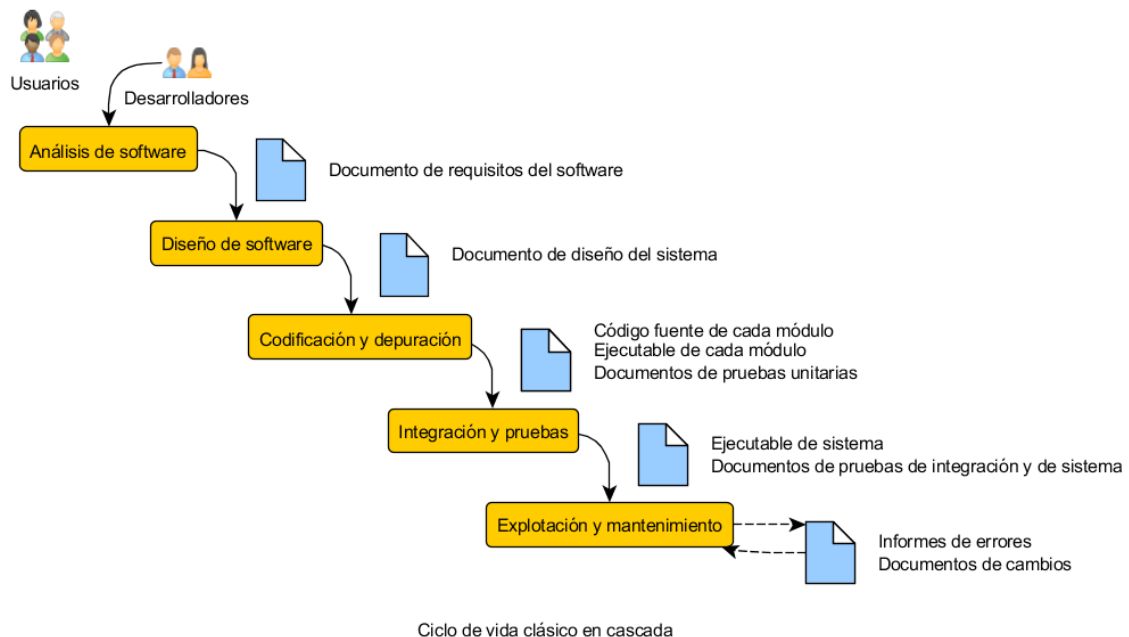


U1 Desarrollo de software

3. Procesos de desarrollo de software

3.1 “Un mundo ideal”. Ciclo de vida clásico en cascada.

En este modelo se identifican distintas actividades o fases que han de realizarse precisamente en el orden indicado, de manera que el resultado de una de estas fases es el elemento de entrada para la fase siguiente.



El modelo de ciclo de vida en cascada trata de aislar cada fase de la siguiente, de manera que las fases sucesivas puedan ser realizadas por grupos de personas diferentes, facilitando la especialización. De esta manera podemos encontrar perfiles profesionales diferentes, tales como: analista, diseñador, programador, etc.

Para conseguir esta relativa independencia es fundamental que en cada fase se genere una información de salida precisa y suficiente para que otras personas puedan acometer la fase siguiente. Se insiste así en la necesidad de establecer unos modelos de documentación apropiados. En general se suelen exigir y aparecer los documentos o artefactos siguientes:

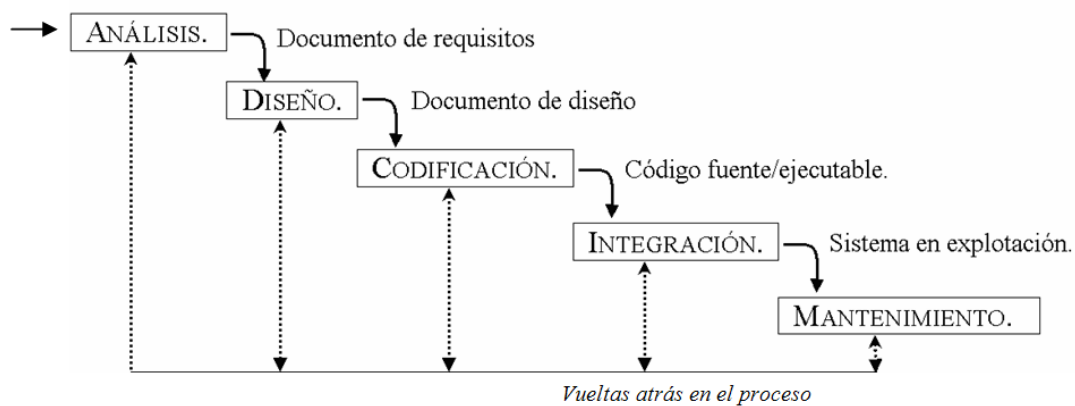
- A) DOCUMENTO DE REQUISITOS DEL SOFTWARE: (en inglés SRD: Software Requirements Document): como producto de la fase de análisis. Consiste en una especificación precisa y completa de lo que debe hacer el sistema, prescindiendo de los detalles internos.
- B) DOCUMENTO DE DISEÑO DEL SOFTWARE: (SDD: Software Design Document): como producto de la fase de diseño. Consiste en una descripción de la estructura global del sistema, y la especificación de qué debe hacer cada una de sus partes y cómo se combinan unas con otras.
- C) CODIGO FUENTE: como producto de la fase de codificación. Contiene los programas fuente, en el lenguaje de programación elegido, y debidamente comentados para conseguir que se entiendan con claridad. También se pueden entender como producto de esta fase los ejecutables de cada módulo.
- D) EL SISTEMA SOFTWARE, ejecutable: como producto de la fase de integración. Deben documentarse también las pruebas realizadas sobre el sistema completo.

