

Modelos de Ciclo de Vida para el Software

Proceso de Producción de Software

- La producción de software incluye diferentes actividades:
 - análisis y especificación de requerimientos,
 - diseño estructural y de datos,
 - pruebas,
 - instalación,
 - otros.
- El orden de ejecución de estas y otras actividades determina el ciclo de vida del software.

2

Procesos Productivos

- El proceso de producción de bienes debe ser:
 - confiable, predecible, y eficiente.
- La definición precisa de los procesos de producción hace posible (en algunos casos) la automatización.
- La producción del software puede verse como un proceso productivo. Pero:
 - la producción de software es una actividad intelectual,
 - los requerimientos del software son inestables y por lo tanto el software evoluciona constantemente.

3

Codificar y Corregir

- En un principio...
 - el desarrollo de software era una tarea unipersonal,
 - el problema a resolver era claramente comprendido,
 - el programador era el usuario de la aplicación,
 - solución de ecuaciones, cálculos astronómicos, balística;
 - la aplicación era simple según estándares actuales,
 - el desarrollo implicaba solamente codificación en lenguaje de máquina.



4

Codificar y Corregir

- Modelo Codificar-y-Corregir :
 - codificar parte del software,
 - corregir errores, agregar funcionalidad o nuevos elementos.
- El proceso de desarrollo no es ni planificado ni controlado.
- Después de una serie de cambios:
 - la estructura del código se hace más y más complicada,
 - los cambios siguientes son más y más difíciles,
 - los resultados son menos confiables.
- Estas desventajas no son graves cuando el programa es chico y bien entendido.

5

Más que Codificar y Corregir

- El crecimiento de la capacidad del hardware llevó a querer usar la computación en áreas diversas:
 - los usuarios ya no fueron los programadores,
 - los dominios de aplicación no siempre eran conocidos por los programadores.
- Ahora el software también debe ser:
 - comercializado,
 - instalado en distintos ambientes.
- Y el desarrollo implica otras actividades:
 - entrenar a los usuarios,
 - asistirlos con sus dificultades.

6

Nuevas Necesidades

- Nuevos aspectos a considerar:
 - económicos,
 - de organización,
 - psicológicos.
- La confiabilidad es esencial:
 - sistemas bancarios,
 - control de procesos industriales.
- El desarrollo de software es una actividad de equipo:
 - estructuras de trabajo,
 - prácticas estándar,
 - planificación y control.

7

Carencias de Codificar y Corregir

Carencias

- Gran complejidad del software debe controlarse.
- Cambios estructurales del software son casi imposibles.
- Prever la rotación de personal.

Consecuencia

→ Es necesaria una etapa de diseño antes de la codificación.

8

Más y más Carencias

Carencias

- El software no satisface las necesidades ni las expectativas del cliente.
- La calidad no es adecuada.
- Productos terminados fuera de plazo y presupuesto.
- La maleabilidad del software no es infinita.

Consecuencia

→ Se requiere una etapa de análisis de requerimientos antes del diseño.

9

Cadena de Consecuencias

Codificar y Corregir



Desarrollo Descontrolado



Crisis del Software



Ingeniería de Software

10

Objetivos de un Proceso Estructurado

“...determinar el orden de las etapas del desarrollo y la evolución del software, y establecer los criterios de transición de un estado al siguiente. Éstos incluyen los criterios de terminación del estado actual más los criterios de selección de la etapa siguiente. Por lo tanto un modelo de proceso aborda las siguientes interrogantes:

- ¿Qué es lo próximo a hacer?
- ¿Por cuanto tiempo debemos hacerlo?”

Boehm [1988]

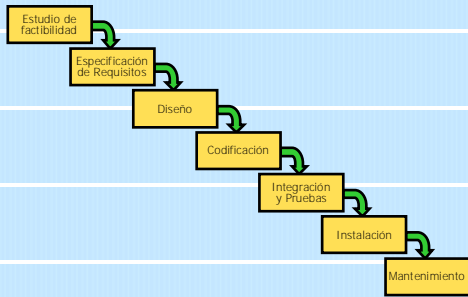
11

Modelo de Cascada

- Se popularizó en la década de los 70 y guía la mayor parte de la práctica actual.
- El proceso es una “cascada” de fases, donde el producto de una fase es la entrada de la siguiente.
- Cada fase se compone de una serie de actividades que deben realizarse en paralelo.
- Existen variantes del modelo básico de cascada, pero todas comparten la misma filosofía.

