# 1. Proceso de Desarrollo del Software

Ph.D Priscilla Jiménez P.

**CURSO: ITIR554 – Programación basada en eventos**
**2020 B – Semana 2**

# ÁGIL

Permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto.

Método alternativo de gestión de proyectos.

# Desarrollo ágil de software - ¿por qué?

Entorno global cambia rápidamente (nuevas oportunidades y mercados).

Responder oportunamente a la competencia.

**Incluso muchas empresas están dispuestas a negociar la calidad de software y requerimientos.**

# Métodos ágiles - Características

- Se diseñan para producir rápidamente un software útil.
- El software no se desarrolla como una sola unidad sino como una serie de incrementos, y cada uno de ellos incluye una funcionalidad.
- Los equipos evalúan el proyecto en reuniones regulares llamadas sprints o iteraciones.
- Los procesos de especificación, diseño e implementación están entrelazados.
- No existe una especificación detallada del sistema.

http://agilemanifesto.org/iso/es/manifesto.html

# Métodos ágiles - Características

- La documentación de diseño es generada por el entorno de programación.
- El documento de requerimientos del usuario define sólo las características más importantes del sistema.
- El sistema se desarrolla en versiones y los usuarios finales y otros colaboradores intervienen en la especificación y evaluación de cada versión.
- Las interfaces de usuario del sistema se desarrollan usando con frecuencia un sistema de elaboración interactivo.
- Entregas frecuentes.

# Métodos ágiles. Se valora:

1. Los individuos y las interacciones sobre procesos y herramientas.
2. El software operativo sobre la documentación exhaustiva.
3. La colaboración con el cliente sobre la negociación del contrato.
4. La respuesta al cambio sobre el seguimiento de un plan.

# Métodos ágiles

Son útiles para cierto tipo de desarrollo de sistemas:

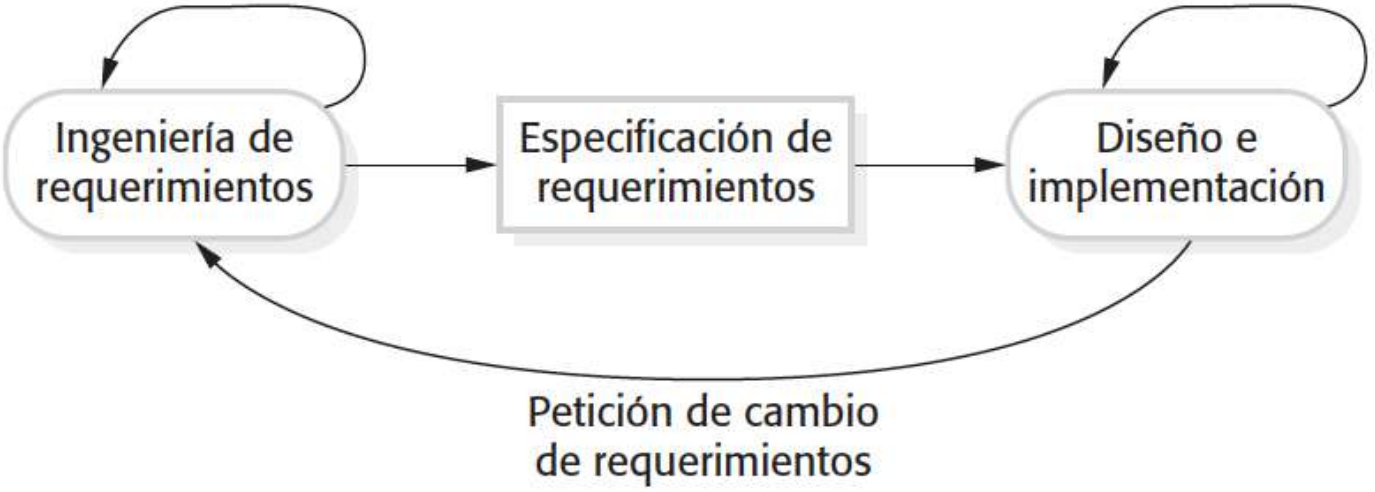
1. Desarrollo de un producto pequeño o mediano.
2. Diseño de sistemas a la medida dentro de una organización.
   - Hay un compromiso del cliente de involucrarse.
   - No existen muchas reglas ni regulaciones externas.

# Principios de los métodos ágiles

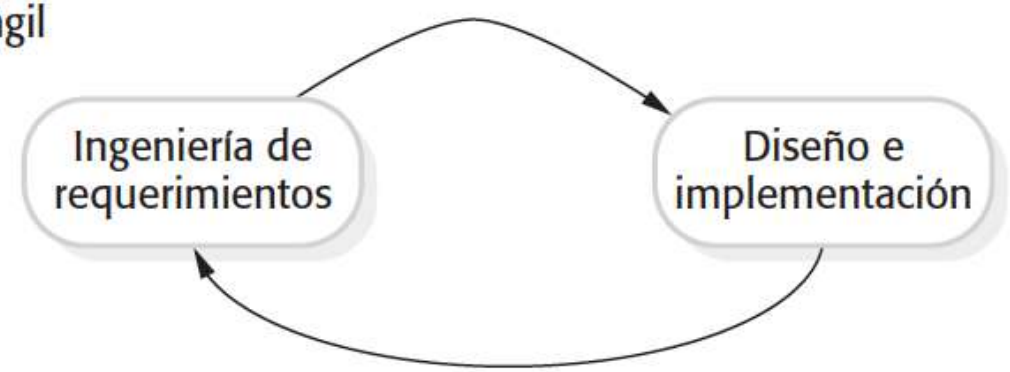
| Principio | Descripción |
|---|---|
| Participación del cliente | Los clientes deben intervenir estrechamente durante el proceso de desarrollo. Su función consiste en ofrecer y priorizar nuevos requerimientos del sistema y evaluar las iteraciones del mismo. |
| Entrega incremental | El software se desarrolla en incrementos y el cliente especifica los requerimientos que se van a incluir en cada incremento. |
| Personas, no procesos | Tienen que reconocerse y aprovecharse las habilidades del equipo de desarrollo. Debe permitirse a los miembros del equipo desarrollar sus propias formas de trabajar sin procesos establecidos. |
| Adoptar el cambio | Esperar a que cambien los requerimientos del sistema y, de este modo, diseñar el sistema para adaptar dichos cambios. |
| Mantener simplicidad | Enfocarse en la simplicidad tanto en el software a desarrollar como en el proceso de desarrollo. Siempre que sea posible, trabajar de manera activa para eliminar la complejidad del sistema. |

# Desarrollo dirigido por un plan y desarrollo ágil

Desarrollo basado en un plan




Desarrollo ágil



# Métodos ágiles – Retos en la práctica

- Tiempo con el cliente y que este represente a todos los participantes de sistema.
- Personalidad de los miembros del equipo. (participación intensa).
- Priorizar los cambios es difícil cuando hay muchos participantes.
- Dificultad al realizar las simplificaciones deseables al sistema.
- Los procesos informales y definidos por equipos de desarrollo no son bienvenidos por todos (grandes compañías).