- E) DOCUMENTOS DE CAMBIOS: tras cada modificación realizada durante el mantenimiento. Los documentos de cambios suelen recopilar información de cada problema detectado, descripción de la solución adoptada, y las modificaciones realizadas sobre el sistema para aplicar dicha solución.

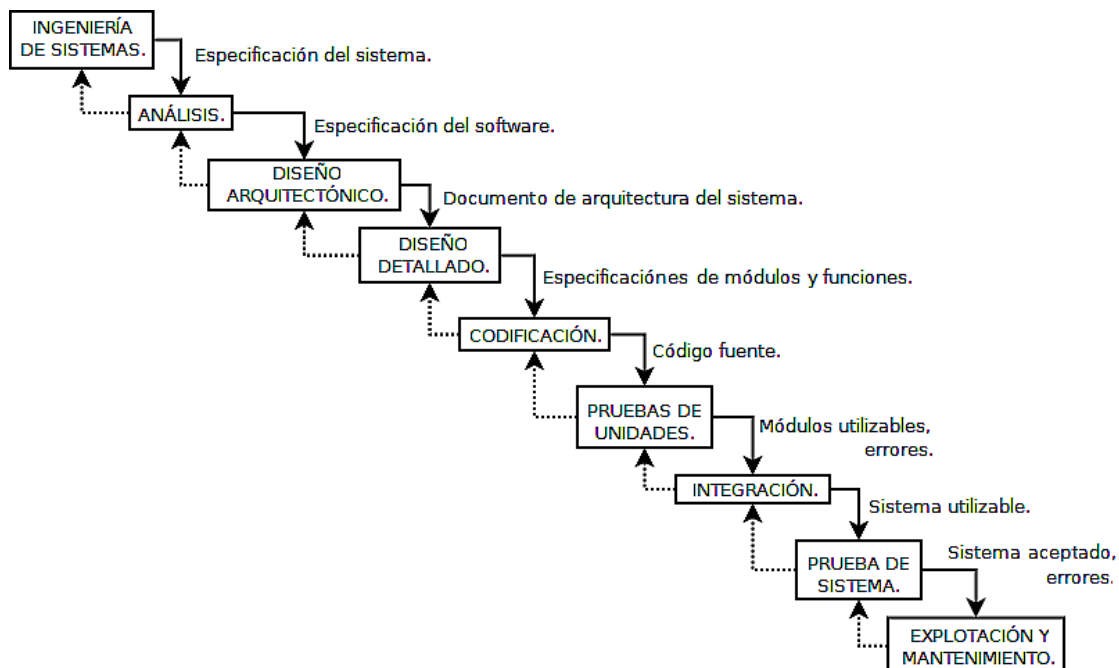
El modelo en cascada hace énfasis también en la necesidad de terminar correctamente cada fase antes de comenzar la siguiente. Esto se debe a que los errores producidos en una fase son muy costosos de corregir si ya se ha realizado el trabajo de las fases siguientes.

Para detectar los errores lo antes posible, y evitar que se propaguen a fases posteriores, se establece un proceso de revisión al completar cada fase antes de pasar a la siguiente. Esta revisión se realiza fundamentalmente sobre la documentación producida en cada fase, y se hace de manera formal, siguiendo una lista de comprobaciones establecidas de antemano.

Si, a pesar de todo, durante la realización de una fase se detectan errores de fases anteriores, será necesario rehacer parte del trabajo volviendo a un punto anterior del ciclo de vida (línea discontinua de la figura). Una variante natural del modelo en cascada, que resalta estos retrocesos, es el **modelo en cascada con realimentación**:




Existen también variantes en cuanto a la colección particular de actividades que componen el proceso en cascada. Ejemplo de **variante ampliada del modelo en cascada**:



3.2 Tareas y fases del desarrollo de software

Análisis del software o Especificación de requisitos: Consiste en analizar las necesidades de los usuarios potenciales del software para determinar QUÉ debe hacer el sistema a desarrollar, y de acuerdo con ello, escribir una especificación precisa de dicho sistema. Muchas veces estas especificaciones se concretan como una lista de requisitos que el sistema debe cumplir a la hora de ponerlo en explotación.



