

The background of the slide features a light blue to green gradient. Overlaid on this are faint, large-scale binary digits (0s and 1s) and a network of thin, white, curved lines that suggest a globe or a complex data network.

Ingeniería de Software

Marco Villalobos Abarca

Objetivos del Curso



Iniciar un
proyecto de
software

Planificar un
proyecto de
software

Estimar Costos
de un Proyecto
de Software

Ejecutar un
Proyecto de
Desarrollo de
Software

Controlar un
Proyecto de
Software

INTRODUCCIÓN

- Motivación
- Qué es el Software
- Tipos de Software
- Características o Cualidades del Software
- Qué es la Ingeniería de Software
- Responsabilidad Profesional
- Aspectos Éticos
- Proceso Software

MOTIVACIÓN 1

- La industria del software no ha acabado de salir de la fase artesanal
- Padecemos de “prisa patológica”, que es consecuencia directa de:
 - Desorganización
 - Falta de planificación
- Alta dependencia de los “héroes”
- Dedicamos nuestros esfuerzos de hoy a arreglar lo que se hizo mal ayer



MOTIVACIÓN 2

- El producto (software) es algo intangible y no constreñido por las leyes físicas
- La disciplina, ingeniería del software, es relativamente reciente y muchos de sus conceptos importantes están aún inmaduros
- Carencia de un corpus de conocimiento aceptado mayoritariamente que sirva como fundamentos
- Escasa presión del mercado

MOTIVACIÓN 3

En una organización inmadura:

- Procesos software normalmente improvisados
- Si se han especificado, no se siguen rigurosamente
- Organización reactiva (resolver crisis inmediatas)
- Planes y presupuestos excedidos sistemáticamente, al no estar basados en estimaciones realistas



MOTIVACIÓN 4

En una organización inmadura ...:

- Si hay plazos rígidos, se sacrifican funcionalidad y calidad del producto para satisfacer el plan
- No existen bases objetivas para juzgar la calidad del producto
- Cuando los proyectos está fuera de plan, las revisiones o pruebas se recortan o eliminan



MOTIVACIÓN 5

- El 90% de los proyectos no alcanzan los objetivos

- El 40% fracasan por completo

- El 29% no se entregan nunca



- J. Jesús María Zavala Ruiz, Por Qué Fracasan los Proyectos de Software?; Un Enfoque Organizacional, Universidad Autónoma Metropolitana-Iztapalapa, México, D.F.
- The Standish Group International, Inc, "The Chaos Report", 1994, 1999, 2001.

Qué hacer ?

Artesanía



Ingeniería

Cambio cultural de todos los involucrados!



¿Qué es el software?

- **Pressman:**

- Instrucciones (programas de computadora) que cuando se ejecutan proporcionan la función y el rendimiento deseados
- Estructuras de datos que permiten a los programas manipular adecuadamente la información, y
- Documentos que describen la construcción y uso de programas

- **Sommerville:**

- Programas de computador y documentación asociada
- Los productos de software pueden ser
 - Genéricos: desarrollados para clientes muy diversos
 - Hecho a medida: para un cliente particular de acuerdo a su especificación

Evolución del software

- Hace dos décadas (años 80):
 - Centrados en el hardware, factor principal en el presupuesto
 - Ingeniería del hardware, pero no del software
 - Poca difusión software
 - Software de venta específico para el hardware
 - No se vendían sistemas operativos sin hardware
- Actualmente:
 - El software es el factor principal en el presupuesto
 - Software con alto tiempo de desarrollo, incluso fuera de plazo -> Costos elevados
 - Software entregado a clientes con errores (defectos)
 - Gran difusión del software: Los sistemas operativos se anuncian en prensa y televisión

Evolución del software

Los primeros años (1950-1965 aprox.)

- Proceso por lotes (batch)
- Distribución limitada
- Software a medida

La segunda era (1965-1975 aprox.)

- Sistema multiusuario
- Sistemas Interactivos (HCI)
- Tiempo real
- Bases de Datos
- Productos software independientes del hardware: mantenimiento del software con versiones

Evolución del software

La tercera era (1975-1985 aprox.)

- Sistemas distribuidos
- Incorporación de “inteligencia”
- Hardware de bajo costo (microprocesador)
- Impacto en el consumo

La cuarta era (1985-2000 aprox.)

- Sistemas personales potentes
- Tecnologías orientadas a objetos
- Redes de computadoras
- Computación en paralelo
- Técnicas de inteligencia artificial

Evolución del software

Etapa actual (principios del tercer milenio)

- Componentes y arquitecturas software reutilizables
- Web semántica
- Computación ubicua
- Interfaces multi-modales

Tipos de software

- Software de Sistemas
- Software de Tiempo Real
- Software de Gestión
- Software de Ingeniería y científico
- Software empotrado
- Software de computadoras personales
- Software basado en Web
- Software de Inteligencia Artificial

Software de Sistemas

- Programas que han sido escritos para servir a otros programas
- Se caracterizan por una fuerte interacción con el hardware de la computadora
- Compartir recursos
- Sofisticada gestión de procesos
- Estructuras de datos complejas
- Múltiples interfaces externas

Ejemplo: compiladores, editores y utilidades de gestión de archivos, ciertos componentes del sistema operativo, utilidades de manejo de periféricos, procesadores de telecomunicaciones, etc.

Software de Tiempo Real

- Coordina/analiza/controla sucesos del mundo real conforme ocurren.
- Integra diferentes componentes:
 - Recolección de datos
 - Dar formato a la información recibida del exterior
 - De análisis para transformar la información
 - De Control/salida que responda al exterior
 - De Monitorización que coordine a todos los componentes
 - En un tiempo de entre 1 milisegundo a 1 segundo

Software de Gestión

- El proceso de la información comercial constituye la mayor de las áreas de aplicación del software.
- Han evolucionado hacia el software de sistemas de información de gestión (SIG) que accede a una o más bases de datos que contienen información comercial
Por ejemplo: nóminas, cuentas de haberes-débitos, inventarios, etc.
- Se realizan tareas convencionales de procesamientos de datos

Software de Ingeniería y científico

- Está caracterizado por los algoritmos (conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema) de manejo de números

Ejemplo: Astronomía, vulcanología, análisis de la presión de los automotores, dinámica orbital de las lanzaderas espaciales, biología molecular, fabricación automática.

Pero las nuevas aplicaciones del área de ingeniería se han alejado de los algoritmos convencionales numéricos. El diseño asistido por computadora (del inglés CAD), la simulación de sistemas y otras aplicaciones interactivas, han comenzado a tener características del software de tiempo real e incluso del software de sistemas.

Software empotrado

- Reside en memoria de sólo lectura
- Se utiliza para controlar productos y sistemas de los mercados industriales y de consumo
- Ejecuta funciones muy limitadas

Ejemplo: El control de las teclas de un horno de microondas, control de la gasolina de un automóvil, sistemas de frenado, etc.

Software de computadoras personales

El mercado del software de computadoras personales ha germinado en las pasadas dos décadas. Aplicaciones que pertenecen a esta categoría:

- El procesamiento de textos
- Hojas de cálculo
- Los gráficos por computadora
- Multimedia
- Entretenimientos
- Gestión de bases de datos
- Aplicaciones financieras, de negocios y personales
- Redes o acceso a bases de datos externas

Software basado en Web

- Las páginas Web buscadas por un explorador son software que incorpora instrucciones ejecutables (por ejemplo, HTML, Perl, o Java), y datos (por ejemplo, hipertexto y una variedad de formatos de audio y visuales).
- En esencia, la red viene a ser una gran computadora que proporciona un recurso software casi ilimitado que puede ser accedido por cualquier persona mediante el equipamiento que hoy se proporciona ya incorporado al PC

Software de Inteligencia Artificial

- Este tipo de Software hace uso de algoritmos no numéricos para resolver problemas complejos para los que no son adecuados el cálculo o el análisis directo.

Ejemplos:

Los sistemas expertos, reconocimiento de patrones (imágenes y voz), prueba de teoremas y juegos, robótica.

Características del Software

- Clasificación de las características
- Características

Clasificación de las características

- **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.

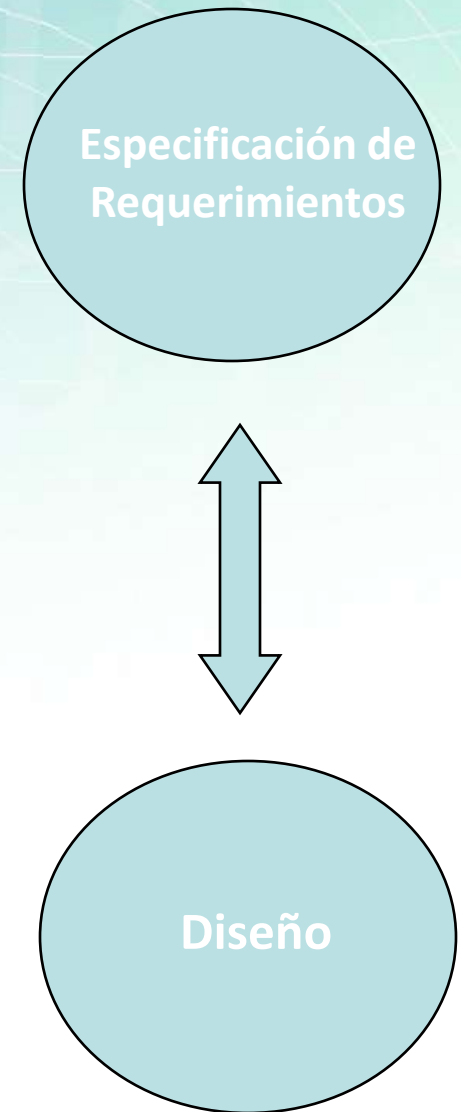
Características del Software

- Correcto (Correctness)
- Confiable (Reliability)
- Robusto (Robustness)
- Eficiente (Efficiency)
- “Amigable” (Friendliness)
- Verificable (Verifiability)
- Reusable (Reusability)
- “Portable” (Portability)
- Interoperable (Interoperability)
- Comprensible (Understandability)
- “Mantenible” (Maintainability)

Correcto

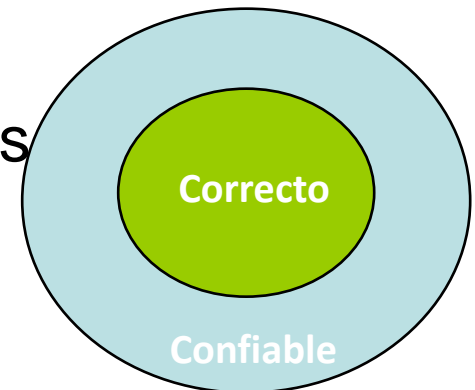
Un software es correcto si se comporta de acuerdo a su especificación

- La definición supone:
 - La existencia es una especificación de requisitos.
 - La posibilidad de determinar sin ambigüedad la correspondencia entre la especificación y el diseño.
- Lo correcto del software puede probarse ejecutándolo o mediante análisis.



Confiable

- El software se comporta de acuerdo con lo esperado por el usuario.
- A diferencia de la corrección, la confiabilidad es algo relativo.
- El mercado puede admitir algunos errores en el software siempre que en general se comporte en forma esperada.
- La confiabilidad es relativa: un SW puede aún ser confiable si la consecuencia de un error no es seria; o si la cantidad de errores por unidad de tiempo no es alta.



Robusto



Un software es robusto si se comporta en forma razonable aún en situaciones no anticipadas.

- Datos de entrada incorrectos o fallas de hardware son las situaciones mas frecuentes.
- La cantidad de código que se dedica a hacer el software robusto depende de la experiencia de los usuarios o lo crítico de su misión.
- Si algo se especifica como requerimiento, cumplirlo es cuestión de corrección; si no está en los requerimientos es cuestión de robustez.

Eficiencia-Performance

Un sistema de software es eficiente si usa sus recursos en forma económica.

- Muy lento ➡ baja la productividad de los usuarios.
- Usa mucho disco ➡ puede ser muy caro ejecutarlo.
- Usa mucha memoria ➡ puede afectar la performance de otros sistemas
- Usualmente es muy difícil mejorar considerablemente el performance sin rediseñar.
- No es bueno evaluar el performance sólo después que el producto esté listo, sino durante todo el proceso.
- Los criterios de eficiencia varían con la tecnología y el tiempo.
- Métodos de evaluación de performance:
 - Monitoreo
 - Análisis
 - Simulación

Amigable

Un software es amigable si sus usuarios lo encuentran fácil de utilizar.

- La interfaz con los usuarios es parte esencial de ser amigable
- Depende de los usuarios:
 - Novicios: lo mejor son largos mensajes explicativos
 - Expertos aprecian los atajos.
- Otros factores importantes para lograr esta cualidad son la facilidad en la configuración y el proceso para realizar una tarea.



Verificable



El software es verificable si sus propiedades pueden ser comprobadas.

- La corrección y el performance pueden verificarse fácilmente
- La verificación puede hacerse mediante el análisis o testing.

“Reusable”/Re-utilizable

Software ya construido se usa con pocos o ningún cambio.



- La reutilización es mas apropiada para componentes que para sistemas completos
- Las bibliotecas (librerías) científicas FORTRAN son los ejemplos más conocidos y Java API's son ejemplos más nuevos.
- Debe apuntarse a ella desde el diseño.
- La reutilización es una cualidad difícil (imposible) de conseguir a posteriori.
- La orientación a objetos tiene potencial para mejorar la reutilización y la evolución.

Portable

Un SW es portable si puede ejecutarse en distintos ambientes (hardware, sistemas operativos, etc.)

- Una forma de lograr portabilidad es suponer la mínima configuración.
- Esto penaliza los sistemas que podrían ejecutarse mejor haciendo uso del ambiente disponible.
- Otra opción es determinar sobre la marcha las disponibilidades del ambiente.

Interoperable



Un sistema es interoperable si puede coexistir y cooperar con otros sistemas.

- Las componentes reutilizables son interoperables.
- La estandarización de las interfaces promueve la interoperabilidad.
- Los sistemas abiertos son casos típicos de sistemas interoperables.

Mantenible



Un sistema es Mantenible si es fácil modificarlo.

