# Glosario

- Paradigmas del ciclo de vida.
  - Ciclo de vida en Cascada.
  - Paradigma de la construcción por incrementos
  - Paradigma de la construcción de prototipos.
  - Uso de técnicas de cuarta generación.
  - Paradigma del modelo en espiral.
- Desarrollo ágil.
  - Modelado Ágil
  - Programación Extrema.

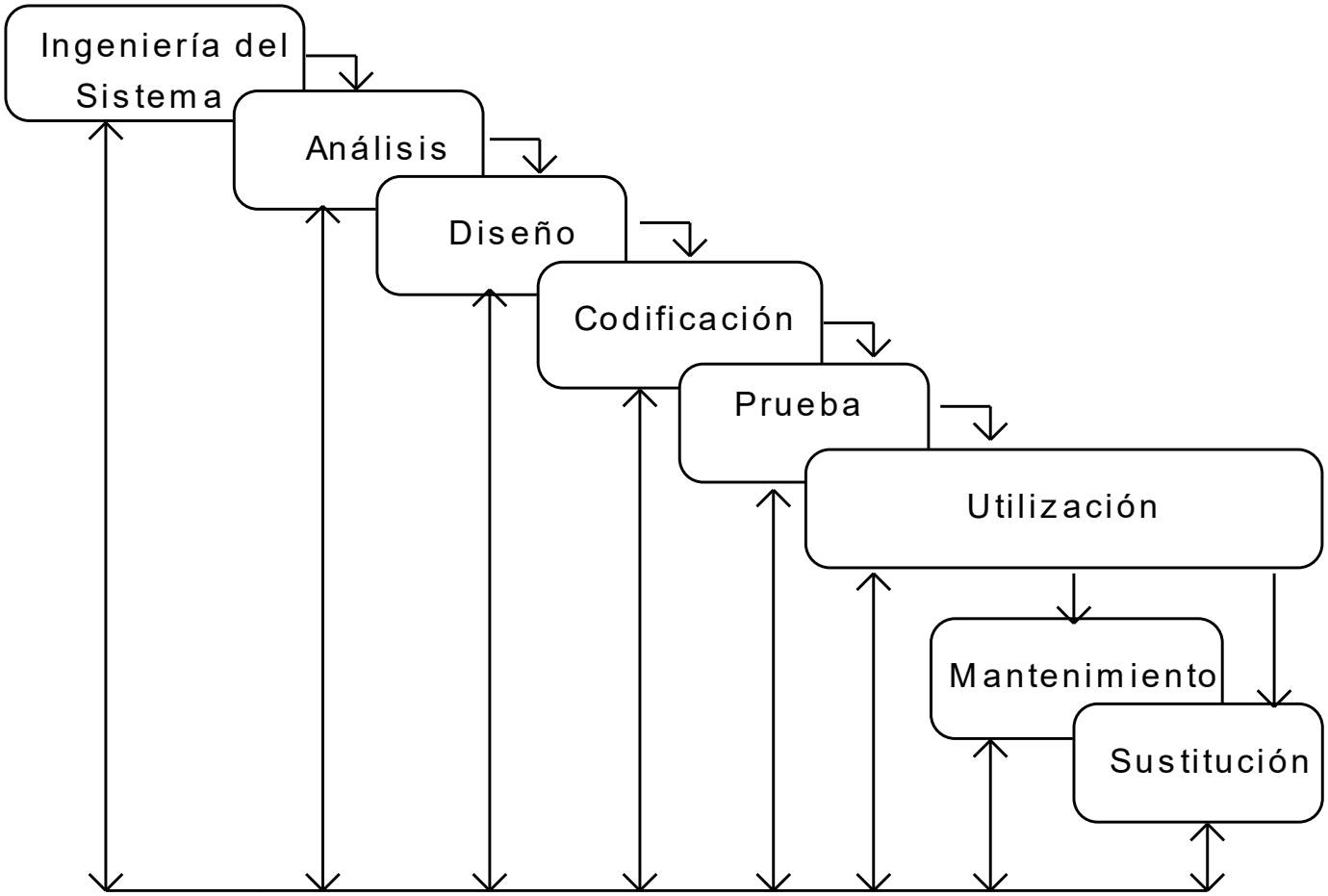
# Ciclo de vida

- La sucesión de etapas por las que pasa el software desde que un nuevo proyecto es concebido hasta que se deja de usar.
- Cada una de estas etapas lleva asociada una serie de tareas que deben realizarse, y una serie de documentos (en sentido amplio: software) que serán la salida de cada una de estas fases y servirán de entrada en la fase siguiente

# Ciclo de vida en **CASCADA**

- El más antiguo (Clásico).
- Incluye toda la vida del producto: desarrollo, pruebas, uso y mantenimiento
- Enfoque sistemático, disciplinado y SECUENCIAL
  - Cada fase empieza cuando se ha terminado la fase anterior.
  - Para pasar de una fase a la siguiente es preciso conseguir todos los objetivos de la anterior.
    - Entrega de documentos (en un sentido amplio)

# Ciclo de vida en cascada



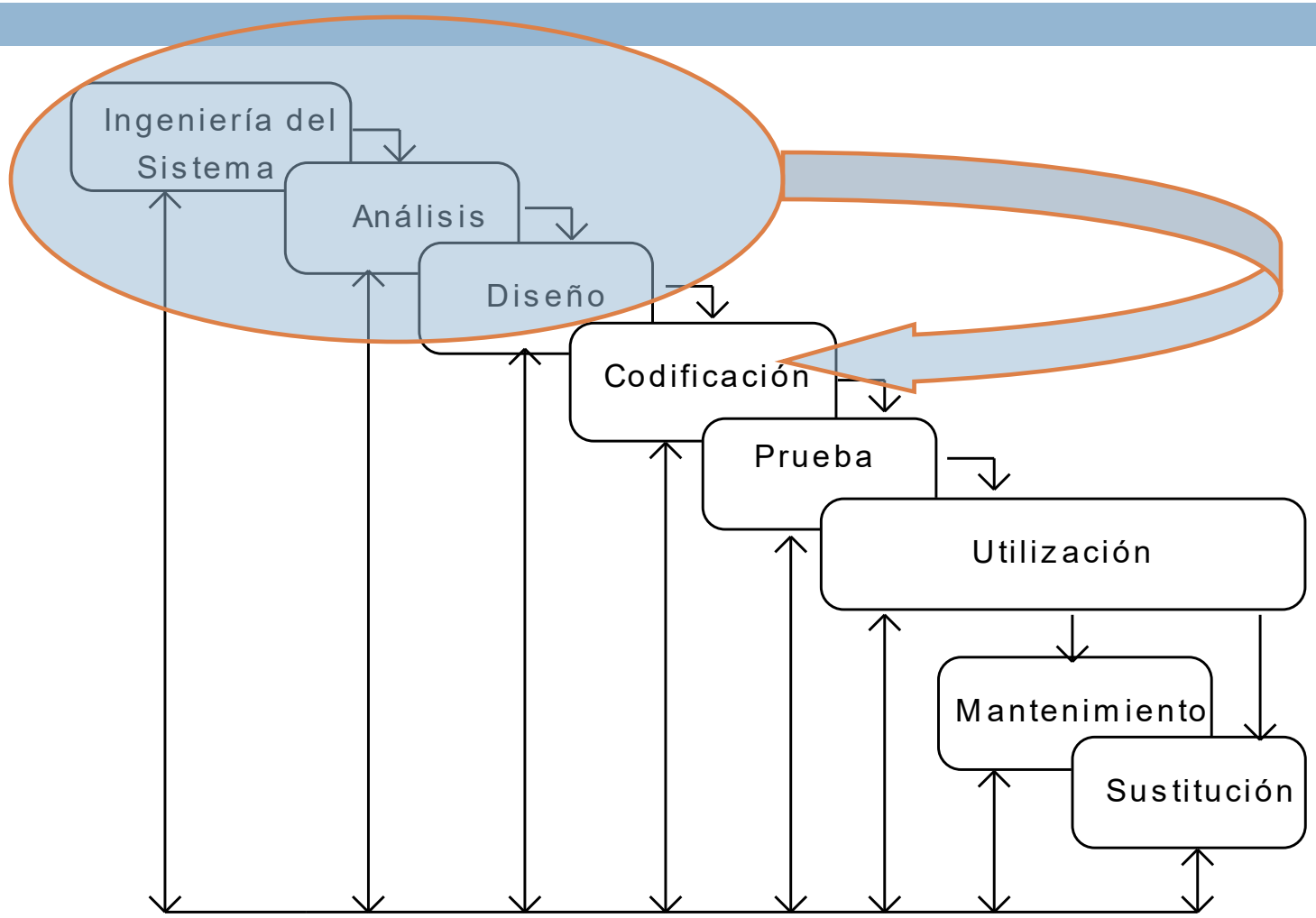
# Documentos del Modelo en Cascada

| Proceso | Documentos Producidos |
|---|---|
| Especificación del Sistema. | Especificación Funcional. Arquitectura del sistema. Plan de Pruebas de Aceptación del sistema. |
| Análisis y definición de Requisitos | Documento de Requisitos. Plan de pruebas de aceptación de la aplicación. |
| Diseño de la Arquitectura del software | Especificación de la Arquitectura, y Plan de Pruebas de la aplicación |
| Diseño de Interfaces | Especificación de la Interfaces y Plan de pruebas de Integración. |
| Diseño Detallado | Especificación del diseño y Plan de prueba de Unidades. |
| Codificación | Código de Programa |
| Prueba de Unidades | Informe de pruebas de unidad |
| Prueba de Módulos | Informe de pruebas de módulo |
| Prueba de Integración | Informe de prueba de integración y Manual de usuario final |
| Prueba del Sistema | Informe de prueba del sistema |
| Prueba de Aceptación | Sistema final más la documentación. |

# Ciclo de vida en cascada

- Ingeniería y análisis del sistema:
  - Definir la interrelación del software con otros elementos del sistema más complejo en el que está englobado
- Análisis de requisitos:
  - Análisis detallado de los componentes software: datos a manejar, funciones y comportamientos a desarrollar, interfaces, restricciones que se deben cumplir.
- Diseño:
  - Estructura de datos, arquitectura de aplicaciones, estructura interna de los programas y las interfaces.

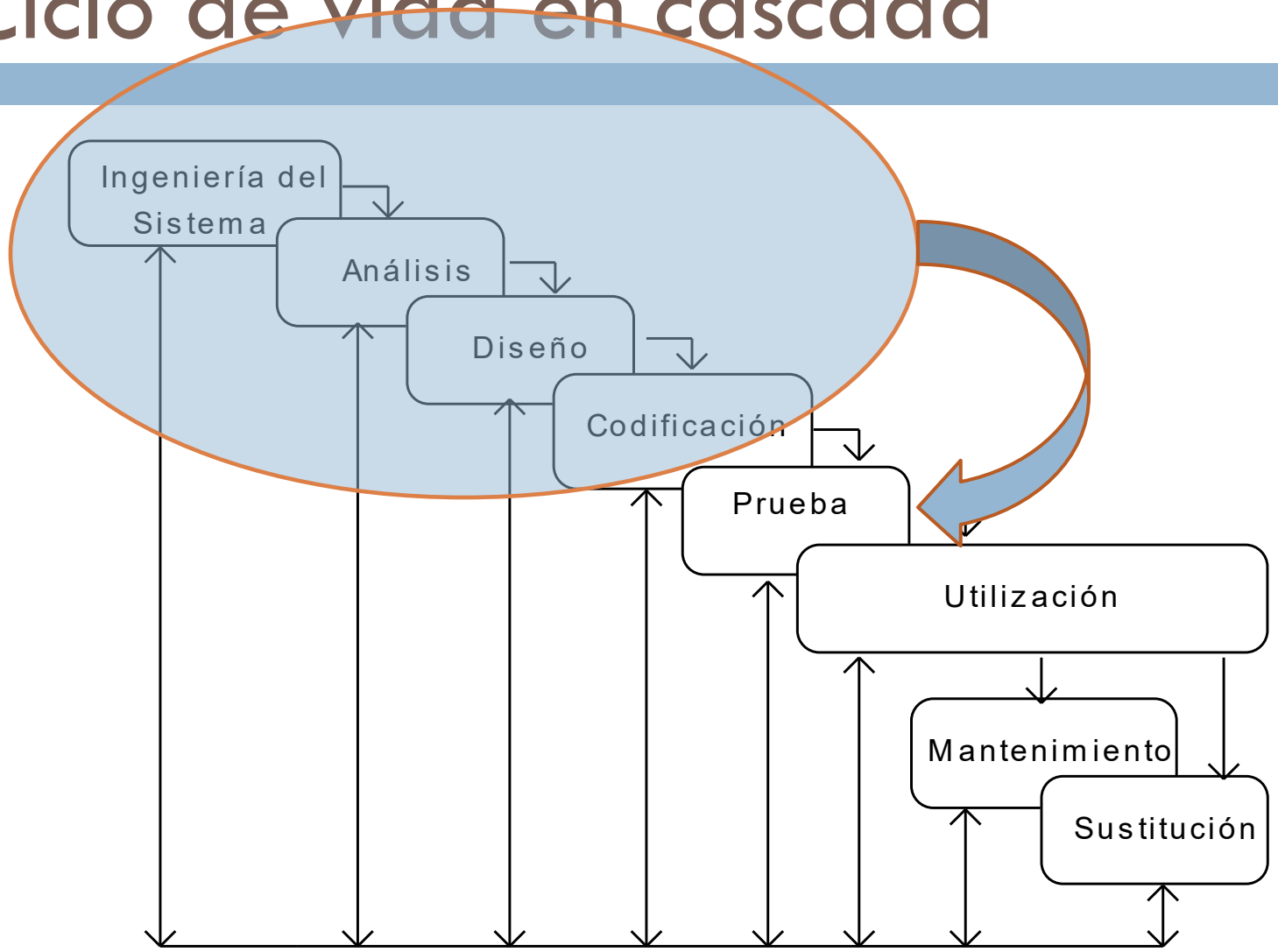
# Ciclo de vida en cascada



# Ciclo de vida en cascada

- Codificación
  - La codificación consiste en la traducción del diseño a un formato que sea legible para la máquina, que se compila y produce un programa ejecutable.

