FIP

Carrera Programador full-stack

Ciclos de vida de un proyecto

¿Qué es?

El ciclo de vida es el conjunto de fases por las que pasa el sistema que se está desarrollando desde que nace la idea inicial hasta que el software es retirado o reemplazado (muere). También se denomina a veces paradigma.

¿Para qué sirve un ciclo de vida?

Determina el orden de las fases del proceso de software:

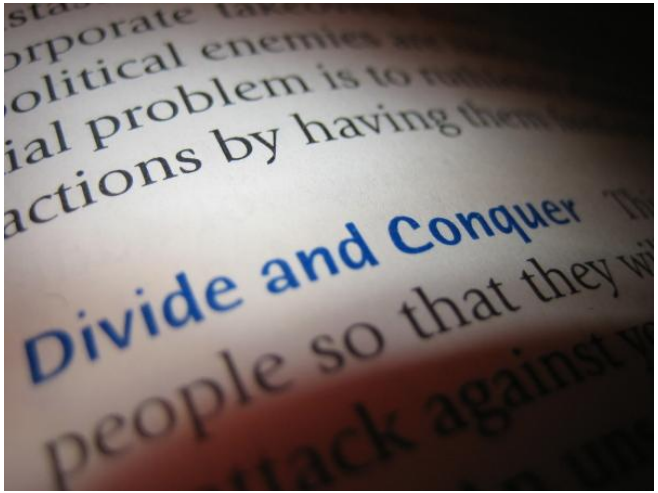
- Establece los criterios de transición para pasar de una fase a la siguiente.
- Define las entradas y salidas de cada fase.
- Describe los estados por los que pasa el producto.
- Describe las actividades a realizar para transformar el producto.
- Define un esquema que sirve como base para planificar, organizar, coordinar, desarrollar…

Preguntas

- ¿Qué significa cada paso, recuerdan?
- ¿Qué pasa luego del paso 6?
- ¿Cuánto creen que dura o debería durar cada etapa?

Fases

La idea general del ciclo de vida es que la mejor forma de administrar los esfuerzos para llevar a cabo un proyecto es tratar dichos esfuerzos como un proyecto por separado. Esos proyectos separados podríamos llamarlos fases.



¿Qué son las fases/etapas?

Fases: son un conjunto de actividades relacionadas con un objetivo en el desarrollo del proyecto. Se construyen agrupando tareas que pueden compartir un tramo determinado del tiempo de vida de un proyecto. La agrupación temporal de tareas impone requisitos temporales correspondientes a la asignación de recursos (humanos, financieros o materiales).

Según el modelo de ciclo de vida, la sucesión de fases puede ampliarse con bucles de realimentación, de manera que lo que conceptualmente se considera una misma fase se pueda ejecutar más de una vez a lo largo de un proyecto, recibiendo en cada pasada de ejecución aportaciones a los resultados intermedios que se van produciendo (realimentación).

Entregables/Artefactos

- Son los productos intermedios que generan las fases.
- Pueden ser materiales o inmateriales (documentos, software).
- Permiten evaluar la marcha del proyecto mediante comprobaciones de su adecuación o no a los requisitos funcionales y de condiciones de realización previamente establecidos.

Actividades genéricas

	Tarea	Artefacto
Especificación	Lo que el sistema debería hacer y sus restricciones de desarrollo.	Documentos: Obtenidos por reunión con el cliente. Alcance del proyecto, especificación y limitaciones.
Desarrollo	Producción del sistema software.	Software o documentos: Creación de prototipo.
Validación	Comprobar que el sistema es lo que el cliente quiere.	Documentos: Verificar que el prototipo haga lo que se pedía.
Evolución	Cambiar el software en respuesta a las demandas de cambio.	Software:Mantenimiento del software. Un poco de todo lo anterior.

¿Qué es una técnica?

La transición de una fase a otra dentro del ciclo de vida de un proyecto generalmente está definida por alguna forma de transferencia técnica.

Técnica: Es un método que aplica herramientas y reglas específicas para completar una o más fases del ciclo de vida del desarrollo de sistemas (Whitten, et al, 1996). Las técnicas, por lo regular, sólo son aplicables a una parte del ciclo de vida, por lo que no pueden sustituirlo en su totalidad.

¿Qué es una metodología?

Es una versión amplia y detallada de un ciclo de vida del desarrollo de sistemas que incluye:

1. Tareas paso a paso por cada fase.

2. Funciones individuales, y en grupo desempeñadas en cada tarea.

3. Productos resultantes y normas de calidad para cada tarea.

4. Técnicas de desarrollo.

¿Porqué necesitamos una metodología?

¿Porqué necesitamos una metodología?

El desarrollo de software puede ser un sector especialmente complejo, sobre todo cuando se trata de grandes aplicativos y equipos de trabajo. Ponerse a desarrollar un producto sin una metodología clara desembocará en un proceso aún más complejo, que conducirá a problemas, retrasos, errores y, en definitiva, un mal resultado final.

Resumen

No existe una única manera para definir el ciclo de vida ideal de un proyecto. Algunas organizaciones establecen políticas que estandarizan todos los proyectos con un ciclo de vida único, mientras que otras permiten al equipo de dirección del proyecto elegir el ciclo de vida más apropiado para el proyecto.

El trabajo con una metodología de desarrollo de software permite **reducir el nivel de dificultad, organizar las tareas, agilizar el proceso y mejorar el resultado final de las aplicaciones a desarrollar.**

Metodologías de desarrollo

Como toda disciplina, el desarrollo de software madura y con el tiempo genera nuevos mecanismos para su propia optimización. De estos esfuerzos surgen las metodologías de desarrollo de software.

Una metodología brinda al equipo de trabajo un marco para construir aplicaciones de manera eficiente y rigurosa, garantizando un producto cercano al esperado. Si no se desarrolla a partir de una metodología, el resultado final será impredecible y no se podrá controlar el avance del proyecto.

Metodologías de desarrollo, ¿Qué son?

Las metodologías de desarrollo de software son un conjunto de técnicas y métodos organizativos que se aplican para diseñar soluciones de software informático. El objetivo de las distintas metodologías es el de intentar organizar los equipos de trabajo para que estos desarrollen las funciones de un programa de la mejor manera posible.

Cuando se trata de desarrollar productos o soluciones para un cliente o mercado concreto, es necesario tener en cuenta factores como los costes, la planificación, la dificultad, el equipo de trabajo disponible, los lenguajes utilizados, etc. Todos ellos se engloban en una metodología de desarrollo que permite organizar el trabajo de la forma más ordenada posible.

¿Qué tipos de metodologías de desarrollo de software existen?

En la actualidad se pueden diferenciar **dos grandes grupos** de metodologías de desarrollo de software: las ágiles y las tradicionales. Los sistemas tradicionales se centran en la planificación proactiva donde factores como el costo, el alcance y el tiempo son importantes, pero la gestión ágil de proyectos prioriza el trabajo en equipo, la colaboración con los clientes y la flexibilidad.

