VERSÍCULOS DE UN SCRUM MASTER

(Dejamos de ser tradicionales, hay que innovar un poquito)

1. ROLES DE EQUIPO

1.1 MBTI: es un indicador de personalidad basado en 4 dicotomías, que son los cuatro estilos de conducta de las personas:

- Extrovertido(E)/Introvertido(I)
- Sentidos (S)/ Intuición (N)
- Pensamiento (T) / Sentimiento (F)
- Juicio (J) / Percepción (P)

Saber la personalidad de los integrantes del grupo es muy importante, pues sirve para ver cómo tratar o actuar ante los mismos y para predecir sus reacciones.

Además, cuando el trabajo es complejo, importante y con calendario ajustado se deberá formar un equipo con personalidades diferentes, porque habrá más discusiones y debates, generando múltiples opiniones diferentes para abordar una tarea o problema, generando resultados más efectivos.

Por otro lado, cuando el trabajo es relativamente simple y con calendario holgado lo mejor es elegir a gente con personalidades similares.

1.2 DISC: Test psicológico que representa en un círculo cuatro cuadrantes que representan cuatro tipos de personalidad:

- Dominante
- Influyente
- Estable
- Consciente

Todo el mundo es una mezcla de todos.

1.3 BELBIN: Un **rol de equipo** es la tendencia de una persona a comportarse, contribuir y relacionarse socialmente.

Para determinarlo, es necesaria su percepción y la que los demás tienen de él.

Para una persona es necesario saber su rol debido a que:

- Reconoce sus mejores habilidades para ponerlas en práctica y que sean efectivas
- Se obtiene mayor confianza en uno mismo y mayor satisfacción laboral
- Se reconoce sus limitaciones
- Posicionarse en el equipo de forma que pueda aportar más

Para un equipo es necesario saber el rol de esa persona para:

- Hacer un reparto adecuado de tareas
- Motivar adecuadamente a todos los miembros del equipo
- Identificar necesidades y poner en marcha mejoras
- Reconocer y valorar las mejores contribuciones de cada miembro

Roles de Belbin: se identifica 3 categorías de roles:

1. Roles Mentales: orientados hacia el mundo de las ideas (Cerebro, Monitor Evaluador, Especialista)
2. Roles de Acción: orientados al desempeño de las tareas (Impulsor, Implementador, Finalizador)
3. Roles Sociales: orientados hacia las relaciones con las personas (Coordinador, Investigador de recursos, Cohesionador)

Teoría de Belbin:

1. En el equipo debe haber una representación de todos los roles y una persona puede representar más de un rol
2. Los roles de acción deben predominar sobre los otros si se quiere cumplir el trabajo en tiempo
3. Los roles mentales y sociales no deben estar sobrerrepresentados

1.4 PASOS DE LOS EQUIPOS:

1. Formación
2. Conflicto
3. Normalización
4. Desempeño
5. Disolución

2. METODOLOGÍAS DE DESARROLLO

Una metodología de gestión de proyectos consiste en la convención de prácticas, métodos, principios, técnicas y herramientas que otorgan un mejor rendimiento del equipo de trabajo y unos mejores resultados.

2.1 CICLOS DE VIDA

2.1.1 Cascada

Las fases del ciclo de vida se realizan de manera lineal, una única vez, y el inicio de una fase no comienza hasta que termina la fase anterior.

No deja claro cómo responder cuando el resultado de una fase no es el esperado.

Fases: Requisitos -> Diseño -> Implementación -> Verificación -> Mantenimiento

2.1.2 Espiral

Cada una de las fases del cascada termina con una evaluación de riesgos y un prototipo.

Los prototipos permiten a los usuarios determinar si el proyecto continúa, debe volver a las fases anteriores o terminar.

Las fases son todavía lineales, los requisitos se realizan en la fase de requisitos, el diseño en la fase de diseño...

2.1.3 Incremental

- Se crean iteraciones, cada iteración con las fases del cascada estándar
- Cada iteración trabaja sobre un subconjunto de los requisitos
- El producto software se va creando por partes y se va integrando

2.1.4 Iterativo

- Se crean iteraciones, cada iteración con las fases del cascada estándar
- En cada iteración se revisa y mejora el producto
- Ejemplo: refactorización

Cada iteración no implica añadir funcionalidades en el producto, pero sí su revisión y mejora

2.1.5 Iterativo e incremental

- Se van liberando partes del producto periódicamente en cada iteración. Cada nueva versión aumenta la funcionalidad y mejora en calidad respecto a la anterior

- No hay que olvidar la parte iterativa (añadir calidad), si no, el producto será inmantenible.

2.2 METODOLOGÍAS TRADICIONALES

Buscan imponer disciplina al proceso de desarrollo de software para de esa forma volverlo predecible y eficiente.

- Orientadas al proceso
- Predictivas
- Ciclos de vida: cascada, espiral, iterativo, incremental

Problemas:

- Poca comunicación con el cliente una vez finalizada la captura de requisitos
- Muchas actividades que hacer para seguir la metodología y eso retrasa el desarrollo
- Cuando tenemos una versión del producto para probar ya es muy tarde para solucionar los posibles errores que se encuentren

Las metodologías tradicionales son valiosas cuando se necesita control, documentación detallada y predictibilidad en proyectos con baja incertidumbre y requerimientos fijos.

2.3 METODOLOGÍAS ÁGILES

Cuándo aplicar:

- Equipos pequeños (unas 10 personas)
- Cuando haya requisitos que cambien con frecuencia
- Alcance o presupuesto variable
- Pocas restricciones legales
- Pocas restricciones en el proceso de desarrollo

Problemas:

- Falta de participación del cliente
- Documentación incompleta, lo que genera una rápida degradación y envejecimiento del software

MVP: El Producto Mínimo Viable es un producto con las características suficientes para satisfacer a los primeros clientes que nos proporcione retroalimentación para seguir desarrollando. Para elegir los requisitos que compondrán el MVP hay que clasificarlos según lo siguiente:

- Valor de dichos requisitos para los usuarios

- Viabilidad

A diferencia de una prueba de concepto o prototipo, el MVP está ya preparado para la producción y se presenta a clientes o usuarios reales.

2.4 METODOLOGÍAS ÁGILES VS TRADICIONALES (IMPORTANTE)

¿Cuál es mejor?

Depende del tipo de proyecto:

- Las metodologías tradicionales funcionan muy bien en proyectos con un problema conocido y una solución al mismo definida
- En un entorno muy cambiante donde no está claro el problema a solucionar ni la forma de resolverlo son más eficaces las metodologías ágiles ya que incorporan mecanismos de gestión del cambio que implican menos esfuerzo
- En una metodología ágil el cliente o usuario tienen un papel clave. Si no se está seguro de la implicación alta del mismo, quizá no sea buena idea emplear dicha metodología.

