Universidad del Chubut | Tecnicatura Universitaria en Desarrollo de Software

Actividad práctica Grupal - Modelos

Objetivo:

El objetivo de esta actividad es que los grupos conformados en la asignatura Programación Web II apliquen los conceptos de modelado en Django teniendo en cuenta la metodología **SCRUM** para gestionar proyectos (ver Anexo). Los grupos deben construir los modelos en Django basados en las descripciones proporcionadas, generar un set de datos de prueba utilizando las herramientas que prefieran, y proponer una mejora en los modelos. Finalmente, deberán realizar consultas utilizando las relaciones y agregaciones vistas en clase para demostrar el funcionamiento de los modelos.

Instrucciones:

1. **Creación de los Modelos en Django:**
   - Según la descripción de los modelos, deberán crear las clases `Tarea`, `Épica` y `Sprint` en el archivo `models.py` de tu proyecto y aplicación Django.
   - Define las relaciones correctas entre las clases, usando `ForeignKey` y `ManyToManyField` cuando corresponda.
2. **Propuesta de Mejora en los Modelos**:
   - Realiza una propuesta de mejora en el modelado actual, considerando las buenas prácticas comentadas y detalladas en los apuntes. Justifica tu propuesta y agrégala al modelo en Django.
3. **Generación de Datos de Prueba**:
   - Crea un conjunto de datos de prueba para los modelos, utilizando una de las siguientes herramientas:
     1. Scripts en Python (Django shell).
     2. Archivo JSON con fixtures.
     3. Migraciones personalizadas.
   - El set de datos debe incluir:
     1. Al menos 10 usuarios (responsables, Scrum Masters y miembros del equipo).
     2. Al menos 30 tareas, distribuidas en diferentes sprints y épicas.
     3. Al menos 3 sprints con equipos de desarrollo y backlog asignado.
     4. Varias dependencias entre tareas y épicas.
4. **Consultas y Agregaciones**:
   - Ejecuta al menos **10 consultas** en el shell de Django o mediante vistas que demuestren el funcionamiento de tu sistema. Utiliza los conceptos de **Consultas y Agregaciones** para extraer información relevante.
   - Las consultas deben incluir:
     1. Obtener todas las tareas asignadas a un usuario específico.
     2. Obtener las tareas completadas dentro de un sprint determinado.
     3. Listar todas las tareas que dependen de una tarea específica.

4. Listar todas las épicas que tienen tareas en progreso.
5. Calcular el número total de tareas por estado (por hacer, en progreso, completada).
6. Obtener la suma de esfuerzo estimado de todas las tareas asociadas a una épica específica.
7. Listar los sprints que tiene un Scrum Master asignado.
8. Obtener el progreso total de una épica en base a las tareas completadas.
9. Obtener el backlog de un sprint específico y sus responsables.
10. Listar todas las tareas que están bloqueadas.

Entrega:

Cada grupo deberá entregar la actividad solicitada mediante correo electrónico indicando los integrantes del grupo. Cada actividad debe contener los cuatro puntos detallados en las instrucciones (Modelos, Mejora, Datos de Prueba, Consultas) en un proyecto Django, también deberá incluir un archivo README.md con la forma de uso. Posteriormente deberán defender su propuesta en un coloquio donde se realizarán preguntas a los integrantes de cada grupo referidas al trabajo realizado.

Fecha de entrega de la actividad: 24 de Octubre de 2024 - 20 hs.

---

1. Descripción de los Modelos

Modelo de Tareas (`Tarea`)

Propósito: Representa una unidad de trabajo o una historia de usuario dentro del proyecto SCRUM. Cada tarea es una actividad específica que el equipo de desarrollo debe completar.

Campos y Descripciones:

- **`titulo`** (`CharField`):
  - **Descripción**: Nombre descriptivo de la tarea.
  - **Restricciones**: Máximo de 200 caracteres.
- **`descripcion`** (`TextField`):
  - **Descripción**: Detalle de lo que implica la tarea.
  - **Restricciones**: No puede ser nulo (`null=False`), pero puede estar vacío (`blank=True`).

