UNIVERSIDAD INTERNACIONAL DE LA RIOJA

Máster en Ingeniería del Software y Sistemas Informáticos

Metodologías, Desarrollo y Calidad en la Ingeniería de Software

Actividad 1: Definición de una metodología basada en Scrum

Presentado por: Grupo 90 (nombre , nombre 2, nombre 3, nombre 4, nombre 5)

Índice general

Índice de figuras

Índice de Tablas

Capítulo 1

Roles y Responsabilidades

En la metodología *Scrum* aplicada al proyecto *"Relatos de Papel"*, se definen tres roles principales: **Scrum Master**, **Product Owner** y **Equipo de Desarrollo** (*Dev Team*). Estos roles, alineados con los pilares de transparencia, inspección y adaptación, promueven valores ágiles como compromiso, coraje, focalización, apertura y respeto, fomentando equipos autoorganizados y multifuncionales.

1.1. Scrum Master

El **Scrum Master** actúa como facilitador y *coach* del equipo *Scrum*, sin asumir funciones de jefe de proyecto. Protege al equipo de interferencias externas, elimina impedimentos, organiza eventos como *Daily Scrum* y *Sprint Retrospective*, y asegura el cumplimiento de prácticas *Scrum*. Ayuda en la autogestión del equipo, resuelve conflictos y lidera la mejora continua del proceso.

En el contexto de *"Relatos de Papel"*, gestiona obstáculos como dependencias técnicas en el desarrollo de funcionalidades de gestión de relatos, promoviendo la adopción efectiva de *Scrum* en la organización.

1.2. Product Owner

El **Product Owner** maximiza el valor del producto gestionando el *Product Backlog*. Determina y prioriza historias de usuario según necesidades de *stakeholders* (usuario final y administrador), define criterios de aceptación claros y aprueba o rechaza los incrementos entregados al final de cada *sprint*. Representa la visión del cliente, evaluando el retorno de inversión para priorizar *features* críticas como la gestión de relatos y usuarios en *"Relatos de Papel"*.

1.3. Dev Team (Equipo de Desarrollo)

El **Dev Team** consta de tres miembros multidisciplinarios (desarrolladores *frontend/backend*, UX y *testers*), autoorganizados y multifuncionales. Estiman el esfuerzo de tareas mediante técnicas como *Planning Poker*, descomponen historias de usuario en unidades de trabajo de máximo 16 horas, y entregan incrementos potencialmente desplegables cubriendo todas las fases: análisis, desarrollo, pruebas e integración.

Como el equipo está distribuido geográficamente en distintos puntos, se realizan reuniones por Microsoft Teams para el *Daily Scrum*, se usan herramientas colaborativas como GitHub (control de versiones) y se ejecuta el desarrollo en ciclos cortos (*sprints*), entregando incrementos funcionales con énfasis en estándares de calidad (pruebas continuas, código limpio) y autonomía (autoorganización).

1.4. Interacción entre Roles

Los roles interactúan diariamente a través de los eventos *Scrum*, clasificados en *"Pigs"* (**Product Owner**, **Scrum Master** y **Dev Team**, comprometidos directamente con el éxito del proyecto) y *"Chickens"* (*stakeholders* externos como clientes, informados, pero no involucrados operativamente).

- En el evento **Sprint Planning**, el *Product Owner* clarifica requerimientos mientras el *Dev Team* estima y selecciona ítems.
- Durante la **Daily Scrum** (15 minutos diarios, coordinado por *Scrum Master*), el equipo reporta avances y bloqueos.
- En la **Sprint Review**, se demuestra el incremento al cliente para recibir *feedback* de calidad.
- Y en la **Retrospective**, se analizan mejoras procesales.

Este flujo iterativo con retroalimentación constante asegura adaptabilidad y alineación continua con los objetivos de negocio.

Capítulo 2

Artefactos clave de Scrum: Product Backlog y Sprint Backlogs

2.1. Product Backlog

En nuestra metodología basada en *SCRUM*, el **Product Backlog** se establecerá como el principal componente de gestión del proyecto *Relatos de papel* y el repositorio central donde se mantienen las historias de usuario y funcionalidades que debe cumplir el sistema. Se definirá como una lista priorizada, detallada y estimada de historias de usuario expresadas en un lenguaje comprensible para el cliente, lo que garantizará que todos los *stakeholders* compartan una visión única del producto durante todo el desarrollo.