Metodologías de desarrollo de software

Metodologías Tradicionales

1 PLANTEA-MIENTO → 2 ANÁLISIS → 3 DISEÑO → 4 PROGRA-MACIÓN → 5 PRUEBAS → PUESTA EN MARCHA

Metodologías Ágiles

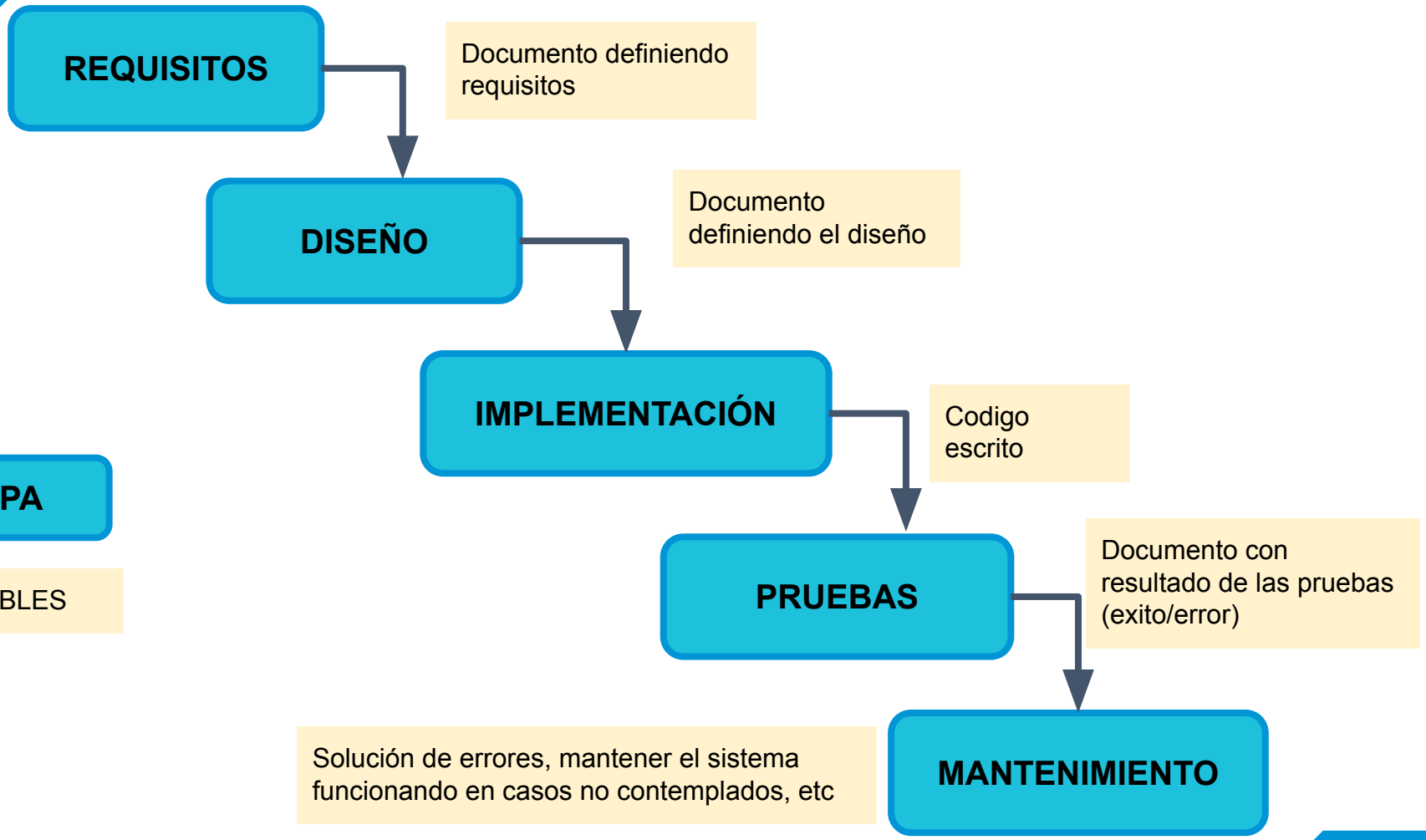
1 PLANTEA-MIENTO → 2 REQUERI-MIENTOS PRIORIZADOS → ITERACIÓN → PUESTA EN MARCHA

Metodologías de desarrollo de software tradicionales

La organización del trabajo de las metodologías tradicionales es lineal, es decir, las etapas se suceden una tras otra y no se puede empezar la siguiente sin terminar la anterior. Tampoco se puede volver hacia atrás una vez se ha cambiado de etapa. Estas metodologías, **no se adaptan nada bien a los cambios**, y el mundo actual cambia constantemente. Las principales metodologías tradicionales o clásicas son:

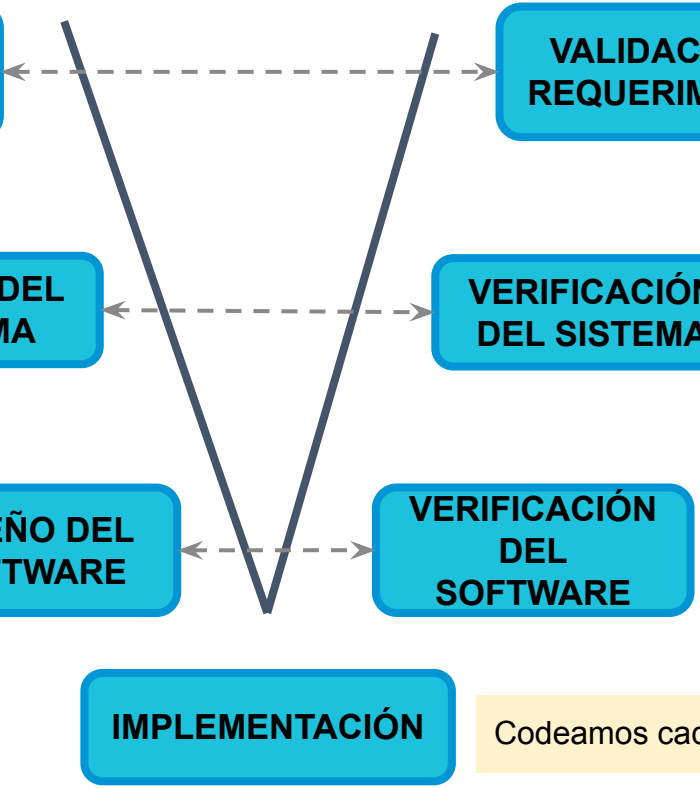
Modelo en cascada

Es una metodología en la que las etapas se organizan de arriba a abajo, de ahí el nombre. Se desarrollan las diferentes funciones en etapas diferenciadas y obedeciendo un riguroso orden. Antes de cada etapa se debe revisar el producto para ver si está listo para pasar a la siguiente fase. Los requisitos y especificaciones iniciales no están predispuestos para cambiarse, por lo que no se pueden ver los resultados hasta que el proyecto ya esté bastante avanzado.


REQUISITOS
Documento definiendo requisitos
DISEÑO
Documento definiendo el diseño
IMPLEMENTACIÓN
Codigo escrito
PRUEBAS
Documento con resultado de las pruebas (exito/error)
MANTENIMIENTO
Solución de errores, mantener el sistema funcionando en casos no contemplados, etc
ETAPA
ENTREGABLES

Modelo V

El modelo en V se desarrolló para terminar con algunos de los problemas que se vieron utilizando el enfoque de cascada tradicional. La parte izquierda de la v representa la descomposición de los requisitos y la creación de las especificaciones del sistema. El lado derecho de la v representa la integración de partes y su verificación. V significa “Validación y Verificación”. Los defectos estaban siendo encontrados demasiado tarde en el ciclo de vida, ya que las pruebas no se introducían hasta el final del proyecto.



Qué quiere el cliente?

INGENIERÍA DE REQUISITOS

VALIDACIÓN DE REQUERIMIENTOS

¿Qué era lo que el cliente quería?

Dividimos el sistema en diferentes partes mas chicas

DISEÑO DEL SISTEMA

VERIFICACIÓN DEL SISTEMA

¿El código anda bien con otros códigos que interactúan todos en conjunto?

Diseñamos cada parte

DISEÑO DEL SOFTWARE

VERIFICACIÓN DEL SOFTWARE

Este código solo hace lo que queríamos que hiciera?

