1. Proceso de Desarrollo del Software

Ph.D Priscilla Jiménez P.

CURSO: ITIR554 – Programación basada en eventos
2020 B – Semana 1

Introducción y Conceptos

1. Ingeniería del software
2. Dominios de aplicación
3. El proceso de software
4. Equipos de desarrollo de software

Ingeniería del Software

Estudia cuál es la mejor manera de producir software de calidad.

Disciplina de la ingeniería que se preocupa de todos los aspectos de la producción de software.

Meta: es el desarrollo costeable de sistemas de software. Comprende todos los aspectos de la producción de software.

Ingeniería del Software

Sistema de Software son todos los documentos asociados y la configuración de datos que se necesitan para hacer que estos programas operen de manera correcta.

Software = Programas + Documentación asociada

Ingeniería del Software

Métodos de ingeniería del software son enfoques estructurados para el desarrollo de software y aspectos de la producción que incluyen:

modelos de sistemas	notaciones	sugerencias diseño	administración del proyecto	guías de procesos

En diferentes fases, desde las especificaciones del sistema hasta el mantenimiento después de lo que se utiliza.

Características de un producto

Debe poder evolucionar

- Satisfacer las necesidades cambiantes del cliente

Confiabilidad y seguridad

- No debe causar daño físico ni económico en caso de falla
- Protección contra usuarios malintencionados

Eficiencia

- No desperdiciar recursos del sistema (capacidad de respuesta, tiempo de procesamiento, utilización de memoria)

Aceptabilidad

- Aceptable al tipo de usuario. (ser comprensible, utilizable y compatible con otros sistemas)

Retos que afronta la Ingeniería del Software: problemas generales que afectan diversos tipos de software

- Heterogeneidad (incremento en la integración con diferentes sistemas)
- Cambio empresarial y social (tiempo de entrega cortos)
- Seguridad y confianza (web requiere que se demuestre que se puede confiar en el software - seguridades)

Diversidad en dominios de aplicación

Aplicaciones Independientes	Resuelven una necesidad específica, y no requieren conectarse a una red (aplicaciones en una oficina, programas CAD, manipulación de fotografías)
Aplicaciones interactivas basadas en transacción o software de aplicación	Controlan funciones de negocios en tiempo real, se ejecutan o tienen acceso a computadoras remotas (sistemas empresariales y servicios basados en la nube)
Sistemas de control embebido	Sistemas de control de software que regulan y gestionan dispositivos de hardware (sw de teléfono móvil, control de frenos antibloqueo de un automóvil)
Software de inteligencia artificial	Uso de algoritmos no numéricos para resolver problemas complejos. Robótica, sistemas expertos, reconocimiento de patrones, etc.

Diversidad en dominios de aplicación

Sistemas de procesamiento en lotes	Procesan datos en grandes lotes (batch) (sistemas de facturación periódica, pagos de salarios). No necesita interacción con el usuario.
Sistemas de entretenimiento	Sistemas para uso personal que tienen la intención de entretener al usuario.
Software de ingeniería y ciencias: sistemas para modelado y simulación	Sistemas desarrollados para modelar procesos o situaciones físicas. A menudo son computacionalmente intensivos.
Sistemas de adquisición de datos	Recopilan datos del entorno usando sensores, y envían esos datos para su procesamiento a otros sistemas.
Sistemas de sistemas	Son sistemas compuestos de un cierto número de sistemas de software.

Proceso de software

Conjunto de actividades, acciones y tareas que se ejecutan cuando van a crearse un producto de software.

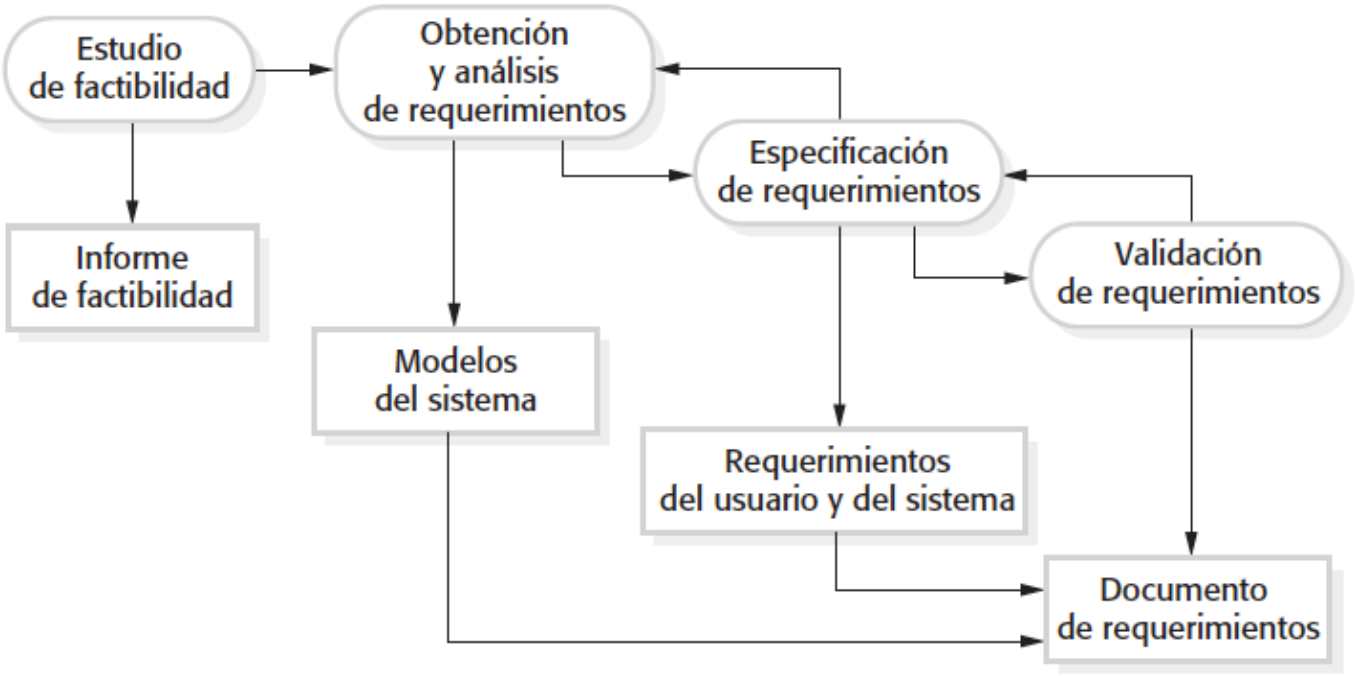
Estas actividades se desarrollan sin importar el dominio de la aplicación.

Este proceso no es rígido, busca siempre entregar el software en forma oportunidad y con buena calidad para satisfacer a clientes y usuarios.

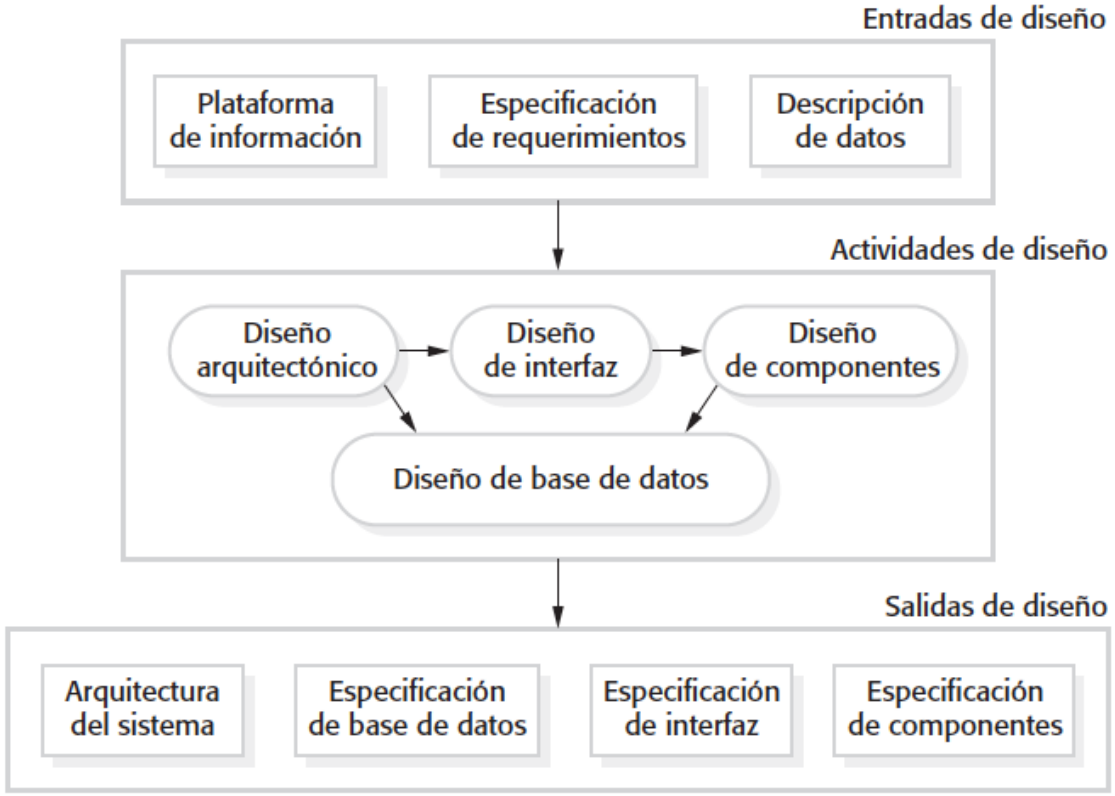
Actividades en el proceso del software

- Especificación del software (definición)
- Planeación (plan del proyecto)
- Modelado/Desarrollo del software (diseño y programación)
- Validación del software (validación de requerimientos) y Despliegue (entrega al consumidor)
- Evolución del software (modificación/adaptación a cambios)