# Métodos ágiles – Retos en la práctica

- Dificultad en elaborar contratos.
- Falta de documentación formal que facilite el mantenimiento.
- Mantener al cliente involucrado luego de entregar el software.
- Continuidad con el equipo de desarrollo. (Puede ocurrir pérdida de conocimiento implícito).

# ¿Cómo decidir sobre un enfoque basado en un plan o uno ágil?

1. ¿Es importante tener una especificación y un diseño muy detallados antes de dirigirse a la implementación?
2. ¿ Es práctica una estrategia de entrega incremental?
3. ¿Qué tan grande es el sistema que desarrollara?
4. ¿Que tipo de sistema se desarrollará?
5. ¿Cuál es el tiempo de vida que se espera de un sistema?

Preguntas técnicas, humanas, y organizacionales

# ¿Cómo decidir sobre un enfoque basado en un plan o uno ágil?

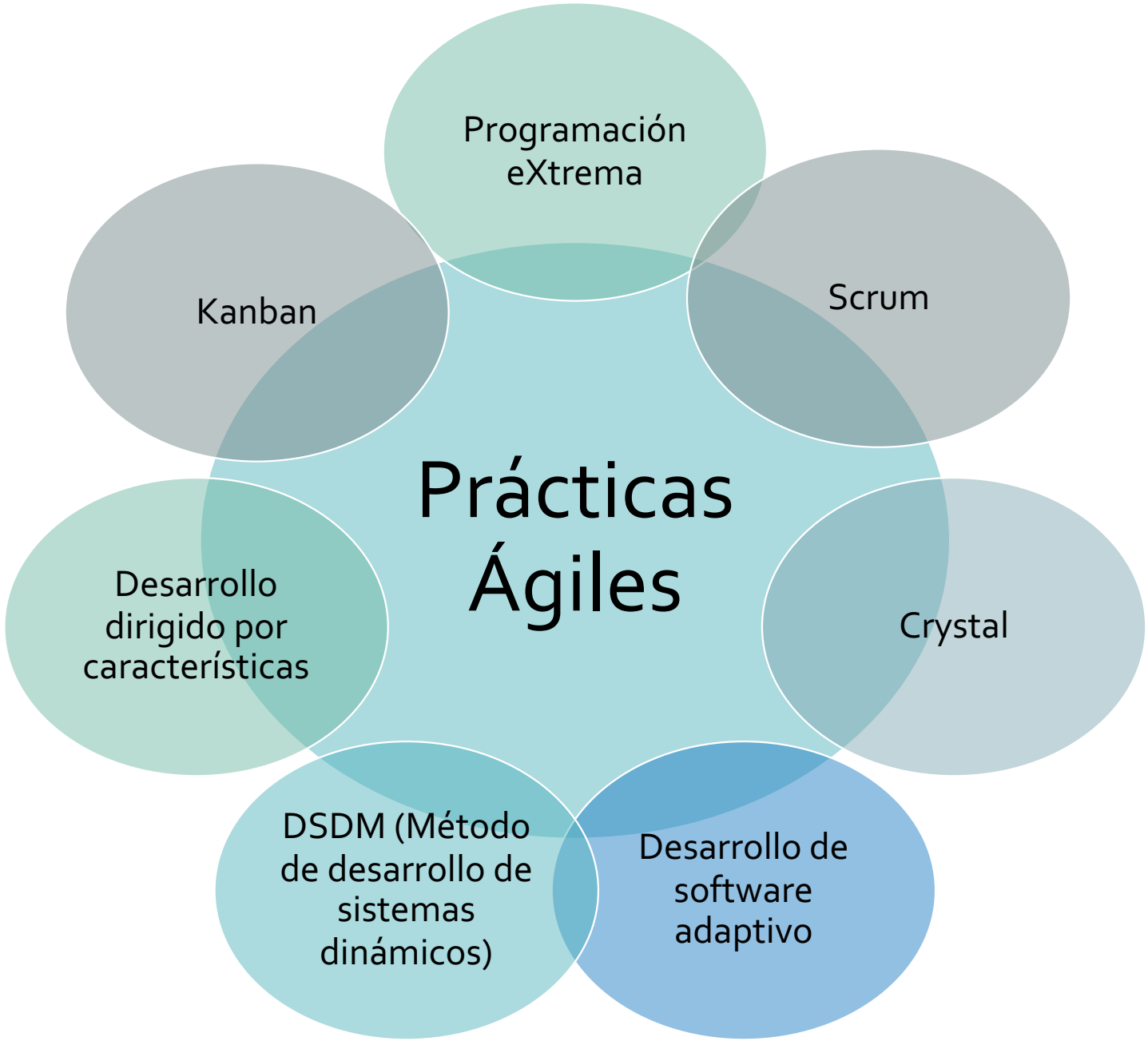
6. ¿Qué tecnologías se hallan disponibles para apoyar el desarrollo y diseño del sistema?
7. ¿Cómo está organizado el equipo de desarrollo?
8. ¿Existen problemas culturales que afecten el desarrollo del sistema?
9. ¿Qué tan buenos son los diseñadores y programadores en el equipo de desarrollo ?
10. ¿El sistema está sujeto a la regulación externa?