- Tipos de mantenimiento:

- * Correctivo (aprox. 20%)

- * Adaptativo (aprox. 20%)

- * Perfectivo (aprox. 50%)

- Software Mantenible:

- * Reparable : que permite corregir defectos,

- * Evolucionable: facilita la introducción de nuevas funcionalidades

- Condiciones

- * Número de componentes,

- * Acoplamiento

- * Documentación:
Completa, Comprensible o al día.

- * Uso de componentes estándar

- La evolucionabilidad decrece con cada versión del software.

FACILIDAD DE MANTENIMIENTO

(¿Puedo arreglarlo?)

FACILIDAD DE PRUEBA

(¿Puedo probarlo?)

FLEXIBILIDAD

(¿Puedo modificarlo?)

INTEROPERABILIDAD

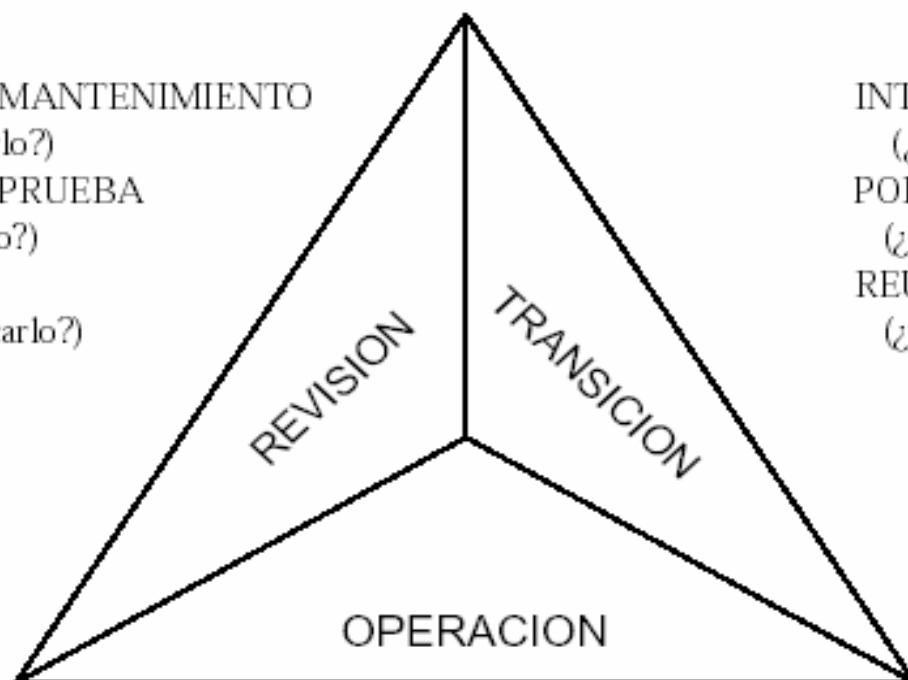
(¿Podré comunicarlo con otros sistemas?)

PORTABILIDAD

(¿Podré utilizarlo en otra máquina?)

REUSABILIDAD

(¿Podré reutilizar parte del software?)



CORRECCION (¿Hace el software lo que yo quiero?)

FIABILIDAD (¿Lo hace de forma exacta todo el tiempo?)

EFICIENCIA (¿Se ejecutará sobre mi hardware lo mejor posible?)

INTEGRIDAD (¿Es seguro?)

FACILIDAD DE USO (¿Puedo ejecutarlo?)

FACTOR	CRITERIOS
Facilidad de uso	Facilidad de operación , Facilidad de comunicación , Facilidad de aprendizaje
Integridad	Control de accesos, Facilidad de auditoria
Corrección	Compleitud, Consistencia, Trazabilidad
Fiabilidad	Precisión, Consistencia, Tolerancia a fallos, Modularidad, Simplicidad
Eficiencia	Eficiencia en ejecución, Eficiencia en almacenamiento
Facilidad de mantenimiento	Modularidad, Simplicidad, Consistencia, Concisión, Auto descripción
Facilidad de prueba	Modularidad, Simplicidad, Auto descripción, Instrumentación
Flexibilidad	Auto descripción, Capacidad de expansión, Generalidad, Modularidad
Reusabilidad	Auto descripción, Generalidad, Modularidad, Independencia entre sistema y software, Independencia del hardware
Interoperabilidad	Modularidad, Compatibilidad de comunicaciones, Compatibilidad de datos
Portabilidad	Auto descripción, Modularidad, Independencia entre sistema y software, Independencia del hardware

QUÉ ES INGENIERÍA DE SOFTWARE

- ✓ una disciplina de ingeniería
 - Aplicación de teorías, métodos, herramientas para hacer cosas que funcionen: Software que sea fiable y trabaje en máquinas reales
 - Teniendo en cuenta restricciones financieras, organizacionales y técnicas
- ✓ que comprende todos los aspectos de la producción de software
 - Desde la especificación inicial al mantenimiento del sistema
 - Administración y gestión del proceso de producción: Principios y metodologías para desarrollo y mantenimiento de sistemas de software

QUÉ ES INGENIERÍA DE SOFTWARE

- Ingeniería de Software es el estudio de los principios y metodologías para desarrollo y mantenimiento de sistemas de software (Zelkovitz, 1978)
- Ingeniería de Software es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora y la documentación asociada requerida para desarrollar, operar (funcionar) y mantenerlos (Bohem, 1976)
- Ingeniería de Software trata del establecimiento de los principios y métodos de la ingeniería a fin de obtener de modo rentable que sea fiable y trabaje en máquinas reales (Bauer, 1972)
- La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento del software; es decir, la aplicación de ingeniería al software (IEEE, 1993)

QUÉ ES INGENIERÍA DE SOFTWARE

Ingeniería de Software y Ciencia de la Computación

- La Ciencia de la Computación se refiere a las teorías y los fundamentos subyacentes en los sistemas de computación
- La Ingeniería del Software trata los problemas prácticos del desarrollo de software
- Con las teorías de la ciencia de la computación no es suficiente para desarrollar software (al menos cuando el sistema tiene suficiente envergadura)

QUÉ ES INGENIERÍA DE SOFTWARE

Ingeniería de Software e Ingeniería de Sistemas

- Ingeniería de Sistemas se refiere a todos los aspectos del desarrollo de sistemas basados en computadora, tanto del hardware como del software y los procesos de diseño y distribución de sistemas
 - La Ingeniería de Software es solo parte de este proceso
 - Los ingenieros de sistemas se encargan de especificar el sistema, definir su arquitectura, integrar sus partes. Están menos relacionados con la ingeniería de los componentes del sistema (hw y sw)
- Al ser el software muchas veces la parte más importante del sistema, las técnicas de ingeniería del software se aplican en el proceso de ingeniería de sistemas

QUÉ ES INGENIERÍA DE SOFTWARE

Relevancia de la IS

- Las economías de TODOS los países desarrollados dependen en gran medida del software
- Cada vez más sistemas son controlados por software
 - Comunicaciones, Seguridad, Administración, Fábricas, Comercio, Agricultura, Etc.
- El gasto en La Ingeniería de Software, representa un alto porcentaje del PIB de los países desarrollados

QUÉ ES INGENIERÍA DE SOFTWARE

Retos de la IS

- Sistemas heredados (legacy systems): Mantenimiento, actualización, integración
- Heterogeneidad (sw y hw) de sistemas distribuidos: Integración y evolución
- Tiempos de desarrollo cada vez más cortos: con menos recursos, proyectos web: 3 meses–3 personas–3 kilos
- Modas: Métodos, lenguajes, ...
- Cultura de ingeniería
- Formalidad: Existe una gran demanda de que exista formalidad en el proceso de desarrollo de software

Responsabilidad profesional

- Los Ingenieros de software no solo deben considerar aspectos técnicos. Deben tener una visión mas amplia, en lo ético, social y profesional.
- No existe estatutos para ninguno de estos aspectos.
 - Desarrollo de sistemas militares.
 - Piratería.
 - Que es mejor para la profesión de Ingeniero de Software.

Aspectos Éticos

- Confidencialidad.
- Competencia.
- Derechos de propiedad intelectual.
- Mal uso de la computadora.

INTRODUCCIÓN

- Motivación
- Qué es el Software
- Tipos de Software
- Características o Cualidades del Software
- Qué es la Ingeniería de Software
- Responsabilidad Profesional
- Aspectos Éticos
- Proceso Software

PROCESO SOFTWARE

INTRODUCCIÓN (1)

- La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento del software; es decir, la aplicación de ingeniería al software (IEEE,1993)
- Ingeniería del Software, Problemática actual:
 - El desarrollo y mantenimiento de software es un **trabajo altamente complejo**.
 - Los proyectos software son **difíciles de gestionar**.
 - La tecnología de **Proceso Software(PS)** intenta **simplificar la gestión** de proyectos software.

PROCESO SOFTWARE

CONCEPTOS BÁSICOS (1)

Proceso de negocio:

- Definición (Sharp, 2001):
 - Colección de tareas de trabajo interrelacionadas, iniciadas en respuesta a un evento, que permiten alcanzar un resultado específico para el cliente del proceso.
 - Es decir, un proceso de negocio (PN) es un proceso para entregar un resultado a un cliente.
- Algunas características son:
 - Medible: debe poderse medir el PN en la forma que interese a los participantes (stakeholders).
 - Automatizable: las tareas pueden ser manuales, semiautomáticas y automáticas.
 - Niveles: se puede definir a distintos niveles de detalle (hitos, flujos de trabajo, etc.).
 - Para un cliente interno o externo.

PROCESO SOFTWARE

CONCEPTOS BÁSICOS (2)

Proceso Software:

- Definición (Fugetta, 2000)
 - Un Proceso Software (PS) es un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos que son necesarios para concebir, desarrollar, instalar y mantener un producto software.
- Un PS aprovecha diversas contribuciones y conceptos:
 - Tecnologías de desarrollo de software: herramientas, infraestructuras y entornos.
 - Métodos y Técnicas de desarrollo de software: cómo usar la tecnología.
 - Comportamiento Organizacional: la ciencia de las organizaciones y la personas.
 - Marketing y economía: como cualquier otro producto, el software debe dirigirse a clientes reales y, por tanto, está sujeto a las reglas del mercado.

PROCESO SOFTWARE

CONCEPTOS BÁSICOS (3)

Proceso Software vs Proceso de Negocio

- Por tanto, debemos prestar atención a la compleja interrelación que se produce en un PS entre los diversos factores organizacionales, culturales, tecnológicos y económicos.
- Un PS es un PN realizado por una organización para desarrollar y mantener un producto software.

PROCESO SOFTWARE

Naturaleza

- **Son complejos.**
- **No son procesos de producción:**
 - Dirigidos por excepciones,
 - Muy determinados por circunstancias impredecibles,
 - Cada uno con sus peculiaridades.
- **No son procesos de ingeniería “pura”:**
 - Desconocemos las abstracciones adecuadas,
 - Dependen demasiado de demasiada gente,
 - Diseño y producción no están claramente separados,
 - Presupuestos, calendarios, calidad no pueden ser planificados de forma fiable.
- **Están basados en descubrimientos que dependen de la comunicación, coordinación y cooperación dentro de marcos de trabajo predefinidos:**
 - Los entregables generan nuevos requisitos,
 - El éxito depende de la implicación del usuario y de la coordinación de muchos roles (ventas, desarrollo técnico, cliente, etc.).

PROCESO SOFTWARE

Gestión

- **Calidad enfocada al proceso:**
 - Los Procesos Software tienen una influencia directa en la calidad de los productos software:
 - Creciente interés en las empresas software para promover la mejora de los procesos software a la hora de mejorar la calidad de los productos
 - Las aplicaciones software son productos cada vez más complejos lo que influye en la complejidad de sus procesos de desarrollo y mantenimiento
- **Requisitos de calidad de los procesos software:**
 - Producir los resultados esperados
 - Correcta definición
 - Ser mejorados en función de los objetivos de negocio

PROCESO SOFTWARE

Perspectiva Histórica

El Proceso como una caja negra (Sólo visibilidad de las entradas y salidas)

- No hay forma de controlar lo que sucede en el proceso
- Consecuencias:
 - Grandes retrasos en la entrega
 - Incremento significativo de los costos
- En PS la no visibilidad del proceso tiene consecuencias aun peores respecto a otros tipos de procesos dada la naturaleza del software:
 - Especificación informal de requisitos
 - Variabilidad de los requisitos
- Consecuencias:
 - El producto no coincide con los requisitos especificados
 - Los costos de modificación del producto una vez desarrollado son muy altos
- Necesidad de un Proceso Visible (Transparente)



PROCESO SOFTWARE

Perspectiva Histórica

- 1. Ciclos de Vida del Software**
- 2. Metodologías**
- 3. Desarrollo Formal**
- 4. Automatización**
- 5. Gestión y Mejora**
- 6. Programación del Proceso**

PROCESO SOFTWARE

Perspectiva Histórica

1. Ciclos de Vida del Software:

- Años 60
- Definen la vida de un producto desde su concepción hasta la finalización de su uso
- Un Modelo de Ciclo de Vida estandarizan la descomposición de un proceso en fases, actividades y los artefactos que fluyen entre fases
- Ejemplo: CV en cascada

PROCESO SOFTWARE

Perspectiva Histórica

2. Metodologías:

- Inicio Años 60, 70 con propuestas de metodologías de desarrollo estructurado
- Guía de actividades, procedimientos, productos, técnicas, participantes, etc..
- Incluyen buenas prácticas basadas en la experiencia
- Algunos inconvenientes:
 - Algunas metodologías fueron aplicadas en contextos diferentes a aquellos en los que se obtuvo la experiencia sobre buenas prácticas de desarrollo
 - En ocasiones implicaban una documentación pesada y falta de soporte automático
 - Notaciones informales implican dificultad para evaluar la corrección/consistencia de los productos

PROCESO SOFTWARE

Perspectiva Histórica

3. Desarrollo Formal:

- Finales 60
- Enfoque de desarrollo de software basado en las matemáticas
- Los programas son entidades matemáticas que pueden ser especificadas formalmente:
 - Transformación automática de Especificación Formal a Código
- Como solución general para el problema del software presentaba en sus orígenes algunos inconvenientes:
 - Imposibilidad de conocer todos los requisitos al comienzo
 - Escalabilidad
 - Requisitos no funcionales