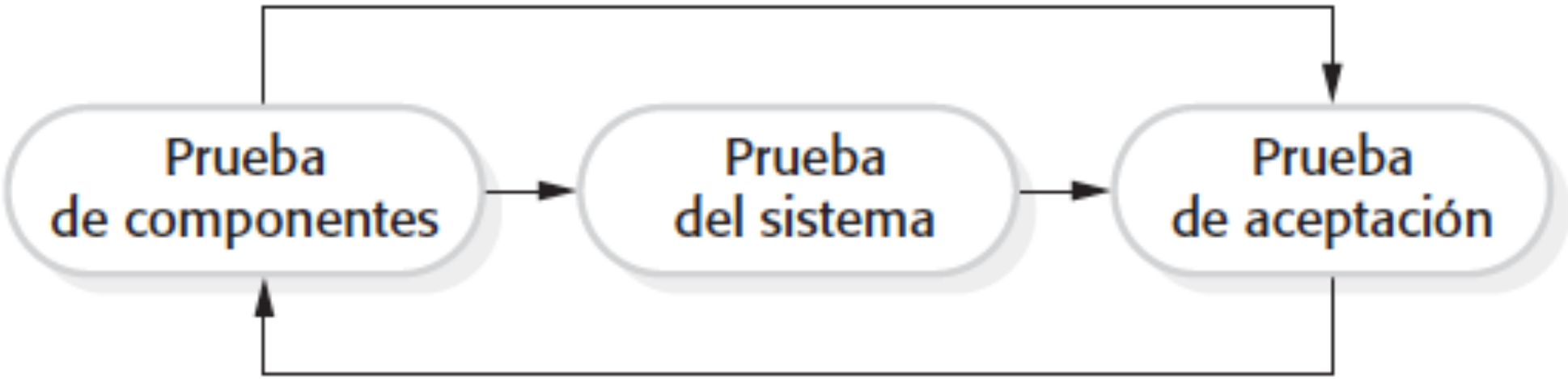
Diferentes tipos de proyectos/sistemas necesitan diferentes procesos de desarrollo



Actividades del proceso – Especificación de Requerimientos



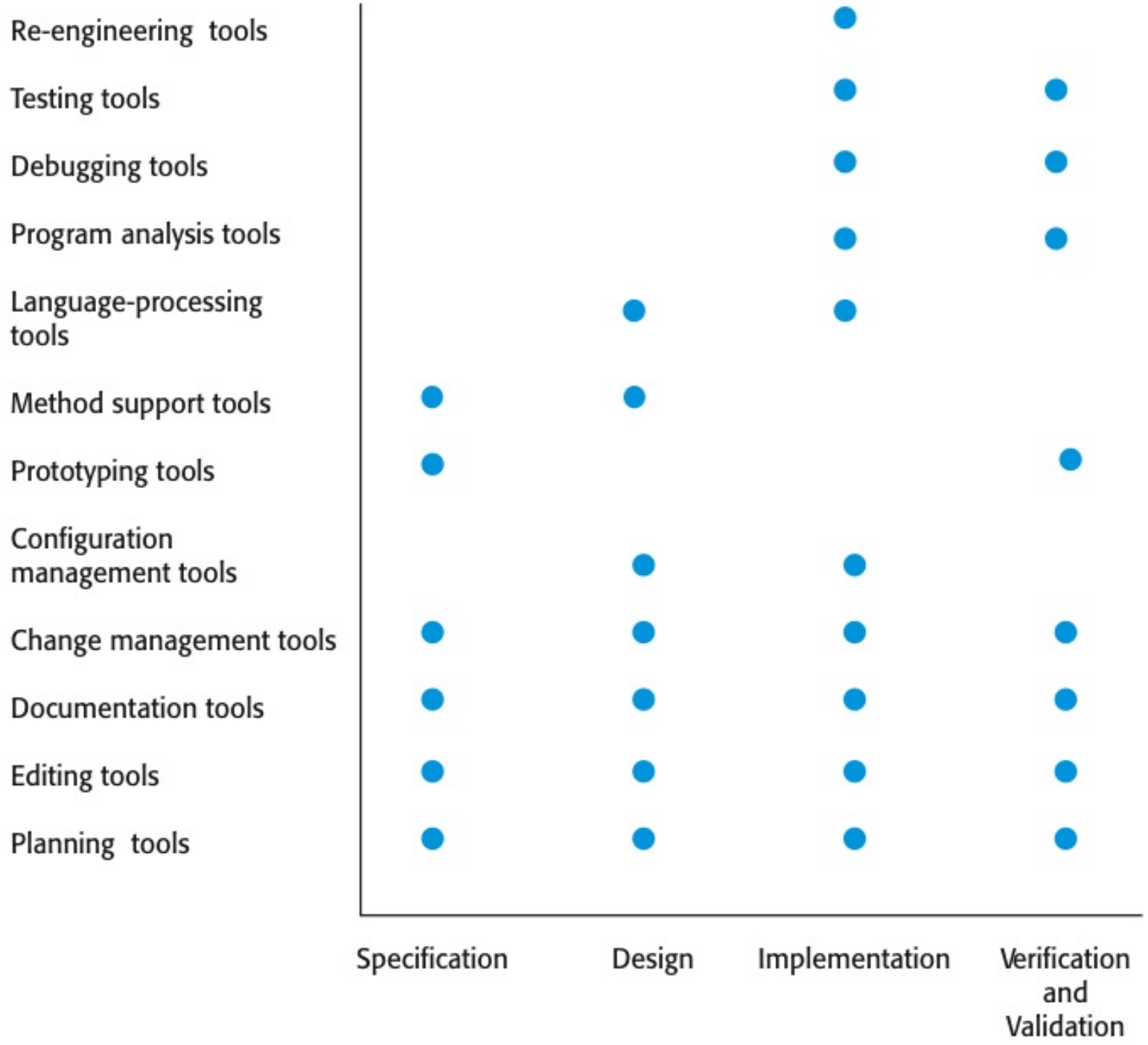
Actividades del proceso – Diseño e implementación del software

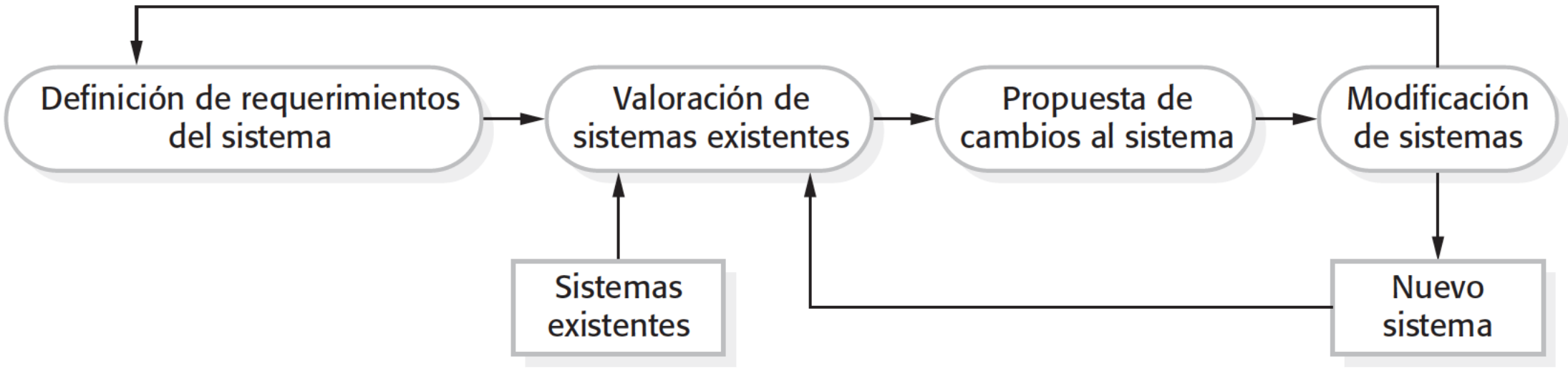


Actividades del proceso – Validación de Software

Sistemas CASE

Computer Aided Software Engineering





Evolución del Sistema –

Refactorización:

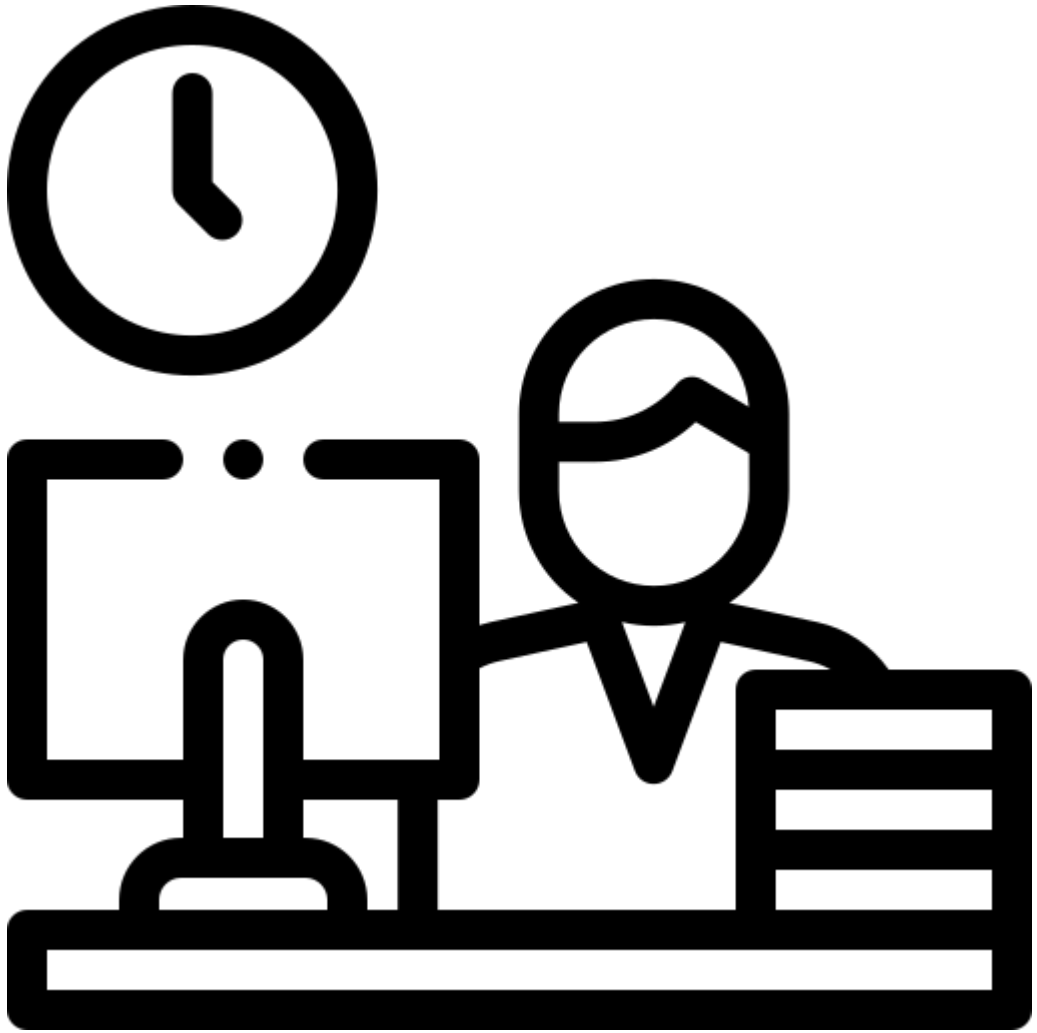
Mejoramiento de la estructura y organización de un programa.

Ética en la Ingeniería del Software

- Confidencialidad
- Competencia
- Derechos de propiedad intelectual
- Mal uso de computadoras

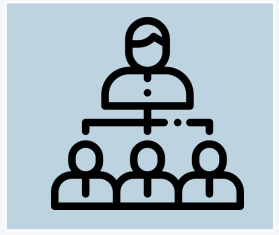
Práctica 1

Revisión – Estudio de caso

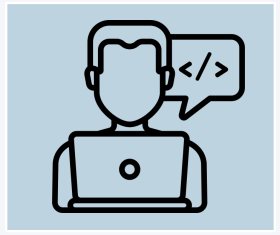


Roles y Responsabilidades

Lider del proyecto



Lider de Equipo



Desarrollador

Asegurador de Calidad

Arquitecto de Software

Ingeniero de pruebas

Diseñador - UX

Lider de documentación

¿Cómo trabajar bien en equipo ?

- El desarrollo de software es un esfuerzo en equipo.
- Cómo convertirse en un ingeniero eficiente y exitoso que gasta menos energía tratando con personas problemas y más tiempo siendo productivo.
- Recordar: La única variable sobre la que definitivamente tiene control es usted.
- Principios básicos: humildad, respeto y confianza.

Wright, H., Winters, T., y Manshreck, T. (2020). Software Engineering at Google. O'Reilly

El mito del genio

- El mito del genio es la tendencia que nosotros, como humanos, necesitamos atribuir el éxito de un equipo a una sola persona / líder.
- Genios para ingenieros de software: Linus Torvalds, Guido Van Rossum, Bill Gates, Steve Jobs.

El mito del genio

1. En el fondo, muchos ingenieros secretamente desean ser vistos como genios.
2. Tienen una idea brillante de un nuevo concepto.
3. Se desvanecen en su cueva durante semanas o meses.
4. Luego libera su software, sorprendiendo a todos con su genio.
5. Sus compañeros están asombrados por su inteligencia.
6. La gente hace cola para usar su software.
7. La fama y la fortuna siguen naturalmente.

El mito del genio – Recordar:

- Personas inteligentes también comenten errores.
- Tener ideas brillantes y habilidades de programación de élite no garantiza que su software sea un éxito.
- Peor aún, podría encontrarse resolviendo solo problemas analíticos y no humanos.
- Ser una persona inteligente definitivamente no es una excusa para ser un subnormal, cretino, insensato…
- Cualquiera, genio o no, con malas habilidades sociales tiende a ser un pobre/mal compañero de equipo.

El mito del genio – Recordar:

- La gran mayoría del trabajo en las empresas no requiere intelecto nivel genio, pero el 100% de los trabajos requiere un nivel mínimo de habilidades sociales.
- Ser un ***buen colaborador*** le ayudará a hacer una buena carrera profesional.
- Resulta que este mito del genio es solo una manifestación de nuestra inseguridad.

- La colaboración ayuda en la revisión.
- Detección temprana de errores.
- Se corre el riesgo de reinventar la rueda.

Trabajar solo y ocultarse es dañino

- Aumenta el riesgo de fallas innecesarias y está engañando su potencial de crecimiento.
- "Fallar temprano, fallar rápido, fallar con frecuencia".
- También es importante fortalecer lo que llamamos el factor de bus de su proyecto.

Trabajar solo y ocultarse es dañino

El factor de bus

Cantidad de personas que necesitan ser atropelladas por un autobús antes de que su proyecto esté completamente condenado.

Se espera que los ingenieros reconozcan que es mejor ser parte de un proyecto exitoso que la parte crítica de un proyecto fallido.

El factor de bus

- Si tiene un pequeño equipo diseñando y creando prototipos juntos, las cosas son mejores: el proyecto no quedará abandonado cuando un miembro del equipo desaparezca.
- Asegurarse de que hay al menos una buena documentación, además de un propietario primario y uno secundario para cada área de responsabilidad.
- Esto ayuda a preparar el éxito de su proyecto y aumenta el factor de bus de su proyecto.

Trabajar solo y ocultarse es dañino

Ritmo de progreso/avance

- Trabajar solo es a menudo un trabajo duro, mucho más lento de lo que la gente quiere admitir.
- ¿Cuánto aprendes cuando trabajas solo?
- ¿Qué tan rápido progresas?
- No se puede sustituir la experiencia humana con foros en línea.

Trabajar con otras personas aumenta directamente la sabiduría colectiva detrás del esfuerzo.

Trabajar solo y ocultarse es dañino

Ingenieros y oficinas – opiniones controversiales.

- Hace 25 años – para ser productivos debes de tener oficina y con una puerta cerrada.
- Compañías modernas piensan que oficinas "piso abierto" son más beneficiosas.
- Encontrar un término medio es la mejor solución.

Encontrar el equilibrio correcto es un arte.

- En el mundo de la programación el logro que cambia el mundo casi siempre es el resultado de una chispa de inspiración seguida de un heroico esfuerzo de equipo.
- Comparte tu visión.
- Divide el trabajo.
- Aprende de los demás.

Todo se trata del equipo

Los equipos de alto funcionamiento son oro y la verdadera clave del éxito.

Los tres pilares de la interacción social

Si el trabajo en equipo es la mejor ruta para producir un excelente software, ¿cómo se puede construir (o encontrar) un gran equipo?

Tres pilares:

Humildad	Respeto	Confianza

Las relaciones siempre duran más que los proyectos.

Perder el ego

- ¿Sientes que necesitas tener la primera y última palabra en cada tema ?
- ¿Sientes la necesidad de comentar cada detalle de una propuesta o discusión ?
- No hay nada malo en la confianza en uno mismo. Simplemente no salgas como un sabelotodo.
- Trata de desarrollar un sentido de logro en equipo.

Humildad, Respeto y Confianza en la práctica

Aprende a dar y tomar una crítica

- Ofrecer soluciones/mejoras no solicitadas vs. discutir la idea con el equipo por adelantado y preguntar a los miembros del equipo probarlo por un tiempo.
- En un entorno profesional/académico la crítica casi nunca es personal, generalmente es solo parte del proceso de hacer un mejor trabajo.

Humildad, Respeto y Confianza en la práctica

Su autoestima no debe estar conectada al código que escribe, ni a ningún proyecto creativo que cree.

Aprende a dar y tomar una crítica

- Cambiar el "Hombre, entendiste mal el flujo de control de ese método allí. Deberías usar el patrón de código xyz estándar como todos los demás ”.
- Por "Oye, estoy confundido por el flujo de control en esta sección aquí. Me pregunto si el patrón del código xyz podría hacer esto más claro y más fácil de mantener."

Humildad, Respeto y Confianza en la práctica

La discusión se centra en el código en sí.

Fallo rápido e iterado

- Google: “El fracaso es una opción” -> Si no falla de vez en cuando, no está siendo lo suficientemente innovador o no está asumiendo suficientes riesgos.
- El fracaso es visto como una oportunidad de oro para aprender y mejorar para la próxima vez.
- "Si encuentro 10,000 formas en que algo no funcionará, no he fallado. No estoy desanimado, porque cada intento equivocado descartado es otro paso adelante". – Thomas Edison

Humildad, Respeto y Confianza en la práctica

Cultura “postmortem” autopsia sin culpa

- La clave para aprender de sus errores es documentar fallas realizando un análisis de la causa raíz y redactando una “autopsia”.
- No es una lista de disculpas o excusas, o señalar con el dedo.
- Un documento adecuado debe contener una explicación de lo que se aprendió y lo que va a cambiar como resultado de la experiencia de aprendizaje.

Humildad, Respeto y Confianza en la práctica

Cultura “postmortem” autopsia sin culpa

- Luego asegurarse de que la autopsia sea fácilmente accesible y que el equipo realmente cumpla con los cambios propuestos.
- Documentar las fallas facilita que otras personas eviten repetir la historia.
- El documento debe contener: Resumen del evento, línea de tiempo, la causa del evento, evaluación del impacto y daños, elementos de acción para solucionar el problema, evitar que el evento vuelva a ocurrir, lecciones aprendidas.

