng chọn bản phát hành tiếp theo bằng cách chọn các tính năng có giá trị nhất (được gọi là các câu chuyện trong XP) trong số tất cả các câu chuyện có thể có, được xác định bởi chi phí của các câu chuyện và tốc độ của nhóm trong việc cài đặt các câu chuyện.

Khách hàng chọn câu chuyện của lần lặp lại tiếp theo bằng cách chọn những câu chuyện có giá trị nhất còn lại trong bản phát hành, được thông báo lại bằng chi phí của các câu chuyện và tốc độ của nhóm.

Các lập trình viên biến các câu chuyện thành các nhiệm vụ chi tiết nhỏ hơn mà họ tự nhận trách nhiệm.

Sau đó, lập trình viên biến một nhiệm vụ thành một tập hợp các trường hợp kiểm thử để chứng minh rằng nhiệm vụ đã hoàn thành.

Làm việc với một đối tác, lập trình viên làm cho các trường hợp kiểm thử chạy, đồng thời cải tiến thiết kế để duy trì thiết kế đơn giản nhất có thể cho toàn bộ hệ thống.

XP Practices I (Usually associated with Agile)

Planning game. Khách hàng quyết định phạm vi và thời gian phát hành dựa trên các ước tính do lập trình viên cung cấp. Các lập trình viên chỉ triển khai các chức năng được yêu cầu bởi các câu chuyện trong lần lặp này.
Small releases. Hệ thống được đưa vào sản xuất trong vài tháng, trước khi giải quyết toàn bộ vấn đề. Các bản phát hành mới được thực hiện thường xuyên — bất cứ nơi nào từ hàng ngày đến hàng tháng.
Metaphor. Hình dạng của hệ thống được xác định bằng một phép ẩn dụ hoặc một tập hợp các phép ẩn dụ được chia sẻ giữa khách hàng và lập trình viên.
Simple design. Tại mọi thời điểm, thiết kế chạy tất cả các tests, truyền đạt mọi thứ mà lập trình viên muốn giao tiếp, không chứa mã trùng lặp và có ít mã nhất các lớp và các phương thức. Quy tắc này có thể được tóm tắt là, "Nói mọi thứ một lần và chỉ một lần.

XP Practices I (Usually associated with Agile)

Tests. Các lập trình viên viết các bài test đơn vị. Các tests này được thu thập và tất cả chúng phải chạy chính xác. Khách hàng viết các test chức năng cho các câu chuyện trong một lần lặp lại. Tất cả các thử nghiệm này cũng nên chạy, mặc dù trên thực tế, đôi khi phải đưa ra quyết định kinh doanh so sánh chi phí cho một lỗi đã biết và chi phí của sự chậm trễ.

Refactoring. Thiết kế của hệ thống được phát triển thông qua các biến đổi của thiết kế hiện có để giữ cho tất cả các thử nghiệm hoạt động.

Continuous integration. Tích hợp mã mới hệ thống hiện tại sau không quá vài giờ. Khi tích hợp, hệ thống được xây dựng từ đầu và tất cả các bài kiểm tra phải vượt qua hoặc các thay đổi bị loại bỏ.

XP Practices II (also found outside Agile)

Pair programming. Tất cả mã được viết bởi hai người tại một màn hình / bàn phím / chuột.

Collective ownership. Mọi lập trình viên đều cải thiện bất kỳ mã nào ở bất kỳ đâu trong hệ thống vào bất kỳ lúc nào nếu họ thấy có cơ hội.

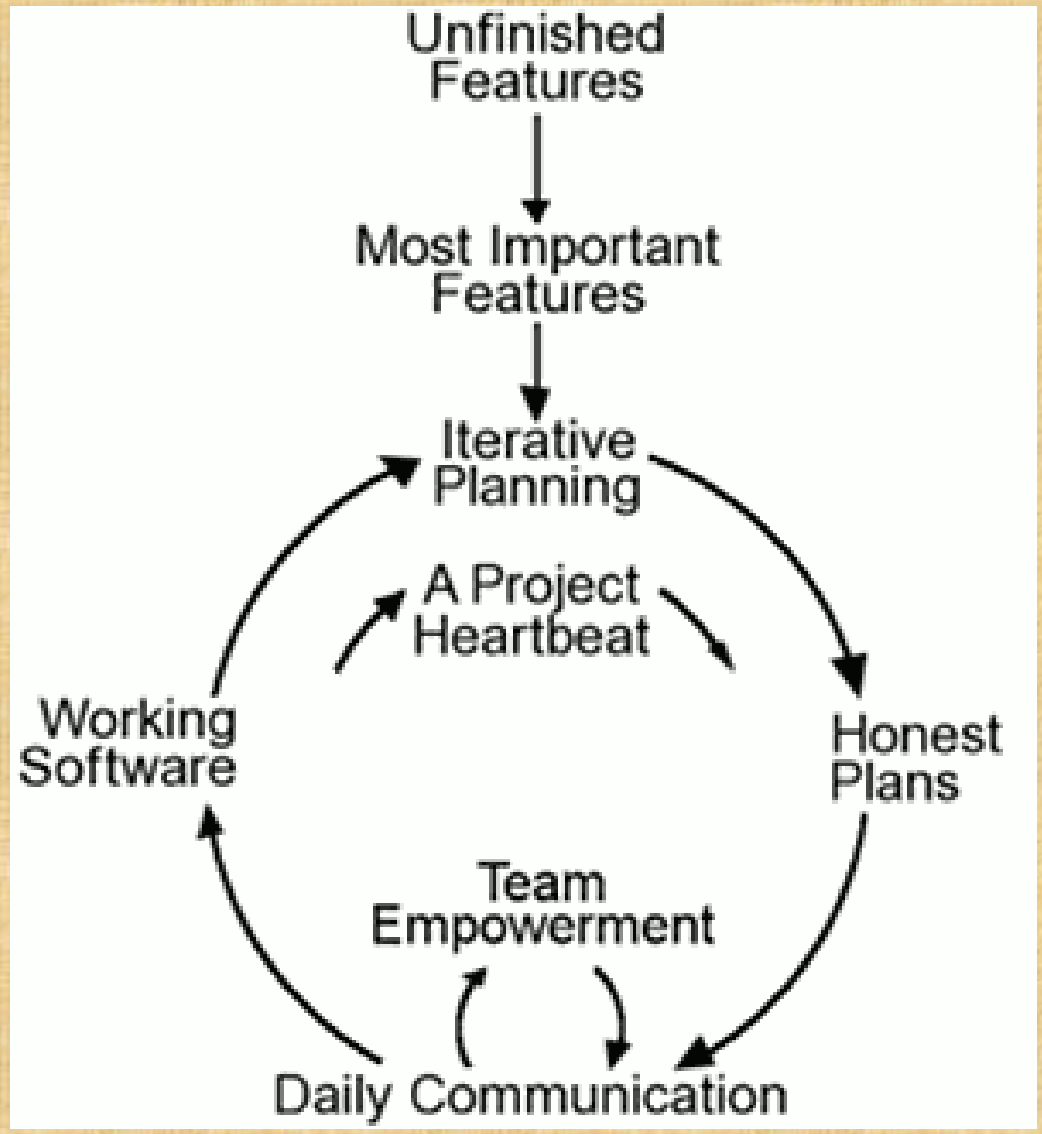
On-site customer. Một khách hàng ngồi với nhóm toàn thời gian.

