

Tema 2: El proceso de desarrollo del software. Paradigmas o modelos de desarrollo del Software

BLOQUE I: INTRODUCCIÓN Y PARADIGMAS
DE DESARROLLO EN INGENIERÍA DEL SOFTWARE

Ingeniería del Software
Grado en Ingeniería Informática
Curso 2024/2025

David Cáceres Gómez



Índice

1. El Proceso: Modelos de Desarrollo	2
1.1. Procesos primarios del ciclo de vida del software	2
1.2. Procesos de soporte del ciclo de vida del software	3
1.3. Procesos organizacionales	3
1.4. Proceso de desarrollo de software	4
2. Paradigmas o Modelos de Desarrollo del Software	6
3. Modelos Tradicionales de Desarrollo de Software	8
3.1. Lineal o secuencial	8
3.2. Cascada	9
3.3. Espiral	10
3.4. Incremental	11
3.5. Evolutivo	12
3.6. Prototipado	13
3.7. Concurrencia	14
4. Proceso Unificado de Desarrollo	15
5. Modelos Ágiles de Desarrollo de Software	17
5.1. Scrum	19
5.2. Otros modelos ágiles	22
5.2.1. El modelo XP	22
5.2.2. Kanban	23
Referencias	24

1. El Proceso: Modelos de Desarrollo

En el contexto del software, un proceso es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema funcional.

Características:

- Tiene una serie de actividades principales.
- Utiliza recursos, está sujeto a restricciones y genera productos intermedios y finales.
- Compuesto por subprocessos que se encadenan de alguna forma.
- Cada actividad tiene sus criterios de entrada y salida, que permiten conocer cuando comienza y termina dicha actividad.
- Existen principios orientadores que explican las metas de cada actividad.

Según el estándar ISO/IEC 12207 de 1995, se definieron y normalizaron dos conceptos básicos:

- **Ciclo de vida del software:** El periodo de tiempo comprendido desde la definición de los requisitos hasta el fin del su uso. Proporciona consistencia y estructura a las actividades, facilitando la correcta realización de las mismas de forma repetitiva.
- **Procesos:** Conjunto de actividades y tareas interrelacionadas que, al ejecutarse de manera conjunta, transforman una entrada en una salida.



Figura 1: Proceso de desarrollo de Software [1]

1.1. Procesos primarios del ciclo de vida del software

- **Adquisición:** Proceso global que sigue el adquiriente para obtener el producto.
- **Suministro:** Proceso global que sigue el suministrador para proporcionar el producto.

- **Desarrollo:** Proceso empleado por el suministrador para el diseño, construcción y pruebas del producto.
- **Operación:** Proceso que sigue el operador en su uso diario del producto.
- **Mantenimiento:** Proceso destinado a mantener el producto, incluyendo tanto los cambios en el propio producto como en su entorno de operación.

1.2. Procesos de soporte del ciclo de vida del software

El estándar 12207 identifica los procesos de soporte que pueden utilizarse tanto desde un proceso primario como desde otro proceso de soporte. Los procesos de soporte son los siguientes:

- **Documentación:** Actividades empleadas para registrar información específica empleada por otros procesos.
- **Gestión de la configuración:** Actividades empleadas para mantener un registro de los productos generados en la ejecución de los procesos.
- **Aseguramiento de la calidad:** Actividades empleadas para garantizar de forma objetiva que el producto y los procesos asociados son conformes a los requisitos documentados y a las planificaciones.
- **Verificación:** Actividades empleadas para comprobar el producto.
- **Validación:** Actividades empleadas para confirmar que el producto cumple su propósito.
- **Reuniones de revisión:** Reuniones empleadas por las dos partes para evaluar el estado del producto y de las actividades.
- **Auditorias:** Actividades para determinar que el proyecto cumple con los requisitos, planes y contratos.
- **Resolución de problemas:** Actividades para analizar y resolver problemas relativas al proyecto, sea cual sea su fuente y naturaleza.

1.3. Procesos organizacionales

El estándar 12207 identifica los procesos que deben llevarse a cabo dentro de la organización encargada de ejecutar el proyecto. Generalmente, estos procesos se

aplican de manera común a múltiples proyectos. De hecho, las organizaciones más avanzadas los reconocen e institucionalizan.

- **Gestión:** Describe las actividades de gestión de la organización, incluyendo también la gestión de proyectos.
- **Infraestructura:** Actividades necesarias para que puedan realizarse otros procesos del ciclo de vida. Incluye entre otros el capital y el personal.
- **Mejora:** Actividades realizadas para mejorar la capacidad del resto de procesos.
- **Formación:** Actividades para capacitar y desarrollar al personal.

1.4. Proceso de desarrollo de software

El proceso de desarrollo de software se divide en actividades, las cuales se descomponen en acciones y tareas. Se pueden identificar cinco actividades principales: comunicación, planificación, modelado, construcción y despliegue. Además, existen actividades “paraguas” que incluyen la gestión y control del proyecto, la gestión del riesgo, el aseguramiento de la calidad y la gestión de la configuración, entre otras.

En las tareas intervienen:

- Productos o artefactos que se utilizan (entrada) o se construyen (salida).
- Aspectos de control de la calidad.
- Hitos de proyecto.

En la Figura 2 se representa el proceso del software de manera esquemática. En dicha figura, cada actividad estructural está formada por un conjunto de acciones de ingeniería de software y cada una de éstas se encuentra definida por un conjunto de tareas que identifica las tareas del trabajo que deben realizarse, los productos del trabajo que se producirán, los puntos de aseguramiento de la calidad que se requieren y los puntos de referencia que se utilizarán para evaluar el avance.

Es importante destacar un aspecto clave del proceso de software. En la Figura 3 se muestra este aspecto, conocido como **flujo del proceso**, que describe cómo están organizadas las actividades estructurales y las acciones y tareas que ocurren dentro de cada una en relación con la secuencia y el tiempo. Los tipos de flujos son los siguientes:

- **Flujo de proceso lineal:** ejecuta cada una de las cinco actividades estructurales en secuencia, comenzando por la comunicación y terminando con el despliegue (Figura 3a).
- **Flujo de proceso iterativo:** repite una o más de las actividades antes de pasar a la siguiente (Figura 3b).
- **Flujo de proceso evolutivo:** realiza las actividades en forma “circular”. A través de las cinco actividades, cada circuito lleva a una versión más completa del software (Figura 3c).
- **Flujo de proceso paralelo:** ejecuta una o más actividades en paralelo con otras (por ejemplo, el modelado de un aspecto del software tal vez se ejecute en paralelo con la construcción de otro aspecto del software) (Figura 3d).