Este *Product Backlog* constituye la fuente autorizada de trabajo para el equipo, asegurando que se concentre en los elementos de mayor valor y alineados con los objetivos del negocio. Será dinámico, de modo que el responsable del *Product Backlog*, el **Product Owner**, será quien realice su valoración, ajuste las prioridades y lo mantenga actualizado según las necesidades y prioridades del proyecto.

2.1.1. Definición de Historias de Usuario

Una vez determinados los requisitos funcionales que debe implementar el sistema, estos se plasmarán en las historias de usuario que formarán parte del *Product Backlog*. Estas historias de usuario se definirán como descripciones breves, simples y comprensibles desde la perspectiva del usuario final. Estas historias se centran en sus necesidades y en aportar valor de manera incremental e independiente, detallando qué desea lograr y por qué. Con ello se garantiza que cada funcionalidad contribuya a resolver un problema real y proporcione al equipo el contexto y la claridad necesarios para su correcta implementación.

La información relativa a las especificaciones técnicas necesarias para el desarrollo de las funcionalidades, así como los requisitos no funcionales, no se definirán dentro de las historias de usuario. En su lugar, se documentarán por separado en artefactos adicionales que quedarán vinculados a dichas historias.

Cada historia de usuario se definirá en primera instancia siguiendo la siguiente estructura:

- **Como [Rol]:** Indica quién es el usuario o beneficiario de la funcionalidad.
- **Quiero [Funcionalidad]:** Describe la característica o acción que el usuario desea.
- **Para [Beneficio]:** Explica el objetivo o beneficio que el usuario espera obtener.

Desde el prisma del proyecto *Relatos de papel*, se establecerán dos roles principales: el **usuario final** y el **usuario administrador**.

A continuación, se ejemplifica la definición de historias de usuario del proyecto:

> **EJEMPLO DE CÓMO QUEDARÍA LA HISTORIA DE USUARIO PARA CLIENTE Y ADMIN**

Además de describir una funcionalidad mediante la estructura establecida, las historias de usuario se desarrollarán de forma incremental a través de:

- Las conversaciones entre los distintos *stakeholders* durante la planificación y la iteración, que permiten aclarar y estabilizar los detalles.
- Las pruebas definidas para confirmar su correcta implementación y verificar su finalización satisfactoria.

Para garantizar que las historias de usuario sean claras, manejables y enfocadas a entregar valor, se empleará la técnica **INVEST**. Esta técnica asegurará la calidad de cada historia de usuario mediante el cumplimiento de seis criterios esenciales:

- **Independiente (Independent).** Las historias de usuario deben ser autocontenidas y no depender de otras para completarse.
- **Negociable (Negotiable).** Las historias no deben ser rígidas, sino puntos de partida para la discusión y refinamiento continuo.
- **Valiosa (Valuable).** Cada historia debe aportar valor tangible al usuario o al negocio.
- **Estimable (Estimable).** Las historias deben ser lo suficientemente claras y detalladas para que el equipo pueda estimar el esfuerzo requerido.

- **Pequeña (Small).** Su alcance debe permitir que se completen dentro de un único *sprint*.
- **Verificable (Testable).** Las historias deben incluir criterios de aceptación claros que permitan determinar objetivamente si la historia está completa y cumple lo esperado.

El formato de cada historia de usuario incluida en el *Product Backlog* estará definido por los siguientes campos:

- **Identificador.** Código único e inequívoco que permita rastrear la historia a lo largo de su evolución dentro del *Product Backlog*.
- **Nombre o descripción breve.** Descripción clara y concisa que indique la funcionalidad que representa la historia desde la perspectiva del usuario siguiendo la estructura previamente descrita.
- **Estimación inicial.** Representación, mediante puntos de historia, de una aproximación del esfuerzo requerido para su implementación. Esta estimación es realizada por el equipo de desarrollo en función del tamaño y complejidad de la historia utilizando técnicas de estimación relativas.
- **Valor.** Evaluación del beneficio que la historia aporta al usuario o al negocio, con el objetivo de maximizar el valor y la satisfacción.
- **Prioridad.** Orden definido por el *Product Owner* durante la gestión del *Product Backlog* y que determina el orden de implementación: las historias de mayor prioridad se incluirán primero en el *Product Backlog*.
- **Criterios de aceptación.** Descripción de alto nivel de cómo se validará la historia de usuario. Estos criterios permiten determinar si cumple la *Definition of Done* y son la base para las pruebas de aceptación.
- **Notas.** Información adicional relevante (comentarios, aclaraciones, decisiones tomadas o referencias externas) que facilite la comprensión y el refinamiento de la historia. El nivel de detalle dependerá de su prioridad: las historias de mayor prioridad estarán más refinadas y detalladas.

