

Ciclo de vida del software

Definición

El proceso que se sigue para construir, entregar y hacer evolucionar el software, desde la concepción de una idea hasta la entrega y el retiro del sistema.

Confiable, predecible y eficiente.

Objetivos

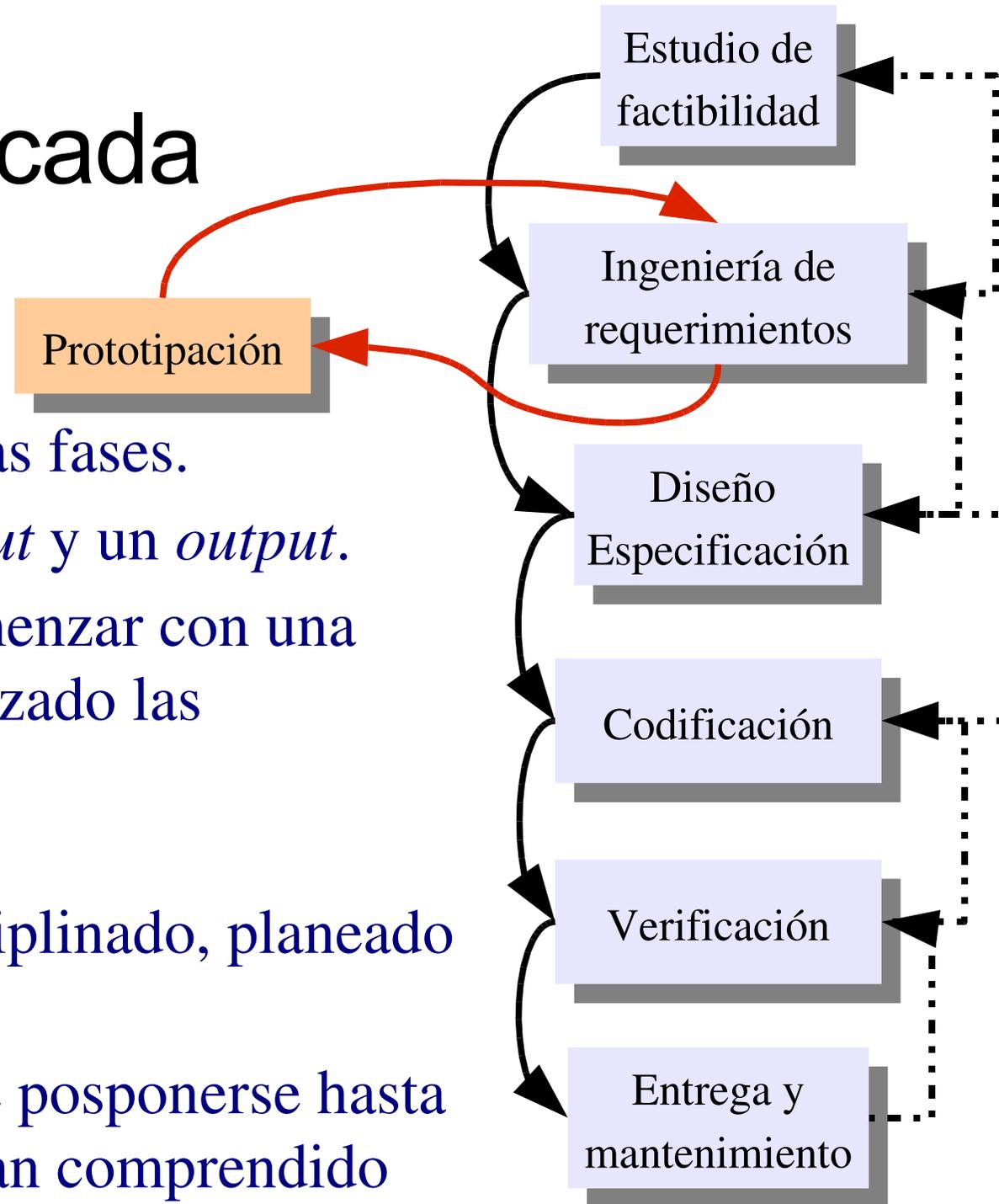
- Bohem: determinar el orden de las etapas involucradas en el desarrollo del soft, establecer el criterio de transición para progresar de una etapa a la siguiente:
 - criterio para determinar la finalización
 - criterio para comenzar y elegir la siguiente.

Así un modelo de proceso apunta a:

¿Qué debemos hacer a continuación?

¿Por cuánto tiempo debemos hacerlo?

Modelo de cascada



- Flujo secuencial entre las fases.
- Cada etapa tiene un *input* y un *output*.
- Se supone que para comenzar con una etapa deben haber finalizado las anteriores.

Contribuciones:

- El proceso debe ser disciplinado, planeado y gerenciado
- La implementación debe posponerse hasta que los objetivos se hayan comprendido

Ingeniería de requerimientos

¿Qué?

Identificar y documentar los requerimientos exactos del sistema según las necesidades de los usuarios finales.

Cualidades del sistema.

Funcionales, no-funcionales, del proceso y del mantenimiento

Estructura

Diseño - Especificación

Función

¿Cómo?

Dividir el sistema en partes y establecer las relaciones entre ellas.

Arquitectura y diseño detallado.

Establecer *qué* hará exactamente cada parte.

En esta fase se crea un modelo funcional-estructural de los requerimientos.

El diseño debe permitir implementaciones que verifiquen los requerimientos.

Verificación

Comprobar que los distintos productos del ciclo de vida del software verifican las propiedades y funciones establecidas en los requerimientos.