PROCESO SOFTWARE

Perspectiva Histórica

4. Automatización (Fuggetta,1993):

- Años 70, 80
- La mejora de la tecnología (HW, SW) facilita la aparición de:
 - Software Development Environments (SDE)
 - Lenguajes de Cuarta Generación
- Este tipo de entornos facilitaba la tarea de programación
- Inconveniente:
 - La programación no es la única actividad involucrada en el proceso de desarrollo/mantenimiento de software

PROCESO SOFTWARE

Perspectiva Histórica

5. Gestión y Mejora:

- Años 80
- Creciente Importancia en la industria Software por la calidad
- Aparecen estándares como la familia ISO 9000 y modelos de madurez como CMM (finales de los 80)
- Estándares ISO 9000
 - Certificación Calidad implica Garantía de que una organización software entregará productos de calidad
- Estos estándares y modelos incluyen prácticas que facilitan la gestión de los procesos software
- Aparecen ciertas limitaciones:
 - ¿una organización con certificación de calidad obtendrá siempre productos de alta calidad?
 - Incremento de Burocracia

PROCESO SOFTWARE

Perspectiva Histórica

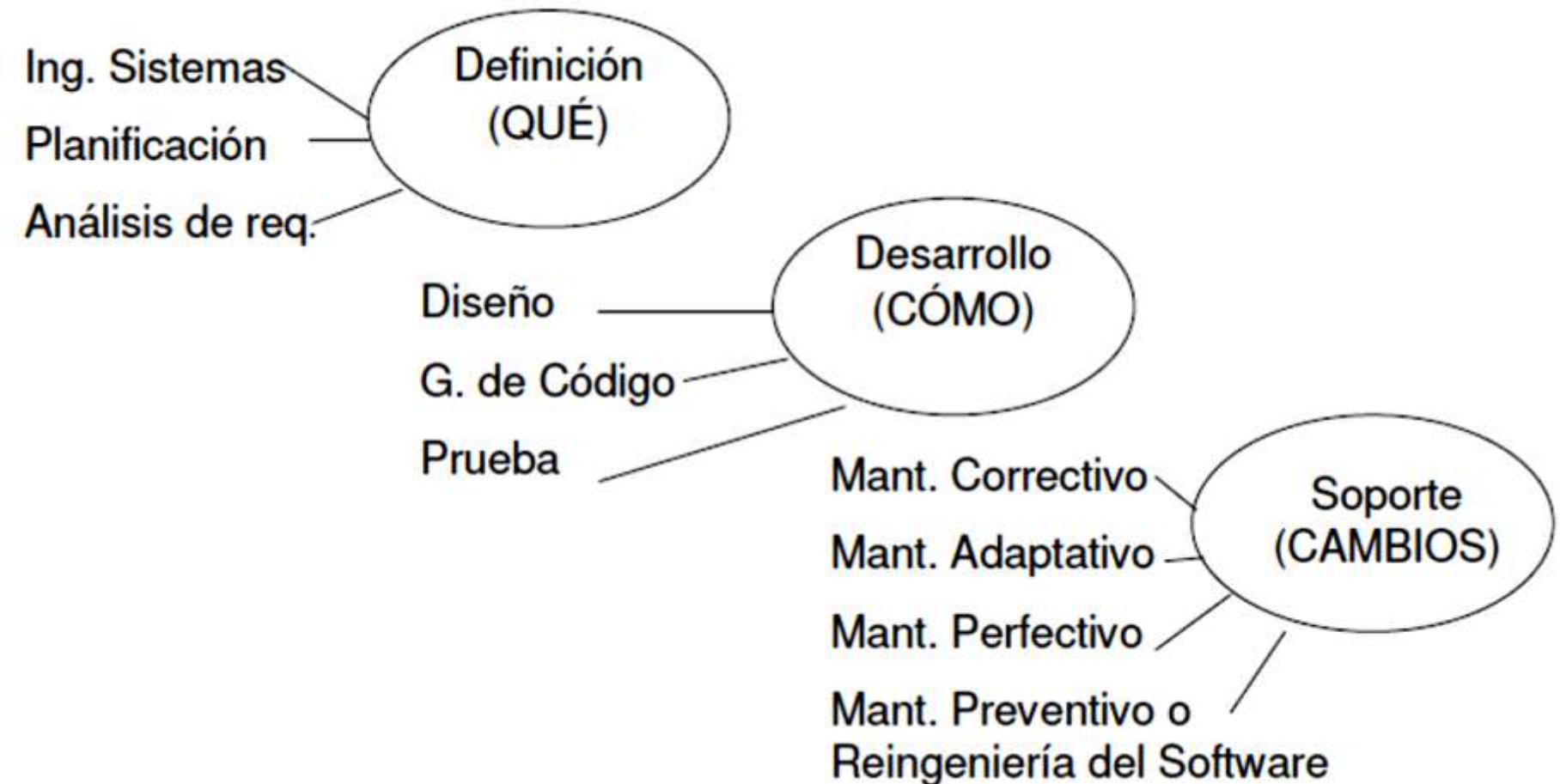
6. Programación del Proceso

- Años 90
- “Software Processes are software too”(Osterweil, 1987)
- Cada organización software es diferente (cultura, habilidades, personal, productos..). Incluso en la misma organización los distintos proyectos pueden ser muy diferentes
 - No hay un único proceso de desarrollo de software
- Necesidad de lenguajes para describir los procesos implica Modelos de Procesos:
 - Verificables
 - Ejecutables
- Entornos para dar soporte al desarrollo, documentación, análisis ejecución y evolución de modelos de procesos
 - PSEE: Entornos de Ingeniería del Software Orientados al Proceso

Ciclos de Vida

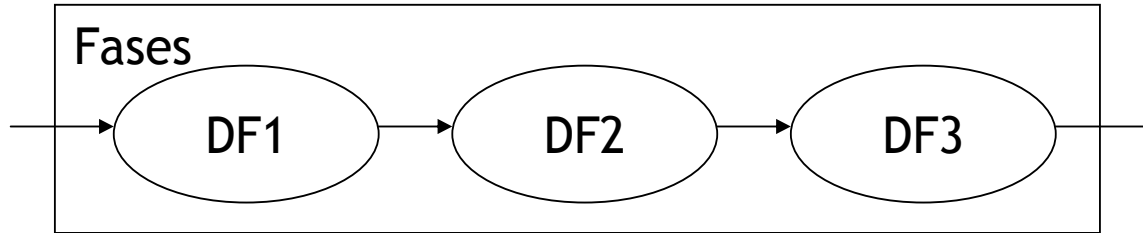
- Una Mirada Genérica
- Secuencial (Lineal, Cascada....)
- Iterativo (Evolutivo, Incremental)
- Espiral

Mirada Genérica (1)

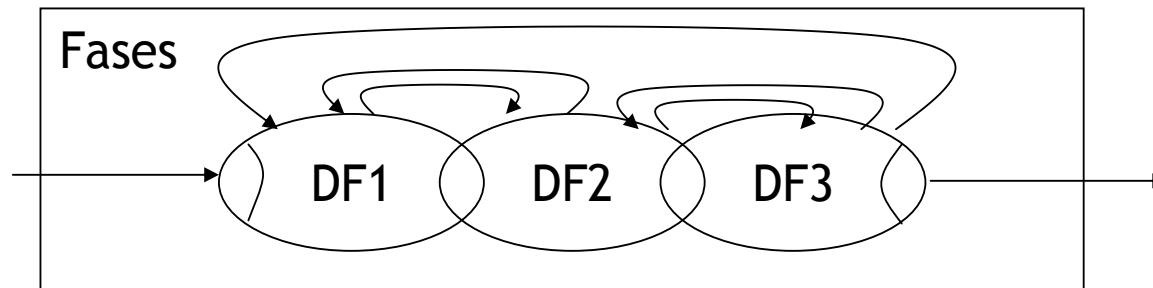


Mirada Genérica (2)

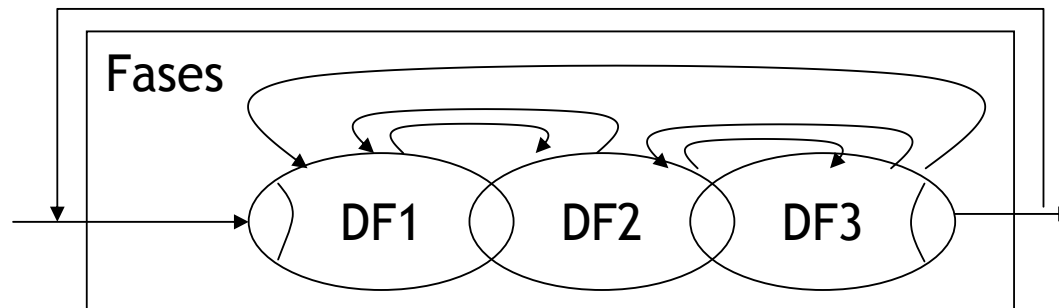
Secuencial (no existe {Interacción, Intersección, retorno})



Iterativo (existe {Interacción, Intersección, retorno})



Recurso (existe {Interacción, Intersección, retorno, re-aplicación})



Secuencial - Lineal

Análisis

Diseño

Implementación

Debugging

Instalación

Aceptación

Secuencial - Lineal

- El proyecto se descompone en etapas separadas que son realizadas de manera lineal y secuencial.
- Enfatiza el cumplimiento de una fase del desarrollo antes de pasar a la fase siguiente.
- Las actividades de cada etapa son independientes entre sí. No hay retroalimentación entre ellas.
- Congela los productos de una fase antes de pasar a la siguiente
- Cada actividad genera entradas y documentación para la siguiente.

Secuencial - Lineal

- ***Fortalezas:***

- Es el más sencillo de todos.
- Menos costos en gestión.

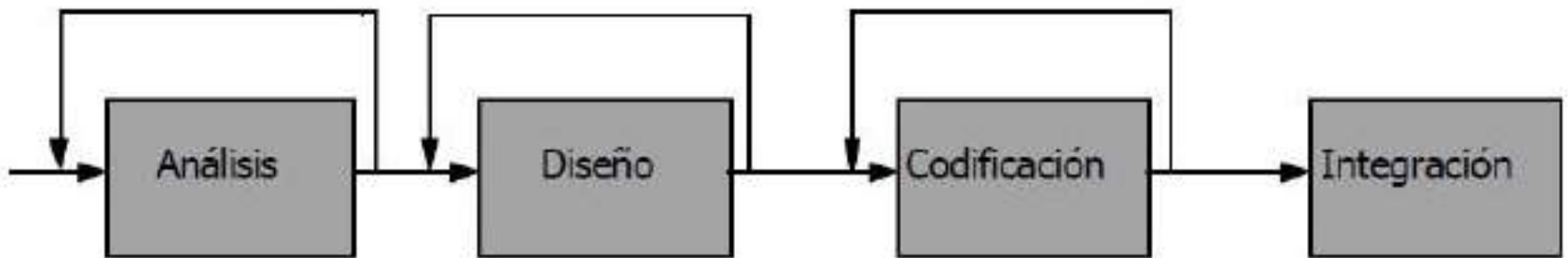
- ***Debilidades***

- Proyectos reales raras veces se ajustan.
- Raras veces el cliente expone todos los req. al inicio.
- No existe retroalimentación, costoso ante cambios.
- Errores detectados tardíamente.
- Producto operativo al final.
- Paciencia (cliente) alta.
- Todo o nada.

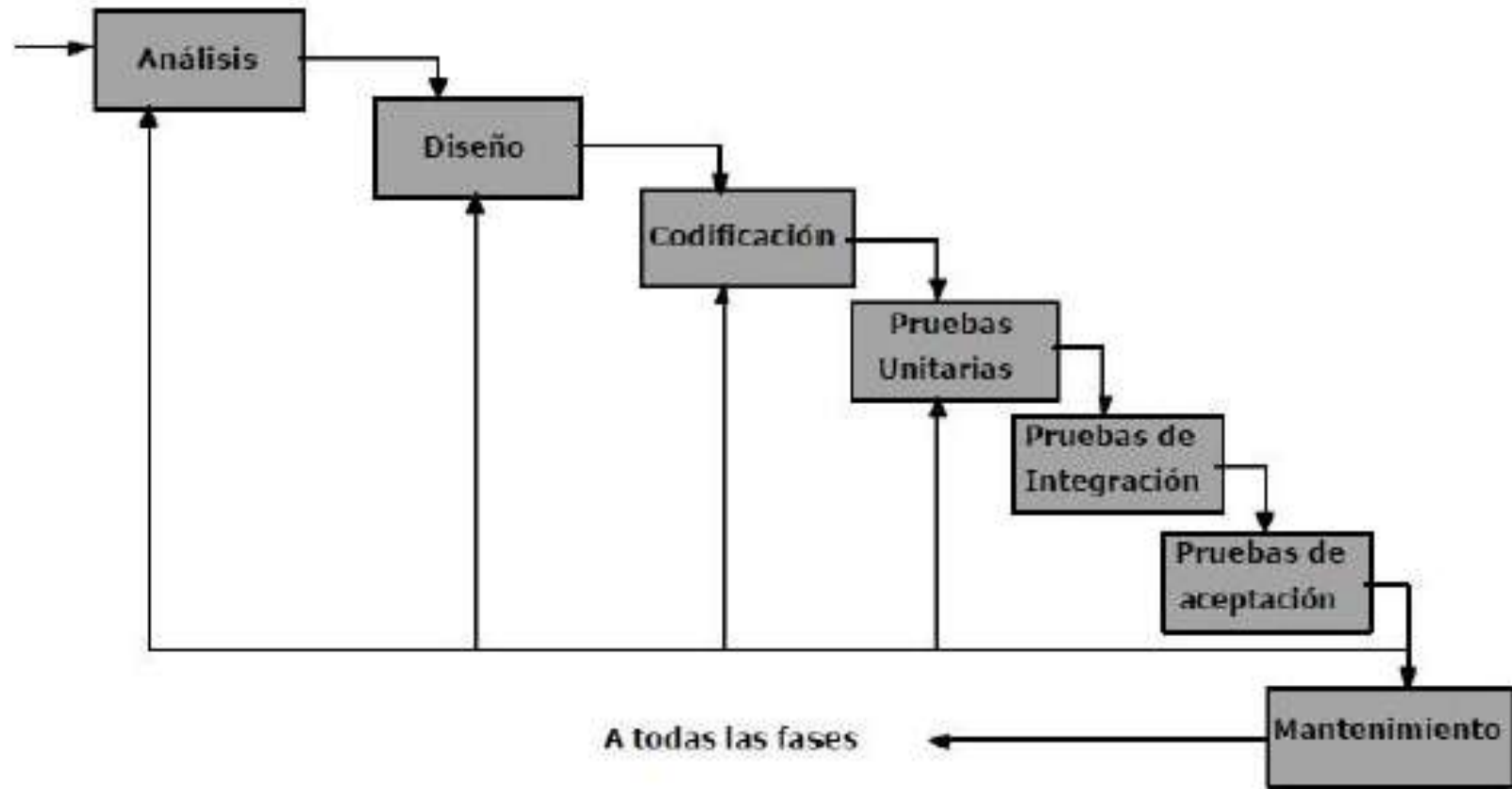
Secuencial - Lineal

- *Dominio de Aplicación*
 - Cuando todos los requerimientos han sido establecidos claramente al inicio o la organización está familiarizada con el dominio.
 - En proyectos pequeños y simples.