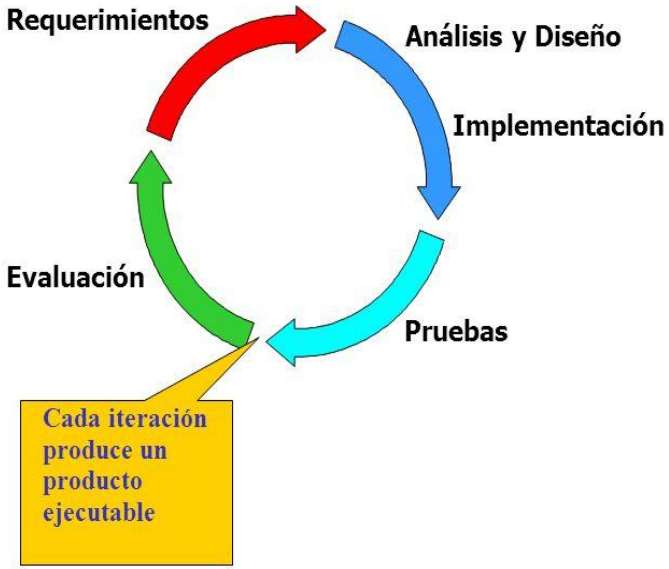
IMPLEMENTACIÓN

Codeamos cada parte

ETAPA

ENTREGABLES

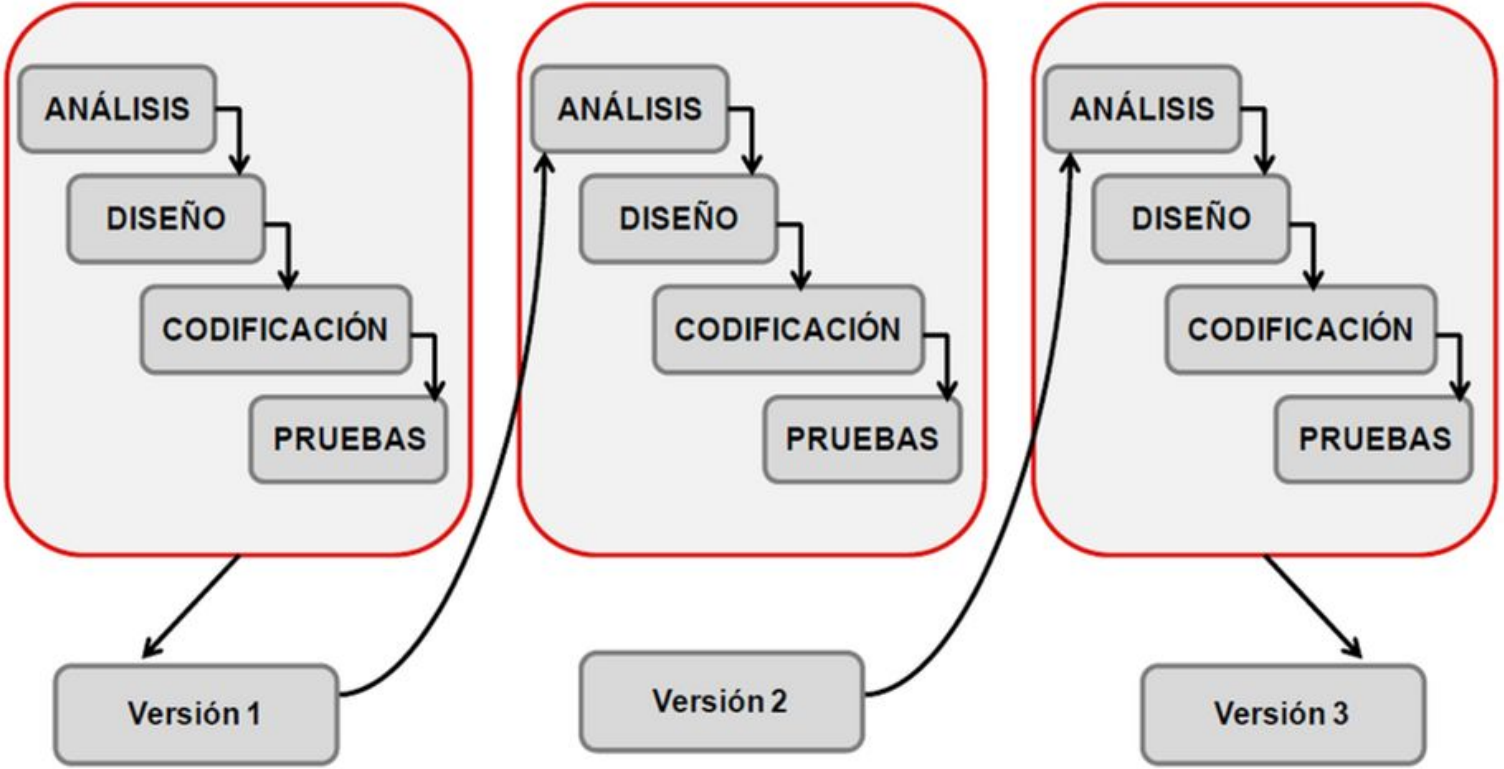
Modelo Iterativo



Consiste en la iteración de varios ciclos de vida en cascada. Al final de cada iteración se le entrega al cliente una versión mejorada o con mayores funcionalidades del producto. El cliente es quien, después de cada iteración, evalúa el producto y lo corrige o propone mejoras. Estas iteraciones se repetirán hasta obtener un producto que satisfaga las necesidades del cliente.

Este modelo se suele utilizar en proyectos en los que los requisitos no están claros por parte del usuario, por lo que se hace necesaria la creación de distintos prototipos para presentarlos y conseguir la conformidad del cliente.

