

# Ciclos de Vida

---



# Material bibliográfico



- “Ingeniería del software. Un enfoque práctico”. Roger S. Pressman. 7ª edición. McGraw-Hill.
- “Software engineering”. Ian Sommerville. 9ª edición. Addison-Wesley.
- “Metrics and models in software quality engineering”. Stephen H. Kan. Addison-Wesley.
- “Ingeniería del software. Aspectos de gestión. Tomo 1: Conceptos básicos, teoría, ejercicios y herramientas”. Román López-Cortijo y García y Antonio de Amescua Seco. Instituto Ibérico de la Industria del Software ([www.iiis.es](http://www.iiis.es)).

# Índice



- Introducción.
- Consideraciones básicas.
- Modelos tradicionales.
- Modelos evolutivos.



# Introducción

# Proyectos software (I)



- Se puede definir un proyecto software como una actividad humana que transforma unos requisitos de un cliente/usuario en un producto software entregable.
- Para dicha transformación se aplica un proceso de desarrollo o ciclo de desarrollo establecido.



# Proyectos software (II)



- Normalmente:
  - El proyecto software arranca con los requisitos del software.
- Sin embargo:
  - Hay proyectos software que no se ajustan a lo anterior:
    - Proyectos de mantenimiento.
- En general:
  - Todos los tipos de proyectos deben seguir un ciclo de desarrollo para llevarlos a cabo.
- Pero:
  - No cualquier proceso es un proyecto.
  - Un proyecto es una secuencia bien definida de eventos, con un principio y un fin temporales, dirigidos a alcanzar unos objetivos (que no son rutinarios) claros y con relaciones entre sí.
    - Una tarea repetitiva mensual, ¿se considera un proyecto?
  - Los proyectos, por tanto, se caracterizan por los siguientes atributos:
    - Objetivos tangibles o productos a entregar previstos.
    - Recursos previstos.
    - Plazo previsto.
    - Costes previstos.
    - Nivel de calidad previsto.
    - Riesgo previsto.
- Los ciclos de desarrollo ayudan a abordar estos atributos de los proyectos.

# Proyectos software (III)



- Los proyectos software son proyectos normales.
  - Por lo tanto, la mayoría de las técnicas de gestión de proyectos son aplicables a la gestión de proyectos software.
- Sin embargo, los proyectos software tienen cuatro características especiales según R. López-Cortijo y A. de Amescua:
  - Invisibilidad.
  - Complejidad.
  - Flexibilidad.
  - Juventud de la ingeniería.
- A continuación se verán estas características y cómo un ciclo de vida/desarrollo permite considerarlas correctamente.

# Características (I)



- Invisibilidad:
  - El hardware es el producto físico (tangible) de la informática.
  - El software se define como un producto lógico, no tangible.
  - Es la primera diferencia con otras ingenierías:
    - El producto software es mucho menos tangible que otros productos o sistemas de ingeniería.
    - El grado de avance en la construcción de un puente es tangible, pero ¿y el grado de avance en el desarrollo de un sistema software?
  - Un ejemplo: ¿Cómo se mide el software?
    - Las medidas o métricas aplicables a otro tipo de productos de ingeniería están siendo normalizadas para los proyectos software:
      - Lines Of Code (LOC).
      - Puntos de Función.
  - La invisibilidad también afecta a la dificultad de estimación.
    - ¿Qué tamaño va a tener este producto software?
    - ¿Cuánto va a llevar este proyecto software?
      - ¿En esfuerzo?
      - ¿En tiempo?
      - ¿En coste?



# Características (II)



- Complejidad:
  - El software posee mayor complejidad que cualquier otro producto de ingeniería.
  - Esto se pone de manifiesto cuando se trata de probar el software:
    - Una prueba al 100% de un sistema software es inviable.
  - Cualquier otro sistema de ingeniería siempre se prueba al 100% al final o en cada fase. Por ejemplo, para probar un puente:
    - Previamente se define el proyecto.
    - Hay un cálculo de estructuras por parte de un ingeniero que se revisa.
    - Se construye, y luego ...
    - Se prueba: se sitúan camiones con arena mojada para ver si el puente resiste.
    - Esta forma de actuar en las pruebas no es viable para el software.
  - El software es el único producto de ingeniería en donde las pruebas no buscan verificar que está libre de errores, sino que el objetivo es encontrar errores.

