**Tarea Módulo 1 - Clase 10 –**

Formas de trabajo de los equipos de testing enla actualidad

En la práctica la mayoría de las empresas tienen bien diferenciado el rol de developer y el rol de tester. Es muy común que los testers no sepan programar y que los developers no estén dispuestos a hacer el trabajo de testing. Son pocos los casos donde se encuentra buena disposición para hacer ambas cosas.

Estas son distintas estrategias que en equipos de desarrollo de software abordando el problema:

1. *Ciclo de Dos Sprints (estilo Cascada)*

Durante el primer Sprint el equipo de desarrollo está programando la funcionalidad mientras el equipo de testing está escribiendo el plan de testing -casos de prueba, test cases-.

Durante el segundo Sprint, el equipo de desarrollo está programando las funcionalidades del nuevo Sprint mientras el equipo de testing está testeando lo que fue programado en el Sprint anterior. Al mismo tiempo debe preparar los test cases para el trabajo del Sprint siguiente.

El equipo de desarrollo, además de programar nuevas funcionalidades tiene que arreglar los defectos que encuentra el equipo de testing del trabajo del Sprint anterior.

Este método es anti-Scrum, ya que el Sprint no termina con software terminado (ni potencialmente entregable). De hecho, toma dos Sprints completar un solo Sprint Backlog. (se duplica el Cycle Time).

Es más común encontrar esta estrategia en equipos donde testing es un departamento separado, es decir, no es parte del mismo equipo (también, una práctica no recomendada)

El diseño de casos de prueba es una forma de completar la especificación del requerimiento y se terminan definiendo detalles al mismo tiempo (o incluso a posteriori) de la programación. Esto genera re-trabajo.

*2) Un Sprint (Estilo Cascada)*

Siendo más estrictos y con el objetivo de tener software funcionando en cada iteración, el equipo de desarrollo intenta comprimir el enfoque anterior dentro de los límites de un Sprint: primero programa la funcionalidad y luego testing la verifique todo dentro del mismo Sprint.

Como consecuencia, los primeros días el tester no está aplicado a la tarea de Testing. Hasta tanto no esté programada la primera funcionalidad.

También implica que los desarrolladores planifiquen menos trabajo para el Sprint, ya que deben finalizar su trabajo de programación días antes para dar tiempo a testing a verificar y terminar dentro del Sprint. Por lo tanto, los últimos días los developers no pueden desarrollar funcionalidad del Sprint backlog.

El equipo también puede estar adelantando trabajo del Sprint siguiente. Programando funcionalidad que entraría en el siguiente Sprint.

Esto deja tiempo ocioso para ambos roles.

Por supuesto que es muy fácil encontrar tareas en las que mantenerse ocupados. Tareas que son útiles y agregan valor, como hacer test cases, automatización, investigación, bug fixing, refactors, etc. Pero este tiempo está dado por la limitación anterior y no como un objetivo en sí mismo. Con lo cual, si lo más valioso fuese desarrollar funcionalidades, el equipo no lo podría hacer.

Es una práctica superior al caso anterior, pero todavía poco ágil.

*3) El enfoque ágil*

Una forma de agilizar el proceso consiste en tomar porciones de funcionalidades más pequeñas. Así se minimizan los tiempos no productivos. Para lo cual se hace un trabajo previo de refinamiento y slicing.

Esto no soluciona por completo el problema: Hay funcionalidades que no son tan pequeñas. En un Sprint de dos semanas, es my factible que los tres primeros días no haya ninguna funcionalidad lista para testear. De igual manera los desarrolladores no puedan continuar desarrollando funcionalidad de producto los últimos dos días para que Testing pueda completar sus tareas a tiempo y solucionar los defectos encontrados.

Carry Over

Para optimizar el tiempo de ambos roles, el equipo intenta maximizar la cantidad de trabajo comprometido en el Sprint y esperar poder terminar todo dentro del mismo (Wishful Thinking)

Si no se llega a completar un requerimiento este “pasa al siguiente Sprint” (“Carry Over” o “Roll Over”). Como “solo queda el testing” los equipos pueden re-estimar el esfuerzo restante.

Esta práctica impacta en el resultado del Sprint. No se completa al 100%, sino que quede un margen de Carry Over al siguiente Sprint.

Cuando Testing encuentra más bugs de los esperados, pone el riesgo el Sprint siguiente.

*4) Automatización de Casos de Prueba (la forma recomendada)*

El problema se resuelve con un robusto enfoque en automatización de casos de prueba. Mientras el developer desarrolla la funcionalidad el tester desarrolla casos de prueba automatizados que verifican que la funcionalidad haga lo que se espera.

La ejecución del caso de prueba es muy rápida y se puede ejecutar repetidas veces sin consumir tiempo de las personas, además de que puede ser ejecutada por el Developer o ser ejecutada automáticamente al momento del commit de código.

Aquí es donde las prácticas de ATTD (Acceptance Test-Driven Development), TDD (Test-Driven Development) se ven más aplicadas, como así también el sentido de responsabilidad compartida por la calidad y por todas las partes del proceso de desarrollo.