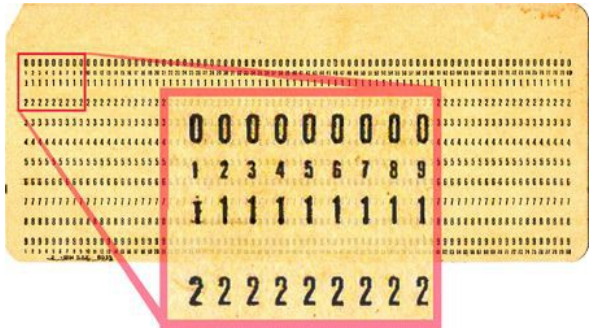
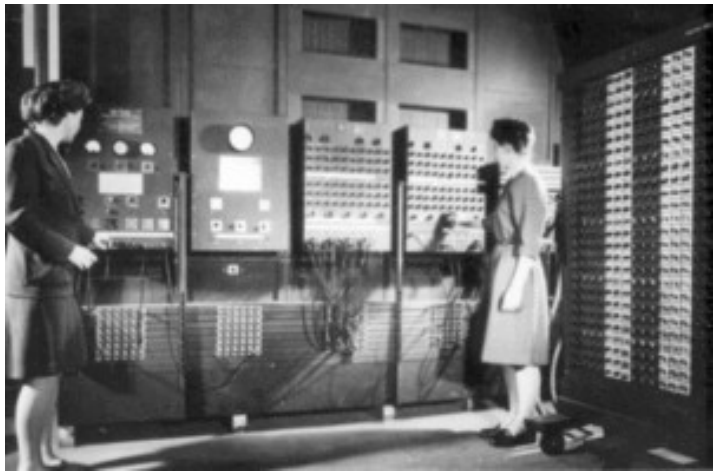
Introducción al desarrollo del software

Entornos de desarrollo

Historia del software (1930-50).

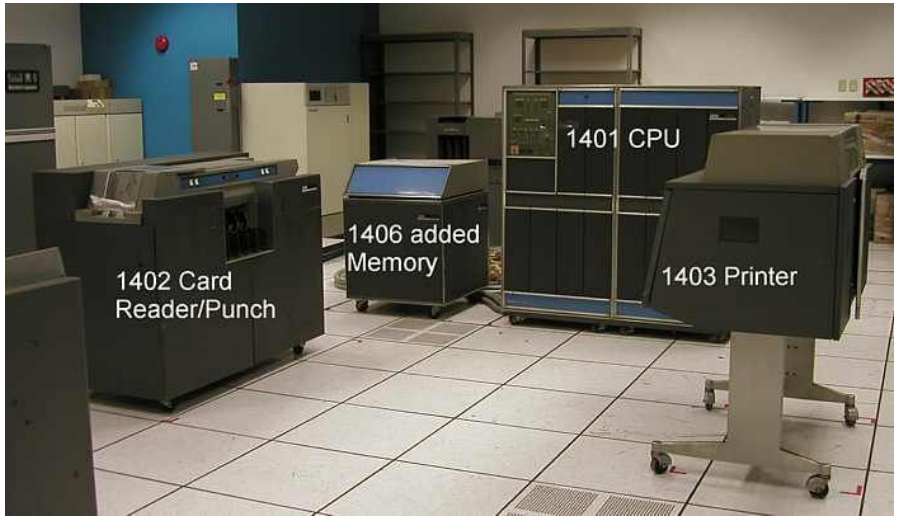
Desde la década de 1930 hasta la década de 1950, las tarjetas perforadas se convirtieron en la fuerza motriz de las empresas, ya que se utilizaron en prácticamente todas las máquinas de contabilidad de oficina. Las tarjetas fueron creadas con lenguajes de programación como FORTRAN de IBM y COBOL del Departamento de Defensa de EEUU.
Proyecto ENIAC.





Historia del software (1950-60).

El término «software» se creó a finales de 1950 y pronto fue adoptado por toda la industria.
Dividiendo el software en dos tipos principales: software de sistema y programa aplicaciones.
Software del sistema incluye los procesos generales de la ejecución del programa, tales como compiladores y sistema operativo de disco.
Aplicaciones del programa como pueden ser aplicaciones de oficina.
Mainframe IBM 1401 (1959)



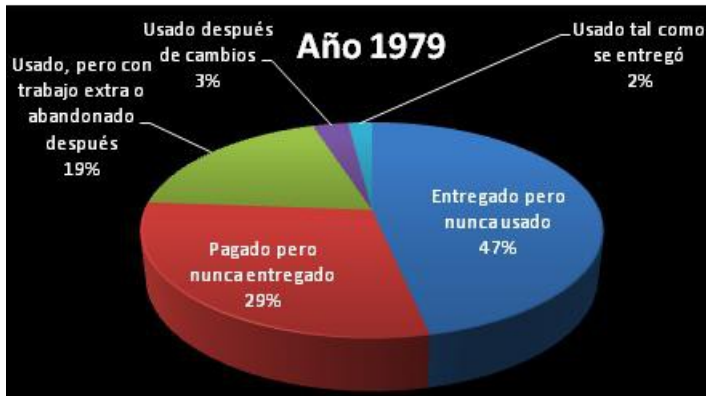
Historia del software (1960-80).

A finales de los 60, la potencia de las maquinas empezó a aumentar de forma considerable. Empezaron a aparecer los lenguajes de programación de alto nivel, y las maquinas necesitaban programas mucho más complejos de los desarrollados hasta la época.
En definitiva, fue un salto tremendo en cuanto a potencial de hardware, que no fue acompañado por un salto en el desarrollo de software

Historia del software (1960-80).

En esta época, se empezó a concebir el Software como producto, y se empezaron a desarrollar algunos proyectos para que funcionaran en las máquinas de la época.
Pero aparecieron importantes problemas:

- Los productos excedían la estimación de costes.
- Había retrasos en las entregas.
- Las prestaciones no eran las solicitadas.
- El mantenimiento se hacía extremadamente complicado y a veces imposible, las modificaciones tenían un coste prohibitivo…



Crisis del software

La Crisis del software fue acuñada principios de los años 70, cuando la ingeniería de software era prácticamente inexistente.
El término expresaba las dificultades del desarrollo de software frente al rápido crecimiento de la demanda por software, de la complejidad de los problemas a ser resueltos y de la inexistencia de técnicas establecidas para el desarrollo de sistemas que funcionaran adecuadamente o pudieran ser validados.
Historia del Altair 8800

(1974)

Crisis del software

Una de las principales causas de todo esto, si no la principal, era el enfoque dado al proceso de desarrollo de software, el cual era malo e incluso a veces era inexistente.

En este proceso, solo ¼ del tiempo de desarrollo se dedicaba a las fases de análisis, diseño, codificación y pruebas, y más de ¾ del tiempo se dedicaba a correcciones y mantenimiento.

Es evidente que dedicándole sol ¼ del tiempo a las primeras fases, se arrastran errores graves, sobre todo procedentes de las fases de análisis y diseño, lo que dificultaba muchísimo la implementación, produciendo constantes paradas y retrocesos para revisar este análisis/diseño.

Proyectos fallidos de la crisis del software

Accidente de un F-18 (1986): En abril de 1986 un avión de combate se estrelló por culpa de un giro descontrolado atribuido a una expresión “if then”, para la cual no había una expresión “else”, debido a que los desarrolladores del software lo consideraron innecesario.