# Características (III)



- Flexibilidad:
  - Se cree que:
    - El software es fácil de cambiar:
      - Sólo es tocar código.
    - Los requisitos del software se pueden cambiar sin problemas debido a lo anterior.
  - Sin embargo, en otras ingenierías esto no es así:
    - Una vez que hay planos, ¿las especificaciones de un edificio cambian?
    - Después de acabada su construcción, ¿se toma a la ligera una remodelación?
    - ¿Alguien podría pensar en el siguiente cambio sin más? Si se hace, ¿a qué coste?



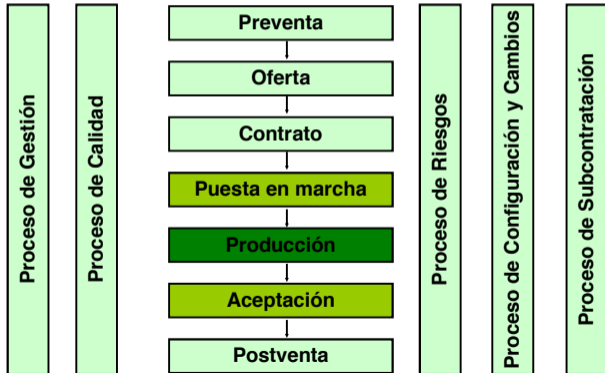
- Según Gartner Group, a finales de los años 90 el 95% de todos los recursos en una empresa de software estaban dedicados a mantenimiento:
  - Cuello de botella: sólo el 5% se pueden dedicar a nuevos desarrollos.

# Características (IV)



- Juventud:
  - Se cree que la ingeniería del software, al ser una ingeniería “reciente”, carece de:
    - Procesos y estrategias de desarrollo variadas, probadas y definidas.
  - Sin embargo, por ejemplo, existen muchas maneras (ciclos de desarrollo) de producir software.
  - En este tema abordaremos los siguientes:
    - Modelos tradicionales:
      - Cascada.
      - V.
      - Prototipado.
      - DRA.
    - Modelos evolutivos:
      - Incremental.
      - Espiral.
  - En definitiva, el desarrollo de software no debe ser algo caótico o ad-hoc para cada proyecto, sino algo claramente definido, implantado y aplicado.

# Contexto de un proyecto software



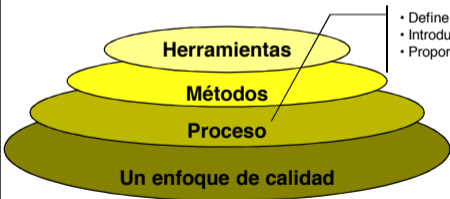


# Consideraciones básicas

# Necesidad



- Según Pressman, para resolver los problemas reales del software se debe incorporar una estrategia de desarrollo que defina el proceso, acompañándolo de métodos y herramientas, sobre una base de calidad:



- Define actividades para desarrollar un proyecto.
- Introduce consistencia entre proyectos.
- Proporciona checkpoints para seguimiento.

**Capas de la ingeniería del software según Pressman**

# Terminología

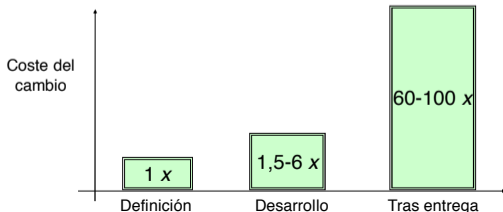


- En español:
  - Ciclo de desarrollo, modelo de proceso, modelo de producción, ...
- En inglés:
  - Development cycle, process model, development model, ...
- Un ciclo de desarrollo se emplea para estructurar las actividades que se llevan a cabo en un proyecto software.
  - Es una formalización del proceso de desarrollo.
  - Se inicia con la decisión de desarrollar y finaliza al entregar el producto.
  - Incluye, en general:
    - Fase de establecimiento de requisitos.
    - Fase de diseño del producto.
    - Fase de implementación del producto.
    - Fase de prueba del producto y, a veces,
    - Fase de instalación, aceptación y mantenimiento (ciclo de vida  $\supset$  ciclo de desarrollo).
- Es uno de los aspectos estratégicos determinantes para el éxito de un proyecto software.
  - Su elección adecuada puede evitar fracasar el proyecto o que haya más costes de los inicialmente previstos. Pueden combinarse ciclos de desarrollo en un proyecto.