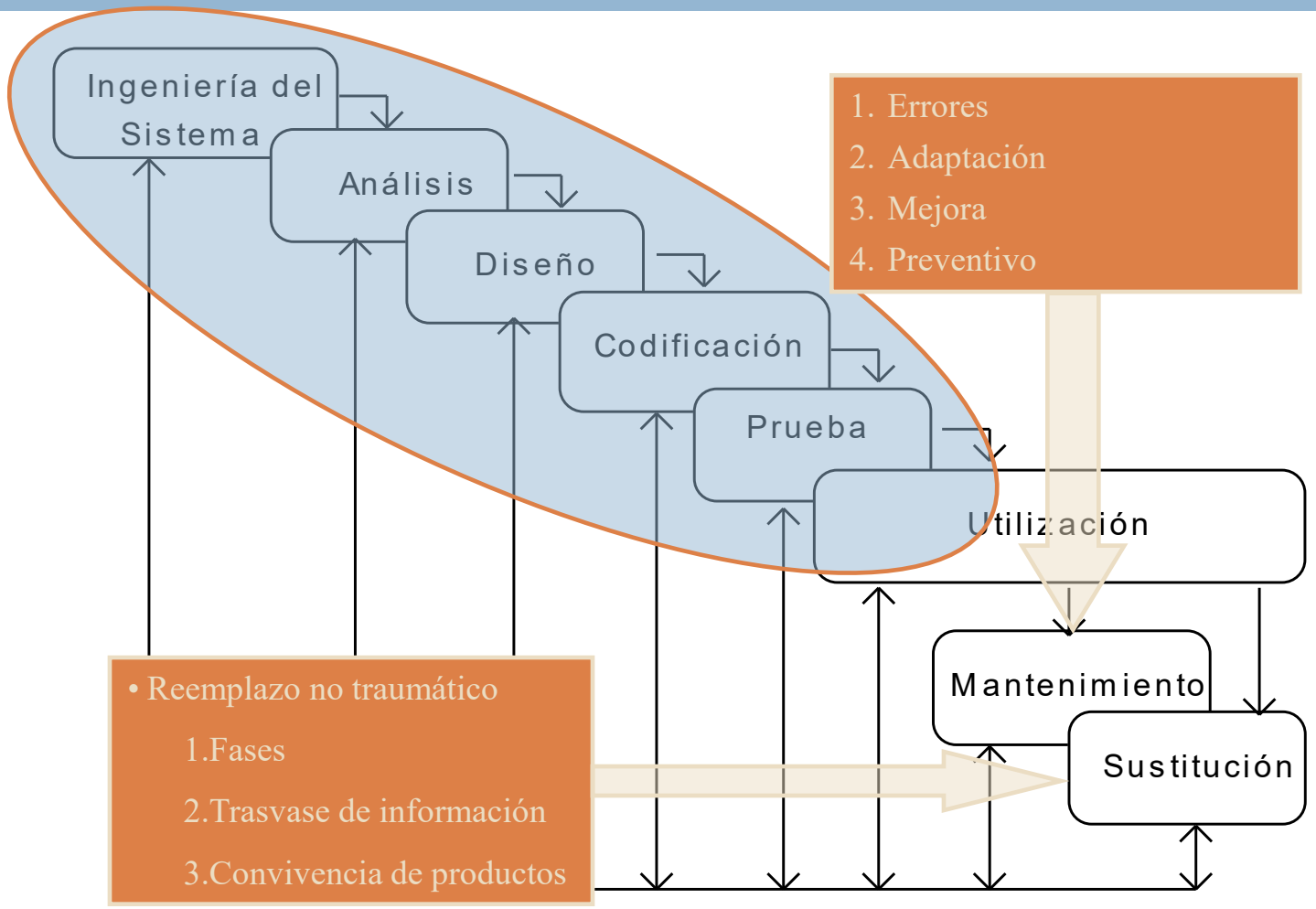
# Ciclo de vida en cascada



# Documentos del Modelo en Cascada

| Proceso | Documentos Producidos |
|---|---|
| Especificación del Sistema. | Especificación Funcional. Arquitectura del sistema. Plan de Pruebas de Aceptación del sistema. |
| Análisis y definición de Requisitos | Documento de Requisitos. Plan de pruebas de aceptación de la aplicación. |
| Diseño de la Arquitectura del software | Especificación de la Arquitectura, y Plan de Pruebas de la aplicación |
| Diseño de Interfaces | Especificación de la Interfaces y Plan de pruebas de Integración. |
| Diseño Detallado | Especificación del diseño y Plan de prueba de Unidades. |
| Codificación | Código de Programa |
| Prueba de Unidades | Informe de pruebas de unidad |
| Prueba de Módulos | Informe de pruebas de módulo |
| Prueba de Integración | Informe de prueba de integración y Manual de usuario final |
| Prueba del Sistema | Informe de prueba del sistema |
| Prueba de Aceptación | Sistema final más la documentación. |

# Ciclo de vida en cascada



# Ciclo de vida en cascada

- Aportaciones
  - Es el más simple, el más conocido y el más fácil de usar.
  - Define una serie de fases o procesos a realizar que posteriormente se formalizaron en las normas.
  - Permite generar software eficientemente y de acuerdo con las especificaciones.
    - Ayuda a prevenir que se sobrepasen las fechas de entrega y los costes esperados
    - Al final de cada fase el personal técnico y los usuarios tienen la oportunidad de revisar el progreso del proyecto.

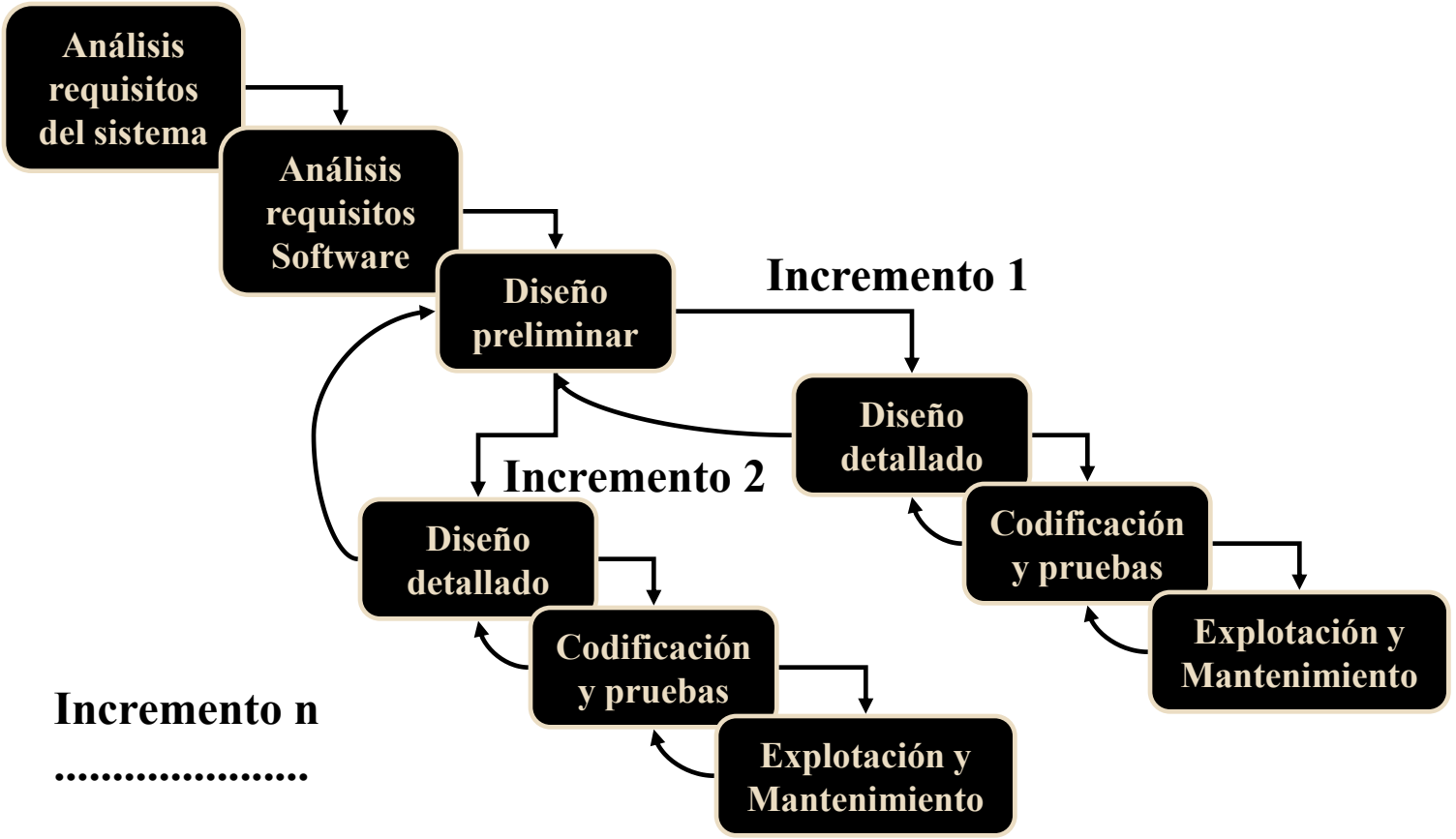
# Ciclo de vida en cascada

- Problemas/Críticas
  - No siempre se pueden establecer los requisitos del sistema desde el primer momento
  - Hasta el final no hay una versión operativa del programa por lo que es al final de la fase de codificación cuando se detectan la mayoría de los fallos
  - En la realidad el ciclo de vida no es secuencial, sino que hay iteraciones, exige refinamiento.
  - Estados de bloqueo, porque hay partes que tienen que esperar por otras.
  - Acentúa el fracaso de la I.S. con el usuario final. El sistema no estará en funcionamiento hasta la fase final.

# Ciclo de vida en cascada

- Problemas/Críticas
  - **No siempre se pueden establecer los requisitos del sistema desde el primer momento**
  - **Hasta el final no hay una versión operativa del programa por lo que es al final de la fase de codificación cuando se detectan la mayoría de los fallos**
  - **En la realidad el ciclo de vida no es secuencial, sino que hay iteraciones, exige refinamiento.**
  - Estados de bloqueo, porque hay partes que tienen que esperar por otras.
  - Acentúa el fracaso de la I.S. con el usuario final. El sistema no estará en funcionamiento hasta la fase final.