Preguntas técnicas, humanas, y organizacionales

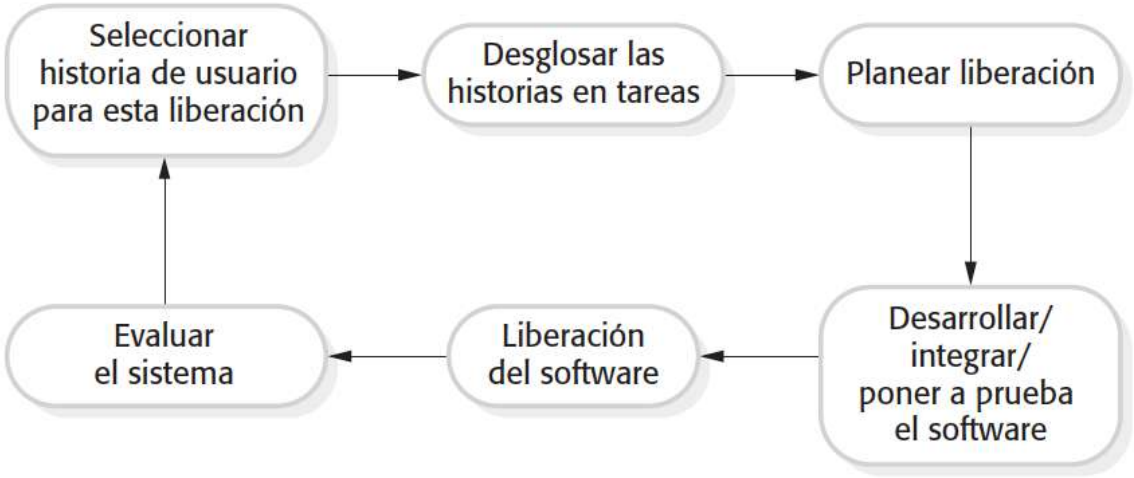
# Factores Humanos

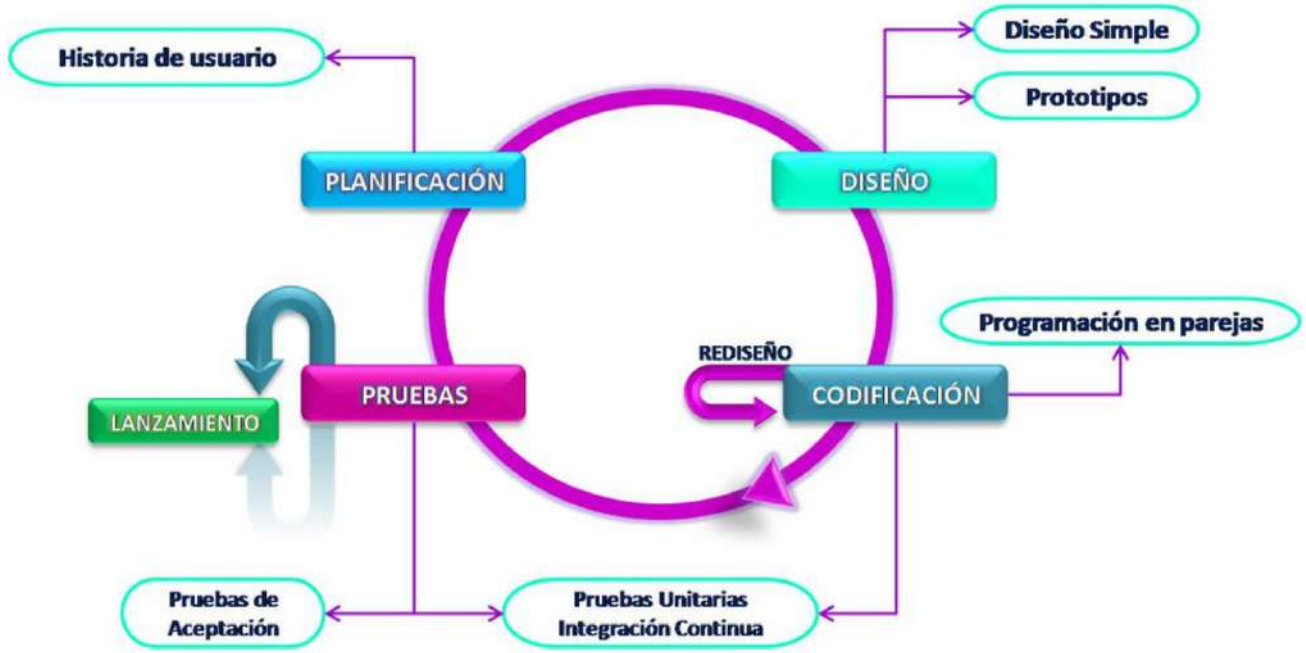
"El desarrollo ágil se centra en los talentos y habilidades de los individuos, y adapta el proceso a personas y equipos específicos."

- Competencia
- Enfoque común
- Colaboración
- Habilidad para tomar decisiones
- Capacidad para resolver problemas difusos
- Confianza y respeto mutuos
- Organización propia




Su diferencia se basa en utilizar diferentes procesos para el desarrollo y la entrega incremental.




# Programación eXtrema (XP)

Muchas versiones actuales de un sistema pueden desarrollarse mediante diferentes programadores, integrarse y ponerse a prueba en un solo día.

Los requisitos se expresan como escenarios (historias de usuarios)

Beck (2000) llevó a niveles extremos las prácticas reconocidas como el desarrollo iterativo.

# Prácticas de programación eXtrema


Los entregables al cliente son aprox. cada dos semanas (1-4)

**Prescripción de medicamentos**

Kate es una médica que quiere prescribir fármacos a un paciente que se atiende en una clínica. El archivo del paciente ya se desplegó en su computadora, de manera que da clic en el campo del medicamento y luego puede seleccionar "medicamento actual", "medicamento nuevo" o "formulario".

Si selecciona "medicamento actual", el sistema le pide comprobar la dosis. Si quiere cambiar la dosis, ingresa la dosis y luego confirma la prescripción.

Si elige "medicamento nuevo", el sistema supone que Kate sabe cuál medicamento prescribir. Ella teclea las primeras letras del nombre del medicamento. El sistema muestra una lista de medicamentos posibles cuyo nombre inicia con dichas letras. Posteriormente elige el fármaco requerido y el sistema responde solicitándole que verifique que el medicamento seleccionado sea el correcto. Ella ingresa la dosis y luego confirma la prescripción.

Si Kate elige "formulario", el sistema muestra un recuadro de búsqueda para el formulario aprobado. Entonces busca el medicamento requerido. Ella selecciona un medicamento y el sistema le pide comprobar que éste sea el correcto. Luego ingresa la dosis y confirma la prescripción.

El sistema siempre verifica que la dosis esté dentro del rango aprobado. Si no es así, le pide a Kate que la modifique.

Después de que ella confirma la prescripción, se desplegará para su verificación. Kate hace clic o en "OK" o en "Cambiar". Si hace clic en "OK", la prescripción se registra en la base de datos de auditoría. Si hace clic en "Cambiar", reingresa al proceso de "prescripción de medicamento".

**Tarea 1: Cambiar dosis del medicamento prescrito**

**Tarea 2: Selección de formulario**

**Tarea 3: Verificación de dosis**

La verificación de dosis es una prevención de seguridad para comprobar que el médico no prescribe una dosis riesgosamente pequeña o grande.

Al usar el ID del formulario para el nombre genérico del medicamento, busca el formulario y recupera las dosis, máxima y mínima, recomendadas.

Verifica la dosis prescrita contra el mínimo y el máximo. Si está fuera de rango, emite un mensaje de error señalando que la dosis es muy alta o muy baja.

Si está dentro del rango, habilita el botón "Confirmar".

# Programación eXtrema (XP)

Las tarjetas de historia de usuario encapsulan las necesidades del cliente.