# Verificación y Validación



- En los modelos más avanzados destaca la consideración de actividades de V&V:
  - Inspecciones.
  - Revisiones.
  - Walktroughs.
  - Pruebas unitarias, de integración, de sistema y de aceptación.
- Motivo:
  - Detección de defectos lo antes posible para evitar costes, que según Pressman obedecen a la siguiente gráfica:





# Los inicios ...



- La crisis del software fue provocada principalmente por el modelo de desarrollo de entonces:
  - Ciclo de vida Code and Fix (en español, codificar y arreglar) o de Codificación Directa.
- Supone programar “sobre la marcha”:
  - No se planifica, no se analizan requisitos, no se diseña, no se documenta suficientemente, ...
    - Se toman unas pocas notas, se programa, se prueba, se detectan fallos y vuelta a empezar.
- Es sinónimo de arte y antónimo de ingeniería.
  - Tiende a ser el más usado en organizaciones inmaduras y el que acarrea más costes en todos los aspectos (calidad, económicos, de plazos, ...).
- Empleado frecuentemente en:
  - Proyectos muy pequeños (más programas independientes que proyectos).
  - Prototipos para analizar la viabilidad técnica (tampoco hay que “matar moscas a cañonazos”).
- Sin embargo, no es nada recomendable para proyectos algo más grandes que “enanos”.

# Actualmente ...



- En cualquier ciclo de vida actual se identifican tres macro-fases genéricas:
    - Definición (pensar): se centra en el **qué**.
      - Se intenta identificar **qué** información ha de ser procesada, **qué** función y rendimiento se desea, **qué** interfaces van a ser establecidas, **qué** restricciones de diseño existen y **qué** criterios de validación se necesitan para definir un sistema correcto.
      - Los métodos a aplicar variarán dependiendo del paradigma elegido, pero hay tres tareas principales: ingeniería de sistemas o de información, planificación del proyecto y análisis.
    - Desarrollo (hacer): se centra en el **cómo**.
      - Se intenta definir **cómo** han de diseñarse las estructuras de datos y la arquitectura del software, **cómo** han de implementarse los detalles procedimentales, **cómo** ha de traducirse el diseño a un lenguaje de programación y **cómo** ha de realizarse la prueba.
      - Los métodos aplicados variarán, pero hay tres tareas principales: diseño del software, generación del código y prueba del software.
    - Mantenimiento (mantener): se centra en el **cambio**.
      - Está asociada a la **corrección** de errores, a las **adaptaciones** requeridas por evolución del entorno del software y a los **cambios** debidos a las mejoras producidas por los requisitos del cliente.
      - Esta fase vuelve a aplicar los pasos de las anteriores, pero en el contexto de un software ya existente.
-



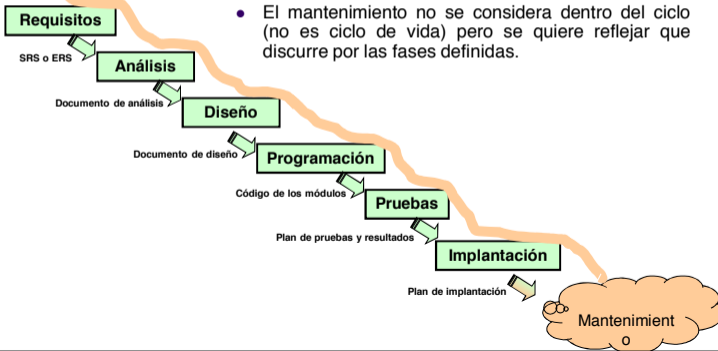
# Modelos tradicionales

# Cascada (I)



- También conocido como Waterfall (cascada de agua) en inglés.
- Establecido durante los años 70 como alternativa al Code and Fix, que provocó la crisis del software.
- Es el que sigue un gran número de metodologías y es el más conocido, estudiado, difundido y empleado.
- Pone especial énfasis en la realización temprana de actividades de definición de requisitos y de documentación de análisis y diseño como paso previo a la codificación.
- Es una encadenación lineal y en secuencia de las siguientes actividades generales:
  - Analizar.
  - Diseñar.
  - Codificar.
  - Probar.
  - Implantar.
- Cada fase genera productos de salida que servirán a la siguiente fase para desarrollar la actividad de la misma como productos de entrada.

# Cascada (II)

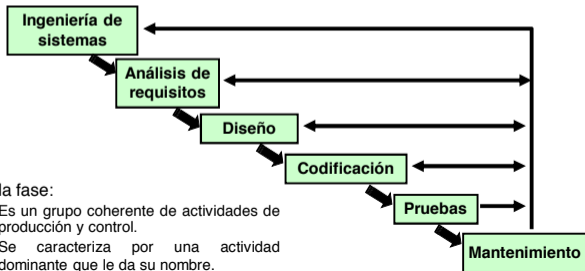


- Se aborda un paso **completamente** antes de empezar el siguiente.
- El mantenimiento no se considera dentro del ciclo (no es ciclo de vida) pero se quiere reflejar que discurre por las fases definidas.