Humildad, Respeto y Confianza en la práctica

¡No borres tus huellas, enciéndelas como una pista de aterrizaje para los que te siguen!

Aprender paciencia

- Aprender las formas de trabajar de otros, y con paciencia, esto ayudará a encontrar nuevas maneras de colaborar.

Humildad, Respeto y Confianza en la práctica

Esté abierto a la influencia

- Cuanto más abierto seas a influir, más podrás influir.
- Personas tercas, con el tiempo tienden a ser ignoradas.

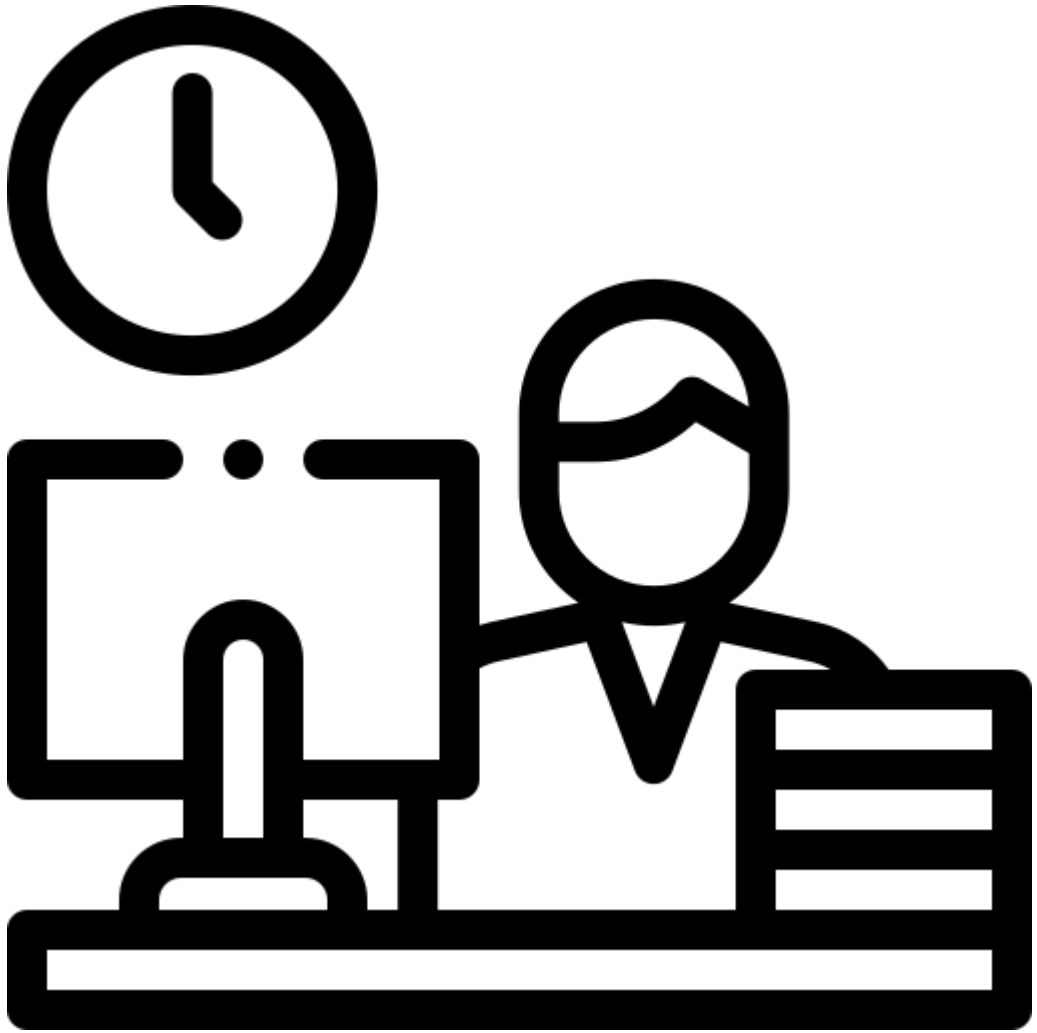
Humildad, Respeto y Confianza en la práctica

Para recordar:

- Tenga en cuenta las ventajas y desventajas de trabajar de forma aislada.
- Reconozca la cantidad de tiempo que usted y su equipo pasan comunicándose y en conflictos interpersonales.
- Una pequeña inversión en la comprensión de personalidades y estilos de trabajo propios y ajenos puede ser de gran ayuda para mejorar la productividad.
- Si desea trabajar eficazmente con un equipo o una gran organización, tenga en cuenta su estilo de trabajo preferido y el de los demás.

Práctica 2

Formación de grupos y definición del proyecto.

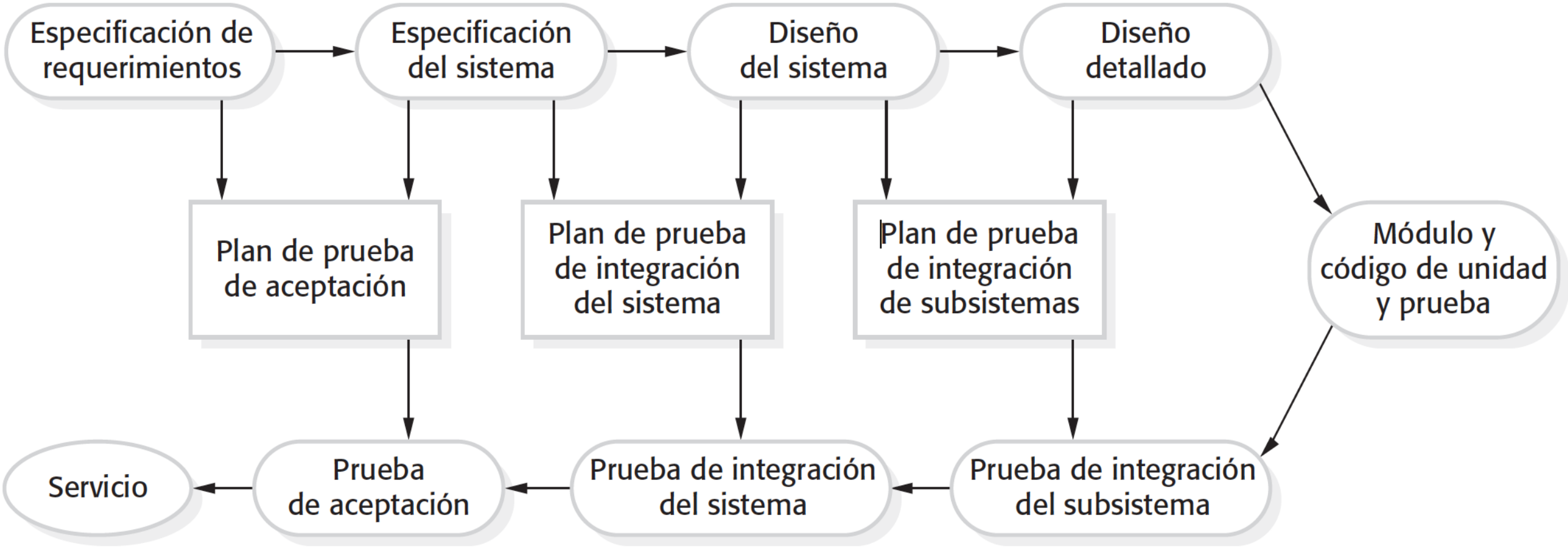


Modelos de desarrollo de software

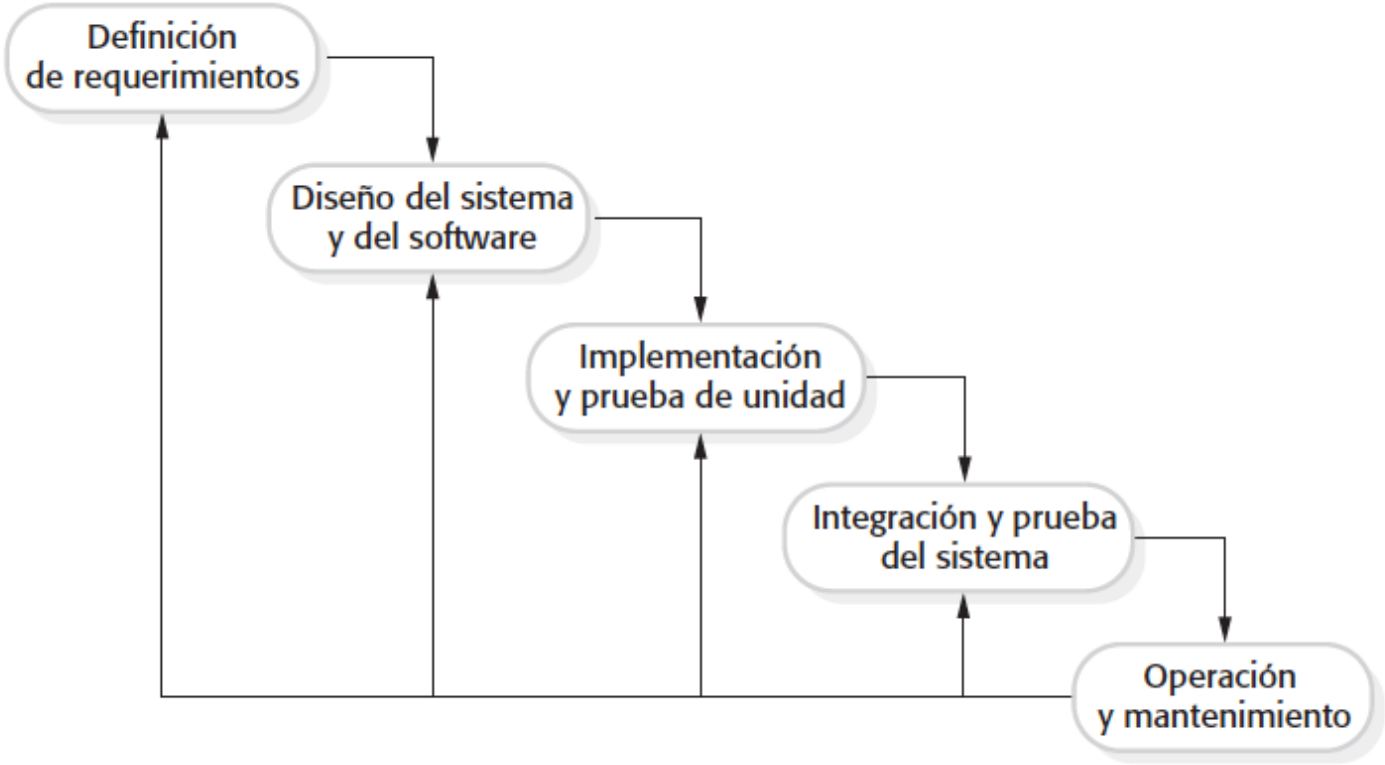
1. Cascada
2. Evolutivo
3. Basado en componentes
4. Espiral
5. Proceso Unificado Racional
6. Ágil

