

Lectura 5. Sprint y Elementos de SCRUM

En Scrum, el trabajo no se realiza de forma lineal ni mediante entregas únicas al final del proyecto. El avance se organiza a través de bloques fijos de tiempo llamados **eventos o ceremonias**, diseñados para planificar, sincronizar al equipo, revisar avances e implementar mejoras de manera regular.

1. Sprint: El corazón de SCRUM

El **Sprint** es un contenedor temporal de duración fija (generalmente de **1 a 4 semanas**) durante el cual el equipo colabora para transformar un conjunto de requerimientos en un **incremento funcional de software**.

- **Cadencia constante:** Un Sprint comienza inmediatamente después de concluir el anterior. Su duración no se modifica sobre la marcha para asegurar un ritmo de trabajo predecible y sostenible.
- **Propósito:** Entregar valor utilizable y tangible al usuario final en ciclos cortos, reduciendo el riesgo de construir un sistema equivocado.

2. Eventos del ciclo de Scrum

1. Sprint Planning

Define prioridades (*¿qué hacer?*) y plan técnico (*¿cómo construirlo?*).

2. Daily Scrum

15 minutos diarios frente al tablero para sincronizar y quitar bloqueos.



3. Sprint Review

Demostración en vivo del software funcionando ante los interesados

4. Sprint Retrospective

Revisión interna del equipo para acordar mejoras en sus procesos.

Cuadro comparativo de los eventos de Scrum

Evento	Momento de ejecución	Propósito central	Participantes clave
Sprint Planning	Inicio del Sprint	Definir qué se desarrollará y cómo se ejecutará técnicamente.	Product Owner, Scrum Master, Scrum Team
Daily Scrum	Diario (15 min)	Sincronizar tareas diarias e identificar impedimentos.	Scrum Team y Scrum Master
Sprint Review	Fin del Sprint	Presentar el incremento terminado y recibir retroalimentación.	Scrum Team, PO y Stakeholders
Sprint Retrospective	Cierre del ciclo	Inspeccionar procesos y definir mejoras del equipo.	Scrum Team y Scrum Master

Act008. Glosario técnico y dinámica de eventos Scrum

Materiales: Papel bond, tijeras, plumines y cinta adhesiva.

Con base en la Lectura 5, elabora el glosario de los 4 eventos Scrum (momento, propósito y roles). En equipo, diseñen en papel bond el cartel del ciclo iterativo con sus entradas y salidas, y preparen la simulación en clase del evento asignado.

Lectura 6. Gestión del desarrollo de software: Requerimientos funcionales y no funcionales

1. ¿Qué es un requerimiento de software?

En el desarrollo de sistemas, un **requerimiento** es una necesidad, condición o característica que el software debe cumplir para resolver un problema real o satisfacer a sus usuarios. Antes de comenzar la etapa de codificación, el equipo de desarrollo debe delimitar con total claridad qué funciones construirá y bajo qué condiciones operará la plataforma.

1.2 Requerimientos funcionales (RF)

Los **Requerimientos Funcionales (RF)** definen las acciones, procesos, cálculos y comportamientos específicos que un sistema de software debe ejecutar para satisfacer las necesidades operativas de sus usuarios.

Características clave:

- **Orientados a la acción:** Describen tareas concretas como guardar, buscar, calcular o mostrar información.
- **Activados por el usuario:** Se ejecutan al presionar un botón, llenar un formulario o seleccionar una opción.
- **Fáciles de comprobar:** Su cumplimiento es directo; la función realiza la acción o simplemente no funciona.
- **Manejan datos:** Crean, consultan o actualizan registros dentro del sistema.

Regla de redacción: Se redactan siempre con un verbo en acción directa:

1.1 Requerimientos no funcionales (RNF)

Los **Requerimientos No Funcionales (RNF)** definen las propiedades, restricciones y condiciones de calidad bajo las cuales debe operar el sistema.

Características clave:

- **Criterios de calidad:** Regulan la velocidad, seguridad, estabilidad y apariencia visual.
- **Condiciones del entorno:** Establecen la compatibilidad con dispositivos móviles o navegadores.
- **Medibles y cuantificables:** Se evalúan con métricas objetivas (tiempo en segundos, usuarios simultáneos).
- **Afectan a todo el sistema:** No dependen de un solo botón, sino de la experiencia global.

Regla de redacción: Se redactan estableciendo una condición de calidad o restricción:

- *"El sistema debe responder en menos de..."*
- *"Las contraseñas deben almacenarse de forma..."*
- *"La interfaz debe adaptarse a pantallas de..."*

EJEMPLOS

Elemento	Requerimiento Funcional (RF)	Requerimiento No Funcional (RNF)
Buscador	El sistema busca palabras en español y otomí.	Devuelve los resultados en menos de 1 segundo.
Audio	Reproduce cómo se pronuncia la palabra.	El sonido se escucha claro y sin trabarse
Inicio de sesión	Permite entrar con usuario y contraseña.	Protege la contraseña para que nadie más la vea.