12

El Modelo de Cascada



13

Aplicación del Modelo de Cascada

- Se sigue una secuencia lineal de etapas.
- Las empresas establecen los productos y documentos que deben producirse en cada etapa.
- Estos productos y/o documentos permiten seguir el avance del proyecto.
- También se establece qué métodos aplicar en cada etapa.
- Estos métodos se organizan en forma coherente en lo que es la "metodología de desarrollo de la empresa".

14

Variaciones al Modelo de Cascada

- Sistemas de áreas conocidas pueden omitir el análisis de factibilidad.
- Sistemas complejos y/o grandes requieren dividir su desarrollo en porciones más pequeñas que puedan desarrollarse en forma más controlada y rigurosa.
- Un sistema con usuarios no especialistas puede requerir una etapa de entrenamiento y preparación de material de apoyo.

15

Tipos de Desarrollos

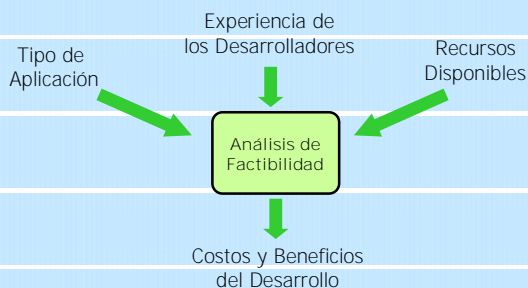
- Una casa de software desarrolla un sistema a medida para un cliente externo.
- El departamento de computación desarrolla sistemas de software para otros departamentos de la misma empresa.
- Una casa de software desarrolla un producto para vender en un mercado horizontal.



El estudio de factibilidad es diferente en cada caso.

16

Estudio de Factibilidad



17

Contenido del Estudio

- Se identifican posibles soluciones al problema planteado.
- Se analizan costos y fechas de entrega.
- Se concluye:
 - si el desarrollo vale la pena,
 - bajo qué condiciones (tiempo y dinero),
 - qué modelo de proceso seguir.
- A veces es demasiado prematuro tomar decisiones a esta altura, pero no hay alternativa.

18

Contenido del Documento de Factibilidad

- Definición del problema.
- Soluciones alternativas.
- Recursos necesarios, costos y plazos para cada alternativa.
- Alternativa elegida y su justificación.

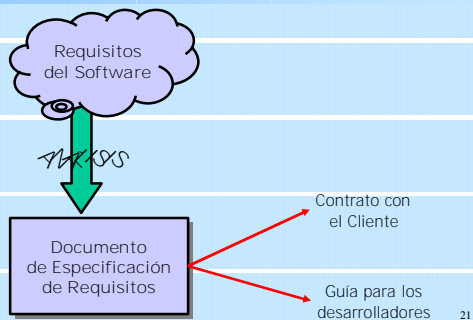
19

Análisis y Especificación de Requisitos

- El propósito de esta etapa es identificar las cualidades del software a desarrollar:
 - funcionalidad,
 - eficiencia,
 - facilidad de uso,
 - portabilidad, etc.
- Se debe especificar *qué* cualidades debe tener el software y no *cómo* lograrlo.
- La especificación de requisitos no debe limitar innecesariamente al desarrollador.

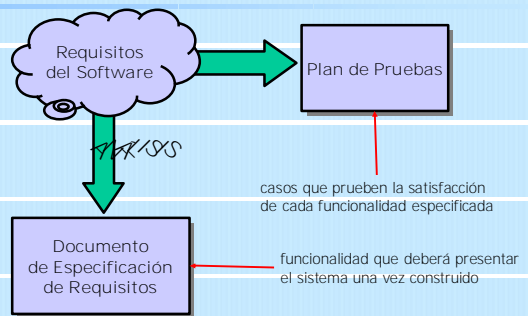
20

Utilidad de la Especificación



21

Otro Producto



22

Cualidades y Principios

- La especificación de requisitos debe tener las siguientes cualidades:
 - comprensible,
 - precisa, completa y consistente,
 - no ambigua.
- Los principales principios a aplicar:
 - separación de intereses
 - distintos puntos de vista,
 - abstracción
 - de lo general a los detalles,
 - modularización
 - datos, funciones y control.