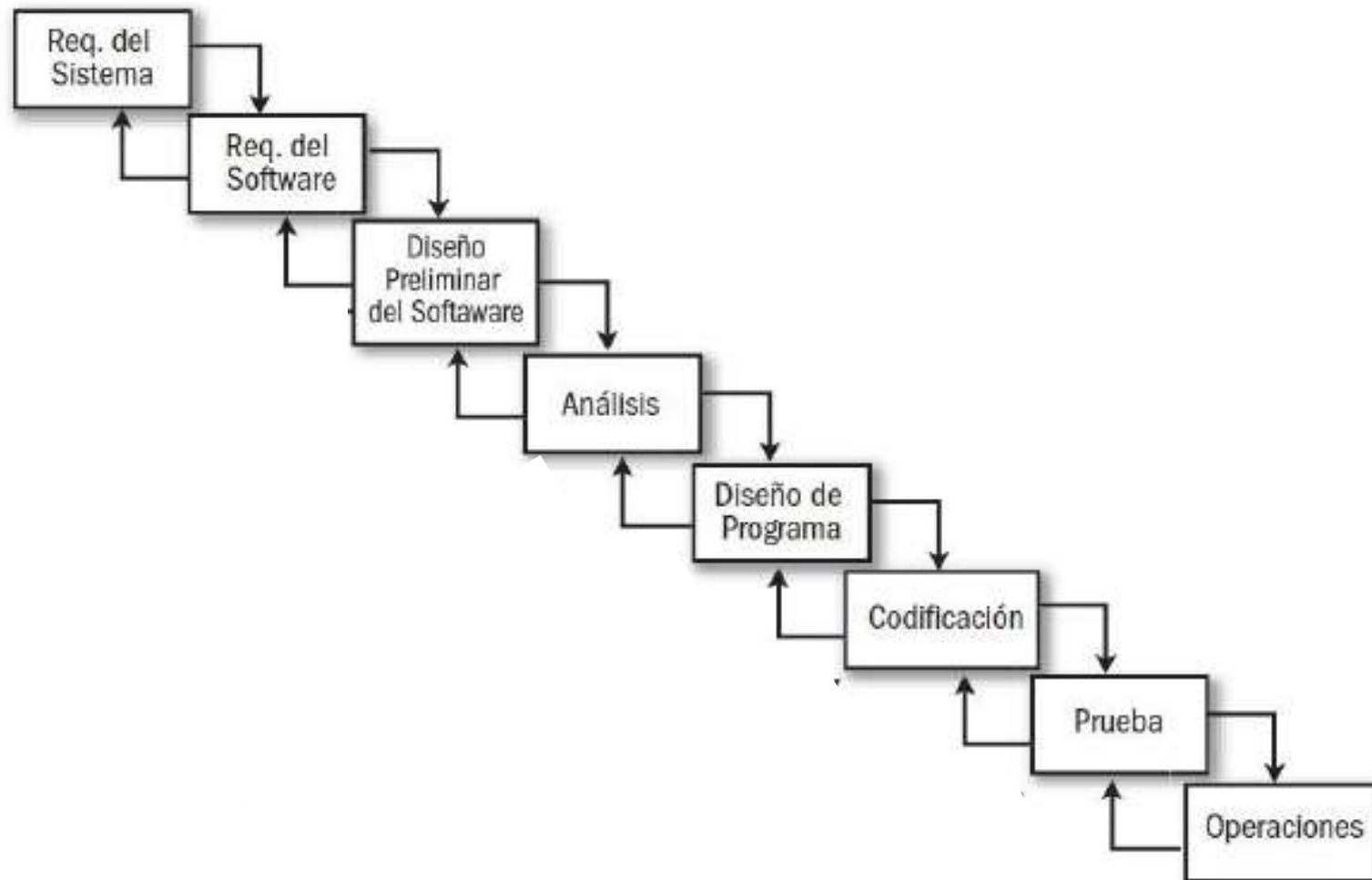
Secuencial – Lineal (con retroalimentación en cada etapa)



Secuencial – Lineal (con retroalimentación desde el final)



Secuencial-Cascada



Secuencial-Cascada

- Admite iteraciones.
- Después de cada etapa se realiza una o varias revisiones para verificar si se puede pasar a la siguiente.
- Fue uno de los primeros modelos, sirvió de base para muchos otros ciclos.
- Planificación muy sencilla.
- Necesita usar un mecanismo formal de cambios para hacer modificaciones a los requerimientos.

Secuencial-Cascada

- *Fortalezas:*
 - Es un modelo que ya está normado en otras disciplinas, lo que hace más fácil su entendimiento
 - Cada fase está definida como un conjunto de funciones, metas, hitos y entregables, haciendo el proceso altamente visible y que el proyecto sea de fácil seguimiento
 - Dado que los requerimientos y especificaciones son determinadas en el lineamiento general, el encargado del proyecto está mejor preparado para determinar los recursos necesarios y establecer la planificación.

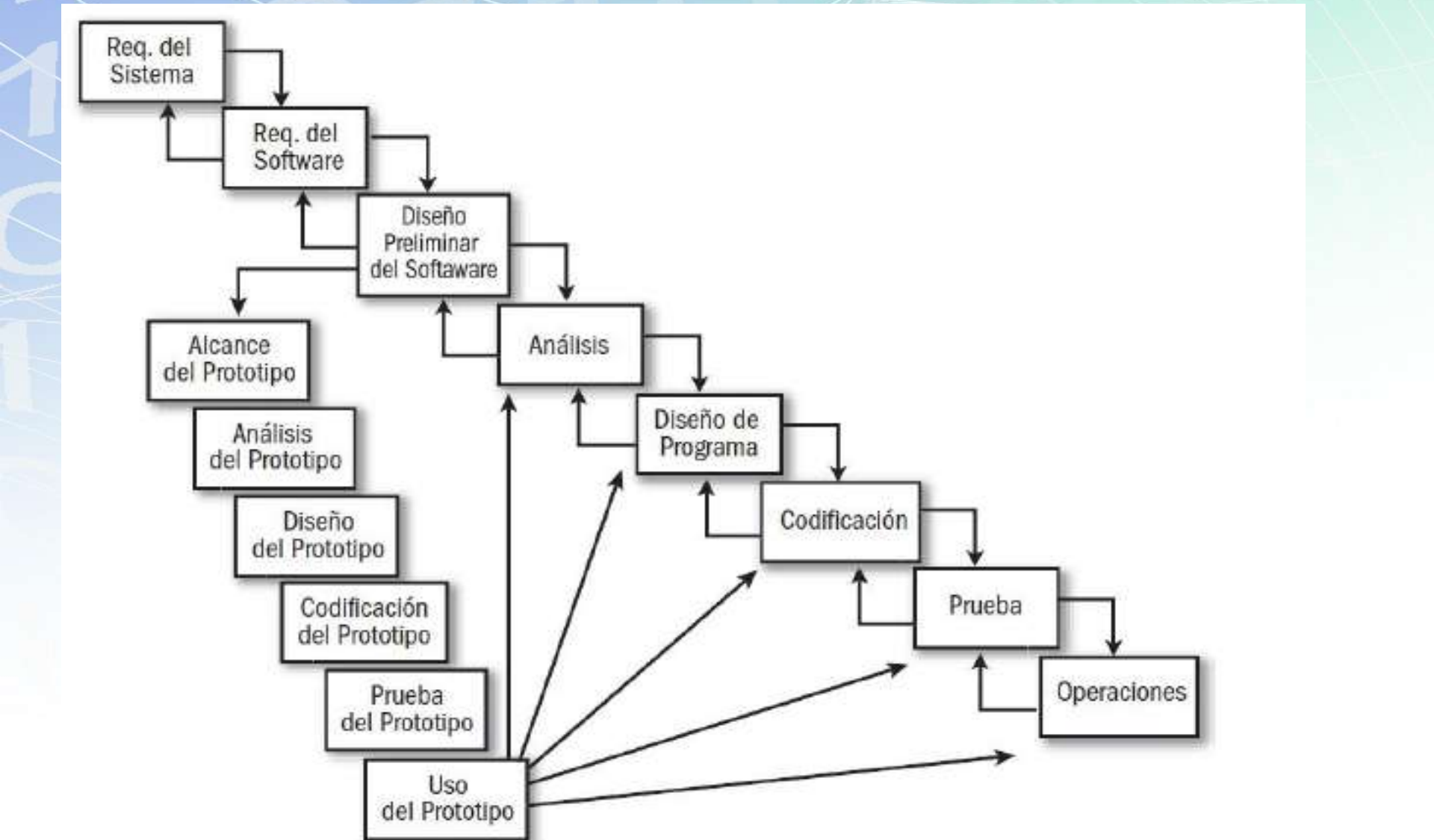
Secuencial-Cascada

- *Debilidades:*
 - No funciona bien en situaciones donde los requerimientos no están bien definidos al principio del proceso
 - La mayor debilidad del modelo es el costo de los cambios de requerimientos. Mientras más avanzado esté el proyecto, más costosos se hacen los cambios en los requerimientos.
 - Los clientes no ven productos funcionales hasta fases avanzadas en el ciclo de vida. Cuando éstos tienen la oportunidad de ver los productos, los costos de corrección ya son demasiado altos de corregir.
 - Los proyectos rara vez fluyen de manera secuencial.

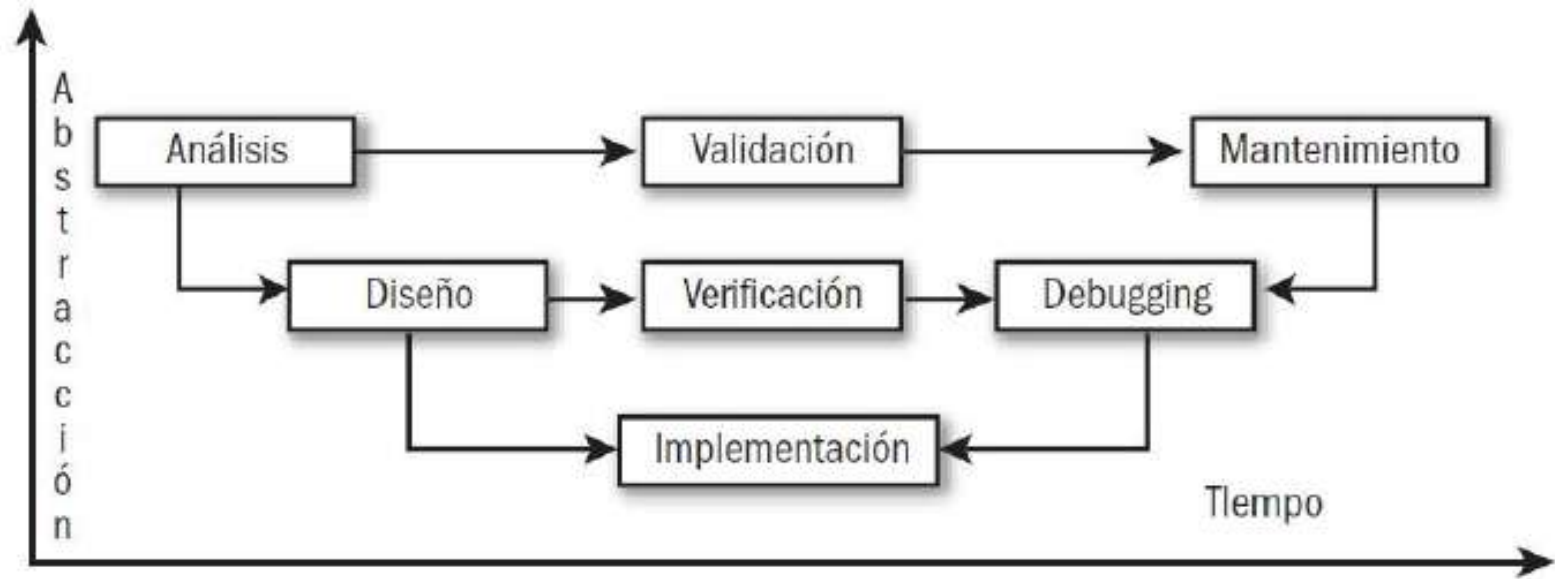
Secuencial-Cascada

- *Dominio de Aplicación*
 - Programas que tienen requerimientos bien definidos al principio del proyecto y programas donde los costos y las planificaciones necesitan ser determinadas desde un principio.
 - La organización está familiarizada con el dominio.