Metodologías tradicionales	Metodologías ágiles
Predictivas	Adaptativas
Orientadas a proceso	Orientadas a personas
Proceso rígido	Proceso flexible
El desarrollo se concibe como un único proyecto	Proyecto subdividido en proyectos más pequeños
Poca comunicación con el cliente. El cliente interactúa con el equipo mediante reuniones	Comunicación constante con el cliente. El cliente es parte del equipo de desarrollo (in situ)
Entrega de software al finalizar el desarrollo	Entregas constantes de software
Más roles, más específicos	Pocos roles, más genéricos y flexibles
Documentación extensa	Poca documentación
Existe un contrato prefijado	No existe un contrato tradicional, debe ser flexible
La arquitectura se define al comienzo	La arquitectura se define y se va mejorando
No se espera que ocurran cambios de gran impacto	Se esperan cambios
Enfocado a entregar "Todo lo acordado"	Lo más valiosos se entrega lo primero, reduciendo el riesgo a tener un producto inutilizable si se acaban los fondos

3. SCRUM

3.1 PILARES

- Transparencia: todo el mundo debe saber lo que está pasando

- Inspección: se debe revisar el producto y el proceso con frecuencia para detectar posibles desviaciones y problemas
- Adaptación: si alguna parte del proceso se desvía fuera de los límites aceptables o el producto es inaceptable, el proceso seguido y los materiales que se producen deben adaptarse lo antes posible.

La transparencia permite la inspección. La inspección permite la adaptación.

3.2 EL SCRUM TEAM

- **Product Owner:** representa al cliente. Es el encargado de maximizar el valor del producto
- **Scrum Master**: se encarga de que todo el equipo entienda y aplique correctamente Scrum
- **Equipo de desarrollo**: convierte lo que el cliente quiere en iteraciones funcionales del producto.

3.2.1 Product Owner

- Una sola persona
- Trabaja desde la tesitura del usuario final y el cliente
- Tiene una visión clara de lo que se va a construir
- Gestiona el Product Backlog
- Este rol lo toma o el cliente o una persona de la empresa que lo representa

3.2.2 Scrum Master

- Una sola persona
- Se encarga de eliminar impedimentos que puedan afectar a la entrega del producto
- Se encarga de hacer coaching al resto del equipo y de facilitar reuniones y eventos si es necesario
- Ayuda al PO a entender Scrum y a priorizar y gestionar correctamente el PB
- Participa en todas las reuniones y se encarga de que se cumpla el tiempo y objetivo establecidos.

3.2.3 Equipo de desarrollo

- **No tiene jerarquías**, todos tienen el mismo nivel y cargo: desarrollador
- **Multifuncional:** dentro del equipo hay profesionales con experiencia suficiente en cada uno de los campos necesarios para entregar una versión 100% funcional
- **Auto-gestionados**: el ED es el único responsable de decidir cómo hacer su trabajo sin que nadie ajeno le imponga condiciones
- Tamaño 3-9 personas

3.3 EVENTOS

3.3.1 Sprint

- Cada sprint es el contenedor para el resto de eventos
- El resultado del sprint es un incremento ejecutable del producto que se muestra al cliente
- Cada sprint es considerado como un proyecto independiente
- El objetivo del sprint nunca se cambia
- No hay espacios entre sprints
- Duración fija
- **Duración corta** (1-4 semanas), lo que facilita la planificación, un feedback rápido o entusiasmo renovado y más facilidad de inspección.
- **Duración limitada**, lo que motiva al cierre
- **Duración consistente**: no se debe cambiar la duración del sprint salvo por una buena razón como para probar con sprints más cortos para ver si así se obtiene más y mejor feedback o en agosto, donde es más práctico pasar de un sprint de 3 a 4 semanas para terminar el último sprint antes de vacaciones. Ventajas de la duración consistente:
    1. Facilita que el equipo se habitúe
    2. Mantener constante el ritmo de trabajo
    3. Facilita la planificación
    4. Mejora la predictibilidad
    5. Facilita la organización de reuniones en la agenda

3.3.2 Reunión de planificación

Se divide en 2 partes:

Parte 1: se deciden cuáles serán las funcionalidades del PB que se incluirán en el sprint y el Sprint Goal. Al finalizar esta parte de la reunión el ED se compromete ante el PO a desarrollar aquello que consideren que podrá ser entregado funcionando y testeado al finalizar el sprint. El SM debe asegurarse de que la reunión se desarrolla correctamente.

Parte 2: El equipo debe crear el Sprint Backlog. Durante esta parte de la reunión el SM debe estar atento y vigilar que se desarrolla correctamente, mientras que el PO debe estar disponible por si surgen dudas.

Pasos:

1. El PO presenta las funcionalidades del PB que desea que se implementen en el sprint y un Sprint Goal que les da coherencia
2. El ED hace las preguntas necesarias para comprender los requisitos con un detalle suficiente como para ponerse a estimar

3. El ED estima cada una de las funcionalidades que ha propuesto el PO y decide cuánto puede entregar
4. PO y ED negocian cuántas de las funcionalidades entran finalmente en el sprint y fijan el Sprint Goal definitivo
5. El ED debe crear un diseño a alto nivel de las funcionalidades que se ha comprometido a realizar
6. El ED descompone cada una de las funcionalidades en tareas, dando como resultado el Sprint Backlog.

3.3.3 Daily Scrum

Reunión diaria cara a cara y de pie que no dura más de 15 minutos.

- Se realiza siempre a la misma hora y el mismo lugar para reducir la complejidad
- Cada miembro del equipo de desarrollo cuenta:
    1. Lo que hizo el día anterior
    2. Los problemas que encontró y cómo los resolvió (si los consiguió solventar)
    3. Lo que va a hacer ese día
    4. Comunican si existe riesgo de no alcanzar el Sprint Goal

El PO puede participar de forma pasiva, hablando si se le hacen preguntas.

El SM debe conseguir que

- Que todos los miembros del equipo se comprometan a tener algo listo al final del día
- Eliminar lo antes posible los impedimentos que puedan surgir
- Que la reunión no dure más de 15 minutos (ni mucho menos)

El equipo puede utilizar cualquier estructura y técnicas que deseen siempre y cuando la reunión se centre en el progreso hacia el Sprint Goal y produzca un plan de acción para el siguiente día de trabajo.

Es una reunión clave:

- Identifica impedimentos
- Mejora la comunicación
- Promueve toma rápida de decisiones
- Informa a todo el equipo del trabajo del resto (transparencia)

Objetivo: mantener la transparencia; todo el equipo sabe lo que han hecho los otros, que van a hacer, si está en riesgo el sprint goal o si ha habido problemas sin solucionar, en cuyo caso serán impediments.