Act009. Gestión de Requerimientos: Glosario Técnico y Conceptual

1. Realiza la lectura comprensiva de la Lectura 6 y redacta en tu cuaderno o procesador de texto la definición con tus propias palabras de: Requerimiento Funcional (RF), Requerimiento No Funcional (RNF), Criterio de calidad e Incremento de software.
2. Formula dos ejemplos aplicados al proyecto escolar Ñätho Digital: un requerimiento funcional y una condición o restricción no funcional.

Lectura 7. Gestión del desarrollo de software: Técnica de priorización MoSCoW

1. ¿Qué es la técnica MoSCoW?

En el desarrollo de software bajo metodologías ágiles, los recursos y el tiempo de entrega son limitados. La técnica **MoSCoW** es un método de priorización que permite al equipo de desarrollo y al cliente clasificar los requerimientos según su nivel de urgencia e importancia para la primera entrega funcional del sistema.

LETRA	SIGNIFICADO	DESCRIPCIÓN CLAVE
M	Must have – Debe tener	Obligatorio y vital. Si falta, el sistema no funciona ni se puede entregar.
S	Should have – Debería tener	Muy importante, no vital. Aporta mucho valor, pero puede esperar a la siguiente entrega.
C	Could have – Podría tener	Deseable o secundario. Mejoras visuales o de comodidad que solo se hacen si sobra tiempo.
W	Won't have – No tendrá por ahora	Fuera de esta entrega. Se descarta en esta versión y se guarda para desarrollarse a futuro.

1.2 ¿Para qué sirve y cuál es su objetivo?

En proyectos con metodologías ágiles, el tiempo de entrega y el presupuesto son fijos. El objetivo de MoSCoW es evitar que el equipo gaste tiempo en detalles decorativos antes de tener listo lo esencial. Permite acordar con el cliente qué se entregará primero para garantizar un **Producto Mínimo Viable (MVP)** funcional.

3. Reglas de distribución y criterios de decisión

Para que un proyecto sea realista y no colapse en las fechas de entrega, los equipos ágiles aplican tres principios clave:

- **Distribución del esfuerzo:** No todo puede ser urgente. Por regla general, los requerimientos clasificados como **Must have** no deben superar el **60%** del esfuerzo total del equipo. El **40%** restante se reparte entre **Should** y **Could** para tener un margen de maniobra ante imprevistos.
- **La prueba de la alternativa (Workaround):** Si una función parece obligatoria, pero existe una forma manual o temporal de resolver la necesidad del usuario sin que el software se rompa, automáticamente se clasifica como **Should have**.
- **Gestión de cambios (Won't have):** Esta categoría no significa que una idea sea mala o se deseche para siempre. Su objetivo es delimitar el alcance actual (*scope*) y proteger al equipo de sobrecargas, programando esas funciones formalmente para la versión 2.0 o siguientes entregas.

2. Ejemplos aplicados al proyecto escolar (Ñätho Digital)

- **Must have (Debe tener):** Catálogo de palabras con traducción español–otomí.
Justificación: Es la función principal; sin palabras no hay aplicación.
- **Should have (Debería tener):** Reproducir el audio con la voz nativa.
Justificación: Ayuda mucho a aprender a hablarlo, pero si no hay audio, aún se puede leer la palabra.
- **Could have (Podría tener):** Modo oscuro y cambio de tamaño de letra.
Justificación: Hace que se vea más cómoda la pantalla, pero no afecta si estudias o no.
- **Won't have (No por ahora):** Inteligencia artificial que califique cómo hablas.
Justificación: Es muy difícil de programar y no da tiempo en este parcial; se deja para después.

Act010. Cuadro comparativo de la técnica MoSCoW y clasificación de requerimientos

Elabora un cuadro comparativo donde expliques las cuatro categorías de priorización (Must have, Should have, Could have, Won't have), su significado técnico y la regla de decisión para clasificar una función.

Act010. Clasificación de requerimientos

Consulta el catálogo preliminar de Ñätho Digital en Google Drive, selecciona 5 requerimientos individuales y clasifícalos justificando brevemente tu decisión con base en el valor para el usuario y la viabilidad técnica.

Lectura 8. Gestión del desarrollo de software: Historias de Usuario y Criterios de Aceptación

1. ¿Qué es una Historia de Usuario (User Story)?