UNIVERSIDAD DE ZARAGOZA
CENTRO DE INVESTIGACIONES
CIENTÍFICAS Y TECNOLÓGICAS
INVESTIGACIÓN EN SISTEMAS DE INFORMACIÓN
INVESTIGACIÓN EN SISTEMAS DE INFORMACIÓN

LISTADO DE REQUISITOS FUNCIONALES

Nº	Función	Ref.
FR01	El sistema debe permitir al usuario crear categorías para cualquier tipo de producto.	APROBADO
FR02	El sistema debe permitir al usuario crear productos y asociarles un precio y un tipo de producto.	APROBADO
FR03	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR04	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR05	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR06	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR07	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR08	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR09	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR10	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR11	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR12	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR13	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR14	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR15	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR16	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR17	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR18	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR19	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO
FR20	El sistema debe permitir al usuario crear productos con un nombre, código de barras y una descripción.	APROBADO

Página 1

Diseño del software: La tarea de diseño de software consiste en dar solución mediante elementos informáticos al problema descrito en la tarea de análisis.

Si en la tarea de análisis se define ¿QUÉ? se quiere que cumpla el software en desarrollo, en la de tarea de diseño se elige ¿CÓMO? lo va a satisfacer.

Para ello, en esta tarea el sistema se descompone y organiza en módulos que puedan ser desarrollados por separado, para aprovechar las ventajas de la división del trabajo y poder hacer el desarrollo en equipo. El resultado del diseño es la colección de especificaciones de cada elemento componente.



Ejemplo de descomposición del sistema en módulos

Codificación o implementación: En esta fase se programa cada elemento componente por separado, es decir, se escribe el código fuente de cada uno. Normalmente se harán también algunas pruebas o ensayos para garantizar en lo posible que dicho código funciona correctamente.

Pruebas: Se realizan pruebas al sistema para verificar su funcionamiento. Pueden realizarse pruebas de cada elemento, o de una parte o de todo el sistema en conjunto. Se considera adecuado que el equipo de pruebas sea diferente al del desarrollo. Dependiendo de los resultados de esta tarea, puede ser necesario rehacer el trabajo de alguna otra fase o tarea.

Integración: Esta tarea consiste en combinar diferentes elementos componentes del sistema y probar el sistema completo. Habrá que hacer pruebas exhaustivas para garantizar el funcionamiento correcto de cada subsistema y del conjunto, antes de ser puesto en explotación.

Documentación: Puede considerarse una tarea aparte o el resultado de otras tareas. Un sistema bien documentado es más fácilmente mantenible. Además de la documentación técnica del sistema, suele necesitarse una guía de instalación y la guía de usuario.

Explotación: Se pasa a decir que el sistema está en explotación cuando se encuentra funcionando “en casa del cliente”, habiendo “dado por válido” (validación, aceptación) su comportamiento previamente.



Mantenimiento: Durante la explotación del sistema software, es necesario realizar cambios ocasionales, bien para corregir errores no detectados con anterioridad o bien para introducir mejoras. Para ello hay que rehacer parte de los trabajos anteriores.

3.3 El otro extremo: “Sí, somos cutres, ¿y qué?”.

Con esta visión se ignoraría las recomendaciones de la ingeniería del software, como las de seguir unas fases y pautas, y se comenzaría a codificar directamente. El trabajo en grupo se basaría sólo en la comunicación oral. El programa iría creciendo en funcionalidad a medida que se va descubriendo “por el camino” las diferentes necesidades de usuario y se sucedan los errores en el programa.

Al haber trabajado de esta manera serán frecuentes los “parcheos” sobre el programa realizado, por lo general no muy elegantes, pero funcionales. Se pierden las ventajas de una estructura de programa ordenada, impidiendo en ocasiones el trabajo en grupo en paralelo. Una ventaja es que no se pierde tiempo en tareas y documentación de diseño, de análisis, ni otros formalismos. De hecho, ni siquiera resulta necesario conocer ninguna técnica organizativa. Como peor desventaja se aprecia la posibilidad de “quedar mal” ante el cliente por la entrega de un software “excesivamente artesanal”.

3.4 “Ni tanto, ni tan calvo”. Uso de prototipos.

Los modelos clásicos tienen el inconveniente de estar muy orientados hacia una forma de desarrollo lineal, en que cada fase del desarrollo tiene una duración limitada en el tiempo de forma que una vez terminada una fase, puede dedicarse a otra cosa los recursos humanos o materiales que se han empleado en ella. No se contemplan de manera organizada las vueltas atrás en el desarrollo.