3.3.4 Reunión de revisión

- Reunión centrada en el producto

- Objetivo: presentar al PO y a los stakeholders lo que se ha hecho durante el Sprint
- No se trata de una demostración, sino de una reunión de trabajo que sirve para marcar la estrategia de negocio
- Reunión abierta a todo el mundo
- El ED presenta su producto y su funcionamiento (se deben mostrar las nuevas funcionalidades realizadas en el sprint)
- Se inspecciona:
    1. El incremento
    2. El Product Backlog
- Lo que se adapta en la reunión de revisión es el PB con las condiciones actualizadas de negocio

Pasos:

- El PO y los stakeholders inspeccionan el incremento y dan feedback
- El PO identifica las HUs que se pueden considerar hechas y las que no y revisa el PB a la luz del nuevo incremento

El SM debe vigilar que se desarrolle correctamente y en tiempo.

3.3.5 Reunión de retrospectiva

- Centrada en el proceso y dirigida por el Scrum Master
- Reunión restringida a los miembros del equipo (SM, PO, ED)
- Finaliza el sprint
- Se inspecciona:
    1. El último sprint
    2. Definición de Hecho
    3. Herramientas y procesos técnicos
    4. Relaciones entre los miembros del grupo
    5. Procesos
- El objetivo es inspeccionar lo anterior, fijar las expectativas para el siguiente sprint y la creación de un plan de mejoras para el siguiente sprint.
- El resultado es una serie de acciones de mejora para la forma en que el equipo trabaja.
- Pasos:
    1. Inspeccionar cómo fue el último sprint. Analizar qué fue bien, qué problemas surgieron y cómo esos problemas fueron o no resueltos
    2. Fijar las expectativas para el siguiente sprint, viendo qué cosas se quieren evitar, qué riesgos pueden surgir, qué objetivos se quieren marcar..
    3. Búsqueda de acciones que pueda realizar el equipo de cara a evitar los problemas, mitigar los riesgos o conseguir los objetivos

3.4 ARTEFACTOS

Según la Guía Scrum los artefactos que hay son: Product Backlog, Sprint Backlog, Incremento

3.4.1 Product Backlog

Lista ordenada por valor de todo aquello que el propietario del producto cree que necesita.

El responsable es el PO, que lo crea, lo actualiza y lo ordena.

Es un documento “vivo” pues está cambiando continuamente. Se denomina grooming del PB a las actividades de priorización, ampliación y mejora de los elementos que lo componen.

Objetivo: definir y gestionar el trabajo pendiente para asegurar que el producto evolucione de manera alineada con los objetivos de negocio.

Mantener la transparencia en cuanto a lo que se quiere implementar.

3.4.2 Sprint Backlog

Formado por el Sprint Goal y la descomposición de tareas del subconjunto de ítems del PB que el equipo se ha comprometido a desarrollar durante el sprint.

Lo crea de forma conjunta todo el equipo de desarrollo.

Las tareas están en 3 estados: To Do, Doing, Done.

Objetivo: mantener la transparencia dentro del desarrollo del proyecto y permitir el seguimiento de trabajo

3.4.3 Incremento

Parte del producto producido en el sprint. Debe estar completamente terminada y operativa en condiciones de ser entregada al cliente.

No se deben considerar como incremento a prototipos, módulos o partes pendientes de pruebas o integración.

Si el proyecto necesita documentación, debe también estar realizado para que el incremento se considere como “hecho”.

Un desarrollo no se puede considerar como parte del incremento si no cumple con la Definición de Hecho.

Objetivo: que el cliente de feedback y decida si ponerlo en producción o añadir mejoras y avances, o en su defecto, desecharlo.

3.4.4 Definición de Hecho

El equipo debe acordar qué significa para ellos esta Definición de Hecho.

Un elemento del PB está hecho cuando su lista de tareas se ha realizado con éxito.

Si algún elemento no pasa la lista entera no está hecho y no puede considerarse como parte del incremento ni presentarse en la reunión de revisión.

3.4.5 Burndown Chart

Trabajo que falta por realizar en el sprint. Permite al PO evaluar el progreso de desarrollo.

Los integrantes del ED lo actualizan constantemente indicando cuánto consideran que les falta por terminar.

Se mide en puntos de historia por día.

3.4.6 Impediment Backlog

Lista de situaciones que está impidiendo que el equipo progrese y que deben ser solucionadas por el Scrum Master dentro de las 24h posteriores a su identificación.

4. GESTIÓN DE LA CONFIGURACIÓN SOFTWARE

Los resultados del proceso de ingeniería del software se pueden dividir en: programas software, documentos y estructuras de datos. Todos estos elementos se denominan configuración software.

La GCS es el conjunto de actividades diseñadas para:

- Identificar y definir los elementos del sistema que probablemente cambien
- Controlar el cambio de estos elementos a lo largo de su ciclo de vida
- Establecer relaciones entre ellos
- Definir mecanismos para gestionar distintas versiones
- Auditar e informar de los cambios realizados

4.1 ECS

Un ECS es la información creada como parte del proceso de IS.

Ejemplos:

- Plan de proyecto software
- Product Backlog
- Sprint Backlog
- Código Fuente

- Programas ejecutables
- ...

4.2 CONTROL DE VERSIONES

Sirve para registrar los cambios realizados sobre un archivo o conjunto de archivos para poder recuperar versiones específicas más adelante.

Permiten:

- Recuperar archivos perdidos
- Revertir archivos a un estado anterior
- Comparar los cambios
- Revertir el proyecto entero a un estado anterior
- Ver qué cambio está ocasionando un problema
- ...

Versión: instancia de un elemento de configuración, en un momento dado del desarrollo, que es almacenada en un repositorio y que puede ser recuperada en cualquier momento.

Revisión: versión que se construye sobre otra versión anterior. Generalmente se asocia a la noción de corrección de errores.

Variante: es una versión de un elemento de configuración que coexisten en un determinado momento y que se diferencian en ciertas características. Una variante no reemplaza a otra, sino que abre un nuevo camino de desarrollo. Tipos:

- **Temporal:** se acabará mezclando con otra variante en algún momento. Se suelen emplear cuando es necesario que varias personas trabajen simultáneamente sobre la misma versión de un objeto. Ejemplo de nuestro proyecto: **develop**, que es variante temporal de Main
- **De usar y tirar**: sirven para explorar diferentes soluciones alternativas y quedarnos con la mejor. Ejemplo de nuestro proyecto: **NO USAMOS** Variantes de pruebas: sobre las que se introducen elementos que nos facilitan las pruebas
- **Permanentes**: sirven para realizar cambios de forma definitiva. Ejemplo de nuestro proyecto: **main**.
    1. Variantes de requisitos de usuario: el caso más típico es el idioma
    2. Variantes plataforma: una variante por cada SO o plataforma sobre la que queremos que se ejecute la aplicación.