Secuencial-Cascada



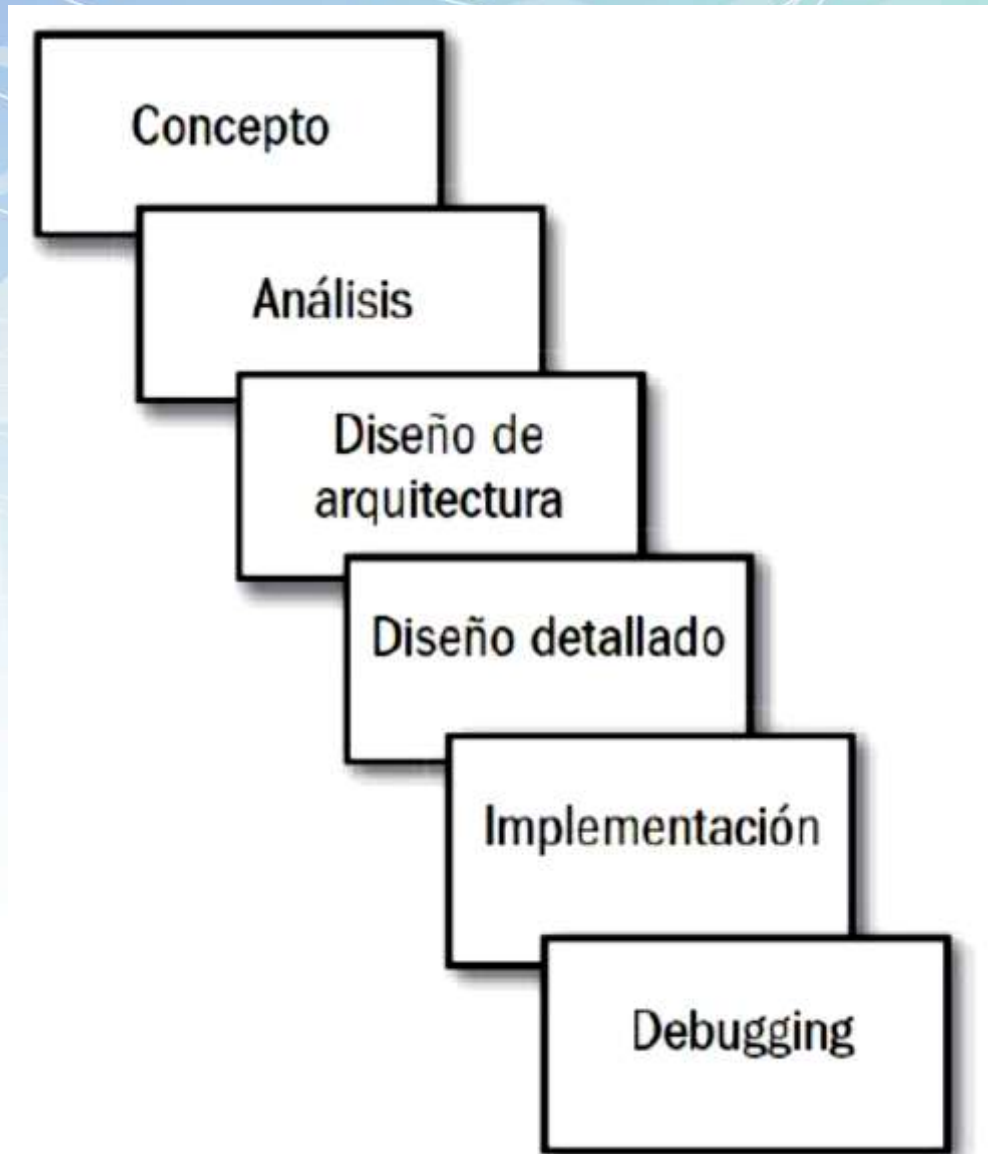
Secuencial-Cascada V&V



Secuencial-Cascada V&V

- Contiene las mismas etapas del ciclo de vida en cascada puro. Pero, se agregan dos etapas de retroalimentación: validación y verificación.
- Tiene las mismas ventajas y desventajas del ciclo anterior.
- Cuenta con el plus de contar con controles cruzados entre etapas para lograr mayor corrección.

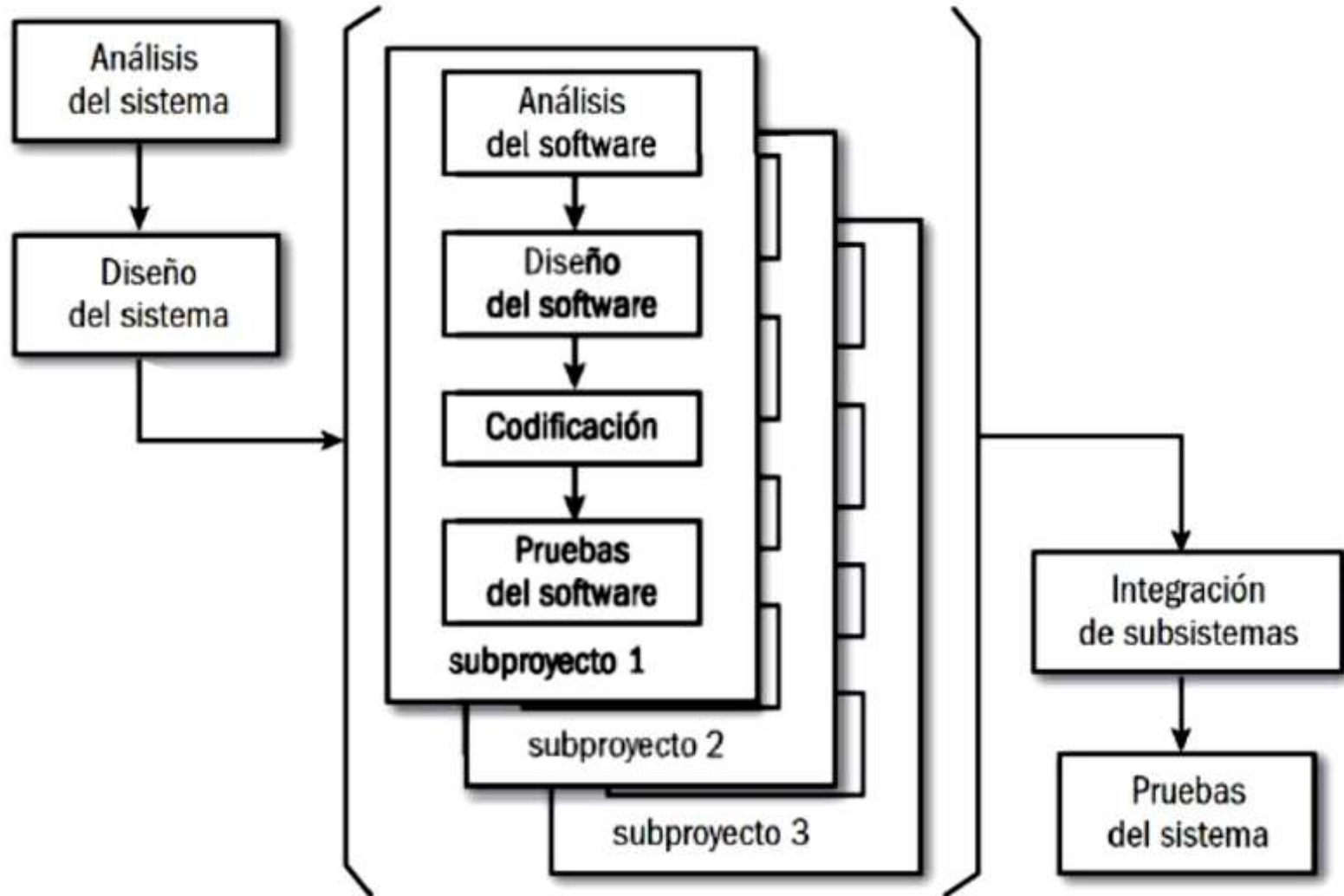
Secuencial-Cascada Sashimi



Secuencial-Cascada Sashimi

- Contiene las mismas etapas del ciclo de vida en cascada puro. Pero, en éste se pueden solapar las etapas.
- La retroalimentación entre etapas está implícita en el modelo, lo que suele aumentar la eficiencia.
- Debido al solapamiento, es difícil gestionar al inicio y fin de cada etapa. Puede producirse problemas de comunicación haciendo inconsistente el proyecto.

Secuencial-Cascada con Subproyectos

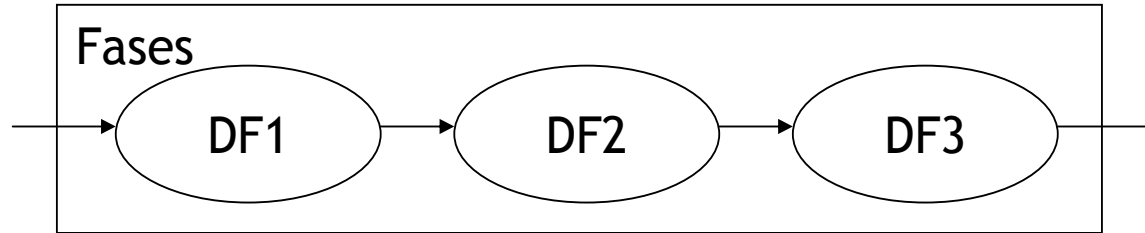


Secuencial-Cascada con Subproyectos

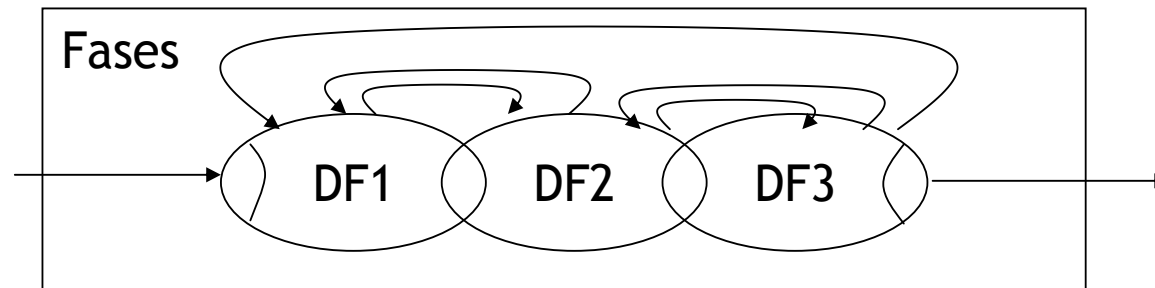
- Sigue el modelo de ciclo de vida en cascada puro. Pero, cada una de las cascadas se dividen en subetapas independientes que se pueden desarrollar en paralelo.
- La ventaja es que se puede contar con más gente trabajando al mismo tiempo.
- La desventaja es que pueden surgir dependencias entre las distintas subetapas y que afecten el proyecto.

Mirada Genérica (2)

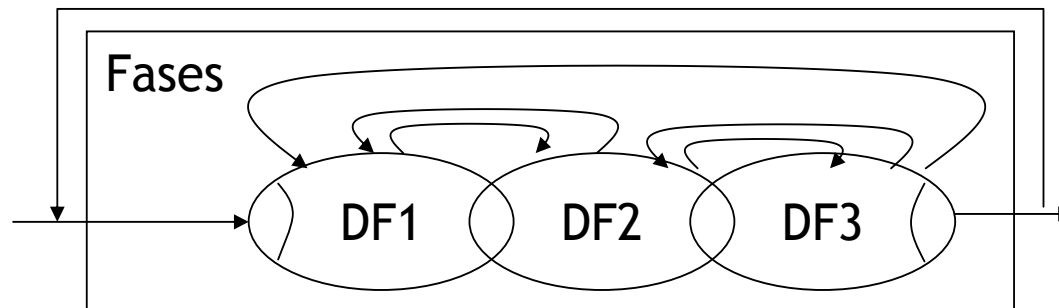
Secuencial (no existe {Interacción, Intersección, retorno})



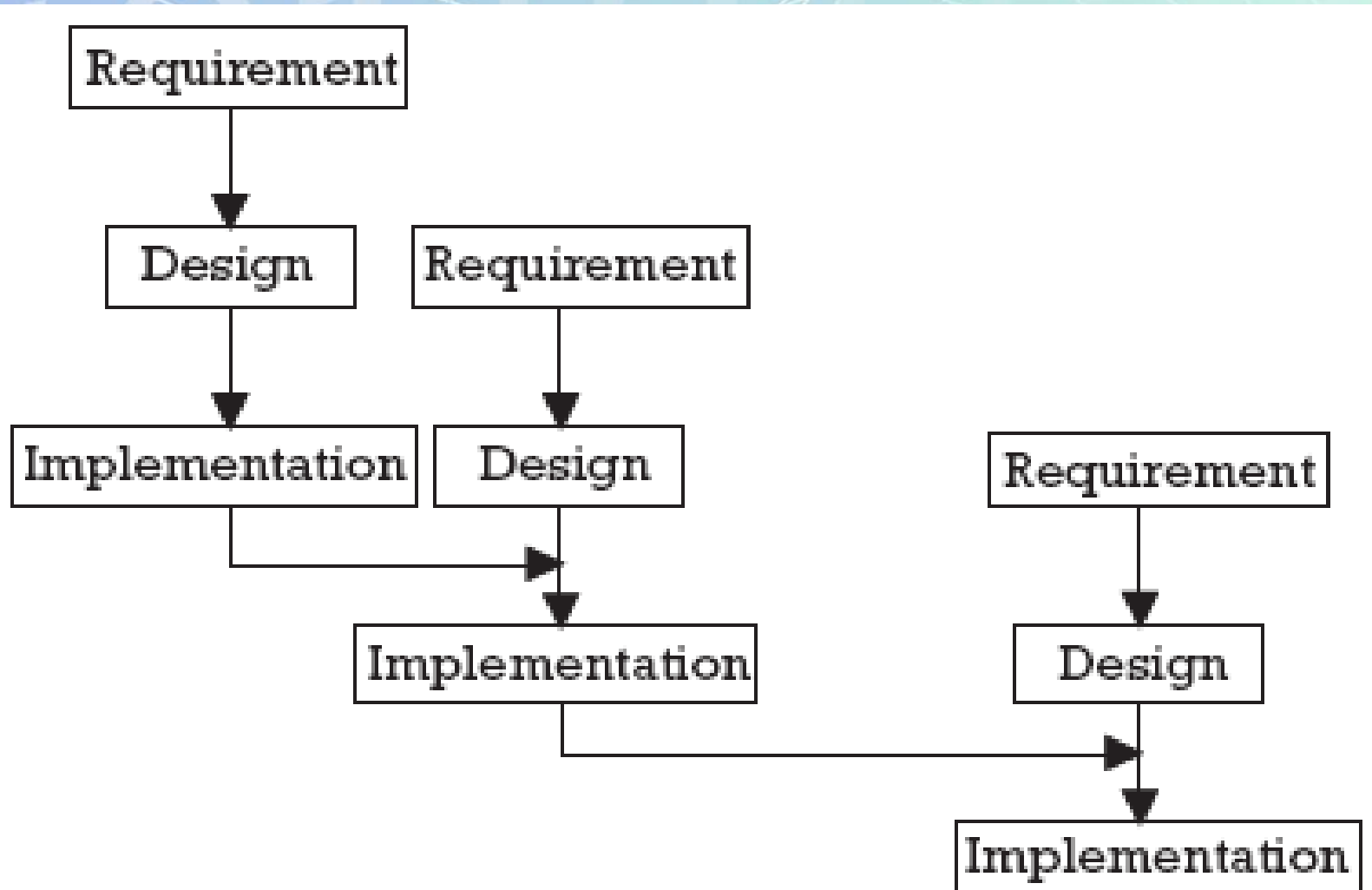
Iterativo (existe {Interacción, Intersección, retorno})



Recurso (existe {Interacción, Intersección, retorno, re-aplicación})



Iterativo - Incremental



Iterativo - Incremental

- Es básicamente el mismo proceso que se realiza en el modelo cascada, pero desarrollándose en secciones que se entrecruzan
- Se basa en la filosofía de construir incrementando las funcionalidades del programa.
- Se realiza construyendo por módulos que cumplen las diferentes funciones del sistema.
- Aumenta gradualmente las funcionalidades.
- Es una repetición del ciclo de vida en cascada en cada funcionalidad.
- Reduce los riesgos ya que va construyendo partes del sistema adoptando este modelo.