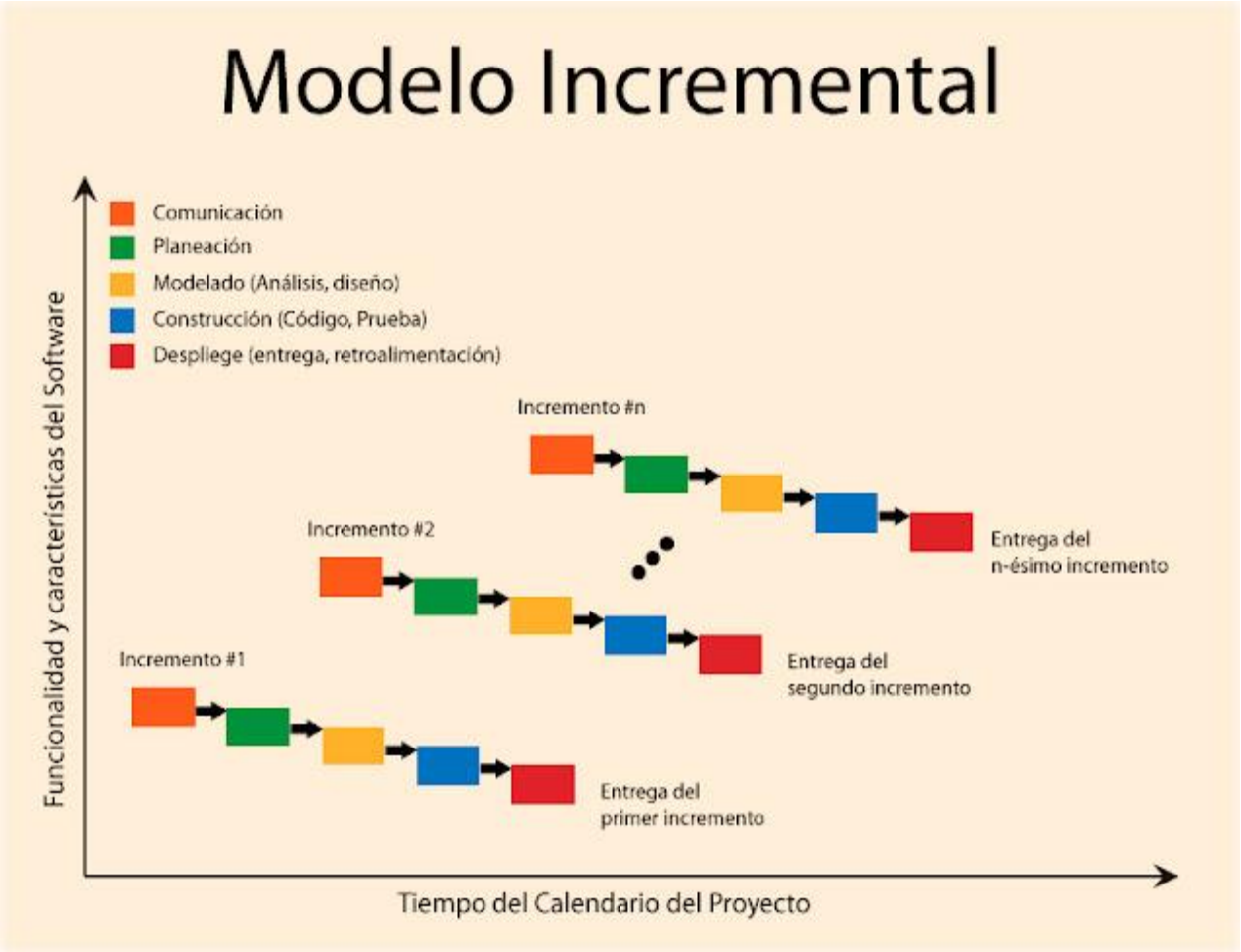
Modelo Iterativo



Modelo iterativo ventajas-desventajas

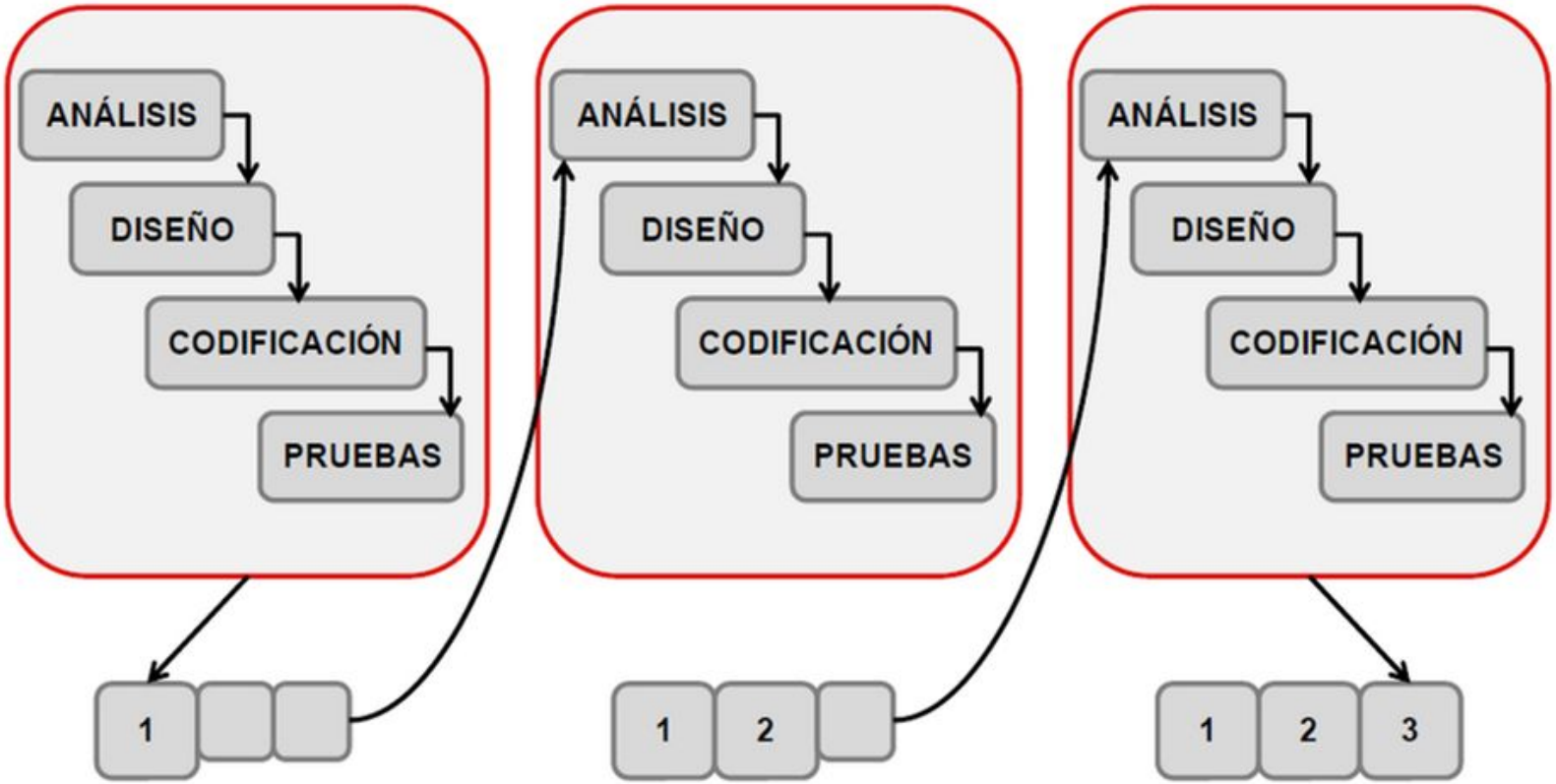
- No hacen falta requisitos definidos al inicio, se pueden refinar en cada iteración.
- Desarrollo en pequeños ciclos
- Permite gestionar mejor los riesgos, gestionar mejor las entregas.

- No necesita requisitos definidos al principio y eso puede causar problemas de arquitectura
- El impacto sobre el proceso de contratación (los requerimientos pueden cambiar a medida que se desarrollan los incrementos)



El modelo incremental combina elementos del modelo en cascada con la filosofía interactiva de construcción de prototipos. Se basa en la filosofía de construir incrementando las funcionalidades del programa. Este modelo aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software.

Modelo Incremental

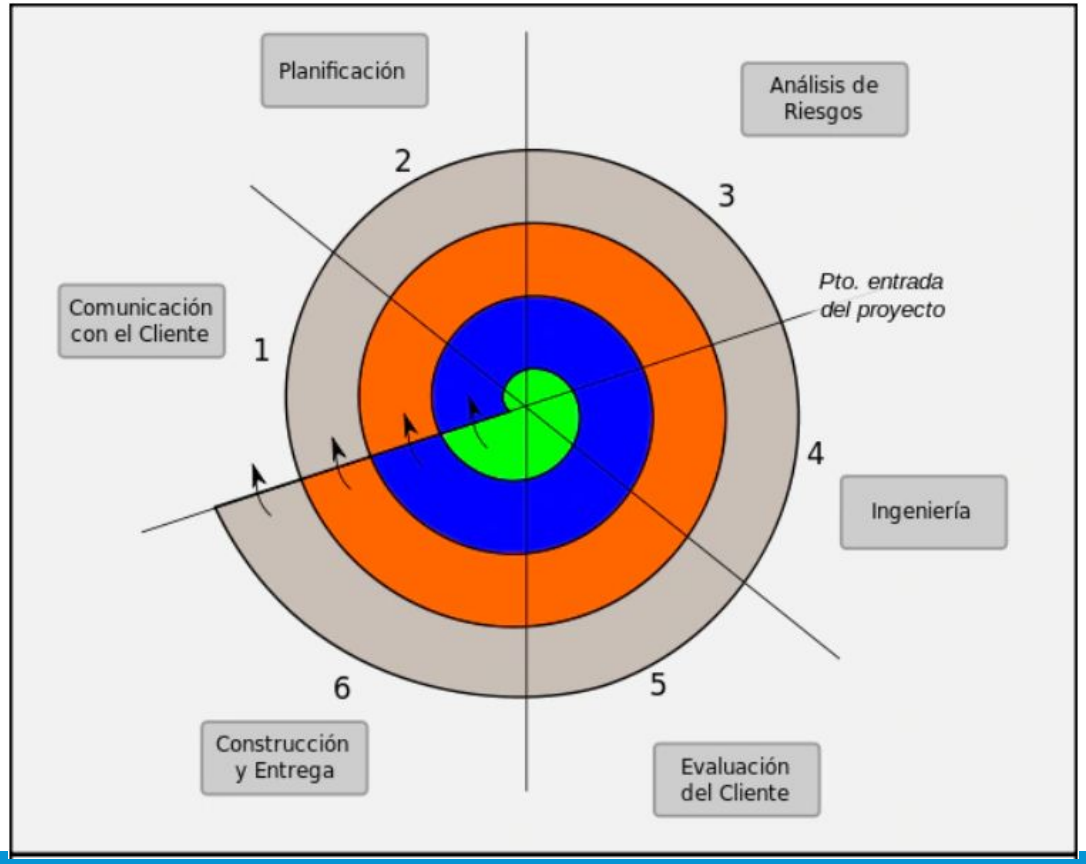


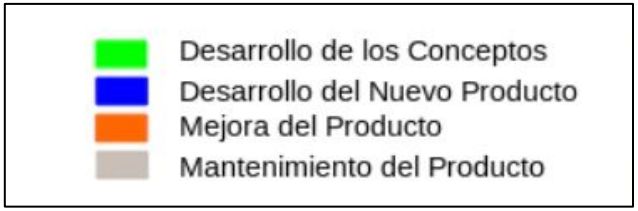
Modelo incremental-ventajas

- Veloz generación de software operativo.
- Flexibilidad que reduce el coste en el cambio de alcance y requisitos.
- Facilidad de probar y depurar en una iteración más pequeña.
- Es más fácil gestionar riesgos.
- Cada iteración es un hito gestionado fácilmente.

- Cada fase de una iteración es rígida y no se superponen con otras.
- Posibles problemas de arquitectura.
- Se requiere una experiencia importante para definir los incrementos y distribuir en ellos las tareas de forma proporcionada.

Modelo en espiral





Las actividades de este modelo se conforman en una espiral, cada bucle representa un conjunto de actividades. Las actividades no están fijadas a priori, sino que las siguientes se eligen en función del análisis de riesgos, comenzando por el bucle anterior.

Al ser un modelo de ciclo de vida orientado a la gestión de riesgos, uno de los aspectos fundamentales de su éxito radica en que el equipo que lo aplique tenga la necesaria experiencia y habilidad para detectar y catalogar correctamente riesgos.

Modelo en espiral

El proceso empieza en la posición central. Desde allí se mueve en el sentido de las agujas del reloj.