Modelos de Procesos de Software

- Los modelos no son mutuamente excluyentes.
- Los subsistemas dentro de un sistema más grande se desarrollan usando diferentes enfoques.
- Partes del sistema que deben ser bien comprendidas y son críticas pueden especificarse y desarrollarse al utilizar un proceso basado en cascada.
- Partes del sistema que por adelantado son difíciles de especificar (interfaz de usuario), siempre deben desarrollarse con un enfoque incremental.



Probando fases en un proceso de software dirigido por un plan



Modelo en cascada (Waterfall model)

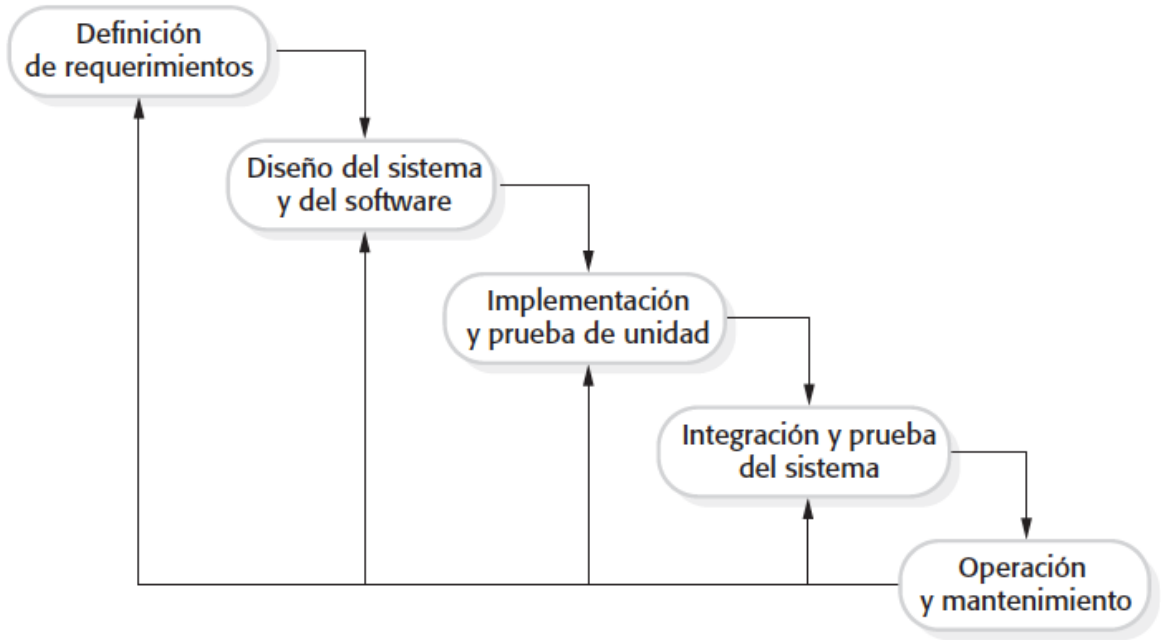
Es un proceso de desarrollo secuencial

Considera las actividades fundamentales del proceso de desarrollo del software.

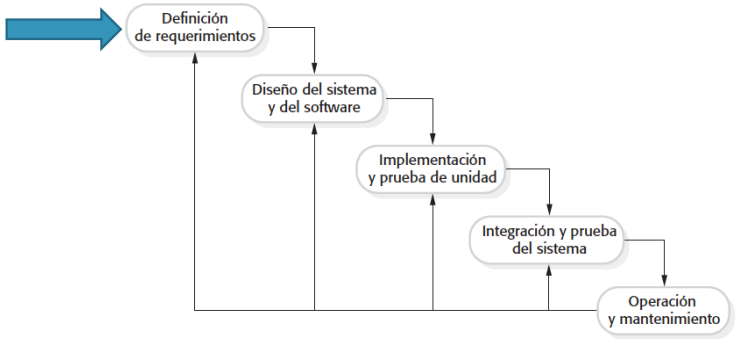
Cada actividad es una fase independiente

Modelo en cascada (o ciclo de vida del software)

- Es un ejemplo de un proceso dirigido por un plan.
- El resultado de cada fase consiste de 1+ documentos.
- Retroalimentación de una fase a otra produce modificaciones en docs. de otras fases.



Etapas del modelo en cascada

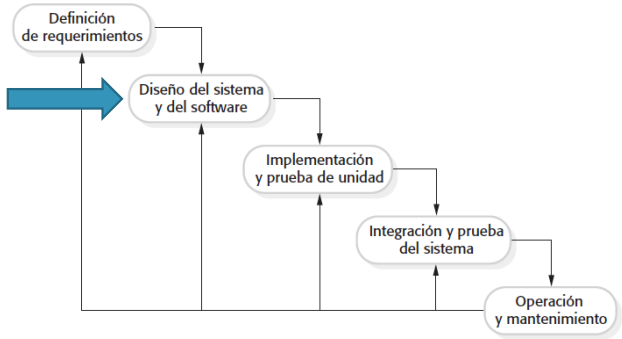


1. Análisis y definición de requerimientos
   - Servicios
   - Restricciones
   - Metas

   Se establecen mediante consulta a los usuarios

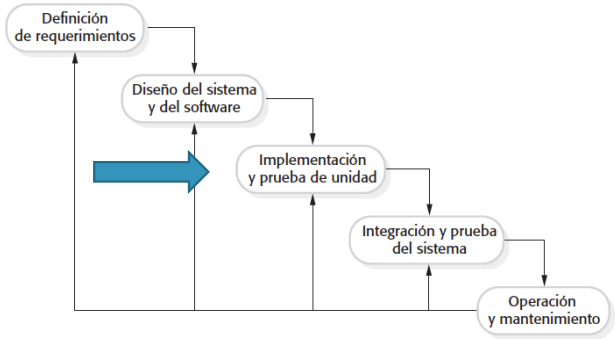
   Se definen con detalle y sirven como una especificación del sistema

Etapas del modelo en cascada



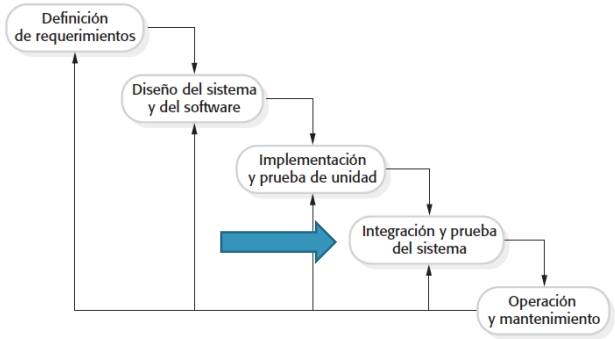
2. Diseño del sistema y del software
   - Asigna requerimientos para el HW y SW
   - Define una arquitectura de sistema global
   - Identifica y describe las abstracciones del sistema y sus relaciones

Etapas del modelo en cascada



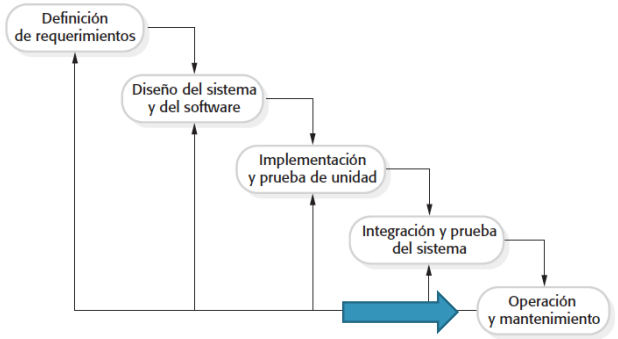
3. Implementación y prueba de unidad
   - El diseño de SW se realiza como un conjunto de programas.
   - Las pruebas de unidad verifican que cada unidad cumpla con su especificación.

Etapas del modelo en cascada



4. Integración y prueba de sistema
   - Los programas individuales se integran como un sistema completo.
   - Se verifica que cumplan los requerimientos de software.

Etapas del modelo en cascada



5. Despliegue y mantenimiento
   - El sistema se instala y se pone en práctica.
   - Corregir errores que no se detectaron en etapas anteriores.
   - Mejorar la implementación de las unidades del sistema.
   - Incrementar los servicios del sistema conforme se descubran nuevos requerimientos.