# Desarrollo en **INCREMENTOS**

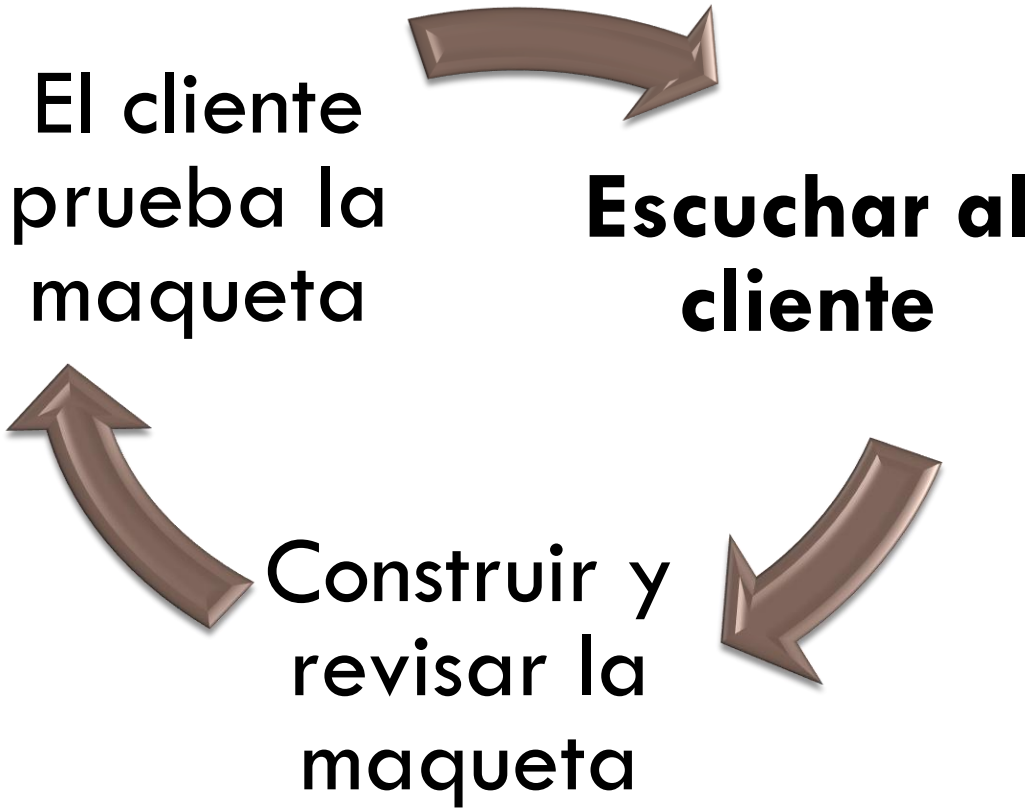


# Desarrollo de incrementos

- Avances
  - El software no se piensa como una unidad monolítica sino la integración de resultados sucesivos.
  - Adecuado a entornos con relativa incertidumbre.
- Ventajas
  - Modelo iterativo.
  - Mejora la comunicación con el cliente.
- Críticas al modelo de incrementos:
  - Dificultad de ver si los requisitos son válidos.
  - Los errores en los requisitos se detectan tarde.

# Construcción de **PROTOTIPOS**

- Dirigido por producto.



# Construcción de prototipos

- Tipos
  - Un prototipo, en papel (storyboard) o ejecutable en ordenador, que describa la interacción hombre-máquina.
  - Un programa que realice en todo o en parte la función deseada pero que tenga características (rendimiento, consideración de casos particulares, etc.) que deban ser mejoradas durante el desarrollo del proyecto.
  - Un prototipo que implemente algún(os) subconjunto(s) de la función requerida, y que sirva para evaluar el rendimiento de un algoritmo o las necesidades de capacidad de almacenamiento y velocidad de cálculo del sistema final.

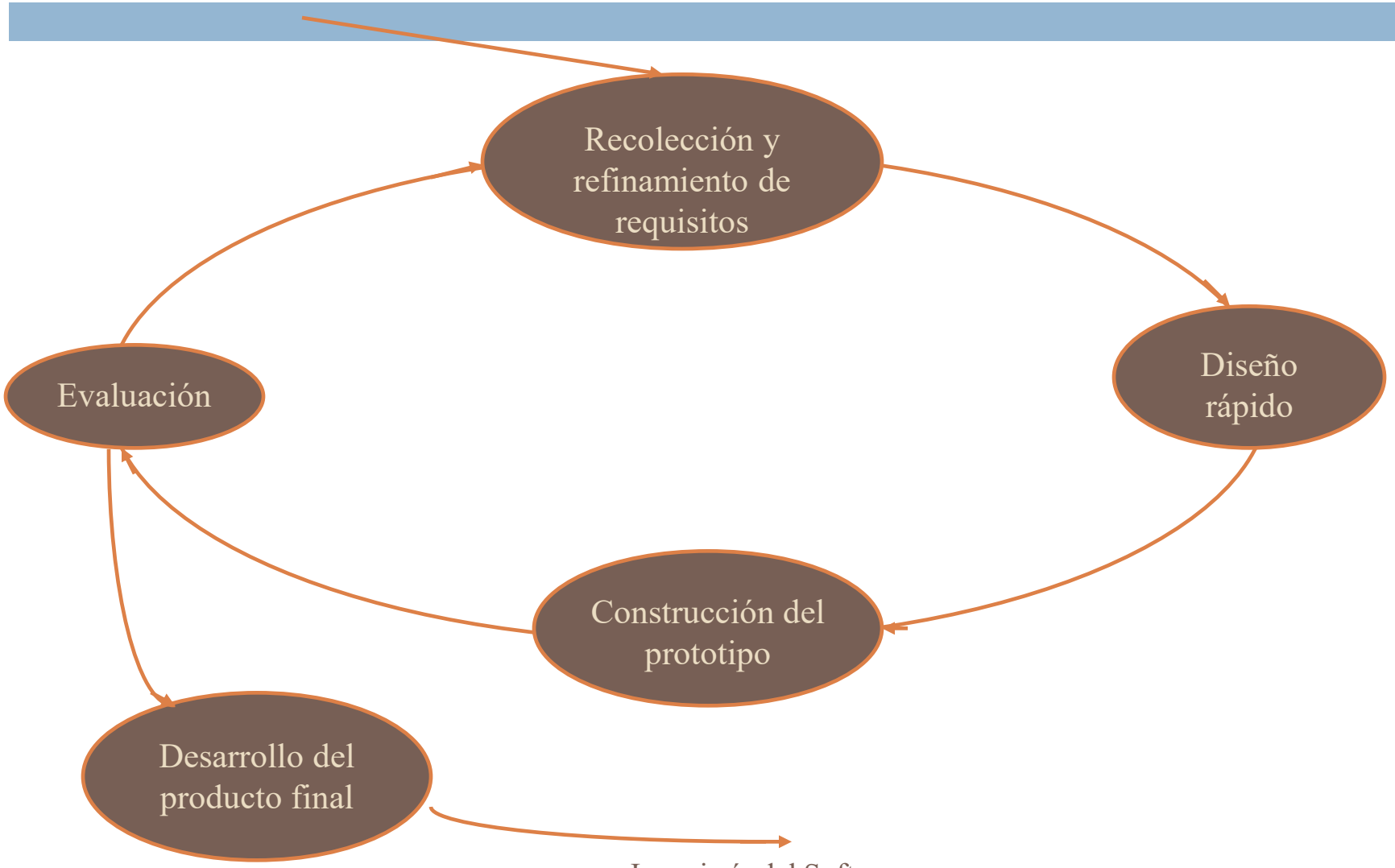
# Construcción de prototipos: ejemplo

- *Con un video explicativo como **producto mínimo viable**, un producto puede ser explicado claramente en palabras sencillas y así ser presentado a una amplia audiencia - sin que este producto ya exista en su forma final.*
- *Dropbox ha demostrado cómo puede funcionar. Al principio, el servicio consistía solo en una versión de prueba con funciones mínimas. Un vídeo que transmitía las ventajas del servicio de compartición de archivos de una manera corta y original aumentó el número de registros de la noche a la mañana a 75.000. Y esto a pesar de que la verdadera solución de software aún no existía.*
- *https://www.youtube.com/watch?v=xPJoq_QVsY4*
- *https://www.youtube.com/watch?v=qxFLfY7_Gqw*

# Construcción de prototipos

- Elección: ¿Cuándo usaríamos prototipos?
  - Nivel muy alto de incertidumbre
  - Mucha interacción con el usuario, algoritmos refinables
  - Disposición del cliente a probar los prototipos.
- ¿Cuándo estaría desaconsejado?
  - Mucha Complejidad => Mal candidato
  - Problema bien comprendido

# Construcción de prototipos



# Construcción de prototipos

- Ventajas:
  - Podemos aprovechar trabajo
    - Refinamiento del diseño.
    - Diseño de pantallas e Informes.
    - Algoritmos y módulos probados.
  - Permite
    - refinar requisitos. Comprensión del problema
    - Analizar alternativas. Explorar soluciones
    - Analizar viabilidades. Verificar rendimientos
- Desventajas:
  - Dificulta la predicción fiable del coste
  - El prototipo se convierte en el resultado final
    - Se asumen elecciones apresuradas de arquitectura, plataforma de desarrollo y alternativas.
    - Gran cantidad de errores latentes.
    - Producto ineficiente.
    - Poco fiable.
    - Difícil de mantener.

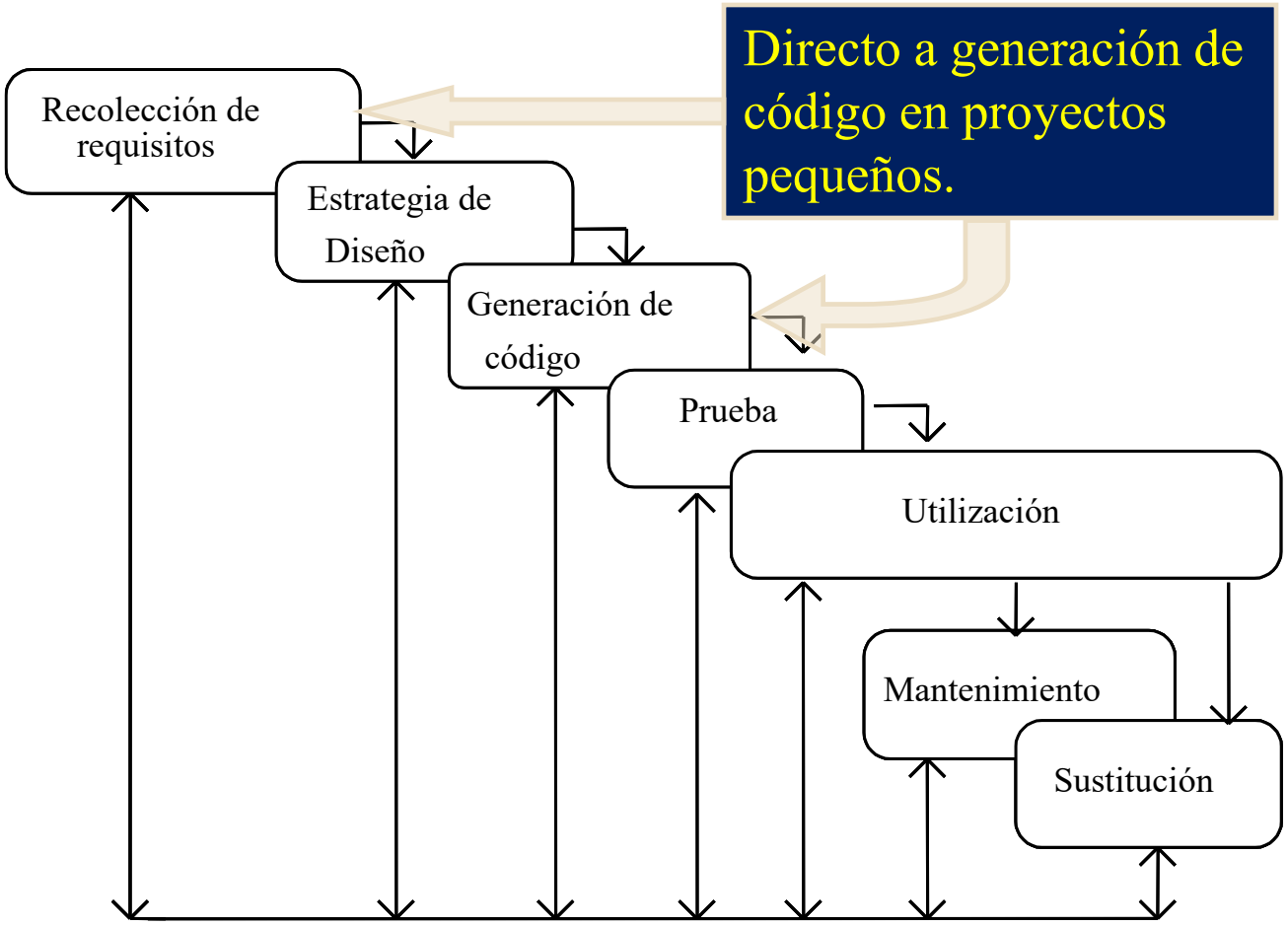
# Técnicas de 4ª GENERACIÓN

- Incluyen todas o algunas de las herramientas:
  - Acceso y definición de BD utilizando lenguajes de consulta de alto nivel (derivados de SQL, QBE).
  - Interacción y definición de pantallas.
  - Generación de informes.
  - Análisis de datos.
    - Capacidad de hojas de cálculo.
    - Capacidades gráficas de alto nivel.
    - Algoritmos parametrizables de análisis
  - Generación de código. Gran evolución.