En la Figura 1 se incluye un ejemplo de la definición de una historia de usuario del *Product Backlog* del proyecto *Relatos de papel*.

Figura 1: Historia de usuario definida en el *Product Backlog* del proyecto *Relatos de papel* y que recoge una de las funcionalidades que debe implementar el sistema.

2.1.2. Estimación de las Historias de Usuario

El esfuerzo de las historias de usuario del *Product Backlog* se estimará mediante **puntos de historia**, una medida relativa del tamaño de cada funcionalidad que tiene en cuenta factores como el riesgo, la complejidad y la incertidumbre. Esta estimación permite conocer el alcance del trabajo, apoyar la priorización y facilitar la planificación del producto.

La asignación de puntos se basará en la **secuencia de Fibonacci**, lo que ayuda al equipo a centrarse en el tamaño relativo de las historias. Así, una historia estimada con más puntos implicará un esfuerzo proporcionalmente mayor que otra con un valor inferior.

Como referencia inicial, el equipo utilizará una **matriz de puntos de historia** (Figura 2), una adaptación de la secuencia de Fibonacci que ofrece pautas más claras sobre cómo clasificar las tareas según su esfuerzo, complejidad y riesgo. Esta matriz evolucionará con la experiencia del equipo a lo largo de los *sprints*.

Figura 2: Ejemplo de matriz de puntos de historia de usuario de referencia.

La estimación de puntos se realizará mediante la técnica **Planning Poker** (Figura 3), que evita sesgos y favorece el consenso. Cada miembro del equipo seleccionará de forma individual una carta con un valor de Fibonacci, se revelarán las elecciones ante el resto del equipo, se discutirán las discrepancias y se repetirán las rondas hasta alcanzar un acuerdo. Este método fomenta el debate, identifica riesgos y promueve una comprensión compartida del trabajo, obteniendo estimaciones rápidas, colaborativas y suficientemente precisas para planificar.

Figura 3: Etapas de la técnica de estimación planning poker.

2.1.3. Priorización del Product Backlog

El *Product Backlog* se priorizará para garantizar que el equipo se concentre en los elementos más críticos, ofreciendo valor al cliente con rapidez y optimizando el uso de los recursos. La priorización permite identificar qué funcionalidades aportan mayor y menor impacto, considerando las necesidades del usuario, los objetivos del negocio y la viabilidad técnica.

El **Product Owner**, con el apoyo del **Scrum Master**, liderará este proceso de priorización, integrando el valor para el negocio y las aportaciones de las partes interesadas. El equipo de desarrollo aportará información sobre complejidad, esfuerzo, dependencias y viabilidad mediante las estimaciones previamente descritas, contribuyendo a establecer prioridades realistas.

Para complementar la priorización se empleará la técnica **MoSCoW**, que clasifica las historias en cuatro niveles según su nivel de importancia:

- **Must have (Debe tener):** Funciones críticas esenciales para que el producto funcione correctamente y se considere exitoso.
- **Should have (Debería tener):** Funciones importantes que pueden mejorar el producto y deberían incluirse en la solución si resulta posible, pero no son esenciales para su éxito.
- **Could have (Podría tener):** Funciones deseables pero que no son urgentes e incluso podrían ser eliminadas. Se deben implementar solo si el tiempo y los recursos lo permiten.
- **Won't have (this time) (No tendrá, por ahora):** Funciones que no se consideran necesarias en este momento, aunque pueden reservarse para futuras versiones del producto.

Esta clasificación asegurará que los elementos más valiosos y estratégicos se implementen primero, mientras que los menos prioritarios podrán posponerse sin afectar los objetivos del proyecto.

