Ingeniería de software - IC

Facultad de Informática

UNIVERSIDAD NACIONAL DE LA PLATA

Ingeniería de Software

»Contenidos genéricos a desarrollar a lo largo del curso

- Conceptos de ingeniería de software.
- Modelos de proceso.
- Requerimientos.
- Gestión de Proyectos
  - *Planificación*
  - *Métricas*
  - *Riesgos*
- Diseño
  - *Conceptos, Arquitectónico, Interfaz*
- Verificación y Validación
- Mantenimiento
- Calidad de software.

Fuente:

Ingeniería de Software

»Bibliografía general de la materia:

Pfleeger Shari Lawrence. Ingeniería de software. Teoría y práctica. Prentice Hall.

Sommerville Ian. Ingeniería de software. Addison Wesley.

Pressman Roger. Ingeniería de Software. Un enfoque práctico. Mc Graw Hill.

Whitten & Bentley. System Analysis and Design Methods.

Kendall y Kendall. Systems Analysis and Design

Fuente:

Ingeniería de Software

- Teorías
  - Lunes 11 a 14:00 – Aula 1-4
  - Prof. Ariel Pasini
- Prácticas
  - Miércoles 13 Hs a 16 Hs – Aula 1-2
  - Viernes de 8:00 Hs a 11 Hs – Aula 10 B
  - JTP: Nicolás Galdámez
  - Ayudante :Sebastián Rodríguez Eguren
- Contacto
  - http://blogs.unlp.edu.ar/ingenieriadesoftware/
  - Webunlp - curso de Ingeniería De Software

Fuente:

Conceptos generales

5

Software

- ¿Qué es Software?

  Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación (IEEE)

- Tipos de producto de software

  Genéricos: Sistemas aislados producidos por organizaciones desarrolladoras de software y que se venden en un mercado abierto.

  Personalizados: Sistemas requeridos por un cliente en particular. Desarrollados por la propia organización interesada o un contratista.

Software

- Clasificación de Software
  - De sistemas (sirve a otros programas)
  - De gestión (proceso de información comercial)
  - Científico (algoritmos de manejo de números)
  - Empotrado (reside en memoria)
  - De tiempo real (coordina/analiza/controla sucesos del mundo real)
  - Basados en la Web (sitios)
  - De Inteligencia artificial (uso de algoritmos no numéricos para resolver problemas complejos)
  - Otros…

Fuente:

Un poco de historia sobre el software

»1950-60

Hardware de propósito general, con continuos cambios. Software “a medida” Procesamiento batch Desarrollo sin planificación → “DOCUMENTACIÓN INEXISTENTE”

»1960-70

Multiprogramación Multiusuario Minicomputadoras. Tiempo real. Bases de datos. Lenguajes. Nace el “producto software” → "MANTENIMIENTO DEL SOFTWARE“

»1970-90

Complejidad. Microprocesadores. Redes. Procesos distribuidos. PC. Software de aplicación en todas las áreas. Baja el costo del hardware → “PRODUCTIVIDAD”

»1990-2000

Tecnologías O.O. Procesamiento paralelo. Recurso de procesamiento ilimitado. Capacidad de aprendizaje del usuario es un límite → "INGENIERÍA"

»2000-act

Aplicaciones WEB. Sistemas Expertos. Reconocimiento de Patrones. Redes Neuronales. Computación Ubicua. Código abierto.→ “COMUNICACIÓN Y DISTRIBUCIÓN”

Fuente:

Características del Software

»Características del Software

Es un elemento lógico

El software se desarrolla, no se fabrica como otros productos

Mayor costo en la ingeniería que en la producción.

Tendencia importante de construcción por componentes, pero aún se construyen a medida

Fuente:

Evolución del software

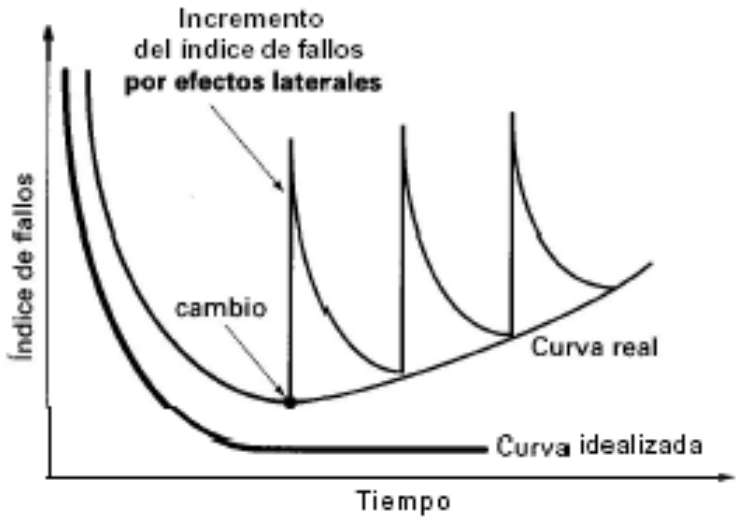
»Evolución del software

El software no se desgasta.

No sigue una curva clásica de envejecimiento.

Es inmune a los males que desgastan al hardware.

El problema no está en el tiempo de operación, sino en los cambios.



¿Qué es la ingeniería de software?

- ¿Qué es la ingeniería de software?
  - Disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iniciales de la especificación del sistema incluyendo la evolución de éste luego que se comienza a ejecutar.

Disciplina de la ingeniería

Hace que las cosas funcionen.

Se aplican teorías, métodos y herramientas.

Todos los aspectos de la producción de software

No sólo comprende los procesos técnicos del desarrollo de software, sino también se realizan actividades como la gestión de proyectos y el desarrollo de herramientas, métodos y teorías de apoyo a la producción de software

Fuente: Sommerville – Capítulo 1

¿Qué es la ingeniería de software?

»La IEEE define a la Ingeniería de Software como:

El uso de métodos sistemáticos, disciplinados y cuantificables para el desarrollo, operación y mantenimiento de software

El estudio de técnicas relacionadas con lo anterior

»Richard Fairley propone:

La Ingeniería de Software es la disciplina tecnológica y de administración que se ocupa de la producción y evolución sistemática de productos de software que son desarrollados y modificados dentro de los tiempos y costos estimados

Fuente:

¿Qué es la ingeniería de software?

- Usa métodos sistemáticos “Cuantificables”

  La cuantificación rigurosa de recursos, procesos y productos es una precondición para optimizar productividad y calidad. La “metrificación” y el control estadístico de procesos son claves en Ingeniería de Software

- “Dentro de tiempos y costos estimados”

  Un Ingeniero de Software debe cumplir contratos en tiempo y costos como es normal en obras de Ingeniería. Ello presupone la capacidad de medir, estimar, planificar y administrar proyectos

- Para el “Desarrollo, operación y mantenimiento”

  La Ingeniería de Software se ocupa de todo el ciclo de vida de un producto, desde su etapa inicial de planificación y análisis de requerimientos hasta la estrategia para determinar cuándo y cómo debe ser retirado de servicio

Fuente:

¿Qué es la ingeniería de software?

- La Ingeniería de Software surgió como reacción a las dificultades de desarrollar software sobre la base de habilidad, experiencia o intuición individuales
- La aplicación de “métodos sistemáticos” y repetibles permite la producción y evolución de software por organizaciones distribuidas geográficamente y en el tiempo.
- La aplicación de estándares facilita la integración, reusabilidad mantenimiento de los productos
- Es una “Disciplina tecnológica y de administración”

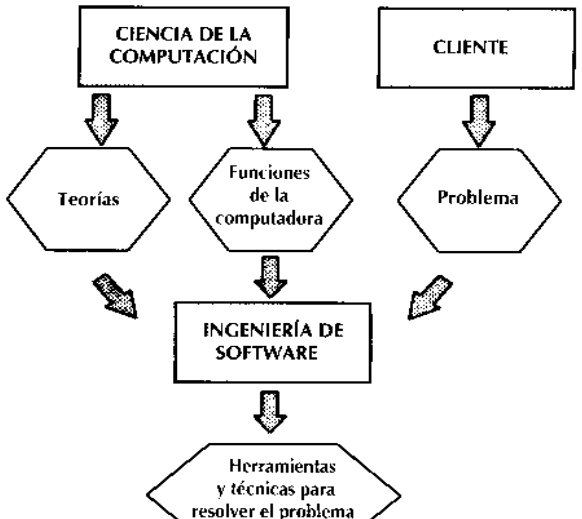
Fuente: Sommerville – Capitulo 1

¿Qué es la ingeniería de software?

»Existen elementos que distinguen a la Ingeniería de Software de las ciencias de la computación

- Ciencia de la computación
  - *Se refiere a las teorías y métodos subyacentes a las computadoras y los sistemas de software*
- Los IS requieren ciertos conocimiento de la ciencia de la computación

La Ingeniería de Software es una disciplina de “producción y evolución” de productos, no trata temas abstractos.



Fuente:

¿Qué es la ingeniería de software?

- En resumen:

  La ingeniería de software trata de dar principios y métodos que permitan producir software confiable y eficiente, al menor costo posible.

  Para esto la ingeniería de software establece métodos, desarrolla herramientas automáticas o semiautomáticas y define procedimientos que establecen la relación de métodos y herramientas.

Fuente:

Un poco de historia sobre la IS

»En los años 60, con la sofisticación creciente de los sistemas de software crecieron también las dificultades para desarrollarlos o adaptarlos. Las dificultades desbordaban los recursos técnicos de una heterogénea clase profesional.

"CRISIS DEL SOFTWARE"

»El valor estratégico del software llevó a la Organización del Tratado del Atlántico Norte (OTAN) a organizar un par de conferencias que tuvieron carácter fundacional para la Ingeniería de Software (Garmish 1968 y Roma 1969).

»El propósito de estas conferencias fue identificar la raíz de los problemas que enfrentaba la incipiente industria del software y sentar las bases de procesos sistemáticos, repetibles y confiables. En la reunión de Roma se comenzó a utilizar la expresión "Ingeniería de Software".

Fuente: ¿Qué es la Ingeniería de Software? – Guillermo Arango – Sadio 1993

Un poco de historia sobre la IS

- Era funcional – años 60

  Se estudia cómo explotar la tecnología para hacer frente a las necesidades funcionales de las organizaciones

- Era de control – años 70

  Aparece la necesidad de desarrollar software en tiempo, planeado y controlado. Se introduce el modelo de ciclo de vida en fases.

- Era de costos – años 80

  La importancia de la productividad en el desarrollo de software se incrementa sustancialmente. Se ponen en práctica varios modelos de costos.

- Era de calidad – años 90 a la actualidad

  Se intensifica la necesidad de que el producto tenga atributos que satisfagan las necesidades explícitas e implícitas del usuario: mantenibilidad, confiabilidad, eficiencia, usabilidad

Fuente: Sommerville – Capítulo 1

¿Qué conocimientos debe tener un IS?

