Etapa 3: Desarrollo de los incrementos

7.3 - Etapa 3: Desarrollo del Sprint

Esta etapa contiene 2 prácticas que se realizan durante el desarrollo de los incrementos del producto. Ambas prácticas son iterativas y se realizan en todos los Sprints del proyecto.

Como se aprecia en el gráfico a continuación, el Scrum Diario es una práctica que se realiza en forma paralela al desarrollo de los entregables del Sprint.

dibujo

ID	Práctica	Rol vs Nivel de Involucramiento
	Etapa 3: desarrollo del sprint	
10	7.3.1 - Desarrollar los entregables (10)	Product Owner: Bajo - Scrum Master: Medio - Equipo de desarrollo: Alto
11	7.3.2 - Scrum diario (11)	Product Owner: Nulo - Scrum Master: Alto - Equipo de desarrollo: Alto

7.3.1 - Desarrollar los entregables (10)

El objetivo de esta actividad es el desarrollo de los entregables del Sprint, los manuales o documentación relacionada considerando siempre la definición de “terminado”.

Nota: En conjunto con esta práctica, se ejecutan los Scrums Diarios, y Las pruebas de los entregables.

7.3.1.1 – El ciclo de desarrollo para proyectos de software

Para el desarrollo de proyectos de software, normalmente se sigue el flujo que se muestra en el gráfico a continuación:

dibujo

7.3.1.1.1 – Las pruebas

Las Pruebas son una forma de comprobar el correcto funcionamiento del producto. Por lo general existen varios tipos de pruebas, siendo los principales:

- Pruebas unitarias: Son escritas por el mismo desarrollador siguiendo los criterios de aceptación de la historia de usuario. Estas pruebas suelen realizarse de manera automatizada.
- Pruebas de colegas: Suelen ser manuales, en las cuales el colega revisan no solo el funcionamiento, sino la calidad del código, y compara el resultado vs prototipos.
- Pruebas de integración: Suelen ser automatizadas. Éstas se ejecutan en cada integración y antes de cada entrega.

También se distinguen 2 tipos de ejecución de las pruebas:

- Pruebas manuales.
- Pruebas automatizadas.

Las 2 posibles salidas de una prueba son:

- Aprobada: Se cumple con todas las expectativas y/o criterios de aceptación de las historias de usuario que se están probando.
- Rechazada: NO se cumple con todas las expectativas y/o criterios de aceptación de las historias de usuario que se están probando. Cuando se rechaza una historia de usuario durante las pruebas, se debe reportar el error en un “Log de Errores”, normalmente dentro del Product Backlog.

Reporte de errores Es importante tener en cuenta que los errores deben pasar por un proceso de estimación igual que las Historias de Usuario (se planifica su resolución en los diferentes Sprints del proyecto).

Cuando se reporten los errores, se debe tener en cuenta:

- Evitar escribir declaraciones vagas o suposiciones no probadas.
- Poner títulos genéricos a los errores.
- Se deberían categorizar los errores, ej: Errores en producción, Errores de colegas, Errores en la revisión, etc.
- Deberían describirse como pasos de reproducción sencillos y repetibles para que la persona que corregirá el error pueda seguir la secuencia.

7.3.1.1.2 – Integración continua

La Integración Continua es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica,

tras lo cual se ejecutan versiones y pruebas automáticas para asegurar la integridad del código.

Para facilitar la integración continua:

- Se debe tener un repositorio único para el código fuente.
- Se debe garantizar que el código se actualiza constantemente tanto en el repositorio, como en el ambiente de trabajo de cada desarrollador.
- Siempre ejecutar pruebas automatizadas para garantizar que cuando se mezcla el código, no se pierden o dañan las funcionalidades existentes.

7.3.1.1.3 – Documentación

La documentación es un aspecto vital en los proyectos de software, esto permitirá el posterior entendimiento del código fuente y así garantizar la propiedad colectiva del producto. Para que la documentación sea efectiva, se debe garantizar que:

- Sea fácil de redactar y entender.
- Debe estar compartida con todos.
- Se actualiza diariamente.
- Debe tener un índice o búsqueda.
- Debe estar organizada por módulos o componentes.
- Debe ser alimentada por todo el equipo.

7.3.1.1.4 – Refactorización

El objetivo de esta técnica es mejorar el mantenimiento del código existente y hacerlo más simple, más conciso y más flexible. Refactorizar significa mejorar el diseño del código actual, sin cambiar el comportamiento del código. Algunas de las ventajas de la refactorización son:

- Mejorar la facilidad de comprensión del código.
- Mejorar su estructura y diseño.
- Eliminar código muerto.
- Facilitar el mantenimiento en el futuro.

Algunos de los momentos clave para realizar la refactorización, son:

- Después de pasar una prueba automatizada.
- Cuando agregar una nueva característica es bastante complicado.
- Cuando mejora el conocimiento o experiencia del equipo de desarrollo.
- Cuando el equipo dedica bastante tiempo a entender el detalle del código.

Algunos ejemplos de refactorización son:

- Organizar el repositorio de documentos/archivos.
- Separación de funciones.
- Renombrar variables.

- Simplificación de las interfaces.

7.3.2 - Scrum diario (11)

Para más detalles ver la Sección 3.3 - Daily Scrum (Scrum Diario)

7.3.2.1 - Seguimiento del Progreso del Sprint

En cualquier momento durante un Sprint es posible conocer el progreso del Sprint sumando el trabajo restante total en los elementos del Sprint Backlog. El Equipo de Desarrollo hace seguimiento de este trabajo restante total al menos en cada Scrum Diario para proyectar la posibilidad de conseguir el objetivo del Sprint. Haciendo seguimiento del trabajo restante a lo largo del Sprint el Equipo de Desarrollo puede gestionar su progreso.