2.1.4. Mapas de Historias de Usuario

Para complementar el *Product Backlog* y ayudar a su organización y priorización se construirá un **mapa de historias de usuario**, construyéndose un mapa para cada uno de los roles definidos en *Relatos de papel*: el usuario final y el usuario administrador.

En el nivel superior del mapa se situarán las **epopeyas** (Figura 4), es decir, las historias de usuario de mayor tamaño, funcionalidad y alcance. Las epopeyas se encuentran a un mayor nivel de abstracción que las historias de usuario, con un objetivo más estratégico. Debajo de cada una de ellas, se colocarán las historias de usuario en las que se descomponen.

Figura 4: Ejemplo de epopeya y las historias de usuario en las que se descompone.

Además, las epopeyas se pondrán de izquierda a derecha en función del orden en el que el usuario realice las actividades que definen en la aplicación y en la que mejor describa el sistema a implementar. A su vez, las historias de usuario que formen parte de una epopeya concreta se ordenarán de arriba hacia abajo en función de su prioridad. Esta vista de las historias de usuario jerarquizadas y priorizadas permitirán la definición de las diferentes versiones del sistema a entregar, o *releases*, que se van a realizar, agrupando verticalmente aquellas historias que pertenezcan a la misma *release*.

En el proyecto *Relatos de papel* se establecerá un mínimo de tres *releases* incluyendo el **Minimum Viable Product (MVP)**, que se definirá como el conjunto mínimo esencial de historias de usuario, en base a su prioridad, para considerar que el sistema es exitoso.

El mapa de historias de usuario se diseñará en primera instancia empleando un *template* de Excel, y una vez establecida su estructura, se construirá mediante la herramienta **Featmap**.

En la Figura 5 y la Figura 6 se recoge una previsualización de los mapas de historias de usuario que se establecerán para el usuario final y el usuario administrador del proyecto *Relatos de papel*.

Figura 5: Mapa de historias de usuario del usuario final del proyecto Relatos de papel construido con Featmap.

Figura 6: Mapa de historias de usuario del usuario administrador del proyecto Relatos de papel construido con Featmap.

2.1.5. Criterios de Aceptación de las Historias de Usuario

Para cada historia de usuario del *Product Backlog* se definirán **criterios de aceptación** específicos y verificables que describan el comportamiento esperado del sistema. Estos criterios deben cumplirse para que la historia se considere completada y pueda ser aceptada por el *Product Owner*.

Estos criterios de aceptación permiten:

- Aclarar el alcance de la historia de usuario.
- Identificar restricciones o condiciones necesarias.
- Facilitar la validación al poder transformarse fácilmente en pruebas de aceptación.

Para establecerlos se identificarán los escenarios relevantes (qué debe pasar, qué no debe pasar, y qué resultados se esperan) y se formularán empleando un lenguaje claro y libre de tecnicismos orientado al usuario final. Deben ser:

- Medibles.
- Específicos.
- No ambiguos.
- Independientes entre sí.

Para formalizarlos se utilizará la técnica **Gherkin/BDD** (*Behavior-Driven Development*), empleando el formato **Given – When – Then** (Dado – Cuando – Entonces), que permite describir el comportamiento del sistema de manera estructurada y comprobable.

Un criterio de aceptación para una de las historias de usuario de *Relatos de papel* podría expresarse del siguiente modo:

> **INCLUIR EJEMPLO DE CRITERIO DE ACEPTACIÓN PARA EL EJEMPLO DE HISTORIA DE USUARIO EN APARTADOS ANTERIORES**

2.1.6. Evolución y Refinamiento del Product Backlog

El *Product Backlog* es un artefacto **vivo y emergente** que evolucionará continuamente conforme avance el desarrollo. Las funcionalidades pueden cambiar, añadirse o eliminarse en respuesta a nuevas necesidades del negocio o de los usuarios, cambios en el mercado o descubrimientos técnicos. Esta adaptación constante permitirá maximizar el valor entregado en cada iteración.

El *Product Owner* mantendrá y actualizará el *Product Backlog*, introduciendo modificaciones basadas en la visión del producto y en criterios de negocio. El proceso de **refinamiento del backlog** se realizará de forma periódica e incluirá:

- Incorporación de historias de usuario nuevas.
- División de historias grandes en otras más pequeñas.
- Estimación el esfuerzo requerido.
- Repriorización según valor y necesidades del negocio.
- Eliminación elementos obsoletos o irrelevantes.