# Cascada (III)



- El modelo original, propuesto por Winston Royce en 1970, preveía bucles de realimentación.
- La gran mayoría de las organizaciones que lo aplican, sin embargo, lo hacen como si fuese estrictamente lineal.

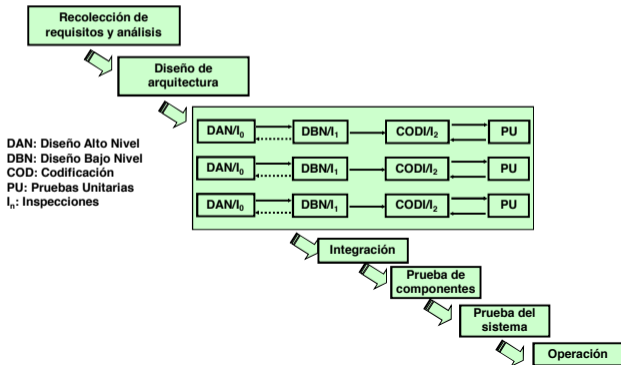


- Cada fase:
  - Es un grupo coherente de actividades de producción y control.
  - Se caracteriza por una actividad dominante que le da su nombre.

# Cascada (IV)



- Debido a su profuso uso, han ido surgiendo “personalizaciones” como por ejemplo puede ser:



# Cascada (V)



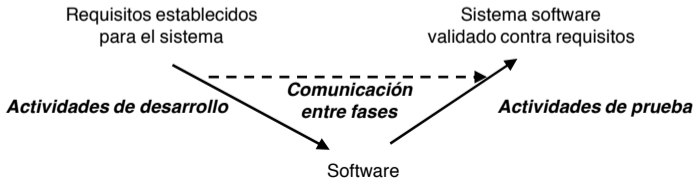
- Ventajas:
  - Conformar un marco de referencia para asignar todas las actividades de desarrollo software.
  - Es un método muy estructurado que establece pautas de trabajo muy claras.
  - Facilita mucho la coordinación de los recursos implicados.
  - Facilita la disposición de hitos de seguimiento/control en el desarrollo de proyectos.
  - Facilita la estimación y el seguimiento/control del progreso de las actividades.
  - Facilita la detección de desviaciones y la realización de acciones correctivas.
  - Proporciona productos entregables intermedios que conforman el producto final.
- Inconvenientes:
  - Comienza estableciendo **todos** los requisitos del sistema a desarrollar:
    - Muchas veces esto no es viable:
      - Puede ser difícil para el propio usuario.
      - Hay cambios de parecer de los usuarios, habiendo finalizado ya la fase de requisitos.
  - Posee una gran rigidez: cada actividad es prerrequisito de las que le siguen.
  - Su aplicación en el mundo real no contempla la “vuelta atrás”: es utópicamente lineal.
  - No soporta prácticas modernas de desarrollo (e.g., Prototipado).
  - Los posibles problemas se detectan tarde aunque se incluyan actividades de V&V.
  - Los únicos productos (parciales) aprovechables están en forma de documentos:
    - “Nada está hecho hasta que todo está hecho”.
    - Un cambio debido a un error puede suponer un gran coste.



# V (I)



- Es un modelo similar al de Cascada, aunque “mejorado”.
- Tiene dos tiempos (tipos de actividades) claramente marcados:
  - Rama descendente: actividades de desarrollo.
  - Rama ascendente: actividades de prueba.
  - Ambas ramas se comunican en materia de pruebas:
    - Cada fase de la rama descendente tiene su homóloga en la rama ascendente para las actividades de prueba correspondientes.
- El esquema básico es el siguiente:



# V (II)

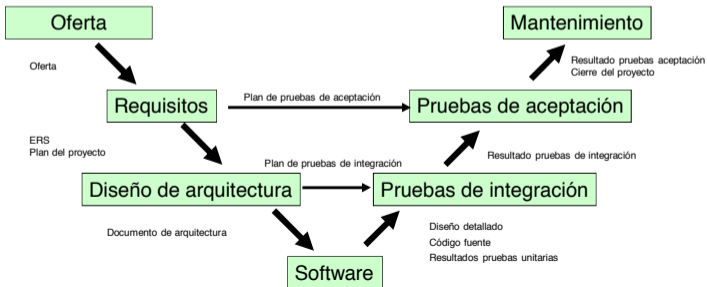


- Cada fase de la rama descendente (rama de desarrollo):
  - Parte de las salidas de la fase previa y las verifica.
  - Aborda las actividades propias definidas para dicha fase.
  - Prepara las pruebas que van a permitir validar lo que se ha obtenido.
    - Estas pruebas se preparan aquí pero se ejecutarán en su fase homóloga situada a su mismo nivel de la rama ascendente.
  - Sirve de punto de partida a las actividades de la fase siguiente a través de los productos obtenidos.
- Cada fase de la rama ascendente (rama de pruebas):
  - Ejecuta las pruebas (ya preparadas) para validar lo definido durante la fase homóloga de su mismo nivel de la rama descendente.
  - Verifica las actividades llevadas a cabo durante la fase.
  - Sirve de punto de partida a las actividades de la fase siguiente.