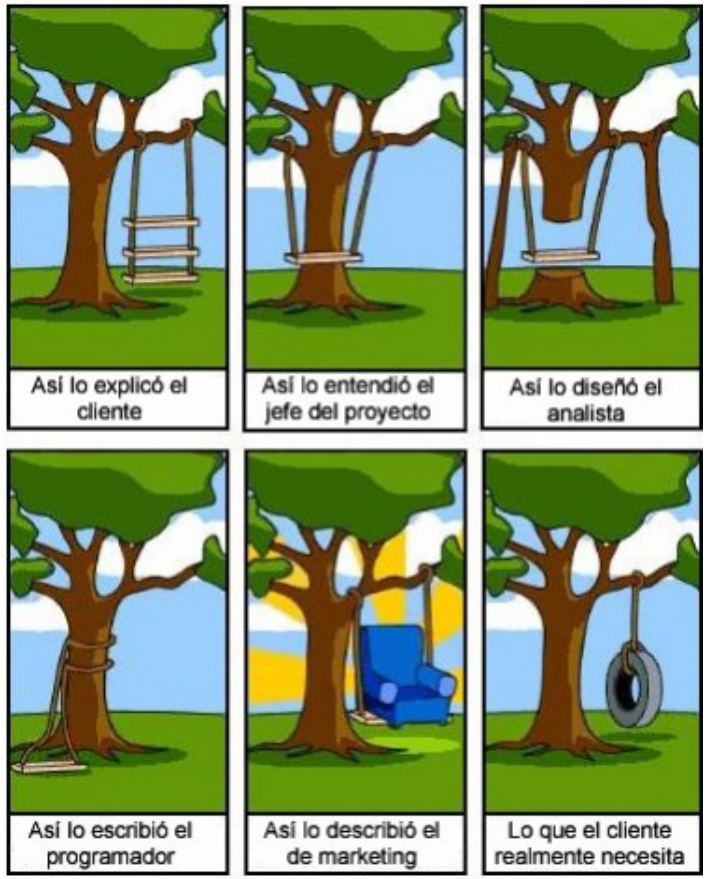
Muertes por el Therac-25 (1985-1987): El Therac-25 fue una máquina de radioterapia que causó la muerte de varios pacientes en diversos hospitales de Estados Unidos y Canadá, debido a las radiaciones de alto poder aplicadas sin control, las cuales fueron atribuidas a la falta de control de calidad del software médico.

Sobrecosto, retraso y cancelación en el sistema del Bank of America (1988): En el año de 1988, este banco invirtió 23 millones de dólares en un sistema computarizado llamado MasterNet, el cual servía para contabilidad y reportes de fideicomisos. No obstante, para que el sistema funcionara, se tuvo que invertir 60 millones de dólares más, por lo que finalmente el sistema fue cancelado.

Nacimiento de la Ingeniería del Software.

La primera conferencia sobre Ingeniería de Software fue allá por 1968, en Múnich, financiada por la OTAN. Allí fue donde se adoptó el término, hasta entonces prácticamente desconocido, de «ingeniería de software», y quien primero lo usó fue Fritz Bauer.

La **Ingeniería del Software** es una disciplina que se ocupa de proporcionar un marco adecuado para la gestión de los proyectos software, y para el diseño de aplicaciones empresariales con una arquitectura software que favorezca su mantenimiento y evolución a lo largo del tiempo.



Según estimaciones, el 26% de los grandes proyectos de software fracasan, el 48% deben modificarse drásticamente y sólo el 26% tienen rotundo éxito. La principal causa del fracaso de un proyecto es la falta de una buena planificación de las etapas y mala gestión de los pasos a seguir. ¿Por qué el porcentaje de fracaso es tan grande? ¿Por qué piensas que estas causas son tan determinantes?

Ingeniería del software

Estos facilitan la gestión del proceso de desarrollo y suministran a los desarrolladores bases para construir de forma productiva software de alta calidad.

La ingeniería del software abarca un conjunto de tres elementos clave:

- Métodos
- Herramientas
- Procedimientos

Métodos

Indican cómo construir técnicamente el software, y abarcan una amplia serie de tareas que incluyen la planificación y estimación de proyectos, el análisis de requisitos, el diseño de estructuras de datos programas y procedimientos, la codificación, las pruebas y el mantenimiento.

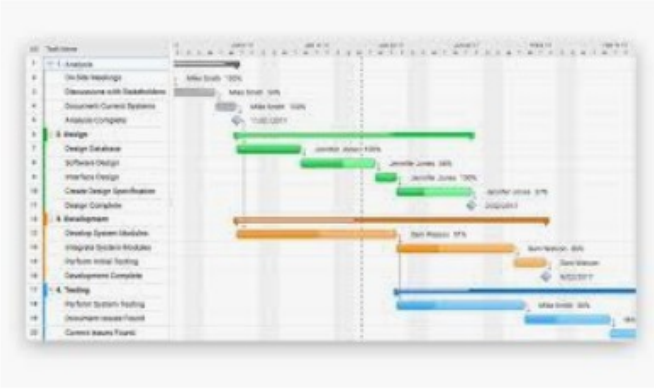
Los **métodos** introducen frecuentemente una notación específica para la tarea en cuestión y una serie de criterios de calidad.

Ejemplo: Cuestionarios, entrevistas, grupos de trabajos …

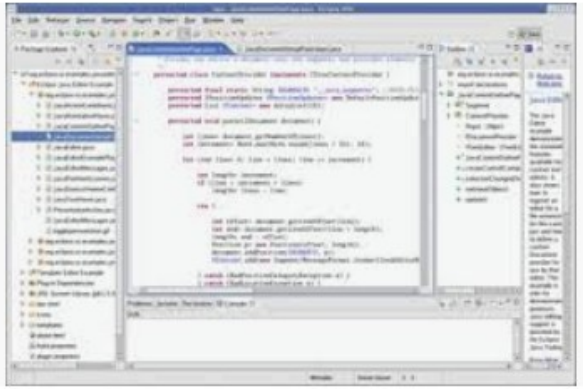
Herramientas

Las **herramientas** proporcionan un soporte automático o semiautomático para utilizar los métodos. Existen herramientas automatizadas para cada una de las fases vistas anteriormente, y sistemas que integran las herramientas de cada fase de forma que sirven para todo el proceso de desarrollo.

MagicDraw



Microsoft Project Integration - ProjectManager.com



Eclipse desktop & web IDEs | The Eclipse Founda...

Procedimientos

Definen la secuencia en que se aplican los métodos, los documentos que se requieren, los controles que permiten asegurar la calidad y las directrices que permiten a los gestores evaluar los progresos.

En los procesos se distinguen 4 fases básicas:

- **Análisis**: Proceso de reunión de requisitos funcionales y no funcionales de un sistema.
- **Diseño**: Se refiere ala establecimiento de las estructuras de datos, la arquitectura general de software, interfaz…
- **Implementación**: traducimos el diseño en un forma legible por la máquina. Lo programamos.
- **Pruebas**: Una vez generado el software comienzan las pruebas. Para demostrar que no es encuentran fallos.

Fases de desarrollo del software.

Ciclo de vida:

"Todas las etapas por las que pasa un proyecto de desarrollo de software desde que se inicia hasta que se finaliza y se considera una solución completa, correcta y estable".

Ingeniería del software.

- QUÉ
  - Se va a desarrollar

1. Estudio del Sistema
2. Planificación del Sistema
3. Especificación de Requisitos
4. Análisis del Sistema

- CÓMO
  - Se va a desarrollar

1. Diseño (arquitectura, estructura de datos, ...)
2. Codificación e implementación
3. Pruebas

- CAMBIO
  - Que se puede producir en el sistema

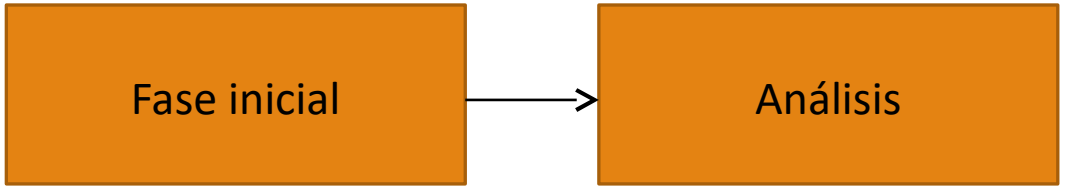
Adaptación o mejora del sistema

Ingeniería del software: Fases de desarrollo.

Fase inicial

En esta fase se hacen estimaciones de **viabilidad** del proyecto, es decir, si es rentable o no y se establecen las bases de las fases del proyecto.

Ingeniería del software: Fases de desarrollo.