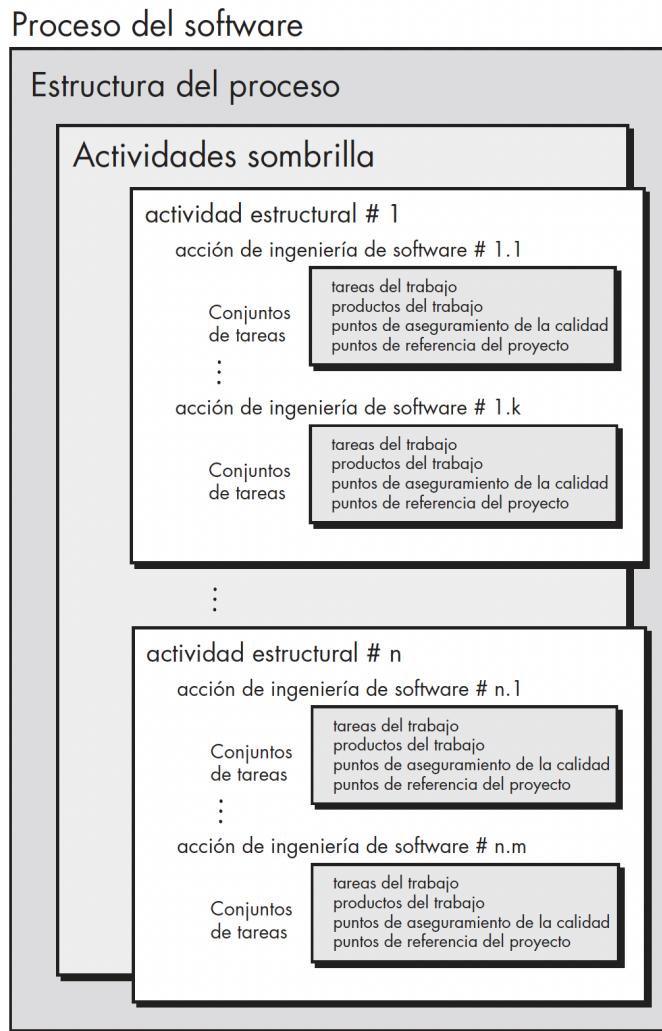


Figura 2: Estructura de un proceso de software [2]

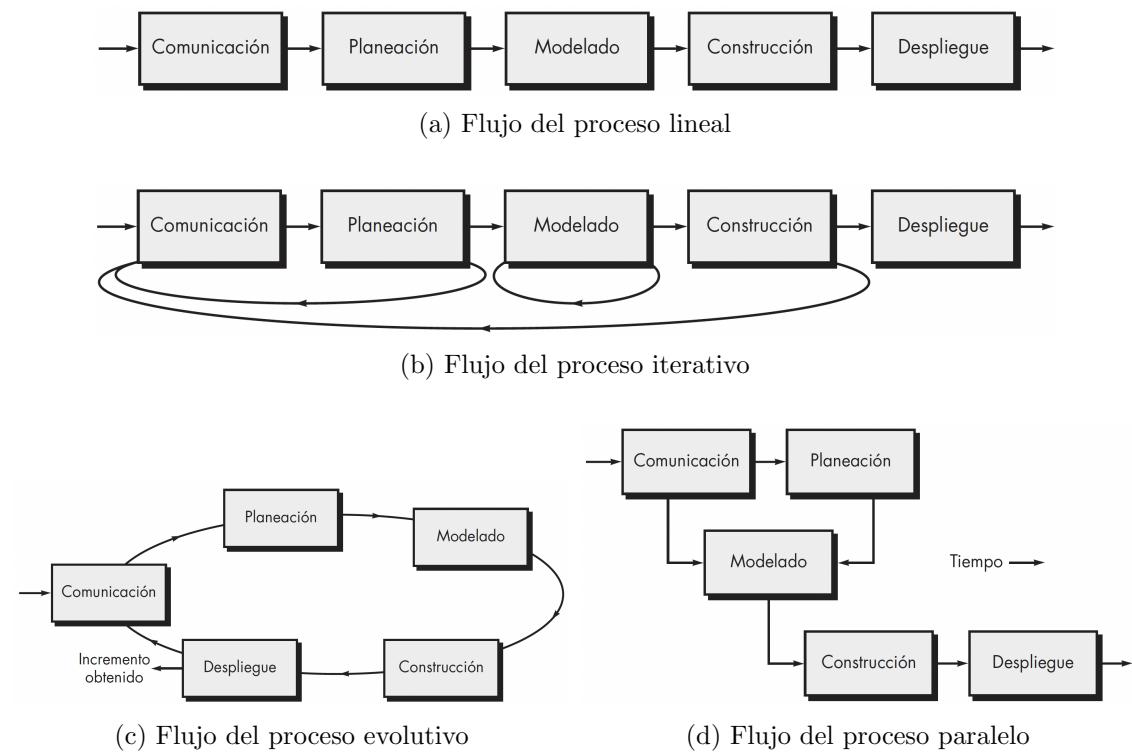


Figura 3: Flujo del proceso [2]

2. Paradigmas o Modelos de Desarrollo del Software

Las metodologías de desarrollo de software son un conjunto de técnicas y métodos organizativos diseñados para optimizar la creación de soluciones informáticas. Su principal objetivo es estructurar los equipos de trabajo para que desarrollen las funciones de un programa de manera eficiente.

Al desarrollar productos para un cliente o mercado específico, es crucial considerar factores como costes, planificación, complejidad, recursos disponibles y lenguajes de programación. Estas consideraciones se integran en una metodología que organiza el trabajo de forma ordenada.

El desarrollo de software puede ser especialmente complejo, especialmente en proyectos grandes. Abordar un desarrollo sin una metodología clara puede llevar a problemas, retrasos y errores, resultando en un producto final deficiente. Utilizar una metodología adecuada ayuda a reducir la complejidad, organizar las tareas, agilizar el proceso y mejorar la calidad del resultado final.