# V (III)



- Un ejemplo muy simplificado podría ser el siguiente:



# V (IV)



- **Ventajas:**
  - Al ser una variante del de Cascada, hereda todas sus ventajas.
  - Conformar un marco de referencia para asignar todas las actividades de desarrollo software, incluyendo las actividades V&V de lo que se hace en las sucesivas etapas.
  - Favorece la consideración de las pruebas lo antes posible (comunicación horizontal entre las dos ramas de la V).
- **Inconvenientes:**
  - Al igual que en Cascada, se comienza estableciendo **todos** los requisitos del sistema a desarrollar:
    - Muchas veces esto no es viable:
      - Puede ser difícil para el propio usuario.
      - Hay cambios de parecer de los usuarios, habiendo finalizado ya la fase de requisitos.
    - No soporta prácticas modernas de desarrollo (e.g., Prototipado).
    - Posee una gran rigidez, aunque menos que en el de Cascada por haber comunicación horizontal.
    - Su aplicación en el mundo real no contempla la “vuelta atrás”: es utópicamente lineal.
    - Los únicos productos (parciales) aprovechables están en forma de documentos:
      - “Nada está hecho hasta que todo está hecho”.
      - Un cambio debido a un error puede suponer un gran coste.

# Prototipado (I)

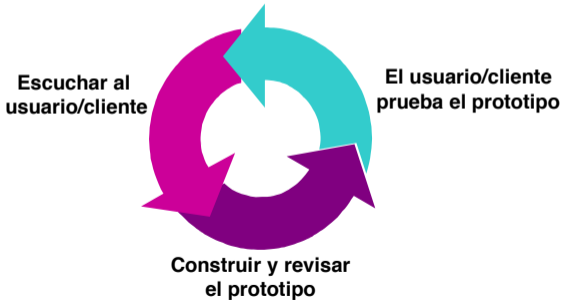


- Su sentido es análogo al que tiene en otras ingenierías:
  - El prototipado o construcción de un prototipo es un proceso que facilita al desarrollador la creación de un modelo del software a desarrollar.
- Un prototipo puede ser de muchos tipos:
  - En papel, describiendo por ejemplo la interacción hombre-máquina.
  - Funcional, implementando un subconjunto de las funciones requeridas.
  - Un programa ya existente, que ejecuta toda o parte de la funcionalidad deseada pero que tiene características que deben ser mejoradas o abordadas en el nuevo desarrollo.
- Es muy adecuado cuando no se sabe exactamente lo que se quiere construir. Algunas razones para no saberlo son:
  - El propio cliente/usuario no sabe exactamente lo que quiere.
  - El desarrollador no está seguro de la eficiencia de una aproximación.
  - Existen dudas en la interfaz hombre-máquina.
  - ...

# Prototipado (II)



- El proceso básico del prototipado involucra las siguientes tres fases que se detallarán a continuación:



# Prototipado (III)



- Escuchar al usuario/cliente:
  - Se corresponde con la recolección de requisitos.
  - Se definen los objetivos globales del sistema.
  - Se identifican los requisitos conocidos y las áreas donde se precisa una mayor definición.
- Construir y revisar el prototipo:
  - Se realiza un desarrollo rápido centrado exclusivamente en los aspectos del software visibles para el usuario/cliente:
    - Formatos de entrada.
    - Formatos de salida.
    - Pantallas.
    - Etc.
  - Este desarrollo rápido conforma el prototipo elaborado en este ciclo.
  - En esta fase resulta muy beneficioso emplear herramientas de 4ª generación (4GL).  
Por ejemplo:
    - Delphi.
    - Cristal Report para los informes.
- El usuario/cliente prueba el prototipo:
  - El prototipo es evaluado por el usuario/cliente.
  - Esta evaluación permite refinar los requisitos del software.
  - Este refinamiento es la entrada al siguiente ciclo del Prototipado.

