

¿Qué es el software?

□ Definición:

- instrucciones de ordenador que cuando se ejecutan proporcionan la función y el rendimiento deseado,
- estructuras de datos que facilitan a los programas manipular adecuadamente la información
- y los documentos que describen la operación y el uso de los programas.

Pressman

□ Definición:

- Programas de cómputo y documentación asociada.

Sommerville

¿Qué es la Ingeniería del Software?

- Una disciplina de la ingeniería que concierne a todos los aspectos de la PRODUCCIÓN DEL SOFTWARE desde su inicio hasta que se sustituye.
 - Ingeniería: Aplicación de Métodos, Herramientas, Técnicas, etc., de forma sistemática y organizada
 - Todos los aspectos de producción: aspectos técnicos, de gestión de RRHH, de gestión del tiempo, coste, calidad, etc.

Atributos del buen software

- Mantenibles

- Debe ser posible que el software evolucione y siga cumpliendo sus especificaciones

Sommerville.

- Confiables

- El software no debe causar daños físicos ni económicos en caso de fallo. Seguridad, fiabilidad y protección.

- Eficientes

- El software no debe desperdiciar los recursos del sistema

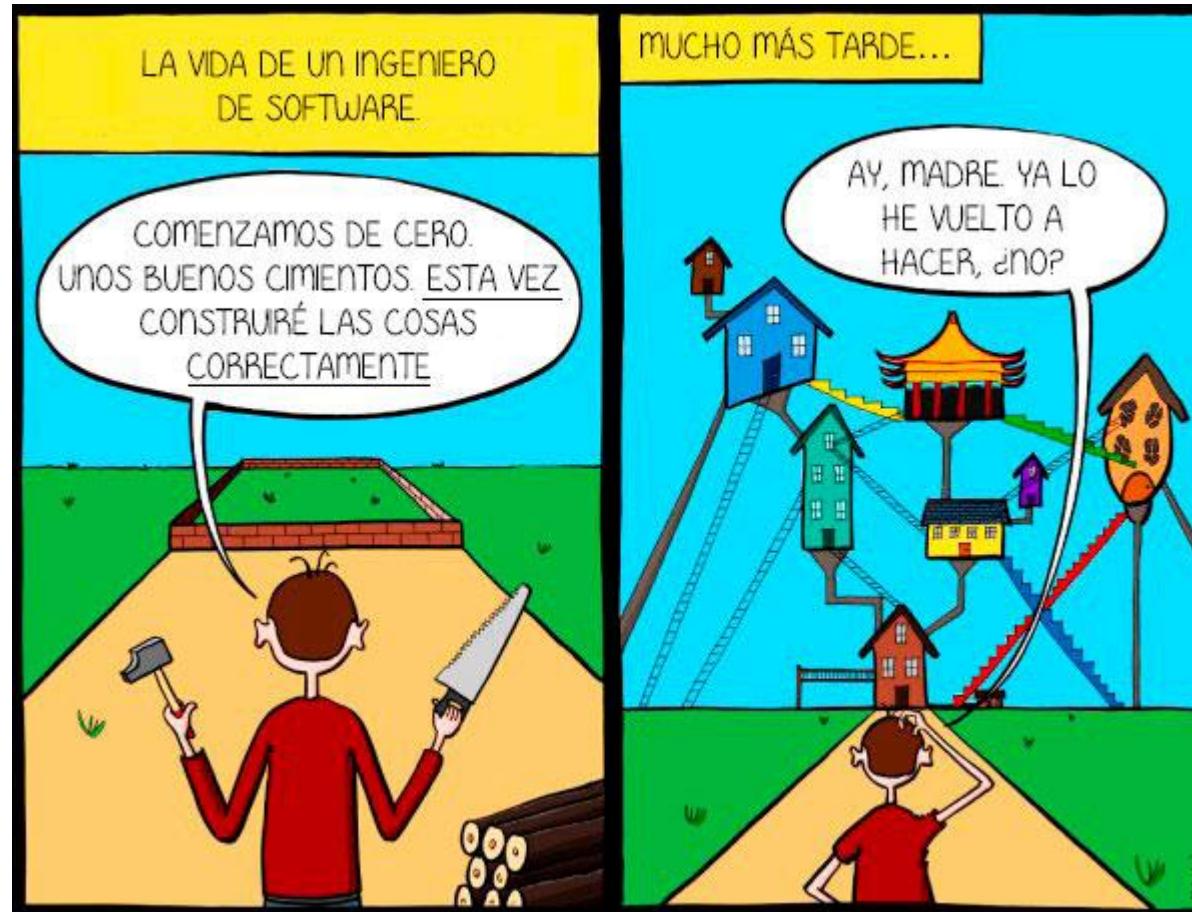
- Usables