Iterativo - Incremental

- *Fortalezas:*
 - Los requerimientos no necesitan ser totalmente especificados/clarificados desde el principio
 - Para cada incremento que se completa, los requerimientos son clarificados

Iterativo - Incremental

- *Debilidades:*

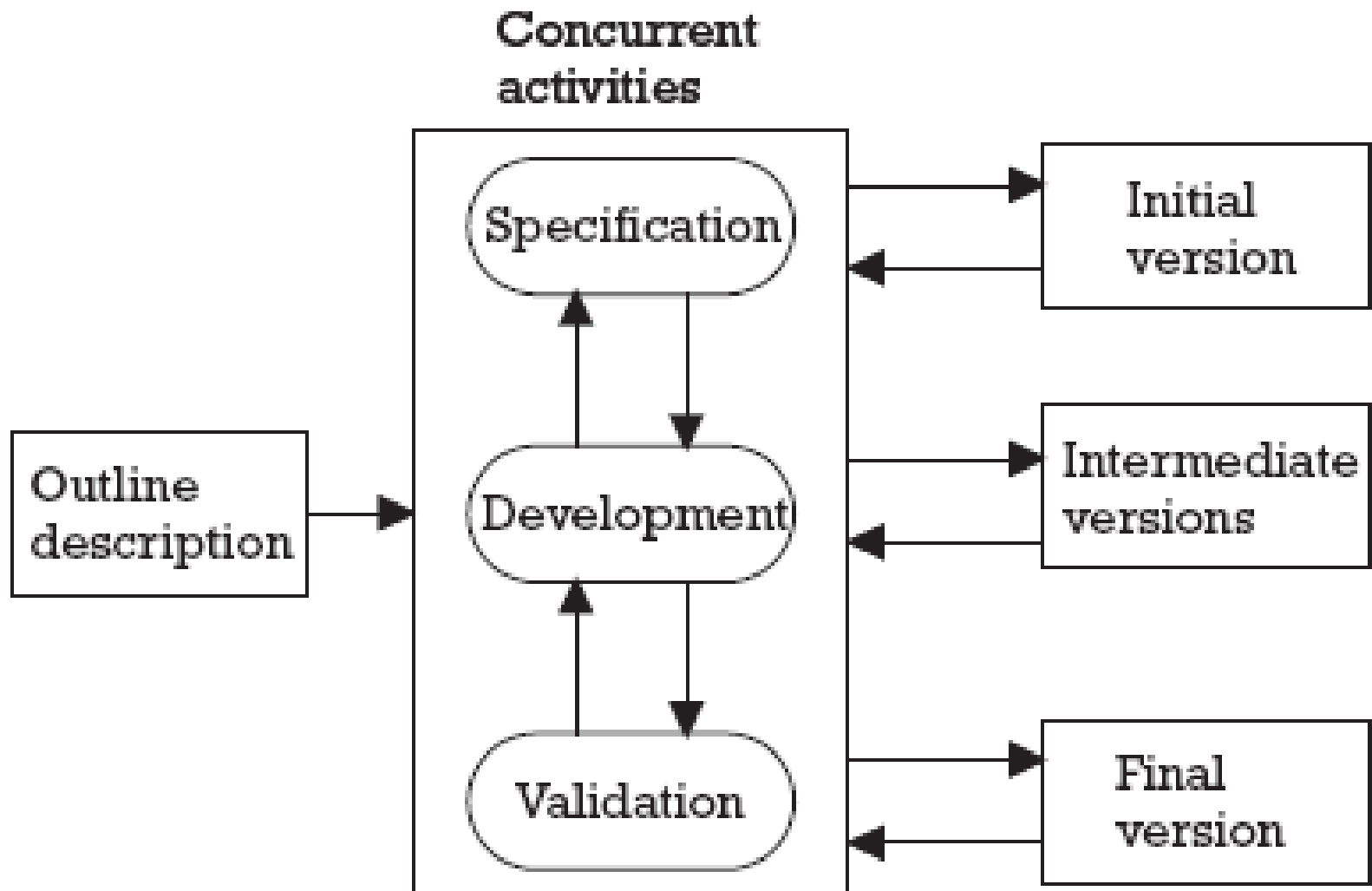
- Existe una tendencia a dejar los problemas complejos para el futuro de manera de demostrar éxito temprano en el manejo
- Dificultad para manejar y medir el proyecto porque no se puede determinar cuando todos los requerimientos serán completados

Iterativo-Incremental

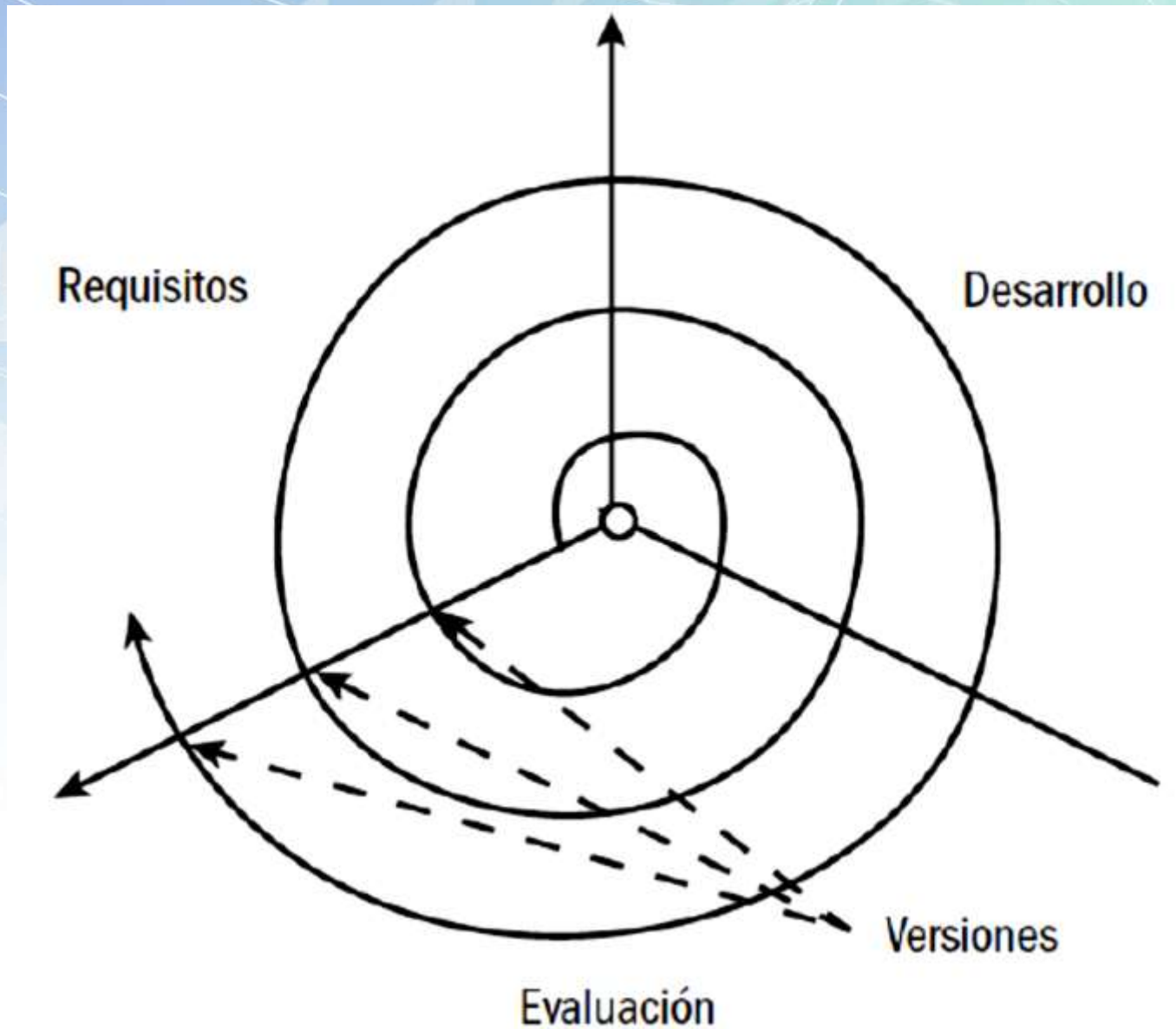
- *Dominio de aplicación*

Proyectos donde los requerimientos no puedan ser bien especificados

Iterativo-Evolutivo



Iterativo-Evolutivo



Iterativo-Evolutivo

- La etapa de desarrollo se ejecuta en una serie de incrementos.
- Cada incremento construye un subconjunto del sistema completo.
- Un incremento es un ciclo de vida completo de análisis, diseño, codificación, testeo e integración.
- Cada incremento puede ser mostrado al usuario, pero no necesariamente.
- Reduce los riesgos ya que va construyendo partes del sistema adoptando este modelo.

Iterativo-Evolutivo

- *Fortalezas:*
 - Está focalizado en la entrega temprana y continua de requerimientos definidos por los stakeholders.
 - Permite que los requerimientos detallados emerjan gradualmente.

Iterativo-Evolutivo

- *Debilidades:*

- Requiere un cambio mental cultural con respecto a los métodos convencionales.
- Requiere entrenamiento y experiencia para ser aplicado efectivamente.

Iterativo-Evolutivo

- Dominio de Aplicación:

Sistemas donde los requerimientos no puedan ser especificados antes del comienzo de las actividades de desarrollo del ciclo de vida.

Iterativo-Prototipos

- Qué son:
 - Son una representación limitada de un diseño y que permite a los usuarios interaccionar con él y explorar sus posibilidades.
- Tipos:
 - Prototipo de papel
 - Storyboard (historietas)
 - Escenario
 - Prototipo de software

Características de Desarrollo de Prototipos

- Debe demandar poco esfuerzo para realizar los cambios
- Debe poseer amplia flexibilidad para el manejo de las interfaces de usuario.
- Debe consumir poco tiempo para generar un nuevo prototipo (maqueta).

Desventajas de los Prototipos

- Costo de entrenamiento/capacitación en la herramienta
- Costo de realizar el prototipo.
- Problema de calendario.
- Incompletitud (puede confundir a los usuarios, haciéndolos pensar que el producto final quedará como el prototipo, incompleto).

Prototipo en papel

- Este tipo de prototipo se basa en la utilización de papel, tijeras, lápiz o instrumentos que se puedan utilizar para describir un diseño en un papel.
- Este sistema nos permite una gran velocidad y flexibilidad.

Prototipo de papel - Cómo se hace

- Para poder simular las diferentes interacciones que vamos a realizar con el sistema, haremos una hoja para cada uno de las diferentes posibles interacciones que podemos realizar.
- Apilaremos estas hojas que nos permitirán simular la aplicación.

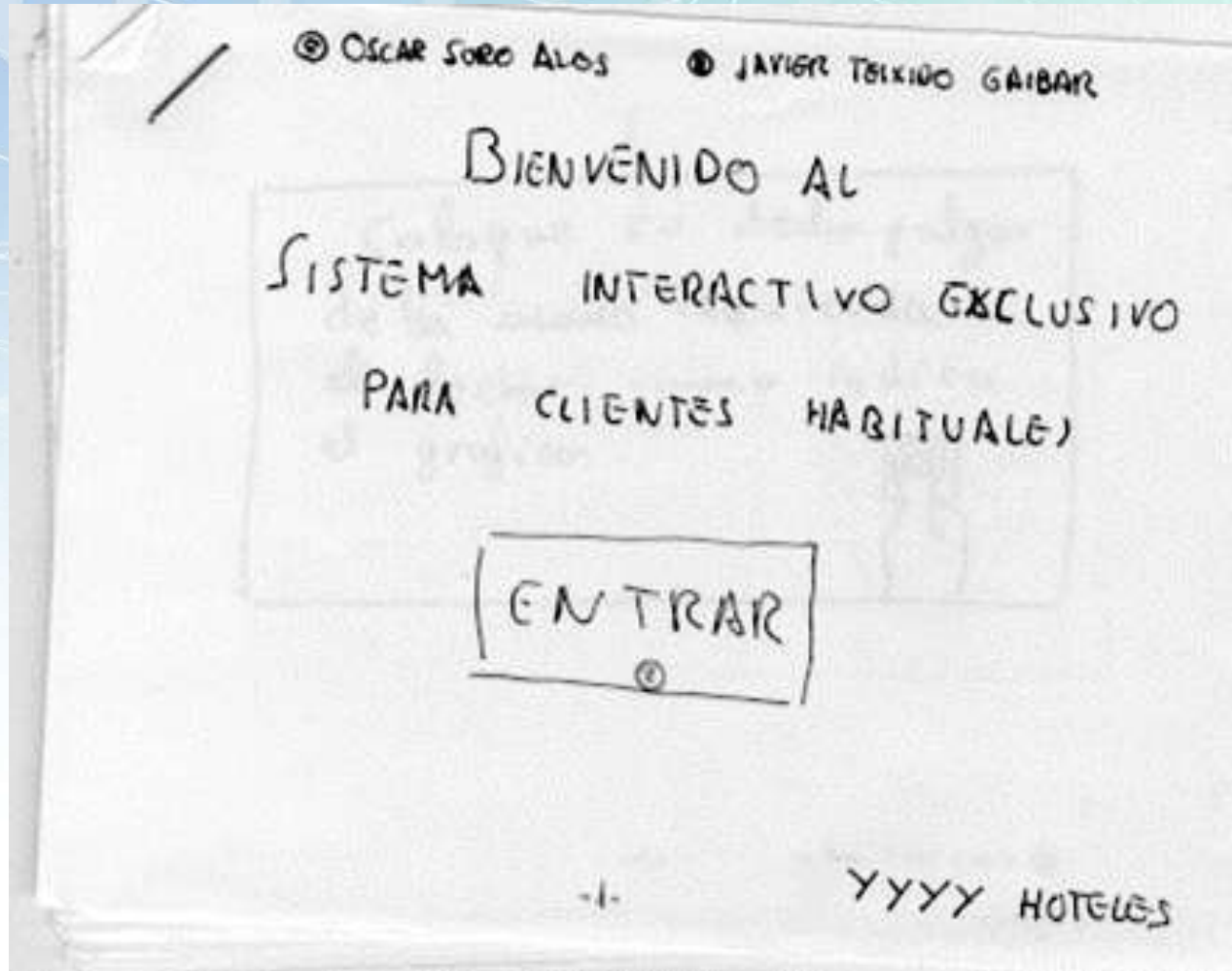
Prototipo en papel - Uso

- Nos situaremos en un escenario de uso de futuro en el que el diseñador actúa como coordinador.
- El prototipo será analizado por un posible usuario que intentará realizar algunas de las tareas a diseñar.
- En voz alta se irán realizando las interacciones y le iremos cambiando las hojas de papel en función de las interacciones que vaya realizando.