# Prototipado (IV)



- La pregunta surge cuando se ha finalizado el prototipo:
  - ¿Qué se hace con él?
- En la mayoría de los casos, el prototipo será:
  - Demasiado lento.
  - Demasiado grande.
  - Demasiado torpe en su operativa.
  - En definitiva y según Pressman, está hecho con chicles y alambres.
- Por lo tanto, idealmente, se debe de desechar el prototipo y empezar de nuevo.
  - El prototipo sólo debe servir para identificar los requisitos del software cuando no estén claros.
- Con los 4GL o herramientas similares, las pantallas y la dinámica de pantallas se podría aprovechar; pero siempre adecuándolos a los estándares y normas seguidos en un producto no prototipo.
- Después de un Prototipado, típicamente se suele continuar con un ciclo de vida clásico: combinación de ciclos de vida.



# Prototipado (V)



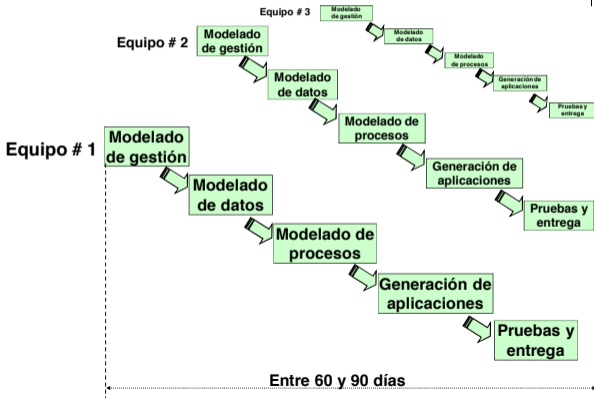
- Ventajas:
  - El prototipo es un mecanismo ideal para extraer requisitos cuando no están claros, incluso por parte del usuario.
    - “No sé lo que quiero hasta que veo lo que no quiero”.
- Inconvenientes:
  - Existe una clara tendencia del usuario/cliente a creer que el trabajo ya está hecho y que en breve dispondrá de un sistema funcional.
    - En realidad, se está simplemente en la primera fase.
  - El desarrollador toma necesariamente decisiones de implementación simplemente porque le son conocidas y le permiten desarrollar rápidamente el prototipo.
    - No son decisiones adecuadas para el sistema real.
    - El problema radica en que estas decisiones a menudo se mantienen en el sistema real.

# DRA (I)



- Son las siglas correspondientes a Desarrollo Rápido de Aplicaciones.
  - RAD (Rapid Application Development) en inglés.
- Es una “adaptación a alta velocidad” del modelo en Cascada.
  - Por tanto, es un modelo lineal y secuencial que establece un ciclo de desarrollo extremadamente corto.
- Se logra dicho desarrollo rápido empleando una filosofía de construcción basada en componentes (reutilización).
- Se aconseja su empleo si los requisitos están claros y se limita el ámbito del proyecto que se va a acometer.
  - Sino, mejor seguir inicialmente un ciclo de vida de Prototipado.
- Permite crear un sistema en poco tiempo (entre 60 y 90 días).
- Un esquema de este modelo es el siguiente:

# DRA (II)



# DRA (III)



- **Modelado de gestión:**
  - El flujo de información entre las funciones de gestión se modela de forma que se responda a las siguientes preguntas básicas:
    - ¿Qué información conduce el proceso de gestión?
    - ¿Qué información se genera?
    - ¿Quién la genera?
    - ¿A dónde va la información?
    - ¿Quién la procesa?
- **Modelado de datos:**
  - El flujo de información establecido en la fase anterior se detalla en la forma de un conjunto de objetos de datos:
    - Se definen los atributos de cada objeto y las relaciones entre ellos.
- **Modelado de procesos:**
  - Los objetos de datos detallados en la fase anterior sufren una transformación para conseguir el flujo de información necesario para implementar una función de gestión.
    - Las descripciones de los procesos de transformación se crean para manipular los objetos de datos detallados.

# DRA (IV)



- Generación de aplicaciones:
  - Se asume el manejo de herramientas de 4ª generación (4GL).
  - En vez de manejar 3GL, el modelo DRA prioriza la reutilización basada en componentes. Así, establece que:
    - Se vuelvan a utilizar componentes de programas ya existentes cuando sea posible.
    - Se creen componentes reutilizables cuando sea necesario.
  - Siempre se deben emplear herramientas CASE para facilitar la construcción del software.
  
- Pruebas y entrega:
  - Al priorizar la reutilización, muchos de los componentes empleados en el desarrollo actual ya están probados.
    - Se reduce por tanto el tiempo de pruebas, pero:
      - Es necesario probar todos los nuevos componentes que se han desarrollado ex-proceso.
      - Es necesario ejercitar detalladamente todas las interfaces entre componentes.