- El Ingeniero de Software debe:
  - Tener una combinación de conocimientos científicos, metodológicos, tecnológicos y administrativos.
  - Estar familiarizado con la aplicación de métodos formales: lógica, estadística, simulación y con el uso de notaciones de modelización, especificación, diseño, programación
  - Poder aplicar metodologías de documentación, análisis, especificación, diseño, implementación y prueba. Debe conocer las ventajas y limitaciones de cada notación y cada técnica. Debe saber cómo y cuándo aplicarlas.
  - Conocer las tecnologías y productos: sistemas operativos, lenguajes, herramientas CASE, bases de datos, sistemas generadores de interfaces, bibliotecas de código.
  - Conocer técnicas de administración de proyectos: planificación, análisis de riesgos, control de calidad, seguimiento de proyectos, control de subcontratistas, etc.
  - En los últimos años se observa una especialización de los ingenieros de software por dominio de aplicación o por actividad

Fuente:

Responsabilidad profesional y ética

»La Ingeniería de Software se desarrolla en un marco económico, social y legal.

Los IS deben aceptar responsabilidades más amplias que las responsabilidades técnicas

»No debe utilizar sus capacidad y habilidades de forma deshonesta, o de forma que deshonre la profesión.

Fuente: Sommerville – Capítulo 1

Responsabilidad profesional y ética

- Confidencialidad
  - Respetar la confidencialidad de sus empleados y clientes
- Competencia
  - No falsificar el nivel de competencia y aceptar responsabilidades fuera de su capacidad
- Derechos de la propiedad intelectual
  - Conocer la leyes vigentes sobre las patentes y copyright
- Uso inapropiado de las computadoras
  - No debe utilizar sus habilidades técnicas para utilizar de forma inapropiada otras computadoras
- Existen diferentes organización cono ACM o IEEE que sugieren diferentes códigos de ética a respetar

Fuente: Sommerville – Capítulo 1

Responsabilidad profesional y ética

Falla del misil Patriot. Mal cálculo del tiempo desde el comienzo debido a errores aritméticos. En unas 100 horas acumulaba 0.34 segundos de error.

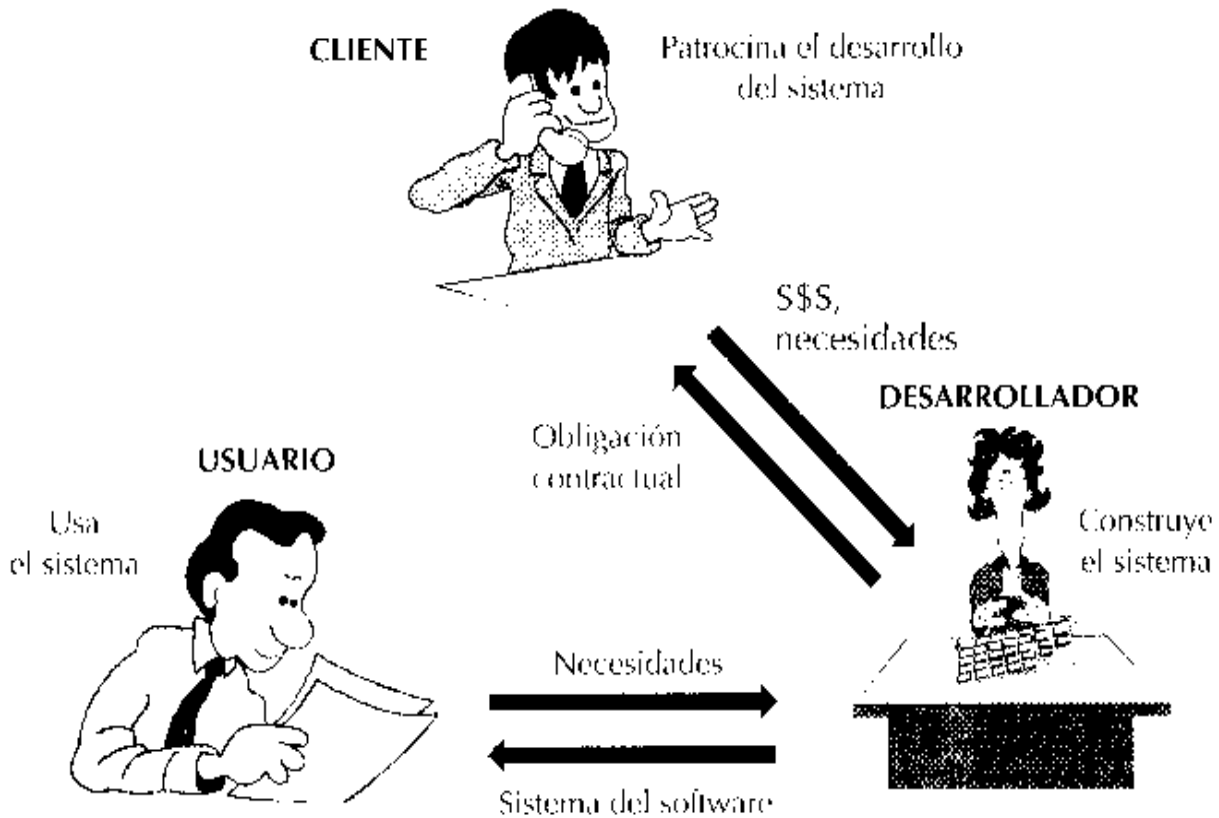
Explosión del Arianne 5. Explota a poco de salir por pérdida total de información de guiado y altitud. El origen es un error de especificación y diseño en el software del sistema de referencia inercial.

Plataforma Sleipner A. Producía petróleo y gas en el Mar del Norte. Una falla en las paredes submarinas originada en un error en análisis de elementos finitos e insuficiente refuerzo.

Fuente:

Participantes en el Desarrollo del Software

»Participantes en el Desarrollo del Software



¿Qué es un proceso de software?

»Es un conjunto de actividades y resultados asociados que producen un producto de software

»Actividades fundamentales de los procesos

- Especificación del software
- Desarrollo del software
- Validación del software
- Evolución del software

»Los IS son los responsables de realizar estas actividades

Fuente: Sommerville – Capitulo 1

¿Qué es un modelo de proceso de software?

- Es una descripción simplificada de un proceso de software que presenta una visión de ese proceso.
- Estos modelos pueden incluir actividades que son partes de los procesos y productos de software, y el papel de las personas involucradas.
- La mayoría de los modelos de proceso de software se basan en uno de los siguientes modelos generales o paradigmas
  - **Modelo en cascada**: Representa las actividades anteriores y las representa como fases de proceso separadas.
    - *Especificación de requerimientos, diseño, implantación, etc.*
  - **Desarrollo iterativo** : Un sistema inicial se desarrolla rápidamente a partir de una especificación abstracta. Éste se refina basándose en las peticiones del cliente.
  - **IS basada en componentes**: Esta técnica supone que las partes ya existen. El proceso se enfoca en la integración de las partes.
- .

Fuente: Sommerville – Capitulo 1

¿Cuáles son los atributos de un buen software?

- Los productos de software tiene un cierto número de atributos asociados que reflejan su calidad.
  - Estos atributos reflejan su comportamiento durante su ejecución y la estructura y organización de los programas fuentes en la documentación asociada
- Los atributos básicos son
  - Mantenibilidad
    - *Posibilidad de modificaciones ante los cambios del negocio*
  - Confiabilidad
    - *Fiabilidad, seguridad, no debe causar daños físico o económicos ante fallas*
  - Eficiencia
    - *Hacer un uso apropiado de los recursos*
  - Usabilidad
    - *Fácil de usar sin esfuerzo adicional*

Fuente: Sommerville – Capitulo 1

Modelo de Procesos - Tradicionales

27

Facultad de Informática
UNIVERSIDAD NACIONAL DE LA PLATA

Modelo de Proceso

- Proceso
  - Actividades que involucran, restricciones y recursos que producen una determinada salida
  - Características
    - *Establece todas las actividades*
    - *Utiliza recursos, está sujeto a restricciones y genera productos intermedios y finales*
    - *Puede estar compuesto por subprocesos*
    - *Cada actividad tiene entradas y salidas definidas*
    - *Las actividades se organizan en una secuencia*
    - *Existen principios que orientan sobre las metas de cada actividad*
    - *Las restricciones pueden aplicarse a una actividad, recurso o producto.*

Modelo de Proceso

»Ciclo de vida

Proceso que implica la construcción de un producto

»Ciclo de vida del Software

Describe la vida del producto de software desde su concepción hasta su implementación, entrega utilización y mantenimiento

»Modelos de proceso de software

Es una representación abstracta de un proceso del software.

Modelo de proceso
Paradigma de software
Ciclo de vida del software

} Términos Equivalentes

Modelo de Proceso

- Modelos prescriptivos

  Prescriben un conjunto de elementos del proceso: actividades del marco de trabajo, acciones de la ingeniería del software, tareas, aseguramiento de la calidad y mecanismos de control.

  Cada modelo de proceso prescribe también un “flujo de trabajo”, es decir de que forma los elementos del proceso se interrelacionan entre sí.

- Modelos descriptivos

  Descripción en la forma en que se realizan en la realidad

- Ambos modelos deberían ser iguales

Fuente: Pfleeger

Modelo de Proceso

- Modelo en Cascada
- Modelo en V
- Modelo de Prototipos
- Desarrollo por fases
  - Incremental
  - Iterativo
- El modelo espiral

Fuente: Pfleeger

Modelo de Proceso

»Modelo en Cascada

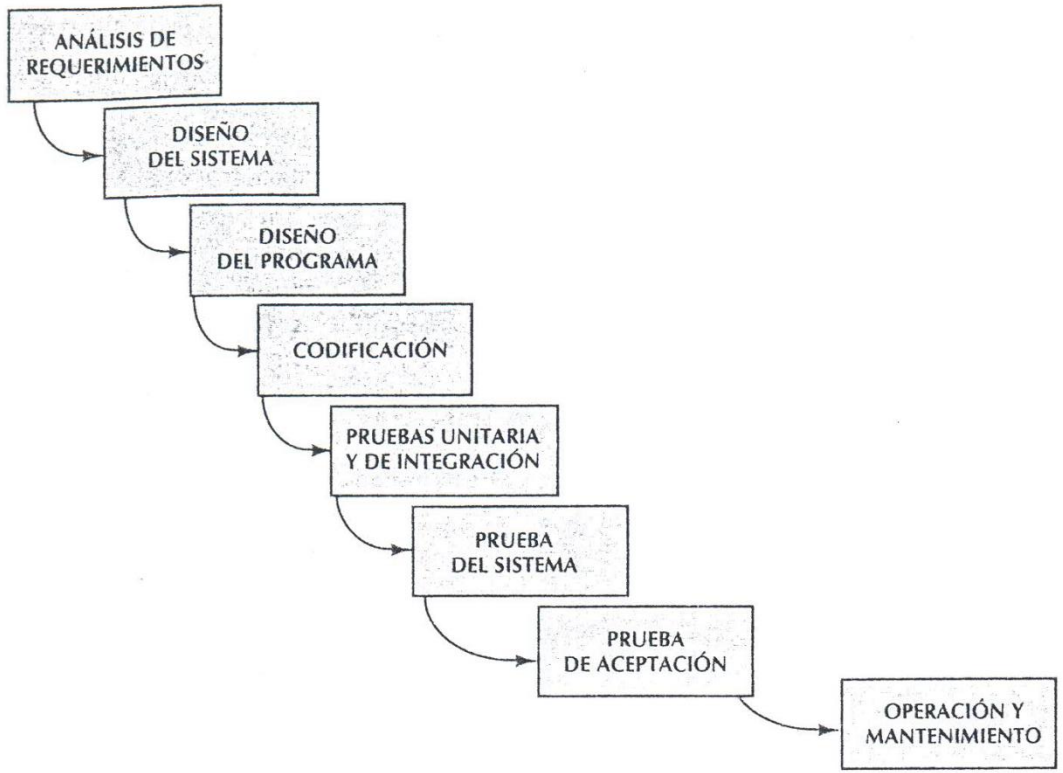
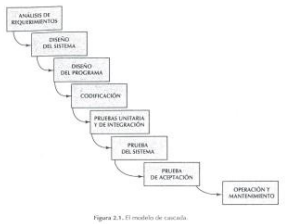


Figura 2.1. El modelo de cascada.