Este ajuste continuo asegura que el *Product Backlog* se mantenga alineado con los objetivos del producto y preparado para guiar el trabajo del equipo en las siguientes iteraciones.

2.2. Sprint Backlog

Para cada *sprint* se establecerá un **Sprint Backlog**, entendido como una selección de historias de usuario y la lista de tareas que el equipo se compromete a completar en dicho *sprint*. Este *Sprint Backlog* constituye el plan de trabajo detallado y acotado del *sprint* y se define como un subconjunto estable del *Product Backlog*, lo que garantiza el foco del equipo y evita cambios de alcance durante la iteración.

La selección de historias de usuario se realizará durante el **Sprint Planning**, basándose en su prioridad y estimación previamente establecidas, y en la capacidad del equipo. A partir de estas estimaciones se escogerán las historias necesarias hasta alcanzar la cantidad adecuada de trabajo que el equipo pueda abordar a lo largo de la duración del *sprint*.

Una vez seleccionadas las historias, el equipo llevará a cabo su desglose en tareas, definiendo con claridad las acciones necesarias para implementar cada historia de usuario. Con ello quedará conformado el *Sprint Backlog*, que incluirá:

- Historias de usuario seleccionadas.

- Nombre y descripción de las tareas del *sprint*.
- Prioridad de cada tarea en relación con las demás.
- Gráfico *burndown* del *sprint* que representa el trabajo que queda por hacer en comparación con el tiempo que lleva completarlo.
- Estimación del tiempo requerido para cada tarea.

El cierre del *Sprint Backlog* durante la reunión de planificación marcará el inicio formal de la iteración, momento a partir del cual el equipo se comprometerá con el *Product Owner* a entregar las funcionalidades acordadas y no se admitirán cambios en su implementación durante el *sprint*.

La creación y el almacenamiento de este *Product Backlog* se realizará con la herramienta **Jira**.

Capítulo 3

Eventos, Prácticas de Colaboración y Calendario

3.1. Eventos Scrum

3.1.1. Alineamiento

Consiste en una o varias reuniones con una duración de hasta 4 horas cuyo objetivo es establecer la estimación y puntuación de todas las tareas del *Product Backlog*. Estas sesiones se realizan al inicio del proyecto, basándose en las especificaciones iniciales. Al finalizar esta reunión, debe quedar definido un **Backlog completo y priorizado**.

3.1.2. Sprint Planning

El *Sprint Planning* va a marcar el inicio de cada *sprint* en *Relatos de papel*, el objetivo es alinear al equipo en torno a un objetivo claro y alcanzable. En este proyecto los *sprints* son de 2 semanas, por lo que la planificación se realiza cada 15 días, y tiene una duración aproximada de 2 horas. Durante la sesión se define **qué se hará y cómo se hará**.

Durante esta reunión el *Product Owner* expone las prioridades que hay en el *Product Backlog* basadas en las necesidades de los *stakeholders*, estableciendo un **objetivo de sprint**. Por ejemplo:

> *"Implementar el catálogo de libros que se mostrará en la plataforma de Relatos de Papel para la futura compra"*

El equipo de desarrollo revisa las historias de usuario definidas durante el Alineamiento para determinar cuáles pueden incluirse en el próximo *sprint*. Durante este proceso:

- Analiza cada Historia de Usuario.

- La descompone en tareas o historias más pequeñas.
- Refina los criterios de aceptación.
- Detecta dependencias y riesgos.

El resultado es el conjunto de tareas seleccionadas que formarán parte del *Sprint Backlog*.

3.1.3. Dailys

Las *Dailys* son reuniones diarias de 15 minutos donde el equipo de desarrollo, habla del desarrollo. Cada miembro del equipo expone brevemente:

- ¿Qué hice ayer para contribuir al objetivo del *sprint*?
- ¿Qué haré hoy para contribuir al objetivo del *sprint*?
- ¿Hay algún impedimento que me bloquee?

Por ejemplo para el objetivo de este *sprint* un desarrollador comentaría:

- Ayer desarrollé el *endpoint* de `/books` y realicé pruebas unitarias para validar la carga del listado.
- Ajustar la paginación para mejorar tiempos de cargas.
- Estamos a la espera de que la editorial y el proveedor de contenido validen y entreguen el catálogo actualizado de libros, incluyendo portadas correctas, descripciones y metadatos.