- Debe contar con Interfaces y documentación adecuadas

Ingeniería vs Preguntas sin resolver

- ¿Por qué lleva tanto tiempo terminar los programas?
 - Planificación (EDT, Gantt, etc.)
- ¿Por qué es tan elevado el coste?
 - Estimaciones
 - Construcción a medida
 - Construcción desde cero.
- ¿Por qué no es posible encontrar todos los errores antes de entregar el software al cliente?
 - Planes de Pruebas
- ¿Por qué resulta tan difícil constatar el progreso conforme se desarrolla el software?
 - Elemento lógico
 - Procesos
 - Ciclos de vida

Ingeniería vs Preguntas sin resolver



Ingeniería del software.

¿Qué es el proceso de desarrollo del Software?

- La secuencia de actividades que conducen a la elaboración de un producto software.
- Fundamentalmente, pero no sólo:
 - Especificación
 - Desarrollo
 - Validación
 - Mantenimiento

Características del software

- El software se desarrolla, no se fabrica en un sentido clásico

Tabla 1.1. Influencia de los costes de ingeniería en el coste total del producto

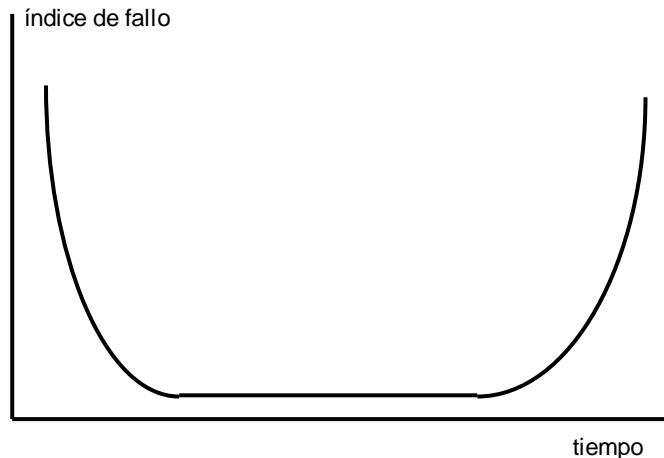
	Ingeniería	Producción o Desarrollo	Coste unitario / 100 unidades	Coste unitario / 100.000 unidades
Hardware	1000	50 c.u.	60	50.01
Software	1000	2000	30	0.03

Ingeniería del software.