4.3 GESTIÓN DE RAMAS

Cuando hay varios desarrolladores tocando el mismo código se producen continuos cuellos de botella o continuos conflictos a la hora de integrar. La solución es hacer ramas por funcionalidad.

4.3.1 Main

Fundamental para las pruebas. A esta rama se le proporcionará código con ciertas garantías y que hayan pasado todas las pruebas automáticas.

4.3.2 Release

4.3.3 Hotfixes

Se utilizan para corregir fallos urgentes e imprevistos del código de producción. Se originan de MAIN.

4.4 SISTEMAS DE CONTROL DE VERSIONES

Las herramientas de control de versiones no almacenan físicamente todas las versiones, sino solamente una de ellas.

Para poder recuperar otra versión guardan toda la historia de cambios que han ocurrido sobre el elemento.

Siendo r1 y r2 dos revisiones consecutivas en el grafo de evolución, llamamos **Delta** a una secuencia de operaciones que aplicadas sobre r1 dan como resultado r2.

Tipos de delta:

Según su dirección:

- **Deltas directos**: son los cambios que hacen pasar de una versión a la siguiente. Lo que se guarda es la primera versión
- **Deltas inversos**: son los cambios que hacen pasar de una versión a la anterior. Lo que se guarda es la última versión.

Según su localización:

- **Deltas separados**: los deltas están en ficheros distintos al documento original.
- **Deltas mezclados**: los delta se almacenan dentro del mismo documento.

5. USER STORY MAPPING

Propuesta de valor: expresa de forma clara y concisa aquello que el producto hace mejor que nadie para solucionar un problema o satisfacer una necesidad.

5.1 VALUE PROPOSITION CANVAS

Objetivo: conocer lo mejor posible quién es tu cliente, sus hábitos, qué problemas reales tiene y qué beneficios obtiene al consumir tu producto.

Hay parte del cliente y parte del producto.

Parte del cliente:

- **Jobs**: actividades, problemas o necesidades que el cliente intenta resolver
- **Pains**: situaciones no deseadas que viven los clientes cuando están tratando de realizar los jobs
- **Gains**: beneficios que esperan obtener tus clientes al realizar esas actividades o situaciones agradables

Parte del producto:

- **Products & Services**: lista de productos y servicios que ofreces para ayudar a los usuarios a realizar sus tareas
- **Pain relievers:** de qué forma puede contribuir nuestro producto a paliar las frustraciones de los clientes
- **Gain Creators:** Beneficios que el producto ofrecerá a los clientes.

5.2 USER STORY MAPPING

Permite generar requisitos de forma colaborativa entre el Product Owner, los usuarios finales y el Equipo de Desarrollo. El Scrum Máster suele actuar como moderador o facilitador.

Sirve para generar el primer PB del producto.

Muestra los flujos de trabajo reales

Nos servirá para:

- Tomar decisiones y tener muchas conversaciones sobre el producto
- Durante el resto del proyecto nos ayudará a no perdernos en las piezas.

Elementos:

- **Backbone**: tareas esenciales que proporciona nuestra aplicación
- **Walking Skeleton**: producto con el menor número de historias necesarias para obtener una experiencia de usuario completa (MVP)

Ventajas:

- Ayuda a hacer un MVP y luego iterar sobre ella
- Para priorizar
- Tener conversaciones sobre el producto

6. INGENIERÍA DE REQUISITOS

Los requisitos son clave en el desarrollo de productos software

La Ingeniería de Requisitos es el proceso que permite identificar los requisitos. El resultado final es la SRS en las metodologías tradicionales y el PB en las metodologías ágiles.

Fases:

- **Estudio de viabilidad**. Estimación sobre si se puede resolver el problema del usuario y si es rentable
- **Extracción**: descubrimiento de los requisitos iniciales del sistema
- **Análisis**: análisis de los requisitos para clasificarlos, completarlos..
- **Especificación**: documentación de los requisitos acordados con el cliente con un nivel de detalle
- **Validación**: verificación de los requisitos
- **Gestión de requisitos**: proceso de entender y controlar los cambios en los requisitos del sistema

6.1 SRS

Documento de requisitos del sistema en las metodologías tradicionales.

Debe centrarse en qué hace el sistema, no en el cómo lo hace ni cómo se construirá.

Tiene como objetivo establecer la base para un acuerdo entre clientes y desarrolladores sobre qué debe hacer el software y proporcionar una base para la estimación de costes y esfuerzo.

6.2 HISTORIAS DE USUARIO

Descripción breve de una funcionalidad software desde la perspectiva del usuario. Son el resumen de las conversaciones mantenidas con el Product Owner para tener una comprensión compartida de la funcionalidad que se pide.

Elementos:

- **Identificador** de la historia de usuario único en el PB.
    1. Objetivo: identificar de forma unívoca a la HU
    2. Suele constar de letras y números
- **Título**
    1. Objetivo: describir la HU en una sola frase
    2. Deben ser únicos, cortos, descriptivos y no engañosos
- **Descripción**
    1. Objetivo: que los miembros del equipo sepan claramente cuál es el problema que hay que resolver
    2. Estructura: como <<rol>> quiero << el qué>> para <<beneficio>>
- **Criterios de aceptación**
    1. Describen cómo debe comportarse la aplicación para que una determinada acción pueda llevarse a cabo

2. Son los criterios para saber si una HU cumple con las expectativas del PO y si se puede dar como hecha y sirven de guía para el desarrollo de los tests.
3. Deben cumplir **SMART** (**Específicos** (deben ser concretos), **Medibles** (se deben poder medir para saber si se han alcanzado), **Alcanzables**, **Relevantes** (deben poder tener una importancia y dar un beneficio), **Time-Boxed** (limitados en el tiempo))
4. Estructura: CUANDO acción que se produce, SI/DADO precondición de la que se imparte ENTONCES consecuencias de las acciones esperadas
5. Ventajas:
1. Ayudan a la estimación de la historia al imponerle límites
2. Fomentan la comunicación entre equipo y PO
3. Ayudan al equipo a realizar las pruebas adecuadas
4. Reducen la necesidad de hacer consultas al Product Owner durante el sprint

Las historias de usuario siguen **INVEST**

- **Independientes**: se pueden hacer en cualquier orden
- **Negociables**: no son contratos, son promesas
- **Valiosas**: cada HU debe ser valiosa para el negocio
- **Estimables:** si no podemos estimarlas es porque falta comunicación con el cliente
- **Pequeñas:**
- **Testeables**: se deben poder probar

Técnicas de priorización:

- **DOT Voting**: cada miembro recibe un número determinado de votos y los distribuye entre las HUs
- **Comparación entre pare**s: se va comparando entre dos HUs y se va decidiendo cuál es más valiosa entre ellas.

7. CALIDAD DE SOFTWARE

La calidad es la concordancia con los requisitos, los estándares de desarrollo explícitamente documentados y las características de todo software implementado correctamente.