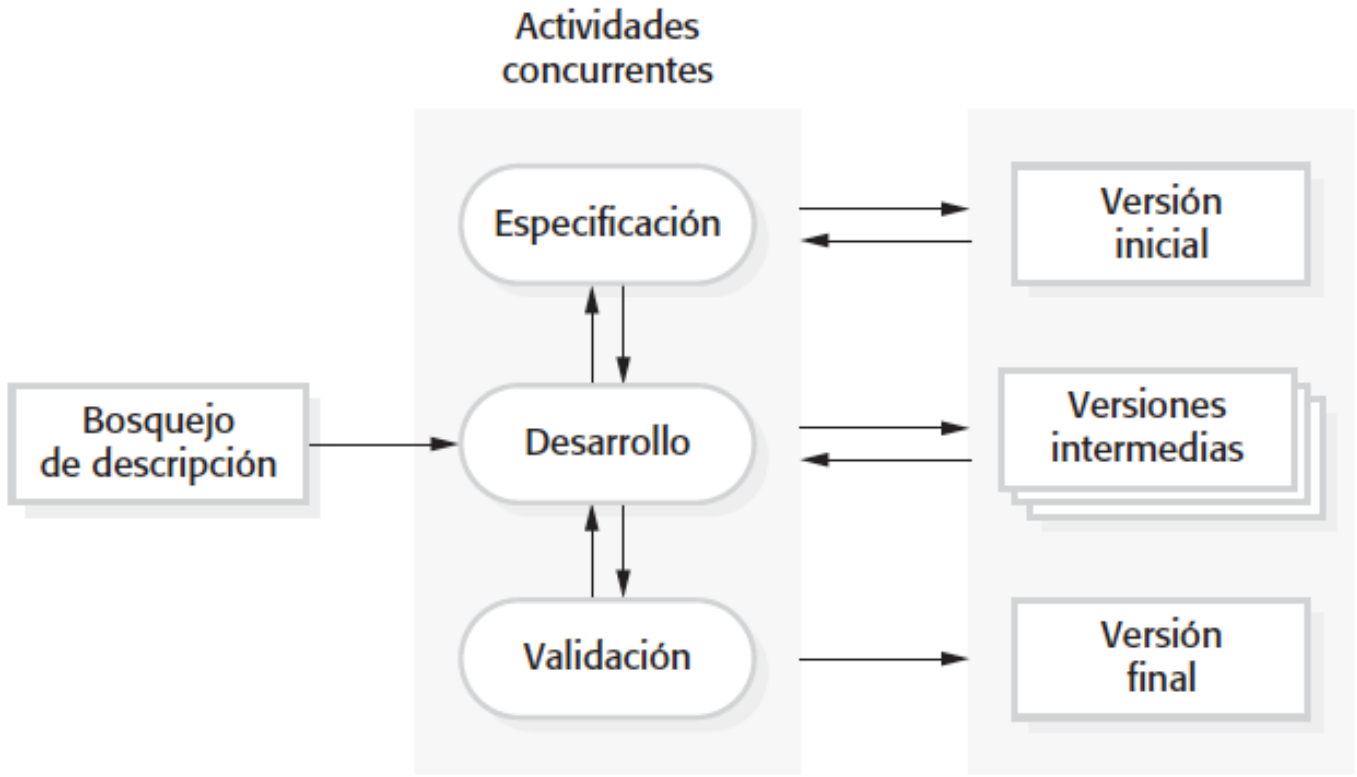
Ventajas y desventajas del modelo en cascada

✓El proceso de desarrollo del software es visible.

✓ Facilita el monitoreo y progreso del desarrollo del software.

✗Partición inflexible del proyecto en distintas etapas.

✗Dificulta responder a los requerimientos cambiantes del cliente.

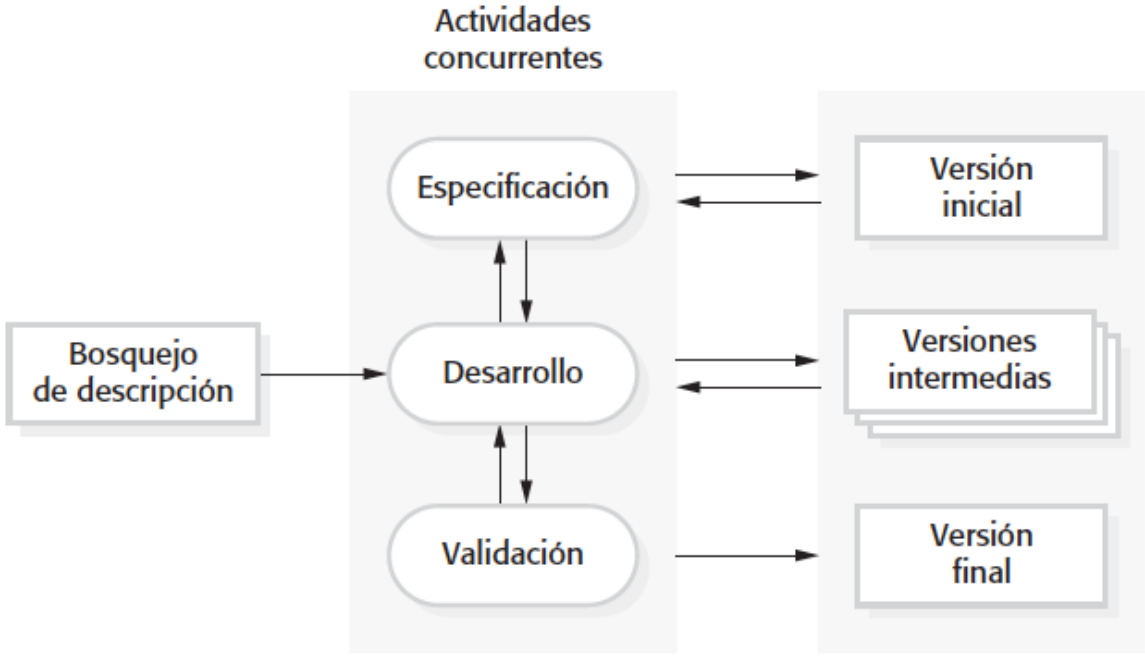


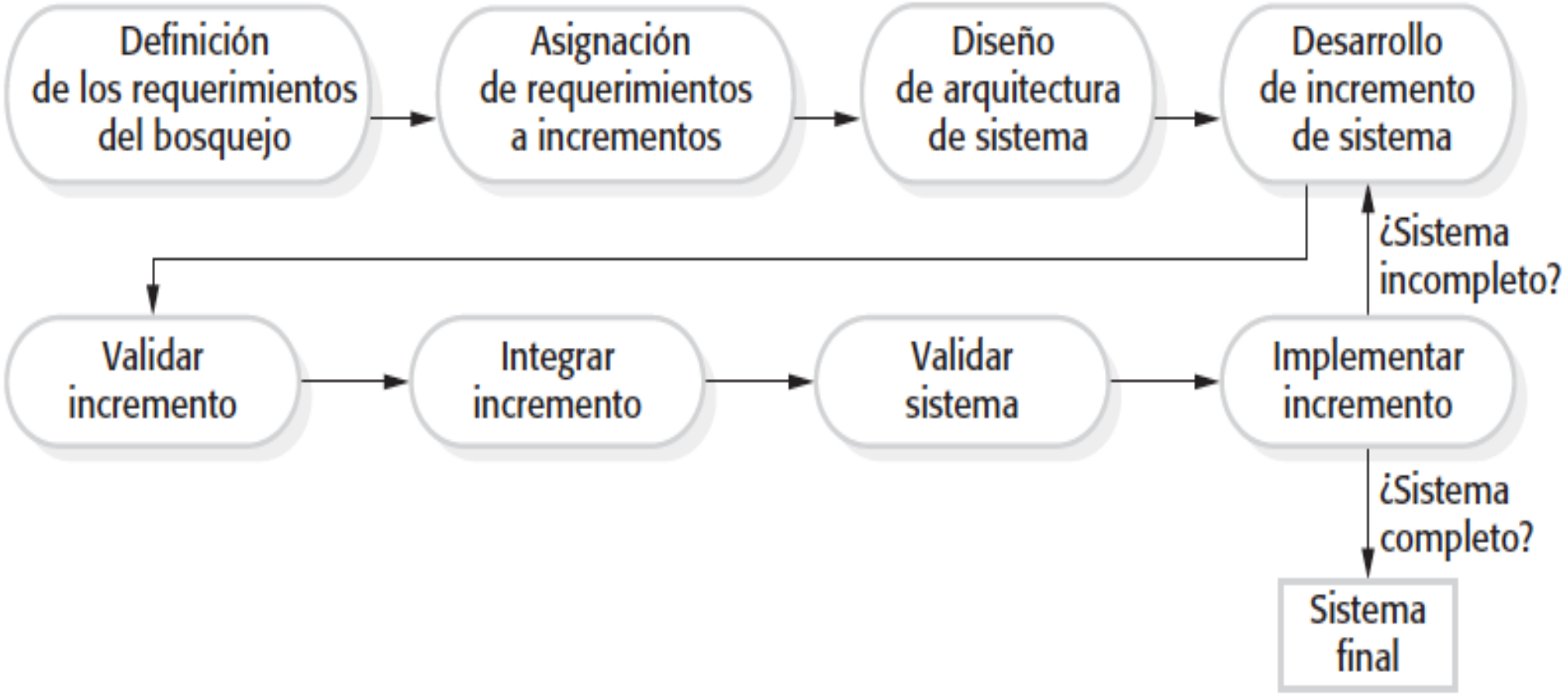
Desarrollo incremental o evolutivo

- Se diseña una implementación inicial y se la expone al comentario del usuario.
- Entrelaza las actividades de especificación, desarrollo y validación.
- Se desarrollan diversas versiones hasta producir un sistema adecuado.

Desarrollo incremental o evolutivo

- Se refina basándose en las peticiones del cliente
- Se avanza en serie de pasos hacia una solución.
- Parte fundamental de los enfoques ágiles.
- Útil para sistemas personales, de comercio electrónico y empresariales.