- **`criterios_aceptacion`** (`TextField`):
  - **Descripción**: Condiciones que deben cumplirse para considerar la tarea como completada.
  - **Restricciones**: Opcional (`blank=True`, `null=True`).
- **`prioridad`** (`IntegerField`):
  - **Descripción**: Nivel de prioridad de la tarea, donde un valor más alto indica mayor urgencia.
  - **Restricciones**: Valor mínimo de 0 (`prioridad__gte=0`).
- **`estado`** (`CharField`):
  - **Descripción**: Estado actual de la tarea.
  - **Opciones**:
    - `"POR_HACER"`: La tarea aún no ha comenzado.
    - `"EN_PROGRESO"`: La tarea está en desarrollo.
    - `"COMPLETADA"`: La tarea ha sido finalizada.
  - **Restricciones**: Debe ser uno de los valores predefinidos.
- **`esfuerzo_estimado`** (`IntegerField`):
  - **Descripción**: Estimación del esfuerzo requerido para completar la tarea, medida en puntos de historia o horas.
  - **Restricciones**: Valor mínimo de 0 (`esfuerzo_estimado__gte=0`).
- **`responsable`** (`ForeignKey` a `User`):
  - **Descripción**: Usuario asignado para completar la tarea.
  - **Restricciones**: Puede ser nulo (`null=True`). Si el usuario es eliminado, el campo se establece en `NULL` (`on_delete=models.SET_NULL`).
- **`sprint_asignado`** (`ForeignKey` a `Sprint`):
  - **Descripción**: Sprint en el cual está planificada la tarea.
  - **Restricciones**: Opcional (`null=True`, `blank=True`). Si el sprint es eliminado, el campo se establece en `NULL` (`on_delete=models.SET_NULL`).
- **`fecha_creacion`** (`DateTimeField`):
  - **Descripción**: Fecha y hora en que se creó la tarea.
  - **Restricciones**: Se establece automáticamente al crear la instancia (`auto_now_add=True`).
- **`fecha_actualizacion`** (`DateTimeField`):
  - **Descripción**: Fecha y hora de la última actualización de la tarea.
  - **Restricciones**: Se actualiza automáticamente cada vez que se guarda la instancia (`auto_now=True`).
- **`dependencias`** (`ManyToManyField` a `self`):
  - **Descripción**: Relaciones de dependencia con otras tareas. Una tarea puede depender de múltiples tareas y viceversa.
  - **Restricciones**: Relación no simétrica (`symmetrical=False`). Opcional (`blank=True`).

- **`bloqueadores`** (`TextField`):
    - **Descripción**: Descripción de problemas o impedimentos que están bloqueando el progreso de la tarea.
    - **Restricciones**: Opcional (`blank=True`, `null=True`).

Restricciones (`Meta`):

- **`prioridad_no_negativa`**: Asegura que la prioridad sea mayor o igual a 0.
- **`esfuerzo_estimado_no_negativo`**: Asegura que el esfuerzo estimado sea mayor o igual a 0.
- **`estado_valido_tarea`**: Garantiza que el estado de la tarea sea uno de los valores predefinidos (`POR_HACER`, `EN_PROGRESO`, `COMPLETADA`).

Modelo de Épicas (`Epica`)

Propósito: Agrupa un conjunto de tareas relacionadas que contribuyen a un objetivo mayor dentro del proyecto SCRUM. Las épicas representan grandes funcionalidades o áreas de trabajo que se descomponen en tareas más pequeñas.