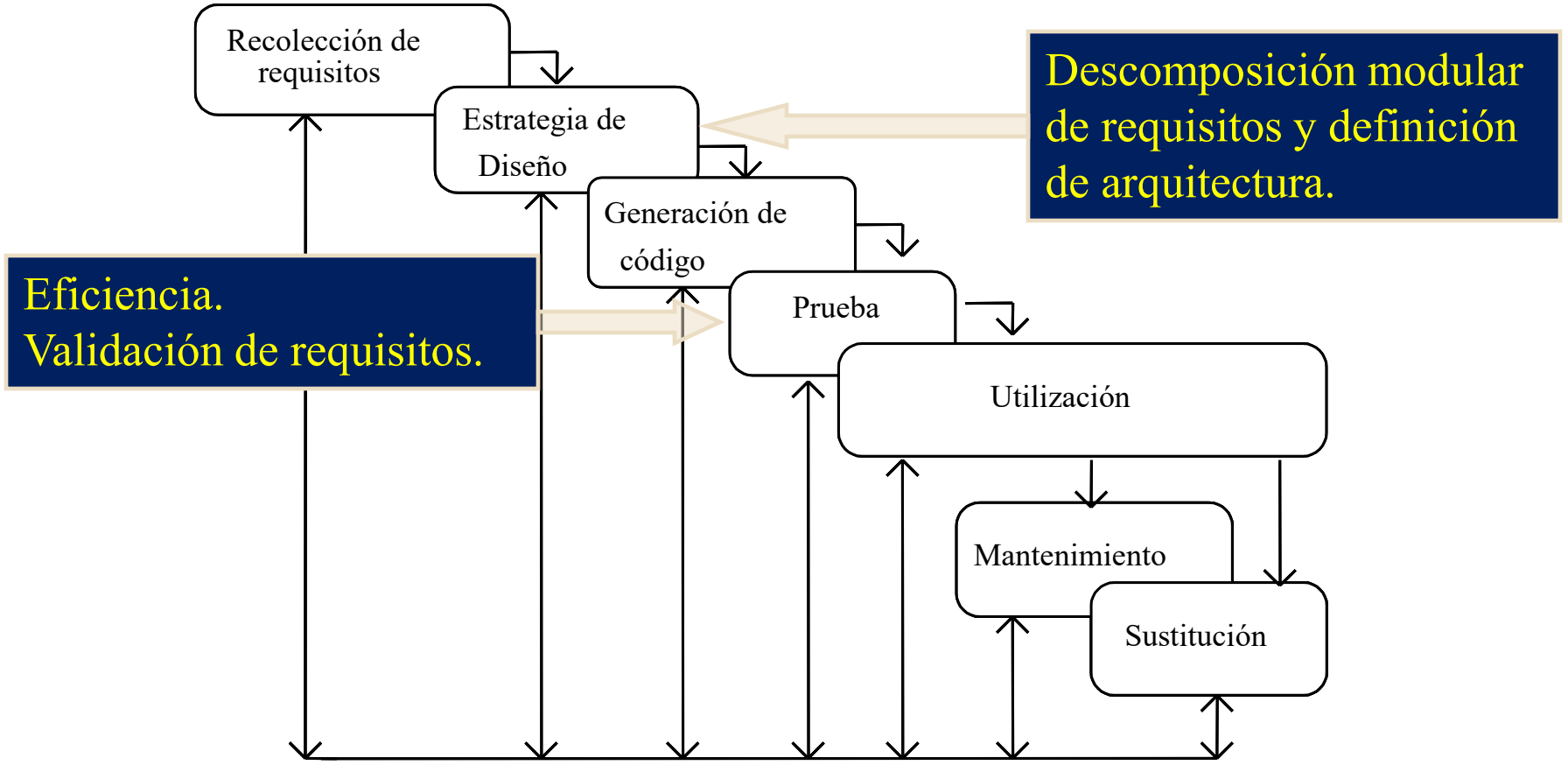
# Técnicas de 4ª generación: ejemplo

- Scrach, lenguaje de programación para niños:
  - https://scratch.mit.edu/projects/430007515/editor/
- Tabnine:
  - https://www.tabnine.com/?mc_cid=271e704263&mc_eid=db
- Copilot de github:
  - https://github.com/features/copilot?mc_cid=271e704263&mc_

# Técnicas de 4ª generación



# Técnicas de 4ª generación



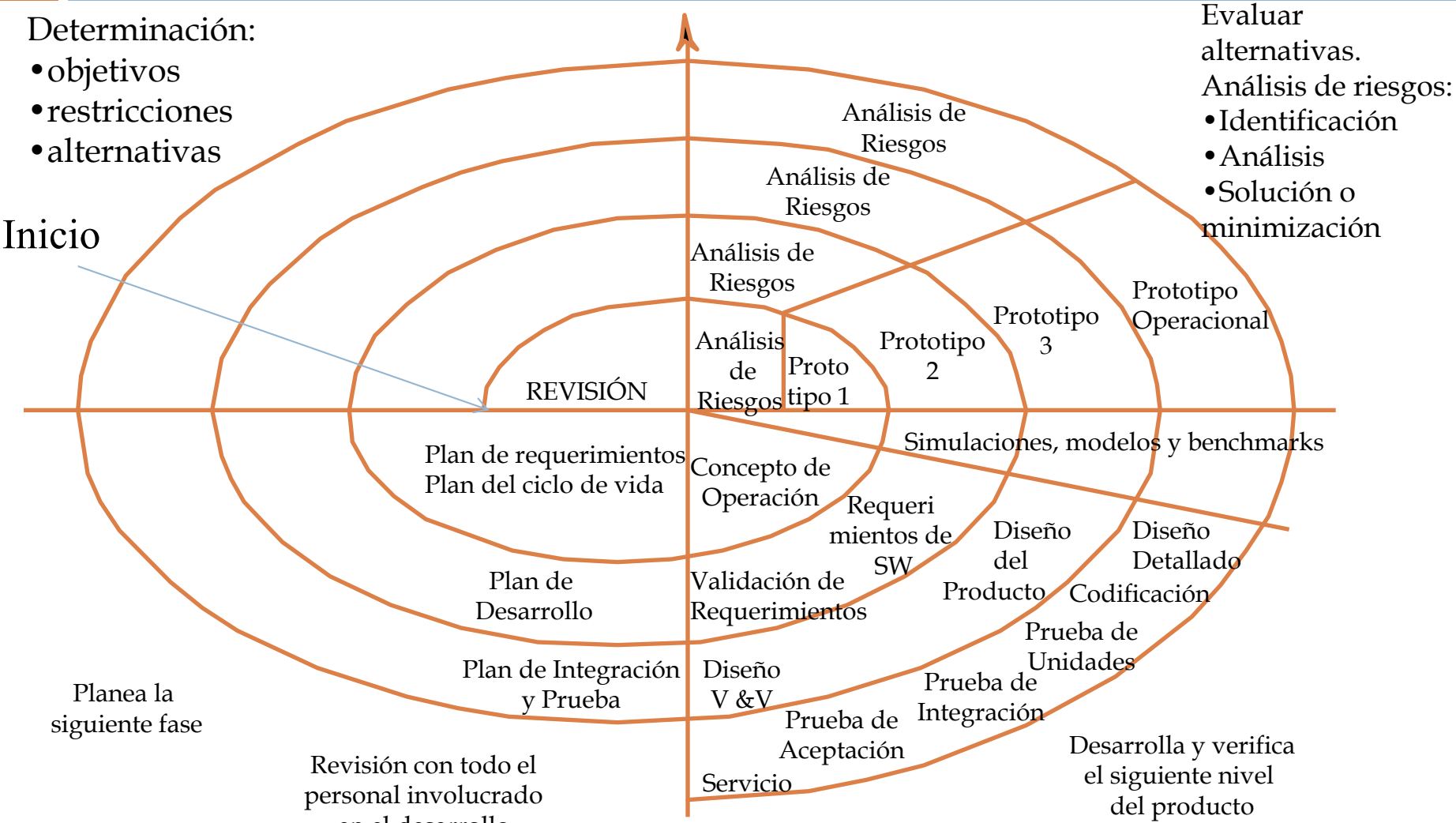
# Técnicas de 4ª generación

- Consideraciones:
  - Pueden reducir significativamente el tiempo.
    - Algunos estudios invierten esta tendencia debido a la necesidad de revisar y ajustar las propuestas de la IA
  - Soluciones sobre problemas bien conocidos.
    - Gestión, web
- Críticas:
  - El código generado puede usar funciones obsoletas, ser oscuro, ineficiente o tener alucinaciones generando resultados imprevistos.
  - Mantenimiento cuestionable.

# Modelo en **ESPIRAL**

- Proporciona un modelo evolutivo para el desarrollo de sistemas de software complejos
  - Más realista que el ciclo de vida clásico.
  - Mantiene el enfoque sistemático del desarrollo secuencial.
  - Permite la utilización de prototipos, pero cada iteración implica la generación de un producto.
  - Incorpora análisis de **riesgos**.
    - Define estrategias para la reducción de riesgos (Prototipos).
    - Permite acabar en fases tempranas el desarrollo de un producto final demasiado costoso o arriesgado
- Cascada + Prototipos

# Modelo en Espiral



# Modelo en Espiral

- Se divide en actividades estructurales o regiones neutras
  - Definición de objetivos
    - Objetivos específicos, restricciones y alternativas.
  - Análisis de riesgo
    - Identificación, análisis detallado y búsqueda de estrategias que los minimicen.
  - Desarrollo y validación:
    - Se centra en la generación de los entregables de los procesos clásicos.
  - Revisión y Planificación:
    - Valoración de resultados y planificación de la ejecución del siguiente ciclo.

# Ejemplo de Modelo en Espiral

## Elaboración de un catálogo.

- Objetivos
  - Desarrollar un catálogo de componentes de software
- Restricciones.
  - A un año.
  - Debe soportar los tipos de componentes existentes.
  - Costo total menor de 100.000 €.
- Alternativas.
  - Comprar software de captura de información.
  - Comprar bases de datos y desarrollar el catálogo utilizando la BD.
  - Desarrollar catálogo de propósito especial.

# Ejemplo de Modelo en Espiral

- Riesgos.
  - Puede ser imposible satisfacer las restricciones.
  - La funcionalidad del catálogo puede ser inapropiada.
- Solución de riesgos.
  - Relaja restricciones de tiempo.
  - Desarrolla un prototipo del catálogo (utilizando lenguajes de cuarta generación 4GL y una BD existente) para clarificar los requisitos.

# Ejemplo de Modelo en Espiral

- Resultados.
  - Los sistemas de captura de información son inflexibles. Los requisitos no pueden cumplirse.
  - El prototipo que utiliza la BD puede mejorarse para completar el sistema.
  - El desarrollo de un catálogo de propósito específico no es costeable.
- Planes.
  - Desarrolla el catálogo utilizando una BD existente mejorando el prototipo y la interfaz de usuario.

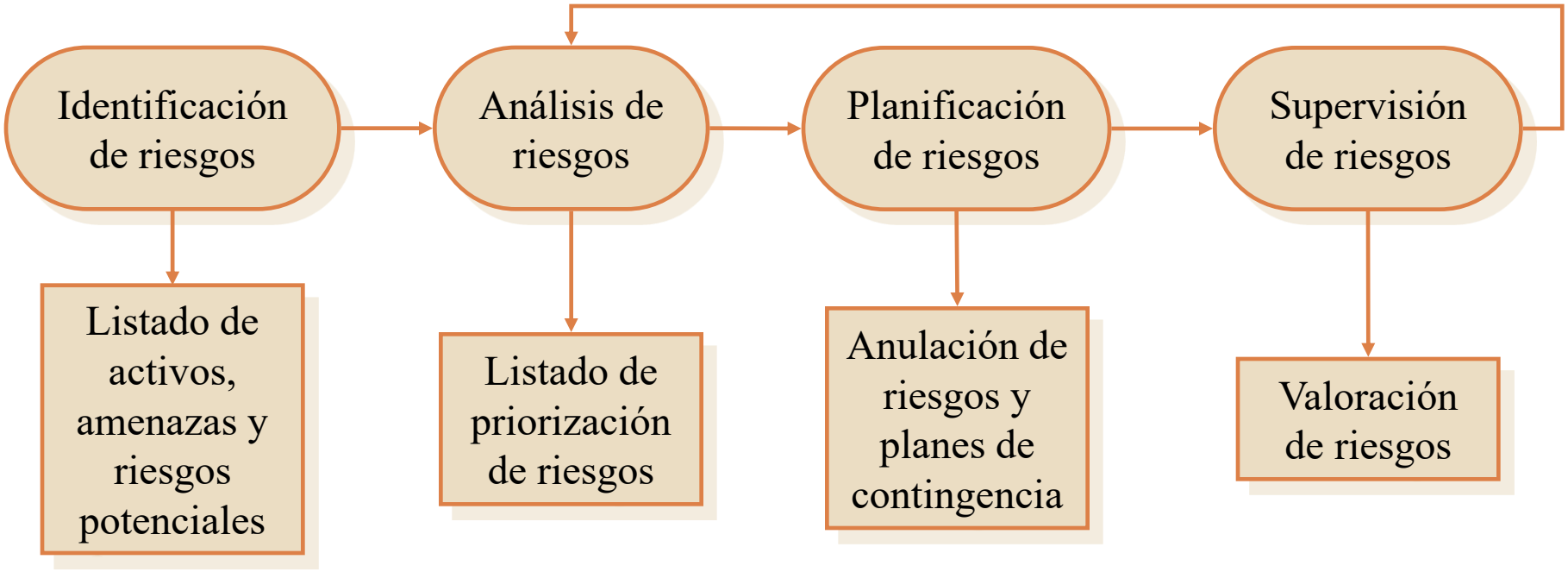
# Modelo en Espiral

- Limitaciones
  - Es difícil de adaptar a un contrato. Un planteamiento sería un contrato por iteración
  - Requiere habilidad en la gestión de riesgos
  - Un riesgo no detectado a tiempo equivale en coste a un requisito mal definido de los modelos secuenciales.
  - Puede ser difícil de controlar y de convencer al cliente de que es controlable
- Principal Aportación
  - Gestión del riesgo