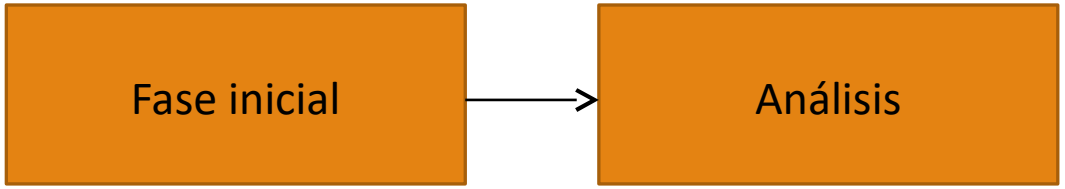
Se analiza el problema, es decir, se recopila, examina y formulan los requisitos del cliente.

Es fundamental, tener claro **QUÉ hay que hacer (comprender el problema)**.

Se realizan varias reuniones con el cliente, generando documentación en la que se recopilan los requisitos, Especificación de Requisitos del Software (**ERS**) .

Al finalizar estas reuniones, se genera un acuerdo en el que el cliente se compromete a no variar los requisitos hasta terminar una primera **release**.

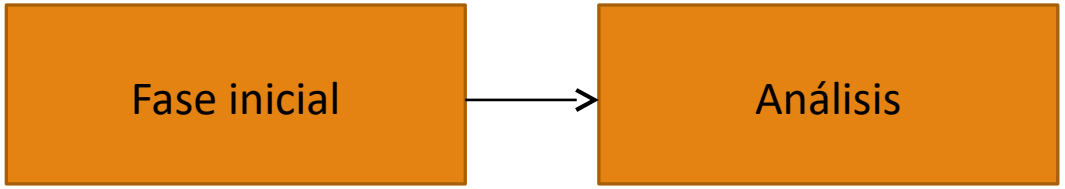
Ingeniería del software: Fases de desarrollo.



Para facilitar la comunicación con el cliente, se utilizan técnicas como:

- Casos de uso (técnica **UML**, en la que se representan los requisitos funcionales del sistema).
- Brainstorming (desde distintos puntos de vista).
- Desarrollo conjunto de aplicaciones (**JAD**) y Planificación conjunta de requisitos (**JRP**). Apoya en la dinámica de grupos y, respectivamente, los productos generados comprenden los requisitos claves o estratégicos.
- Prototipos (versión inicial del sistema).

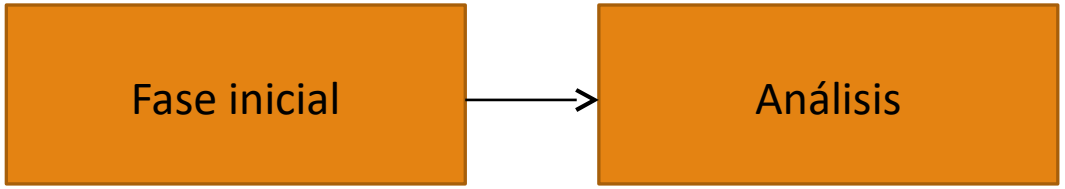
Ingeniería del software: Fases de desarrollo.



Hay dos tipos de requisitos:

- **Requisitos funcionales**: Describen con detalle la función que realiza el sistema.
- **Requisitos no funcionales**: Tratan de las características del sistema como fiabilidad, mantenibilidad, plataforma hardware, S.O, restricciones, etc.

Ingeniería del software: Fases de desarrollo.



Para su representación, se pueden utilizar:

- Diagrama de Flujo de Datos (**DFD**) de distintos niveles: Representan el flujo de datos entre los distintos procesos, entidades externas y almacenes que forman el sistema.
- Diagrama de Flujo de Control (**DFC**): Similares a los DFD pero muestran el flujo de control.
- Diagramas de transición de Estados (**DTE**): Representa cómo se comporta el sistema como consecuencia de sucesos externos.
- Diagrama Entidad/Relación (**DER**): Se usa para representar los datos y sus relaciones entre ellos.

Ingeniería del software: Fases de desarrollo.

Consiste en dividir el problema en partes y se determina la función de cada una de estas. Se decide **CÓMO se hace (CÓMO se resuelve el problema).**

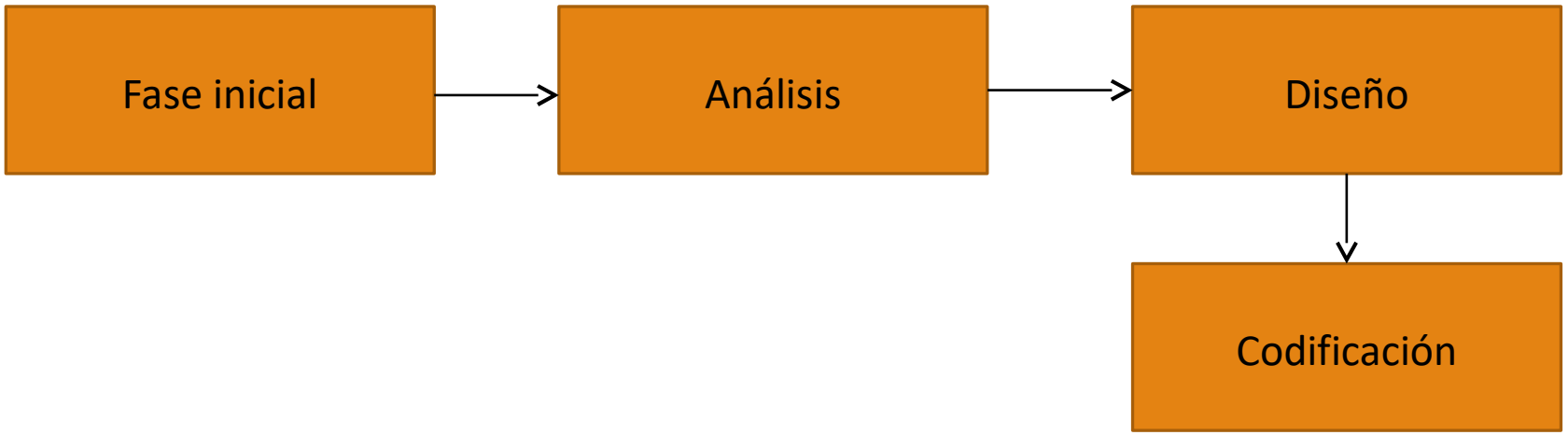
Se elabora una documentación técnica y detallada de cada módulo del sistema.

La documentación es realizada por el *analista junto al jefe de proyecto*.

Los tipos de diseños más utilizados son:

- **Diseño estructurado (clásico)**, está basado en el flujo de datos a través del sistema.
- **Diseño Orientado a Objetos**, el sistema se entiende como un conjunto de objetos. Los cuales, tienen propiedades, comportamientos y se comunican entre ellos.

Ingeniería del software: Fases de desarrollo.

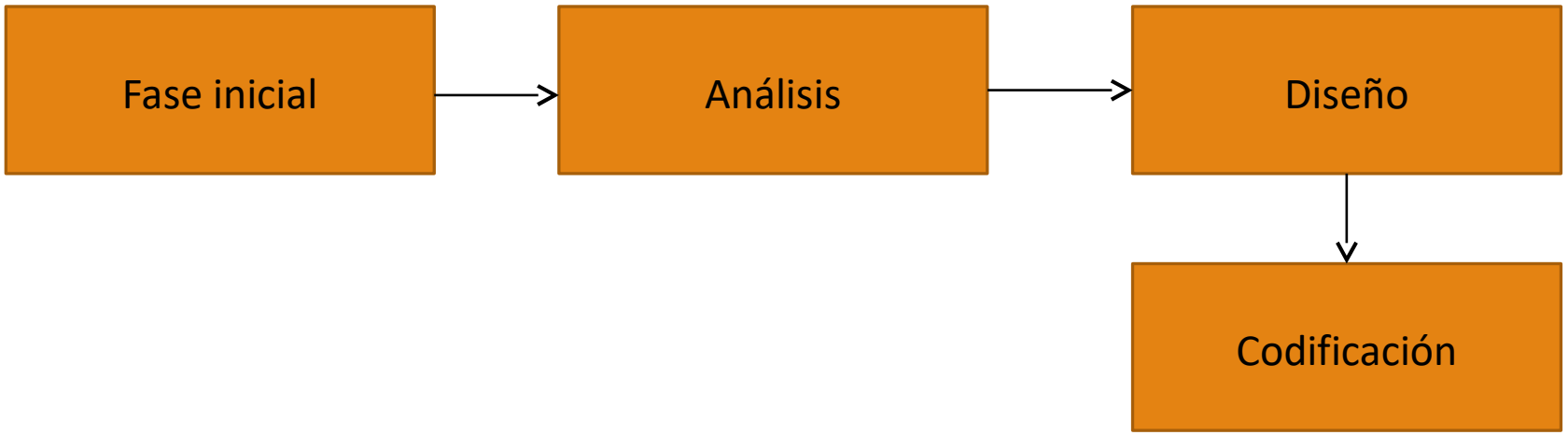


El programador, codificará el software según los criterios seleccionados en la etapa de diseño.

