CHAPTER 5

Planificación en Scrum

5.1 Introducción

La planificación efectiva es esencial para el éxito de cualquier tarea o proyecto, y el desarrollo de software no es una excepción. En Scrum, un marco de trabajo ágil ampliamente utilizado, la planificación es un proceso continuo y adaptativo que abarca varios niveles, desde la iteración hasta el *release* y más allá. En este capítulo, exploraremos cómo se planifica en Scrum, considerando analogías sugerentes y estrategias clave para abordar los desafíos inherentes a la planificación en el desarrollo de software.

Imagina el atraco meticulosamente planeado en la famosa serie de TV "Casa de Papel". Un grupo liderado por un "profesor" superinteligente ejecuta un robo de gran complejidad, que requirió años de planificación meticulosa. De manera similar, cualquier tarea de envergadura necesita ser minuciosamente planificada para alcanzar el éxito. Este principio se aplica con fuerza al desarrollo de software, donde la incertidumbre y la evolución constante son la norma.

En proyectos de otros ámbitos, las "Cartas Gantt" suelen estructurar la planificación, pero en el desarrollo de software es difícil establecer un plan rígido desde el principio. Aquí es donde entran en juego los modelos de proceso ágiles. Un ejemplo elocuente es el esquí en terreno desconocido: los esquiadores se enfrentan a un terreno cambiante y se adaptan en tiempo real. De manera análoga, los equipos ágiles ajustan su enfoque según las circunstancias cambiantes del proyecto.

5.2 Planeación de la Iteración (Sprint)

La iteración, o *sprint*, es una unidad fundamental de tiempo en Scrum. Durante la planeación del sprint, el equipo selecciona historias de usuario de acuerdo con la prioridad y la capacidad del equipo. Las historias de usuario, expresadas en Puntos de Historia, son estimaciones relativas de complejidad. El equipo equilibra la cantidad de trabajo con la capacidad, asegurando un compromiso realista.

Puntos de Historia

Los Puntos de Historia constituyen una técnica valiosa para evaluar la complejidad relativa de las historias de usuario. Esta estimación se efectúa en una unidad abstracta, lo que facilita la comparación del esfuerzo involucrado en diferentes historias. Una estrategia efectiva para estimar mediante puntos de historia es comenzar seleccionando una historia que la mayoría de los miembros del equipo comprenda en términos de su complejidad de desarrollo. Para esta historia de referencia, el equipo asigna un número específico, conocido como "puntos de historia", que puede ser cualquier valor. Luego, utilizando esta referencia, el equipo atribuye puntos de historia a las demás historias en función de su complejidad relativa. Por ejemplo, si una historia se considera más desafiante que la historia de referencia, podría asignársele más puntos, mientras que si se la percibe como menos compleja, podría recibir menos puntos.

Estimación con Scrum Poker

La destreza esencial de priorizar historias de usuario se potencia a través de Scrum Poker, una técnica colaborativa que permite al equipo de desarrollo y al Product Owner estimar la complejidad de las historias de manera conjunta. En una sesión de Scrum Poker, cada miembro del equipo elige una carta que representa su percepción de la cantidad de esfuerzo necesario para completar una historia en particular. Luego, se revelan las cartas y se abordan las diferencias en las estimaciones. Esta colaboración en la estimación a través de Scrum Poker facilita la convergencia hacia un consenso acerca de la complejidad de la historia, permitiendo al equipo asignar puntos de historia de manera más precisa y colectiva.

Velocidad de Desarrollo

La velocidad de desarrollo en el contexto de Scrum se refiere a la medida del trabajo que un equipo ágil puede completar en un sprint. Representa la cantidad de puntos de historia, tareas o funcionalidades que el equipo puede entregar de manera exitosa y consistente en un período de tiempo determinado, generalmente en una iteración. La velocidad de desarrollo es una métrica clave que proporciona una comprensión cuantitativa de la capacidad

y eficiencia del equipo, lo que permite una planificación más precisa de futuros sprints y entregas. A medida que el equipo avanza en su colaboración y optimiza sus procesos, la velocidad de desarrollo puede ajustarse y utilizarse como referencia para estimar la cantidad de trabajo que se puede abordar en ciclos futuros.

El cálculo de la velocidad de desarrollo en Scrum se basa en la recopilación de datos de sprints anteriores. Para determinarla, se suman los puntos de historia de las historias de usuario que se completaron en cada uno de los sprints previos. La velocidad de desarrollo se obtiene al calcular el promedio de estos puntos de historia completados en los sprints anteriores. Es fundamental tener en cuenta que este cálculo únicamente considera los puntos de historia que se completaron al 100% durante los sprints anteriores, excluyendo aquellos que quedaron parcialmente resueltos o no se abordaron por completo. Esta métrica proporciona una estimación valiosa de la capacidad del equipo para realizar trabajo en un sprint y es una guía fundamental para la selección de historias y la planificación del próximo sprint.

Selección de Historias de Usuario

La selección de historias de usuario para un sprint conlleva una evaluación integral, considerando tanto la prioridad de las historias como sus estimaciones en Puntos de Historia. Este proceso requiere una combinación de factores clave: la velocidad de desarrollo del equipo, las estimaciones en puntos de historia de las historias en el backlog y la priorización basada en el valor para el cliente. Con estos elementos en mente, el equipo de desarrollo y el Product Owner colaboran para determinar qué historias son viables y estratégicas para ser abordadas en el próximo sprint. Idealmente, se seleccionan historias de menor complejidad (menos Puntos de Historia) y alta prioridad, asegurando un enfoque en aquellas que generan un mayor impacto para el cliente y pueden ser entregadas con éxito en el período del sprint. La suma total de los puntos de historia de las historias seleccionadas debe aproximarse a la velocidad de desarrollo del equipo, lo que garantiza que el conjunto de historias sea manejable y que el equipo tenga la capacidad de completarlas en el sprint. En última instancia, esta meticulosa selección se traduce en un plan de trabajo realista y enfocado, optimizando la probabilidad de éxito en la entrega y el cumplimiento de los objetivos establecidos.

Ejemplo

Consideremos las siguientes historias de usuario, ordenadas por prioridad en el backlog:

- Historia A: 4 puntos de historia
- Historia B: 8 puntos de historia
- Historia C: 2 puntos de historia
- Historia D: 12 puntos de historia

- Historia E: 4 puntos de historia
- Historia F: 16 puntos de historia

Si nuestra velocidad de desarrollo es de 12 puntos, entonces en el siguiente sprint sería conveniente abordar las historias A y B. Sin embargo, la historia C no podría ser incluida, dado que nuestra capacidad es de 12 puntos por sprint.

5.3 Desglose Táctico en Tareas

Las historias de usuario se desglosan en tareas concretas durante la reunión de planificación del sprint. Cada tarea representa un paso hacia la finalización de una historia. Estas tareas se estiman en horas o días y se asignan a miembros del equipo. Este desglose permite un monitoreo detallado del progreso y proporciona una estructura clara para la ejecución. A continuacion, se muestra un desglose de tareas y sus estimaciones en horas para la historia de usuario: “Crear una página de inicio de sesión para los usuarios”.

Tareas:

- Diseñar la interfaz de la página de inicio de sesión.
- Configurar el controlador de sesiones en Ruby on Rails.
- Crear una vista para el formulario de inicio de sesión.
- Implementar la lógica de autenticación en el controlador.
- Establecer las rutas necesarias para la página de inicio de sesión.
- Agregar validaciones de entrada y manejo de errores.
- Crear pruebas unitarias para verificar la funcionalidad del inicio de sesión.
- Integrar estilos CSS para mejorar la apariencia visual.
- Configurar las sesiones de usuario y el almacenamiento seguro de contraseñas.
- Realizar pruebas de integración para asegurarse de que el flujo de inicio de sesión funcione correctamente.

Estimaciones:

- Diseño de la interfaz: 4 horas.
- Configuración del controlador de sesiones: 2 horas.
- Creación de la vista del formulario: 3 horas.
- Implementación de la lógica de autenticación: 6 horas.
- Establecimiento de rutas: 1 hora.
- Validaciones y manejo de errores: 3 horas.
- Pruebas unitarias: 5 horas.
- Integración de estilos CSS: 2 horas.
- Configuración de sesiones y seguridad de contraseñas: 4 horas.
- Pruebas de integración: 4 horas.

Este desglose de tareas proporciona una visión detallada de los pasos necesarios para completar la historia de usuario y permite al equipo estimar el esfuerzo requerido para cada tarea. Cada tarea puede asignarse a un miembro del equipo, lo que facilita el seguimiento del progreso y la ejecución eficiente del trabajo.

5.4 Planificación a Nivel de *Release*

La planificación a nivel de *release* es crucial para garantizar la visión a largo plazo. Implica estimar las iteraciones necesarias y las funcionalidades que compondrán el entregable final. Aquí, la colaboración entre el equipo de desarrollo y el Product Owner es vital. A pesar de la adaptabilidad inherente, se debe mantener una visión clara de las funcionalidades esenciales que conformarán el *release*.

Un *release* implica la entrega de un conjunto de historias de usuario. Por lo general, estas historias deben desarrollarse a lo largo de varios sprints. Por ejemplo, un *release* puede abarcar 4 sprints, en los cuales se trabajarán en 10 historias de usuario. Si cada sprint tiene una duración de 2 semanas, el *release* se completará en un período de 8 semanas.

5.5 Estrategias de Ajuste en la Planificación

omo ya hemos dicho antes, por diversas razones, la gente que hace esta planeación se ve sujeta a restricciones que muchas veces no están basadas en estimaciones reales o evidencia técnica, sino a factores comerciales u otro tipo de restricciones: debemos salir con esto en septiembre, tienes 3 meses para lograrlo, etc.

Debido a lo anteriorm es muy difícil que se logre construir el software con todas las prestaciones solicitadas en la fecha inicialmente prevista y se hace necesario ajustar a medio camino. Hay varias formas de ajustar, cada una con sus ventajas y desventajas.

Ajuste por Tiempo

Implica renegociar la fecha de entrega (extensión). Digamos que se planificaron 8 sprints de 2 semanas en que se iban a implementar 40 relatos de usuario. Al cerrar el sprint 4, vemos solo se han completado 16. A ese ritmo es probable que los 24 siguientes nos tome 6 sprints y no 4 por lo que negociamos un alargue de un mes adicional. Este tipo de ajuste, que suele parecerles muy lógico a los desarrolladores, generalmente tiene una muy mala recepción porque “no se cumple con lo prometido”. A veces de verdad hay una necesidad de tener el producto en una fecha dada y el atrasarse un mes no es una opción.

Ajuste por Alcance

Implica renegociar las funcionalidades que serán incluidas en el *release*. Tomando el mismo ejemplo anterior, al cerrar el sprint 4 vemos que a ese ritmo lograremos completar solo 32 de los 40 relatos. Dado que el desarrollo se ha llevado a cabo priorizando las funcionalidades más importantes para el negocio, esos 32 relatos deben tenerle 90% o más de las cosas que realmente le importan al cliente. Entonces ofrecemos entregar el producto en la fecha acordada, pero con un pequeño porcentaje de funcionalidades que quedarán para un *release* futuro del producto. Este tipo de ajuste es por lo general mucho mejor recibido. De hecho, si se ha priorizado correctamente, es posible que al cliente ni siquiera le importe demasiado lo que quedará fuera del *release*.

Ajuste por Calidad

Este es el tipo de ajuste que **NUNCA** se debería hacer, y sin embargo, el que más se hace. Tomando el mismo ejemplo anterior, el equipo de desarrollo, al darse cuenta del atraso al final del sprint 4 decide que "hay que apurar". Esto suele traer consigo el sacrificio de cosas que permiten asegurar la calidad del producto como una reducción en *testing*, eliminar las revisiones de código, eliminar la programación en pares, eliminar las revisiones de producto y retrospectiva, etc. Finalmente, se entrega el producto a tiempo con todas las funcionalidades prometidas, pero con dos efectos laterales indeseados importantes: un compromiso serio de la calidad (puede tener serias consecuencias más adelante) y un desgaste y frustración extrema del equipo de desarrollo.

En estas situaciones se habla de haber contraído una deuda técnica. Al igual que una deuda financiera, una deuda técnica implica un compromiso a pagar más adelante lo que no puedo pagar hoy, pero con los intereses acumulados. Por ejemplo, un diseño apresurado puede significar un ahorro de tiempo hoy que tendrá que ser pagado más adelante cuando nos demos cuenta del error. Un *bug* no detectado puede posteriormente significar muchas horas de *debugging* para corregir.