Características del software

- El software no se fabrica, se desarrolla
- El software no se estropea

Curva de fallos del hardware



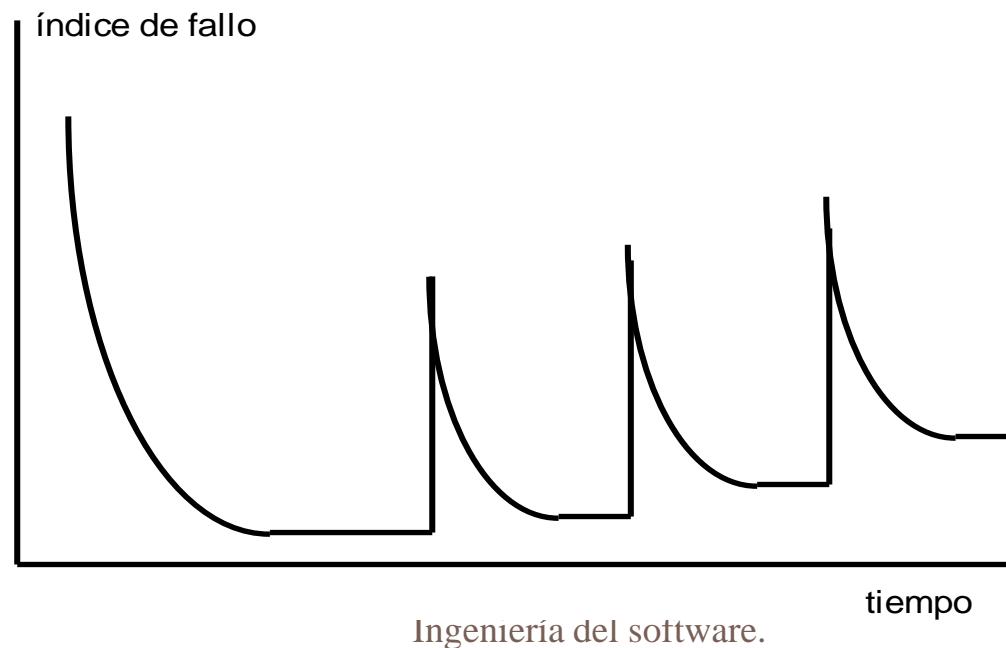
Curva ideal de fallos del software



Características del software

- El software no se fabrica, se desarrolla
- El software no se estropea, se degrada

Curva real de fallos del Software



Características del software

- El software no se fabrica, se desarrolla
 - El software no se estropea, se degrada
 - Software a medida, o personalizado.
 - Productos Genéricos: Producidos por una organización para introducir en el mercado
 - Productos a Medida: Desarrollados bajo pedido
 - La mayor parte del gasto es en productos genéricos pero hay más esfuerzo en el desarrollo de los productos a medida
 - Tuning de soluciones genéricas a problemas particulares (ERPs)
- Ingeniería del software.

Características del software

- Reutilización.
 - Posibilidades: Activos o artefactos
 - Bibliotecas de reutilización
 - Base de datos de componentes
 - Gestión de la biblioteca para acceso a datos
 - Sistema de recuperación de componentes
 - Módulos o Componentes.
 - Integrar, adaptar, envolver
 - Configurar, extender frameworks
 - Patrones de diseño (FAÇADE, MVC, etc.)
 - Requisitos???

Características del software

□ Reutilización.

□ Ventajas

- Reduce los costes
- Aumenta productividad
- Aumenta la calidad
- Mejora mantenimiento
y soporte
- Mejora control y
planificación

□ Limitaciones

Características del software

□ Reutilización.

□ Ventajas

- Reduce los costes
- Aumenta productividad
- Aumenta la calidad
- Mejora mantenimiento y soporte
- Mejora control y planificación

□ Limitaciones

- Complejos vs óptimos
- Inversión inicial
- Reestructuración
- Organización y proceso
- Recuperación de activos

Características del software

□ Reutilización.

□ Estrategia

- Ver qué es reutilizable
- Difundir la información
- Obligar al reuso
enseñando a operar los
activos

□ Futuro

- Búsqueda de activos de
cualquier tipo en un
repositorio (en la
nube?)

El software heredado

- **Características**
 - Es crítico para las empresas que lo usan
 - Es longevo.
 - Ha sido modificado de forma continuada
 - Debe adaptarse a nuevos entornos.
 - Nuevo hardware
 - La red
 - Nuevos sistemas (BDs)
 - Debe implementar nuevos requerimientos
- **Problemas: poca calidad**
 - Diseños complejos
 - Código oscuro
 - Historia de cambios manejada con pobreza
 - No está documentado
 - Pruebas sin documentar
- **Consecuencias**
 - Costoso en mantenimiento
 - Su evolución tiene un altísimo riesgo para la empresa.
- **Evolución del software.**

Evolución del software

□ Evolución de grandes sistemas soft. Leyes de Lehman *(Estudio referido a finales del siglo XX)*

- El cambio es continuo
- Crecimiento continuo
 - nuevas funcionalidades continuamente
- La complejidad es creciente
 - Costes extra para simplificación
- Estabilidad organizacional
 - Velocidad de trabajo invariante
- Conservación de la familiaridad:
 - Cambios incrementales constantes, independientes de los recursos
- Decremento de la calidad

Aplicaciones del software

- **Tipos de problema**
 - Problemas con solución por pasos específicos
 - Algoritmo, con lenguajes de prog. Procedimentales
 - Problemas que pueden describirse formalmente
 - Lenguajes declarativos. SQL
 - Problemas que no sabemos como se resuelven, pero conocemos algunas soluciones concretas
 - Utilizaremos redes neuronales
 - Problemas basados en conocimiento heurístico
 - Sistemas expertos, basados en la ejecución de reglas