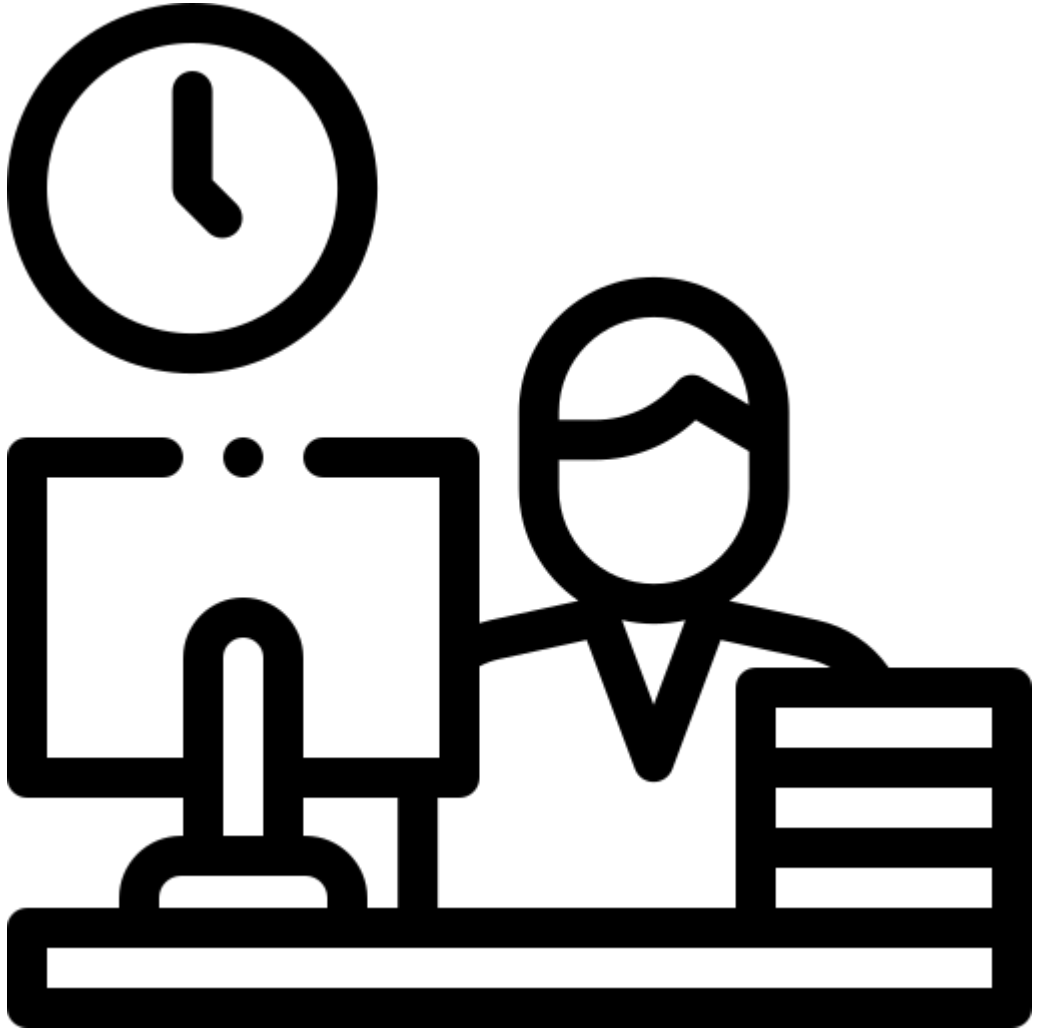
# Programación eXtrema (XP)

**Tarjetas de historia**

- Son las entradas principales al proceso de planeación XP o "juego de planificación".
- Una vez diseñadas las tarjetas de historia, el equipo de desarrollo las descompone en tareas, y estima el esfuerzo y recursos requeridos para implementar cada tarea.
- Intención: Identificar funcionalidad útil que pueda implementarse en aprox. 2 semanas.

# Práctica 3

Tarjetas de Historia del Usuario



# Programación eXtrema (XP) - Pruebas

Proceso de pruebas informal, comparado con las pruebas dirigidas por un plan.

Características de pruebas en XP:

| | | | |
|---|---|---|---|
| Desarrollo de primera prueba | Desarrollo de pruebas incrementales a partir de escenarios | Involucramiento del usuario en el desarrollo y validación de pruebas | Uso de marcos de pruebas automatizadas |

# Programación eXtrema (XP) - Pruebas

- Las pruebas se elaboran antes de escribir el código.
- Se pueden descubrir problemas durante el desarrollo.
- Reduce los problemas de mala interpretación de requerimientos y la interfaz.
- Evita el problema del retraso en la prueba.
- La automatización es esencial para el desarrollo de la primera prueba.

**Prueba 4: Comprobación de dosis**

**Entrada:**

1. Un número en mg que represente una sola dosis del medicamento.
2. Un número que signifique el número de dosis individuales por día.

**Pruebas:**

1. Probar las entradas donde la dosis individual sea correcta, pero la frecuencia muy elevada.
2. Probar las entradas donde la dosis individual sea muy alta y muy baja.
3. Probar las entradas donde la dosis individual × frecuencia sea muy alta y muy baja.
4. Probar las entradas donde la dosis individual × frecuencia esté en el rango permitido.

**Salida:**

OK o mensaje de error que indique que la dosis está fuera del rango de seguridad.

# Programación eXtrema (XP) - Pruebas

Cada tarea genera una o más pruebas de unidad.

El cliente ayuda a desarrollar pruebas de aceptación.

Los componentes de pruebas deben ser independientes, simular el envío de la entrada a probar y verificar que el desarrollo cumple con la especificación de salida.

# Programación eXtrema (XP)

- Este enfoque de pruebas no conduce necesariamente a pruebas minuciosas del programa.
- Los programadores prefieren programar que probar.  Las pruebas son incompletas.
- Algunas pruebas pueden resultar difíciles de escribir de manera incremental.
- Es difícil juzgar la totalidad de un conjunto de pruebas. No ofrece cobertura completa.

# Programación eXtrema (XP)

**Programación en pares**

- Los programadores trabajan en pares para desarrollar el software.
- Los mismos pares no siempre trabajan juntos, de modo que todos los miembros del equipo trabajan entre sí durante el proceso de desarrollo.

# Programación eXtrema (XP)

**Ventajas de Programación en pares**

- Apoya la idea de la propiedad y responsabilidad colectiva.
- Actúa como un proceso de revisión informal.
- Ayuda a la refactorización, que es un proceso de mejoramiento de software.
- Reduce riesgos globales de un proyecto cuando salen miembros del equipo.
- Beneficios observados en estudios (no expertos): Los pares discuten el software antes de desarrollarlo resultando en menos salidas en falso y menos rediseño.

# Administración de un proyecto ágil

Responsabilidad de los administradores de proyectos de software es:

1. dirigir el proyecto, de modo que se entregue a tiempo y con el presupuesto planeado para ello,
2. supervisar el trabajo de los ingenieros, y
3. monitorear el avance en el desarrollo del software.


# Administración de un proyecto ágil - SCRUM

Método ágil que ofrece a un proyecto un marco de referencia administrativo.

Un sprint es una de planeación en el que se valora el trabajo que se va a realizar.

La planeación se basa en priorizar un atraso de trabajo y seleccionar tareas de importancia más alta para un sprint.

# SCRUM Actividades o Reuniones

# SCRUM - Roles

## Product owner (Dueño del producto)

Delinea el producto. Refina items del product backlog.