Campos y Descripciones:

- **`nombre`** (`CharField`):
    - **Descripción**: Nombre descriptivo de la épica.
    - **Restricciones**: Máximo de 200 caracteres.
- **`descripcion`** (`TextField`):
    - **Descripción**: Detalle de lo que implica la épica.
    - **Restricciones**: Opcional (`blank=True`, `null=True`).
- **`criterios_aceptacion`** (`TextField`):
    - **Descripción**: Condiciones que deben cumplirse para considerar la épica como completada.
    - **Restricciones**: Opcional (`blank=True`, `null=True`).
- **`estado`** (`CharField`):
    - **Descripción**: Estado actual de la épica.
    - **Opciones**:
        - `"POR_HACER"`: La épica aún no ha comenzado.
        - `"EN_PROGRESO"`: La épica está en desarrollo.
        - `"COMPLETADA"`: La épica ha sido finalizada.
    - **Restricciones**: Debe ser uno de los valores predefinidos.
- **`responsable`** (`ForeignKey` a `User`):
    - **Descripción**: Usuario responsable de supervisar y gestionar la épica.

    - **Restricciones**: Puede ser nulo (`null=True`). Si el usuario es eliminado, el campo se establece en `NULL` (`on_delete=models.SET_NULL`).
- **`tareas_asociadas`** (`ManyToManyField` a `Tarea`):
    - **Descripción**: Conjunto de tareas que pertenecen a esta épica.
    - **Restricciones**: Opcional (`blank=True`).
- **`esfuerzo_estimado_total`** (`IntegerField`):
    - **Descripción**: Estimación total del esfuerzo requerido para completar la épica.
    - **Restricciones**: Valor mínimo de 0 (`esfuerzo_estimado_total__gte=0`).
- **`fecha_inicio`** (`DateField`):
    - **Descripción**: Fecha de inicio prevista para la épica.
    - **Restricciones**: Opcional (`null=True`, `blank=True`).
- **`fecha_fin`** (`DateField`):
    - **Descripción**: Fecha de finalización prevista para la épica.
    - **Restricciones**: Opcional (`null=True`, `blank=True`). Debe ser posterior o igual a `fecha_inicio` (`fecha_fin__gte=fecha_inicio`).
- **`progreso`** (`FloatField`):
    - **Descripción**: Porcentaje de completitud de la épica, expresado como un valor decimal entre 0.0 y 1.0.
    - **Restricciones**: Debe estar entre 0 y 1 (`progreso__gte=0`, `progreso__lte=1`).
- **`dependencias`** (`ManyToManyField` a `self`):
    - **Descripción**: Relaciones de dependencia con otras épicas. Una épica puede depender de múltiples épicas y viceversa.
    - **Restricciones**: Relación no simétrica (`symmetrical=False`). Opcional (`blank=True`).

Restricciones (`Meta`):

- **`esfuerzo_total_no_negativo`**: Asegura que el esfuerzo total estimado sea mayor o igual a 0.
- **`progreso_valido`**: Garantiza que el progreso esté entre 0 y 1.
- **`estado_valido_epica`**: Garantiza que el estado de la épica sea uno de los valores predefinidos (`POR_HACER`, `EN_PROGRESO`, `COMPLETADA`).
- **`fecha_fin_posterior_epica`**: Asegura que la fecha de fin sea posterior o igual a la fecha de inicio.

Modelo de Sprints (`Sprint`)

Propósito: Representa una iteración de trabajo dentro de SCRUM, generalmente de 1 a 4 semanas de duración. Cada sprint tiene objetivos específicos y un conjunto de tareas que el equipo de desarrollo se compromete a completar.

Campos y Descripciones:

- **`nombre`** (`CharField`):
  - **Descripción**: Nombre descriptivo del sprint.
  - **Restricciones**: Máximo de 200 caracteres.
- **`objetivo`** (`TextField`):
  - **Descripción**: Objetivo principal del sprint, definiendo lo que se espera lograr.
  - **Restricciones**: Opcional (`blank=True`, `null=True`).
- **`fecha_inicio`** (`DateField`):
  - **Descripción**: Fecha de inicio del sprint.
  - **Restricciones**: Obligatorio.
- **`fecha_fin`** (`DateField`):
  - **Descripción**: Fecha de finalización del sprint.
  - **Restricciones**: Obligatorio. Debe ser posterior o igual a `fecha_inicio` (`fecha_fin__gte=fecha_inicio`).
- **`velocidad`** (`IntegerField`):
  - **Descripción**: Capacidad del equipo de desarrollo en puntos de historia para el sprint.
  - **Restricciones**: Valor mínimo de 0 (`velocidad__gte=0`).
- **`scrum_master`** (`ForeignKey` a `User`):
  - **Descripción**: Usuario que desempeña el rol de Scrum Master para este sprint.
  - **Restricciones**: Puede ser nulo (`null=True`). Si el usuario es eliminado, el campo se establece en `NULL` (`on_delete=models.SET_NULL`).
- **`equipo_desarrollo`** (`ManyToManyField` a `User`):
  - **Descripción**: Conjunto de usuarios que forman parte del equipo de desarrollo para este sprint.
  - **Restricciones**: Opcional (`blank=True`).
- **`backlog_sprint`** (`ManyToManyField` a `Tarea`):
  - **Descripción**: Conjunto de tareas que se abordarán durante este sprint.
  - **Restricciones**: Opcional (`blank=True`).
- **`fecha_creacion`** (`DateTimeField`):
  - **Descripción**: Fecha y hora en que se creó el sprint.
  - **Restricciones**: Se establece automáticamente al crear la instancia (`auto_now_add=True`).
- **`fecha_actualizacion`** (`DateTimeField`):

- **Descripción**: Fecha y hora de la última actualización del sprint.
- **Restricciones**: Se actualiza automáticamente cada vez que se guarda la instancia (`auto_now=True`).

Restricciones (`Meta`):

- `fecha_fin_posterior`: Asegura que la fecha de fin del sprint sea posterior o igual a la fecha de inicio.
- `velocidad_no_negativa`: Garantiza que la velocidad del sprint sea mayor o igual a 0.

---

2. Descripción de la Interacción entre las Clases

Las clases **Tarea**, **Épica** y **Sprint** interactúan entre sí a través de diversas relaciones que permiten una gestión eficaz del trabajo dentro de la metodología SCRUM. A continuación, se detalla cómo se relacionan estas clases:

Relaciones de ForeignKey

- `Tarea.responsable`:
  - **Relación**: Cada tarea está asignada a un usuario específico (`User`), quien es responsable de completarla.
  - **Tipo**: Uno a Muchos (Un usuario puede tener múltiples tareas asignadas, pero cada tarea tiene un único responsable).
- `Tarea.sprint_asignado`:
  - **Relación**: Una tarea puede estar asignada a un sprint específico (`Sprint`).
  - **Tipo**: Uno a Muchos (Un sprint puede tener múltiples tareas, pero cada tarea puede estar asignada a un único sprint o ninguno).
- `Epica.responsable`:
  - **Relación**: Cada épica está supervisada por un usuario específico (`User`).
  - **Tipo**: Uno a Muchos (Un usuario puede ser responsable de múltiples épicas, pero cada épica tiene un único responsable).
- `Sprint.scrum_master`:
  - **Relación**: Cada sprint tiene asignado un Scrum Master (`User`), quien facilita el proceso SCRUM.
  - **Tipo**: Uno a Muchos (Un usuario puede ser Scrum Master de múltiples sprints, pero cada sprint tiene un único Scrum Master o ninguno).