Figura 4: Metodologías de desarrollo de software [3]

En la actualidad se pueden diferenciar dos grandes grupos de metodologías de desarrollo de software: las ágiles y las tradicionales. A continuación, se explican las características de cada una de ellas:

- **Metodologías de desarrollo de software tradicionales:** Se caracterizan por definir de manera rígida los requisitos al inicio de los proyectos, lo que las hace inflexibles ante cambios. Su enfoque lineal implica que las etapas deben completarse secuencialmente sin posibilidad de retroceso, lo que dificulta la adaptación a un entorno en constante cambio.
- **Metodologías de desarrollo de software ágiles:** Se basan en un enfoque incremental, donde cada ciclo de desarrollo agrega nuevas funcionalidades de manera rápida y en pequeñas porciones. Además, fomentan la formación de equipos autosuficientes que se reúnen regularmente para compartir novedades, permitiendo que el cliente aporte requerimientos y correcciones mientras el proyecto avanza, lo que asegura un desarrollo más colaborativo y en tiempo real.

3. Modelos Tradicionales de Desarrollo de Software

3.1. Lineal o secuencial

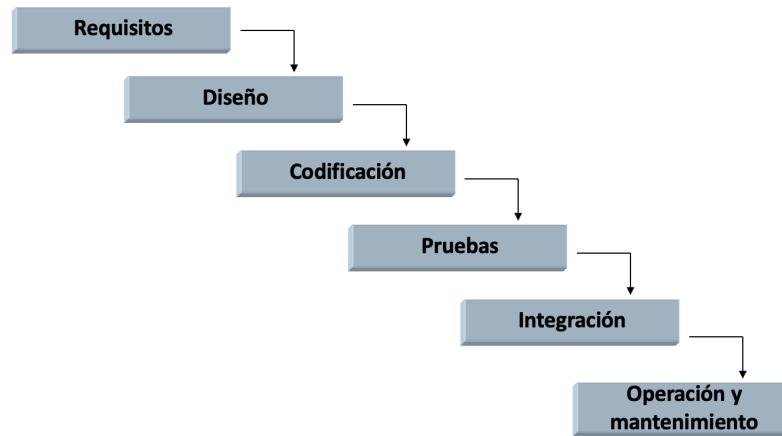


Figura 5: Modelo lineal o secuencial

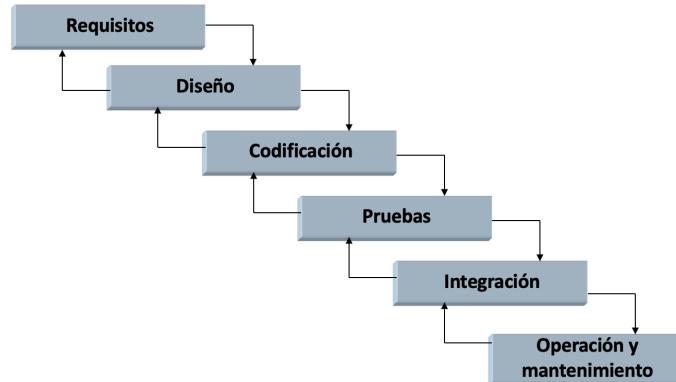
Este modelo de desarrollo se caracteriza por la sucesión escalonada de etapas: requisitos, diseño, codificación, pruebas e integración. Es fundamental completar cada etapa antes de pasar a la siguiente, lo que lo hace un enfoque muy rígido. Desde su identificación a principios de la década de 1950, este modelo ha demostrado ser problemático en situaciones reales, ya que cada fase requiere como entrada el resultado completo de la anterior.

Esta rigidez puede generar dificultades, ya que frecuentemente es complicado contar con requisitos completos o un diseño detallado del sistema en las fases iniciales, lo que se convierte en una barrera para el avance del proyecto. Sin embargo, este modelo puede ser adecuado en ciertas circunstancias, como:

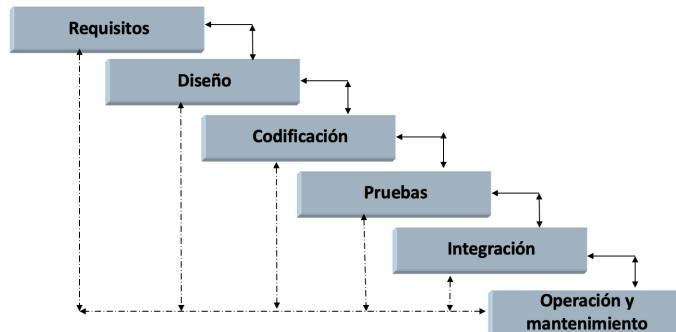
- **Desarrollo de nuevas versiones de sistemas ya establecidos**, donde el desconocimiento de las necesidades del usuario y del entorno operativo no plantea riesgos significativos.
- **Sistemas pequeños**, sin previsión de evolución a corto plazo.

Un modelo que evita esta rigidez es el modelo en cascada, que se presentará a continuación y permite una mayor flexibilidad en el desarrollo.

3.2. Cascada



(a) Modelo en cascada con retroalimentación a fase anterior



(b) Modelo en cascada con retroalimentación a cualquier fase

Figura 6: Modelo en cascada

En 1970, Winston Royce introdujo el concepto de flujos de retorno en el modelo secuencial, dando lugar al modelo en cascada. Este modelo refleja la necesidad de regresar con frecuencia desde una fase hacia las anteriores, utilizando la información generada a medida que avanza el desarrollo.

Las representaciones más habituales del modelo en cascada muestran que el retorno puede ocurrir únicamente entre una fase y la anterior, aunque es más preciso considerar que en cualquier fase puede surgir la necesidad de modificar cualquiera de las etapas previas. Al igual que el modelo secuencial, el modelo en cascada subraya la importancia de tener requisitos y un diseño bien definidos antes de comenzar la codificación. Sin embargo, en la práctica, la dificultad de obtener documentación completa sobre requisitos y diseño antes de iniciar la codificación puede convertirse en una barrera que impida el avance a la siguiente fase.

Estas limitaciones han llevado a que el modelo no sea muy popular, y los equipos

que lo aplican a menudo se ven tentados a comenzar el diseño o incluso la codificación sin tener un conocimiento adecuado de los requisitos.