En el desarrollo ágil de software, una **Historia de Usuario** es una descripción breve, simple y en lenguaje natural de una funcionalidad que el software debe incorporar. A diferencia de la ingeniería de software tradicional donde se redactaban extensos manuales técnicos de cientos de páginas, la historia de usuario describe el requerimiento **desde la perspectiva, necesidad y beneficio de la persona que interactúa con la aplicación**.

Su función primordial no es documentar código, tablas de bases de datos o especificaciones técnicas rígidas, sino representar una unidad funcional de valor y servir como recordatorio para entablar una conversación continua entre el cliente (*Product Owner*) y el equipo técnico de programadores.

1.1 Estructura estándar: La fórmula de las tres partes

Para garantizar que una Historia de Usuario capture con precisión la necesidad real sin perderse en tecnicismos, se utiliza la **fórmula canónica de Mike Cohn**, compuesta por tres cláusulas esenciales:



Como [Rol]	Quiero [Acción]	Para [Beneficio]
¿Quién lo necesita?	¿Qué desea hacer?	¿Por qué es importante?
Define el perfil del beneficiario. Se evitan términos genéricos como "usuario" y se especifica el rol real (<i>estudiante, docente, administrador</i>).	Describe la meta o tarea dentro de la aplicación, redactada en lenguaje funcional cotidiano, sin incluir código ni detalles de bases de datos.	Explica la utilidad o valor real que obtiene el usuario. Si la función no aporta un beneficio claro, se descarta por innecesaria.

1.2 Características de calidad: El modelo INVEST

Para asegurar que una Historia de Usuario esté bien formulada antes de pasar al desarrollo, debe cumplir con las seis propiedades del acrónimo **INVEST**:

I	N	V	E	S	T
Independiente <i>Independent</i>	Negociable <i>Negotiable</i>	Valiosa <i>Valuable</i>	Estimable <i>Estimable</i>	Pequeña <i>Small</i>	Comprobable <i>Testable</i>
Se puede construir y probar sin depender del desarrollo previo de otra historia.	No es un contrato cerrado; sus detalles se ajustan dialogando con el cliente.	Resuelve una necesidad real; si no aporta beneficio claro al usuario, se descarta.	Es lo bastante clara para calcular el tiempo y esfuerzo técnico que tomará crearla.	Su alcance es corto para poder completarse dentro de una sola entrega o clase.	Permite formular pruebas directas (<i>pasa / no pasa</i>) para validar su funcionamiento.

1.3 Buenas prácticas para la redacción de Historias de Usuario

Para garantizar que los requerimientos aporten valor real antes de iniciar la programación en el aula técnica, se deben seguir tres principios fundamentales:

- ✓ **Enfocarse en el valor, no en la tecnología:** Describir la meta desde la perspectiva de la necesidad y el beneficio que se obtiene, omitiendo detalles técnicos internos (como nombres de tablas, consultas SQL o lenguajes de programación).
- ✓ **Especificar el rol real:** Evitar términos genéricos como "*el usuario*" y definir con precisión quién interactúa con el sistema (*estudiante principiante, docente, administrador*).
- ✓ **Delimitar el alcance (Principio KISS):** Cada historia debe resolver una sola funcionalidad a la vez. Si incluye la conjunción "y" para enlazar dos acciones complejas, debe dividirse en dos historias independientes.

2. Criterios de Aceptación (claros, observables y comprobables)

Son las condiciones mínimas y reglas de negocio que el software debe cumplir para considerar que una Historia de Usuario está terminada satisfactoriamente (Definition of Done). Su función principal es guiar las pruebas funcionales del equipo de desarrollo y evitar discrepancias en la entrega.

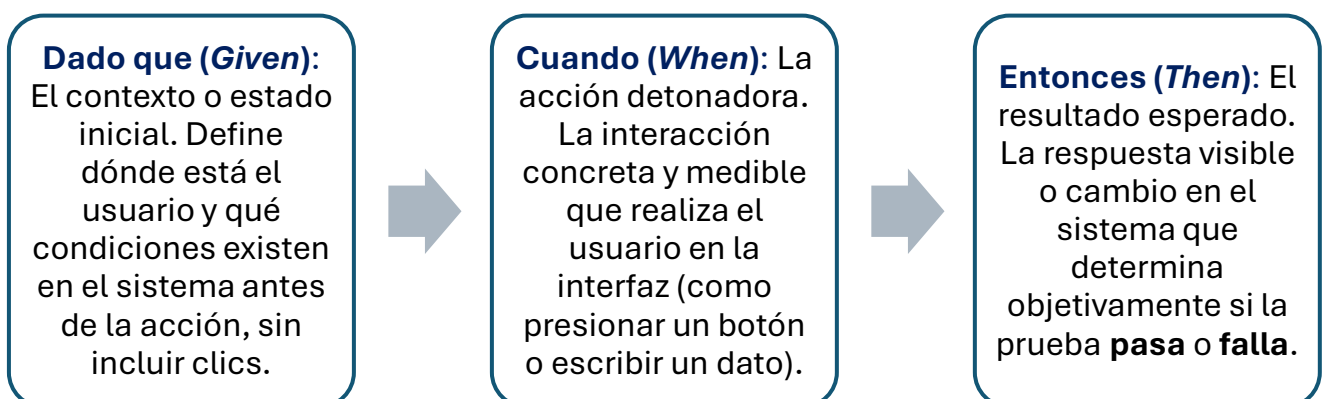
Para formularlos de manera objetiva, deben cumplir tres propiedades esenciales:

Atributo	Significado técnico	Criterio de verificación
Claros	Redactados en lenguaje directo, sin tecnicismos ambiguos.	No deja margen a interpretaciones subjetivas.
Observables	El resultado se percibe directamente en el sistema.	Es visible en pantalla, audible o medible en la interfaz.
Comprobables	Permite diseñar una prueba formal e inmediata.	El resultado de la evaluación es binario (pasa / no pasa).

2.1 Estructura estándar de comportamiento: Sintaxis Dado / Cuando / Entonces (Given-When-Then)

Esta técnica proviene del enfoque **BDD (Behavior-Driven Development o Desarrollo Guiado por el Comportamiento)** y formaliza los escenarios de prueba en un lenguaje natural compartido entre el cliente, los programadores y el equipo de calidad. Su propósito es definir con exactitud cómo debe reaccionar el software ante eventos concretos, transformando los requerimientos en pruebas funcionales medibles.

La sintaxis organiza la lógica de aceptación a través de tres cláusulas secuenciales:



EJEMPLO: Subir una foto a Instagram

Imagina que te piden programar el botón para subir fotos en Instagram. Si redactas lo que quieres como si fueras una computadora, nadie te va a entender y no sabrás si la app funciona bien o mal.

Lo que NO debes hacer	Lo que SÍ debes hacer
Historia de Usuario : × Como base de datos del teléfono, × Quiero guardar archivos JPG y procesar código. × Para no quedarme sin memoria.	Historia de Usuario : ✓ Como usuario con cuenta de Instagram, ✓ Quiero elegir una foto de mi galería y publicarla ✓ Para que mis amigos y seguidores la vean.
Criterio de Aceptación: × Dado que abro la aplicación en mi teléfono × Cuando toco cualquier botón para subir una foto × Entonces la app funciona rápido y todo sale chido.	Criterio de Aceptación: ✓ Dado que elegí la foto de mi galería y escribí el pie de foto ✓ Cuando toco el botón azul de "Compartir", ✓ Entonces la foto aparece en mi perfil y sale un mensaje que dice "Publicado".
¿Por qué está mal? × Una base de datos no es una persona; las aplicaciones las usan humanos. × Usa tecnicismos de programación (<i>JPG, procesar código</i>) en vez de la necesidad del usuario. × "Todo sale chido" es ambiguo y no describe un resultado observable que se pueda evaluar.	¿Por qué está bien? ✓ Está redactada desde el rol de un usuario real. ✓ Describe la acción en lenguaje claro y sin tecnicismos innecesarios. ✓ Es observable y medible: el resultado en pantalla permite validar objetivamente si la prueba pasa o falla.

Guía de autoevaluación rápida antes de validar tu Historia de Usuario:

- ✓ **¿Tiene un rol humano y real?** Evita poner "Como sistema" o "Como usuario general". Especifica quién (ej. estudiante, docente, hablante otomí).
- ✓ **¿El Criterio de Aceptación es binario (pasa / no pasa)?** Debe comprobarse directamente en pantalla, sin adjetivos subjetivos como "rápido" o "fácil".
- ✓ **¿Cumple con el principio INVEST?** Verifica que sea lo bastante pequeña para resolverse en una sola entrega técnica.

Actividad 011: Formulación preliminar de Historias de Usuario y Criterios de Aceptación

Instrucciones: Con base en lo revisado en la Lectura 8 y los requerimientos priorizados en el Tablero MoSCoW del proyecto Ñätho Digital, selecciona dos requerimientos individuales (uno clasificado como Must have y uno como Should have). Redacta para cada uno:

1. Su Historia de Usuario siguiendo la estructura estándar:
 - Como [rol específico]
 - Quiero [acción funcional en lenguaje cotidiano]
 - Para [beneficio o valor real obtenido]
2. Su Criterio de Aceptación observable y comprobable con sintaxis BDD:
 - Dado que [contexto o estado inicial]
 - Cuando [acción medible en la interfaz]
 - Entonces [resultado visible o respuesta del sistema]