Aplicaciones del Software

- Clasificación por Categoría (Pressman).
 - S. empotrado: lavadoras, hornos, coches, etc.
 - S. de sistemas: Sistemas operativos, compiladores,...
 - S. de aplicación: Software a medida.
 - S. científico y de ingeniería: Cálculo numérico
 - S. de línea de productos: Office, juegos, etc.
 - Aplicaciones basadas en Web, o en la nube.
 - S. de Inteligencia Artificial

Otras Clasificaciones del software (Sommerville)

- Por su estructura:
 - Orientados a función.
 - Orientados a componentes.
 - Orientados a listas.
 - Orientados a objetos.
- Por su función:
 - Programas o Sistemas de Usuario
 - Interfaces Hombre-Maquina.
 - Herramientas de Software.
 - Librerías.
 - Sistemas de uso genérico:
Compiladores, S.O's,
Procesadores de Texto, etc.
 - Bases de Datos.
 - Sistemas basados en Web.
- Por su plataforma de computo:
 - Sistemas embebidos.
 - Sistemas de computo distribuido.
 - Sistemas de computo paralelo.
 - Sistemas de tiempo real.
 - Sistemas basados en Chips.
 - Wearable computing systems.
 - Sistemas de computo ubicuos.

Problemas del desarrollo de software

- Planificación y estimación de costes, imprecisos
 - No hay estudios de realizaciones previas
 - CMMI: 1^a Norma: Medir.
 - Los responsables de proyecto pueden no ser expertos informáticos
 - Los expertos en informática dirigen proyectos sin conocimientos de gestión.
- Productividad baja: duración mayor que la esperada
- Mala calidad a la entrega del producto
- Cliente insatisfecho ⇒ Rediseño del producto

Problemas del desarrollo de software

- Planificación y estimación de costes, imprecisos
- Productividad baja: duración mayor que la esperada
 - Especificaciones ambiguas o incorrectas
 - Muchas modificaciones sobre la marcha
 - Falta de documentación
 - Soluciones:
 - Administración de requisitos
 - Metodologías ágiles
 - Gestión de la configuración ⇒ G. del cambio.

Problemas del desarrollo de software

- Planificación y estimación de costes, imprecisos
- Productividad baja: duración mayor que la esperada
- Mala calidad a la entrega del producto
 - Aseguramiento de la calidad
 - Procesos de Verificación y Validación
- Cliente insatisfecho → Rediseño del producto

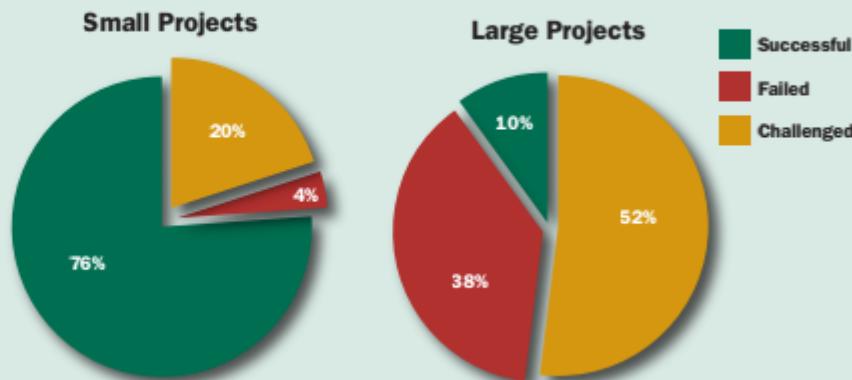
Problemas del desarrollo de software

□ CHAOS Report 2013 (Standish Group)

	1994	1996	1998	2000	2002	2004	2011	2012	2013	2014	2015
Éxito	16%	27%	26%	28%	34%	29%	29%	27%	31%	28%	29%
Comprometidos	53%	33%	46%	49%	51%	53%	49%	56%	50%	55%	52%
Cancelados	31%	40%	28%	23%	15%	18%	22%	17%	19%	17%	19%

CHAOS RESOLUTION BY LARGE AND SMALL PROJECTS

Project resolution for the calendar year 2012 in the new CHAOS database. Small projects are defined as projects with less than \$1 million in labor content and large projects are considered projects with more than \$10 million in labor content.