7.1 TIPOS DE CALIDAD

7.1.1 Calidad No Funcional

- Eficiencia: el sistema responde con la velocidad adecuada
- Usabilidad: el sistema es cómodo y su utilización intuitiva
- Mantenibilidad: el sistema es fácil de modificar
- Portabilidad: el sistema opera en diferentes entornos

7.1.2 Calidad funcional

- Fiabilidad: el sistema se mantiene operativo
- Funcionalidad: el sistema cumple con los requisitos

7.2 COSTE DE LA CALIDAD

Hay tres tipos de coste de calidad:

- De prevención: planificación de calidad, RTFs, formación...
- De evaluación: inspección en el proceso y entre procesos, pruebas, mantenimiento del equipo
- De fallos:
    1. Internos: se detecta el error antes de entregarlo al cliente
    2. Externos: se detecta el error después de entregarlo. Ejemplos: devoluciones de productos, resolución de quejas, coste de los internos...

7.3 TIPOS DE PRUEBAS SOFTWARE

Según la ejecución del código:

- Estáticas: no se ejecuta código
- Dinámicas: se ejecuta y hay dos tipos:
    1. Funcionales: se centra en qué debería hacer el sistema
    2. No funcionales: revisan aspectos no funcionales como usabilidad, seguridad, eficiencia, mantenibilidad, portabilidad....

Según la forma de realizarlas:

- Manuales: ejecutadas por desarrolladores o testers
- Automatizadas: ejecutadas mediante herramientas y/o scripts
    1. Una vez creados los tests se pueden ejecutar las veces que se quiera
    2. No son capaces de entender la satisfacción de los usuarios
    3. Menos posibilidad de error que con las manuales

Según el nivel de opacidad:

- **Caja negra**: se prueba el sistema sin conocer ningún detalle de la estructura, diseño e implementación interna
- **Caja blanca:** es fundamental tener conocimiento sobre la estructura, diseño o implementación interna, porque el objetivo será proponer distintos casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa y todas las condiciones tanto en su condición verdadera como falsa.

Según el nivel de profundidad:

- **Pruebas de unidad**: se prueba cada unidad software de manera independiente
- **Pruebas de integración**: se prueban agrupaciones de unidades software
- **Pruebas de sistema**: se prueba el sistema completo
- **Pruebas de aceptación:** se prueba el sistema completo en el entorno real de trabajo

7.3.1 Pruebas de unidad

- Son de caja blanca
- Son pruebas automatizadas que verifican la funcionalidad de una unidad software
- Su objetivo es tomar la unidad más pequeña del software, aislarla del resto del código y comprobar si funciona como se espera
- La idea es escribir diferentes casos de prueba para cada unidad y deben ser pruebas simples
- Cada unidad se prueba por separado antes de integrarlas
- Deben ser rápidas y poder repetirse tantas veces como se quiera.
- Deben cubrir todo el código
- No pueden afectar a las pruebas de otro

Stubs: fragmento de código que simula la actividad de los componentes faltantes en las pruebas reemplazándolos o simulándolos. Son componentes de “mentira” que sustituyen a componentes aún no implementados.

7.3.2 Pruebas de integración

- En estas pruebas se trata de validar el funcionamiento de los distintos módulos como grupo. Tratan de identificar problemas que ocurren cuando los componentes se combinan
- Son de caja blanca
- Los errores que hay en este punto deberían producirse por fallos al combinarse y no ser fallos de la unidad.

Top-Down: la integración se lleva a cabo de arriba a abajo. Primero se empieza con los componentes que no son llamados por otros componentes y paso a paso se van integrando aquellos componentes que son llamados desde la parte integrada.

Bottom-Up: la integración se lleva a cabo de abajo a arriba. Primero se empieza con los componentes que no llaman a otros y se continúa con los componentes que solo llaman a la parte ya integrada.

7.3.3 Pruebas de sistema

- Son las pruebas para validar que el sistema completo cumple los requisitos marcados
- Son de caja negra

- Hay:
    1. Pruebas funcionales: lo que importan son las entradas y salidas del software.
    2. Pruebas de carga: se evalúa cómo se comporta el sistema ante distintas cargas de usuarios pero siempre dentro de los límites del sistema.
    3. Pruebas de estrés: se somete el sistema a cargas crecientes hasta que falle. Permite saber cuál es la carga máxima que soporta
    4. Pruebas de usabilidad
    5. Test de seguridad

7.3.4 Pruebas de aceptación

- Pruebas manuales que prueban las funcionalidades de la aplicación desde el punto de vista del usuario final para ver si se ha realizado correctamente todo lo especificado
- Las hace el PO en Scrum
- Son de caja negra

7.4 INTEGRACIÓN CONTINUA

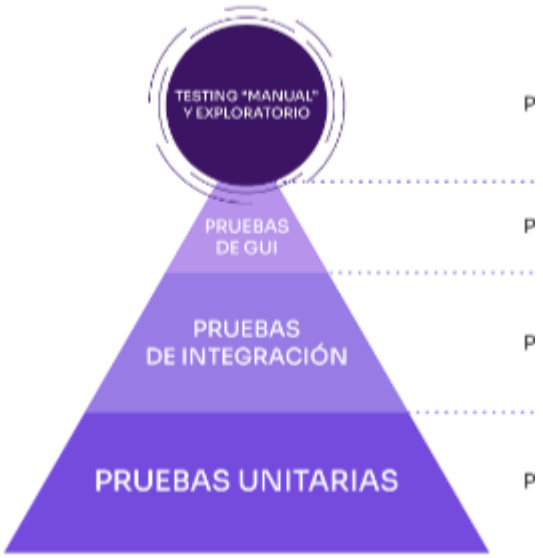
Es la práctica de desarrollo software donde los miembros del equipo integran frecuentemente su trabajo, al menos una vez al día.

El objetivo es detectar los errores lo más pronto posible para poder solucionarlos rápidamente.

Hay que definir un pipeline.

7.5 PIRÁMIDE DE COHN

Establece que hay varios niveles de pruebas y señala el grado en que deberíamos automatizarlas.



- Muchos test unitarios automáticos: si una funcionalidad de este punto falla podría fallar lo de los siguientes niveles
- Bastantes test automáticos a nivel de API, integración de componentes o servicios que son candidatos a automatizarse
- Menos test de interfaz gráfica automatizados, ya que son variables y lentos en su ejecución
- Un nivel estable de pruebas automáticas que vayan detectando los testers manuales y se vayan automatizando paulatinamente

7.6 TRIÁNGULO DE HIERRO

Hace referencia a los pilares básicos de cualquier proyecto: alcance, tiempo, coste y calidad

La calidad debe ser el pilar fundamental y ser fija y así se refleja en el triángulo.