Fuente: Pfleeger

Modelo de Proceso

- Modelo en Cascada
  - Las etapas se representan cayendo en cascada
  - Cada etapa de desarrollo se debe completar antes que comience la siguiente
  - Útil para diagramar lo que se necesita hacer
  - Su simplicidad hace que sea fácil explicarlo a los clientes



Fuente: Pfleeger

Modelo de Proceso

»Modelo en Cascada

Dificultades:

No existen resultados concretos hasta que todo este terminado.

Las fallas más triviales se encuentran al comienzo del período de prueba y las más graves al final.

La eliminación de fallas suele ser extremadamente difícil durante las últimas etapas de prueba del sistema.

Deriva del mundo del hardware y presenta una visión de manufactura sobre el desarrollo de software.

El software no se desarrolla de la misma manera

La necesidad de prueba con la computadora aumenta exponencialmente durante las etapas finales de prueba.

"CONGELAR" una fase es poco realista.

Existen errores, cambios de parecer, cambios en el ambiente.

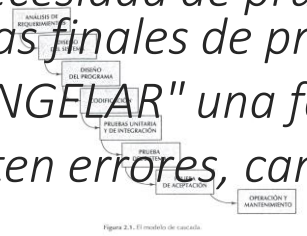


Figura 2.1. El modelo de cascada.

Fuente: Pfleeger

Modelo de Proceso

»Modelo de la realidad (sin control entre las etapas)

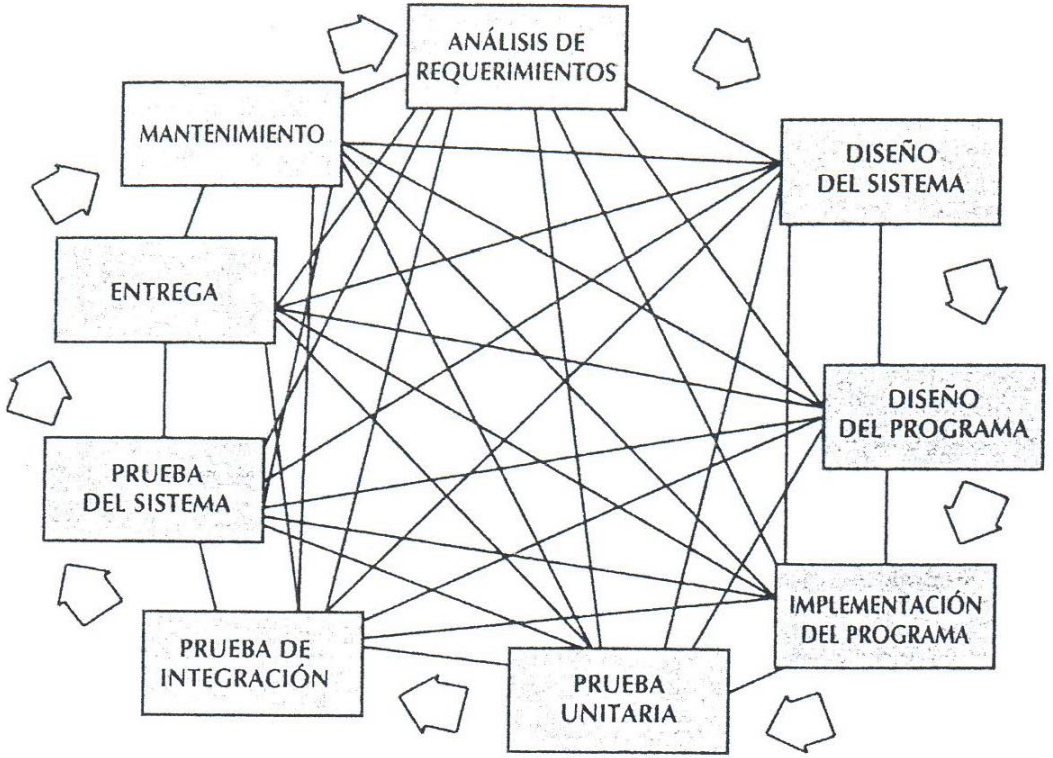


Figura 2.2. El proceso de desarrollo del software en la realidad.

Modelo de Proceso

»Modelo en Cascada
Con prototipo

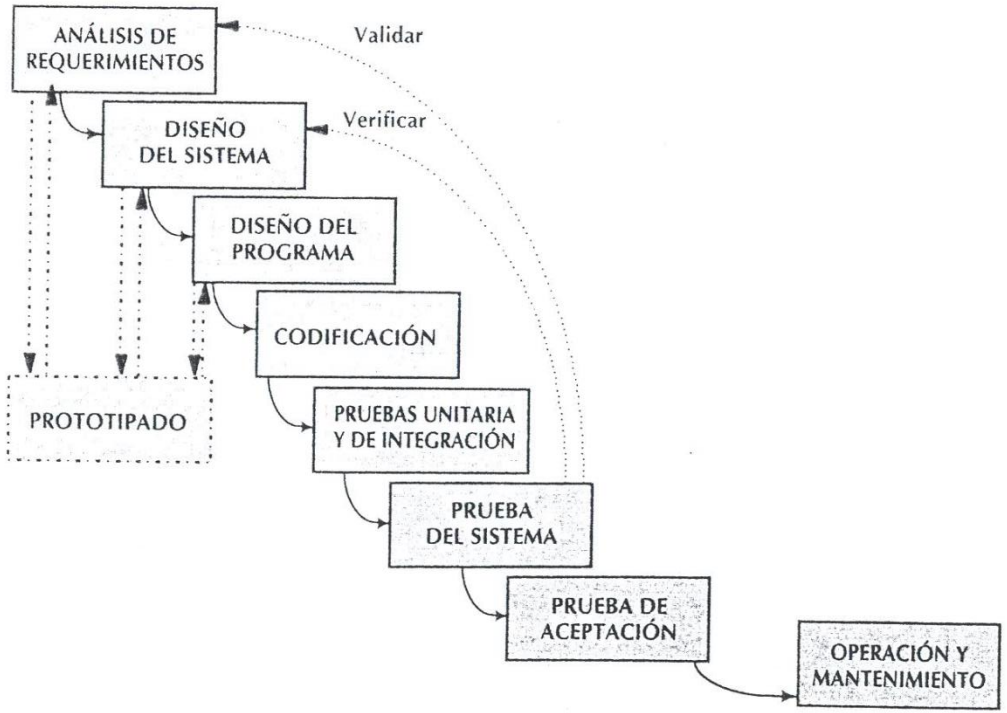


Figura 2.3. El modelo de cascada con prototipado.

Fuente: Pfleeger

Modelo de Proceso

»Modelo en V

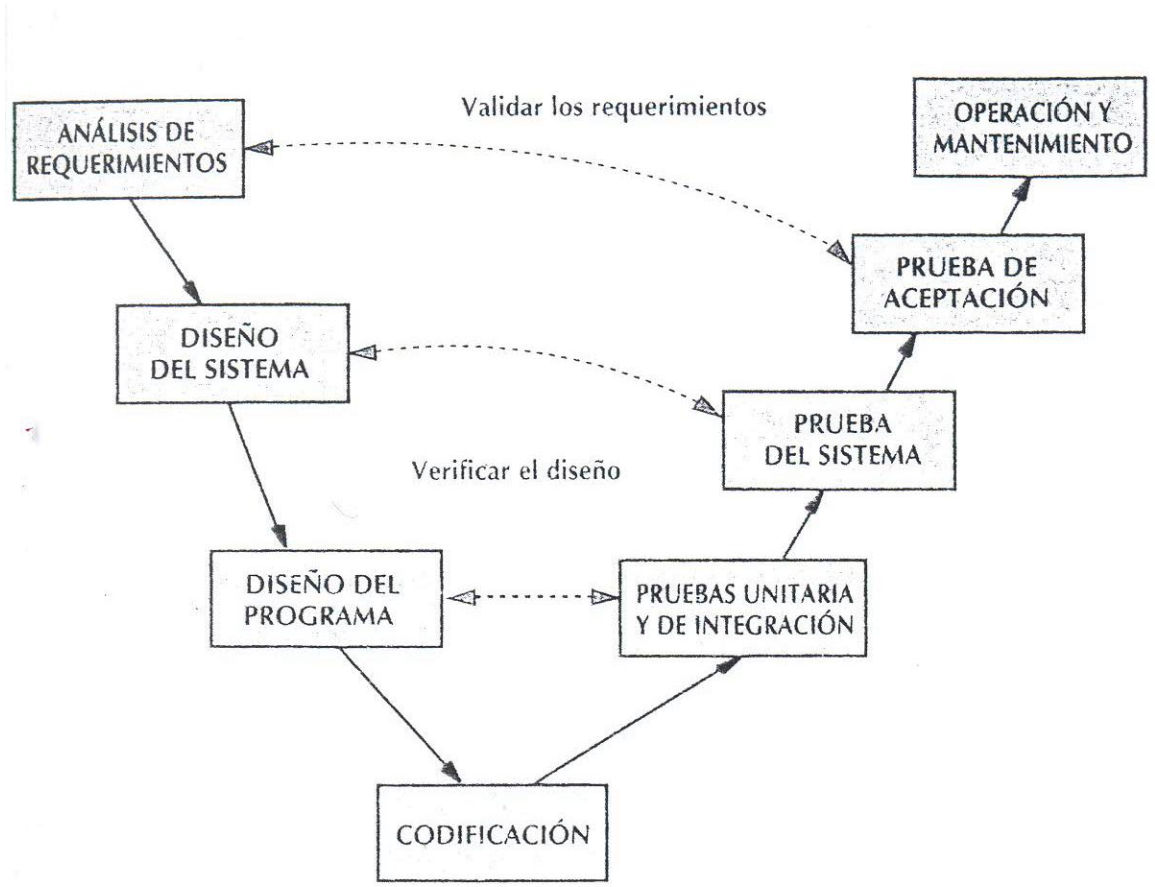


Figura 2.4. El modelo V. Pfleeger

Modelo de Proceso

»Modelo en V

Demuestra como se relacionan las actividades de prueba con las de análisis y diseño.