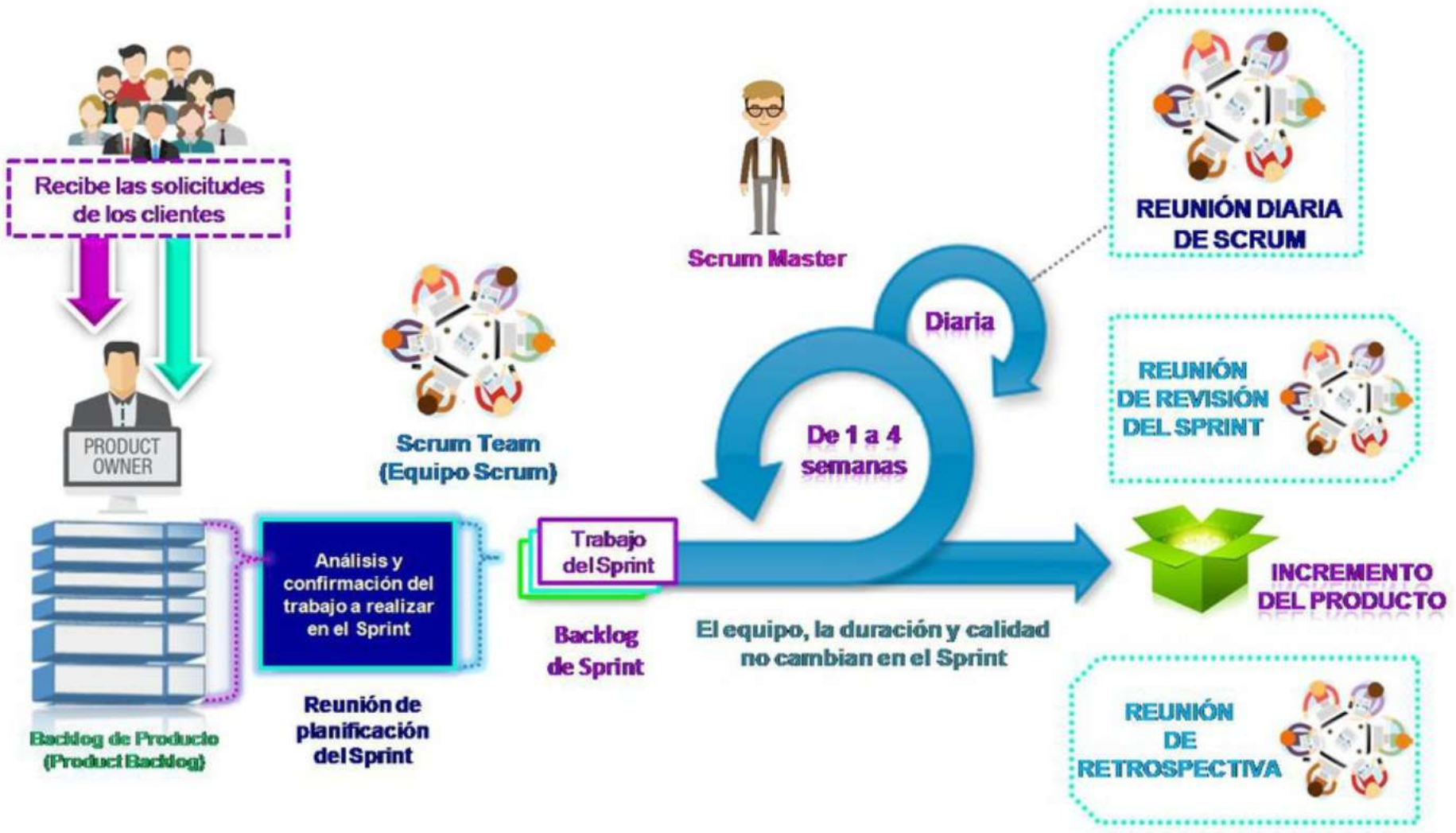
## Scrum Master

Lider servicial. Ayuda al equipo a seguir su proceso.

## Equipo de desarrollo

Determina cuánto trabajo puede ser tomado en un sprint.
Produce un incremento de producto.

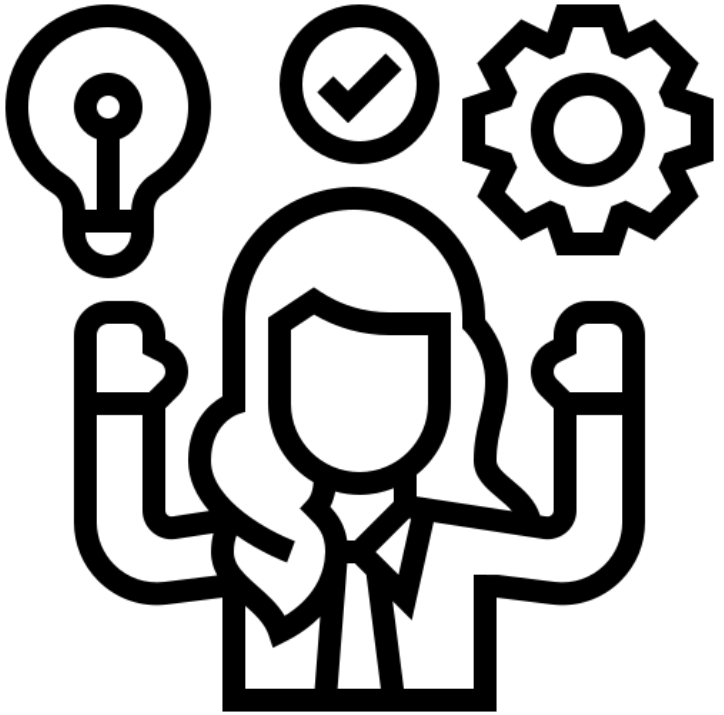
# SCRUM


Recibe las solicitudes de los clientes
PRODUCT OWNER
Backlog de Producto (Product Backlog)
Scrum Team (Equipo Scrum)
Análisis y confirmación del trabajo a realizar en el Sprint
Reunión de planificación del Sprint
Scrum Master
Trabajo del Sprint
Backlog de Sprint
Diaria
De 1 a 4 semanas
El equipo, la duración y calidad no cambian en el Sprint
REUNIÓN DIARIA DE SCRUM
REUNIÓN DE REVISIÓN DEL SPRINT
INCREMENTO DEL PRODUCTO
REUNIÓN DE RETROSPECTIVA