El *Scrum Master* en las *dailys* ayuda a detectar y recopilar todos los impedimentos, pero las soluciones se tratan fuera de las *dailys*, donde el enfoque es informar, no debatir posibles soluciones a los impedimentos.

3.1.4. Sprint Review

La *Review* es la oportunidad de mostrar lo que ya está terminado, de una manera práctica mostrando el **incremento real del producto**, que en el caso de este *sprint* podríamos incluir pantallas funcionales del catálogo de esta librería.

Esta reunión se realiza cada quincena al final de cada *sprint*, con una duración aproximada de 1 hora, donde se reunirá todo el equipo, PO y *Scrum Master* junto a los *stakeholders*.

El *Product Owner* valida las historias completadas y los *stakeholders* aportan *feedback*. Esto permite ajustar prioridades para el siguiente *sprint* y mantener el desarrollo centrado en el valor. Una vez terminada esta reunión se subirá el desarrollo a **producción** dependiendo de si está completa o no la *release*.

3.1.5. Sprint Retrospective

Se realiza después de la *Sprint Review*, con una duración de 1 hora, al final de cada *sprint*. En ella se reúne el equipo y *Scrum Master* para evaluar el *sprint*.

En esta reunión se analizarán las cosas que se hicieron bien, las cosas mejorables y qué acciones se deben aplicar para el siguiente *sprint*. Por ejemplo en este *sprint* se podrían detectar los problemas recurrentes con datos de inventario, como solución en la reunión se propone mejorar la sincronización con la base de datos y se establece un encargado y cuándo se podría llevar a cabo esta mejora teniendo en cuenta las etapas de desarrollo de los *sprints*.

3.1.6. Reunión con el Área de Atención al Cliente

Se realiza cada *sprint* antes de la *Planning* y es una reunión del equipo y PO junto al personal de atención al cliente.

En esta reunión de una duración aproximada de 45 minutos, el equipo revisa los flujos de incidencias de envío, la validación de formularios de soporte y discuten distintos escenarios para que el equipo tenga más contexto del producto. Con ello el equipo podrá asegurar que en la web se refleja la realidad operativa de la librería.

3.1.7. Reunión de Validación Multilenguaje y Accesibilidad

Esta reunión se realizará cada 3 *sprints* con una duración aproximada de 1 hora donde se reunirán el equipo, expertos UX y un traductor/validador.

Debido a que *Relatos de papel* quiere priorizar las traducciones y adaptaciones a discapacidades, cada cierto tiempo se realizará una reunión donde se revisarán traducciones, se harán pruebas con lectores de pantalla y se verificará la usabilidad para las distintas capacidades. Esto lo que busca es un público internacional y diverso para la web y una accesibilidad para todo el mundo.

3.2. Calendario y Cronograma de Sprints

Para garantizar un ritmo sostenible y predecible, se establece una cadencia fija de 2 semanas por *Sprint*. Se ha diseñado un ciclo operativo de Miércoles a Martes. Esta estrategia busca evitar los cierres de *sprint* en viernes, evitando el riesgo de realizar despliegues o actualizaciones críticas justo antes del fin de semana, cuando la capacidad de reacción es menor.

Horarios y Rutinas: La jornada laboral del equipo es de 9:00 a 18:00 (con pausa para comer). Se ha establecido la política de realizar los eventos síncronos (*Dailys*, *Planning* y *Review*)

a primera hora (09:15 AM).

Justificación: Esta decisión busca asegurar la puntualidad del equipo al inicio de la jornada y liberar el resto del día de interrupciones. Al concentrar la gestión a primera hora, se protegen bloques largos de tiempo de desarrollo ininterrumpido incentivando la productividad de los programadores.

Innovación y Mejora Continua (Hack Time): Inspirados en la cultura de Spotify [*Método Spotify*], se reserva la tarde de los martes de cierre de *Sprint* (después de la *Retrospective*, de 12:15 a 18:00) como un espacio flexible:

1. **Prioridad 1 - Estabilización:** Si surgieron incidencias críticas tras la *Sprint Review*, este tiempo se dedica obligatoriamente a *hotfixes* para asegurar la estabilidad de la entrega antes del despliegue.