Desgraciadamente hay situaciones en que no es posible garantizar adecuadamente al concluir una fase que su resultado es correcto. Esto ocurre por ejemplo, en sistemas innovadores, en que no se dispone de una experiencia previa para contrastar si las decisiones adoptadas durante el análisis y diseño son apropiadas.

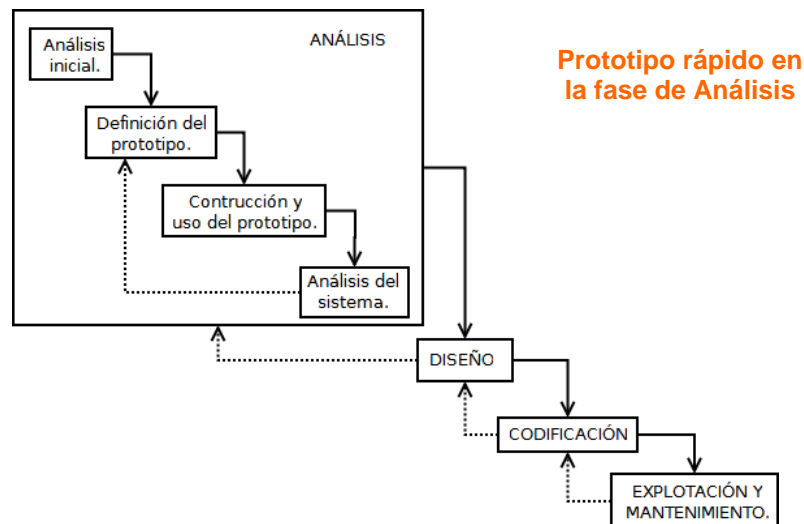
Definición de prototipo: Un prototipo es un sistema auxiliar que permite probar experimentalmente ciertas soluciones parciales a las necesidades del usuario o a los requisitos del sistema.

Normalmente se distinguen dos clases de prototipos, según se pretenda aprovechar el código del mismo, o sólo la experiencia obtenida con él, tal como indicaremos a continuación.

a) Prototipos rápidos.

Estos prototipos son aquellos cuya finalidad es sólo adquirir experiencia. Estos prototipos se aprovechan dentro de las fases de análisis y/o diseño de un sistema, para experimentar algunas alternativas y garantizar en lo posible que las decisiones tomadas son correctas. Una vez completadas estas fases, el sistema final se codifica totalmente partiendo de cero (de ahí que también se denominen prototipos de *usar y tirar*, “*throw-away*” o *maquetas*).

A continuación está representado un ejemplo de ciclo de vida que utiliza un prototipo “de usar y tirar” en la fase de análisis:



b) Prototipos evolutivos. Proceso de desarrollo por prototipos evolutivos

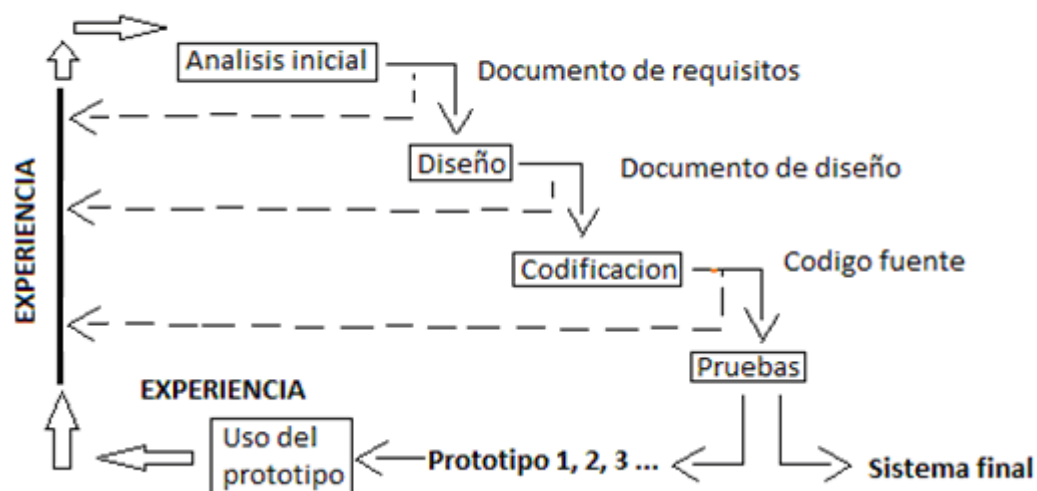
Otra manera de utilizar un prototipo es tratando de aprovechar al máximo su código. En este caso el prototipo se desarrollará sobre el mismo soporte hardware/software que el sistema final, pero sólo realizará algunas de las funciones, o en general, será solo una realización parcial del sistema deseado.

El prototipo inicial se construirá tras unas fases parciales de análisis y diseño. La experimentación con el prototipo permitirá avanzar en esas fases parciales, y a continuación ampliar el prototipo inicial para irlo convirtiendo en el sistema final mediante adiciones sucesivas.