# SCRUM - Roles

**Product Owner**

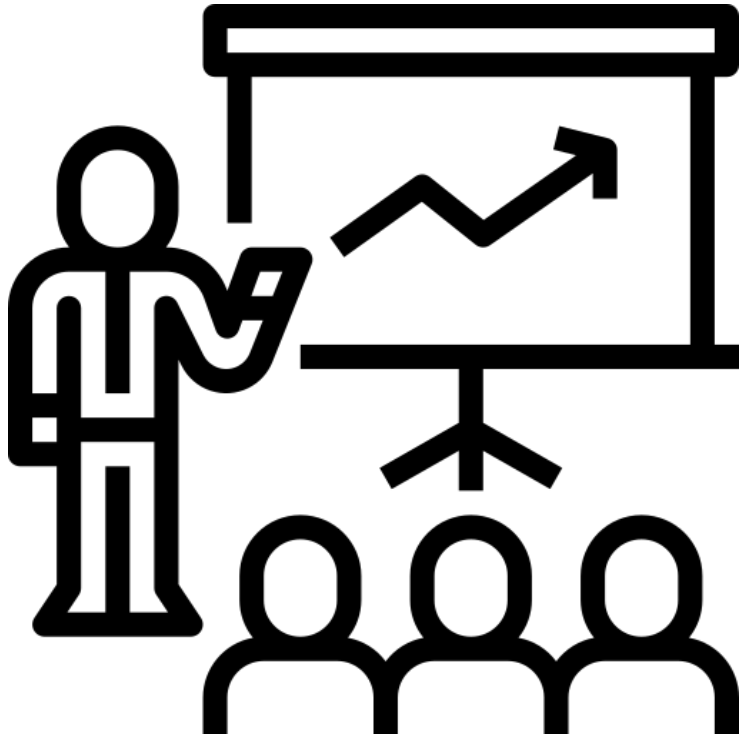
- Comunicador y Negociador
- Visión del negocio
- Requisitos
- Presupuesto
- Gestión de cambio
- Gestión de riesgo



# SCRUM - Roles

**Scrum Master**

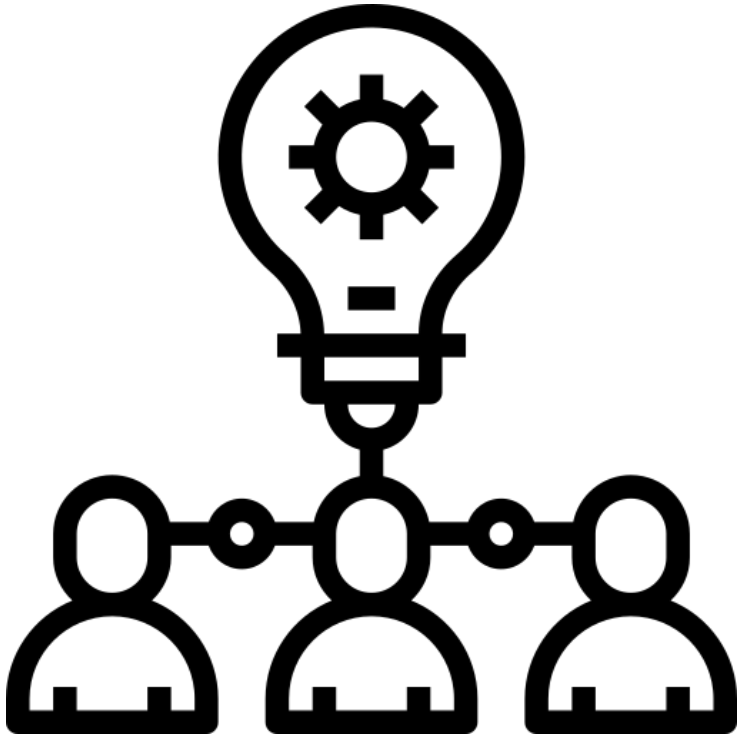
- Líder para el equipo
- Protector, Inspirador y Facilitador
- Gestión de estrés
- Accesible
- Empatía
- Trabajo en equipo



# SCRUM - Roles

**Equipo de Desarrollo**

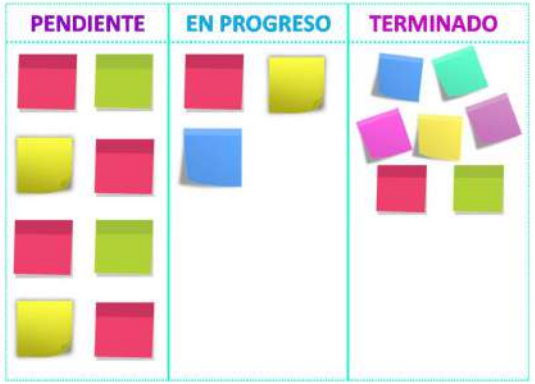
- Realizan el trabajo para entregar un incremento.
- Auto-organizan.
- Grupo interdisciplinario de personas.



# Ventajas del uso exitoso de Scrum

- El producto se desglosa en un conjunto de piezas manejables y comprensibles.
- Los requerimientos inestables no retrasan el progreso.
- Todo el equipo tiene conocimiento de todo, lo que mejora la comunicación entre el equipo.
- Los clientes observan la entrega a tiempo.
- Se establece confianza entre clientes y desarrolladores.