Al trabajar en equipo, debe haber unas normas de codificación y estilo, claras y homogéneas

Cuanto más exhaustivo haya sido el análisis y el diseño, la tarea será más sencilla, pero nunca está exento de necesitar reanálisis o un rediseño si se encuentran problemas al programar.

Ingeniería del software: Fases de desarrollo.

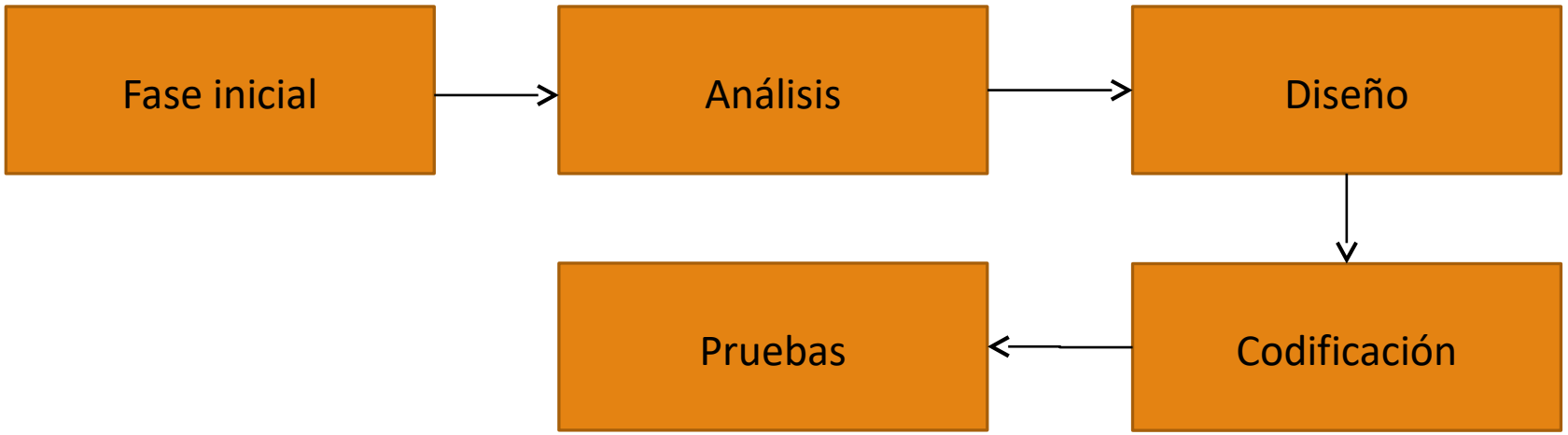


En esta fase, se genera documentación del código que se desarrolla: entradas, librerías, etc.

Se pueden distinguir dos tipos:

- **Documentación del proceso**: Comprende las fases de desarrollo y mantenimiento.
- **Documentación del producto**: Describe el producto que se está desarrollando, se divide en dos tipos: documentación del usuario y documentación del sistema.

Ingeniería del software: Fases de desarrollo.

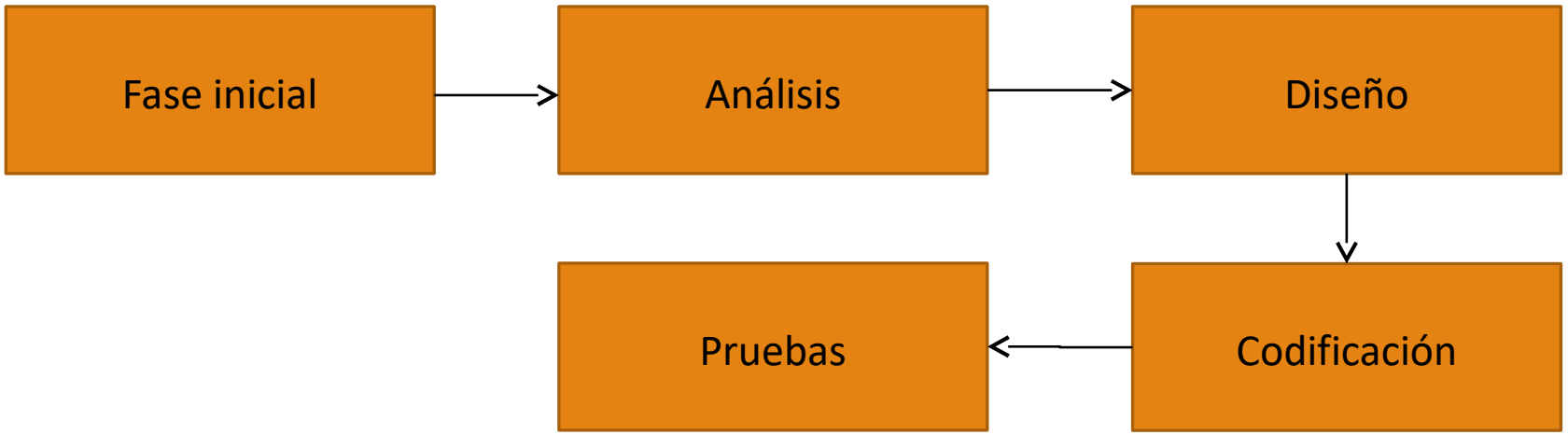


Se realizan pruebas para verificar que el programa se ha realizado acuerdo las especificaciones originales, es decir, se comprueba lo que **DEBE HACER**.

Existen varios tipos de pruebas:

1) **Funcionales**: Se validan las especificaciones establecidas por el cliente y, posteriormente, será validada por este.
2) **Estructurales**: Se realizan pruebas técnicas sin necesidad del consentimiento del cliente. Se realizarán cargas reales de datos.

Ingeniería del software: Fases de desarrollo.

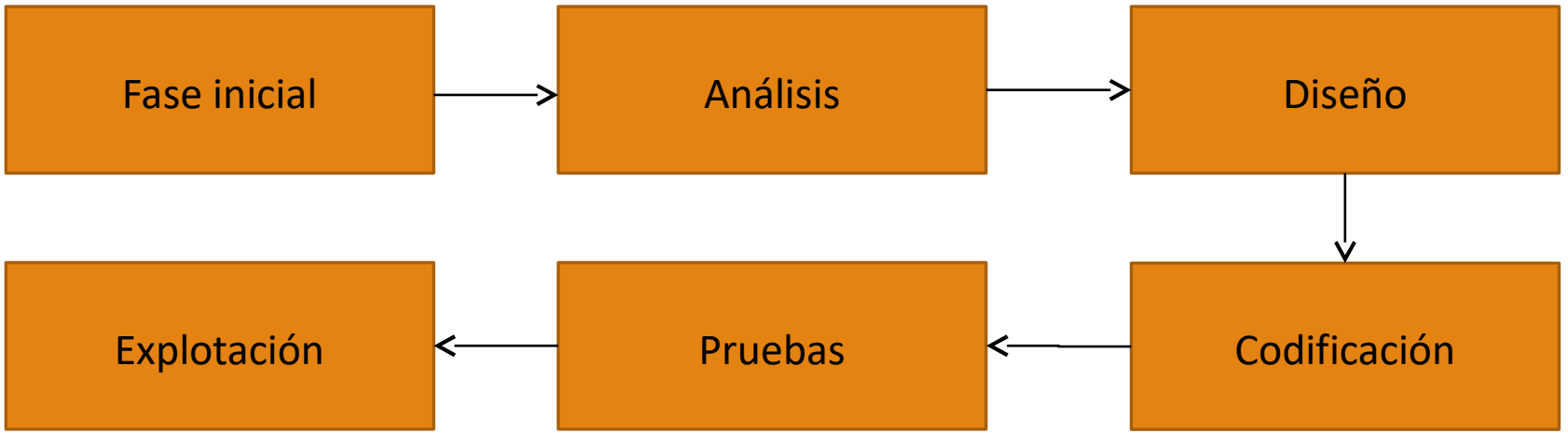


Esta fase comprende las siguientes tareas:

- **Verificación**: Se trata de comprobar si se está construyendo el software correctamente, es decir si implementa correctamente la función.
- **Validación**: Trata de comprobar si el software construido se ajusta a los requisitos del cliente.