De esta manera se van construyendo distintas versiones del prototipo, cada vez más completas, hasta obtener el sistema deseado. Al mismo tiempo los documentos de especificación, diseño, etc. van siendo también desarrollados progresivamente.

Este modelo puede considerarse como un proceso iterativo en bucle sobre el proceso cascada, de manera que en cada **iteración** se hace solo una parte del desarrollo, avanzando un poco en cada fase (esto es, produciendo un **incremento** del sistema). Gráficamente podría quedar así:

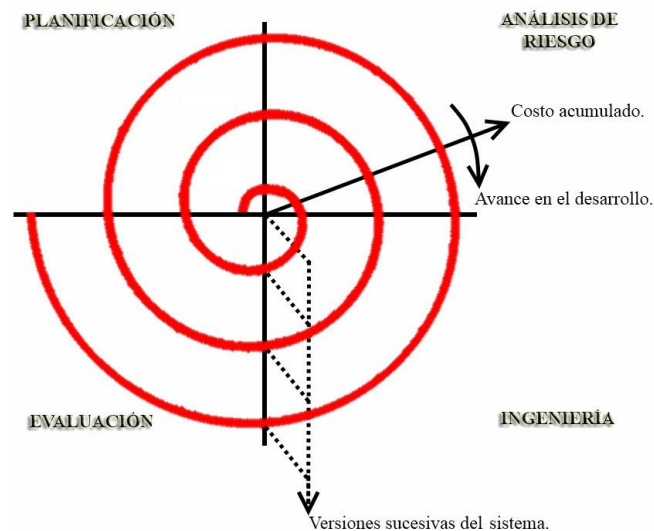
Ciclo de vida evolutivo



3.5 Modelo en espiral.

El modelo espiral desarrollado por B. Boehm, puede considerarse como un refinamiento del modelo evolutivo general. Como elemento distintivo respecto a otros modelos del ciclo de vida, introduce la actividad de *análisis de riesgo* como elemento fundamental para guiar la evolución del proceso de desarrollo.

El ciclo de iteración del modelo evolutivo, se convierte en una espiral al añadir como dimensión radial una indicación del esfuerzo total realizado hasta cada momento, que será un valor siempre creciente, como se indica en la figura. Las distintas actividades se representan sobre cada cuadrante:



En cada ciclo de la espiral se realiza una parte de cada actividad (PLANIFICACIÓN, ANÁLISIS DE RIESGO, INGENIERÍA Y EVALUACIÓN), y por lo tanto, del desarrollo total.

Las actividades de **planificación** sirven para establecer el contexto del desarrollo, y decidir qué parte del mismo se abordara en ese ciclo de la espiral.

Las actividades de **análisis de riesgo** consisten en evaluar diferentes alternativas para la realización de la parte del desarrollo elegida, seleccionando la más ventajosa y tomando precauciones para evitar los inconvenientes previstos.

Las actividades de **ingeniería** corresponden a las indicadas en los modelos clásicos: análisis, diseño, codificación, etc. Su resultado será ir obteniendo en cada ciclo una versión más completa del sistema.

Las actividades de **evaluación** analizan los resultados de la fase de ingeniería, habitualmente con la colaboración del "cliente" para el que se realiza el desarrollo. El resultado de esta evaluación se utiliza como información de entrada para la planificación del ciclo siguiente.

Según qué parte del desarrollo se decida hacer en cada ciclo, tendremos distintas variantes del modelo espiral, que podría significar en un parecido con otros modelos. En cualquier caso, el modelo espiral se distingue por las actividades de análisis de riesgo, que no aparecen en los modelos anteriores. El hecho de realizar iteraciones consecutivas sobre estas fases permitiría, en caso de que el proyecto deje de resultar viable, su detención en el momento oportuno, minimizando los costes.

Preguntas: ¿A qué modelo se parecería más el modelo en espiral si...

- a) en cada vuelta se realizase en la fase de ingeniería sólo una de las tareas de análisis, diseño, ... pero de manera completa?
- b) en cada vuelta se realiza en la fase de ingeniería lo suficiente de todas y cada una de esas fases como para obtener una versión ampliada del sistema?