40-hour weeks. Không ai có thể làm thêm tuần thứ hai liên tiếp. Ngay cả thời gian làm thêm giờ được sử dụng quá thường xuyên cũng là dấu hiệu của những vấn đề lớn cần được giải quyết.

Open workspace. Nhóm làm việc trong một căn phòng lớn với các buồng nhỏ. Lập trình viên theo cặp làm việc trên máy tính được thiết lập tại trung tâm.

Just rules. Khi là thành viên của nhóm Extreme, bạn đăng ký để tuân theo các quy tắc. Nhưng chúng chỉ là các quy tắc. Nhóm có thể thay đổi các quy tắc bất kỳ lúc nào miễn là họ đồng ý về cách họ sẽ đánh giá tác động của thay đổi.

XP – Daily Communication is Key



Extreme Programming (XP)

Để XP thành công, cần có kiến thức chuyên môn quan trọng.

- Xây dựng câu chuyện người dùng - Một câu chuyện người dùng mô tả các vấn đề cần giải quyết của hệ thống đang được xây dựng. Những câu chuyện này phải được viết bởi người dùng và phải dài khoảng ba câu. Câu chuyện của người dùng không mô tả một giải pháp, sử dụng ngôn ngữ kỹ thuật hoặc chứa các yêu cầu truyền thống.
- Chuyển câu chuyện thành mã - Vì câu chuyện của người dùng ngắn và hơi mơ hồ, XP sẽ chỉ hoạt động nếu đại diện khách hàng có mặt để xem xét và phê duyệt việc triển khai câu chuyện của người dùng.
- Biến các câu chuyện thành mã thử nghiệm - Kiểm thử đơn vị là trọng tâm của XP, trong đó hai điểm xoay quanh các chiến lược kiểm thử thông thường giúp kiểm tra hiệu quả hơn nhiều: Các lập trình viên viết các bài kiểm tra của riêng họ và họ viết các bài kiểm tra này trước khi viết mã.
- Thiết kế phát triển - Không được phá vỡ các thử nghiệm hiện có và cũng phải hỗ trợ phát triển gia tăng có thể mở rộng

The pros of XP

- Nếu được hoàn thành tốt, XP cải thiện tinh thần đồng đội.
- Nó xây dựng năng lực thực sự ở tất cả các thành viên trong nhóm.
- Nó làm cho một ngày làm việc thú vị và trung thực.
- Nó đưa mọi người ra khỏi khối của họ và nói chuyện với nhau.
- TDD **(Test-driven development)** dạy các nhà phát triển về cách viết mã chất lượng và cách cải thiện quan niệm của họ về thiết kế; nó giúp họ cải thiện ước tính. Nó cải thiện hồ sơ của các nhà phát triển.
- Nó cung cấp cho quản lý nhiều công cụ, bao gồm khả năng dự đoán, tính linh hoạt của các nguồn lực, tính nhất quán và khả năng hiển thị về những gì thực sự đang diễn ra.
- Nó cung cấp cho khách hàng khả năng xem liệu một công ty có thể thực hiện những lời hứa của mình hay không.
- Bạn không dành nhiều thời gian cho những cuộc họp ngu ngốc, lãng phí và bạn không tạo ra nhiều tài liệu vô ích.

The cons of XP

- Thiết kế trở nên tiềm ẩn thay vì rõ ràng
- Dựa vào thiết kế là rủi ro
- Rất khó để viết các tests tốt
- Lặp lại quá thường xuyên có thể làm giảm chất lượng
- Để làm tốt, bạn cần phải làm thường xuyên - vì vậy rất khó để giới thiệu thành công