23

Metodología

- Una empresa de desarrollo puede (debería) tener un estándar de métodos y procedimientos para realizar el análisis y especificación de requisitos:
 - modelo de datos
 - diagramas ER, tarjetas CRC;
 - funcionalidad
 - diagramas de flujo de datos, casos de uso;
 - control
 - redes de Petri, máquinas de estados.

24

Contenido de la Especificación de Requisitos

- Requisitos funcionales.
 - ¿Qué hace el producto?
 - Forma en que transforma entradas en salidas.
- Requisitos no funcionales.
 - Confiabilidad (disponibilidad, integridad, seguridad).
 - Precisión de los resultados, eficiencia.
 - Amigable, portable.
- Requisitos sobre el proceso de desarrollo y mantenimiento.
 - Exigencias de procedimientos de control de calidad.
 - Lenguaje de programación.

25

Diseño

- El diseño describe cómo hará el sistema para satisfacer sus requisitos.
- Es la descomposición del sistema en componentes.
- Arquitectura del sistema:
 - ¿qué hacen las componentes?
 - ¿cómo interactúan?
- Las componentes más grandes son divididas iterativamente en sub-componentes:
 - diseño de alto nivel,
 - diseño detallado.

26

Codificación

- Consiste en escribir programas usando un lenguaje de programación.
- El resultado de esta etapa es un conjunto de módulos programados y probados.
- Existen estándares de la empresa desarrolladora:
 - nombres de programas y variables,
 - encabezamientos,
 - número máximo de líneas.

27

Prueba de Módulos

- Las empresas pueden establecer estándares de pruebas:
 - definición de un plan de pruebas,
 - criterios de pruebas (caja negra, caja blanca),
 - criterios de fin de las pruebas,
 - administración de los casos de prueba.
- La depuración ("debugging") es parte de esta etapa.
- Es el control de calidad llevado a cabo en esta etapa.
- Inspecciones para comprobar adhesión a los estándares.

28

Integración y Prueba del Sistema

- Las componentes desarrolladas separadamente se unen para formar el sistema completo.
- La integración puede hacerse:
 - todo junto (big bang),
 - incrementalmente.
- Después de las pruebas del sistema completo, el software se pone en funcionamiento en la organización desarrolladora (test alfa).

29

Instalación y Mantenimiento

- La distribución del software se hace en dos etapas:
 - se instala la aplicación a un grupo selecto de usuarios (test beta) como experimento controlado,
 - se distribuye el software a todos sus usuarios.
- El mantenimiento son todas las actividades que ocurren después que el software está instalado:
 - corregir errores,
 - agregar nueva funcionalidad.

30

Mantenimiento del Software

- El análisis de requisitos es una fuente de problemas, especialmente para los usuarios finales:
 - los requisitos son difíciles de especificar,
 - los requisitos cambian con el tiempo.
- Muchos errores no son resueltos hasta después de instalar el software en el cliente:
 - es más caro corregir errores cuanto más tarde se detectan.
- Los cambios son (casi) siempre posibles pero también (casi) siempre muy difíciles.

31

Otras Actividades

- Paralelamente a las actividades esenciales del modelo de Cascada, existen otras:
 - documentación,
 - verificación,
 - administración.

32

Críticas al Modelo de Cascada

- El modelo de Cascada tiene dos grandes aportes:
 - debe aplicarse disciplina, planificación y administración al proceso de desarrollo de software,
 - la construcción del sistema en sí se pospone hasta que los objetivos del sistema sean suficientemente comprendidos.
- Pero tiene también serias desventajas:
 - lineal,
 - rígido,
 - monolítico.

33

Un Modelo Lineal

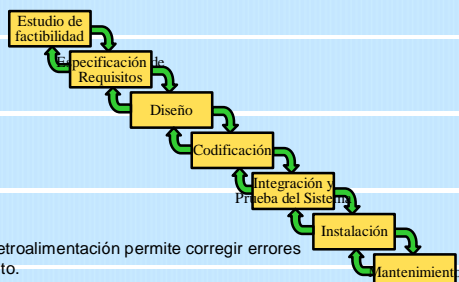
- Un modelo lineal supone que
 - todos los requisitos están disponibles,
 - el diseño elegido es el más apropiado,
 - la programación ocurre sin contratiempos,
 - los errores encontrados son fácilmente corregidos.
- La verdad es que estas cosas raramente suceden:
 - el desarrollo comienza con requisitos incompletos,
 - los errores y contratiempos hacen (casi) volver a empezar.