2. **Prioridad 2 - Pago de Deuda Técnica:** Si la entrega es estable pero durante el *Sprint* se tomaron .ªtajos técnicosçonscientes para llegar a la fecha (código complejo, falta de comentarios, *tests* pendientes de refactorizar), este tiempo se invierte en sanear el código. **Regla de oro:** No se inicia la innovación si el código base está "sucio.º es difícil de mantener.

3. **Prioridad 3 - Innovación:** Si la entrega fue exitosa y el código está saneado, el equipo dedica este tiempo a investigar nuevas ideas, probar nuevas librerías, realizar formación interna o compartir conocimientos. Esto incentiva al equipo a entregar código de calidad durante el *sprint* para poder liberar su tiempo de investigación, además de fomentar la motivación, reducir el *burnout* y mantener al equipo actualizado tecnológicamente.

Code Freeze (Jueves S2): Se recomienda cerrar el código el jueves para dedicar el viernes y lunes de la semana 2 y 3 a pruebas finales de integración y preparación del entorno de demostración.

A continuación, se presenta la simulación del *sprint* tipo del proyecto Relatos de Papel".

3.3. Prácticas de Colaboración

3.3.1. Estrategia de Desarrollo, Infraestructura y Despliegue

Para asegurar un flujo de trabajo ordenado, automatizado y predecible, se define la siguiente arquitectura técnica y política de entregas basada en **GitHub Flow**.

Tabla 1: Visualización del Sprint Tipo

Semana	Lunes	Martes	Miércoles	Jueves	Viernes
Semana 1	(Sprint anterior)	(Sprint anterior)	**INICIO SPRINT** Sprint Planning (9:15 - 13:15)	Daily (9:15) Desarrollo	Daily (9:15) Desarrollo
Semana 2	Daily (9:15) Desarrollo	Daily (9:15) Desarrollo	Daily (9:15) Desarrollo	Daily (9:15) **Code Freeze**	Daily (9:15) Desarrollo
Semana 3	Daily (9:15) QA / Demo	**CIERRE SPRINT** Review Retro Hack Time	**INICIO SIGUIENTE**	(Siguiente Sprint)	(Siguiente Sprint)

Tabla 2: Simulación del Sprint Tipo

Gestión de Ramas y Entornos

- **Rama `main` (Staging / Pre-producción):** Es la rama de integración continua. Todo código aquí ha pasado revisión y pruebas.
- **Entorno Asociado:** *Staging*. Es un espejo de producción utilizado para QA y validación del *Product Owner*.
- **Rama `prod` (Producción):** Contiene exclusivamente el código estable y aprobado para los usuarios finales.
- **Entorno Asociado:** Producción. El entorno vivo de la librería.
- **Ramas `feat/*`, `fix/*` y `refactor/`:** Ramas temporales de vida corta para desarrollo. Se eliminan tras integrarse en `main`.

Pipeline de CI/CD y Política de Entregas

La automatización se gestiona mediante **GitHub Actions**, integrando las reglas de negocio con la ejecución técnica:

1. **Integración Continua (CI) - Al abrir Pull Request**
   - **Disparador:** Creación de una PR hacia `main`.
   - **Acciones Automáticas:**
     - Instalación de dependencias.
     - *Linter*: Revisión automática de estilo de código.

- *Tests*: Ejecución de pruebas unitarias (mínimo 80 % cobertura requerida).

- **Política:** Si algún paso falla, se bloquea el *merge*. El código no puede integrarse hasta que esté "verde".

2. **Despliegue Continuo a Staging - Al hacer Merge**

   - **Disparador:** Aprobación de la PR y fusión (*merge*) en `main`.
   - **Acciones Automáticas:** *Build* de la aplicación y despliegue inmediato al entorno de *Staging*.
   - **Política:** Cada Historia de Usuario terminada debe estar disponible en *Staging* lo antes posible para su validación por QA/PO durante el *Sprint*.

3. **Entrega a Producción - Cierre de Sprint**

   - **Disparador:** Aprobación del incremento en la *Sprint Review* (Martes) y coincidencia con plan de *Release*.
   - **Acciones:** Creación de *Pull Request* de `main` hacia `prod`.
   - **Política de Ventana de Mantenimiento:** Aunque el código esté listo el martes tras la *Review*, el despliegue a Producción se programa para el miércoles por la mañana. Esto asegura disponibilidad del equipo técnico para monitorizar el despliegue y evita incidencias a última hora del día.