Para mantener una calidad adecuada cualquier modificación en una de las 3 otras variables implica la modificación de alguna/s de las otras dos.

7.6.1 En Scrum

- Tiempo (duración del sprint): cada sprint tiene una duración fija
- Coste (número de personas): está compuesto por un número fijo
- Calidad: no es modificable porque Scrum se basa en entregar calidad
- Alcance (número de HUs entregadas al final del sprint): esto sí que es variable según la capacidad del equipo.

7.7 TDD

Es una práctica de programación que consiste en escribir primero las pruebas, después el código fuente que pase la prueba y finalmente refactorizar el código.

Se consigue un código más robusto, seguro y mantenible.

8. ESTIMACIÓN DE HISTORIAS DE USUARIO

8.1 SPRINT GOAL

Es la meta que se establece para el sprint. Justifica por qué el sprint debe llevarse a cabo y por qué aporta valor.

Objetivo: servir al equipo scrum como guía sobre por qué se está construyendo el incremento. Además, favorece que el equipo trabaje conjuntamente y no en iniciativas separadas.

8.2 ESTIMACIÓN SOFTWARE Y CONO DE INCERTIDUMBRE

La estimación es fundamental para tomar decisiones informadas y gestionar el proyecto de forma efectiva.

La estimación no busca una precisión perfecta, sino una guía para la toma de decisiones y la planificación.

La estimación inicial nos va a llevar a tomar decisiones críticas. ¿El objetivo es todavía alcanzable? ¿Sigue teniendo sentido continuar con el proyecto?

Cono de incertidumbre: describe la evolución de la incertidumbre durante la ejecución de un proyecto.

- Al comienzo se conoce poco el producto y el resultado del trabajo, lo que provoca estimaciones con más incertidumbre
- Según se avanza el proyecto se va teniendo más conocimiento y la incertidumbre se va reduciendo progresivamente hasta desaparecer (al final del proyecto)

8.3 ESTIMACIÓN EN SCRUM

Se debe estimar cuanto menos mejor, porque es costoso y añade burocracia.

Antes de proceder a estimar hay que preguntarse el porqué se va a hacer y qué valor proporciona.

En Scrum:

- El PO estima valor
- El ED estima esfuerzo

8.4 ESTIMACIÓN EN HISTORIAS DE USUARIO

El esfuerzo exacto necesario para implementar una HU no se puede prever de forma absoluta y el tiempo tampoco se puede estimar con exactitud porque además de la incertidumbre del trabajo necesario se suma la incertidumbre por el tiempo de trabajo. (No todas las personas necesitan el mismo tiempo para hacer el mismo trabajo)

La estrategia utilizada por Scrum para estimar las HUs consiste en:

- Trabajar con estimaciones aproximadas por comparación, pues las personas son mejores comparando y diciendo ¿en qué medida esta historia es más grande que la anterior? NO SE USAN UNIDADES ABSOLUTAS
- Estimar con expertos

8.4.1 Puntos de historia

Un **punto de historia** es una unidad de medida para establecer el tamaño de una HU en función del esfuerzo necesario para completarla.

La definición de punto de historia utilizada en el proyecto debe ser creada por el equipo y consistente en el proyecto.

Se puede definir:

- En base a horas ideales necesarias para terminar un trabajo
- Como trabajo relativo de tareas conocidas que normalmente se emplea. Por ejemplo, un punto de historia equivale al tiempo que se tarda al hacer la pestaña de login

La estimación en HU con puntos de historia debe incluir también el tiempo que se tarda en hacer pruebas o refactorización y lo que hace que se cumpla la DoD.

8.4.2 Escalas de estimación

Escala de Cohn: la más usada en metodologías ágiles. Tiene los valores 0,1,2,3,5,8,13,20,40,100

Escala de Fibonacci: existe en la vida real y tiene los valores 1,2,3,5,8,13,21,40

La distancia entre los números de la serie refleja que la incertidumbre crece proporcionalmente con el tamaño de la HU.

Para HU pequeñas la incertidumbre es mucho menor que para HU grandes.

Rompen con la tendencia humana de pensar en múltiplos de 2.

8.4.3 Técnicas de estimación

Planning Poker:

- Técnica muy usada en metodologías ágiles
- Se utilizan cartas numeradas con la escala de Fibonacci o Cohn
- Se entrega un conjunto de cartas a cada participante
- El SM o el PO son los moderadores y leen una HU. El equipo discute sobre esa HU antes de ponerse a estimar.
- Cada participante selecciona una carta para hacer su estimación y la deja boca abajo
- Los participantes dan las vueltas a sus cartas: si son similares se escoge la que tiene puntuación mayoritaria. Si no son similares, cada persona explica sus razones y se vuelve a votar.
- El objetivo es ofrecer un proceso de estimación ágil para obtener valores útiles para que el equipo pueda planificar el sprint. Además, se consigue que el equipo entienda lo mismo de la HU estimada, pues si hay puntuación muy dispar es que hay gente que sabe algo que el resto no en ese momento.

Magic Estimation:

- Es más rápida que el Planning Poker
- Se coloca las cartas del Planning Poker en la mesa o en una pizarra o algo (también se puede escribir los números) y se prepara una tarjeta por HU a estimar
- Uno de los miembros del ED coloca una de las HU a estimar junto al valor que cree que pertenece
- El siguiente miembro puede elegir entre estimar una nueva HU o cambiar la estimación ya realizada, en cuyo caso el moderador debe tomar nota de las que se van cambiando.
- Una vez que todas las HUs han sido estimadas y nadie quiere cambiar nada se pasa a la segunda parte.
- Durante la primera parte nadie habla
- Segunda parte:
    1. Las HU que no se han movido se quedan como están
    2. Las HU que se han movido necesitan una revisión. Se pide a las personas que han cambiado la HU que expliquen sus razones y entre todos se llega a un acuerdo
- Cuándo usar: cuando el equipo tenga poca experiencia en estimar o cuando se desee hacer estimaciones rápidas y aproximadas de muchas HU

Affinity Estimation

1. Tamaño relativo (en silencio). El PO proporciona las HU al equipo y el ED establece en silencio los tamaños relativos de las HU
    a. Se dibuja una escala horizontal. Un extremo será marcado como “Más pequeño” y otro como “Más grande”
    b. El PO reparte tarjetas con todas las HU del sprint entre todos los miembros del ED

c. Cada miembro del ED colocas sus HUs en la escala, en cualquier lugar entre “Más pequeño” y “Más grande”
d. Si se considera que 2 HU tienen el mismo tamaño, se deben colocar en vertical una debajo de la otra

2. Se edita la ordenación. Los miembros del ED modifican la ordenación realizada y esto incluye discusiones con el PO. Finalmente, se agrupan en “buckets” según la escala de estimación escogida

8.4.4 Estimación de las HUs que entran al sprint

Se utiliza Yesterday’s Weather, que consiste en pensar que el número de puntos de historia completados en el anterior sprint es la medida más confiable de cuántos puntos de historia se podrán completar en el sprint que comienza

Cuándo usar:

- Equipos que han trabajado juntos durante algunos sprints
- Equipo nuevo que hace un trabajo similar al de un equipo existente (habrá que tener en cuenta si los equipos se parecen en cuanto a técnica)

8.5 BURNDOWN CHART

Es el artefacto que sirve para comprobar si el ritmo de avance es el previsto o se puede ver comprometida la entrega del sprint. Permite evaluar al PO cómo se está desarrollando el sprint.