Relaciones de ManyToManyField

- `Epica.tareas_asociadas`:

    - **Relación**: Una épica puede agrupar múltiples tareas, y una tarea puede pertenecer a múltiples épicas.
    - **Tipo**: Muchos a Muchos (Bidireccional).
- `Sprint.equipo_desarrollo`:
    - **Relación**: Un sprint puede tener múltiples miembros en el equipo de desarrollo (`User`), y un usuario puede pertenecer a múltiples sprints.
    - **Tipo**: Muchos a Muchos (Bidireccional).
- `Sprint.backlog_sprint`:
    - **Relación**: Un sprint puede tener múltiples tareas en su backlog, y una tarea puede estar en el backlog de múltiples sprints.
    - **Tipo**: Muchos a Muchos (Bidireccional).
- `Tarea.dependencias`:
    - **Relación**: Una tarea puede depender de múltiples otras tareas, y una tarea puede ser dependida por múltiples tareas.
    - **Tipo**: Muchos a Muchos (No simétrico).
- `Epica.dependencias`:
    - **Relación**: Una épica puede depender de múltiples otras épicas, y una épica puede ser dependida por múltiples épicas.
    - **Tipo**: Muchos a Muchos (No simétrico).

---

3. Interacción entre los Modelos

La interacción entre los modelos **Tarea**, **Épica** y **Sprint** facilita una gestión estructurada y eficiente del trabajo dentro del marco SCRUM. A continuación, se explica cómo se interrelacionan y colaboran estos modelos para cumplir con los objetivos de SCRUM:

Flujo de Trabajo Integrado

1. **Planificación y Asignación de Tareas**:
    - **Product Owner** define las **épicas** que representan grandes funcionalidades o áreas de trabajo.
    - Las **épicas** se descomponen en **tareas** más pequeñas y manejables (User Stories).
    - Cada **tarea** se asigna a un **responsable** (usuario) y, opcionalmente, a un **sprint** específico para su desarrollo.
2. **Organización de Sprints**:
    - Durante la **Planificación del Sprint**, el **equipo de desarrollo** selecciona las **tareas** del **Product Backlog** que se abordarán en el próximo sprint.
    - Las **tareas** seleccionadas se agregan al **Sprint Backlog** del **Sprint** correspondiente.
    - El **Scrum Master** facilita este proceso, asegurando que el sprint tenga un objetivo claro y que el equipo esté preparado para cumplirlo.

3. **Desarrollo y Seguimiento**:
    - Durante el sprint, el **equipo de desarrollo** trabaja en las **tareas** asignadas, actualizando su **estado** según el progreso (por ejemplo, de "POR_HACER" a "EN_PROGRESO").
    - Si una **tarea** encuentra **bloqueadores**, se registran para que el **Scrum Master** los resuelva.
    - Las **épicas** monitorean el progreso general al acumular el avance de las **tareas** asociadas, reflejando un porcentaje de completitud.
4. **Revisión y Retrospectiva**:
    - Al final del sprint, se realiza una **Revisión del Sprint**, donde el equipo presenta el trabajo completado.
    - Se evalúa el **progreso** de las **épicas** basándose en la finalización de sus **tareas** asociadas.
    - En la **Retrospectiva del Sprint**, el equipo reflexiona sobre el proceso y busca mejoras continuas.

Gestión de Dependencias

- **Dependencias entre Tareas**:
    - Algunas **tareas** pueden depender de la finalización de otras. Por ejemplo, la tarea B no puede comenzar hasta que la tarea A esté completada.
    - Esta relación se gestiona mediante el campo `dependencias` en el modelo **Tarea**, permitiendo al equipo identificar y gestionar estas dependencias para evitar bloqueos.
- **Dependencias entre Épicas**:
    - De manera similar, una **épica** puede depender de la finalización de otra. Esto es útil cuando una funcionalidad grande requiere la finalización de otra para poder avanzar.
    - Las dependencias entre épicas se gestionan mediante el campo `dependencias` en el modelo **Epica**.

Asignación y Capacidad del Equipo

- **Velocidad del Sprint**:
    - El campo `velocidad` en el modelo **Sprint** representa la capacidad del equipo de desarrollo para completar **tareas** en términos de puntos de historia.
    - Esta métrica ayuda al equipo a planificar cuántas **tareas** pueden abordar en un sprint dado su rendimiento histórico.
- **Equipo de Desarrollo**:
    - El campo `equipo_desarrollo` en el modelo **Sprint** define qué usuarios forman parte del equipo de desarrollo para ese sprint específico.
    - Esto permite asignar tareas de manera efectiva, aprovechando las habilidades y disponibilidades de los miembros del equipo.