Para cada ciclo habrá cuatro actividades:

1. Determinar o fijar objetivos:

- Fijar también los productos definidos a obtener: requerimientos, especificación, manual de usuario.
- Fijar las restricciones.
- Identificar riesgos del proyecto y estrategias alternativas para evitarlos.
- Hay una cosa que solo se hace una vez: planificación inicial o previa

2. Análisis del riesgo:

- Estudiar todos los riesgos potenciales y se seleccionan una o varias alternativas propuestas para reducir o eliminar los riesgos

Modelo en espiral

3. Desarrollar, verificar y validar (probar):

- Tareas de la actividad propia y de prueba.
- Análisis de alternativas e identificación de resolución de riesgos.
- Dependiendo del resultado de la evaluación de riesgos, se elige un modelo para el desarrollo, que puede ser cualquiera de los otros existentes, como formal, evolutivo, cascada, etc. Así, por ejemplo, si los riesgos de la interfaz de usuario son dominantes, un modelo de desarrollo apropiado podría ser la construcción de prototipos evolutivos.

4. Planificar:

- Revisar todo lo que se ha llevado a cabo, evaluando y decidiendo si se continúa con las fases siguientes y planificando la próxima actividad.

Modelo espiral ventajas-desventajas

- La funcionalidad adicional o los cambios se pueden hacer en una etapa posterior.
- Fácil estimación del coste, ya que la construcción del prototipo se hace en pequeños fragmentos.
- El desarrollo continuo o repetido ayuda en la gestión de riesgos
- Rápido desarrollo, las características se añaden sistemáticamente.

- Riesgo de no cumplir con la planificación o el presupuesto.
- Funciona mejor para proyectos grandes.
- Debe ser seguido estrictamente.
- Más documentación al tener fases intermedias.
- No es aconsejable para proyectos pequeños, la ratio coste beneficio no es rentable.

Modelo de prototipos

También conocido como modelo de desarrollo evolutivo, se inicia con la definición de los objetivos globales para el software, luego se identifican los requisitos conocidos y las áreas donde es necesaria más definición.
Se utiliza para dar al usuario una vista preliminar de parte del software. Es prueba y error ya que si al usuario no le gusta una parte del prototipo se debe corregir el error hasta que quede satisfecho.
El prototipo debe ser construido en poco tiempo y no se debe utilizar mucho dinero pues a partir de que este sea aprobado nosotros podemos iniciar el verdadero desarrollo del software. El construir el prototipo nos asegura que nuestro software sea de mejor calidad, además de que su interfaz sea de agrado para el usuario.

Tipos de modelos de prototipos

- **Modelo de Prototipos rápido:** Metodología de diseño que desarrolla rápidamente nuevos diseños, los evalúa y prescinde del prototipo cuando el próximo diseño es desarrollado mediante un nuevo prototipo.
- **Modelo de Prototipos reutilizables:** También conocido como "Evolutionary Prototyping"; no se pierde el esfuerzo efectuado en la construcción del prototipo pues sus partes o el conjunto pueden ser utilizados para construir el producto real. Mayormente es utilizado en el desarrollo de software, si bien determinados productos de hardware pueden hacer uso del prototipo como la base del diseño de moldes en la fabricación con plásticos o en el diseño de carrocerías de automóviles.
- **Modelo de Prototipos Modular:** También conocido como Prototipado Incremental (Incremental prototyping); se añaden nuevos elementos sobre el prototipo a medida que el ciclo de diseño progresa.
- **Modelo de Prototipos Horizontal:** El prototipo cubre un amplio número de aspectos y funciones pero la mayoría no son operativas. Resulta muy útil para evaluar el alcance del producto, pero no su uso real.
- **Modelo de Prototipos Vertical:** El prototipo cubre sólo un pequeño número de funciones operativas. Resulta muy útil para evaluar el uso real sobre una pequeña parte del producto.
- **Modelo de Prototipos de Baja-fidelidad:** El prototipo se implementa con papel y lápiz, emulando la función del producto real sin mostrar el aspecto real del mismo. Resulta muy útil para realizar tests baratos.
- **Modelo de Prototipos de Alta-fidelidad:** El prototipo se implementa de la forma más cercana posible al diseño real en términos de aspecto, impresiones, interacción y tiempo.

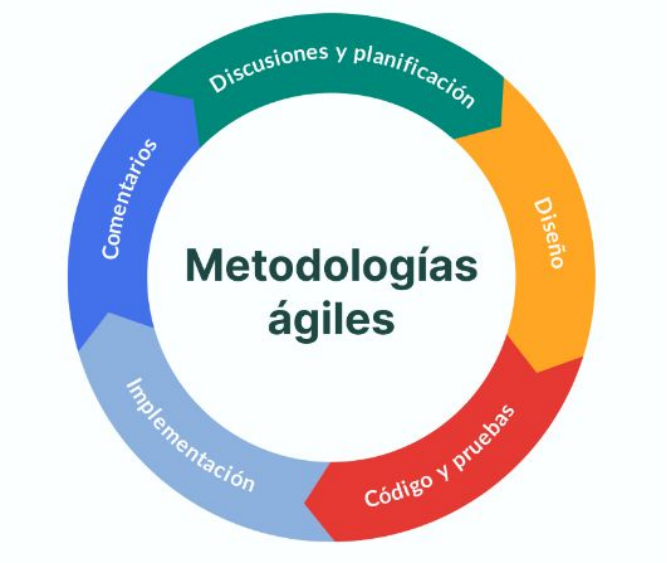
Modelo de prototipos-ventajas

- No modifica el flujo del ciclo de vida
- Reduce el riesgo de construir productos que no satisfagan las necesidades de los usuarios
- Reduce costo y aumenta la probabilidad de éxito
- Es útil cuando el cliente conoce los objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida.

- Debido a que el usuario ve que el prototipo funciona puede creer que es el producto terminado.
- El desarrollador puede caer en la tentación de ampliar el prototipo para construir el sistema final sin tener en cuenta los compromisos de calidad y mantenimiento que tiene con el cliente.

¿Descanso?

Metodologías de desarrollo de software ágiles



Las metodologías ágiles de desarrollo de software buscan proporcionar en poco tiempo pequeñas piezas de software en funcionamiento para aumentar la satisfacción del cliente. El desarrollo ágil de software implica que pequeños equipos autoorganizados de desarrolladores y representantes empresariales se reúnan regularmente en persona durante el ciclo de vida del desarrollo de software.

La metodología ágil favorece un enfoque sencillo de la documentación de software y acepta los cambios que puedan surgir en las diferentes etapas del ciclo de vida, en lugar de resistirse a ellos.

ORIGEN: Fechas claves

La primera constancia documental de un **documento ágil** fue en 1939, con **Walter Shewart**.

Taiichi Ohno puso en marcha en Toyota la **Metodología Lean**. Se analizan los desperdicios en el trabajo.

En 1958, la **NASA** emplea metodologías ágiles en el desarrollo del **Proyecto Mercury**.

En los años 60' y 70' se llevan las metodologías ágiles al **desarrollo informático**.

Surgimiento: Filosofía Ágil

Son formas de trabajo basadas en el **modelo iterativo** que permiten adaptar el **ciclo de vida del software** a las condiciones cambiantes del proyecto.

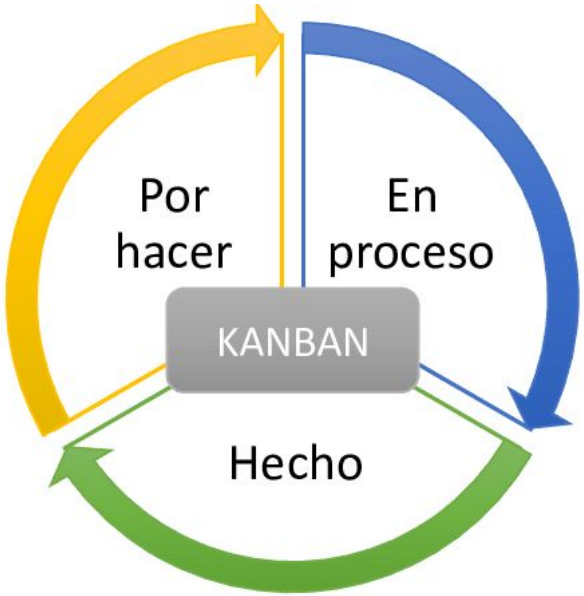
La filosofía “Ágil” nació en 2001, en Utah (EE. UU.), donde un grupo de programadores se reunieron para hablar sobre el futuro del desarrollo de software. Juntos llegaron a la conclusión de que los desarrolladores se centran mucho en la planificación y documentación de las etapas de desarrollo de software y se olvidaban de lo más importante, la satisfacción del cliente. De este modo nacieron los **12 principios** y **4 valores de la filosofía ‘agile’** recopilados en un documento que se llamó el **Manifiesto Ágil.** El documento proponía los conceptos básicos para la gestión ágil de proyectos de desarrollo software.

Manifiesto:

De acuerdo con lo que establecieron, los equipos de desarrollo ágil de software debían valorar:

- **Las personas y las interacciones** antes que los procesos y las herramientas
- **El software en funcionamiento** antes que la documentación exhaustiva
- **La colaboración con el cliente** antes que la negociación contractual
- **La respuesta ante el cambio** antes que el apego a un plan

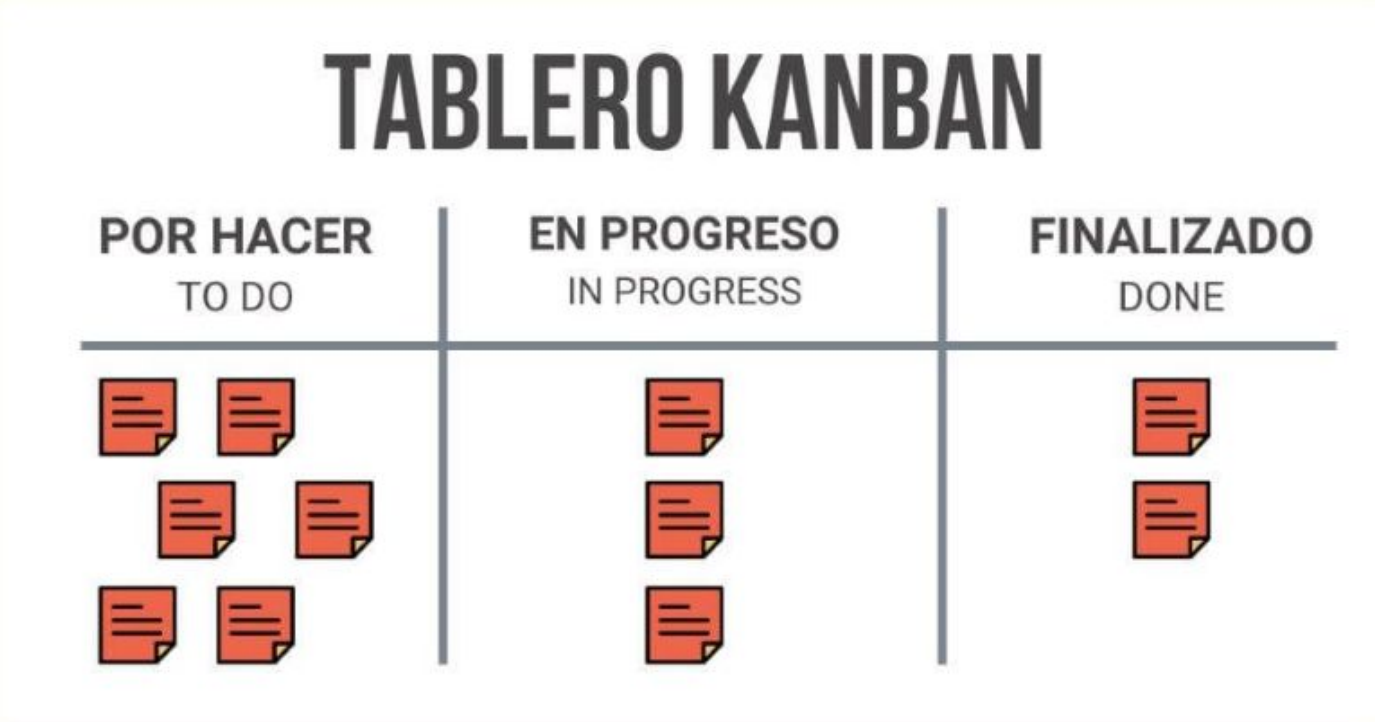
Kanban



Metodología de trabajo inventada por la empresa de automóviles Toyota. Consiste en dividir las tareas en porciones mínimas y organizarlas en un tablero de trabajo dividido en tareas pendientes, en curso y finalizadas. De esta forma, se crea un flujo de trabajo muy visual basado en tareas prioritarias e incrementando el valor del producto.La metodología Kanban se implementa por medio de tableros Kanban. Se trata de un método visual de gestión de proyectos que permite a los equipos visualizar sus flujos de trabajo y la carga de trabajo. En un tablero Kanban, el trabajo se muestra en un proyecto en forma de tablero organizado por columnas

Kanban

Kanban es un término de procedencia japonesa. Etimológicamente, se compone de “kan”, que significa visual; mientras que “ban” significa “tarjeta”.



Principios y links útiles

Los principios de Kanban son:

- Calidad garantizada
- Reducción del desperdicio
- Mejora continua
- Flexibilidad

Una de las herramientas más conocidas es Trello.
Lo puedes encontrar en https://trello.com/es
Hay otras como Monday
https://monday.com/lang/es/
O
ClickUp
https://clickup.com/

Lean

La metodología Lean: el ‘menos es más’ aplicado a la empresa

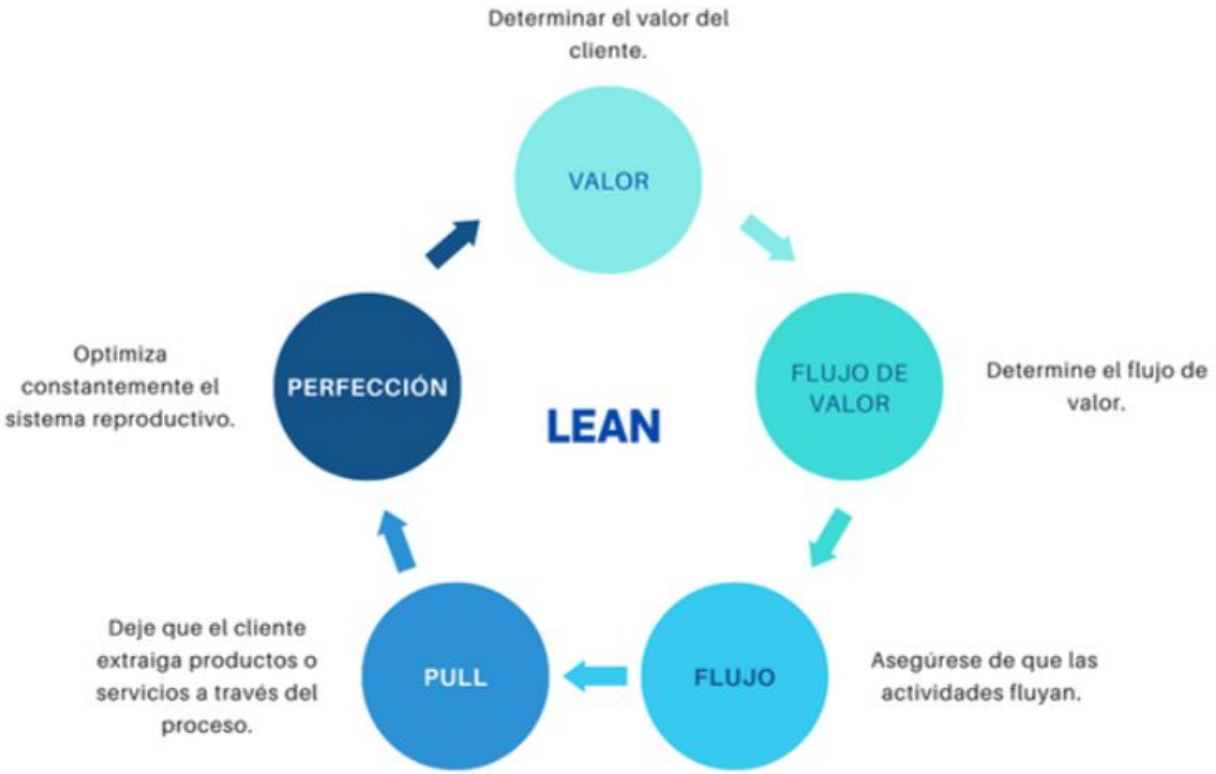
La metodología Lean es una forma innovadora de gestionar los procesos de una empresa eliminando actividades que no aportan valor, para así poder obtener un producto o servicio de mayor calidad y que mejore la experiencia de los clientes.