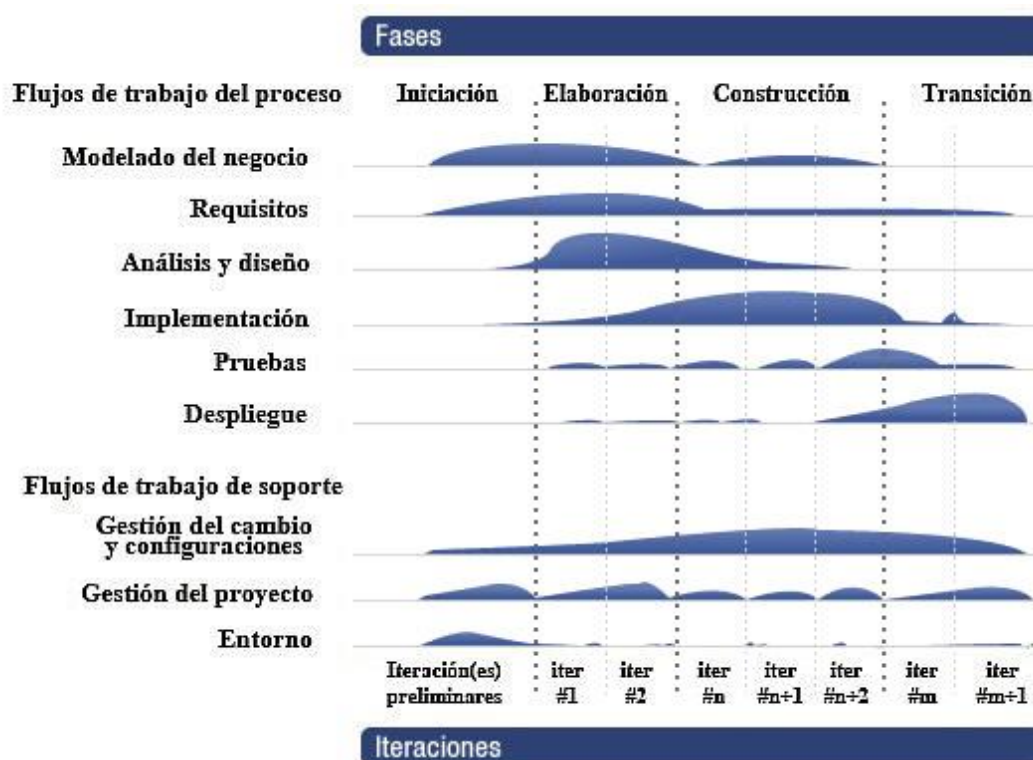
3.6 Proceso Unificado de Rational (RUP).

Tras el desarrollo de UML (Lenguaje Unificado de Modelado) por los autores Ivar Jacobson, Grady Booch y James Rumbaugh se ideó un proceso de desarrollo de software íntimamente relacionado con este lenguaje de modelado.

Según sus autores, el proceso unificado va más allá de un mero análisis y diseño orientado a objetos. Como descriptores fundamentales se dan los siguientes. El *PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE* es un proceso:

1. Basado en componentes
2. Dirigido por los casos de uso
3. Centrado en la arquitectura
4. Iterativo e incremental

El siguiente gráfico muestra flujos de trabajo propuestos, tanto para el proceso de desarrollo como para el soporte de éste, y la cantidad de trabajo dedicada a estos, tanto por iteración como por fases.



Dirigido por los casos de uso significa que los casos de uso se utilizan como un artefacto básico para establecer el comportamiento deseado del sistema, para verificar y validar la arquitectura, para las pruebas y para la comunicación entre las personas involucradas en el proyecto.

Centrado en la arquitectura significa que la arquitectura del sistema se utiliza como un artefacto básico para conceptualizar, construir, gestionar y hacer evolucionar el sistema en desarrollo.

Un *proceso iterativo* es aquel que incluye la repetición de la secuencia de actividades para el desarrollo del sistema. Necesitará por tanto de algún mecanismo para la gestión de los artefactos y ejecutables del sistema. Un *proceso incremental* es aquel donde, integrando la arquitectura anterior, cada nueva versión incorpora ampliaciones o mejoras sobre las anteriores. Se recomienda que los procesos iterativos e incrementales estén *dirigidos por el riesgo*, lo que significa que cada nueva versión se encarga de atacar y reducir los riesgos más significativos para el éxito del proyecto.

Actividad 1: ¿Eres capaz de seguir las iteraciones, de entender la propuesta del gráfico anterior?

Actividad 2: Trata de relacionar las tareas y el avance con el paso del tiempo entre el *proceso unificado* con el *modelo clásico en cascada*. ¿Encuentras la relación? ¿Con qué coincidirían las fases de Elaboración, Construcción y Transición respecto al *modelo clásico en cascada*?