El modelo en cascada resulta apropiado en ciertos contextos, tales como:

- **Desarrollo de nuevas versiones de sistemas ya establecidos**, donde el desconocimiento de las necesidades del usuario y del entorno operativo no plantea riesgos significativos.
- **Sistemas pequeños**, sin previsión de evolución a corto plazo.

Es importante señalar que algunos textos utilizan el término “cascada” para referirse al modelo lineal, y “cascada modificada” para describir el modelo en cascada. Esta diferenciación resalta las variaciones en la implementación de los flujos de trabajo dentro del desarrollo de software.

3.3. Espiral

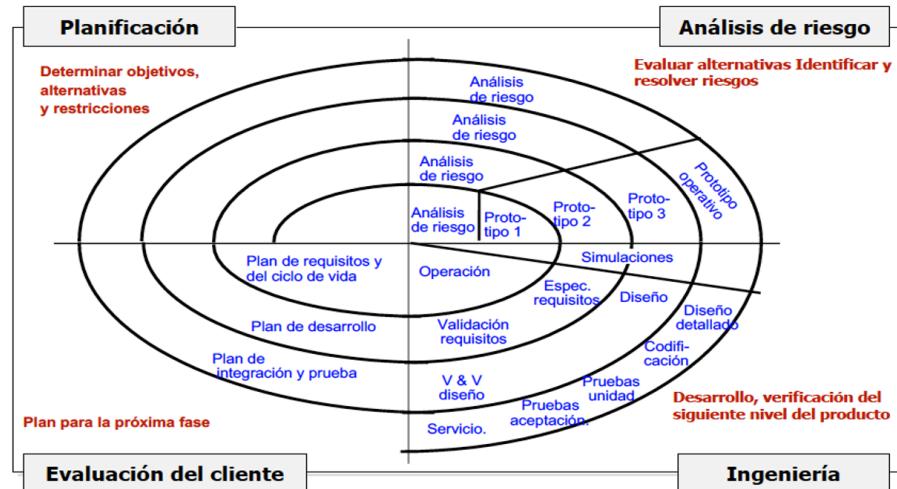


Figura 7: Modelo espiral de Boehm [4]

Definido por Boehm en 1988, este modelo presenta un enfoque evolutivo al desarrollo de software, en contraste con la linealidad de los modelos anteriores. Una de sus características distintivas es la introducción del “análisis de riesgo”, que guía la evolución del proceso de desarrollo.

El ciclo de **iteración** de este modelo evolutivo se convierte en una espiral, que al representarse sobre ejes cartesianos muestra en cada cuadrante una clase particular

de actividad: Planificación, Análisis de riesgo, Ingeniería y Evaluación, que se suceden de forma consecutiva a lo largo del ciclo de vida del desarrollo. La dimensión angular representa el avance relativo en el desarrollo de las actividades de cada cuadrante. En cada ciclo de la espiral se realiza una parte del desarrollo total, a través de los cuatro tipos de actividades.

- En la **planificación** de cada vuelta se establece el contexto del desarrollo y se decide qué parte del mismo se abordará en el ciclo siguiente.
- Las **actividades de análisis de riesgo** evalúan las alternativas posibles para la ejecución de la siguiente parte del desarrollo, seleccionando la más ventajosa y previendo los riesgos posibles.
- Las **actividades de ingeniería** corresponden a las indicadas en los modelos lineales (secuencial y cascada): análisis, diseño, codificación, etc.
- Las actividades de evaluación analizan los resultados de la fase de ingeniería, tomando el resultado de la **evaluación como punto de partida para el análisis de la siguiente fase**.

Este modelo permite múltiples combinaciones ya que en la planificación de cada ciclo se determina el avance que se va a ejecutar durante la vuelta. Éste puede consistir en la obtención y validación de requisitos, o en el desarrollo del diseño, o el diseño junto con la codificación, o en la obtención de un subsistema completo (cascada de requisitos – diseño – codificación – pruebas – integración).

3.4. Incremental

El modelo incremental mitiga la rigidez del modelo en cascada, descomponiendo el desarrollo de un sistema en partes; para cada una de las cuales se aplica un ciclo de desarrollo (Figura 8)

Las ventajas que ofrece son:

- El usuario dispone de pequeños subsistemas operativos que ayudan a perfilar mejor las necesidades reales del sistema en su conjunto.
- El modelo produce entregas parciales en periodos cortos de tiempo, comparados con el tiempo necesario para la construcción del sistema en su conjunto, y permite la incorporación de nuevos requisitos que pueden no estar disponibles o no ser conocidos al iniciar el desarrollo.



Figura 8: Modelo incremental: subsistemas

Los desarrollos de cada subsistema o versión pueden solaparse en el tiempo o llevarse a cabo de manera secuencial. A la funcionalidad seleccionada para el primer incremento se le conoce como el producto “núcleo” (*core product*). Este enfoque resulta apropiado para:

- Desarrollo de sistemas en los que el cliente necesita disponer de parte de la funcionalidad antes de lo que costaría desarrollar el sistema completo.
- Desarrollo de sistemas en los que por razones del contexto interesa realizar la obtención de los requisitos de forma escalonada a través de subsistemas.

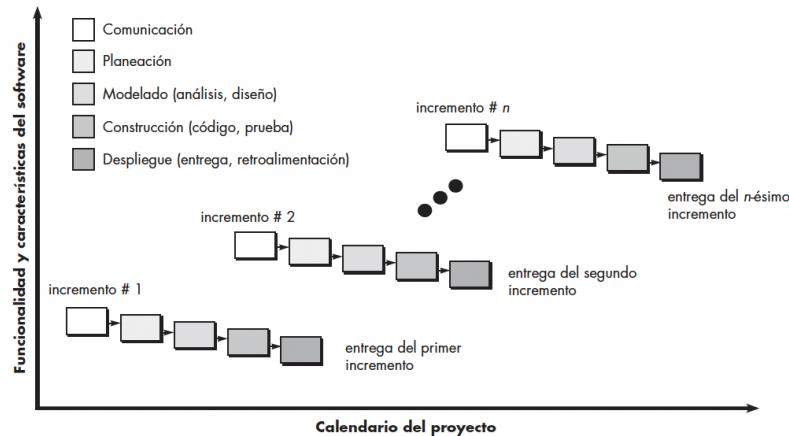


Figura 9: Modelo incremental [2]

3.5. Evolutivo

Este modelo se compone de varios ciclos de desarrollo, donde cada ciclo produce un sistema completo que se utiliza en el entorno operativo. La información acumulada durante el desarrollo y la fase de operación de cada sistema se utiliza para mejorar y ampliar los requisitos y el diseño del siguiente. En realidad, representa un ciclo de vida aplicable a todos los sistemas desarrollados, los cuales se optimizan a través de versiones sucesivas.