Se requiere una forma de retroalimentación

34

Cascada con Retroalimentación



- La retroalimentación permite corregir errores pronto.
- Debemos intentar ceñirnos al desarrollo lineal.

35

Un Modelo Rígido

- Cada etapa se termina completamente antes de comenzar la siguiente.
- Así, los requisitos están completos antes del diseño y el cliente no participa más hasta la instalación del sistema completo.
- Si algunas personas terminan su tarea de una etapa antes que los demás, deberán esperar a que todos terminen para comenzar la siguiente etapa.

36

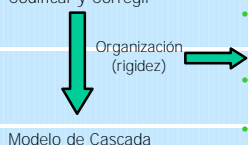
Un Modelo Monolítico

- Se planifica el ciclo de vida de desarrollo para entregar el sistema completo en una fecha dada.
- El cliente no tiene adelantos del progreso del sistema.
- Los desarrolladores construyen el sistema basados en los requisitos originales.
- Los errores sólo se detectan después de instalado el sistema.

37

Crítica al Modelo de Cascada

- Codificar y Corregir



Modelo de Cascada

- Estimación de tiempo y costo con muy poca información.
 - Especificación de requisitos precisa, pero difícil evaluar si cumple las expectativas del cliente.
 - El cliente no siempre tiene claros sus requisitos.
 - La anticipación del cambio no se considera esencial.
 - La cantidad de documentos puede volver burocrático el desarrollo.

38

Consecuencias de la Cascada

- Los altos costos de mantenimiento de los sistemas actuales se deben en parte a que los requisitos desarrollados no son siempre las necesidades reales.
- Problemas contractuales:
 - ¿quién paga por el mantenimiento?
 - ¿qué hay de las cosas que debieran haber estado en el sistema y no están?

39

Modelo Alternativo

- Los errores existen siempre y el software está destinado a cambiar.
- “Do it twice” (Brooks, 1975).
 - Primera versión: comprobar factibilidad y requisitos (prototipo desechable).
 - Versión definitiva: construida con el conocimiento ganado con la primera versión (modelo de cascada).
- Este modelo no soluciona la calidad monolítica de la entrega.

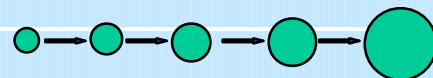
40

Modelo de Proceso Evolutivo

- “Es el modelo cuyas etapas consisten en expandir incrementos de un producto de software operacional, donde la dirección de la evolución la dicta la experiencia con el sistema.”
[Boehm, 1988]
- El cliente recibe pequeños incrementos del sistema a medida que van siendo desarrollados: distribución incremental.

41

Incrementos



- El modelo evolutivo de desarrollo no implica necesariamente entregas incrementales.
- Entregas incrementales implican no sólo código, sino también manuales de uso.
- Los incrementos deben ser unidades autocontenidas.

42

Modelo Evolutivo

- Etapas del modelo:
 - entregar al cliente algo útil,
 - medir el valor agregado del incremento,
 - ajustar el diseño y los objetivos en base a las mediciones.
- Sin rigor, el modelo evolutivo degenera rápidamente en codificar y corregir.

43

Variaciones

- Implementación Incremental.
 - Se sigue el modelo de cascada hasta la etapa de diseño y luego se implementan sucesivos incrementos.
- Integración Incremental.
 - Los módulos programados independientemente se van integrado y probando hasta tener el sistema completo.
- Desarrollo Incremental.
 - Se sigue una cascada completa para cada incremento. El usuario da sugerencias para los siguientes desarrollos.
 - sucesión de pequeñas cascadas,
 - más fácil estimar tiempo y costo de pequeños desarrollos,
 - el mantenimiento es parte del desarrollo.
- Prototipo Evolutivo.
 - El prototipo se hace evolucionar y crecer hasta obtener el sistema final.

44

Modelo de Transformaciones

- El desarrollo de software es una secuencia de transformaciones que van de la especificación de requisitos a la implementación.
 - Se analizan los requisitos y se los especifica formalmente.
 - Se toma la especificación y se la refina a un mayor nivel de detalle.
 - Se realizan transformaciones para hacer ejecutable cada parte de la especificación.
 - Se optimizan algunas implementaciones.