Estado y Progreso

- **Estados de Tareas y Épicas**:
  - Los campos `estado` en ambos modelos indican el progreso actual de las **tareas** y **épicas**.
  - Este seguimiento ayuda al equipo y a los stakeholders a comprender en qué fase se encuentran diferentes partes del proyecto.
- **Progreso de las Épicas**:
  - El campo `progreso` en el modelo **Epica** refleja el avance global basado en la finalización de las **tareas** asociadas.
  - Esto proporciona una visión clara de cuánto queda para completar una épica específica.

Integración y Coordinación

- **Asignación a Sprints**:
  - Las **tareas** pueden ser asignadas a sprints específicos, lo que permite al equipo enfocarse en un conjunto definido de actividades durante un periodo corto y manejable.
  - La asignación también ayuda a gestionar el tiempo y los recursos, asegurando que las **tareas** se completen dentro del plazo establecido.
- **Relación entre Épicas y Sprints**:
  - Aunque no hay una relación directa entre **épicas** y **sprints**, la interacción se produce a través de las **tareas**. Las **tareas** de una **épica** pueden distribuirse en múltiples **sprints**, permitiendo que una épica se desarrolle de manera incremental a lo largo de varios ciclos de trabajo.

Resumen de la Interacción entre los Modelos

- **Tarea**:
  - **Asignada a**: Un **responsable** (`User`) y opcionalmente a un **sprint** (`Sprint`).
  - **Agrupada en**: Una o más **épicas** (`Epica`).
  - **Puede depender de**: Otras **tareas**.
- **Épica**:
  - **Supervisada por**: Un **responsable** (`User`).
  - **Contiene**: Múltiples **tareas** (`Tarea`).
  - **Puede depender de**: Otras **épicas**.
- **Sprint**:
  - **Facilitado por**: Un **Scrum Master** (`User`).
  - **Compuesto por**: Un equipo de desarrollo (`equipo_desarrollo` - múltiples `User`).
  - **Incluye**: Múltiples **tareas** (`Tarea`) en su **backlog**.

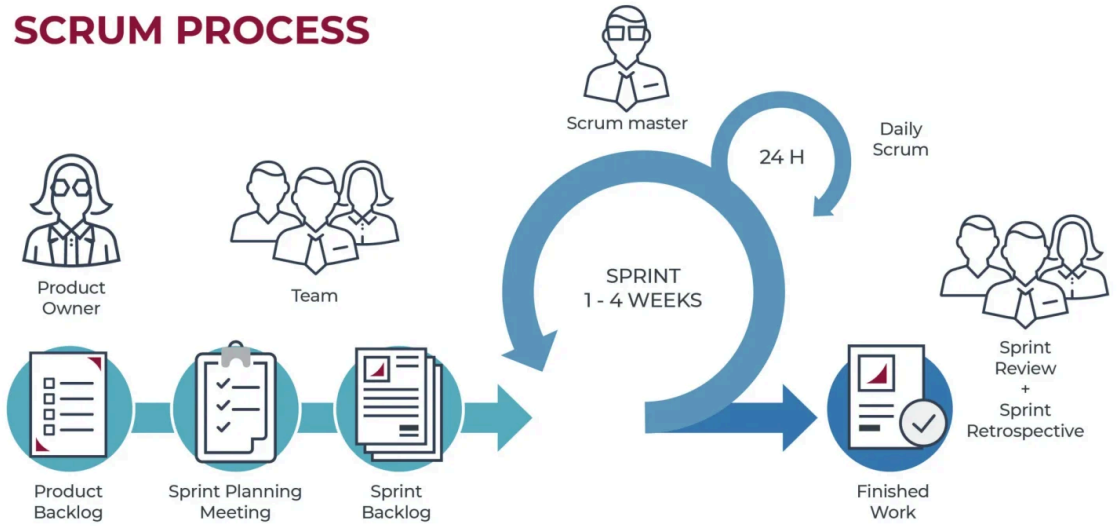
Notas importantes:

- **Relaciones de ManyToMany**:
  - **Sprint** ↔ **Tarea**: Un sprint puede tener múltiples tareas en su backlog, y una tarea puede pertenecer a múltiples sprints.
  - **Epica** ↔ **Tarea**: Una épica puede agrupar múltiples tareas, y una tarea puede pertenecer a múltiples épicas.
- **Relaciones de ForeignKey**:
  - **Tarea** ↔ **User**: Cada tarea está asignada a un responsable.
  - **Sprint** ↔ **User**: Cada sprint tiene un Scrum Master y un equipo de desarrollo compuesto por múltiples usuarios.
  - **Epica** ↔ **User**: Cada épica tiene un responsable.

ANEXO

Funcionamiento de la Metodología SCRUM

SCRUM se organiza en torno a roles, eventos y artefactos específicos, y sigue un ciclo iterativo e incremental. Se basa en ciclos cortos de trabajo llamados **sprints**, que suelen durar entre 1 y 4 semanas, al final de los cuales se debe entregar un incremento de producto funcional. SCRUM facilita la flexibilidad y la capacidad de adaptación, permitiendo a los equipos ajustar el curso de un proyecto según las necesidades cambiantes.



Roles Principales en SCRUM:

1. **Product Owner**:
   - Responsable de definir las características del producto.
   - Prioriza el **Product Backlog** (lista de tareas) para maximizar el valor del producto.
   - Se comunica con las partes interesadas (clientes, usuarios) y representa sus intereses ante el equipo de desarrollo.
2. **Scrum Master**:
   - Facilita el proceso SCRUM y asegura que el equipo siga las prácticas ágiles.
   - Elimina obstáculos que impidan el progreso del equipo.
   - Es un facilitador y protector del equipo, asegurando que puedan trabajar de manera eficiente.
3. **Equipo de Desarrollo**:
   - Un grupo autoorganizado y multifuncional que trabaja para entregar un incremento de producto al final de cada sprint.
   - Participan en todas las etapas del sprint: planificación, ejecución y revisión.

Ciclo de Trabajo en SCRUM

El trabajo en SCRUM se organiza en **sprints**, y cada sprint pasa por las siguientes fases:

1. **Planificación del Sprint (Sprint Planning)**:
   - El equipo selecciona las tareas más prioritarias del **Product Backlog** (lista de requisitos o tareas pendientes) que se van a trabajar durante el sprint.
   - El Product Owner define las prioridades, y el equipo de desarrollo estima el esfuerzo requerido para completar esas tareas.
2. **Sprint**:
   - El sprint es un ciclo de trabajo que dura entre 1 y 4 semanas.
   - Durante el sprint, el equipo se reúne diariamente en reuniones cortas llamadas **Daily Standup** o **Daily Scrum**, en las que revisan el progreso, identifican obstáculos y ajustan su enfoque.
   - No se pueden agregar nuevas tareas al sprint una vez que ha comenzado. Los cambios y nuevas prioridades se manejan en el próximo sprint.
3. **Revisión del Sprint (Sprint Review)**:
   - Al final del sprint, el equipo presenta el trabajo completado (llamado **Incremento**) al Product Owner y otras partes interesadas.
   - Se revisa si las tareas seleccionadas para el sprint fueron completadas, y se discute cualquier retroalimentación o ajustes necesarios.
4. **Retrospectiva del Sprint (Sprint Retrospective)**:
   - El equipo reflexiona sobre lo que funcionó bien y qué puede mejorarse para futuros sprints.
   - El objetivo es mejorar continuamente el proceso de trabajo del equipo.