3.3.2. Definition of Done (DoD)

Para garantizar la calidad técnica y evitar deuda técnica, el equipo acuerda que ninguna Historia de Usuario se considerará terminada a menos que cumpla estrictamente con la siguiente lista de verificación:

Checklist:

- **Código:** El código ha sido subido al repositorio, sigue los estándares de estilo, ha pasado una revisión (*Code Review*) mediante una *Pull Request* aprobada y está correctamente *mergeado* en la rama `main`.
- **Entorno:** La funcionalidad está desplegada y validada en el entorno de *Staging*.
- **Pruebas:** Se han superado los *tests* unitarios (ejecutados por *GitHub Actions*) con una cobertura superior al 80 %, y las pruebas de integración básicas.
- **Criterios de Aceptación:** Se cumplen todos los requisitos específicos detallados en la Historia de Usuario y todas las tareas técnicas asociadas han sido cerradas.

- **Accesibilidad:** Los elementos de interfaz cuentan con etiquetas ARIA, son navegables por teclado y están traducidos a los idiomas seleccionados.
- **No Bugs:** No existen errores críticos o de funcionalidad conocidos asociados a la tarea.
- **RNF:** Se ha validado el cumplimiento de los requisitos no funcionales.
- **Documentación:** La documentación técnica ha sido actualizada reflejando los cambios.

3.3.3. Prácticas de Colaboración y Gestión de Impedimentos

Para mantener la fluidez del trabajo, se establecen los siguientes mecanismos de comunicación:

Gestión Visual del Trabajo: Utilizaremos un tablero digital Jira con las siguientes columnas para visualizar el flujo de valor: `Backlog` → `Sprint Backlog` → `In Progress` → `Code Review` → `Testing/QA` → `DONE`.

Canales de Comunicación

- **Síncrona:** Se utilizará un canal de chat en Teams con canales temáticos y mensajes directos. **Regla:** Si una duda técnica requiere más de 5 mensajes de chat, se hace una videollamada rápida. Todas las videollamadas y reuniones se realizarán a través de la herramienta Microsoft Teams y serán grabadas para poder revisar la conversación.
- **Asíncrona:** Las decisiones técnicas y funcionales quedarán registradas en los comentarios de la tarjeta de la tarea Jira para evitar pérdida de información.

Protocolo de Gestión de Impedimentos: Un impedimento es cualquier factor que bloquee el progreso.

- **Detección:** Se mueve la HU al listado `Blocked` en el tablero digital inmediatamente.
- **Acción:** Es responsabilidad del *Scrum Master* eliminar el bloqueo. Si no puede resolverlo directamente, debe encontrar a la persona adecuada en la organización para hacerlo.
- **Seguimiento:** Si el impedimento pone en riesgo el Objetivo del *Sprint*, se comunica al *Product Owner* para gestionar expectativas.

3.3.4. Gestión de Cambios durante el Sprint

Aunque *Scrum* requiere estabilidad para garantizar el foco del equipo, el proyecto debe mantener flexibilidad ante las necesidades cambiantes del mercado. Para gestionar esta tensión, se distinguen el siguiente protocolo de actuación:

Cambios funcionales solicitados por el negocio: Cualquier solicitud de cambio en los requisitos (nuevas funcionalidades o modificaciones de las existentes) debe seguir este flujo:

- **Registro:** Las solicitudes se canalizan a través del *Product Owner*.
- **Evaluación de Impacto:**
  - **Bajo Impacto / No Urgente:** Se documenta como Historia de Usuario en el *Product Backlog* para ser priorizada en futuros *Sprints*.
  - **Alto Impacto / Urgencia Crítica:** Si el cambio es vital y no puede esperar, se aplica la regla del intercambio: El *Product Owner* debe retirar del *Sprint Backlog* actual una o varias tareas de esfuerzo equivalente que aún no hayan comenzado (estado `To Do`).
- **Aprendizaje:** Si un cambio invalida trabajo ya completado, se trata como un "desperdicio" se analiza en la *Sprint Retrospective* para mejorar la toma de requisitos futura.

Apéndices