- Mide el trabajo restante
- En el gráfico de quemado el eje Y es el trabajo que falta y se actualiza a diario
- Se mide en puntos de historia por día

9. DIVISIÓN DE HUs EN TAREAS

9.1 SWARMING

Estrategia que consiste en dedicar todos los esfuerzos a una única HU . Todo lo necesario para completarla debe hacerse lo más en paralelo posible y todo el equipo debe contribuir a finalizarla.

Para conseguirlo hay que asegurarse de que ningún desarrollador se asigne ninguna tarea de otra HU si hay tareas de la HU de mayor prioridad pendientes.

Las Dailys son muy importantes para esto. La idea es trabajar todos juntos para cerrarla en vez de que cada uno trabaje sus tareas y al final no haya nada terminado.

El Swarming lleva a aumentar la multifuncionalidad del equipo.

Ventajas:

- El valor de lo que se entrega es mayor
- Al tener a más de un desarrollador trabajando se consigue una revisión del código a tiempo real
- Transparencia al trabajar todos en lo mismo.

9.2 PATRÓN REY-SIRVIENTE

- Cualquiera que esté trabajando en la HU de mayor prioridad es el rey
- Todos los demás miembros son sirvientes
- Si quieres ser rey tienes que trabajar en la HU de mayor prioridad
- Cuando un rey necesita ayuda, todos los sirvientes deben ofrecerse inmediatamente
- Un siervo no puede interrumpir a un rey
- Tan pronto como acabe la HU de mayor prioridad, aquel que trabaje en la siguiente historia es ahora el rey

10. GESTIÓN DE RIESGOS

Un riesgo es un evento incierto que en caso de que ocurra tiene un impacto positivo o negativo en uno o más objetivos de proyecto. Para que sea riesgo tiene que tener incertidumbre.

10.1 TIPOS DE RIESGO

Según su polaridad pueden ser:

- **Positivos**: tienen un efecto positivo en los objetivos del proyecto y se llaman también oportunidades.
- **Negativos**: tienen un efecto negativo en los objetivos del proyecto. También llamados amenazas

Según el objetivo al que afecten pueden ser:

- **De proyecto:** afectan al esfuerzo y/o coste
- **Técnicos:** afectan a la calidad del software
- **De negocio:** afectan a la viabilidad del proyecto

10.1.1 Riesgos de proyecto

Si se hacen reales aumentan/disminuyen el esfuerzo y/o coste.

Ejemplo: Miembros clave del proyecto encuentran otro trabajo mejor y se van lo que origina un retraso significativo

10.1.2 Riesgos técnicos

Amenazan a la calidad del software y aparecen porque el sistema puede ser más difícil de construir de los esperado

Ejemplo: Los componentes de software elegidos no trabajan adecuadamente lo que provoca fallos en el sistema

10.1.3 Riesgos de negocio

Amenazan la viabilidad del proyecto.

Ejemplo: Una compañía rival ofrece un producto similar antes y se origina pérdida del mercado para el producto

10.2 GESTIÓN DE RIESGOS

Se ocupa de identificar riesgos y ayudar a aumentar la probabilidad y el impacto de los positivos y disminuir la probabilidad y el impacto de los negativos.

Hay dos tipos de estrategias:

- **Reactivas**: se aplican una vez que el riesgo ya se ha hecho realidad. Se asignan recursos por si los riesgos se hacen realidad y no se preocupan hasta que pase. En ese momento es donde se intenta sofocar el problema.
- **Proactivas**: se aplican antes de que el riesgo haya tenido lugar

3 etapas en las estrategias proactivas:

- Identificación del riesgo:
- Análisis
- Planificación

10.2.1 Identificación del riesgo

- Produce una lista con riesgos específicos que puedan afectar al proyecto
- Cada riesgo debe tener un identificador y una descripción
- La descripción debe ser clara y sin ambigüedades y debe dejar claro
    1. La causa: importante para crear los planes de evitación
    2. El evento
    3. El efecto: fundamental para crear los planes de gestión del riesgo y para identificar el tipo de riesgo
- Técnicas para identificarlos:

1. Historial de los riesgos identificados en otros proyectos de la empresa
2. Documentación
3. Entrevistas con miembros senior del proyecto o con los stakeholders
4. ...

10.2.2 Análisis del riesgo

Consiste en priorizar los riesgos para enfocarnos en gestionar los riesgos de mayor prioridad y así reducir la incertidumbre del proyecto.

Para priorizar se pueden usar distintas medidas:

- Probabilidad de que ocurra el riesgo
- Impacto en caso de hacerse real (gravedad)
- Pérdida estimada en caso de producirse el riesgo
- Nivel de riesgo (probabilidad x gravedad)

10.2.3 Planificación

Consiste en desarrollar acciones para mejorar las oportunidades y reducir las amenazas a los objetivos del proyecto. Deben ser proporcionales y realistas.

1. Planes de evitación o explotación: se desarrolla un plan para reducir la probabilidad de que los riesgos se hagan reales y si se hacen reales para minimizar el impacto. En cuanto a los positivos, se busca eliminar la incertidumbre y favorecer a que ocurran.
2. Planes de supervisión: se analizan los factores que pueden indicar si el riesgo se está haciendo más o menos probable. Se analizan:
    - Se valora si el riesgo se está haciendo real
    - Si se está aplicando el plan de evitación y si está funcionando
3. Planes de gestión: se desarrollan los planes de contingencia que se deben aplicar si el riesgo se vuelve real

10.3 RIESGOS EN METODOLOGÍAS TRADICIONALES

Se aplica el plan RSGR

10.4 RIESGOS EN METODOLOGÍAS ÁGILES

Existen riesgos que las metodologías ágiles ya solucionan directamente:

- En cualquier momento se pueden introducir o eliminar requisitos
- Las iteraciones cortas nos permiten tener feedback y minimizar los riesgos
- Papel fundamental de las pruebas
- Entregas continuas al cliente
- ...

Sin embargo los que no se solucionan se trata de que aparezcan lo antes posible para poder atacarlos.

El enfoque es más ligero que en las tradicionales pero se usan las mismas etapas. Se hace menos documentación.