Sugiere que la prueba unitaria y de integración también sea utilizada para verificar el diseño del programa

La vinculación entre los lados derecho e izquierdo implica que, si se encuentran problemas durante la verificación y validación, entonces el lado izquierdo de la V puede ser ejecutado nuevamente para solucionar el problema.

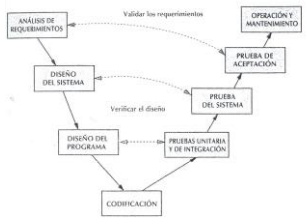


Figura 2.4. El modelo V.

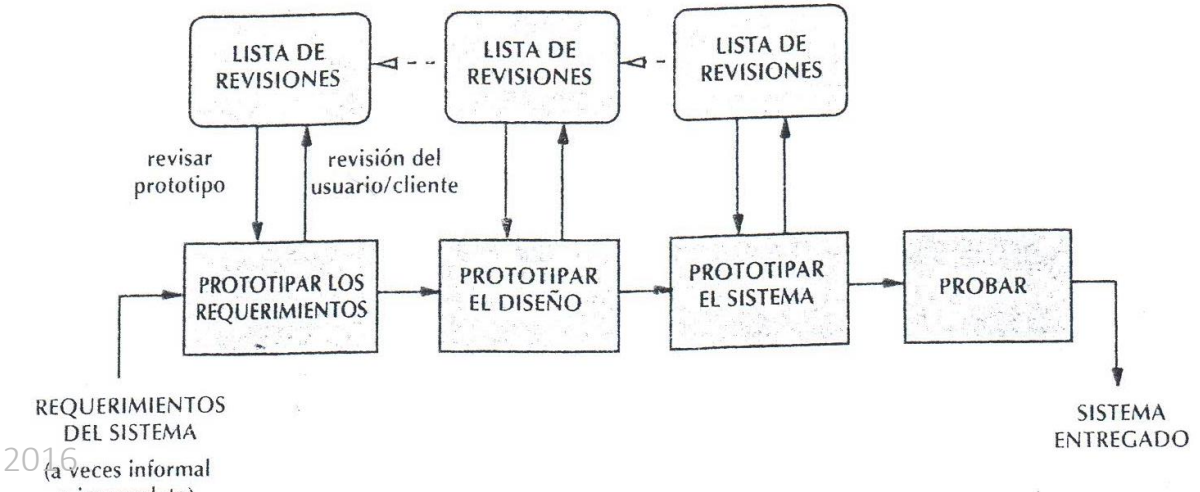
Fuente: Pfleeger

Modelo de Proceso

»Modelo de Prototipos

Un prototipo es un producto parcialmente desarrollado que permite que clientes y desarrolladores examinen algunos aspectos del sistema propuesto, y decidan si éste es adecuado o correcto para el producto terminado.

Esta es una alternativa de especificación para tratar mejor la incertidumbre, la ambigüedad y la volubilidad de los proyectos reales.



Modelo de Proceso

»Modelo de Prototipos

Tipos

Evolutivos

El objetivo es obtener el sistema a entregar.

Permite que todo el sistema o alguna de sus partes se construyan rápidamente para comprender o aclarar aspectos y asegurar que el desarrollador, el usuario y el cliente tengan una comprensión unificada tanto de lo que se necesita como de lo que se propone como solución

Descartables

No tiene funcionalidad

Se utilizan herramientas de modelado

Figura 2.5. El modelo prototipado.

Fuente:

Modelo de Proceso

»Modelo de Prototipos

Proyectos candidatos

Usuarios que no examinarán los modelos abstractos

Usuarios que no determinarán sus requerimientos inicialmente

Sistemas con énfasis en los formatos de E/S más que en los detalles algorítmicos

Sistemas en los que haya que explorar aspectos técnicos

Si el usuario tiene dificultad al tratar con los modelos gráficos para modelar los requerimientos y el comportamiento

Si se enfatiza el aspecto de la interfaz humana

Figura 2.5. El modelo prototipado.

Fuente:

Modelo de Proceso

»Modelo de Prototipos

Para asegurar el éxito

Debe ser un sistema con el que se pueda experimentar

Debe ser comparativamente barato (< 10%)

Debe desarrollarse rápidamente

Énfasis en la interfaz de usuario

Equipo de desarrollo reducido

Herramientas y lenguajes adecuados

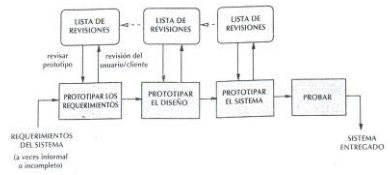


Figura 2.5. El modelo prototipado.

Fuente:

Modelo de Proceso

»Desarrollo por fases

Se desarrolla el sistema de tal manera que puede ser entregado en piezas. Esto implica que existen dos sistemas funcionando en paralelo: el sistema operacional y el sistema en desarrollo.

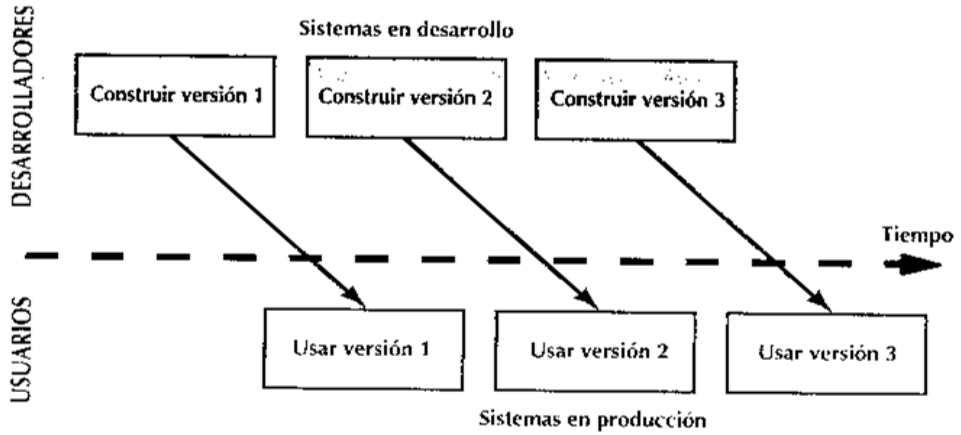


Figura 2.8. El modelo de desarrollo escalonado.

Fuente: Pfleeger

Modelo de Proceso

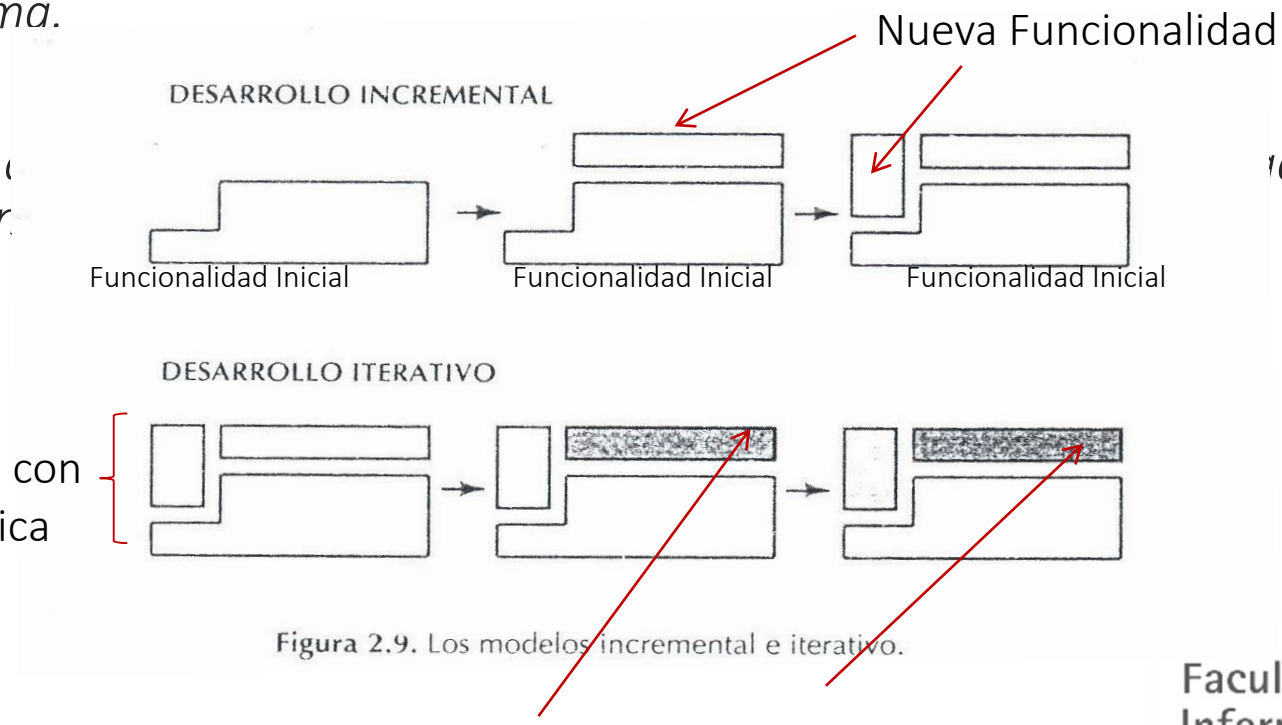
»Desarrollo por fases

Incremental

El sistema es particionado en subsistemas de acuerdo con su funcionalidad. Cada entrega agrega un subsistema.