Entrega incremental

Ventajas y desventajas del modelo incremental

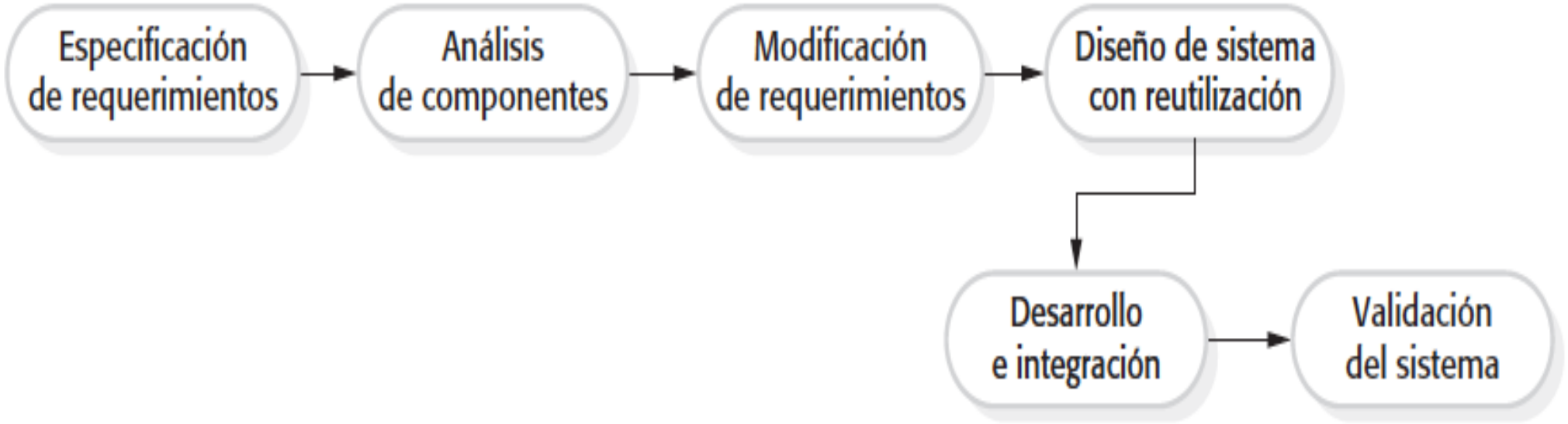
✓Más fácil y barato realizar cambios en el software.

✓ El cliente puede evaluar el desarrollo del sistema en una etapa relativamente temprana.

✓ Es posible que sea más rápida la entrega e implementación de partes del sistema al cliente.

✗El proceso no es visible. Entregas regulares miden el avance.

✗La estructura del sistema tiende a degradarse conforme se tienen nuevos incrementos.

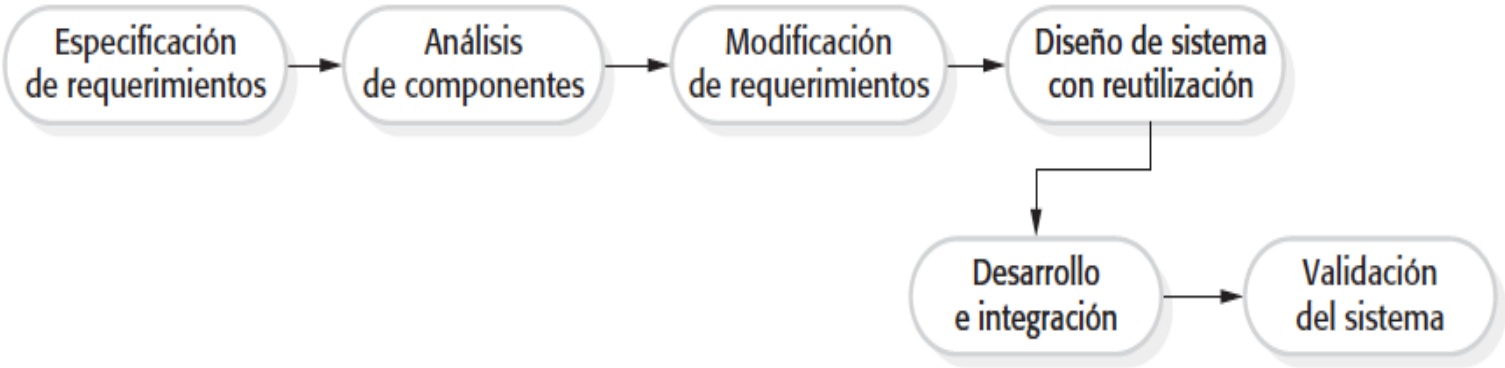


Basado en componentes (orientado a la reutilización)

Se basa en la existencia de un número significativo de componentes reutilizables.

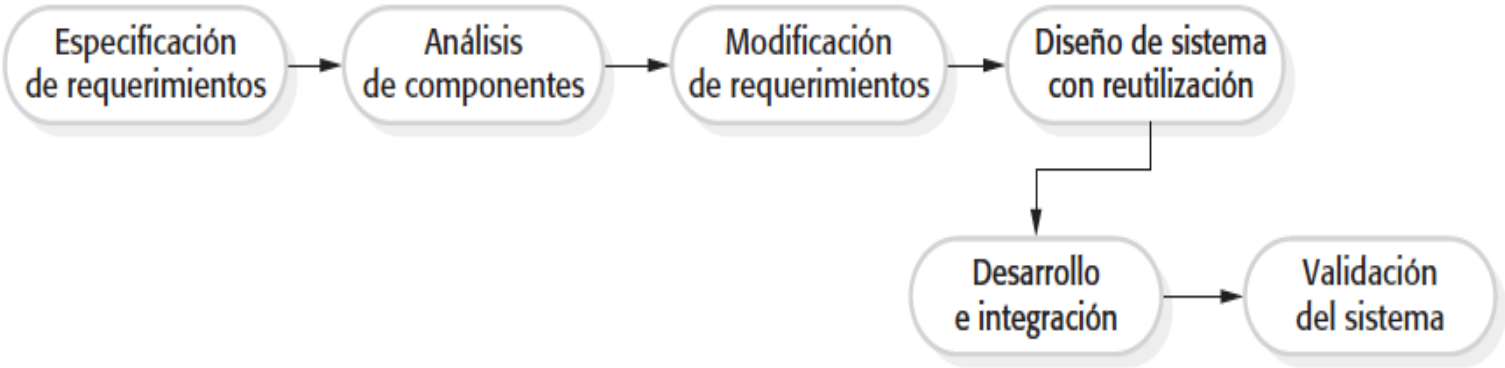
Modelo basado en componentes

1. Especificación de requerimientos
2. Análisis de Componentes
   - Se realiza una búsqueda de componentes para implementar dicha especificación.
3. Modificación de requerimientos
   - Se analizan los requerimientos usando la información de los componentes encontrados, luego se modifican.



Modelo basado en componentes

4. Diseño de sistema con reutilización
   - Se diseña el marco conceptual. Es posible que deba diseñarse algo de software nuevo.
5. Desarrollo e integración.
   - Se desarrolla el SW nuevo y se integran los componentes.
6. Se valida el sistema



Modelo basado en componentes

- Componentes de software reutilizables:
  - Servicios Web que se desarrollan y están disponibles para invocación remota.
  - Colecciones de objetos que se desarrollan como paquetes.
  - Sistemas de software independientes que se configuran para usar en un entorno particular.

Ventajas y desventajas del modelo basado en componentes

✓ Reduce la cantidad de software a desarrollar.

✓ Conduce a una entrega rápida de software.

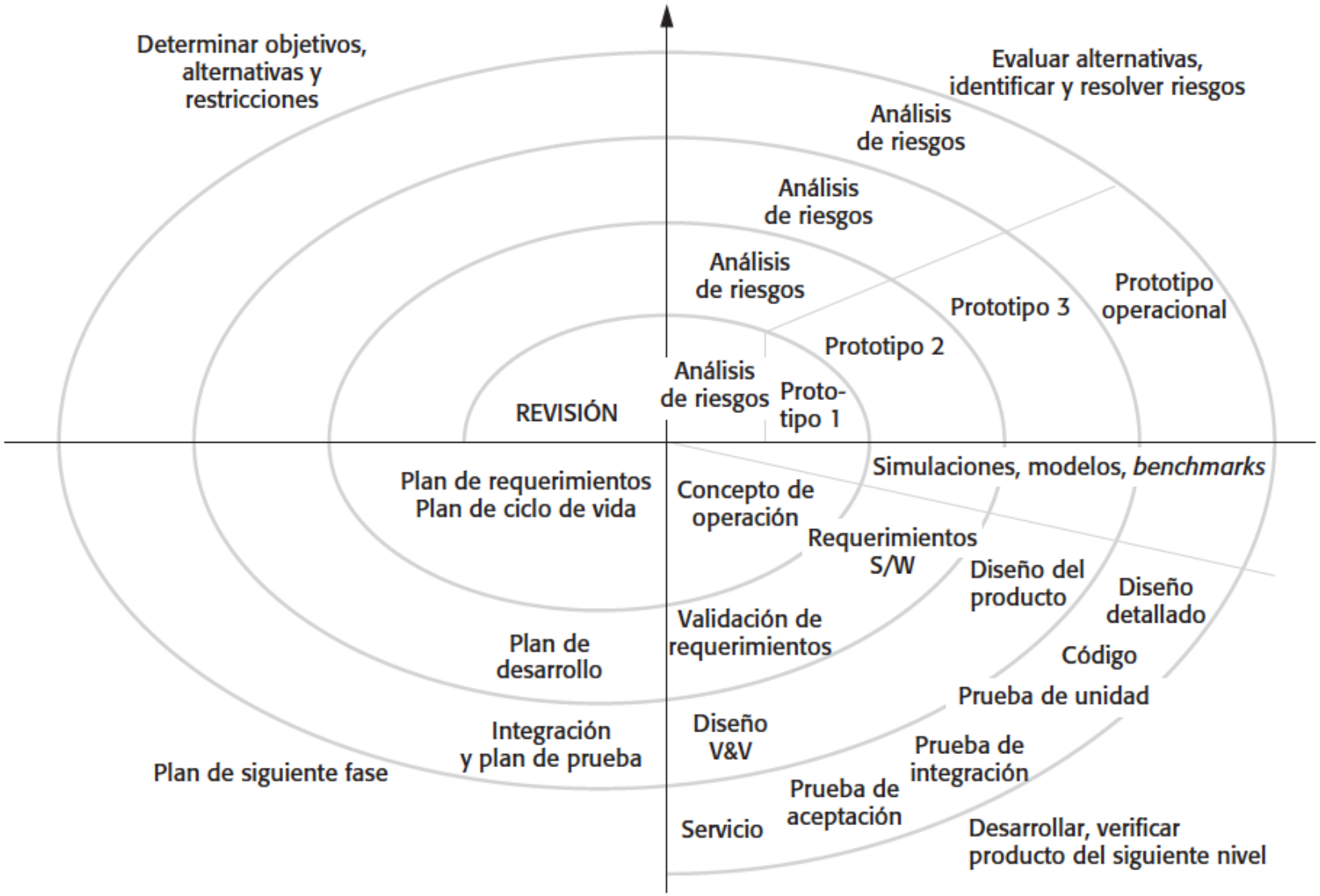
✗ Compromiso de requerimientos, puede que un sistema no cubra las necesidades reales del cliente.