- **QUÉ ES EL RIESGO**

# Modelo en Espiral → Análisis de riesgos

- Administración del riesgo:



# Análisis de riesgos

DEFINICIONES (Magerit):

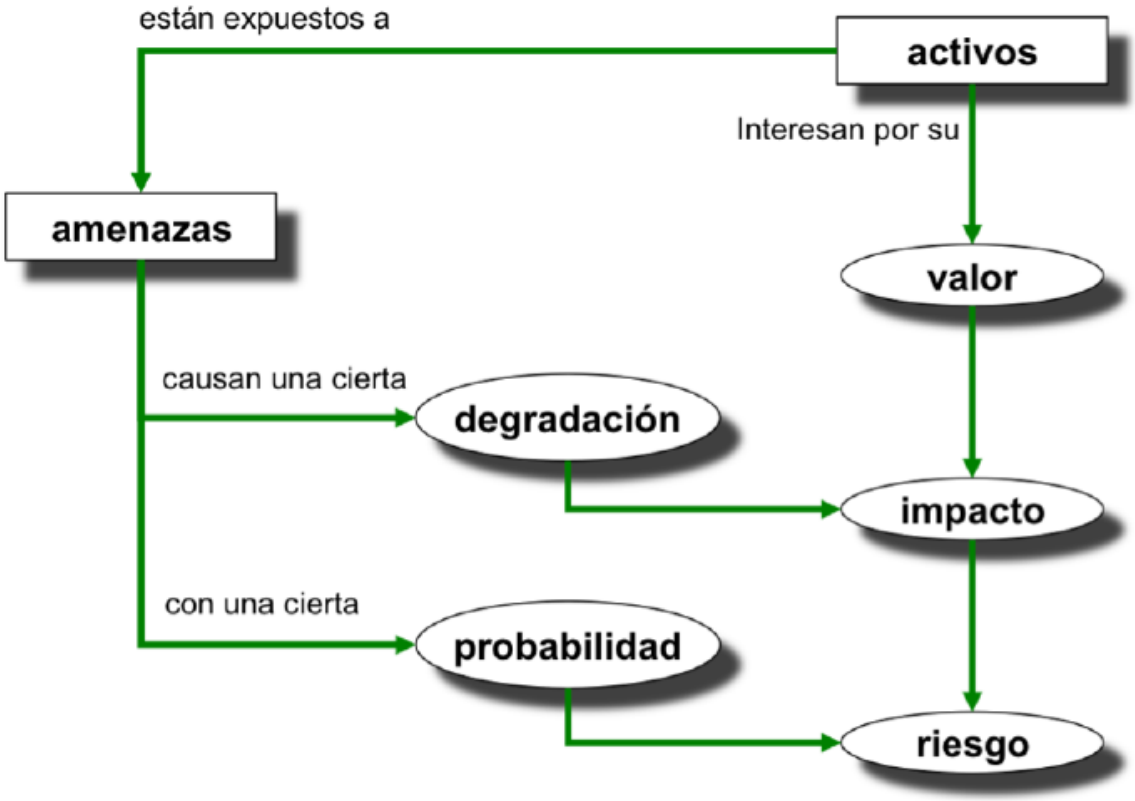
- **Activos,** son los elementos del sistema de información que soportan la misión de la Organización
- **Amenazas,** que son cosas que les pueden pasar a los activos causando un perjuicio a la Organización
- **Salvaguardas** (o contramedidas), que son medidas de protección desplegadas para que aquellas amenazas no causen [tanto] daño

# Análisis de riesgos

DEFINICIONES:

- **Riesgo:** estimación del grado de exposición a que una amenaza se materialice sobre uno o más activos causando daños o perjuicios a la Organización.
- **Análisis de riesgos:** proceso sistemático para estimar la magnitud de los riesgos a que está expuesta una Organización.
- **Proceso de gestión/administración de riesgos:** proceso destinado a modificar el riesgo.

# Análisis de riesgos




MAGERIT *Elementos del análisis de riesgos potenciales*

# Análisis de riesgos: Activos

- ACTIVOS

**Componente o funcionalidad** de un sistema de información **susceptible de ser atacado** deliberada o accidentalmente **con consecuencias para la organización.** [UNE 71504:2008].

# Análisis de riesgos

- ACTIVOS: La información y/o los servicios:
  - **Los soportes de información** que son dispositivos de almacenamiento de datos.
  - **El equipamiento auxiliar** que complementa el material informático.
  - **Las redes de comunicaciones** que permiten intercambiar datos.
  - **Las instalaciones** que acogen equipos informáticos y de comunicaciones.
  - **Las personas** que explotan u operan todos los elementos anteriormente citados.

# Análisis de riesgos

- Dimensiones **básicas** a calibrar de un activo:
  - su **confidencialidad**: ¿qué daño causaría que lo conociera quien no debe? Esta valoración es típica de datos.
  - su **integridad**: ¿qué perjuicio causaría que estuviera dañado o corrupto? Esta valoración es típica de los datos, que pueden estar manipulados, ser total o parcialmente falsos o, incluso, faltar datos.
  - su **disponibilidad**: ¿qué perjuicio causaría no tenerlo o no poder utilizarlo? Esta valoración es típica de los servicios

# Análisis de riesgos

- Otras dimensiones :
  - la **autenticidad:** ¿qué perjuicio causaría que usuario no fuera quién dice ser?
  - la **trazabilidad** del uso del servicio: ¿qué daño causaría no saber a quién se le presta tal servicio? Es decir, ¿quién hace qué y cuándo?
  - la **trazabilidad** del acceso a los datos: ¿qué daño causaría no saber quién accede a qué datos y qué hace con ellos?

# Análisis de riesgos

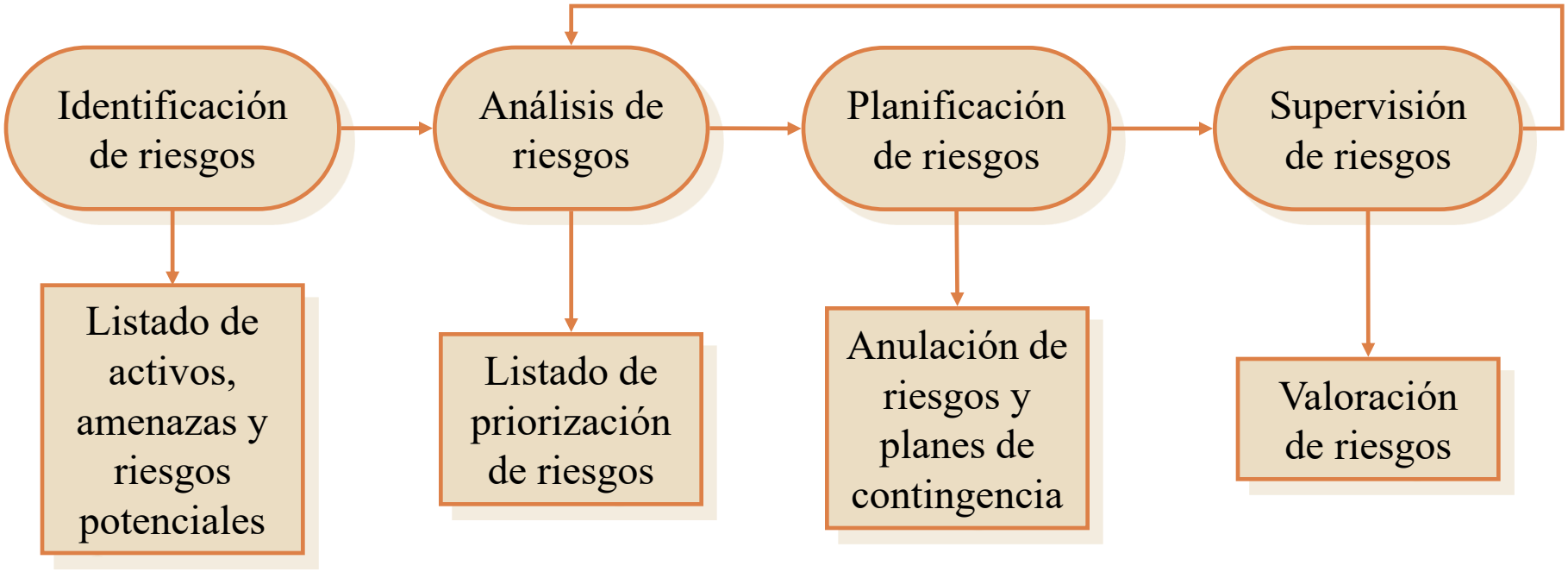
- Nuestros activos:
  - Alcance.
    - Cada entregable del proyecto, cada producto
  - Tiempo.
    - Cada Hito, cada fase
  - Coste. Todos los riesgos amenazan el coste
    - Pérdida de financiación
  - Calidad.
    - Estándares y criterios de calidad
  - Recursos.
    - Personal, materiales
  - Subcontrataciones

# Análisis de riesgos

- **Amenaza:** Causa potencial de un incidente que puede causar daños a un sistema de información o a una organización. [UNE 71504:2008]
- **Tipos:**
  - De origen natural
  - Del entorno (de origen industrial)
  - Defectos en las aplicaciones
  - Causadas por las personas de forma accidental
  - Causadas por las personas de forma deliberada

# Análisis de riesgos

- Administración del riesgo:



# Análisis de riesgos

- Administración del riesgo:
  - **Identificar los riesgos.**
    - Determinar los activos relevantes para el proyecto
    - Determinar a que amenazas están expuestos dichos activos
    - Definir el riesgo como la probabilidad de que una amenaza se materialice sobre un activo causando un impacto sobre él.
    - Clasificación en taxonomías.

      ***Sommerville***
      - Del producto: afectan a la calidad o desempeño del software
      - Del proyecto: afectan a la calendarización o recursos.
      - Del negocio: afectan a la organización responsable del desarrollo

      ***Technical Reports SEI***
      - Tecnológicos
      - Personales
      - De la organización
      - En las herramientas
      - En los requisitos
      - En la estimación
  - **Análisis de riesgos**
  - **Planificación del riesgo**
  - **Gestión de riesgos**

# Análisis de riesgos: Plan de riesgos

- Administración del riesgo:
  - **Identificar los riesgos.**
  - **Análisis de riesgos**
    - Para cada riesgo, evaluamos:
      - La probabilidad de que ocurra: Baja, Moderada, Alta y muy alta
      - Su Impacto, consecuencias, o degradación: Catastrófico, Grave, Tolerable, Insignificante.
  - **Planifica**
  - **Supervis**

***Matriz de riesgos***

| | TOLERABLE | SERIO | CRÍTICO |
|---|---|---|---|
| ALTA | | R4, R19, R26, R32, R33 | R5, R8, R13, R34 |
| MEDIA | R6. R15.R16, R18, R28, R29, R30 | R2, R3. R12, R17, R21, R23, R25, R27, R31 | R1, R7, R9, R11 |
| BAJA | R24. RIO | R14. R20, R22 | |

# Análisis de riesgos: Plan de riesgos

- Administración del riesgo:
  - **Identificar los riesgos.**
  - **Análisis de riesgos**
    - Para cada riesgo, evaluamos:
      - La probabilidad de que ocurra: Baja, Moderada, Alta y muy alta
      - Su Impacto, consecuencias, o degradación: Catastrófico, Grave, Tolerable, Insignificante.
  - **Planificc**
  - **Supervis**

***Matriz de riesgos***

| | TOLERABLE | SERIO | CRÍTICO |
|---|---|---|---|
| ALTA | | R4, R19, R26, R32, R33 | R5, R8, R13, R34 |
| MEDIA | R6. R15.R16, R18, R28, R29, R30 | R2, R3. R12, R17, R21, R23, R25, R27, R31 | R1, R7, R9, R11 |
| BAJA | R24. RIO | R14. R20, R22 | |