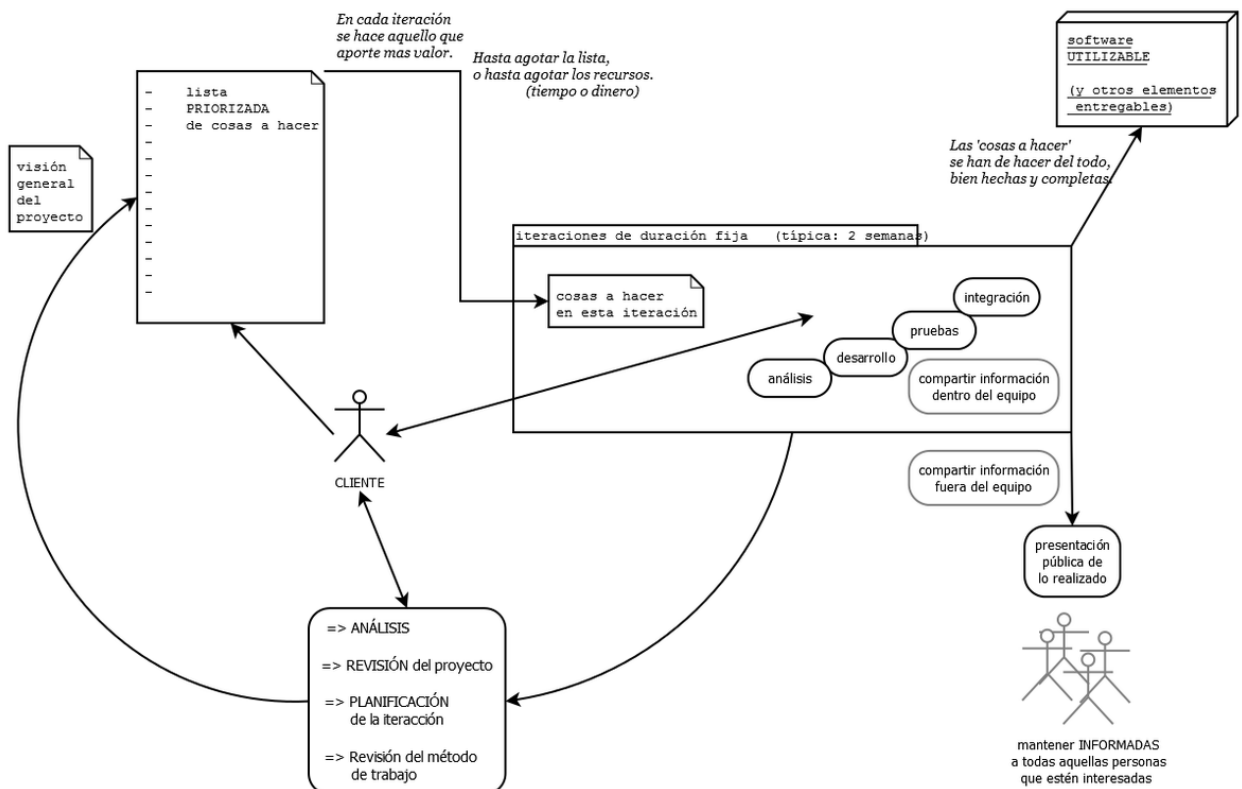
3.7 Metodologías de desarrollo ágiles

Estas metodologías consideran a las anteriores “pesadas”. El trabajo en tareas que no tienen que ver con la generación de código y sí de otra documentación puede llevar mucho tiempo. Además, no siempre los requisitos son estables en el tiempo. Las metodologías de desarrollo ágiles se centran en el avance en el desarrollo, y en el equipo de trabajo; en la adaptabilidad a circunstancias cambiantes, más que en la estabilidad.

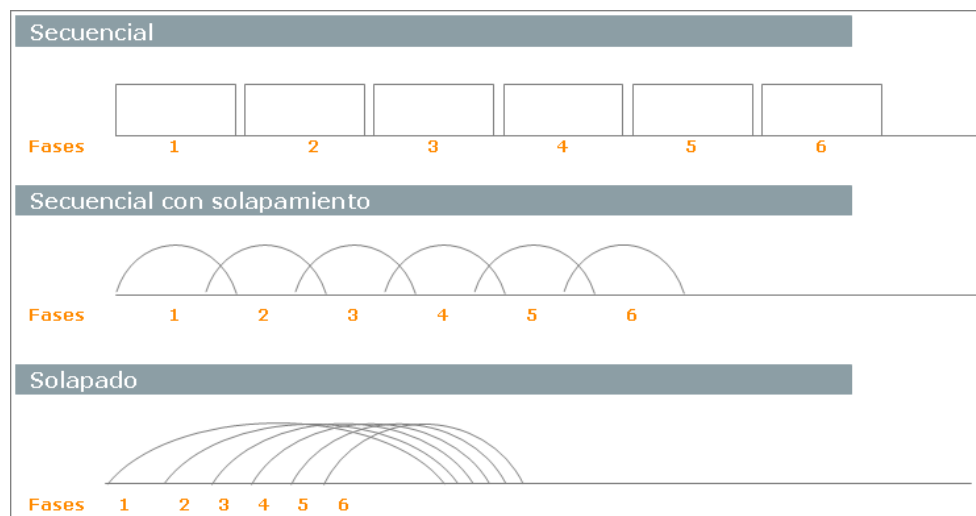
Algunos ejemplos de estas metodologías son:

- eXtreme Programming (XP),
- Test Driven Development (TDD),
- Design Driven Development (3D),
- Agile Project Management,
- SCRUM,
- Kanban, ...