En general, las pruebas las debería realizar, personal diferente al que codificó, con una amplia experiencia en programación.

Ingeniería del software: Fases de desarrollo.

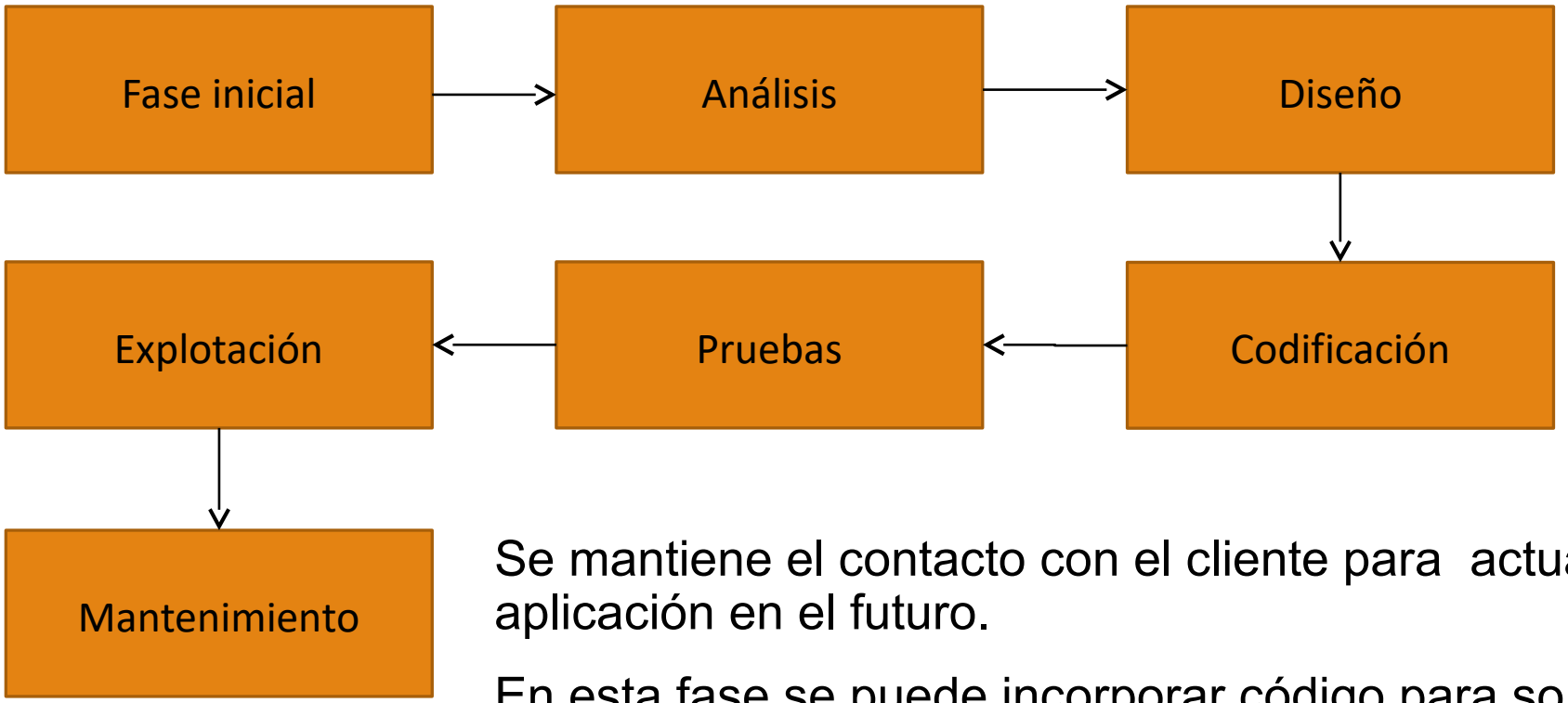


Consiste en la instalación y puesta en marcha del software en el entorno de trabajo del cliente, es decir, el software es instalado y utilizado en un entorno real de uso cotidiano.

Es la fase más larga, ya que suele conllevar múltiples incidencias (**bugs**) y nuevas necesidades.

En esta etapa se realizan las siguientes tareas: Estrategia para la implantación, pruebas de operación, uso operacional del sistema y soporte al usuario.

Ingeniería del software: Fases de desarrollo.



Se mantiene el contacto con el cliente para actualizar y modificar la aplicación en el futuro.

En esta fase se puede incorporar código para soluciones a problemas, ampliar la funcionalidad para un cliente, …

Ingeniería del software. Fases de desarrollo.

Existen distintos tipos de mantenimiento:

- **Mantenimiento adaptativo**: Tiene como objetivo la modificación del software por los cambios que se produzcan, tanto en el hardware (ejemplo: nueva CPU) como en el software (ejemplo: OS) del entorno en el que se ejecutan. Este mantenimiento es el mantenimiento más frecuente.
- **Mantenimiento correctivo**: Es muy probable que después de la entrega, el cliente encuentre fallos o defectos que deben ser corregidos
- **Mantenimiento perfectivo**: Consiste en modificar el producto SW para incorporar nuevas funcionalidades y nuevas mejoras en el rendimiento del software.
- **Mantenimiento preventivo**. Consiste en modificar el producto sin alterar las especificaciones del mismo, con el fin de mejorar y facilitar las tareas de mantenimiento (por ejemplo: reestructurar programas para mejorar su legibilidad, añadir comentarios para facilitar su comprensión, etc.).

Ingeniería del software: Ciclos de vida.

Un ciclo de vida es el conjunto de fases por las que un sistemas software pasa desde que surge la idea hasta que muere.

De los ciclos de vida destacan ciertos aspectos claves que son: las fases , las transiciones entre fases y las posibles entradas y salidas que surgen en cada transición.

Los ciclos de vida tradicionales son:

- Ciclo de vida lineal o en cascada.
- Ciclo de vida evolutivo

Ciclo de vida en Cascada

Ciclo de vida en el que las distintas fases se van encadenado las etapas de manera consecutiva. Es decir, la siguiente no empieza hasta que no haya terminado la anterior.

Este ciclo de vida no comprendía la retroalimentación entre fases y la posibilidad de volver atrás.

Por lo tanto, al llevarlo a la práctica lo que ocasionó que no se supiera reaccionar a errores ocurridos en una fases avanzadas.

Ventajas:

- ✓ Fácil de comprender, planificar y seguir.
- ✓ La calidad del producto resultante es alta.
- ✓ Permite trabajar con personal poco cualificado.

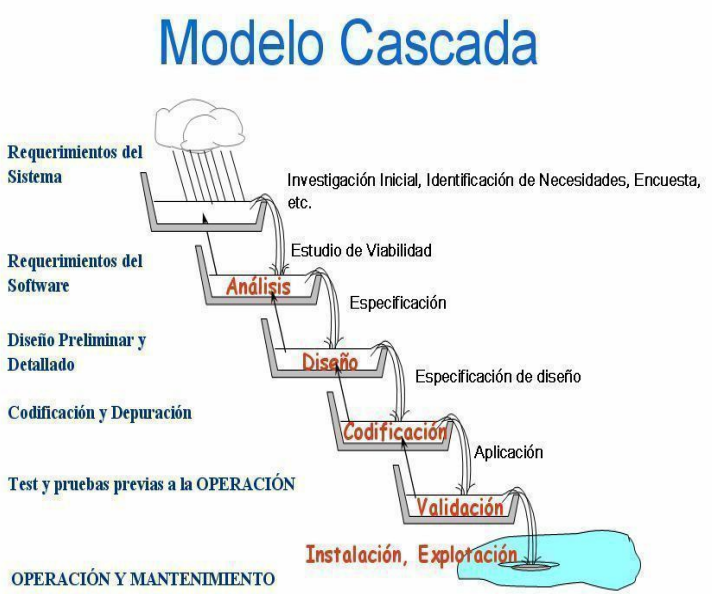
Inconvenientes:

- × Necesidad de tener todos los requisitos definidos desde el principio (pueden surgir imprevistos)
- × Es difícil volver atrás si se cometen errores en una etapa.
- × El producto no está disponible para su uso hasta que no está totalmente terminado.