Iterativo

Entrega un sistema c *d de cada subsistema con las r*



Modelo de Proceso

»El modelo espiral (Boehm)

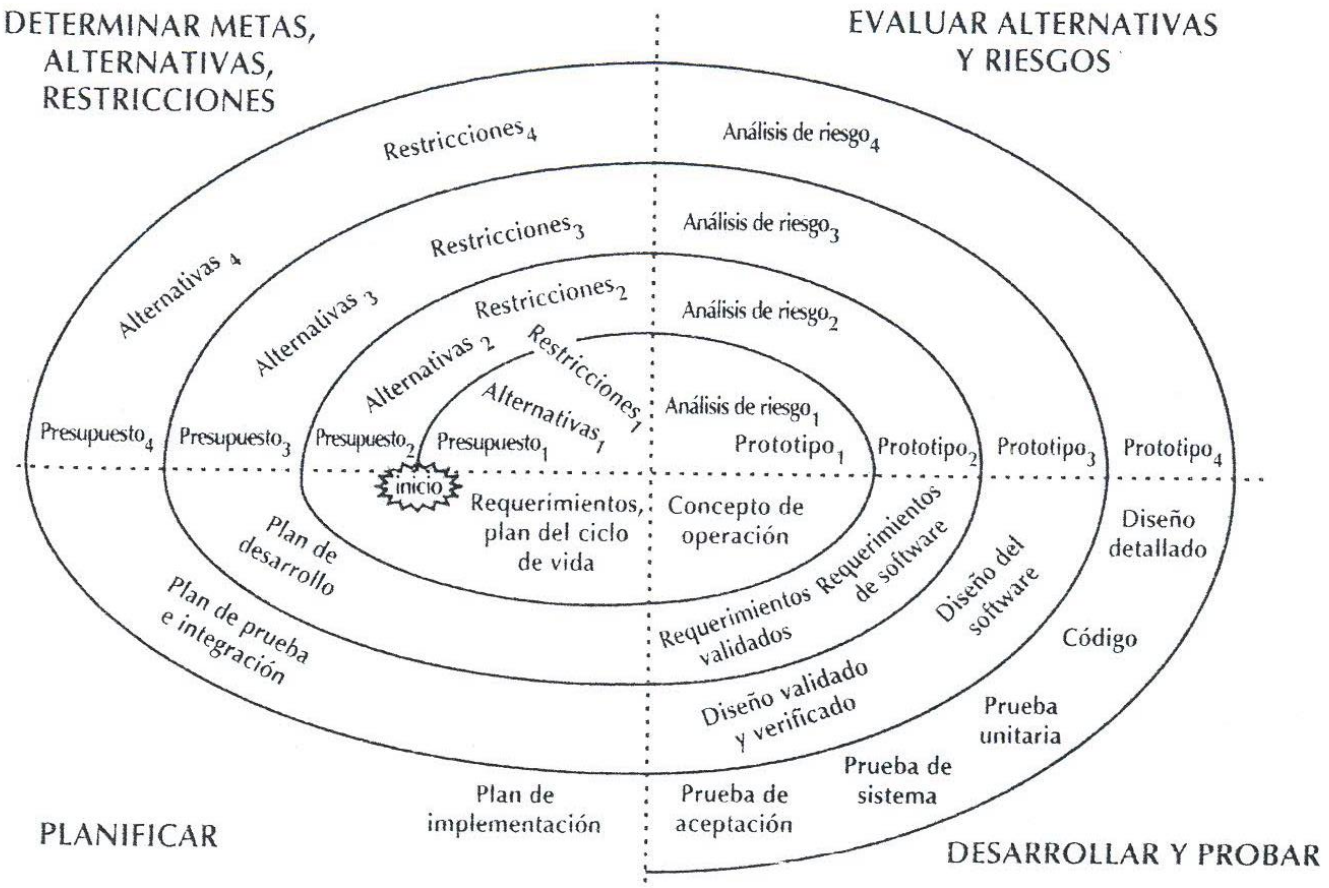


Figura 2.10. El modelo en espiral.

Modelo de Proceso

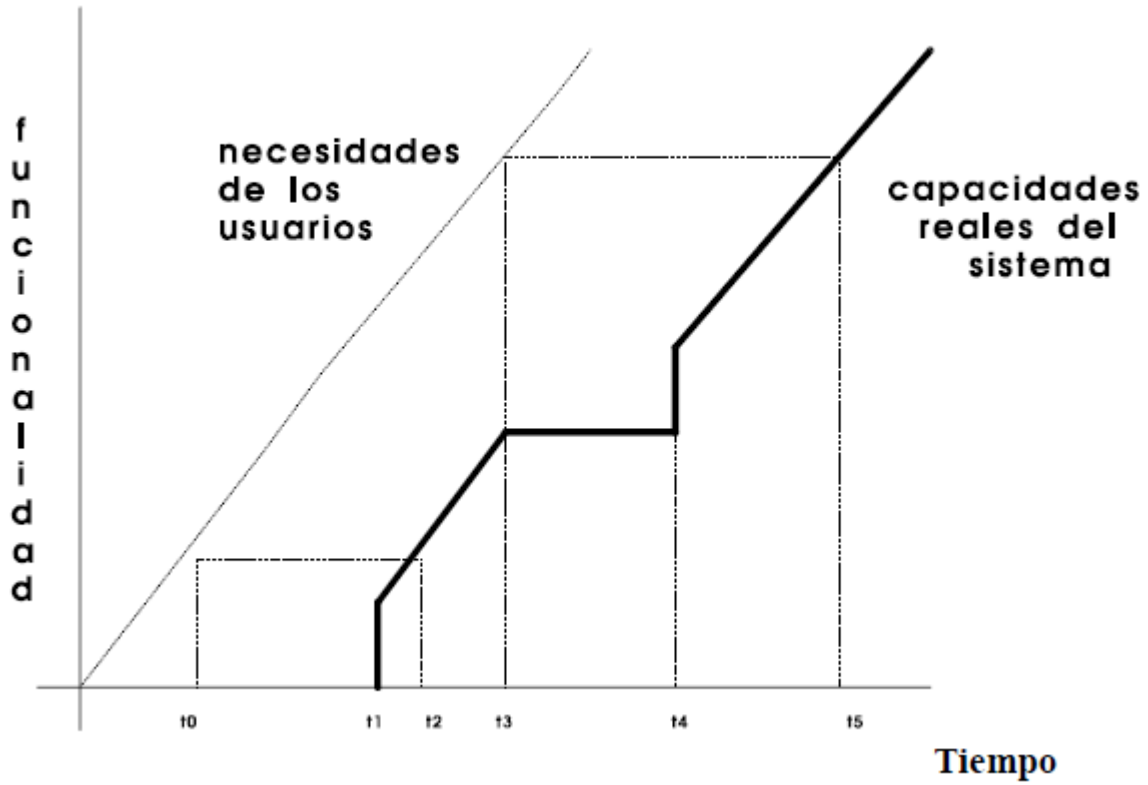
- El modelo espiral (Boehm)
  - Combina las actividades de desarrollo con la gestión del riesgo
  - Trata de mejorar los ciclos de vida clásicos y prototipos.
  - Incorpora objetivos de calidad y gestión de riesgos
  - Elimina errores y alternativas no atractivas al comienzo
  - Permite iteraciones, vuelta atrás y finalizaciones rápidas
  - Cada ciclo empieza identificando:
    - *Los objetivos de la porción correspondiente*
    - *Las alternativas*
    - *Restricciones*
  - Cada ciclo se completa con una revisión que incluye todo el ciclo anterior y el plan para el siguiente