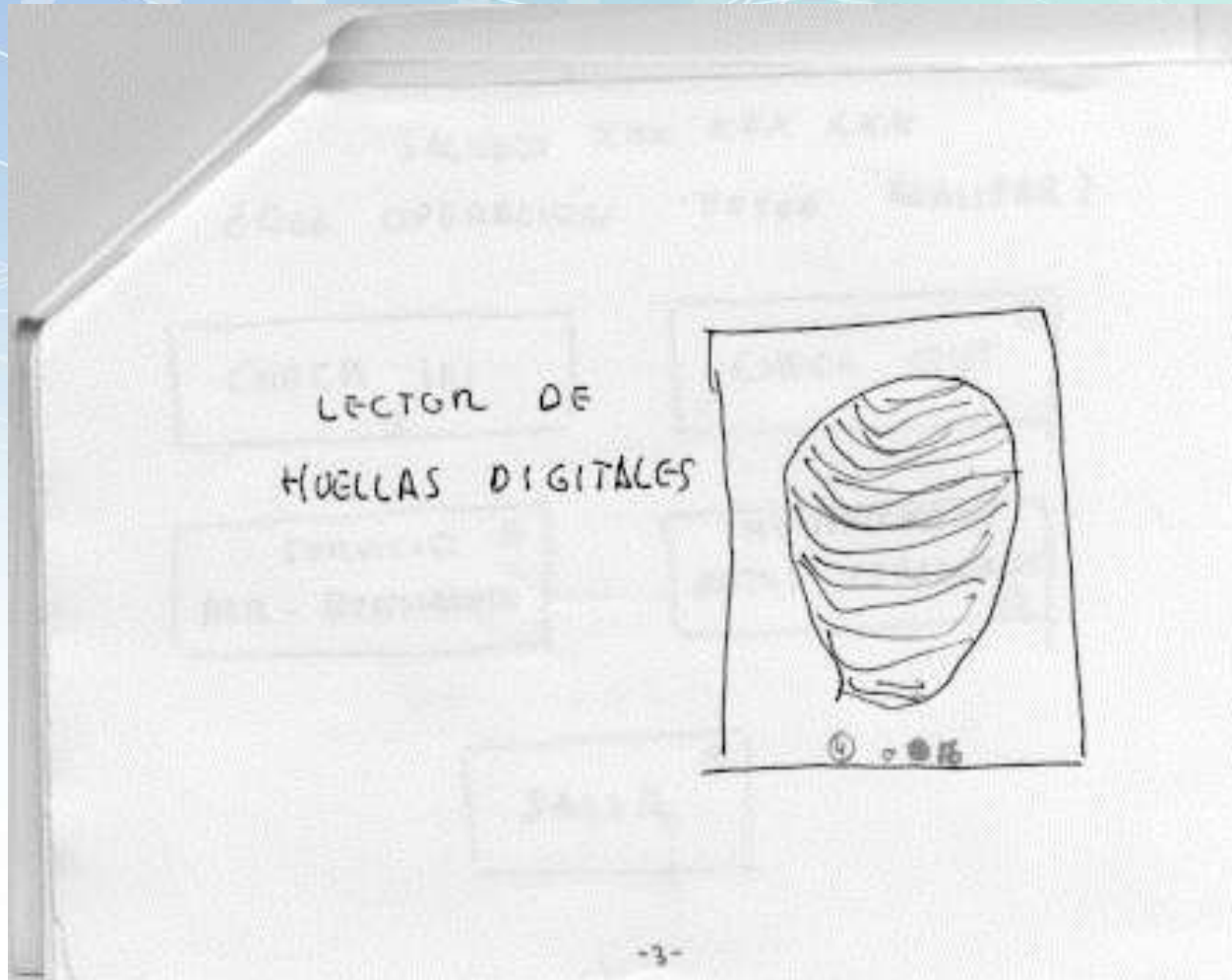
Prototipo de papel - Ventajas

- El costo es muy reducido, necesitando únicamente los recursos humanos dedicados a la realización del prototipo.
- Los cambios se pueden realizar muy rápidamente y sobre la marcha.
- Si el diseño no funciona se puede reescribir las hojas erróneas o rediseñarlas y volver a probar la tarea a realizar.
- Los usuarios o los actores se sienten más cómodos para poder realizar críticas al diseño debido a la sencillez del mismo por lo que no se sienten cohibidos a dar sus opiniones.

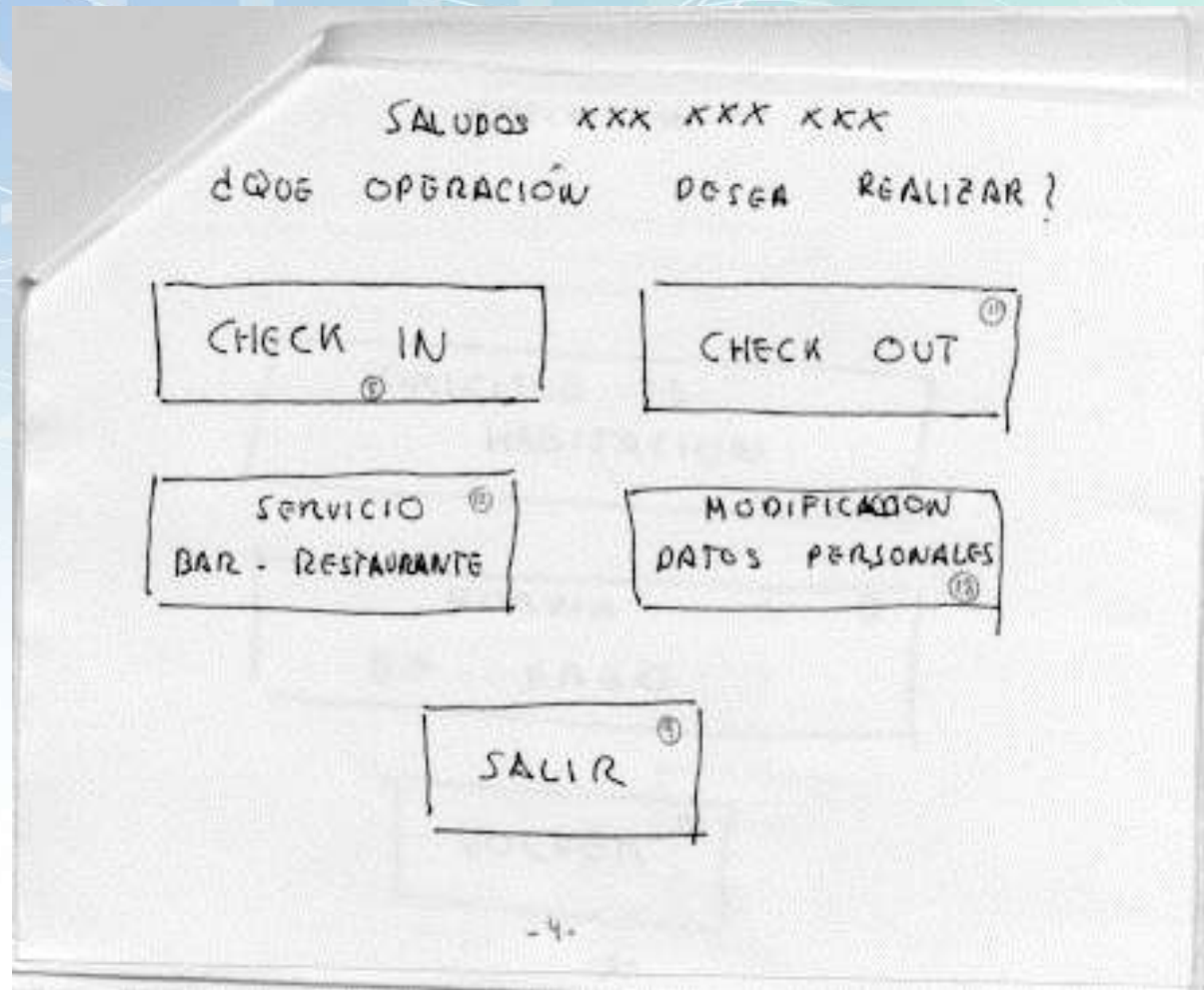
Checkmate - Prototipo en papel



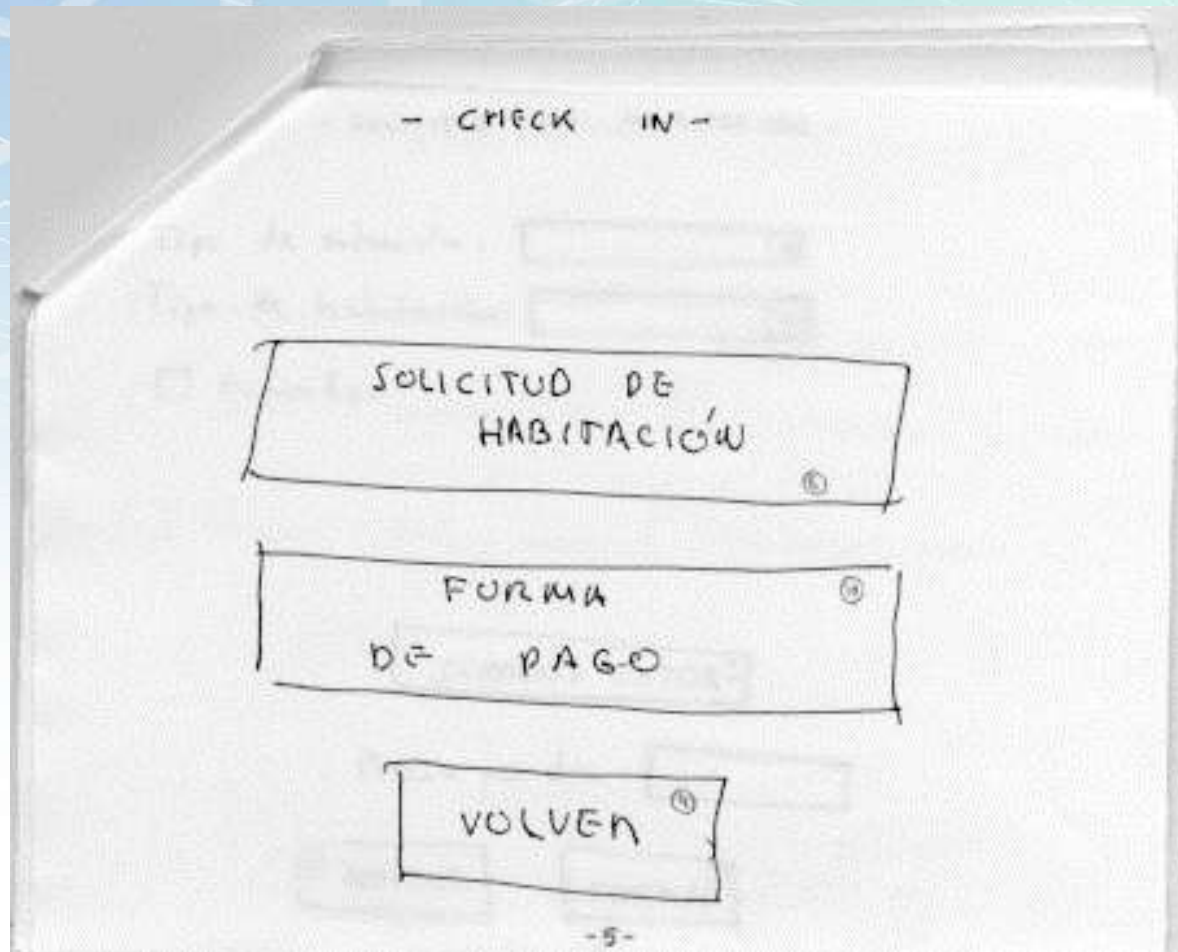
Checkmate - Prototipo en papel



Checkmate - Prototipo en papel



Checkmate - Prototipo en papel



Checkmate - Prototipo en papel

- SOLICITUD DE HABITACIÓN -

Tipo de estancia:

Tipo de habitación:

☐ Fumador

☐ ...

☐ ...

☐ ...