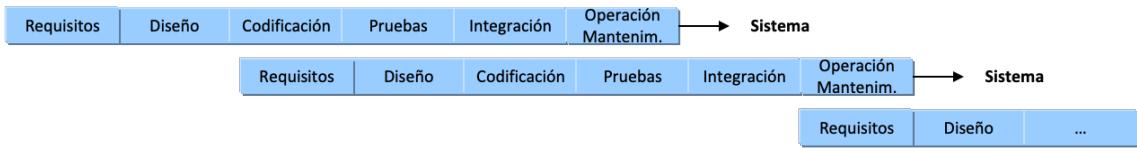


Figura 10: Modelo evolutivo

Las circunstancias en las que este modelo puede resultar apropiado son:

- Desconocimiento inicial de todas las necesidades operativas que serán precisas, generalmente por tratarse del desarrollo de un sistema que operará en un entorno nuevo sin experiencia previa.
- Necesidad de que el sistema entre en operación en tiempos inferiores a los que serían necesarios para diseñarlo y elaborarlo de forma exhaustiva.
- Necesidad de desarrollar sistemas en entornos cambiantes (sujetos a normas legislativas, mejora continua del producto para hacer frente a desarrollos de la competencia, etc.).

Aunque en su concepción inicial contempla desarrollos internos en cascada, también podría plantearse, por ejemplo, un ciclo de vida evolutivo con desarrollos internos en espiral.

3.6. Prototipado

El prototipado se basa en la construcción de un prototipo de software que se construye rápidamente para que los usuarios puedan probarlo y aportar *feedback*. Así, se puede arreglar lo que está mal e incluir otros requerimientos que puedan surgir. Es un modelo iterativo que se basa en el método de prueba y error para comprender las especificidades del producto.

Los prototipos pueden ser:

- **Ligeros:** dibujos de pantallas de interfaz con simulación de funcionamiento por enlaces a otros dibujos...
- **Operativos:** Módulos de software con funcionamiento propio que se desarrollan sin cubrir las funcionalidades completas del sistema, normalmente en entornos RAD (Diseño rápido de aplicaciones).

Esta forma de trabajo previo suele tener como principal objetivo la experimentación con un entorno similar al pretendido, para obtener retro-información del usuario o cliente que ayuda a los desarrolladores en la concreción de los requisitos.

Aunque ofrece muchas ventajas, deben conocerse los riesgos que implica el uso de prototipado:

- Como puede parecer que se ha desarrollado un interfaz de usuario sofisticado y elaborado, el cliente puede llegar a pensar que ya se ha realizado el grueso del trabajo.
- Si se trata de un prototipo operativo, puede empezar a crecer al margen de la planificación, más allá de los objetivos previstos, desbordando agendas y recursos.
- Si se trata de un prototipo ligero desarrollado fuera del departamento de desarrollo (ej. Marketing), puede mostrar al cliente funcionalidades no implementables.
- El prototipo puede llegar a ofrecer funcionalidades superiores a lo consegurable, por estar construido en un entorno diferente al de desarrollo, o no incluir toda la funcionalidad del sistema.

3.7. Concurrencia

Consiste en el solapamiento de actividades, cada una en un estado concreto. La concurrencia puede aportar beneficios sobre la planificación de un proyecto de software, o por el contrario ser origen o consecuencia de problemas.

Los factores que deben tenerse en cuenta para analizar cómo ayuda o perjudica al rendimiento son:

- Índice de concurrencia. Se produce en un grado reducido, generando un escaso flujo de modificaciones; o por el contrario se da de forma intensiva generando situaciones problemáticas en la planificación o en la distribución del trabajo.
- Gestión de la concurrencia. La concurrencia puede producirse en un proyecto de forma planificada o inducida por las circunstancias. En ambos casos resulta muy importante la labor de gestión del proyecto para tratarla de forma adecuada con el mayor beneficio, o el menor perjuicio a los planes y la calidad del proyecto.

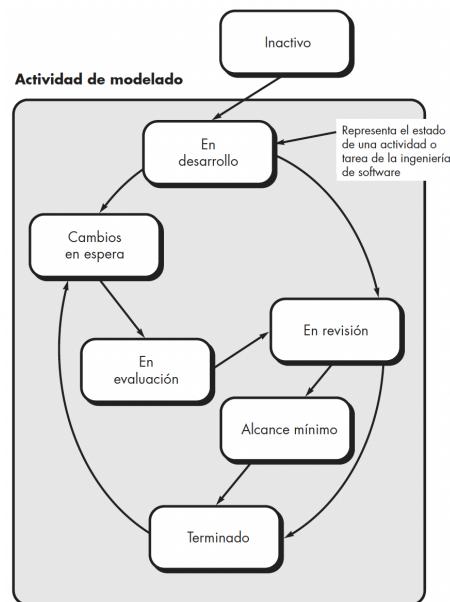


Figura 11: Elemento del modelo de proceso concurrente [2]

4. Proceso Unificado de Desarrollo

El Proceso Unificado de Desarrollo de Software o simplemente Proceso Unificado es un marco de desarrollo de software que se caracteriza por estar **dirigido por casos de uso**, centrado en la **arquitectura** y por ser **iterativo e incremental**. Propuesto por la empresa *Rational Software* (ahora parte de IBM), su objetivo era ser para los métodos de desarrollo de software lo que el UML fue para los lenguajes de modelización. Integra elementos de los enfoques tradicionales y ágiles, por lo que se sitúa en medio de ambos.

El Proceso Unificado (UP) consiste en un conjunto de procesos configurables que pueden adaptarse a diferentes contextos, lo que significa que su aplicación varía según el tipo de desarrollo, ya sea para una *intranet* empresarial o para software militar.

A diferencia del modelo en cascada, el UP facilita el desarrollo en ciclos cortos, lo que permite una adaptación continua. Si bien es más rígido que metodologías ágiles puras como Scrum o XP, ofrece retroalimentación y acepta cambios durante el desarrollo, haciéndolo adaptable a diversas situaciones del proyecto.