# Análisis de riesgos: Plan de riesgos

- Administración del riesgo:
  - **Identificar los riesgos.**
  - **Análisis de riesgos**
    - Riesgos Críticos
      1. Falta de experiencia técnica (programadores nóveles) - Puede generar errores frecuentes, bajo rendimiento o dificultades para estimar tareas.
      2. Mala planificación de sprints - Si no se definen bien los objetivos o se sobrecargan los sprints, se pierde ritmo y motivación.
      3. Falta de pruebas automatizadas o manuales - Puede llevar a errores en producción, especialmente si no hay cultura de testing.
      4. Falta de comunicación efectiva en el equipo - Puede causar malentendidos, trabajo duplicado o tareas mal ejecutadas.
    - Riesgos Moderados
      5. Desalineación con las necesidades del cliente (cafetería) - Si no se involucra al cliente en las revisiones, el producto puede no cumplir sus expectativas.
      6. Rotación o baja de algún miembro del equipo - En un equipo pequeño, la pérdida de una persona impacta mucho.
      7. Subestimación de la complejidad técnica del sistema - Por ejemplo, gestión de reservas en tiempo real, sincronización entre dispositivos, etc.
      8. Falta de documentación mínima - Puede dificultar el mantenimiento o la incorporación de nuevos miembros.
    - Riesgos Menores
      9. Problemas con herramientas ágiles (mal uso de Jira, Trello, etc.) - Puede generar desorganización o pérdida de seguimiento de tareas.
      10. Dependencia de tecnologías no dominadas - Elegir frameworks o lenguajes sin experiencia previa puede ralentizar el desarrollo.
      11. Falta de pruebas con usuarios reales - El sistema puede no ser intuitivo o no adaptarse al flujo real de trabajo de la cafetería.
      12. Problemas de integración con hardware (tablets, impresoras de tickets, etc.) - Puede requerir más tiempo del previsto para configurar e integrar.
  - **Planificación del riesgo**
  - **Supervisión de riesgos**

# Análisis de riesgos: Plan de riesgos

## Administración del riesgo:

- **Identificar los riesgos.**
- **Análisis de riesgos.**
- **Planificación del riesgo.**
  - ***Aceptar***: Gestionar el riesgo resulta más costoso que el impacto que pudiera tener, o la probabilidad es ínfima.
  - ***Evitar***: reducimos la probabilidad de aparición a 0.
  - ***Mitigar***: reducimos la probabilidad de aparición y/o el impacto sobre el proyecto.
  - ***Transferencia***: subcontratamos expertos que nos gestionen (y se responsabilicen) de la gestión de un riesgo (porque son expertos en ello). Contratamos un seguro.
    - Escalar: No es nuestra responsabilidad como equipo de desarrollo.
  - ***Planes de contingencia***: tratamos de llevar a nuestra organización al estado anterior a la ocurrencia del riesgo, siguiendo un plan preconcebido.
- **Supervisión de riesgos**

# Análisis de riesgos: Plan de riesgos

- Administración del riesgo:
  - **Identificar los riesgos.**
  - **Análisis de riesgos**
  - **Planificación del riesgo**
  - **Supervisión/Monitorización de riesgos**
    - Supervisión del proyecto en base a indicadores asociados a cada riesgo, para la detección temprana del mismo, minimizando sus daños, con tareas previstas en caso de aparición
    - Buscaremos indicadores que puedan ser monitorizados en el tiempo (de forma continua o discreta), estableciendo umbrales de alerta que podrían desencadenar acciones de reanálisis del riesgo

# Análisis de riesgos: Ejemplo de prácticas

| Identificador | Categoría | Amenaza | Activos |
|---|---|---|---|
| R3 | C1 | Inseguridad en el entorno labora | 1, 2, 4, 11, 12 |

| Descripción | Probabilidad de aparición | Impacto |
|---|---|---|
| Estado físico de las instalaciones en el cual el empleado puede sufrir algún tipo de accidente debido a las deficiencias en el entorno de trabajo. | Moderada | Serio |

| Riesgos |
|---|
| Para este riesgo se ha elegido la estrategia de transferencia. La acción a desarrollar será la contratación de una empresa especializada en la seguridad en el entorno de trabajo que acudirá y realizará periódicamente revisiones sobre el estado de las instalaciones. |

| Seguimiento de riesgos |
|---|
| Se tendrá en cuenta el número de accidentes laborales mensuales poniendo como límite 3 accidentes. |

# El desarrollo ágil

- Modelos pesados
  - Aproximación sistemática y disciplinada
  - Procesos fuertemente orientados a la documentación
    - La documentación no es un objetivo sino el medio para alcanzarlo garantizando su calidad.
    - El formalismo da calidad al producto y al proceso pero ralentiza el desarrollo.
      - Peso de la metodología.

# El desarrollo ágil

- En el manifiesto para el desarrollo ágil se señala la necesidad de valorar.
  - *A los **individuos y sus intenciones** sobre los procesos y sus herramientas*
  - *Al **software en funcionamiento** sobre la documentación extensa*
  - *A la **colaboración del cliente** sobre la negociación del contrato*
  - *A la **respuesta al cambio** sobre el seguimiento de un plan*

*Kent Beck +16 (2001)*

*http://agilemanifesto.org/iso/es/principles.html*

- Aunque se acepte la importancia de los términos finales en las sentencias se valora más el principio

# El desarrollo ágil

- Trata de aliviar el peso de las metodologías reduciendo la presión sobre la calidad de los modelos.
  - Los modelos deben ser SUFICIENTEMENTE buenos
- Trata de superar limitaciones de las metodologías pesadas
  - Incapacidad para adaptarse al cambio
  - Fragilidad asociada a las debilidades humanas

# El desarrollo ágil: El cambio

- Gestión del cambio.
  - Pesadas: Incapacidad de adaptarse al cambio.
    - Controlar el cambio.
      - Limitar los cambios permitidos.
      - Limitar el momento del cambio.
  - Ligeras: Concebidas para el cambio
    - Acelerar al máximo las entregas de software.
    - Hacer participe activo al cliente.
      - Evaluación del producto entregado.
      - Nuevas requisitos.
      - Nuevas prioridades.

# El desarrollo ágil: Fragilidad

- Fragilidad por las debilidades humanas
  - Pesadas:
    - Disciplina para la construcción de modelos y desarrollo sistemático
  - Ligeras:
    - Tolerancia en la aplicación de los modelos permitiendo que estos se adapten al equipo y no al revés.
    - Admitir la falta de eficiencia en el planteamiento.

# El desarrollo ágil

- Fuerte dependencia del Personal. Rasgos (I)
  - Competencia: sobre el proceso
  - Enfoque común: la entrega rápida
  - Colaboración: entre todos los stakeholders
  - Habilidades para la toma de decisiones (técnicas): necesaria para la entrega rápida
  - Capacidad de resolución de problemas confusos: ante la no concreción de los requisitos

# El desarrollo ágil

- Fuerte dependencia del Personal. Rasgos (II)
  - Confianza y respeto mutuo: equipo cuajado
  - Organización propia.
    1. Se organiza a si mismo. Roles
    2. Organiza el proceso. Método
    3. Organiza el plan de trabajo. Plan

# El desarrollo ágil: La alianza ágil

- Principios (12).
  1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software valioso.
  2. Bienvenidos los requisitos cambiantes, incluso en fases tardías del desarrollo. La estructura de los procesos ágiles cambia para la ventaja competitiva del cliente.
  3. Entregar con frecuencia software en funcionamiento, desde un par de semanas hasta un par de meses, con una preferencia por la escala de tiempo más corta.
  4. La gente de negocios y los desarrolladores deben trabajar juntos a diario a lo largo del proyecto.

# El desarrollo ágil: : La alianza ágil

- Principios.

5. Construir proyectos alrededor de individuos motivados. Darles el ambiente y el soporte que necesitan, y confiar en ellos para obtener el trabajo realizado.

6. El método más eficiente y efectivo de transmitir información hacia y dentro de un equipo de desarrollo es la conversación cara a cara.

7. El software en funcionamiento es la medida primaria de progreso.

8. Los procesos ágiles promueven el desarrollo sostenible. Los patrocinadores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de manera indefinida.

# El desarrollo ágil: : La alianza ágil

- Principios.

9. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.

10. La simplicidad -el arte de maximizar la cantidad de trabajo no realizado- es esencial.

11. Las mejores arquitecturas, los mejores requisitos y los mejores diseños emergen de equipos autoorganizados.

12. A intervalos regulares el equipo reflexiona sobre la forma en que se puede volver más efectivo; entonces su comportamiento se ajusta y adecua en concordancia.

# Desarrollo ágil

- Modelos de desarrollo ágil
  - Desarrollo Adaptativo de Software (DAS)
  - Método de desarrollo de sistemas dinámicos (MDSD)
  - Melé (Scrum)
  - Cristal
  - Desarrollo dirigido por Test (TDD)
  - Desarrollo dirigido por Comportamientos (BDD)
  - Modelado Ágil (MA)
  - Programación Extrema (PE)

# Modelado Ágil (MA)

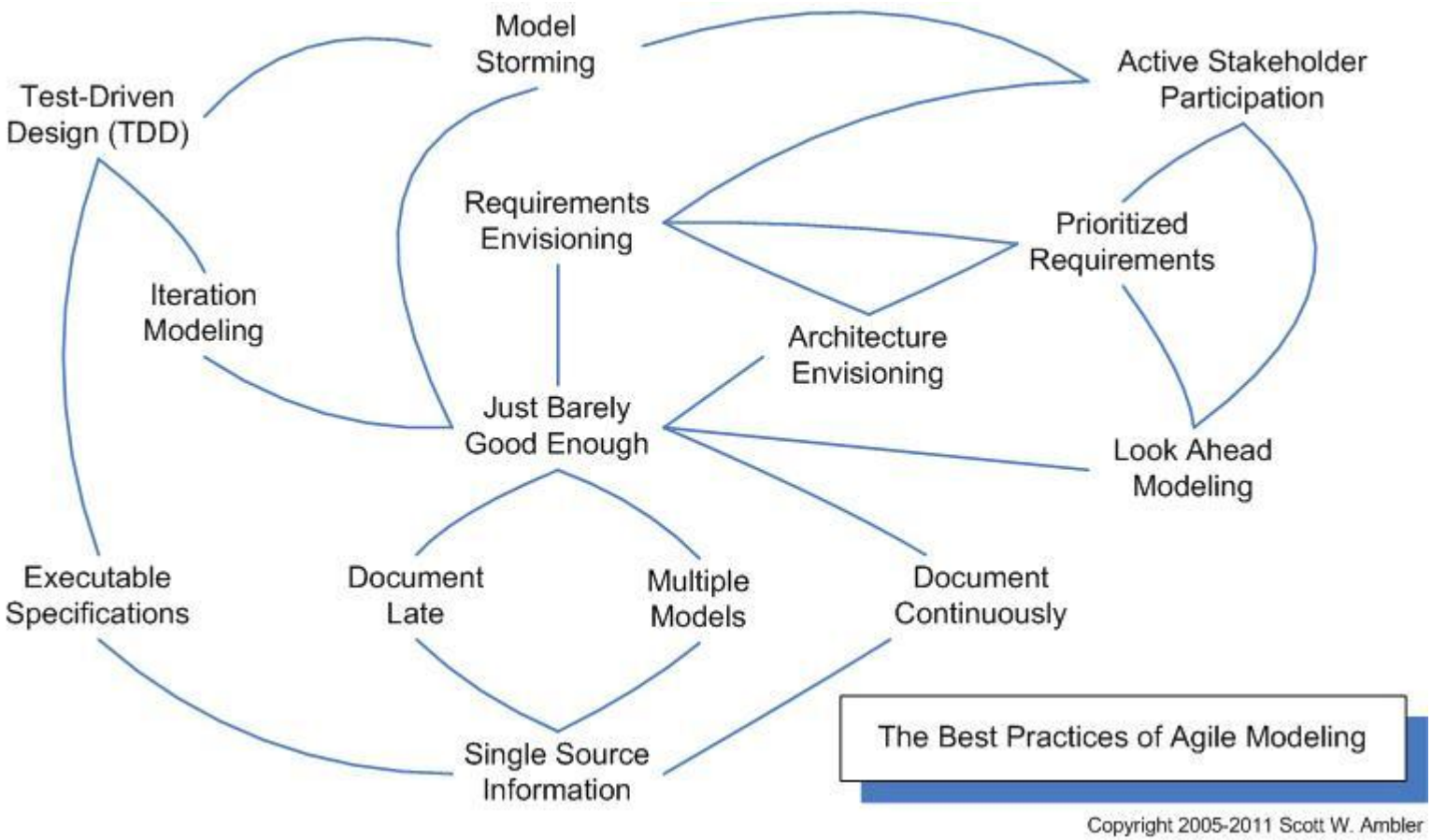
- Principios de Modelado Ágil
  - Modelar con un propósito (comunicación, desarrollo, …)
  - El contenido es más importante que la representación
  - Usar múltiples modelos (Procesos, Datos, Tiempo)
  - Conocer los modelos y las herramientas
  - Viajar ligero (documentación necesaria)
  - Adaptar al equipo ágil.

# Modelado Ágil (MA)

- El modelado ágil (MA) es una metodología basada en la práctica para el modelado y la documentación eficaces de programas de software.
- A alto nivel, el MA es un conjunto de buenas prácticas, representadas en el mapa de lenguaje de patrones de la siguiente diapositiva.
- A un nivel más detallado, el MA es un conjunto de valores, principios y prácticas para el modelado de software.