# KANBAN (Tarjetas visuales)

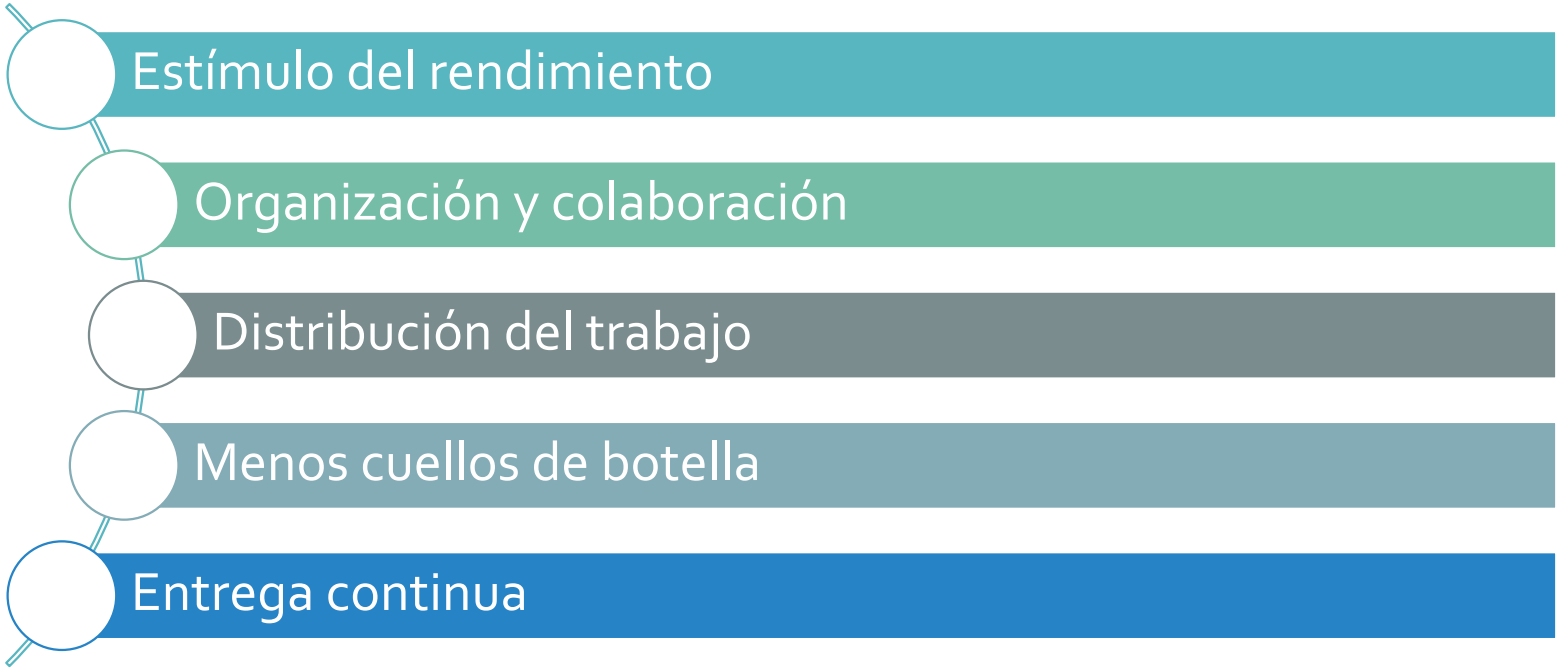



- Técnica creada por Toyota
- Método de administración de tareas y flujo de un trabajo.
- Consiste en un tablero
  - Lista de tareas:
    - Pendiente: Lista de requerimientos -> Listo para empezar
    - En proceso: Diseño/Análisis, En desarrollo, Pruebas, Despliegue
    - Terminada

https://kanbanize.com/es/recursos-de-kanban/software-kanban/ejemplos-de-tableros-kanban

# KANBAN (Tarjetas visuales)

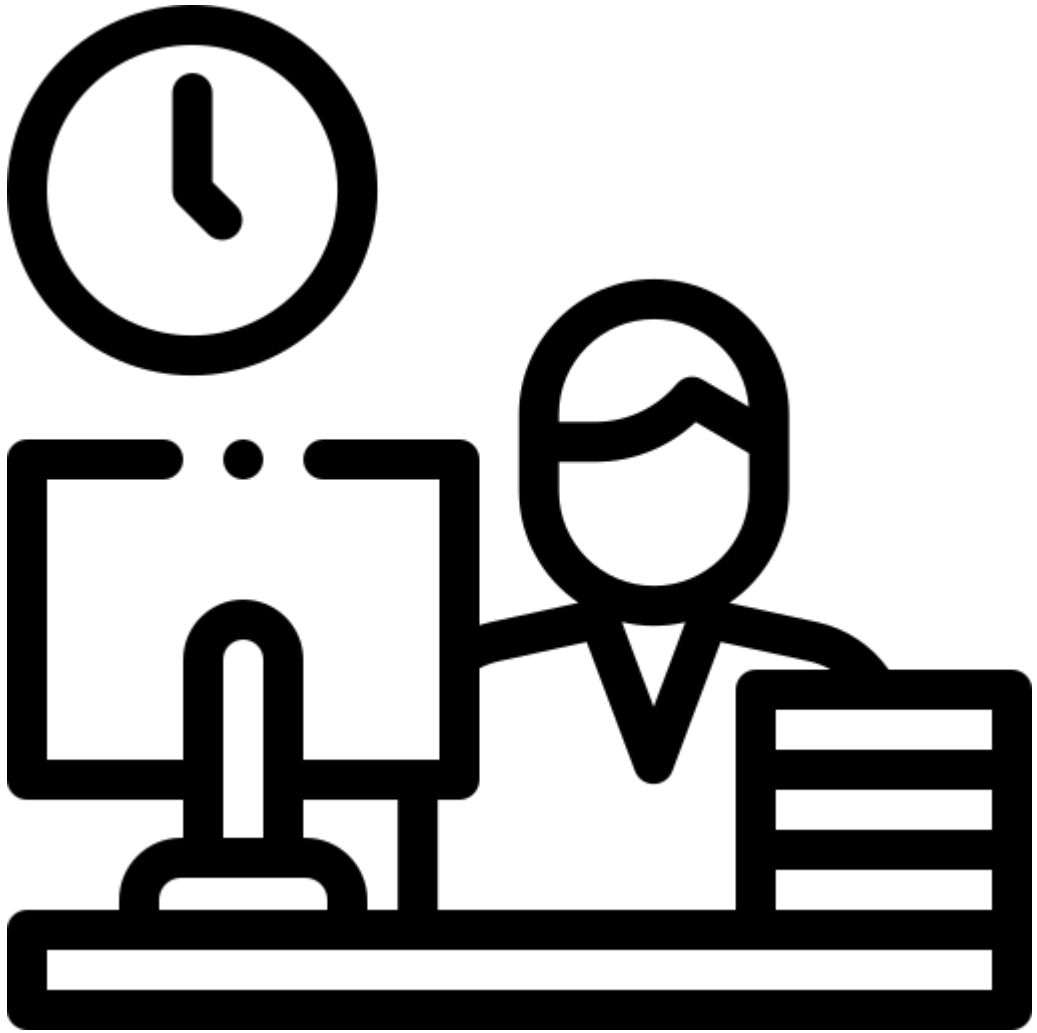
## Beneficios




https://www.youtube.com/watch?v=PMaozjOpYE8

# Práctica 4

Planificación ágil: Tablero Kanban



| Metodología ágil | Metodología tradicional |
| --- | --- |
| Flexibilidad ante los cambios de forma moderada a rápida. | Rigidez ante los cambios, de manera lenta a moderada. |
| Los clientes hacen parte del equipo de desarrollo | Los clientes interactúan con el equipo de desarrollo mediante reuniones. |
| Grupos pequeños (10 personas) en un mismo sitio. | Grupos de gran tamaño y distribuidos. |
| Menor dependencia de la arquitectura de software. | Dependencia de la arquitectura de software mediante modelos. |
| Continua retroalimentación acortando el tiempo de entrega. | Poca retroalimentación, lo que extiende el tiempo de entrega. |

# Diferencias entre metodologías ágiles y tradicionales

| Metodología ágil | Metodología tradicional |
| --- | --- |
| Diversidad de roles. | Mínimos roles. |
| Basada en heurísticas a partir de prácticas de producción de código. | Basada en normas de estándares de desarrollo. |
| Procesos menos controlados, pocas políticas y normas. | Procesos muy controlados por políticas y normas. |
| Capacidad de respuesta ante los cambios. | Seguimiento estricto del plan inicial de desarrollo. |

# Diferencias entre metodologías ágiles y tradicionales

# Principios que guían la práctica

Principios fundamentales

Principios que guían toda actividad estructural

# Principios que guían la práctica

1. Divide y vencerás
2. Entender el uso de la abstracción
3. Buscar la coherencia
4. Centrarse en la transferencia de información

# Principios que guían la práctica

5. Construir software que tenga modularidad eficaz
6. Buscar patrones
7. Cuando sea posible, representar el problema y su solución desde varias perspectivas diferentes
8. Tener en mente que alguien dará mantenimiento al software

1. Escuchar.
2. Antes de comunicarse, prepararse.
3. Alguien debe facilitar la actividad.
4. Es mejor la comunicación cara a cara.
5. Tomar notas y documentar las decisiones.
6. Perseguir la colaboración.
7. Permanecer centrado; hacer módulos con la discusión.
8. Si algo no está claro, hacer un dibujo.
9. ¡Avanzar! Una vez que se acuerde algo, avanzar. Si no es posible ponerse de acuerdo, avanzar. Si una característica o función no está clara o no puede aclararse en el momento, avanzar.
10. La negociación no es un concurso o un juego. Funciona mejor cuando dos partes ganan.