Resumiendo, se trata de optimizar los procesos empresariales (productivos y de gestión), con el fin de utilizar menos recursos en los mismos.

- Identificar sobrecargas en los procesos de producción.
- Eliminar desperdicios (pueden ser de materiales o de tiempo, por ejemplo).

Lean

Valor: Determinar el valor del cliente
Flujo de valor: Determine el flujo de valor
Flujo: Asegúrese de que las actividades fluyan
Pull: Deje que el cliente extraiga productos o servicios a través del proceso.
Perfección: Optimiza constantemente el sistema reproductivo.

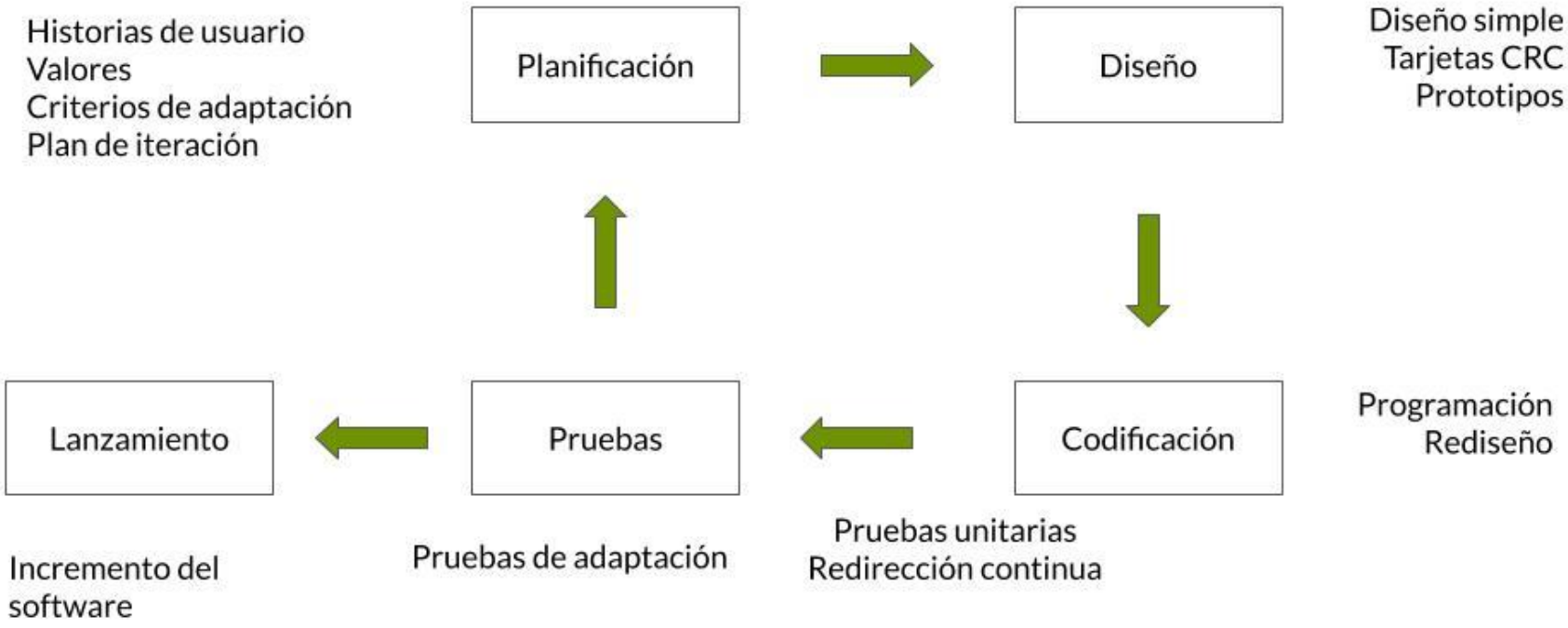


Programación extrema (XP)

La programación extrema es una metodología con bases en la comunicación constante y la retroalimentación. Se basa en valores, principios y prácticas, y su objetivo es permitir que equipos pequeños y medianos produzcan software de alta calidad y se adapten a los requisitos cambiantes y en evolución.

La XP busca desarrollar y gestionar proyectos con eficacia, flexibilidad y control.

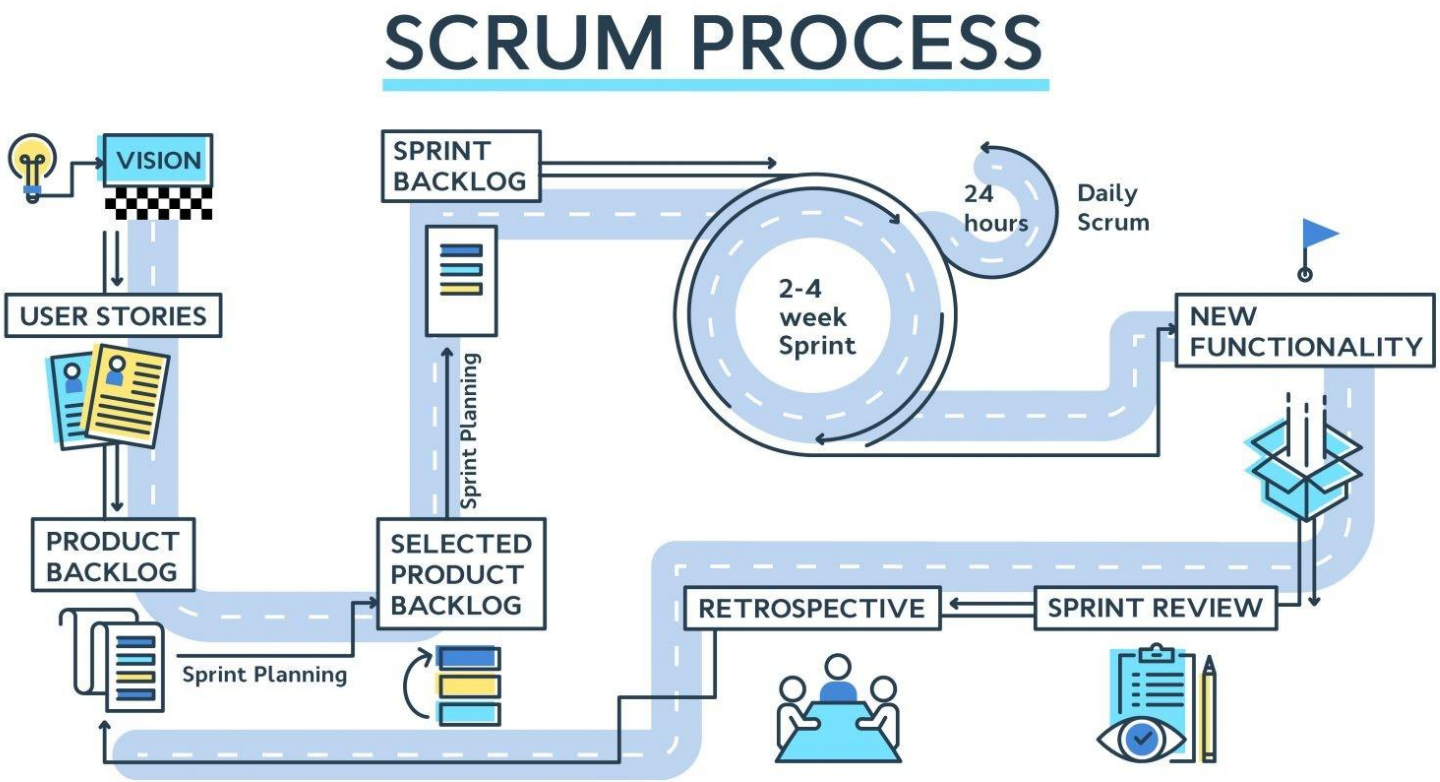
Programación extrema (XP)



Scrum

Es también una metodología incremental que divide los requisitos y tareas de forma similar a Kanban. Se itera sobre bloques de tiempos cortos y fijos (entre dos y cuatro semanas) para conseguir un resultado completo en cada iteración. Las etapas son: planificación de la iteración (planning sprint), ejecución (sprint), reunión diaria (daily meeting) y demostración de resultados (sprint review). Cada iteración por estas etapas se denomina también sprint. Vamos a ahondar más en está metodología en particular.

Proceso SCRUM



Equipo Scrum

Un Equipo Scrum consiste en un Propietario del Producto (Product Owner), un Scrum Master y los Developers.