# DRA (V)



- Restricciones que el propio modelo impone:
  - Las limitaciones temporales impuestas demandan un ámbito de escalas.
    - Un sistema es candidato a desarrollarse por medio del modelo DRA si se puede articular de forma que cada función de gestión principal se pueda completar en menos de 3 meses con una aproximación DRA. En este caso:
      - Cada una de las funciones de gestión principales se asigna a un equipo DRA diferente.
      - Los desarrollos de los diferentes equipos se integrarán en las actividades finales.

# DRA (VI)



- Ventajas:
  - Inherentemente introduce la reutilización. Por tanto, al menos inicialmente, existe:
    - Mayor productividad en el desarrollo.
    - Menor probabilidad de errores.
  - Inherentemente también introduce el paralelismo (varios equipos en paralelo).
  - Ideal para considerarlo conjuntamente con la tecnología OO (objetos como componentes).
- Inconvenientes:
  - En el caso de proyectos grandes, el DRA demanda que haya recursos humanos suficientes en la organización para crear los distintos equipos DRA correctos.
  - Es imprescindible el compromiso de clientes/usuarios y de desarrolladores para con la necesaria celeridad en las actividades del modelo.
  - Si un sistema no se puede segmentar en módulos de forma sencilla, construir los componentes será complicado por los “solapes”.
  - Si los riesgos técnicos son altos (e.g., nuevas tecnologías y sistema complejo), este modelo no es la mejor opción.



# Modelos evolutivos



# Modelos evolutivos (I)



- El software, como todos los sistemas complejos, necesariamente evoluciona con el tiempo.
  - Los requisitos de gestión, del producto, el propio mercado, etc. a menudo hacen que sea imposible finalizar un producto completamente.
  - Esto conlleva a introducir una versión limitada para cumplir la presión competitiva y de gestión:
    - Se comprende perfectamente el conjunto de requisitos principales, pero todavía se tienen que definir los detalles de extensiones del producto.
- En éstas y en otras situaciones semejantes, se necesita un modelo de proceso que se haya diseñado explícitamente para acomodarse a un producto que evolucione con el tiempo.
- Sin embargo:
  - El enfoque en Cascada asume que al final de la secuencia marcada se entrega un sistema completo.
  - El enfoque de Prototipado ayuda a comprender los requisitos para luego acometer el desarrollo propiamente dicho.

# Modelos evolutivos (II)



- En general, en la realidad, no se desarrolla para entregar un sistema listo para pasar a producción de forma completa.
- Sin embargo, los paradigmas vistos anteriormente, no tienen en cuenta la naturaleza evolutiva del software.
- Los modelos evolutivos son interactivos.
  - Se caracterizan por permitir desarrollar versiones cada vez más completas del software.
- Ejemplos de modelos evolutivos, que se verán a continuación:
  - Modelo Incremental.
  - Modelo en Espiral.

# Incremental (I)

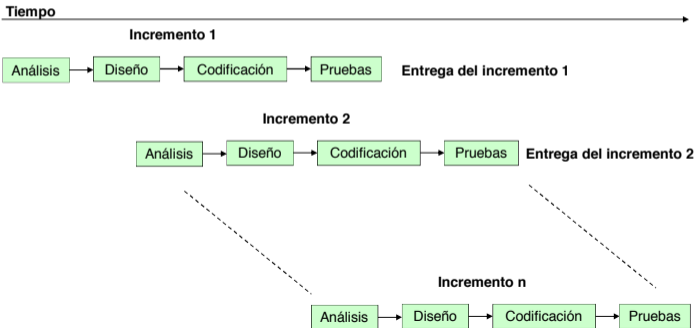


- Es el proceso de construir una implementación parcial del sistema global y posteriormente ir aumentando la funcionalidad del sistema.
- Es la aplicación reiterada de varias secuencias basadas en el modelo en Cascada.
  - Cada aplicación del ciclo constituye un incremento del software.
  - Cada incremento resulta en un producto operativo.
  - Cada incremento puede ser:
    - El refinamiento de un incremento anterior (siguiendo la filosofía del Prototipado).
    - El desarrollo de una nueva funcionalidad no incorporada en incrementos anteriores.
  - En el primer incremento de un sistema se debe abordar el núcleo esencial del sistema (requisitos fundamentales).
  - En incrementos posteriores se abordarán funciones suplementarias (algunas ya conocidas y otras no).
- El usuario/cliente evalúa el resultado de un incremento y se elabora un plan para el incremento siguiente.
- El proceso se repite hasta que se elabore el producto completo.
- Al seguir un desarrollo incremental, el software debe construirse de tal forma que facilite la incorporación de nuevos requisitos.