Fuente: Pfleeger

Modelo de Proceso

»Análisis Comparativo

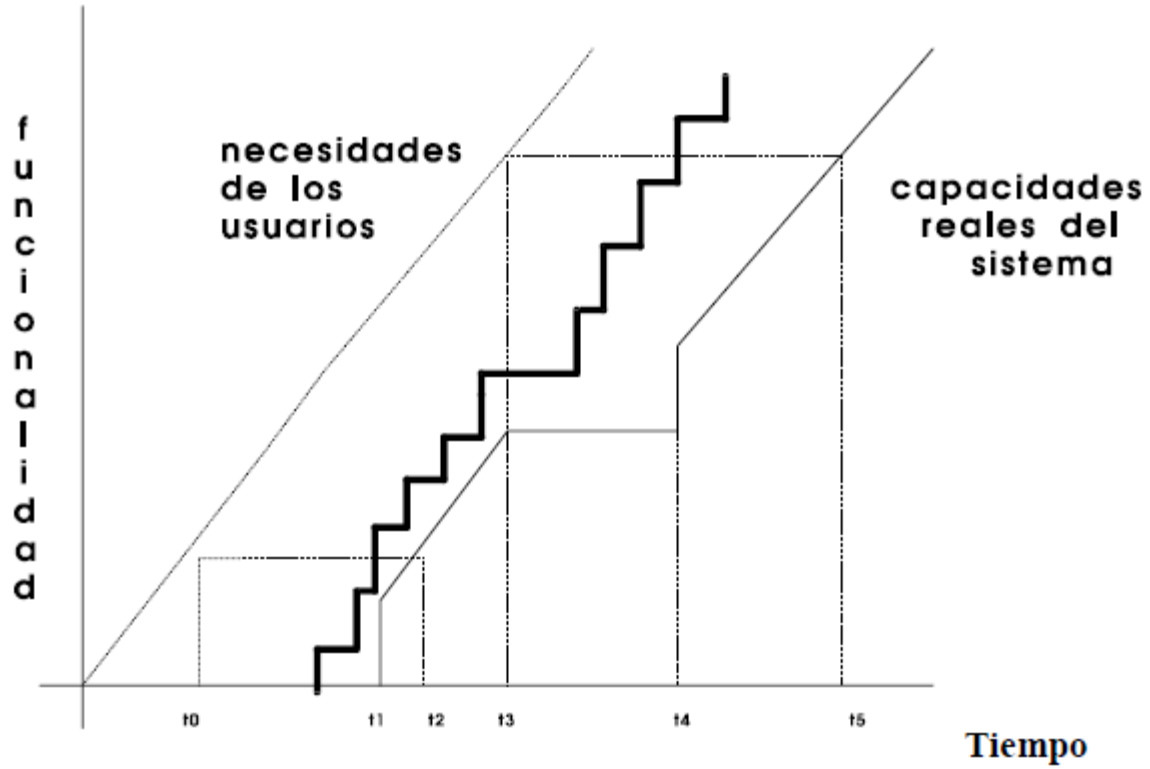
Clásico



Modelo de Proceso

»Análisis Comparativo

Incremental

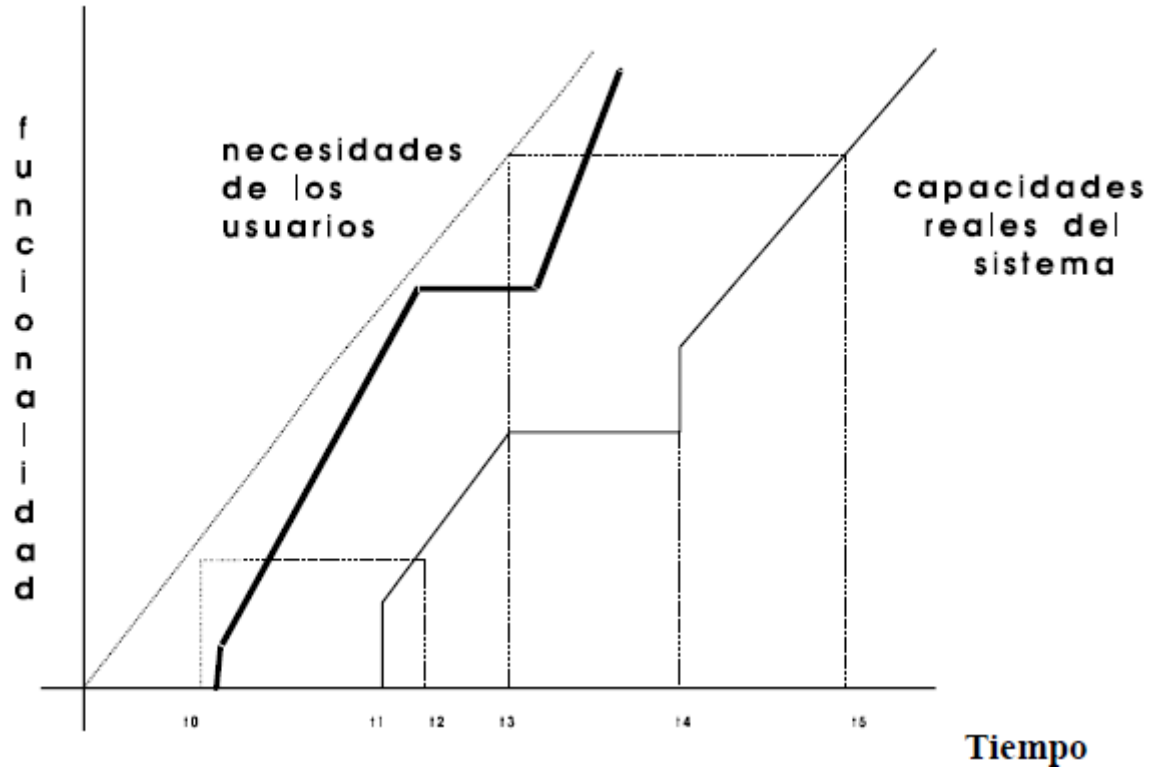


Fuente: Davis

Modelo de Proceso

»Análisis Comparativo

Prototipo Evolutivo



Fuente: Davis

Modelo de Procesos - Metodologías Agiles

50

Facultad de Informática
UNIVERSIDAD NACIONAL DE LA PLATA

Metodologías Agiles

- La ingeniería de software ágil combina una filosofía y un conjunto de directrices de desarrollo.
  - La filosofía busca
    - *La satisfacción del cliente y la entrega temprana del software incremental*
    - *Equipos de proyectos pequeños y con alta motivación*
    - *Métodos informales un mínimo de productos de trabajo de la ingeniería de software*
    - *Una simplicidad en general*
  - Las directrices resaltan
    - *La entrega sobre el análisis y el diseño, aunque estas actividades no se descartan*
    - *La comunicación activa y continua entre los desarrolladores y el cliente*

Metodologías Agiles

- El desarrollo ágil e software son métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requerimientos y soluciones evolucionan mediante la colaboración de grupo auto organizados y multidisciplinares.

- Existen muchos métodos de desarrollo ágil

  la mayoría minimiza riesgos desarrollando software en cortos lapsos de tiempo.

  El software desarrollado en una unidad de tiempo es llamado una iteración, la cual debe durar de una a cuatro semanas. Cada iteración del ciclo de vida incluye:

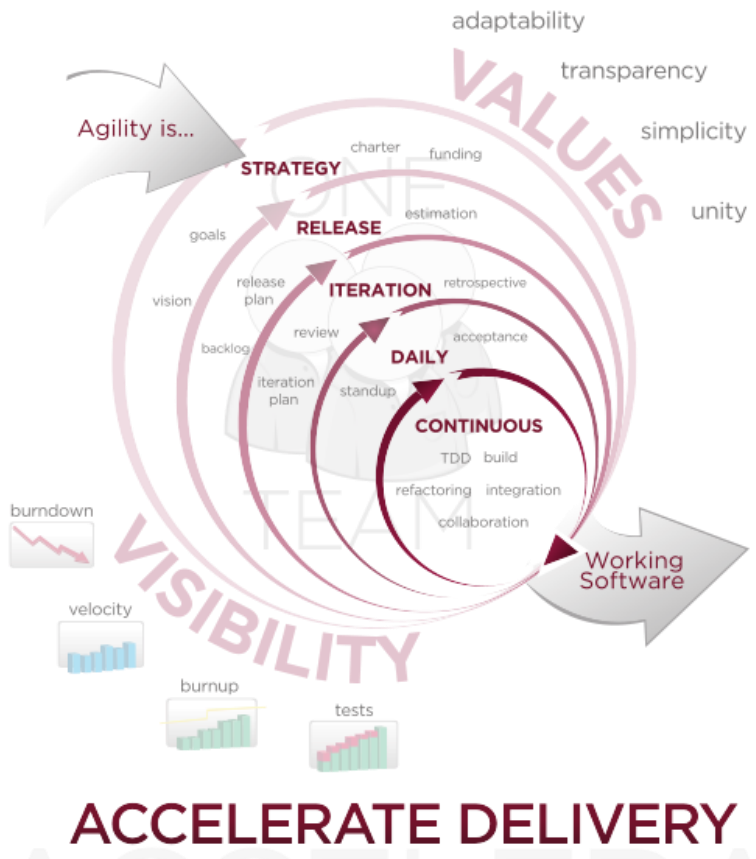
  Planificación, análisis de requerimientos, diseño, codificación, revisión y documentación.

  Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, pero la meta es tener una «demo» (sin errores) al final de cada iteración.

  Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto.

Fuente: Wikipedia

Metodologías Agiles



Fuente:

Metodologías Agiles

»La definición moderna de desarrollo ágil de software evolucionó a mediados de los años 1990 y en el año 2001, miembros prominentes de la comunidad se reunieron en Snowbird, Utah, y se creo la "*alianza ágil*", una organización sin fines de lucro que promueve el desarrollo ágil de aplicaciones y firmaron lo que se denomino el "*Manifiesto para el desarrollo ágil de software*"

Fuente:

Metodologías Agiles

»Manifiesto

Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

Individuos e interacciones *sobre procesos y herramientas*

Software funcionando *sobre documentación extensiva*

Colaboración con el cliente *sobre negociación contractual*

Respuesta ante el cambio *sobre seguir un plan*

Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.

Fuente:

Metodologías Agiles

»Manifiesto

1. Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor.
2. Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
3. Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los periodos breves.
4. Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.
5. Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.
6. La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.

Fuente:

Metodologías Agiles

»Manifiesto

7. El software que funciona es la principal medida del progreso.
8. Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica enaltece la agilidad.
10. La simplicidad como arte de maximizar la cantidad de trabajo que no se hace, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos que se auto-organizan.
12. En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.

Metodologías Agiles

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Tabla 1. Diferencias entre metodologías ágiles y no ágiles

Fuente:

Metodologías Agiles

- Metodologías mas relevantes
  - XP – extreme programming
  - Crystal
  - ASD -Adaptative software development
  - Scrum

Fuente:

eXtreme Programming

- Es una disciplina de desarrollo de software basado en los valores de la *sencillez*, la *comunicación*, la *retroalimentación*, la *valentía* y el *respeto*
- Su acción consiste en llevar a todo el equipo reunido en la presencia de prácticas simples, con suficiente información para que el equipo para ver dónde están y para ajustar las prácticas a su situación particular.
- En la programación extrema, cada colaborador del proyecto es una parte integral del " Equipo".

  Las formas del equipo en torno a un representante de la empresa llama "el Cliente", que se sienta con el equipo y trabaja con ellos todos los días.

eXtreme Programming

Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.

Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión.

Programación en parejas

Frecuente integración del equipo de programación con el cliente o usuario.

Corrección de todos los errores antes de añadir nueva funcionalidad.

Refactorización del código

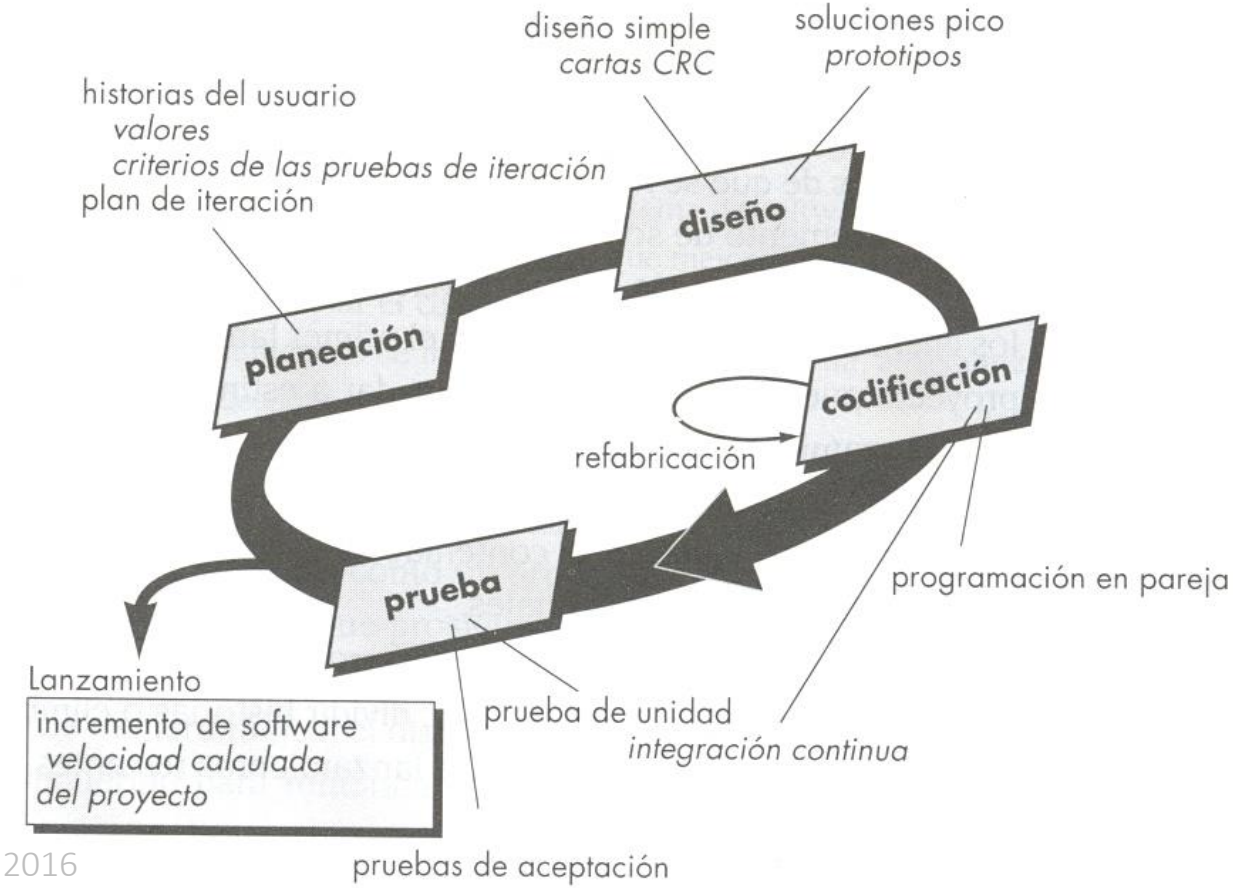
Propiedad del código compartida

Simplicidad en el código

Fuente: Las características fundamentales del método son:

eXtreme Programming

»eXtreme Programming



SCRUM

- Scrum es un proceso en el que se aplican, de manera regular, un conjunto de mejores prácticas para trabajar en equipo y obtener el mejor resultado posible de un proyecto.
- Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.
- En Scrum se realizan entregas parciales y regulares del resultado final del proyecto, priorizadas por el beneficio que aportan al receptor del proyecto

Scrum

- Eliminar el desperdicio
  No generar artefactos, ni perder el tiempo haciendo cosas que no le suman valor al cliente.
- Construir la calidad con el producto
  La idea es inyectar la calidad directamente en el código desde el inicio.
- Crear conocimiento
  En la práctica no se puede tener el conocimiento antes de empezar el desarrollo.
- Diferir las decisiones
  Tomar las decisiones en el momento adecuado, esperar hasta ese momento, ya que uno tiene mas información a medida que va pasando el tiempo. Si se puede esperar, mejor.
- Entregar rápido
  Debe ser una de las ventajas competitivas más importantes.
- Respetar a las personas
  La gente trabaja mejor cuando se encuentra en un ambiente que la motive y se sienta respetada.
- Optimizar el todo
  Optimizar todo el proceso, ya que el proceso es una unidad, y para lograr tener éxito y avanzar, hay que tratarlo como tal.