Ciclo de vida en Cascada con retroalimentación

Se recomienda (dada la necesidad de tener todos los requisitos desde el principio y la dificultad de volver atrás) cuando:

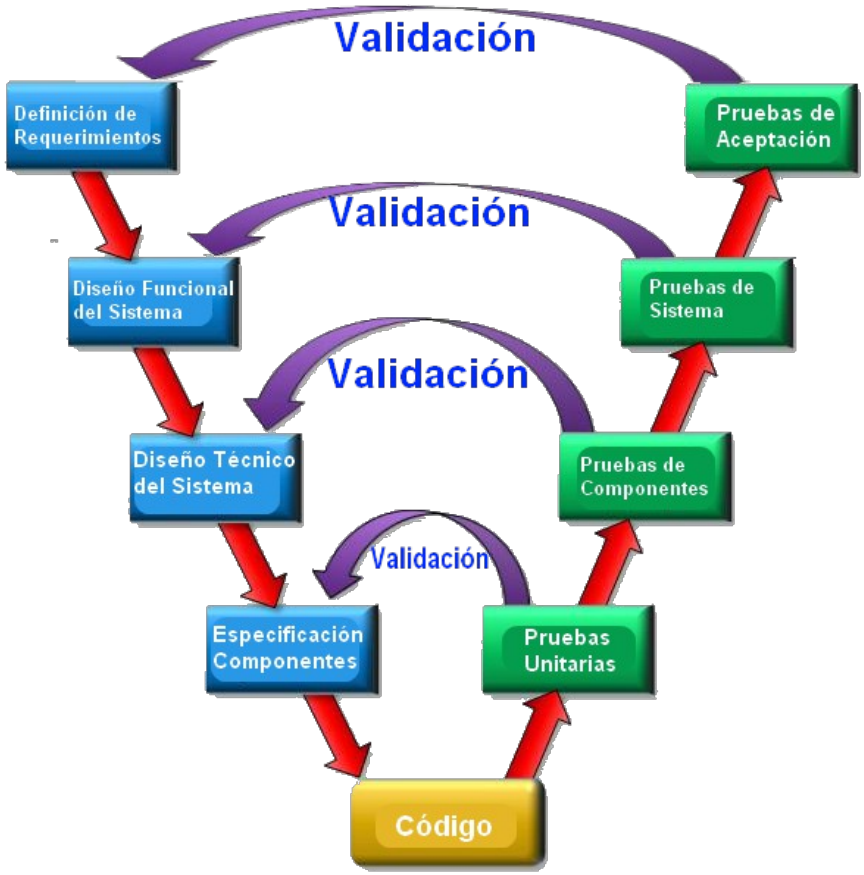
- El proyecto es similar a alguno que ya se haya realizado con éxito anteriormente
- Los requisitos son estables y están bien comprendidos.
- Los clientes no necesitan versiones intermedias.



Ciclo de vida en V

El modelo en V viene para corregir los fallos del modelo en cascada, incluyendo explícitamente la retroalimentación.

El problema es que los fallos detectados en fases tardías de un proyecto obligaban a empezar de nuevo, lo cual supone un gran sobrecoste.



Ciclo de vida incremental

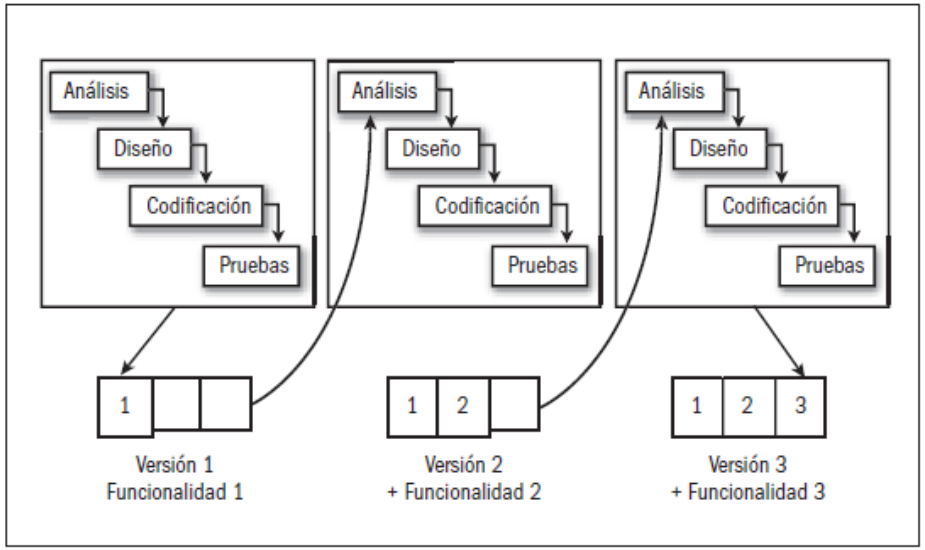
Este modelo busca reducir el riesgo que surge entre las necesidades del usuario y el producto final por malos entendidos durante la etapa de solicitud de requerimientos.

El ciclo se compone de iteraciones, las cuales, están compuestas por las fases básicas.

Al final de cada iteración se le entrega al cliente una versión mejorada o con mayores funcionalidades del producto.

El cliente al finalizar cada iteración, evalúa el producto y lo corrige o propone mejoras.

Estas iteraciones se repetirán hasta que se desarrolle un producto que satisfaga las necesidades del cliente.



Ciclo de vida incremental

Ventajas del desarrollo iterativo:

- No se necesitan conocer todos los requisitos desde el comienzo.
- Permite la entrega temprana al cliente de partes operativas del software.
- Permite separar la complejidad del proyecto, gracias a su desarrollo por parte de cada iteración o bloque.
- Las entregas facilitan la retroalimentación de los próximos entregables.

Debilidades de este modelo de desarrollo:

- Es difícil estimar el esfuerzo y el coste final necesario.
- Se tiene el riesgo de no acabar nunca.
- No es recomendable para sistemas en tiempo real, de alto índice de seguridad, de procesamiento distribuido, y/o de alto índice de riesgos

Ciclo de vida incremental: Modelo en espiral

MODELO ESPIRAL

CONCEPTO

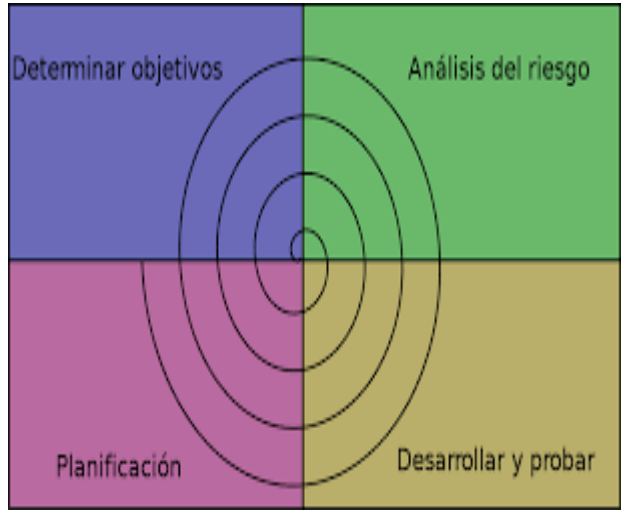
El modelo en espiral del proceso del software que originalmente fue propuesto por Boehm (1988) .El modelo en espiral es una de las mas recomendables para el desarrollo y creación de un programa, ya que consta de pocas etapas o fases, las cuales se van realizando en manera continua y cíclica.

Barry Boehm
Es un ingeniero informático estadounidense y también es profesor emérito de esta materia en el departamento de ciencias tecnológicas en la Universidad del Sur de California. Es conocido por sus múltiples aportes a este campo.