# Incremental (II)



- Esquema básico:



# Incremental (III)



- Ventajas:
  - La dotación de personal no tiene que estar disponible para una implementación completa.
  - Permite la obtención de incrementos operativos a lo largo del proceso de desarrollo, frente a modelos tradicionales basados en el de Cascada.
    - Esto es, reduce el gasto que se tiene antes de alcanzar cierta capacidad inicial.
  - Reduce la posibilidad de que los requisitos del usuario/cliente cambien durante el desarrollo.
  - Mayor flexibilidad ante más funcionalidades.
- Inconvenientes:
  - Puede ser muy complejo definir el núcleo operativo para lograr el primer incremento:
    - Un software con capacidades suficientes como para ser operativo y que facilite la incorporación de nuevos requisitos.
  - Puede resultar complicado establecer y priorizar las funcionalidades que se mejoran o incorporan a los sucesivos incrementos.
  - Las soluciones de incrementos anteriores pueden no ser válidas para incrementos posteriores.

# Espiral (I)



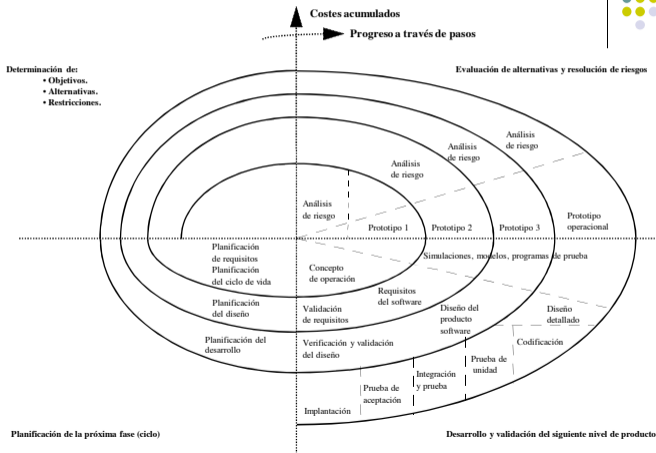
- Es relativamente reciente (1988) y fue propuesto por Barry W. Boehm.
- Es un modelo de proceso evolutivo que:
  - Permite combinar la naturaleza interactiva de construcción de prototipos con los aspectos controlados y sistemáticos del modelo en Cascada.
  - Además, pone especial énfasis en el análisis de los riesgos para reducir su impacto.
- El software se desarrolla en una serie de versiones incrementales.
  - Durante las primeras iteraciones, la versión incremental será un modelo en papel o un prototipo.
  - Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema.
- La idea básica es que cada fase (vuelta de la espiral) establece los siguientes pasos:
  - Determinación de objetivos, alternativas y restricciones.
  - Evaluación de alternativas (considerando análisis de riesgos).
  - Desarrollo del siguiente nivel de producto.
  - Planificación de la siguiente fase (ciclo en la espiral).

# Espiral (II)



- Cada ciclo (vuelta) en la espiral representa una fase del proceso de desarrollo.
  - Por ello, el ciclo más interno se corresponde con la viabilidad del sistema.
  - El siguiente, con el análisis de requisitos y así sucesivamente.
- No hay fases fijas en el modelo. El modelo se adapta a las necesidades que surjan en cada proyecto.
- Los bloques de diseño, codificación y pruebas se ejecutan de forma secuencial para la entrega de unas características iniciales operativas.
- Después de ello se vuelve a revisar el producto y cómo mejorar/ampliar sus características operativas. El producto se actualiza, obteniendo un prototipo operativo “puesto al día” que sirve para demostración y validación.
- El sistema pasa por un proceso actualizado de desarrollo en cascada que finalmente obtiene una nueva versión del producto.

# Espiral (III)





# Espiral (IV)



- Ventajas:
  - Permite adaptar el proceso de desarrollo a las necesidades cambiantes del proyecto y al conocimiento que se va adquiriendo.
  - Permite el manejo de prototipos, enlazándolo con el análisis de riesgos.
  - Gestiona explícitamente los riesgos.
    - Permitirá reducirlos antes de que se conviertan en problemas.
- Inconvenientes:
  - Requiere de una considerable habilidad para la consideración del riesgo.
    - Cuenta con esto para el éxito.
  - El modelo es relativamente nuevo y no se ha manejado tanto como los anteriores.