Scrum

»El Product Owner (Propietario)

Conoce y marca las prioridades del proyecto o producto.

»El Scrum Master (Jefe)

Es la persona que asegura el seguimiento de la metodología guiando las reuniones y a l equipo ante cualquier problema que pueda aparecer. Su responsabilidad es entre otra cer de paraguas ante las presiones externas.

»El Scrum Team (Equipo)

Son las personas responsables de implementar la funcionalidad o funcionalidades elegidas por el Product Owner.

»Los Usuarios o Clientes

Son los beneficiarios finales del producto, y son quienes viendo los progresos, pueden a as, sugerencias o necesidades

Scrum

- Product Backlog

  Es la lista maestra que contiene toda la funcionalidad deseada en el producto. La característica más importante es que la funcionalidad se encuentra ordenada por un orden de prioridad.

- Sprint Backlog

  Es la lista que contiene toda la funcionalidad que el equipo se comprometió a desarrollar durante un Sprint determinado.

- Burndown Chart

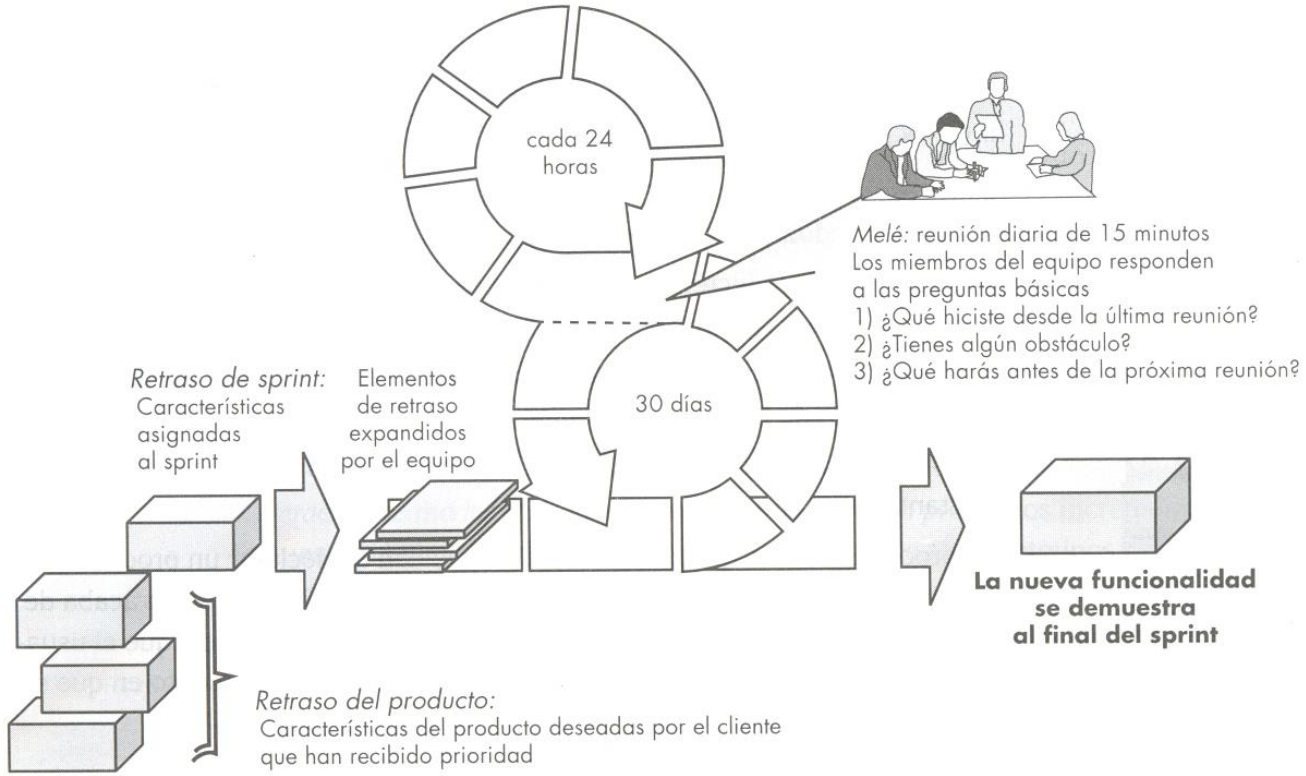
  Muestra un acumulativo del trabajo hecho, día-a-día.

- Entre otros…

Scrum

- Scrum es iterativo e incremental
  - Se busca poder atacar todos los problemas que surgen durante el desarrollo del proyecto
  - El nombre Scrum se debe a que durante los Sprints, lo que serían las fases de desarrollo, se solapan, de manera que no es un proceso de cascada por cada iteración, si no que tenemos todas éstas etapas juntas que se ejecutan una y otra vez, hasta que se crea suficiente.
  - Este solapamiento de fases se puede asemejar a un scrum de rugby, en el cual todos los jugadores (o roles, en nuestro caso), trabajan juntos para lograr un objetivo.

Scrum - Proceso



Fuente:

Scrum

»Scrum está pensado para ser aplicado en proyectos en donde el caos es una constante, aquellos proyectos en los que tenemos requerimientos dinámicos, y que tenemos que implementar tecnología de punta.

»Esos proyectos difíciles, que con los enfoques tradicionales se hace imposible llegar a buen puerto.

Fuente: ¿Cuándo usar Scrum?

Modelo de Procesos - Desarrollo de Software Dirigido por Modelos

Facultad de Informática
UNIVERSIDAD NACIONAL DE LA PLATA

Desarrollo de Software Dirigido por Modelos

- Hacia fines de los 70’ DeMarco introdujo el concepto de desarrollo de software «basado», en modelos.
- Destacó que la construcción de un sistema de software debe ser precedida por la construcción de un modelo, tal como se realiza en otros sistemas ingenieriles.
- Un modelo del sistema consiste en una conceptualización del dominio del problema y actúa como una especificación precisa de los requerimientos que el sistema de software debe satisfacer. (Abstracción de elementos del problema, comunicación, negociación con el usuario)
- El adjetivo «dirigido» en MDD, a diferencia de «basado», enfatiza que este paradigma asigna a los modelos un rol central y activo: son al menos tan importantes como el código fuente.

Fuente:

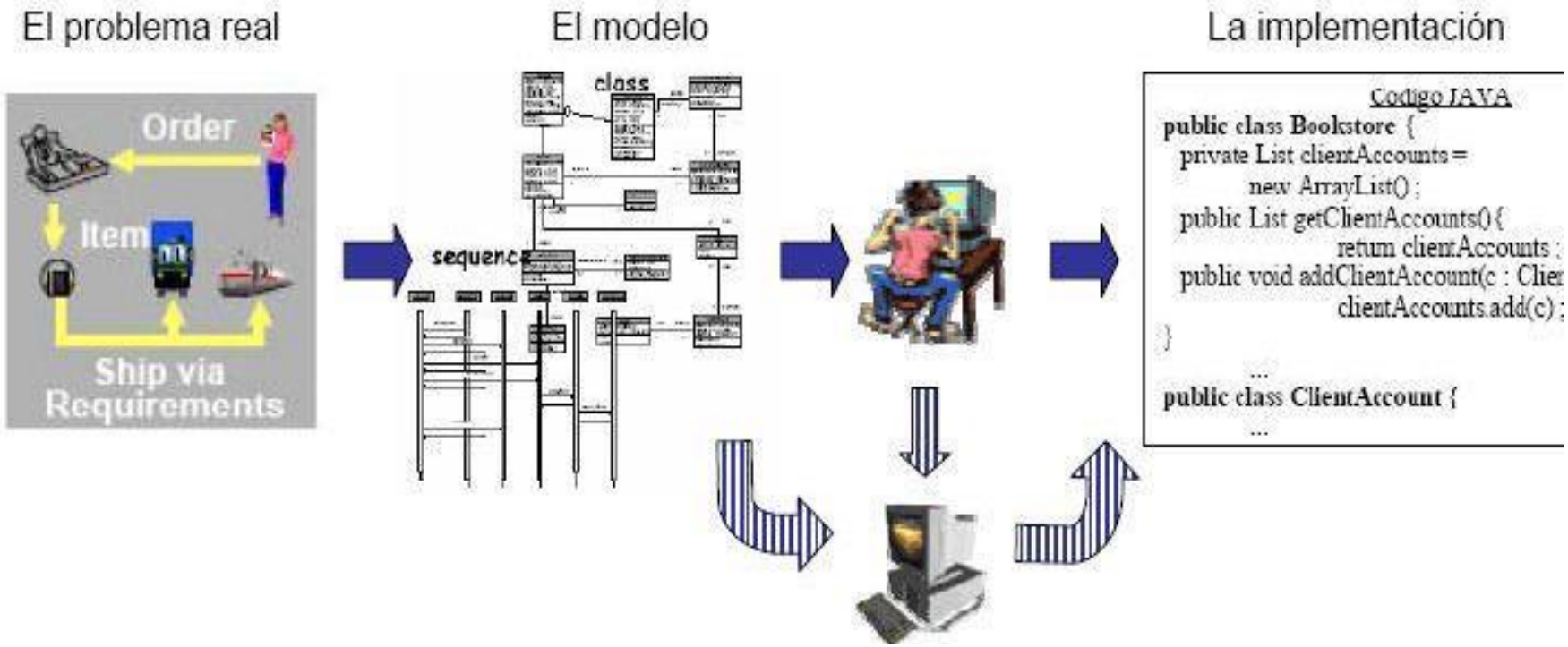
Desarrollo de Software Dirigido por Modelos

- Model Driven Development (MDD) promueve enfatizar los siguientes puntos claves:
  - Mayor nivel de abstracción en la especificación tanto del problema a resolver como de la solución correspondiente.
  - Aumento de confianza en la automatización asistida por computadora para soportar el análisis, el diseño y la ejecución.
  - Uso de estándares industriales como medio para facilitar las comunicaciones, la interacción entre diferentes aplicaciones y productos, y la especialización tecnológica.
  - Los modelos son los conductores primarios en todos los aspectos del desarrollo de software.