Ciclo de vida incremental: Modelo en espiral

El proceso de desarrollo de software se representa como una espiral, donde en cada ciclo se desarrolla una parte del mismo.
Cada ciclo está formado por cuatro :

- **Determinar objetivos:** Se identifican los objetivos, se ven las alternativas para alcanzarlos y las restricciones impuestas a la aplicación (costos, plazos, interfaz …).
- **Análisis de riesgos:** Un riesgo es por ejemplo, requisitos no comprendidos, mal diseño, errores en la implementación, etc. Utiliza la construcción de prototipos como mecanismo de reducción de riesgos.
- **Desarrollar y probar:** Se desarrolla la solución al problema en este ciclo y se verifica que es aceptable.
- **Planificación:** Revisar y evaluar todo lo realizado y con ello decidir si se continúa, entonces hay que planificar las fases del ciclo siguiente.



Ciclo de vida incremental: Modelo en espiral

Ventajas:

- ✓ No requiere una definición completa de los requisitos para empezar a funcionar
- ✓ Análisis del riesgo en todas las etapas
- ✓ Reduce riesgos del proyecto
- ✓ Incorpora objetivos de calidad

Inconvenientes:

- ➢ Es difícil evaluar los riesgos
- ➢ El costo del proyecto aumenta a medida que la espiral pasa por sucesivas iteraciones.
- ➢ El éxito del proyecto depende en gran medida de la fase de análisis de riesgos.

Metodologías

Metodología esta compuesta por modos sistemáticos de realizar, gestionar y administrar un proyecto a través de etapas y acciones, partiendo desde las necesidades del producto hasta cumplir con el objetivo para el que fue creado.

Metodologías Tradicionales
- RUP

Metodologías Ágiles
- XP
- SCRUM
- Kanban

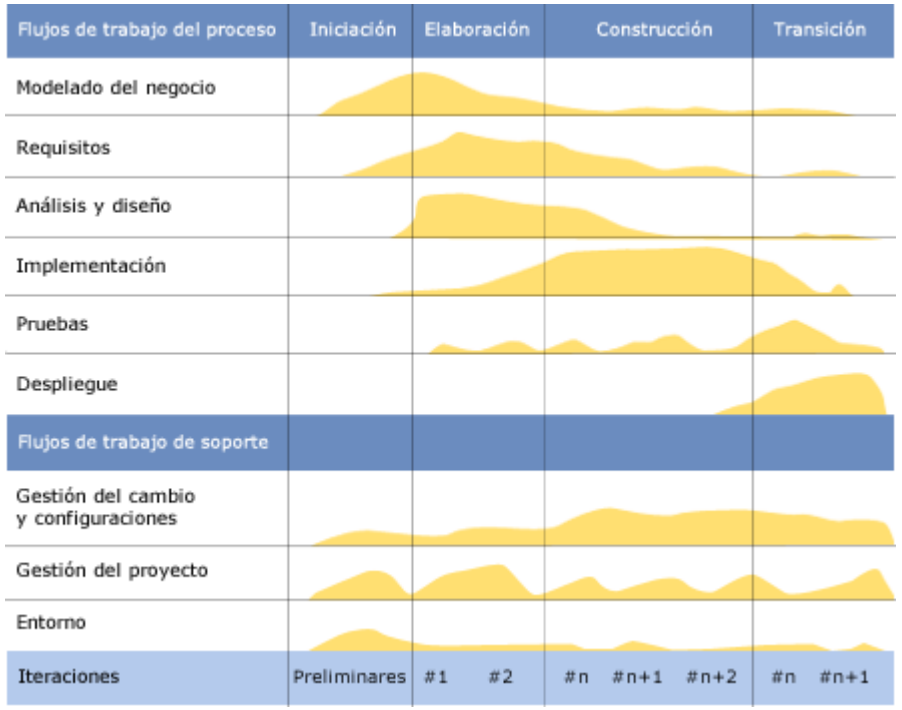
Metodología tradicional: RUP

Dirigido por casos de uso: los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo).

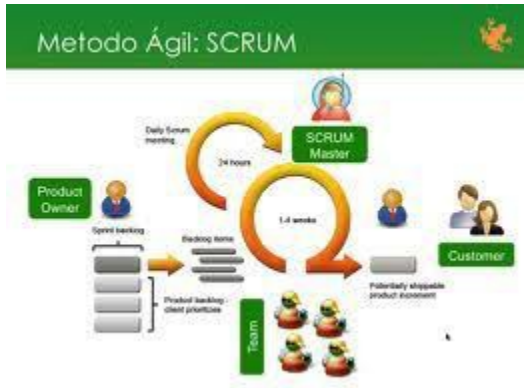
Centrado en la arquitectura: Se definen los Casos de Usos principales, los cimientos de la aplicación y sobre ello se va incrementando.

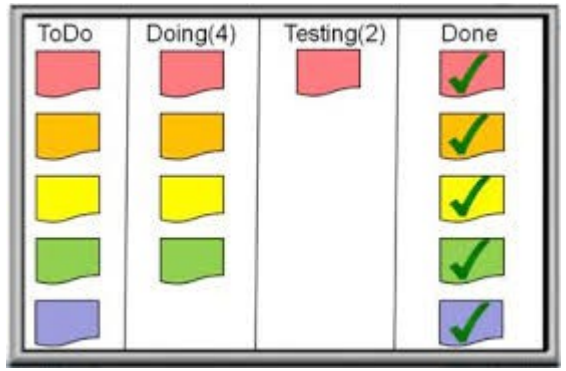
Iterativo e Incremental: Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros.

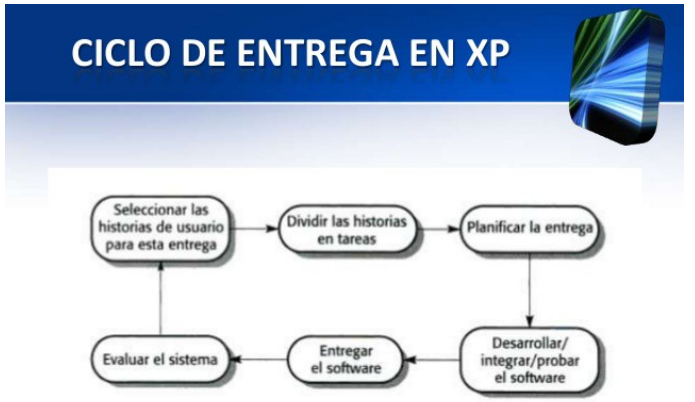
Uso extensivo de documentación.



Metodologías Ágiles





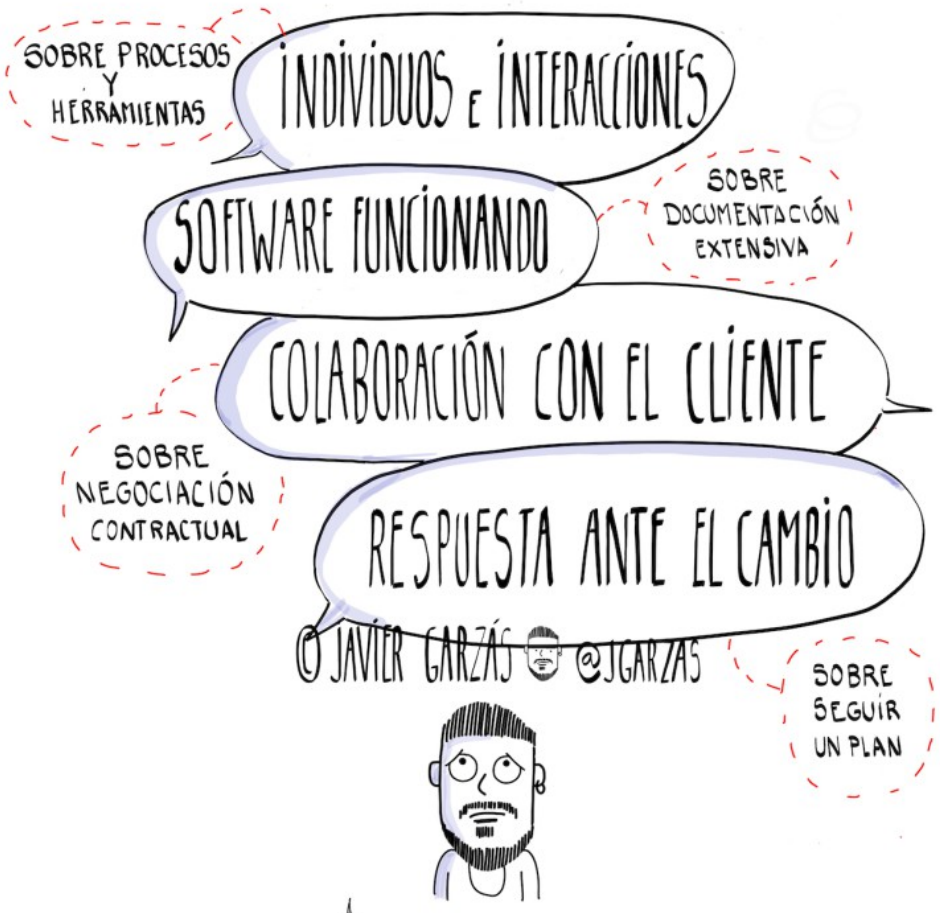


Manifiesto Ágil

El 12 de febrero de 2001 diecisiete críticos de los modelos de mejora del desarrollo de software basados en procesos, convocados por Kent Beck, se reunieron en Snowbird, Utah para tratar sobre técnicas y procesos para desarrollar software.

En la reunión se acuñó el término “Métodos Ágiles” para definir a los métodos que estaban surgiendo como alternativa a las metodologías formales (CMMI, SPICE) a las que consideraban excesivamente “pesadas” y rígidas por su carácter normativo y fuerte dependencia de planificaciones detalladas previas al desarrollo.

Los integrantes de la reunión resumieron los principios sobre los que se basan los métodos alternativos en cuatro postulados, lo que ha quedado denominado como Manifiesto Ágil.



XP: eXtreme Programming

Se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad.

Consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos.
El cliente no sabe lo que quiere hasta que ve lo que no quiere.

Esta metodología lleva las buenas prácticas a niveles extremos. Por ejemplo el TDD (test driven development) fue usado en la NASA en el proyecto mercurio en los 60.

XP: Valores

Comunicación: dentro en un equipo de desarrollo es fundamental para que la información entre los miembros fluya con normalidad, esto ayudará a evitar errores y agilizar las tareas compartidas

Retroalimentación (Feedback): Con el cliente es de gran ayuda entregar un software de calidad que cumpla las expectativas del mismo cliente. Para ello, el desarrollo debe realizarse en ciclos cortos que permitan mantener una retroalimentación constante y continua.

Simplicidad: Cuando se trabaja en el diseño de una historia de usuario, es importante centrarse en esa tarea exclusivamente para mantener un código limpio y sencillo. Esto ayuda a evitar el over-engineering, es decir, no se debe preparar software "por si acaso"

Coraje ser valiente para evitar over-engineering, para desechar un código fuente antiguo inservible, para refactorizar código que funcione o para sobreponerse de problemas que lleva tiempo sin resolverse.

XP: Prácticas

Planificación:
- Historias de usuario
- Plan de entregas
- Plan de Iteraciones
- Reuniones diarias

Diseño
- Simplicidad
- Soluciones
- Refactorización
- Metáforas

Desarrollo de código
- Disponibilidad del cliente
- Uso de estándares
- Programación dirigidas por pruebas
- Programación por pares.
- Integración continua.
- Propiedad colectiva del código
- Ritmo sostenido.

Pruebas
- Pruebas unitarias
- Detección y corrección de errores.
- Pruebas de aceptación.

SCRUM

Scrum es un marco de trabajo que define un conjunto de prácticas y roles, y que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto.

Los roles principales en Scrum son el **Scrum Master**, *que procura facilitar la aplicación de scrum y gestionar cambios, el* **Product Owner**, *que representa a los clientes/usuarios y el* **Team *Developer*** *(equipo de desarrollo) que realiza el desarrollo y demás elementos relacionados con él.*

Durante cada *sprint*, un periodo entre una y cuatro semanas (la magnitud es definida por el equipo y debe ser lo más corta posible), el equipo crea un incremento de software *potencialmente entregable* (utilizable).

El conjunto de características que forma parte de cada sprint viene del *Product Backlog*, que es un conjunto de requisitos de alto nivel priorizados que definen el trabajo a realizar.

SCRUM

La metodología Scrum se basa en:

- ✓ El desarrollo incremental de los requisitos del proyecto en bloques temporales cortos y fijos.
- ✓ Se da prioridad a lo que tiene más valor para el cliente.
- ✓ El equipo se sincroniza diariamente y se realizan las adaptaciones necesarias.
- ✓ Tras cada iteración (un mes o menos entre cada una) se muestra al cliente el resultado real obtenido, para que este tome las decisiones necesarias en relación a lo observado.
- ✓ Se le da la autoridad necesaria al equipo para poder cumplir los requisitos.
- ✓ Fijar tiempos máximos para lograr objetivos.
- ✓ Equipos pequeños (de 3 a 9 personas cada uno).

KANBAN

La palabra Kanban viene del japonés y traducida literalmente quiere decir *tarjeta con signos* o *señal visual*. El tablero más básico de Kanban está compuesto por tres columnas: “Por hacer”, “En proceso” y “Hecho”. Si se aplica bien y funciona correctamente, serviría como una fuente de información, ya que demuestra dónde están los cuellos de botella en el proceso y qué es lo que impide que el flujo de trabajo sea continuo e ininterrumpido.

Las tarjetas Kanban visualizan las tareas de trabajo en un flujo de trabajo y le ayudan a:

- ✓ Registrar documentación clave de una tarea
- ✓ Revisar los detalles de un vistazo
- ✓ Minimizar la pérdida de tiempo
- ✓ Identificar oportunidades para colaboración

Herramientas de apoyo al desarrollo de software

Las **herramientas CASE** (**C**omputer **A**ided **S**oftware **E**ngineering) son diversas aplicaciones informáticas destinadas a aumentar la productividad y calidad en el desarrollo de software.

- Objetivos:
  - **Incrementar**
    - Productividad del equipo.
    - Calidad del software
    - Reusabilidad del software
  - **Reducir**
    - Costes de desarrollo y mantenimiento
  - **Amortizar**
    - Gestión del proyecto
    - Desarrollo del software
    - Mantenimiento del software.

Herramientas CASE

Clasificación según la fase del ciclo de vida que abordan:

- CASE frontales (front-end) o **Upper CASE**: Herramientas de apoyo a las primeras fases:
  - Planificación
  - Análisis de requisitos.
- CASE intermedias o **Middle CASE**: Herramientas de apoyo en las fases intermedias:
  - Análisis
  - Diseño.
- CASE dorsales (back-end) o **Lower CASE**: Herramientas de apoyo a las últimas fases:
  - Implementación (generación de código).
  - Pruebas.
  - Mantenimiento.

Ejemplos de herramientas CASE libres son: ArgoUML, Use Case Maker, ObjectBuilder…

Roles que interactúan en el desarrollo

- **Diseñador del software**: Nace como una evolución del analista y realiza, en función del análisis, el diseño de la solución que hay que desarrollar. Participa en la etapa de diseño.
- **Analista programador**: Comúnmente llamado “desarrollador” domina una visión más amplia de la programación, aporta una visión general del proyecto más detallado, diseñando una solución más amigable para la codificación y participando activamente en ella. Participa en las etapas de diseño y codificación**.**
- **Programador**: Se encarga de manera exclusiva de crear el resultado del diseño realizado por analistas y diseñadores. Escribe el código fuente del software. Participa en la etapa de codificación.
- **Arquitecto del software**: Es la argamasa que cohesiona el proceso de desarrollo. Conoce e investiga:
  - **Frameworks**: Estructura de soporte definida con módulos software concreto (soporte de programas, bibliotecas, lenguajes de scripting) que puede servir de base para la organización, desarrollo y unión de diferentes componentes del software.
  - Tecnologías

  Revisa que todo el procedimiento de desarrollo software se lleva a cabo de la mejor forma posible y con los recursos más apropiados. Participa en las etapas de análisis, diseño, documentación y explotación.