En términos generales todas las variantes del proceso unificado se basan en potenciar seis prácticas fundamentales:

1. Desarrollo iterativo e incremental.
2. Gestión de los requisitos (mediante casos de uso).
3. Arquitecturas basadas en componentes (módulos o subsistemas con una función clara).
4. Utilización de modelos visuales (mediante el lenguaje UML).
5. Verificación de la calidad del software (a lo largo de todas las etapas del proceso, no sólo al final).
6. Control de los cambios al software.

El proceso se puede descomponer de dos formas: en función del tiempo, organizándolo en fases, y en función de las actividades o el contenido, desglosándolo en tareas específicas. Además, define los roles de las distintas personas involucradas y las tareas que debe realizar cada rol.

Fases del proyecto

El UP se repite a lo largo de una serie de ciclos a lo largo del tiempo de manera secuencial, consta de cuatro fases:

1. **Inicio:** Se describe el producto final a partir de una idea inicial y se realiza un análisis de negocio. Se identifican las principales funciones del sistema y los usuarios más relevantes mediante un modelo de casos de uso, así como una posible arquitectura del sistema. También se elabora un plan del proyecto que incluye costos, identificación y priorización de riesgos.
2. **Elaboración:** Se detallan los principales casos de uso y se diseña la arquitectura del sistema, incluyendo las vistas arquitectónicas del modelo de casos de uso, análisis, diseño, implementación y despliegue. Al final de esta fase, se planifican las actividades y se estiman los recursos necesarios para completar el proyecto.
3. **Construcción:** En esta fase, se desarrolla el producto al integrar el software en la arquitectura definida. Al finalizar, se tienen todos los casos de uso acordados para el desarrollo, aunque es posible que se presenten defectos.
4. **Transición:** Este periodo se centra en la conversión del producto en una versión beta, donde los usuarios prueban el sistema e informan sobre defectos y deficiencias. Se corrigen problemas e incorporan sugerencias. Además, incluye actividades como la formación de los usuarios, la provisión de una línea de ayuda y asistencia.

Cada fase se divide a su vez en iteraciones.

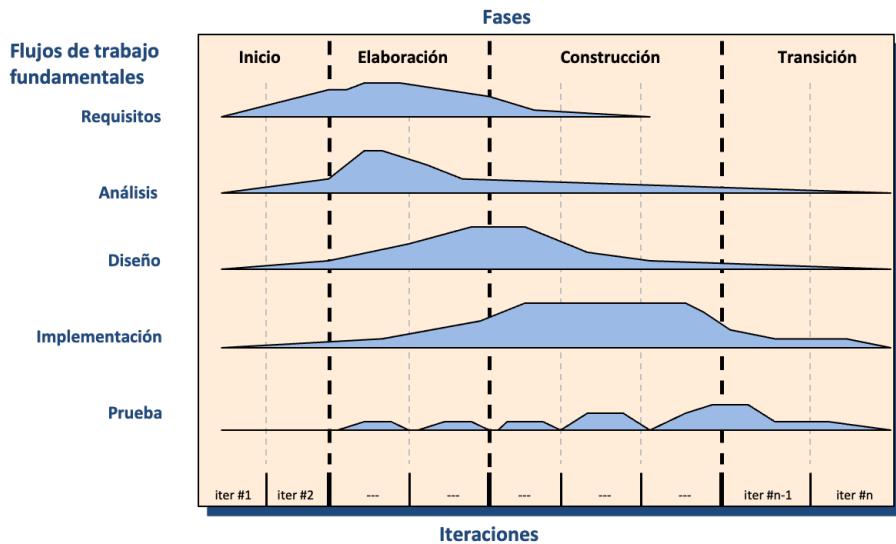


Figura 12: Fases del Proceso Unificado de Desarrollo [1]

5. Modelos Ágiles de Desarrollo de Software

Los modelos de procesos ágiles se basan en el “Manifiesto Ágil de Software” [5], que fue creado en 2001 cuando 16 destacados desarrolladores, escritores y consultores firmaron dicho manifiesto, centrado en cuatro valores:

Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

- *Individuos e interacciones sobre procesos y herramientas*
- *Software funcionando sobre documentación extensiva*
- *Colaboración con el cliente sobre negociación contractual*
- *Respuesta ante el cambio sobre seguir un plan*

Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.

El manifiesto Ágil está basado en 12 principios, que son:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana

y continua de software con valor.

2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

El proceso ágil es candidato a aplicarse en determinadas situaciones clave como son:

- Cuando resulta difícil predecir cuales requisitos del software persistirán y cuales cambiarán (cambios en las prioridades de los clientes).
- Es conveniente que el diseño y la construcción de software estén intercalados.
- El análisis, diseño y construcción de software no son predecibles respecto a la planificación.

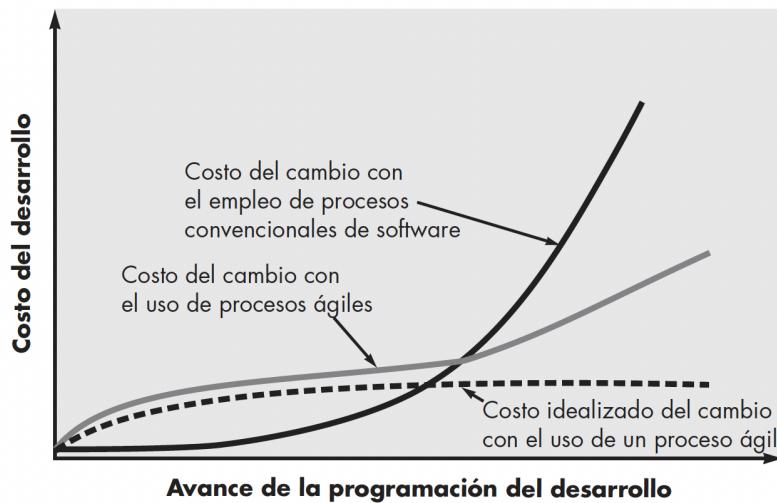


Figura 13: Cambio de los costos en función del tiempo en el desarrollo [2]

La Ingeniería del Software Ágil combina una filosofía y un conjunto de directrices de desarrollo, como son:

- Busca la satisfacción del cliente y la entrega temprana del software incremental
- Equipos de proyectos pequeños y con alta motivación
- Métodos informales
- Mismos productos de trabajo
- Simplicidad general del desarrollo

Actualmente, la Ingeniería del Software Ágil es especialmente relevante debido a que los sistemas basados en computadoras y los productos de software están en constante aceleración y cambio. Además, ofrece una alternativa eficaz a la ingeniería tradicional para determinadas clases de software y tipos específicos de proyectos.