**http://www.agilemodeling.com/**

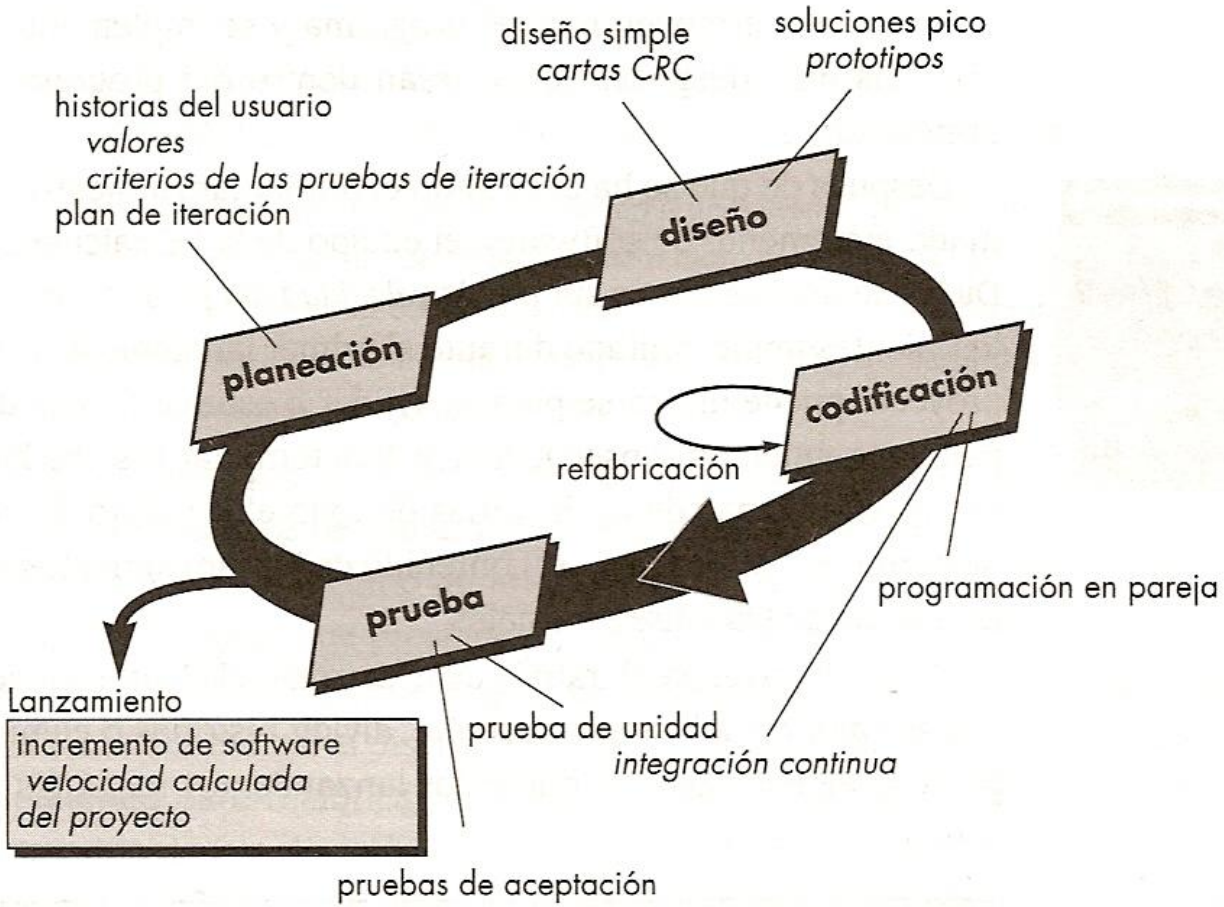
# Modelado Ágil (MA)



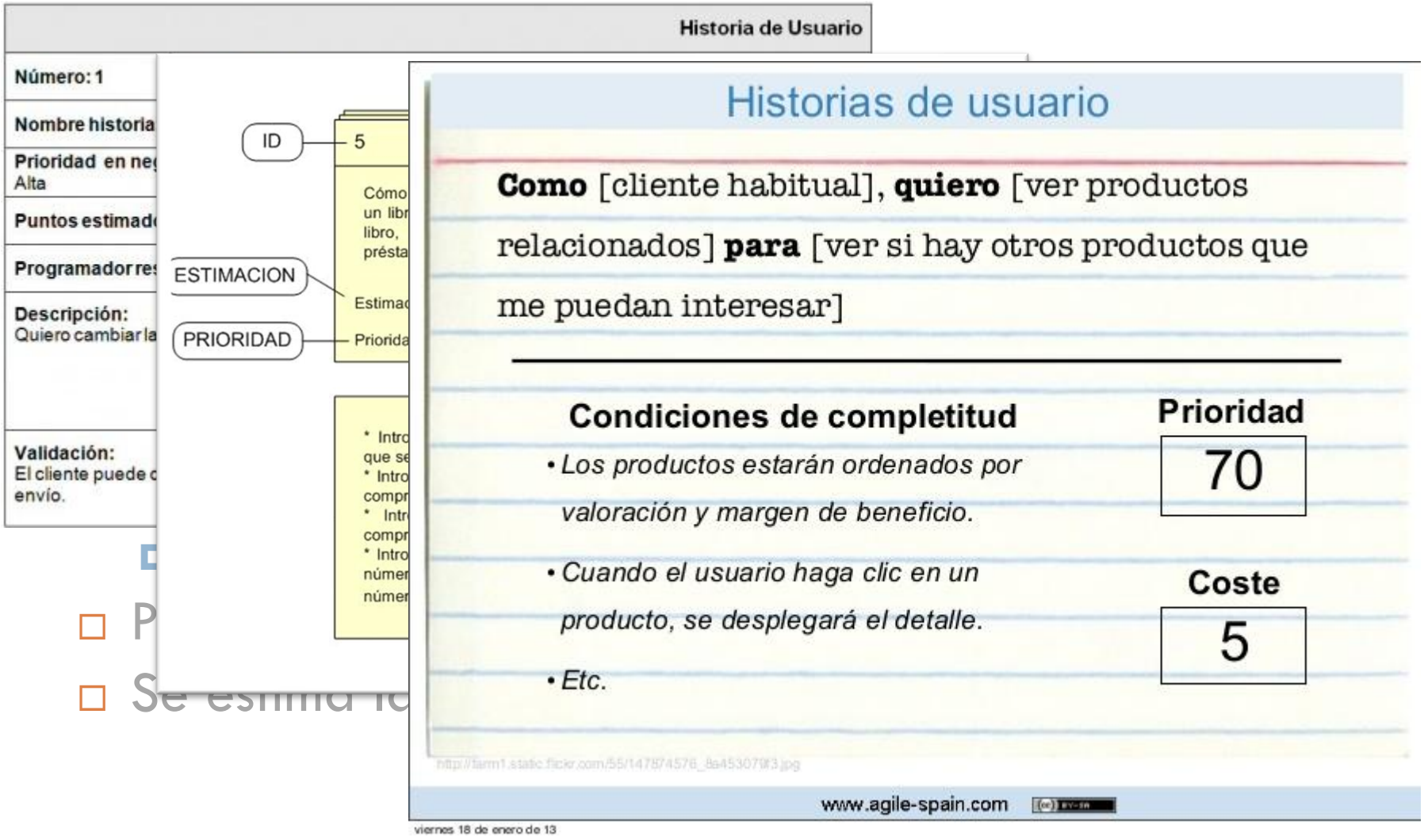



http://www.agilemodeling.com/

# Programación Extrema (PE,XP)



# Programación Extrema (PE): Planificación



# Programación Extrema (PE): Planificación

- Se crean Historias del usuario.
  - Se colocan en una carta índice
  - El usuario le asigna una prioridad
    - Dependiendo del valor de la característica o la función para su negocio o sus intereses.
    - Puede depender de la presencia de otras historias.
  - Se pueden crear, dividir, eliminar o cambiar de valor en cualquier momento
- Los miembros le asignan costo en semanas.
  - Las historias con coste > 3 sem. deben reescribirse
- Plan de construcción
- Se estima la velocidad del proyecto y se usa para:

# Programación Extrema (PE)

## Planificación

- Se crean Historias del usuario.
- Los miembros le asignan costo en semanas.
- Plan de construcción
  - Los clientes y el equipo agrupan y seleccionan juntos las historias del siguiente incremento (cuántas y cuáles) y acuerdan la fecha de entrega del incremento.
  - Para un lanzamiento se reordenan las historias según los criterios
    - Todas se implementan de modo inmediato
    - Las historias con valor más alto se implementarán al principio
    - Las historias de mayor riesgo se implementarán al principio
- En cada lanzamiento se estima la velocidad del proyecto y se usa para:
  - Estimar las fechas de entrega en los siguientes incrementos
  - Determinar si se ha hecho un compromiso excesivo
    - cambiar el contenido de los lanzamientos
    - las fechas de entrega final.

# Programación Extrema (PE)

## Diseño

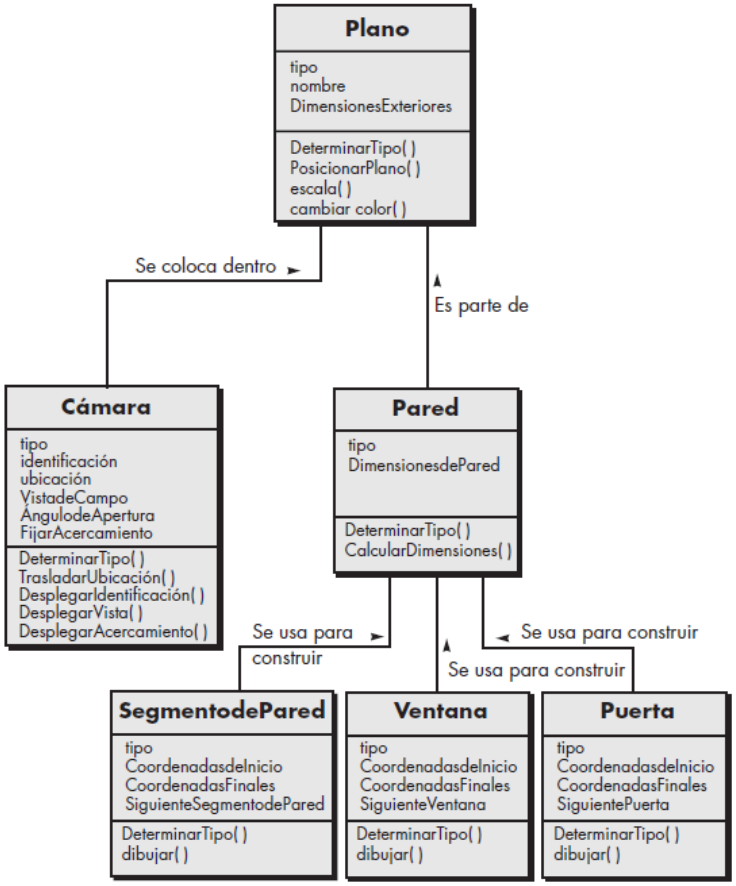
- Principios
  - 1er principio. MANTENERLO SIMPLE
  - Implementar sólo lo requerido
  - Utilizar Tarjetas CRC en el diseño.
  - Ante un diseño complejo solución de pico. Prototipo.
  - El diseño es un artefacto que puede y debe modificarse continuamente.
  - Refactorizacion del código.
    - Puede aumentar radicalmente con el tamaño de la aplicación.
- Productos
  - Tarjetas CRC (Clase-Responsabilidad- Colaborador)
  - Soluciones de Pico (prototipos para problemas de difícil diseño).