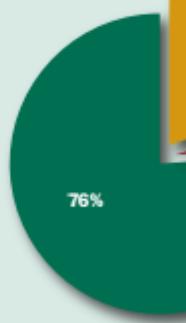
Problemas del desarrollo de software

□ CHAOS Report 2013 (Standish Group)

	1994	1996	1998	2000	2002	2004	2011	2012	2013	2014	2015
Éxito	16%	27%	26%	28%	34%	29%	29%	27%	31%	28%	29%
Comprometidos	53%	33%	46%	49%	51%	53%	49%	56%	50%	55%	52%
Cancelados	31%	40%	28%	23%	15%	18%	22%	17%	19%	17%	19%

CHAOS RESOLUTION BY LARGE AND SMALL PROJECTS

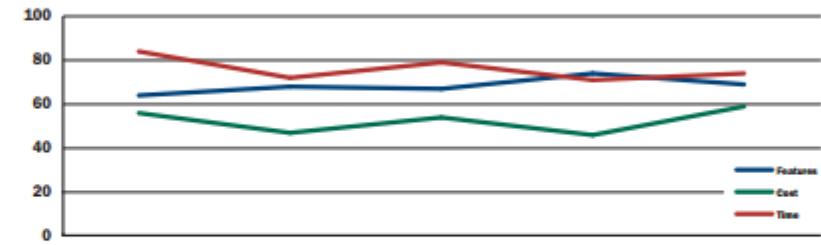
Project resolution for the calendar year 2012 in the new CHAOS database. Small projects are defined as projects with less than \$1 million in labor content and large projects are considered projects with more than \$10 million in labor content.



Small Pro

OVERRUNS AND FEATURES

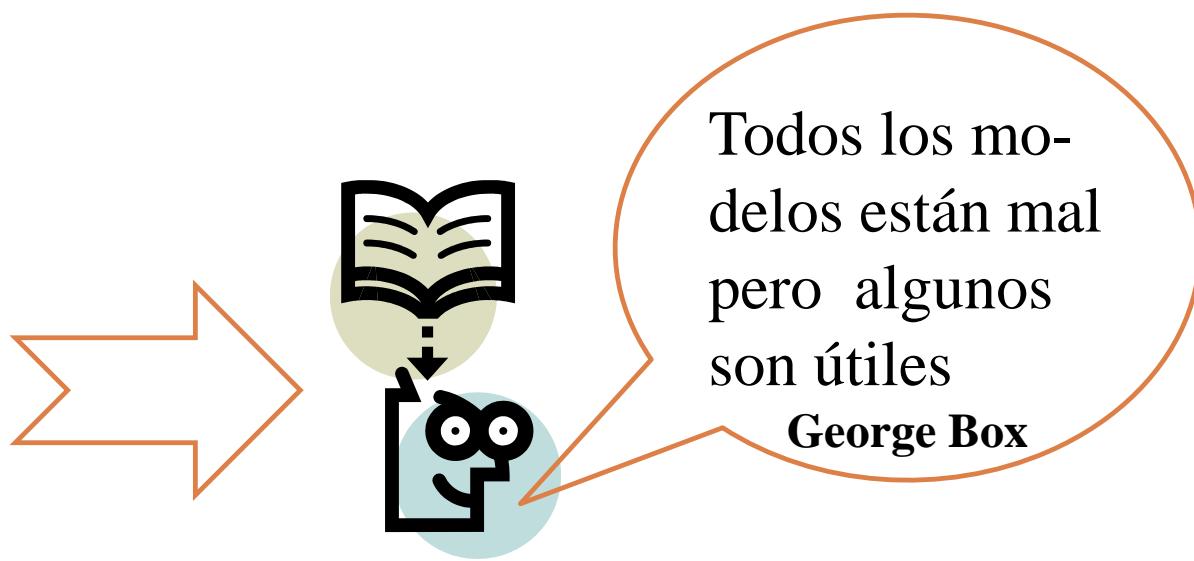
Time and cost overruns, plus percentage of features delivered from CHAOS research for the years 2004 to 2012.



	2004	2006	2008	2010	2012
TIME	84%	72%	79%	71%	74%
COST	56%	47%	54%	46%	59%
FEATURES	64%	68%	67%	74%	69%

Mitos del software

- Mitos de la administración
 - Ya se tiene un libro lleno de estándares y procedimientos para la construcción del software.



Ingeniería del software.