Testing vs. análisis

Testing funcional y estructural

Ejecución simbólica

Verificación & Validación

Modelo de transformaciones formales

Ver el desarrollo de software como una secuencia de pasos que transforman un modelo en implementación.

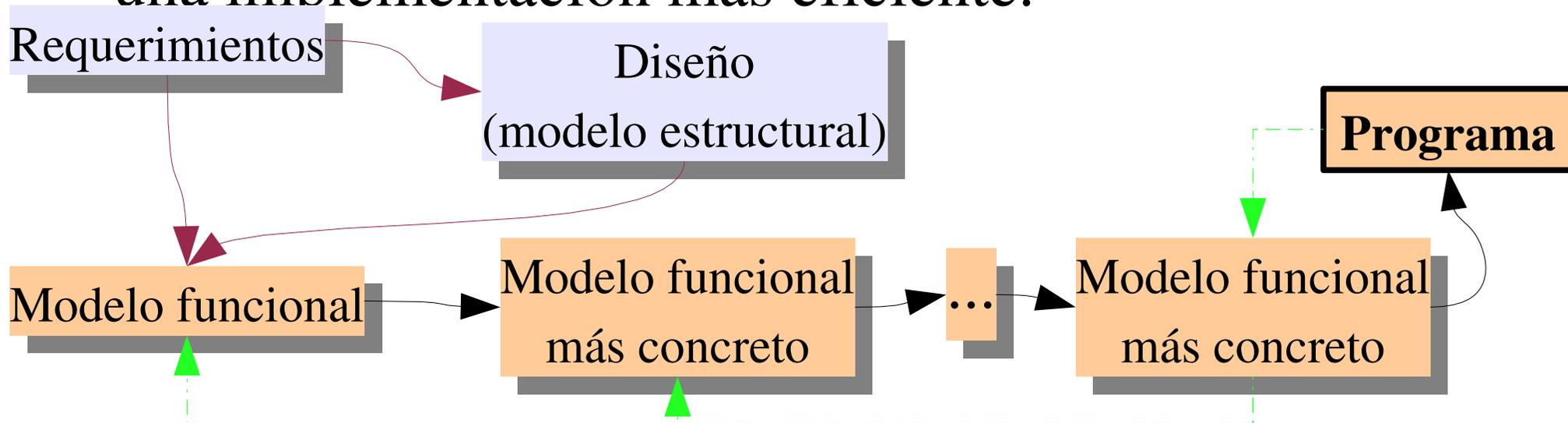
La naturaleza formal de la derivación puede proveer una forma de verificación matemático/lógica que demuestra que un paso es la correcta transformación del anterior.

Construir la prueba de corrección junto con el sistema a partir de un modelo formal.

... transformaciones formales

Cada paso, denominado *refinamiento*, disminuye el nivel de abstracción del modelo hasta llegar a una descripción ejecutable.

También es posible volver hacia atrás para obtener una implementación más eficiente.



The longship's perfect mating of design, structure and material derives neither from a single creative genius nor even a single age. Rather these vessels represent the culmination of 6,000 years of technical **evolution**.
[Tomado de un artículo de *Scientific American*]



Cualidades del software

Atributos de calidad

Las cualidades de un sistema deben estar por encima y por delante de la función del sistema. Lamentablemente, la funcionalidad no sólo ocupa el primer lugar en las prioridades de los desarrolladores sino que muchas veces es el único.

La calidad debe ser considerada en todas las fases del ciclo de vida del software, aunque distintas cualidades se manifiestan de formas diferentes durante el desarrollo.

Clasificación de las cualidades

Externas: son visibles a los usuarios.

Internas: son visibles a los desarrolladores.

Del producto: son observables en los distintos productos y subproductos del ciclo de vida.

Del proceso: describen a la forma en que el producto es producido.

Observables en tiempo de ejecución

No observables en tiempo de ejecución

Corrección

Un programa es funcionalmente correcto si se comporta de acuerdo a la especificación de las funciones que debería proveer.

La corrección no le asegura al usuario que el software se comporte como se espera.

Es una propiedad absoluta: cualquier desviación implica un software no-correcto.

Confiabilidad

Probabilidad de ocurrencia de fallas.

Grado de confianza que el usuario tiene en el software.

Es relativa: un software puede aun ser confiable si la consecuencia de un error no es seria; o si la cantidad de errores por unidad de tiempo no es alta.

Robustez

Un programa es robusto si se comporta *razonablemente* aun en circunstancias que no fueron anticipadas en los requerimientos.

Si se puede pensar en acontecimientos imprevistos, entonces hay que incluirlos en la especificación y se habla de corrección.

Si especificado y verifica P correcto.

Si no especificado y verifica P robusto.

Perfomance

Un sistema es eficiente si usa los recursos económicamente.

Herramientas de medición: complejidad algorítmica, medición, análisis y simulación.

Usualmente es muy difícil mejorar considerablemente la perfomance sin re-diseñar.

Evolucionabilidad

Un software es evolucionable si permite cambios que lo hacen capaz de satisfacer nuevos requerimientos.

Se logra mediante modularización; los sucesivos cambios tienden a destruir un buen diseño (ver entropía)

El diseño original y cada cambio deben hacerse con esta cualidad en mente.

Otras

Verificabilidad: es verificable si sus propiedades pueden verificarse fácilmente.

Reparabilidad: es reparable si permite la corrección de sus defectos con una cantidad limitada de trabajo.

Portabilidad (interoperabilidad): uso (interacción) en (con) diferentes entornos.

Productividad, puntualidad, visibilidad.

Reusabilidad

Amigabilidad