El tamaño óptimo del Equipo Scrum es de máximo 10 miembros, esto le permite ser lo suficientemente pequeño como para permanecer ágil y lo suficientemente grande como para poder completar una cantidad significativa de trabajo.

Product Owner

El product backlog (o pila de producto) es un listado de todas las tareas que se pretenden hacer durante el desarrollo de un proyecto.

El Product Owner es el responsable de maximizar el valor entregado por los Developers.

La gestión del Product Backlog incluye:

- Expresar claramente los elementos del Product Backlog.
- Ordenar los elementos en el Product Backlog para alcanzar los objetivos y las misiones de la mejor manera posible.
- Garantizar que el Product Backlog sea visible, transparente y claro para todos y que muestre lo que el equipo trabajará a continuación.
- Asegurar que los Developers entienden los elementos del Product Backlog a nivel necesario.
- Participar en las reuniones de apertura y cierre del proyecto.
- Participar en las reuniones de planificación y revisión de las iteraciones (sprints).
- Entender las necesidades de las partes interesadas para posteriormente levantar los requerimientos que harán parte del Product Backlog.
- Mantener comunicación frecuente con el cliente (ya que es el único punto de contacto entre el cliente y el Equipo Scrum).
- Etc…

Developers

Los Developers son los profesionales que realizan el trabajo de entregar un incremento de producto “Terminado” (Done) que potencialmente se pueda poner en producción al final de cada Sprint.

Los Developers tienen las siguientes características:

- Son autogestionados. Nadie (ni siquiera el Scrum Master) les da órdenes o imposiciones sobre cómo convertir elementos del Product Backlog en Incrementos de funcionalidad potencialmente desplegables.
- Nadie fuera del Scrum Master y el Product Owner puede pedir a los Developers que trabajen en un conjunto diferente de requisitos que previamente hayan sido planeados y acordados.
- Los Developers son multifuncionales, con todas las habilidades necesarias para crear un incremento de producto.
- Scrum no reconoce títulos para los miembros de un Equipo Scrum, independientemente del trabajo que realice cada persona, quienes ejecuten el trabajo necesario para construir el producto se conocen como Developers.
- Los Developers pueden tener habilidades especializadas y áreas en las que estén más enfocados, pero la responsabilidad del producto recae sobre los Developers como un todo.

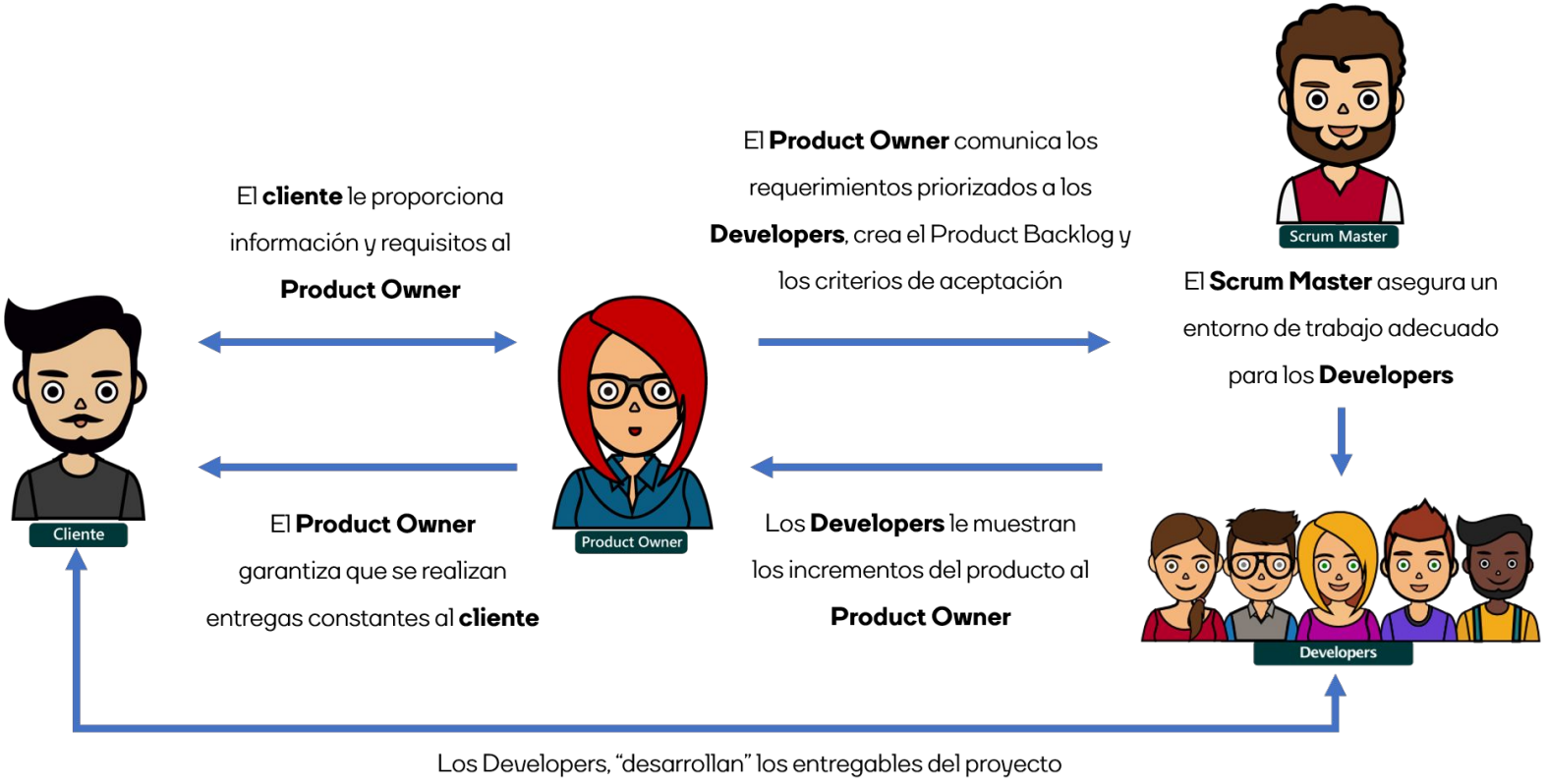
Scrum Master

El Scrum Master es el responsable en promover y apoyar Scrum como se define en la Guía de Scrum. Su principal responsabilidad entonces es garantizar que todos conocen y aplican correctamente la teoría y práctica de Scrum.

El Scrum Master tiene las siguientes características:

- El Scrum Master es un líder que sirve al Equipo Scrum y en general a toda la organización.
- El Scrum Master ayuda a las personas externas al Equipo Scrum a entender qué interacciones con el Equipo Scrum pueden ser útiles y cuáles no.
- Guía a los Desarrolladores para que aprendan a ser autogestionados y multifuncionales.
- Ayuda a los Desarrolladores a centrarse en crear productos de alto valor además de cumplir con la Definición de Terminado.
- Elimina impedimentos para el progreso de los Desarrolladores.
- Facilita los eventos de Scrum según se requiera o necesite, para que sean productivos, respetando el tiempo designado para cada uno (time-box).
- Ayudar a encontrar técnicas para gestionar el Product Backlog, definir el Objetivo de Producto (Product Goal) y gestionar los atrasos en el producto.

Comunicación del equipo



Ceremonias

El scrum master debe ser el responsable de convocar y liderar todas estas ceremonias (en caso contrario debe acordar con el resto de los participantes).

1. Refinamiento: El objetivo de un refinamiento es que el equipo de desarrollo tenga claro los requisitos que el product owner necesita para su producto.
2. Sprint Planning: El objetivo del sprint planning es acordar el backlog (conjunto de historias de usuario) de trabajo en el que el equipo trabajará durante un sprint.
3. Daily 's: El objetivo de esta reunión es que el scrum master tenga conocimiento del estado de cada una de las tareas, si existe algún impedimento para el equipo de desarrollo y si existe riesgo para cumplir las tareas del sprint.
4. Demo: El objetivo de esta reunión es la validación de las historias de usuario realizadas durante el sprint.
5. Retrospectiva: El objetivo es que el desempeño global del equipo mejore.

Metodologías Ágiles-ventajas

- Promueven el trabajo en equipo.
- Documentación mínima.
- Poca planificación.
- Las funcionalidades pueden desarrollarse rápidamente.

- Mayor riesgo(mantenibilidad y extensibilidad)
- Dependen fuertemente en la interacción con el cliente.
- Alta dependencia de los individuos ya que se genera poca documentación.

Fin