# Principios que guían toda actividad estructural - Comunicación

1. Entender el alcance del proyecto.
2. Involucrar en la actividad de planeación a los participantes del software.
3. Reconocer que la planeación es iterativa.
4. Estimar con base en lo que se sabe.
5. Al definir el plan, tomar en cuenta los riesgos.
6. Ser realista.
7. Ajustar la granularidad cuando se defina el plan.
8. Definir cómo se trata de asegurar la calidad.
9. Describir cómo se busca manejar el cambio.
10. Dar seguimiento al plan con frecuencia y hacer los ajustes que se requieran.

# Principios que guían toda actividad estructural - Planeación

1. El equipo de software tiene como objetivo principal elaborar software, no crear modelos.
2. Viajar ligero, no crear más modelos de los necesarios.
3. Tratar de producir el modelo más sencillo que describa al problema o al software.
4. Construir modelos susceptibles al cambio.
5. Ser capaz de enunciar un propósito explícito para cada modelo que se cree.

# Principios que guían toda actividad estructural - Modelado

6. Adaptar los modelos que se desarrollan al sistema en cuestión.
7. Tratar de construir modelos útiles, pero olvidarse de elaborar modelos perfectos.
8. No ser dogmático respecto de la sintaxis del modelo. Si se tiene éxito para comunicar contenido, la representación es secundaria.
9. Si su instinto dice que un modelo no es el correcto a pesar de que se vea bien en el papel, hay razones para estar preocupado.
10. Obtener retroalimentación tan pronto como sea posible.

# Principios que guían toda actividad estructural - Modelado

1. Debe representarse y entenderse el dominio de información de un problema.
2. Debe definirse las funciones que realizará el software.
3. Debe representarse el comportamiento del software (como consecuencia de eventos externos).
4. Los modelos que representen información, función y comportamiento deben dividirse de manera que revelen los detalles en forma estratificada.
5. El trabajo de análisis debe avanzar de la información esencial hacia la implementación en detalle.

## Principios que guían toda actividad estructural – Modelado Requerimientos

1. El diseño debe poderse rastrear hasta el modelado de requerimientos.
2. Siempre tomar en cuenta la arquitectura del sistema que se va a construir.
3. El diseño de los datos es tan importante como el de las funciones de procesamiento.
4. Las interfaces (tanto internas como externas) deben diseñarse con cuidado.
5. El diseño de la interfaz de usuario debe ajustarse a las necesidades del usuario final. Sin embargo, en todo caso debe resaltar la facilidad de uso.

## Principios que guían toda actividad estructural – Modelado Diseño

6. El diseño en el nivel de componentes debe tener interdependencia funcional.
7. Los componentes deben estar acoplados con holgura entre sí y con el ambiente externo.
8. Las representaciones del diseño (modelos) deben entenderse con facilidad.
9. El diseño debe desarrollarse en forma iterativa. El diseñador debe buscar más sencillez en cada iteración.

## Principios que guían toda actividad estructural – Modelado Diseño

1. Entender el problema que se trata de resolver.
2. Comprender los principios y conceptos básicos del diseño.
3. Elegir un lenguaje de programación que satisfaga las necesidades del software a construir y en el ambiente en el que operará.
4. Seleccionar un ambiente de programación que disponga de herramientas que hagan más fácil su trabajo.
5. Crear un conjunto de pruebas unitarias que se aplicarán una vez que se haya terminado el componente a codificar.

## Principios que guían toda actividad estructural – Construcción
## Codificación - Preparación

1. Restringir sus algoritmos por medio del uso de programación estructurada.
2. Tomar en consideración el uso de programación por parejas.
3. Seleccionar estructuras de datos que satisfagan las necesidades del diseño.
4. Entender la arquitectura del software y crear interfaces que son congruentes con ella.
5. Mantener la lógica condicional tan sencilla como sea posible.

## Principios que guían toda actividad estructural – Construcción Codificación - Programación

6. Crear lazos anidados en forma tal que se puedan probar con facilidad.
7. Seleccionar nombres significativos para las variables y seguir otros estándares locales de codificación.
8. Escribir código que se documente a sí mismo.
9. Crear una imagen visual (respetar sangrado) que facilite la comprensión.

## Principios que guían toda actividad estructural – Construcción Codificación - Programación

1. Realizar el recorrido del código cuando sea apropiado.
2. Llevar a cabo pruebas unitarias y corregir los errores que se detecten.
3. Rediseñar el código.

## Principios que guían toda actividad estructural – Construcción Codificación - Validación

1. Todas las pruebas deben poder rastrearse hasta los requerimientos del cliente.
2. Las pruebas deben planearse mucho antes de que den comienzo.
3. El principio de Pareto se aplica a las pruebas de software.
4. Las pruebas deben comenzar "en lo pequeño" y avanzar hacia "lo grande".
5. No son posibles las pruebas exhaustivas.

## Principios que guían toda actividad estructural – Construcción Pruebas

1. Deben manejarse las expectativas de los clientes.
2. Debe ensamblarse y probarse el paquete completo que se entregará.
3. Antes de entregar el software, debe establecerse un régimen de apoyo.
4. Se deben proporcionar a los usuarios finales materiales de aprendizaje apropiados.
5. El software defectuoso debe corregirse primero y después entregarse.

# Principios que guían toda actividad estructural - Despliegue

# Bibliografía y Recursos útiles


- Sommerville, I. (2011). Ingeniería de Software. 9na Edición. (Cap. 3).
- Pressman, R. (2010). Ingeniería del Software. Un Enfoque Práctico. 7ma Edición. (Cap. 3, Cap. 4).
- http://www.extremeprogramming.org
- https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Spanish-SouthAmerican.pdf
- https://www.youtube.com/watch?v=giyNqQmnDek
- https://www.youtube.com/watch?v=8qa6io8wHQA
- https://kanbanize.com/es/recursos-de-kanban/software-kanban/ejemplos-de-tableros-kanban