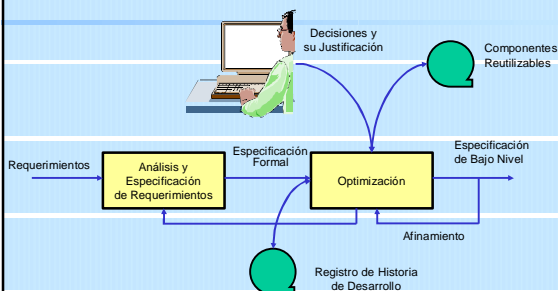
45

Mecanismos de Transformación

- El ingeniero de software decide qué transformación aplicar.
- La derivación de la transformación debe ser formalmente correcta.
- Es conveniente que exista una biblioteca de transformaciones posibles de dónde escoger.
- En cada transformación el cliente corrobora que los requisitos se cumplan.
- El uso de software de apoyo es esencial.
- El mantenimiento del software se realiza a partir de su historia de especificaciones.

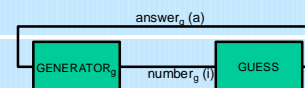
46

Modelo de Transformaciones Ideal



47

Generador de Números (Input/Output Automatas)



```

automaton GENERATOR (type Index, g : Index)
signature
  input answer (a : Bool, const g)
  output number (i : Int, const g)
states
  newtry : Bool := true
transitions
  input answer (a,g)
  eff: newtry := true
  output number (i,g)
  pre: 1 < i < 10 ^ newtry = true
  eff: newtry := false
  
```

48

Transformación del Generador

```

automaton GENERATOR (type Index, g : Index)
signature
  input answer (a : Bool, const g)
  output number (i : Int, const g)
states
  value : Int,
  newtry : Bool := true
transitions
  input answer (a,g)
  eff: newtry := true
  output number (i,g)
  pre: newtry = true;
  i:=choose i where 1 i 10
  eff: newtry := false

  automaton GENERATOR (type Index, g : Index)
  signature
    input answer (a : Bool, const g)
    output number (i : Int, const g)
  states
    value : Int := 1,
    newtry : Bool := true,
    last : Int
  transitions
    input answer (a,g)
    eff: newtry := true
    output number (i,g)
    pre: newtry = true;
    if last = 10 then i := 1
    else i := last + 1
    fi;
    eff: newtry := false;
    last := i
  
```

49

Mantenimiento en Transformación

- Modelo de Cascada:
 - los cambios no se anticipan;
 - el mantenimiento es visto como cambios de emergencia;
 - se realiza bajo presión de los clientes y la gerencia;
 - los cambios suelen hacerse sólo en el código;
 - la especificación y el código divergen con el tiempo.
- Modelo de Transformaciones:
 - la historia de transformaciones de los sistemas está registrada;
 - los cambios se realizan en el punto de la historia apropiado;
 - todas las transformaciones siguientes se aplican nuevamente;
 - se obtiene un sistema formalmente correcto.

50

¿Un Ideal?

- El método de Transformaciones se aplica poco en la práctica.
- Poco software de apoyo más allá de la investigación académica.
- La automatización nunca es total, requiere de la creatividad del ingeniero de software para elegir la transformación más apropiada.
- Un sistema experto que sugiera la transformación posible sería de gran ayuda.

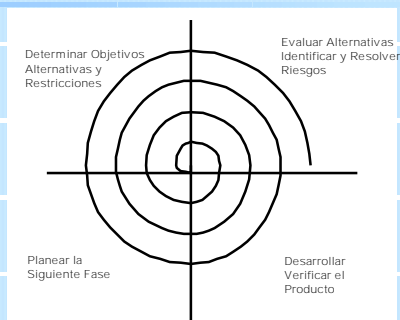
51

Modelo de Espiral

- El Modelo de Espiral brinda un marco para desarrollar software controlando los riesgos asociados.
- Es un metamodelo: aloja los otros modelos como parte de sí.
- Riesgo: circunstancias potencialmente adversas que pueden impedir el proceso de desarrollo y la obtención de un producto de calidad.
- Administración del Riesgo: disciplina cuyo objetivos son identificar, atender y eliminar los riesgos del software antes de que amenacen el éxito del proyecto o los costos de rehacer el software.

52

Etapas del Espiral



53

Un Modelo Cíclico

- El modelo consiste en una serie de ciclos.
- Cada ciclo tiene cuatro etapas (cuadrantes):
 - determinar objetivos,
 - evaluar alternativas e identificar riesgos,
 - desarrollar y verificar,
 - planear la siguiente fase.
- El radio del espiral representa el costo acumulado por el proceso hasta el momento.
- El ángulo representa el progreso del proceso.

54