# Programación Extrema (PE)

## Diseño

**FIGURA 6.10**

**Diagrama de clase para Plano (véase el análisis en el recuadro)**

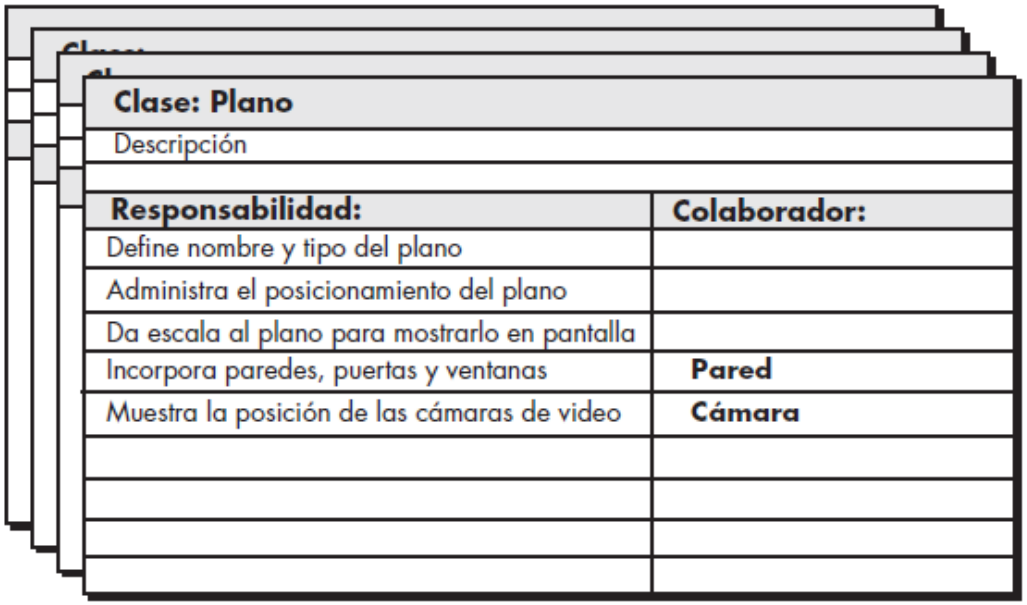


# Programación Extrema (PE)

## Diseño

**FIGURA 6.11**

Modelo de tarjeta CRC índice



**Clase: Plano**

Descripción

| Responsabilidad: | Colaborador: |
|---|---|
| Define nombre y tipo del plano | |
| Administra el posicionamiento del plano | |
| Da escala al plano para mostrarlo en pantalla | |
| Incorpora paredes, puertas y ventanas | **Pared** |
| Muestra la posición de las cámaras de video | **Cámara** |
| | |
| | |
| | |
| | |

# Programación Extrema (PE)

## Diseño

***Refactorización*** *es el proceso de cambiar un sistema de software de tal manera que no altere el comportamiento externo del código y que mejore la estructura interna. Es una manera disciplinada de limpiar el código [y modificar/simplificar el diseño interno], lo que minimiza las oportunidades de introducir errores. En esencia, al refactorizar(rediseñar), se mejora el diseño del código después de que se ha escrito*

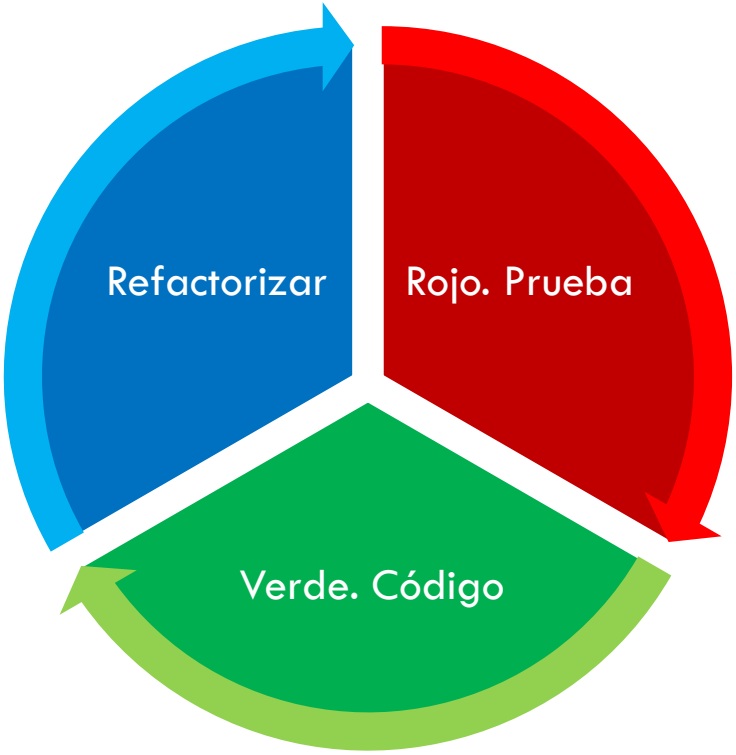
*Fowler, 2000*

# Programación Extrema (PE) Codificación

- Preparar las pruebas de unidad que debe superar el código
  - Programar para superar las pruebas
- Programación en parejas
  - Sobre la misma estación de trabajo
  - Papeles distintos
- Integración continua
  - Generado el código se integran todas las partes
    - Por un equipo de integración
    - Por el propio equipo de desarrollo
  - Prueba de Humo
    - Banco de pruebas que sin ser exhaustivo prueba todo el sistema.
    - Encontrar los errores importantes.

# Programación Extrema (PE)
# Codificación

TFD: Test First Development



# Programación Extrema (PE) Codificación

- Preparar las pruebas de unidad que debe superar el código
  - Programar para superar las pruebas
- Programación en parejas
  - Sobre la misma estación de trabajo
  - Papeles distintos
- Integración continua
  - Generado el código se integran todas las partes
    - Por un equipo de integración
    - Por el propio equipo de desarrollo
  - Prueba de Humo
    - Banco de pruebas que sin ser exhaustivo prueba todo el sistema.
    - Encontrar los errores importantes.

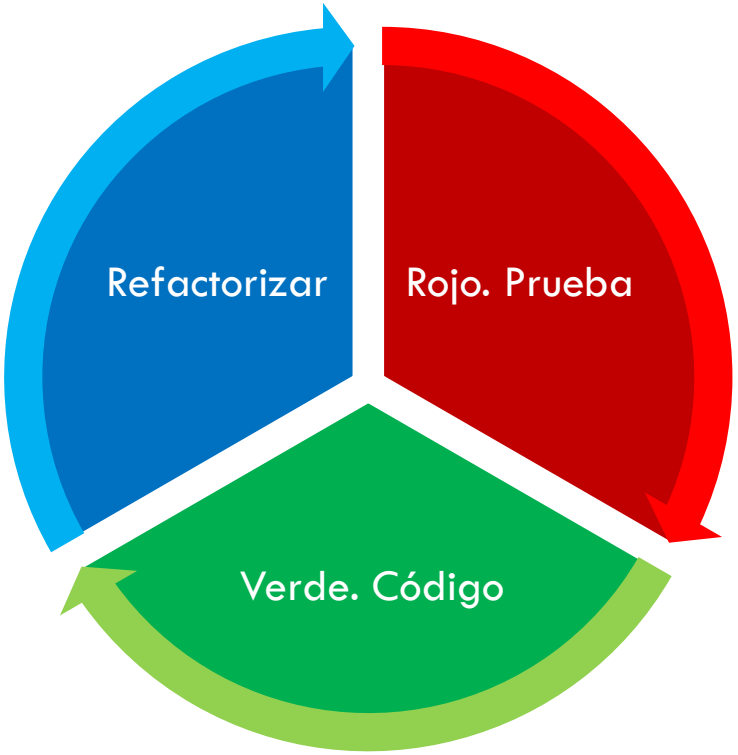
# Programación Extrema (PE)

## Prueba

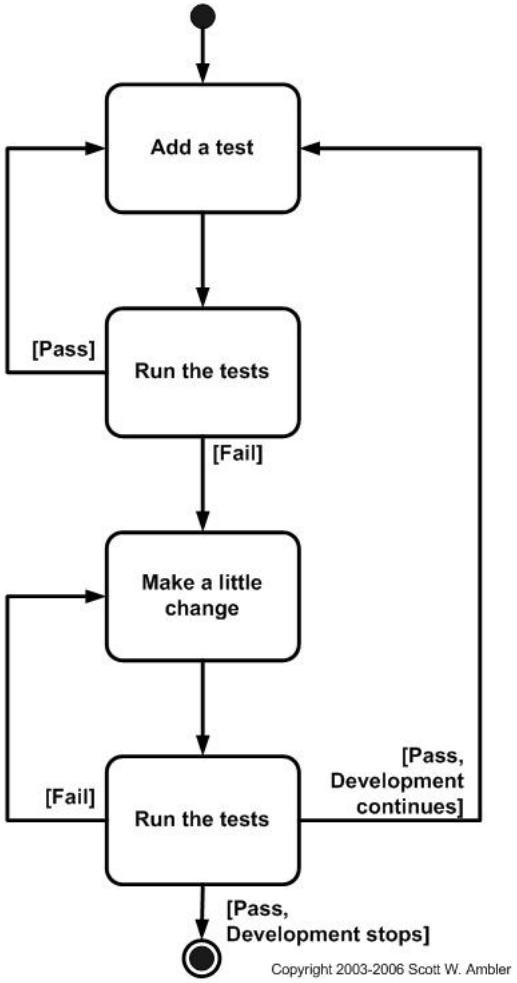
- Pruebas diarias. Refactorización.
  - ¿Cómo?
    - Organizamos las pruebas individuales en un conjunto universal de pruebas
    - Pruebas automáticas (xUnit)
  - ¿Por qué?
    - Indicación continua del progreso
    - Advierte cuando las cosas salen mal
    - Es más fácil arreglar problemas pequeños cada poco tiempo que uno grande antes de la fecha límite.
- Pruebas de aceptación
  - Se derivan de las historias
  - Las especifica el cliente
  - Orientadas
    - Características generales y la funcionalidad del sistema
    - Elementos visibles y revisables por el cliente.

# TDD: Test-Driven Development

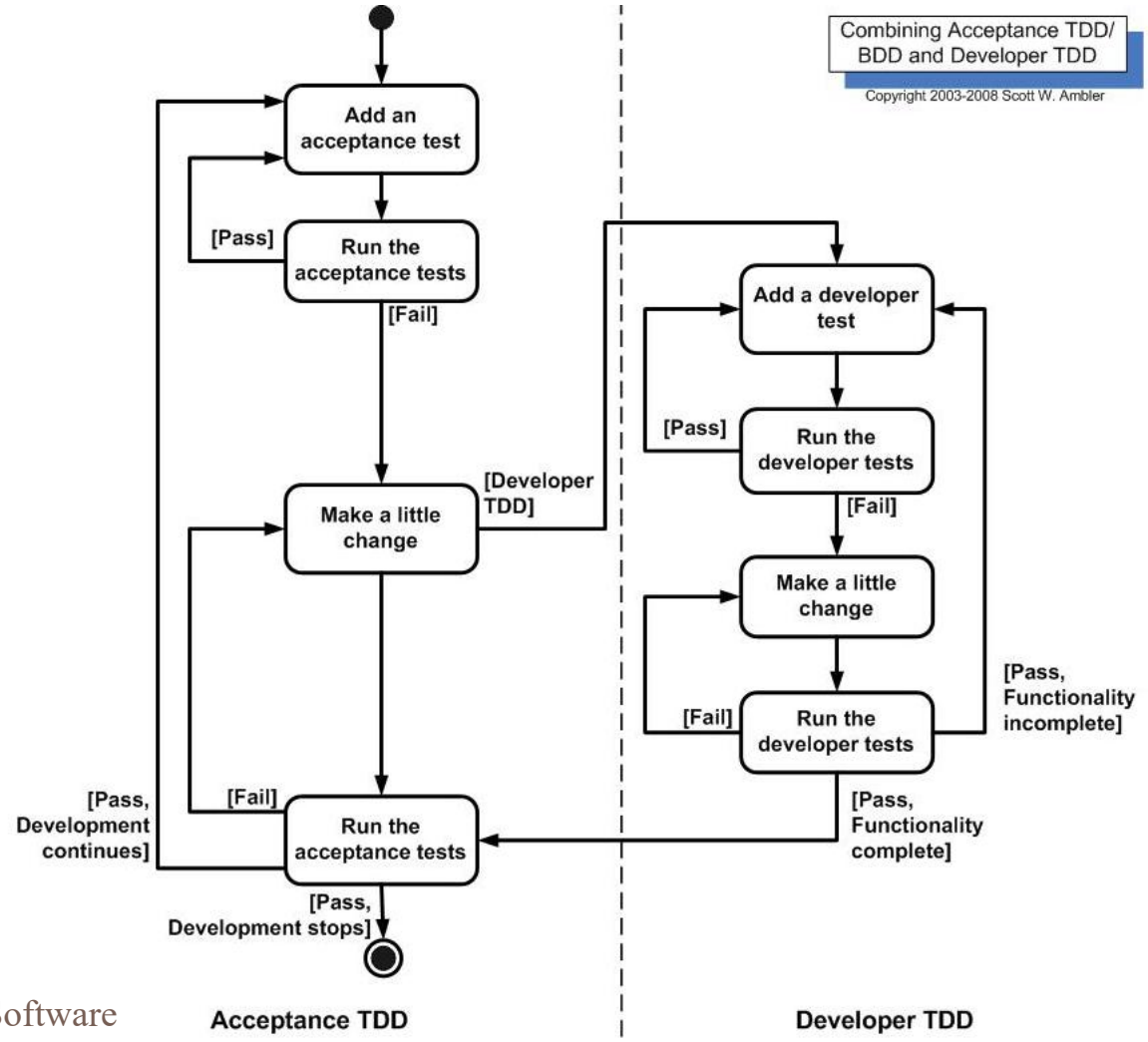
TDD: Refactorización + TFD




TFD: Test First Development



# TDD: Test-Driven Design



# Bibliografía


- Pressman, R.S.
  - Ingeniería del Software. Un enfoque práctico
    - 7ª Edición.
    - 6ª Edición. 2005
    - 5ª Edición. 2002
- Sommersville, I
  - Ingeniería de Software. 6ª Edición 2002
- Piattini, M.
  - Análisis y diseño detallado de Aplicaciones Informáticas de Gestión. 1996
  - Análisis y diseño de Aplicaciones Informáticas de Gestión. Una perspectiva de Ingeniería de Software. 2003