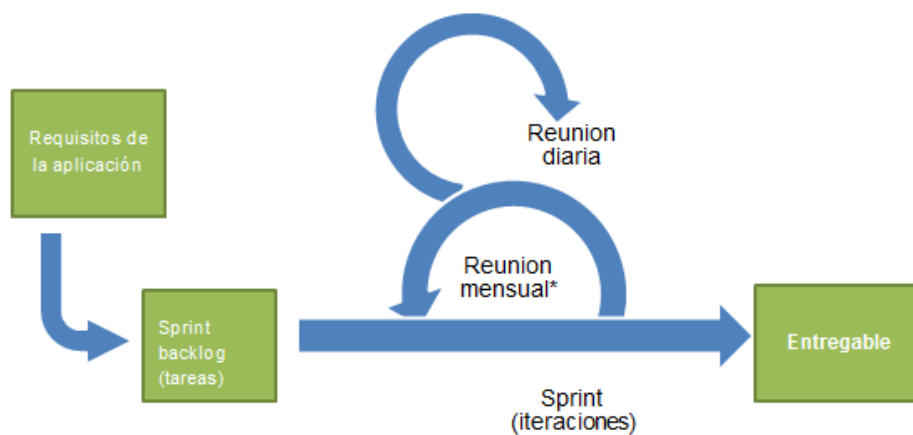
En el siguiente gráfico se puede ver alguna de las prioridades de estas metodologías:



En el caso de SCRUM, por ejemplo, unas iteraciones se solapan con las otras:



y los periodos de desarrollo están manejados mediante “sprints” hacia un objetivo muy concreto, con reuniones del equipo muy frecuentes, cada semana, cada día o cada mes:



3.8 Estándares de desarrollo de software

Ciertas organizaciones como IEEE, DoD, ESA, ISO o ANSI han desarrollado estándares para el desarrollo del software.

El Consejo Superior de Informática del Ministerio para las Administraciones Públicas (MAP) desarrolló “METRICA-2”, para los sistemas de información de las administraciones públicas, basada en la metodología de análisis y diseño estructurado de Yourdon/De Marco.

Actualmente está vigente METRICA Versión 3:

The screenshot shows the PAE (Portal de Administración Electrónica) website. The header includes the Spanish flag, the text 'GOBIERNO DE ESPAÑA', and 'PAE portal administración electrónica'. Navigation links for 'Castellano', 'Català', 'Euskara', 'Galego', 'Valencià', and 'English' are present, along with 'Escuchar', 'Identificarse', and 'Registrarse'. A search bar is on the right. Below the header, a menu bar contains 'Actualidad', 'Estrategias', 'Soluciones - CTT', 'Observatorio - OBSAE', 'Documentación', and 'Organización'. The breadcrumb trail reads: 'Estás en: Inicio > Documentación > Metodologías y Guías > Métrica v.3'. The left sidebar has a 'Documentación' section with links to 'Legislación nacional', 'Legislación autonómica', 'Legislación Unión Europea', and 'Metodologías y Guías'. The main content area is titled 'Métrica v.3' and includes social media icons, a 'Valorar' button, and a download count of '1.7K'. The text describes the methodology as a tool for system lifecycle support. It lists the components of the methodology: 'Metodología de Planificación, Desarrollo y Mantenimiento de sistemas de información'. A detailed list of documents follows, including 'Introducción', 'Planificación de Sistemas de Información (Proceso PSII)', 'Estudio de Viabilidad del Sistema (Proceso EVS)', 'Análisis del Sistema de Información (Proceso ASI)', 'Diseño del Sistema de Información (Proceso DSI)', 'Construcción del Sistema de Información (Proceso CSI)', 'Implantación y Aceptación del Sistema (Proceso IAS)', and 'Mantenimiento del Sistema de Información (Proceso MSI)'. It also lists 'Interfaces' (Aseguramiento de la Calidad, Seguridad, Gestión de Configuración, Gestión de Proyectos), 'Técnicas', and 'Participantes'. A contact email 'metrica@correo.gob.es' is provided at the bottom.

AENOR distribuye normas para procesos de ciclo de vida del software como la Norma ISO/IEC 12207 y un método para evaluar la calidad y madurez de los mismos (con niveles desde el más básico, el 0, hasta el más maduro, el 5), en la Norma ISO/IEC 15504, parte 2 y 7.

Algunas organizaciones de estandarización:

- IEEE: Institute of Electrical and Electronics Engineer
- DoD: Department of Defense
- ESA: European Space Agency
- ISO: International Standards Organization
- ANSI: American National Standards Institute
- AENOR: Asociación Española de Normalización y Certificación
- IEC: International Electrotechnical Commission