✗ Se pierde el control sobre la evolución del sistema.

Modelo en espiral de Boehm

- Es un marco del proceso de software dirigido por el riesgo.
  - El riesgo es algo que puede ir mal y originan problemas en el proyecto.
- Representa el proceso de software como una espiral.
- Cada ciclo de la espiral se divide en cuatro sectores:
  1. Definición de objetivos.
  2. Evaluación y reducción de riesgos.
  3. Desarrollo y validación.
  4. Planificación.

Modelo en espiral de Boehm



La diferencia principal con otros modelos de proceso del software es la consideración explícita del riesgo.

Ventajas y desventajas del modelo en espiral de Boehm

✓El desarrollador y el cliente comprenden y reaccionan mejor ante riesgos

✓ Permite aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución.

✓ Ayuda a reducir riesgos antes de que se conviertan en problemas.

✗Es un poco complejo por lo que no se recomienda en sistemas pequeños.

✗Genera mucho tiempo en el desarrollo del sistema.

✗Requiere experiencia en la identificación de riesgos. (Modelo costoso)

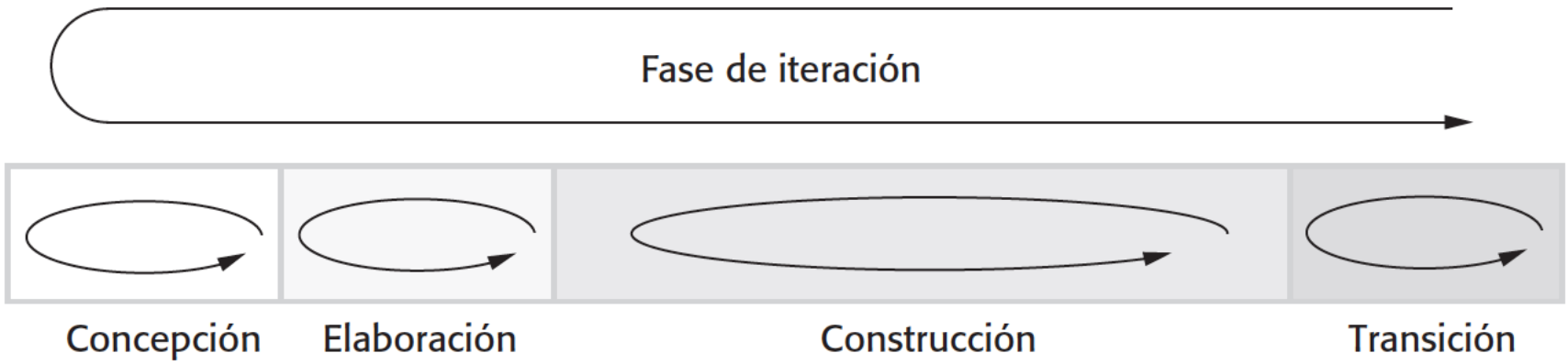
Modelo Proceso Unificado Racional (RUP)

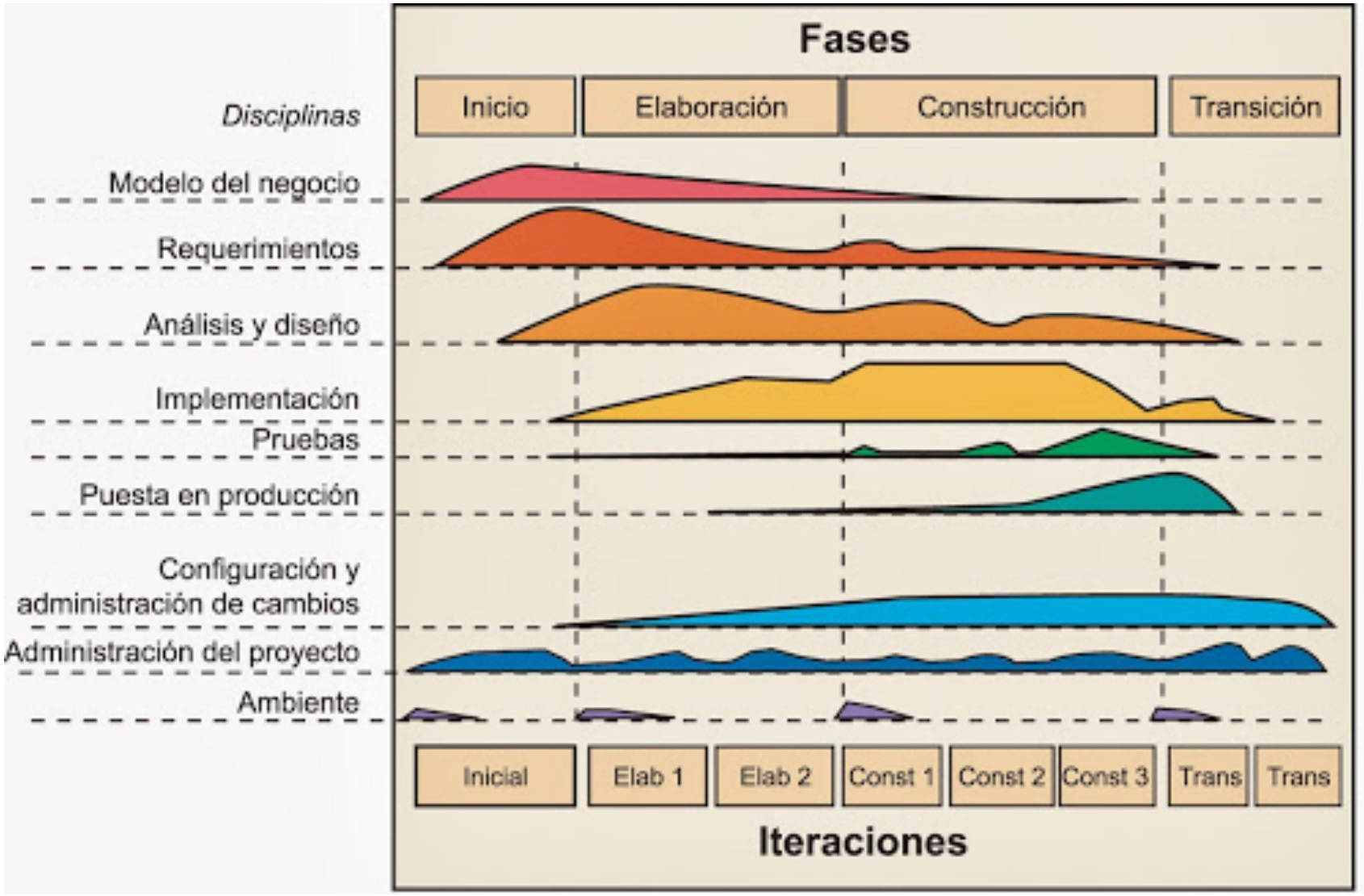
- Proceso impulsado por el caso de uso, centrado en la arquitectura, iterativo e incremental.
- Modelo híbrido: es un intento de obtener los mejores rasgos y características de los modelos tradicionales.

Modelo Proceso Unificado Racional

Se diseño en conjunto con UML, tiene cuatro fases:

1. Concepción
2. Elaboración
3. Construcción
4. Transición





Esquema RUP

- Fases
- Disciplinas
- Iteraciones
- Roles
- Actividades
- Artefactos

Gráfica de RUP tomada de Rational Unified Process. Rational Software Corp. 2001

Ventajas y desventajas del modelo proceso unificado racional

✓Reducción de riesgos en el proyecto.

✓ Garantía de calidad.

✓ Integración entre el desarrollo y mantenimiento.

✗Requiere previsión sobre lo que va a ocurrir, generando trabajo y costos adicionales de comunicación y documentación.

✗No suele ser práctico para pequeños proyectos.

Bibliografía y Recursos útiles

- Sommerville, I. (2011). Ingeniería de Software. 9na Edición. (Cap. 1, Cap. 2).
- Pressman, R. (2010). Ingeniería del Software. Un Enfoque Práctico. 7ma Edición. (Cap. 1, Cap. 2, Cap. 4)
- https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/CASE/CASE-classification.html