5.1. Scrum

Scrum, cuyo nombre proviene de una jugada del rugby, es un método ágil de desarrollo de software creado por Jeff Sutherland y su equipo a principios de la década de 1990. En años recientes, Schwaber y Beedle han ampliado y desarrollado aún más los métodos de Scrum. Los principios de Scrum están alineados con el Manifiesto Ágil y guían las actividades de desarrollo dentro de un proceso de análisis que incluye las siguientes fases: requerimientos, análisis, diseño, evolución y entrega.

Dentro de cada fase, las tareas se organizan en un patrón de trabajo llamado *sprint*. El trabajo realizado en cada *sprint* (el número de *sprints* necesarios para cada fase estructural varía según la complejidad y tamaño del proyecto) se ajusta a las necesidades del problema en cuestión y es definido y frecuentemente modificado en tiempo real por el equipo Scrum. El flujo general del proceso Scrum se muestra en la Figura 14.

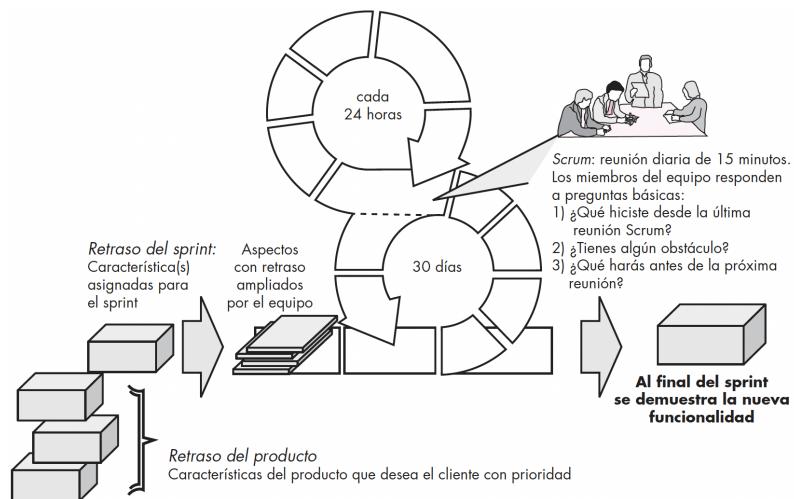


Figura 14: Flujo del proceso Scrum [2]

Roles

Scrum distingue, de entrada, entre dos grandes grupos de participantes de un proyecto: los que están comprometidos en el desarrollo (el equipo) y los que están involucrados pero no forman parte del equipo (lo que otros métodos denominan *stakeholders*). Entre los miembros del equipo, Scrum define tres roles muy particulares:

- **Scrum Master:** Es responsable de asegurar que el equipo sigue los valores, las prácticas y las reglas de Scrum. También se encarga de eliminar los impedimentos que el equipo se pueda encontrar dentro de la organización.
- **Product owner:** Es la persona responsable de velar por los intereses de todos los *stakeholders* y llevar el proyecto a buen término. Es, por lo tanto, el encargado de decidir qué se implementa y qué es más prioritario.
- **Team:** El resto de los miembros del equipo son los desarrolladores. No se especializan, sino que todos deben tener, en mayor o menor medida, habilidades en todas las actividades implicadas. Los miembros del equipo deciden ellos mismos cómo se organizan el trabajo y nadie (ni siquiera el Scrum Master) les puede decir cómo lo deben hacer.

Artefactos

Scrum sólo define cuatro artefactos:

- **Product backlog:** Es la lista de requisitos pendientes de implementar en el producto. Cada entrada (normalmente una “historia de usuario”) tiene asociada una estimación del valor que aporta a la organización y también del coste de su desarrollo.
- **Sprint backlog:** El backlog para una iteración concreta (Scrum denomina sprint a las iteraciones); más detallada que el product backlog y en la que cada historia de usuario se descompone en tareas de entre cuatro y dieciséis horas de duración. Los sprints tienen una duración máxima de 6 semanas, aunque se recomienda realizarlos en 2-3 semanas.
- **Release burndown chart:** Gráfico que muestra el progreso actual del equipo en función del número de historias de usuario que faltan por implementar.
- **Sprint burndown:** El burndown para una iteración concreta, en la que el progreso se puede medir en tareas finalizadas aunque no sean historias de usuario completas.

Reuniones

En cuanto a las reuniones, éstas se basan en dos ciclos de iteraciones: una iteración más larga (como las que podemos encontrar en métodos como UP) y una iteración diaria.

- **Sprint planning meeting:** Reunión que se organiza antes de empezar un sprint y en la que se deciden qué historias de usuario se implementarán en este sprint. Por lo tanto, se crea el sprint backlog.
- **Daily scrum:** Reunión diaria en la que todos los miembros del equipo responden a tres preguntas: qué hicieron ayer, qué piensan hacer hoy y qué impedimentos se han encontrado que les han impedido avanzar. La finalidad de esta reunión es que todos los miembros del equipo estén enterados de lo que está haciendo el resto de los miembros y que así se identifiquen oportunidades de ayudarse unos a otros.
- **Sprint review meeting:** Al finalizar un sprint se revisa el trabajo realizado y se enseña a quien esté interesado el resultado implementado (la demo).
- **Sprint retrospective:** Sirve para reflexionar sobre lo que haya pasado durante el sprint y para identificar oportunidades de mejora en el proceso de desarrollo.

5.2. Otros modelos ágiles

5.2.1. El modelo XP

La Programación Extrema (XP) es una metodología ágil de desarrollo de software enfocada en mejorar la calidad del software y la capacidad de respuesta ante los cambios en los requisitos del cliente. Promueve la colaboración constante entre el equipo de desarrollo y el cliente, entregas frecuentes y mejoras continuas, con el objetivo de crear software de alta calidad de manera eficiente y flexible.

XP utiliza un enfoque orientado a objetos y sigue un conjunto de reglas y prácticas organizadas en cuatro actividades clave: planeación, diseño, codificación y pruebas. Estas actividades estructuran el proceso XP y facilitan un desarrollo ágil y adaptable. En la Figura 15 se puede ver el proceso XP.