Artefactos en SCRUM

SCRUM maneja tres artefactos principales que ayudan a gestionar el trabajo y la entrega de valor:

1. **Product Backlog**:
   - Es la lista priorizada de todas las funcionalidades, características, mejoras y correcciones que el producto necesita. Está gestionada por el Product Owner.
   - El **Product Backlog** está en constante evolución y refleja las prioridades del negocio.
2. **Sprint Backlog**:
   - Es la lista de tareas que el equipo se compromete a completar durante un sprint.
   - Es una selección de los ítems más prioritarios del Product Backlog que el equipo considera realista abordar en el sprint actual.
3. **Incremento**:
   - El incremento es el resultado del trabajo realizado durante el sprint. Debe ser un producto funcional y entregable que avance hacia la meta del producto.

Rol de las Tareas, Épicas y Sprints en SCRUM

1. Tareas (User Stories o Product Backlog Items)

- **Descripción**: Las tareas, conocidas en SCRUM como **User Stories** o **Product Backlog Items**, son las unidades de trabajo que el equipo de desarrollo debe completar.
- **Propósito**: Las tareas detallan un pequeño trozo de funcionalidad o un trabajo concreto que el equipo puede terminar en el sprint.
- **Criterios de Aceptación**: Cada tarea incluye criterios de aceptación que definen cuándo se considera que la tarea está completada. Estos criterios ayudan al Product Owner y al equipo de desarrollo a estar de acuerdo en qué significa "hecho" para cada tarea.
- **Prioridad y Estimación**: El Product Owner asigna una prioridad a cada tarea, y el equipo estima el esfuerzo necesario para completarla, usualmente utilizando puntos de historia o días de trabajo.

2. Épicas

- **Descripción**: Las **épicas** son grandes bloques de trabajo que no pueden completarse en un solo sprint. Son grandes funcionalidades o proyectos que se dividen en varias tareas o **User Stories**.
- **Propósito**: Ayudan a organizar el trabajo a nivel estratégico y a largo plazo. Una épica puede representar una funcionalidad importante, un módulo del producto o una mejora considerable.
- **Descomposición en Tareas**: Las épicas se descomponen en varias tareas más pequeñas, lo que permite que el equipo las aborde a lo largo de varios sprints.
- **Gestión**: Una épica se cierra una vez que todas sus tareas relacionadas han sido completadas.

3. Sprints

- **Descripción**: Un **sprint** es un ciclo de trabajo fijo de entre 1 y 4 semanas. Al final de cada sprint, el equipo debe entregar un **incremento** de producto funcional.
- **Propósito**: Los sprints permiten iterar rápidamente sobre el producto, entregando valor incremental al cliente o usuario final. Permiten la adaptación a los cambios de prioridades o necesidades del negocio.
- **Backlog del Sprint**: Cada sprint trabaja sobre un conjunto de tareas seleccionadas del **Product Backlog**. Estas tareas forman el **Sprint Backlog**, y el equipo se compromete a completarlas dentro del sprint.
- **Cierre del Sprint**: Al final de cada sprint, las tareas que no se completaron se devuelven al Product Backlog para ser priorizadas nuevamente en futuros sprints.

Ejemplo de la interacción entre Tareas, Épicas y Sprints en SCRUM

1. El **Product Owner** identifica una funcionalidad importante (una **épica**) que debe ser implementada. Por ejemplo, una nueva sección de "Perfil de Usuario" en una aplicación web.
2. El equipo descompone esta épica en varias **tareas** o **User Stories**, como:
    - Crear la interfaz de usuario para el perfil.
    - Implementar la funcionalidad de actualización de datos.
    - Integrar el perfil con otras partes del sistema.
3. Durante la **Planificación del Sprint**, el equipo decide abordar las tareas relacionadas con el "Perfil de Usuario" en los próximos 2 sprints.
4. Al final de cada sprint, el equipo revisa el progreso de las tareas relacionadas con la épica. Una vez que todas las tareas de esa épica han sido completadas y entregadas, la épica se cierra.