Se suele usar un tablero de riesgos o un risk burn down chart

11. METODOLOGÍAS ÁGILES

11.1 XP

Se centra en las buenas prácticas para el desarrollo de software.

Promueve el trabajo en equipo

XP Planning Game: es como una reunión de planificación, en ella:

- El cliente define los requisitos usando historias de usuario y recalca su importancia
- Los desarrolladores estiman el coste de implementar los requisitos y definen el número de iteraciones para desarrollarla
- Por cada iteración el cliente decide qué historias quiere que se realicen
- Se puede crear o modificar las HUs en cualquier momento

11.1.1 Valores

- Simplicidad en las soluciones
- Comunicación fluida
- Retroalimentación continua entre cliente y ED
- Coraje para enfrentar los cambios
- Respeto por el trabajo de los miembros

11.1.2 Reglas

- Entregas pequeñas. Iteraciones cortas (1-3 semanas). Objetivo: mostrar sw terminado para recibir feedback del cliente
- Diseño simple: se debe diseñar la solución más simple
- Programación en pareja
- Metáforas: todo el equipo debe tener la misma visión del sistema, por eso se emplean metáforas o un glosario de términos para que todo el equipo lo use
- Se debe usar estándares de código. Objetivo: que sea más fácil de comprender
- Propiedad colectiva del código. Motiva a todos a contribuir con ideas nuevas en todos los segmentos del sistema.
- Reuniones diarias. Para exponer sus problemas y buscar soluciones conjuntas

- Un representante del cliente debe trabajar a tiempo completo con el equipo del desarrollo. El objetivo es resolver dudas en la realización de las HUs

11.1.3 Roles

- Programador:
- Cliente: escribe las HUs y las pruebas funcionales
- Tester: ayuda al cliente con las pruebas funcionales y las ejecuta y comunica los resultados
- Coach: debe guiar a los miembros para que apliquen XP correctamente
- Tracker: realiza el seguimiento del progreso en cada iteración
- Consultor: miembro externo con un conocimiento específico para ayudar en problemas específicos
- Big Boss: su labor es la coordinación

11.1.4 Ciclo de vida

1. Exploración
    - Los clientes plantean las primeras HUs
    - El equipo de desarrollo se familiariza con las herramientas y construye un prototipo
2. Planificación de la entrega
    - Cliente: establece la prioridad de cada HU
    - Programadores: hacen una estimación
    - Se acuerdan qué HUs van en cada iteración
3. Iteraciones
4. Producción
    - Se realizan pruebas adicionales y revisiones
    - Se toman discusiones sobre si añadir nuevas funcionalidades
5. Mantenimiento
    - Se debe mantener mientras se añaden nuevas funcionalidades
6. Muerte del proyecto
    - Una vez al cliente no se le ocurren más cosas para añadir o ya está satisfecho se termina

11.1.5 XP vs Scrum

Semejanzas:

- Ambas usan historias de usuario
- Ambas tienen comunicación fluida con el cliente
- En ambas hay reuniones diarias
- En ambas el cliente prioriza HU y el equipo esfuerzo
- En ambas la calidad es fundamental

Diferencias:

Scrum	XP
Más basada en la administración del proyecto	Más basada en la programación o creación del producto
Iteraciones: 2 – 4 semanas	Iteraciones: 1 – 3 semanas
3 roles	7 roles
Al finalizar el sprint las tareas aprobadas por el Product Owner no se vuelven a tocar en ningún momento	Todas las tareas que se terminan en cada entrega son susceptibles de sufrir modificaciones si así lo pide el cliente, incluso después de que funcionen correctamente
Cada miembro del equipo de desarrollo trabaja individualmente	Los miembros del equipo de desarrollo trabajan en parejas

11.2 KANBAN

11.2.1 Pilares

- Busca entrega frecuente
- Busca mejora continua
- Divide el trabajo en tareas
- Trabaja con equipos auto-organizados

11.2.2 Roles

No hay roles preestablecidos y además los equipos pueden ser multifuncionales o especializados

11.2.3 Reglas

- Visualizar el flujo de trabajo: Kanban se basa en el desarrollo incremental, dividiendo el trabajo en tareas o partes. Utiliza técnicas visuales para ver en qué situación se hallan estas tareas (Kanban Board)
- Determinar y respetar el límite de trabajo en curso (WIP)
- Gestionar el flujo: La medición de los tiempos de entrega aporta información valiosa y útil para mejorar.
- Hacer las políticas explícitas; todas las decisiones sobre la organización del trabajo, tanto de forma individual como grupal, debe hacerse explícitas

Visualizar:

Se usan las Kanban Board.

Clases de servicio: son una serie de categorías que sirven para clasificar cada una de las tareas de nuestro sistema. Son:

- Standard
- Expedite (acelerar): tareas que necesitan ser gestionadas de manera rápida
- Fixed Delivery Date (con fecha fija)
- Intangible

Cost of Delay: es el coste de retraso o aquello que perdemos o dejamos de ganar si no hacemos una tarea. Es una de las grandes diferencias de las clases de servicio.

- Standard: el coste es lineal
- Expedite: exponencial

Determinar y respetar el WIP:

El WIP Debe estar limitado, es decir, el número máximo de tareas que se pueden realizar en cada fase debe estar fijo.

Limitar el WIP hace que se deban cerrar unas cosas antes de empezar otras, mejorando así la productividad.

Gestionar el flujo:

Lead Time: tiempo que se tarda en terminar cada tarea. Se cuenta desde que se hace una petición hasta que se hace la entrega

Cycle Time: tiempo que el equipo pasó trabajando realmente en la tarea (sin contar el tiempo que esperó en el tablero)

Hacer las políticas explícitas:

Todas las decisiones deben hacerse explícitas.

11.2.4 Scrum vs Kanban

Semejanzas:

- Buscan mejora continua
- Visualizan el proceso en curso
- Dividen el trabajo
- Trabajan con equipos auto-organizados

Diferencias:

Scrum	Kanban
Objetivo: maximizar el valor de lo entregado	Objetivo: Optimizar el flujo de trabajo
Más prescriptivo	Menos prescriptivo: solo 4 reglas
Roles predefinidos: Scrum Master, Product Owner y equipo de desarrollo	No hay roles predefinidos
Organizado en sprints	No hay sprints, el trabajo es continuo
El límite de tareas a realizar las pone el sprint	El límite de tareas a realizar lo pone el WIP
Menos propenso al cambio: las HU del sprint no pueden tocarse	Más propenso al cambio: solo las tareas en curso no pueden tocarse
Los entregables están determinados por sprints	Entrega de productos continua según sea necesario
Equipos multifuncionales	Equipos multifuncionales o especializados
Priorización obligatoria	Priorización opcional
Mide productividad por puntos de historia completados en cada sprint (velocidad)	Mide productividad con “lead time”: tiempo que lleva completar inicio-fin una tarea