SERVICIOS EXTRA^①

Precio por día:

-6-

Checkmate - Prototipo en papel

- SERVICIOS EXTRA -

<input type="checkbox"/> Masajes. precio: ~~~~~	<input type="checkbox"/> Parking. precio: ~~~~~
<input type="checkbox"/> Gimnasio. Precio: ~~~~~	<input checked="" type="checkbox"/> Lavanderia precio: ~~~~~
<input type="checkbox"/> Sauna. Precio: ~~~~~	<input type="checkbox"/> Discoteca: precio: ~~~~~

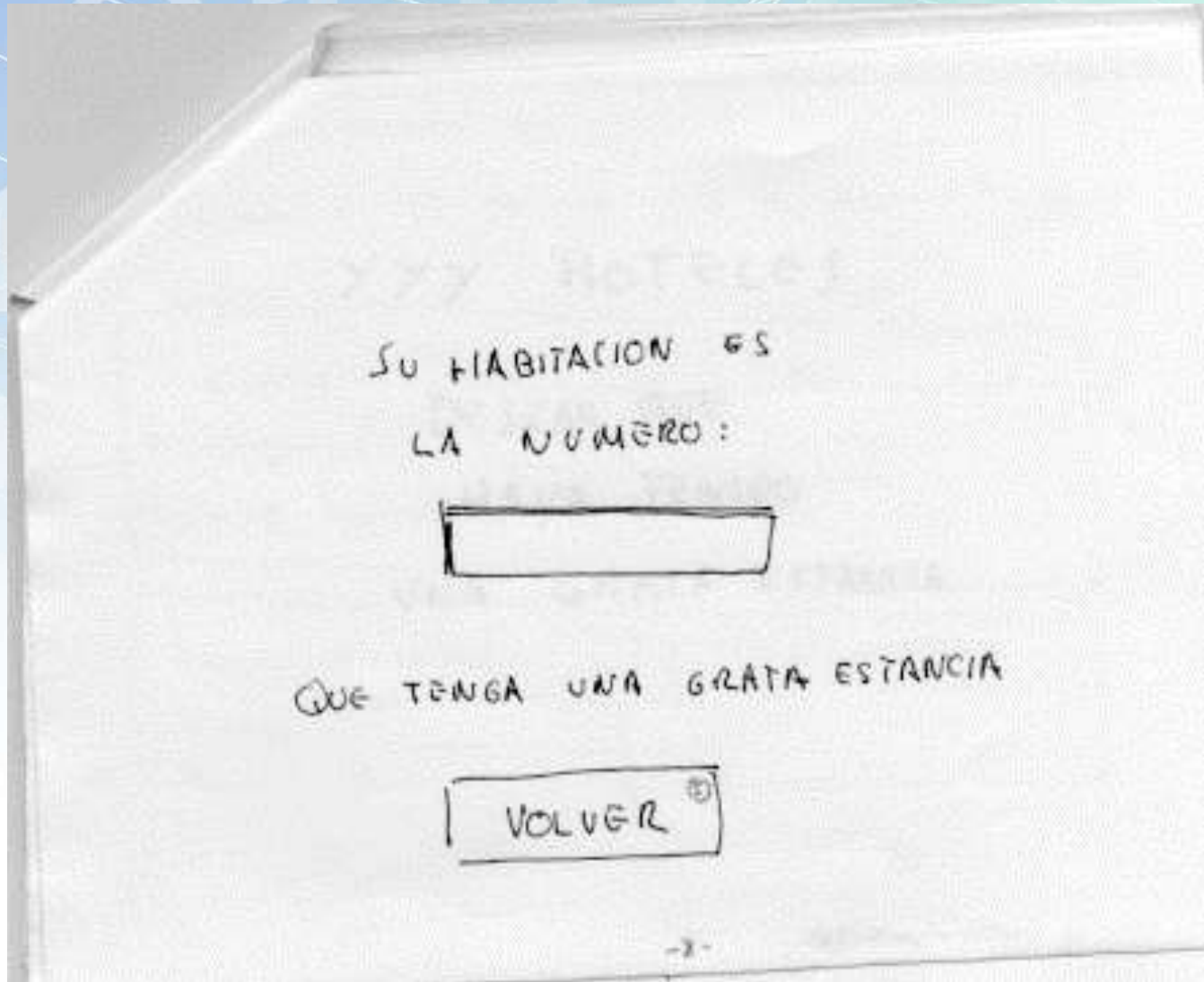
USUARIOS DEL SISTEMA

ACEPTAR CANCELAR

-7-

-8-

Checkmate - Prototipo en papel



Checkmate - Prototipo en papel

- FORMA DE PAGO -

☐ Tarjeta ☐ Efectivo ☐ Distribución bancaria

Tarjeta

☐ Visa ☐ MasterCard ☐ American Express



Número:

Fecha de Caducidad:

Distribución bancaria

Número de cuenta:

Banco:

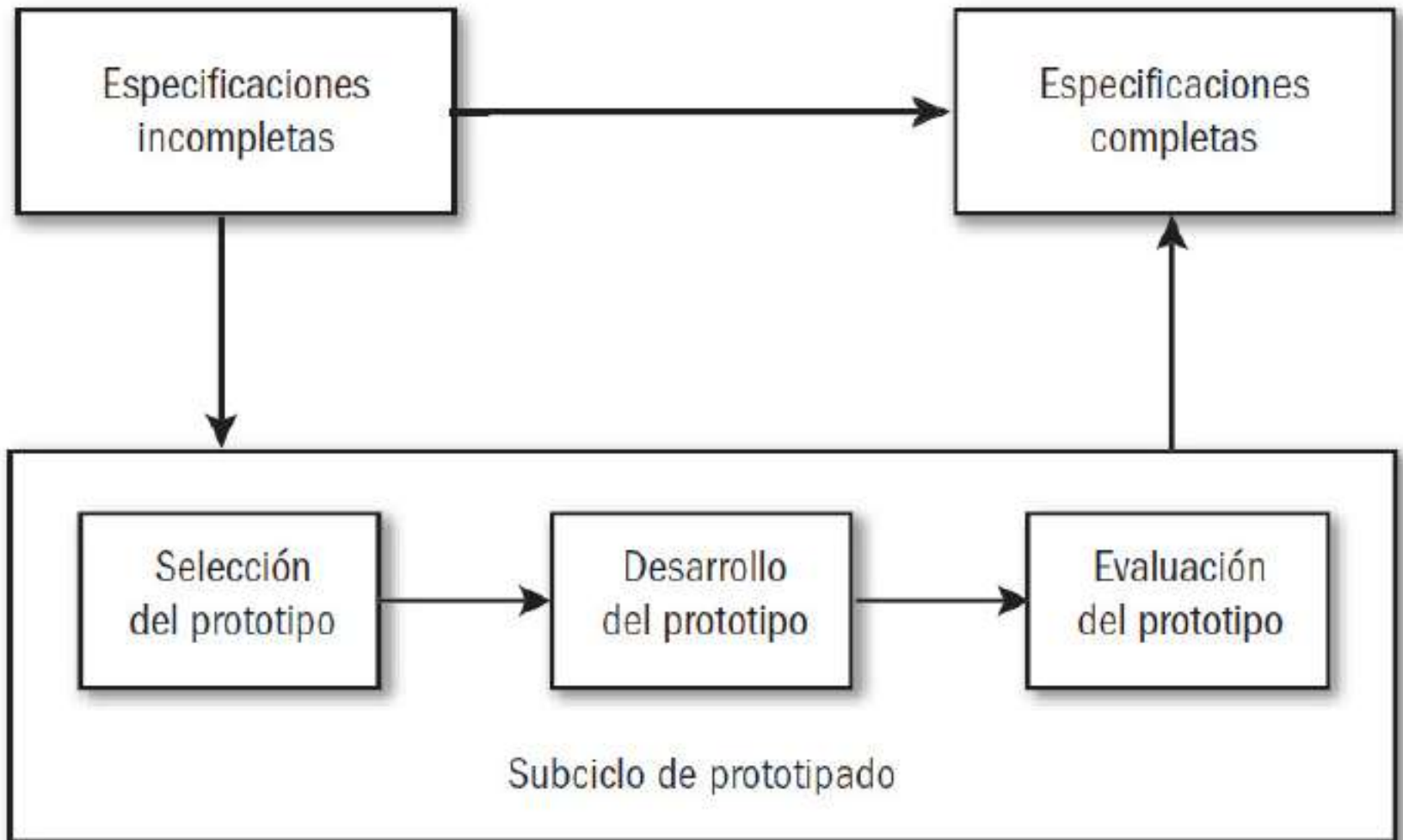
ACEPTAR  **CANCELAR** 

-10-

Prototipo de software



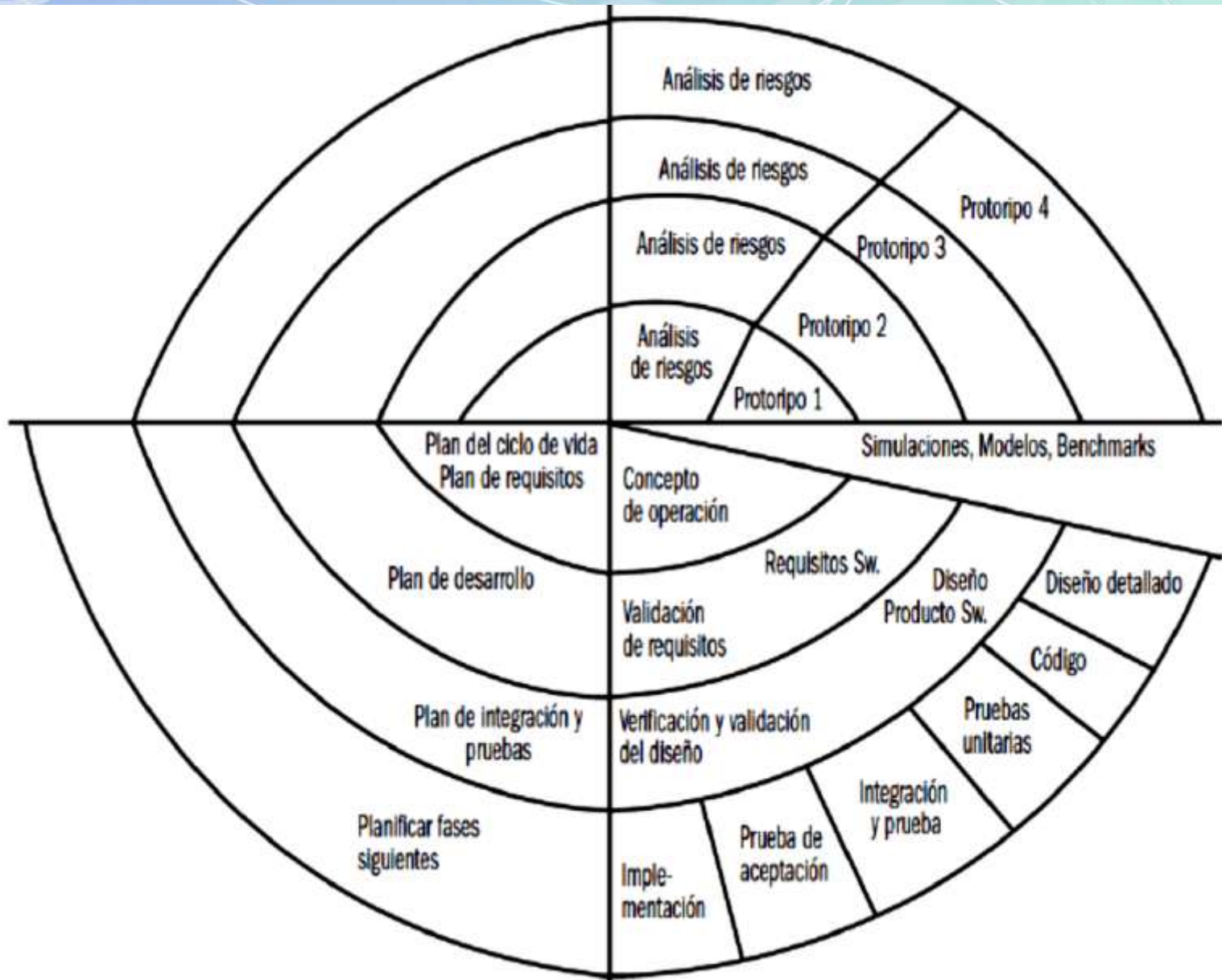
Iterativo-Prototipos



Iterativo-Prototipos

- Los prototipos son un producto parcial, provisional y “desechables”.
- Se utilizan para validar requerimientos, y conocer mediante el prototipo cómo responderán las funcionalidades previstas para el producto final.
- Se debe evaluar si vale la pena el esfuerzo de construir un prototipo.
- El modelo permite suavizar la transición entre los requerimientos iniciales y finales.

Espiral



Espiral

- Variación del modelo por prototipos.
- Se basa en una serie de ciclos repetitivos para ir adquiriendo madurez en el producto final.
- Toma los beneficios de los ciclos de vida incremental y por prototipos.
- A medida que el ciclo se cumple (avance por la espiral) se obtienen prototipos sucesivos hasta lograr plena satisfacción por el usuario.

Espiral

- Divide el desarrollo del sistema en cuatro actividades básicas:
 - Planificación
 - Análisis de riesgos
 - Ingeniería
 - Evaluación
- Con cada vuelta del espiral, los riesgos son identificados y se ejecutan intentos de mitigar los riesgos antes de proceder a las actividades de ingeniería del espiral.

Espiral

- *Fortalezas:*
 - Evita algunas de las dificultades existentes en los modelos de software por el uso de la aproximación guiada por riesgos.
 - Trata de eliminar los errores en fases tempranas.
 - Funciona bien para proyectos complejos, dinámicos e innovadores.
 - Existe reevaluación después de cada fase, permitiendo cambios en las perspectivas del usuario, tecnología, avances o perspectivas financieras.

Espiral

- *Debilidades:*
 - Carencia explicita de guía para determinar objetivos, restricciones y alternativas
 - Provee más flexibilidad de la conveniente para muchas aplicaciones.
 - Necesita experiencia en la determinación de riesgos, y dado que esta no es una tarea sencilla, para realizar una esta tarea, es necesaria una gran experiencia en proyectos para realizarla de manera exitosa.

Espiral

- *Dominio de Aplicación*

Proyectos complejos, dinámicos innovadores y ambiciosos desarrollados por equipos internos (no necesariamente limitados al software)