Fuente:

Desarrollo de Software Dirigido por Modelos

»Modelos y Transformaciones Automaticas



Fuente:

Desarrollo de Software Dirigido por Modelos

- PIM - Platform Independent Model

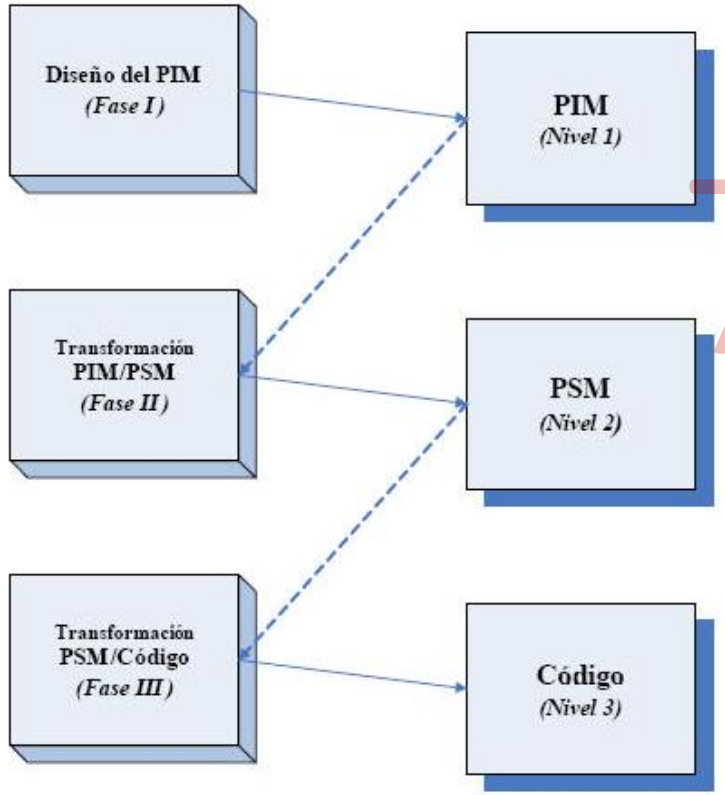
  "Un modelo de un sistema que no contiene información acerca de la plataforma o la tecnología que es usada para implementarlo"

- PSM - Platform Specific Model

  "Un modelo de un sistema que incluye información acerca de la tecnología especifica que se usará para su implementación sobre una plataforma especifica"

- Transformación de modelos

  "Especifica el proceso de conversión de un modelo en otro modelo del mismo sistema."



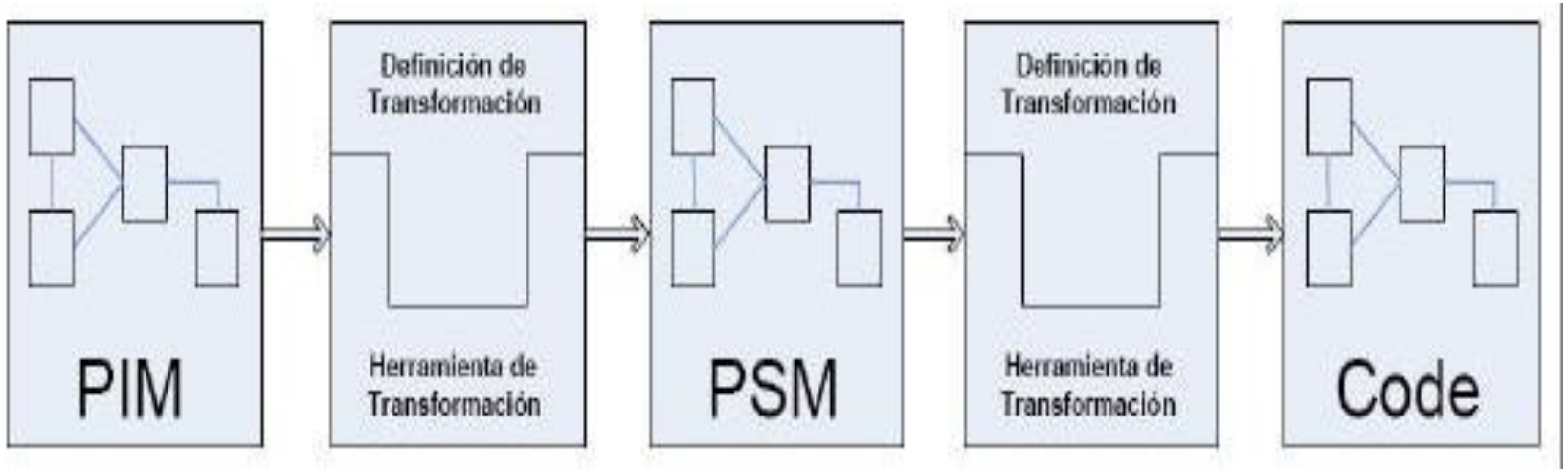
Fuente:

Desarrollo de Software Dirigido por Modelos

»Transformación

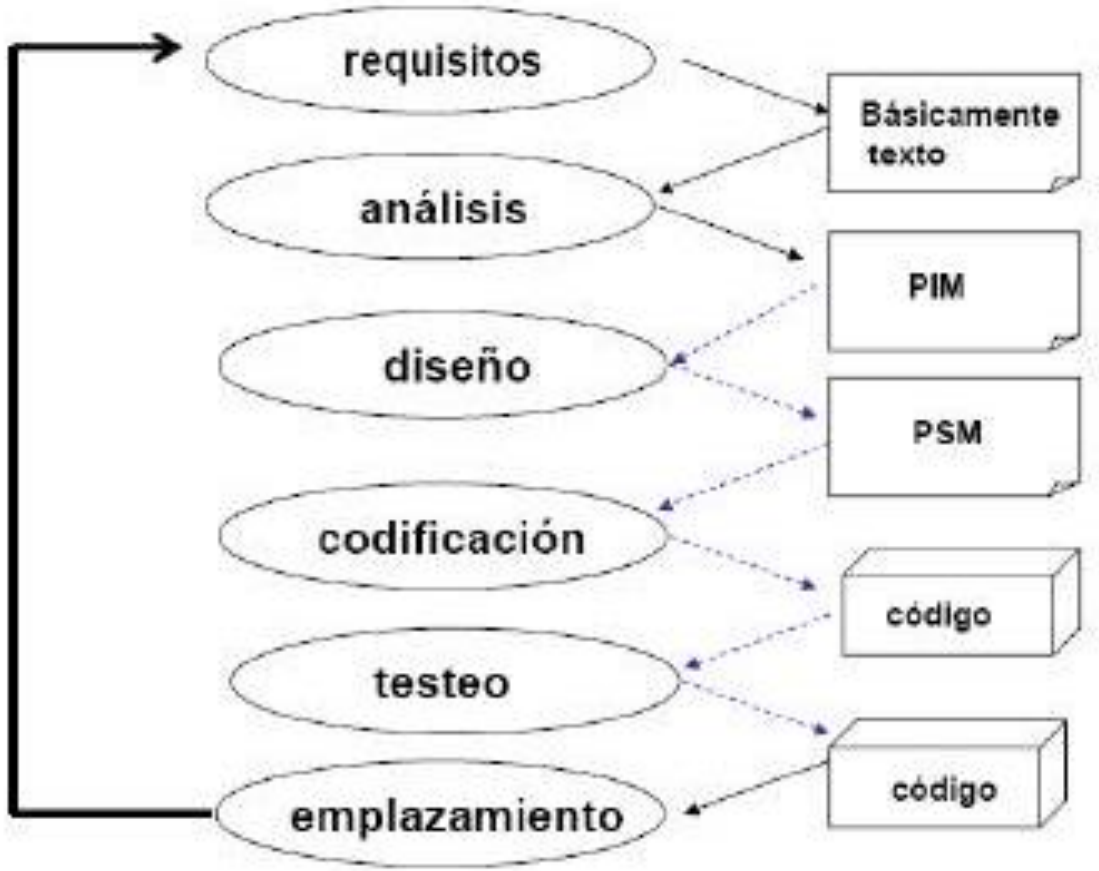
En general, se puede decir que una definición de transformación consiste en una colección de reglas, las cuales son especificaciones no ambiguas de las formas en que un modelo (o parte de él) puede ser usado para crear otro modelo (o parte de él).

El patrón MDD es normalmente utilizado sucesivas veces para producir una sucesión de transformaciones.



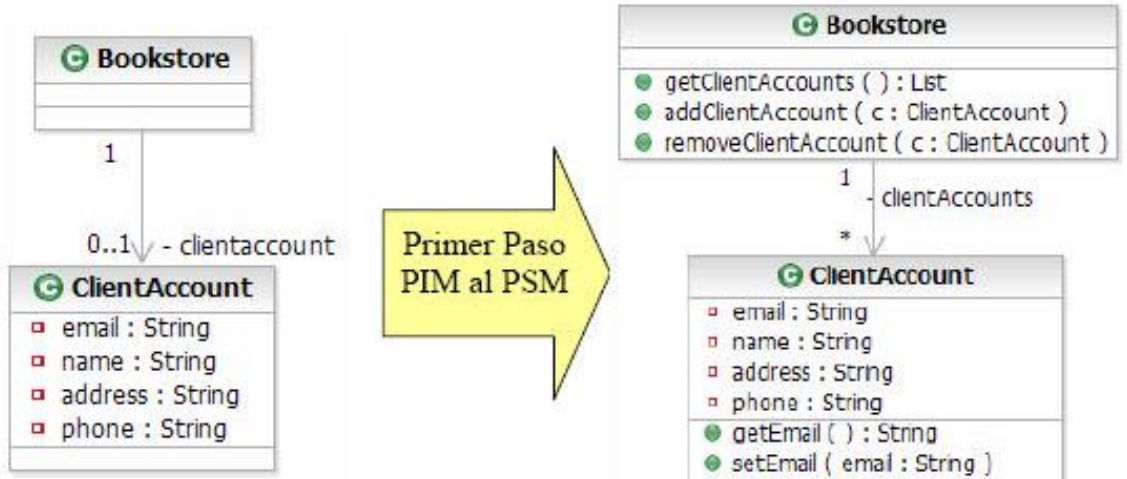
Fuente:

Desarrollo de Software Dirigido por Modelos



Fuente:

Desarrollo de Software Dirigido por Modelos



Codigo JAVA

```
public class Bookstore {
  private List clientAccounts =
          new ArrayList() ;
  public List getClientAccounts(){
                  return clientAccounts ;}
  public void addClientAccount(c : ClientAccount) {
                  clientAccounts.add(c) ;    }
}
        ...
public class ClientAccount {
        ...
```

Segundo Paso
PSM al código

Fuente:

Desarrollo de Software Dirigido por Modelos

»Beneficios de MDD.

- Incremento en la productividad (modelos y transformaciones).
- Adaptación a los cambios tecnológicos.
- Adaptación a los cambios de requisitos
- Consistencia (automatización).
- Re-uso (de modelos y transformaciones).
- Mejoras en la comunicación con los usuarios y la comunicación entre los desarrolladores (los modelos permanecen actualizados).
- Captura de la experiencia (cambio de experto).
- Los modelos son productos de larga duración (resisten cambios).
- Posibilidad de demorar decisiones tecnológicas.

Fuente: