|  |  |  |
| --- | --- | --- |
|  | **Traditional** | **Agile** |
| Requirement | * Assume that requirements can be known at the start of the project * Requirements difficult to define up front. * Users tend to define as many requirementsthat they can think of. | Assume that requirements will emerge |
| Requirements are documented as “user stories” (a backlog of product features) |
| Lack of long range requirements means minimal long range planning |
| Emerging requirements Requires management to delegate decision-making authority to the Scrum team. |
| Customer Meeting | Product Owner/User involvement with development team is minimal after requirements are approved | * Product Owner/User works closely with development team on a daily basis * Guaranteed to meet customer needs (Product Owner determines the user stories to implement each sprint) |
| Upfront Planning & Schedule | Schedule, budget, architecture and design for entire project can be created “up front” given requirements | Schedule, budget, architecture and design for project evolve as requirements emerge |
| Pre- and post-sprint planning & acceptance activities |
| Short (<30 minutes) daily standup meetings to communicate status & problems |
| Change and Rework | Change managed formally. Rework is bad | Change is managed informally. Rework (refactoring) is good |
| Documentation | Lots of documentation  Can produce excessive documentation | Minimal documentation. Requirements defined as user stories |
| Team and Developers | Supports technically weak staff | Teams are self organizing  Development team is trusted  Requires more senior developers  Some workers are not comfortable with the responsibility Scrum enables |
| Process | An over reliance on processes (that may not be working well or followed) |  |
| Release | Few visible signs of progress until the end | Product built so you always have a working system and can “release” at any time. |
| Each sprint produces a potentially shippable product |
| Clycles | Stable product definition  Limitation. | Implementation occurs in short ‘sprints’.  Numberous clycles |
| Risk | Unknow Risks, Major Impact | Well understood risk, minor impact |
|  | Well-understood technologies |  |
|  | Quality constraints stronger than cost & schedule |  |
|  | Traditional is not flexible |  |

****

**Product Change**

• Traditional: Needs to be managed

• Agile: Natural part of meeting customer needs. Only need to measure # of stories

 **Rework and Refactoring**

• Traditional: Rework is bad; should be eliminated

• Agile: Refactoring is a natural part of meeting customer needs. No need to measure

**Quality**

• Traditional: Process & product quality

• Agile: Emphasis on product quality

**Examples:**

Are we building the right product?

• Traditional: Managing Requirements

• Agile: Customer Acceptance after each sprint

How long will the project take?

• Traditional: Big plan up front & earned value

• Agile: Velocity & Burn Down Charts

How much will the project cost?

• Traditional: Big plan up front & earned value

• Agile: Developers are assigned 100% to project, so project cost is # of developers \* # sprints

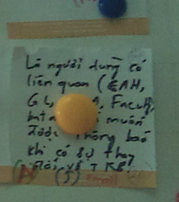
User Story là một bản tóm tắt nhu cầu người dùng nhằm để thể hiện nhu cầu người dùng. User story do khách hàng, hoặc đại diện của khách hàng (Huy gọi là Product Owner) người thực sự hiểu nghiệp vụ và nắm bắt được chính xác yêu cầu của mình đối với nhóm phát triển. Các User Story rất đơn giản, viết trên các thẻ index loại nhỏ hoặc các miếng giấy dán sticky note. Có nhóm dán các User Story này lên bảng, chính là các task board hay các Product Backlog; có nhóm kẹp lại và cất giữ ở nơi phù hợp để nhóm có thể đem ra đọc và thảo luận dễ dàng. 🡪 **Điểm khác biệt với Req**

Các nhà thực hành agile thường sử dụng một khuôn mẫu nhất định để viết User Story. Một trong các mẫu ấy là:

Là *<người dùng cụ thể \ vai trò>* , tôi muốn *<làm gì đó>* để *<phục vụ mục đích nào đó>.*

Ví dụ: *Là quản trị của diễn đàn, tôi muốn xóa một người dùng phạm quy nghiêm trọng để tránh gây gại cho diễn đàn.*

Trong cách thể hiện trên, User Story cho biết rõ chức năng cần phát triển là *“xóa một tài khoản người dùng”*khi có vi phạm nội quy diễn đàn; người dùng chức năng ấy là “quản trị diễn đàn“ chứ không phải một người dùng bất kì; mục đích của việc thực hiện chức năng “xóa” kia là để đảm bảo nội quy cho diễn đàn. Với cách viết đơn giản mà đầy đủ như vậy, thành viên nhóm phát triển dễ dàng nắm bắt được yêu cầu của khách hàng, nghiệp vụ cũng như mục đích của việc làm ra chức năng nào đó, tức giá trị đích thực của chức năng cần làm. Hơn thế, cách viết đơn giản trên một tấm thẻ nhỏ dễ dàng truyền tay để đọc, sẽ giúp nhóm nhanh chóng đi đến thống nhất cách hiểu với khách hàng, tạo điều kiện để phát triển đúng cái khách hàng mong muốn.



Như vậy, cách thể hiện user story như trên cho phép ta biết được **yêu cầu** của khách hàng là gì, **ai** sẽ là người sử dụng chức năng đó, nhằm **mục đích**gì. Đây là các thông tin cơ bản nhất mà nhóm phát triển cần biết để có thể thực hiện công việc của mình.

Không chỉ phục vụ cho việc thể hiện chính xác nhu cầu của người dùng, User Story còn là công cụ để triển khai việc kiểm thử chấp nhận (acceptance test). Khi đó, User Story có thể thêm điều kiện kiểm thử chấp nhận Ví dụ: Là quản trị của diễn đàn, tôi muốn xóa một người dùng phạm quy để tránh di hại về sau cho diễn đàn. *Sau khi xóa xong, người dùng không thể truy xuất hệ thống sử dụng tài khoản đã xóa, các bài viết của người này trên diễn đàn cũng bị hủy khỏi diễn đàn*.

Cách viết này giúp người dùng hoặc một thành viên bất kì của nhóm phát triển có thể kiểm tra xem chức năng làm ra có thỏa mãn người dùng hay không 🡪 **Điểm khác biệt**. Tuy nhiên, cách viết này cũng khiến cho US dài dòng, nhất là trong một số nghiệp vụ phức tạp. Lúc này ta có thể tận dụng mặt sau của thẻ để thể hiện các quy tắc nghiệp vụ hay điều kiện kiểm thử chấp nhận.

Khách hàng (hoặc Product Owner) có thể viết user story, nhưng sẽ tốt hơn là nhóm phát triển cộng tác với khách hàng trong buổi làm việc User Story Writing Workshop tập trung. Khi đó, bên cạnh việc viết ra các User Story, cả nhóm có điều kiện trao đổi, tìm hiểu sâu về các yêu cầu của hệ thống, giúp cho quá trình phát triển sau này.

Công việc phát triển phần mềm thường bắt đầu từ việc hiểu rõ khách hàng thật sự cần gì. User Story và các kĩ thuật liên quan có thể giúp nhóm phát triển làm được điều đó một cách tự nhiên, nhẹ nhàng mà vẫn đảm bảo tính hiệu quả cao. Hãy bắt đầu với agile software development với việc thể hiện toàn bộ các customer requirement bằng User Story!