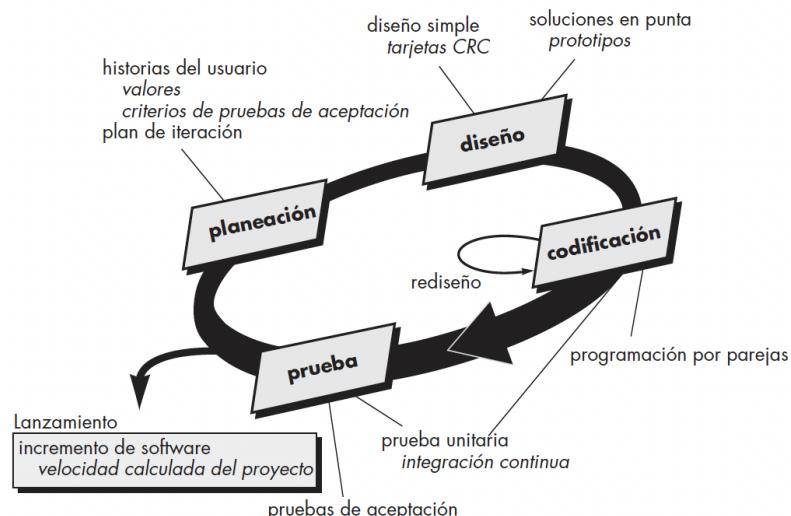


Figura 15: El proceso de la programación extrema [2]

Las 4 actividades fundamentales en XP:

- **Planeación:** Permite al equipo comprender los requisitos del cliente y adaptarse a sus necesidades mediante una comunicación continua, asegurando que el software refleje las expectativas reales y pueda ajustarse rápidamente a los cambios.
- **Diseño:** Promueve la simplicidad y flexibilidad mediante un diseño evolutivo que se ajusta a nuevas necesidades, evitando complejidades innecesarias y manteniendo el sistema fácil de modificar.

- **Codificación:** Es el núcleo de XP, enfocada en un código claro y sencillo, con programación en parejas para garantizar calidad y detectar errores de forma rápida, fomentando la colaboración y el aprendizaje mutuo.
- **Pruebas:** Garantizan la calidad del software mediante pruebas automatizadas que validan cada funcionalidad antes de su implementación, asegurando la fiabilidad del sistema desde el inicio a través del desarrollo basado en pruebas (TDD).

5.2.2. Kanban

El método **Kanban** es un método ágil de gestión de proyectos y de flujo de trabajo, originado en el sistema de producción de Toyota. Su principal objetivo es mejorar la eficiencia y la productividad al visualizar las tareas y gestionar el flujo de trabajo de manera continua. A diferencia de otros métodos ágiles, como Scrum, Kanban es más flexible y no requiere iteraciones o *sprints* predefinidos. En lugar de ello, se centra en gestionar un flujo constante de trabajo y facilitar la entrega continua de productos.

Kanban en sí depende de seis principios básicos:

1. Visualizar el flujo de trabajo mediante el uso de un tablero de Kanban (Figura 16) dividido en columnas que representan diferentes etapas del proceso de trabajo. Las tareas del tablero podrían contener historias de usuarios individuales o defectos recién descubiertos en notas adhesivas y el equipo los pasaría de “por hacer” a “haciendo” y luego “hecho” a medida que el proyecto progrese.
2. Limitar la cantidad de trabajo en progreso (WIP) en cada etapa del flujo. Esto significa que solo se puede trabajar en un número limitado de tareas a la vez, los desarrolladores deben completar su tarea actual antes de comenzar otra.
3. Gestionar el flujo de trabajo para mantener una entrega continua.
4. Hacer explícitas las políticas del proceso.
5. Enfocarse en la mejora continua mediante la creación de lazos de retroalimentación.
6. Realizar los cambios en el proceso de forma colaborativa e involucrar a todos los miembros del equipo y demás partes interesadas según sea necesario.

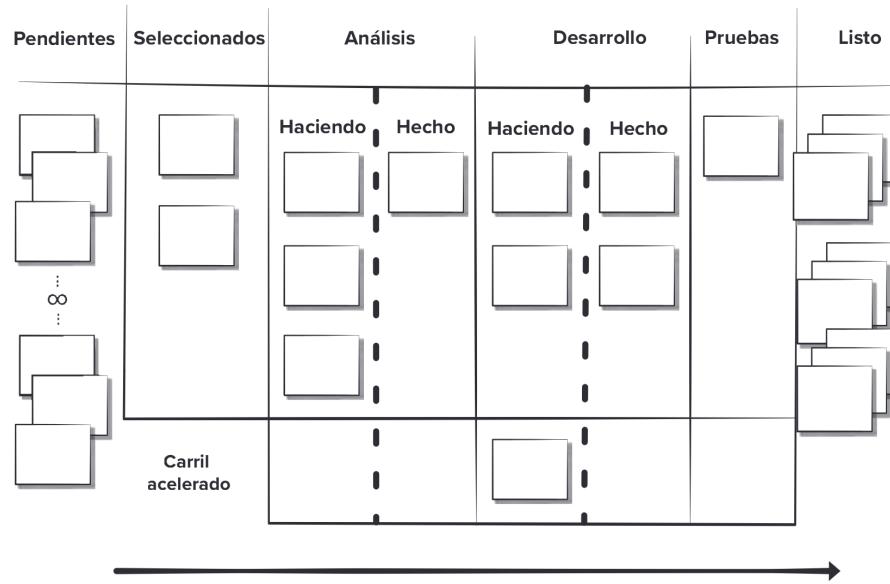


Figura 16: Tablero de Kanban [2]

Referencias

- [1] Irene T. Luque Ruiz, Apuntes de la Asignatura Ingeniería del Software (2013).
- [2] Roger S. Pressman, Ingeniería del Software, Un Enfoque Práctico, 9th Edition, Mc Graw Hill, 2021.
- [3] Metodologías de desarrollo de software: ¿qué son?, <https://www.santanderopenacademy.com/es/blog/metodologias-desarrollo-software.html>.
- [4] Barry Boehm, Spiral Development: Experience, Principles, and Refinements, 2000.
- [5] Manifiesto por el Desarrollo Ágil de Software, <https://agilemanifesto.org/iso/es/manifesto.html>.