Mitos del software

- Mitos de la administración
 - Ya se tiene un libro lleno de estándares y procedimientos para la construcción del software.
 - Si un proyecto se atrasa sólo tengo que añadir personal
 - Si subcontrato un proyecto de software puedo despreocuparme y esperar a que se construya

Mitos del software

- Mitos del Cliente
 - Un enunciado general de los objetivos es suficiente para comenzar a escribir programas
 - Podemos cambiar continuamente los requerimientos del programa porque el software es flexible y se adapta

Mitos del software

□ Mitos del desarrollador

- Una vez el programa ha sido escrito y puesto a funcionar el trabajo está terminado
- Mientras el programa no se esté ejecutando, no existe forma de evaluar su calidad
- El único producto del trabajo que puede entregarse para que un proyecto se considere un éxito es el programa en funcionamiento
- La I.S. obliga a emprender la elaboración de documentación que hará el proceso más lento.

Bibliografía

- Pressman, R.S.
 - Ingeniería del Software. Un enfoque práctico
 - 6^a Edición. 2005
- Sommersville, I.
 - Ingeniería de Software. 6^a Edición 2002
- Chaos Report 2013
 - <https://larlet.fr/static/david/stream/ChaosManifesto2013.pdf>
- Chaos Report 1995
 - <https://www.projectsmart.co.uk/white-papers/chaos-report.pdf>

Glosario

- Definición y conceptos de Ingeniería del software.
- Modelos de procesos para la construcción del software
 - IEEE Std. 1074-2006
 - ISO/IEC 12207-1
 - ISO/IEC 15504-2.
- Evaluación del Proceso del Software.
 - Introducción.
 - Estándares.
 - Capability Maturity Model Integration CMMI.
 - Modelo “IDEAL”

Ingeniería del Software

□ Definiciones:

- La Ingeniería del software es una disciplina de la ingeniería que concierne a todos los aspectos de la producción del software. **Sommerville**
- Ingeniería del software es el establecimiento y uso de principios de ingeniería robustos, orientados a obtener software económico, que sea fiable y funcione de manera eficiente sobre máquinas reales. **Fritz Bauer. [Nau69]**
- La aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software; es decir, la aplicación de la ingeniería al software. **IEEE**

El proceso del software ≡ Ciclo de vida

- 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.

I.S. El Proceso del Software

□ Ingeniería del Software:

- **Modelos de los procesos:** Descripción de los procesos involucrados en el desarrollo de software sin precisar como se desarrollan.
 - IEEE 1074 - 2006
 - ISO 12207-1
 - ISO/IEC TR 15504-2
- **Procesos:** Conjunto de Actividades y tareas
- **Actividad:** Conjunto de Tareas.
- **Tareas:** Cualquier acción que transforma una entrada en salidas.
- **Métodos y/o procedimientos:** Definen la forma de ejecutar las tareas. Determinan el modo en el que se utilizan las técnicas en cada fase del desarrollo.
- **Técnicas:** Cualquier recurso utilizado para llevar a cabo una tarea. Normalmente gráficos con apoyos textuales.
- **Herramientas:** Cualquier software que nos ayude en cualquier etapa del proceso de desarrollo del software. CASE.

I.S. El Proceso del Software

□ Ingeniería del Software: Ejemplo

- Modelos de los procesos: ISO 15504-2
- Procesos: Verificación ó Validación
- Actividad: Desarrollo de un Plan de Pruebas
- Tareas: Ejecución de Casos de prueba
- Métodos y/o procedimientos: Norma IEEE 827 para la ejecución de las pruebas
- Técnicas: AVL, Cobertura de caminos
- Herramientas: JUnit.

MODELOS DE PROCESOS DE DESARROLLO DEL SOFTWARE

- **IEEE 1074:** Estándar para la creación de un proceso que dirija el desarrollo y mantenimiento de un proyecto software, previa selección de un ciclo de vida. (2006)
- **ISO 12207-1:** Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso. (1994)
- **ISO/IEC TR 15504-2:** Un modelo de referencia para los procesos y la capacidad de proceso (1998)

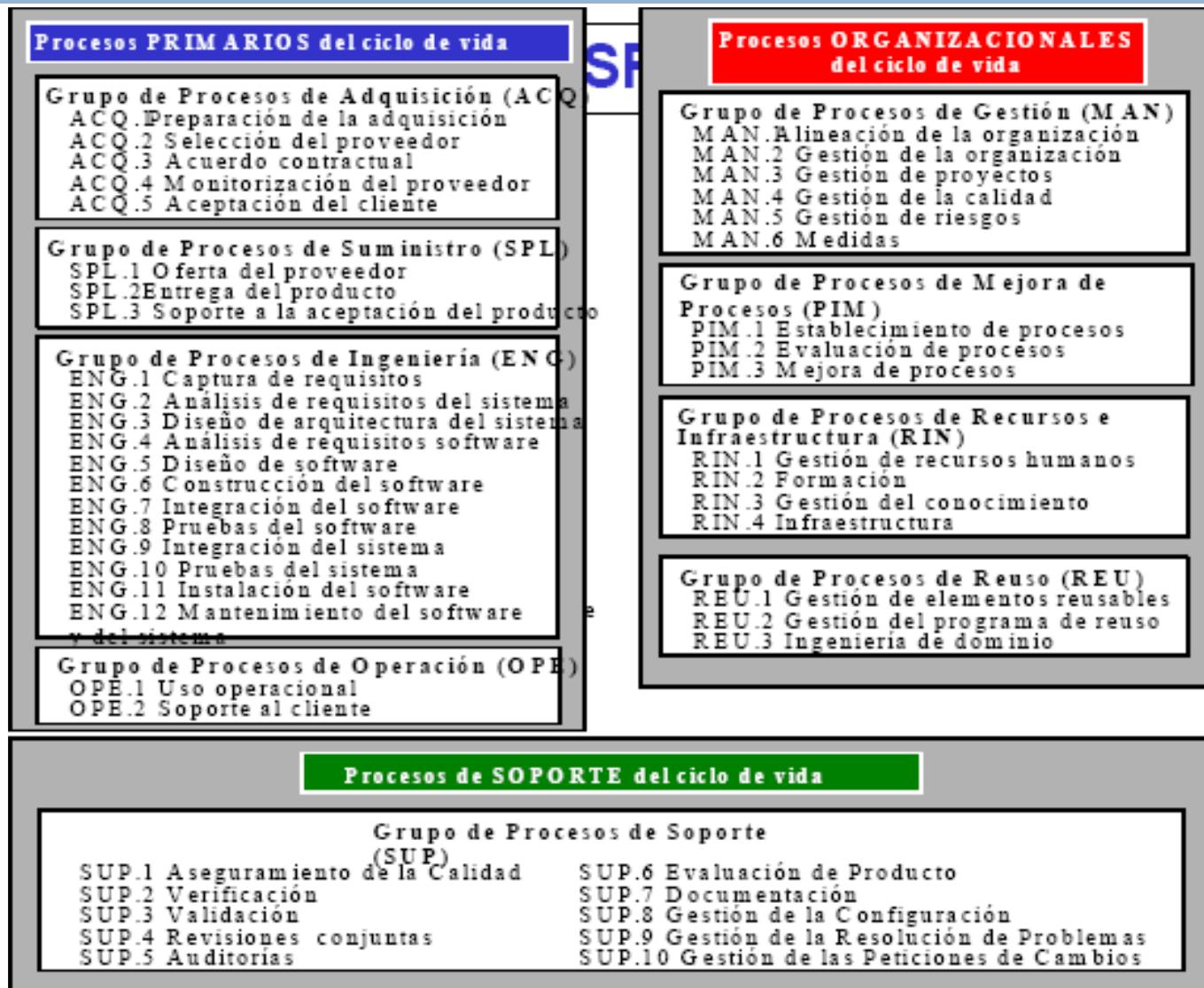
Proceso del Software

(Pressman)

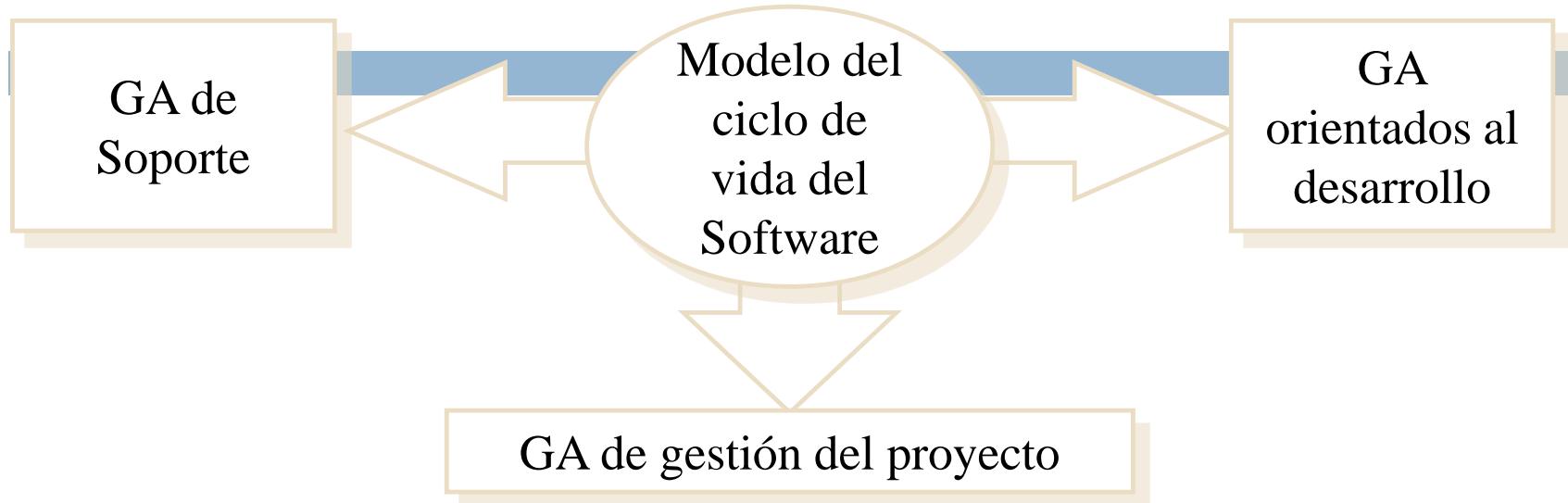
- Definición: ¿Qué?
 - Análisis: Información, Funcionalidad, Comportamiento, Interfaces, Rendimiento y Restricciones
- Desarrollo: ¿Cómo?
 - Diseño: Estructura de datos, Función, Arquitectura, Detalles procedimientos, Interfaces
 - Codificación: Traducción.
 - Pruebas (Sommerville lo plantea como otra fase)
- Mantenimiento: Gestión del cambio
 - Corrección (Errores), Adaptación (Cambio del entorno), Mejora (Cambios en los requisitos), Prevención (reingeniería).
- Otras Actividades
 - Planificación: Riesgos, Recursos, Tareas, Costes, Tiempo
 - Gestión de la configuración, Verificación , Validación
 - Seguimiento, control, aseguramiento de calidad, mediciones y gestión de riesgo.

Tema 2.- El proceso del software

ISO/IEC TR 15504-2 (2003)



IEEE 1074 - 2006



□ 4 Secciones Lógicas

- **Modelos del ciclo de vida:** Procesos orientados a su selección.
- **Sección de grupos de actividades de gestión:** Actividades que inician, supervisan y controla los procesos a lo largo del ciclo de vida
- **Sección de grupos de actividades orientados al desarrollo:** Comprenden los realizados antes, durante y después del desarrollo.
- **Sección de grupos de actividades de soporte:** Son aquellas necesarias para asegurar la terminación y la calidad de las actividades del proyecto (evaluación, gestión de configuración, documentación y formación)

Modelo del ciclo de vida del Software

- El estándar no establece ni define un ciclo de vida específico pero si requiere que se seleccione y utilice uno.
- La selección del modelo de ciclo de vida software concreto se hará en función de las muchas variables que pueden afectar a esta decisión.
- Aunque es posible utilizar el mismo ciclo de vida en distintos proyectos éste debe adaptarse a cada caso particular.
- Introduce la necesidad de evaluar el riesgo

IEEE 1074 - 2006

Sección de grupos de actividades de gestión

- **Iniciación del proyecto:** Se crea el ciclo de vida y se establecen las estimaciones, recursos, métricas y objetivos de seguridad para gestionar el proyecto
- **Planificación y control del proyecto:** Se desarrollan planes para la evaluación, gestión de la configuración, transición, instalación, gestión de proyecto, integración, etc.
- **Monitorización y control:** Actividades de gestión de riesgos, gestión de proyecto, mejoras del ciclo de vida, registros, recolección y análisis de métricas y cierre del proyecto.

Sección de grupos de actividades orientadas al desarrollo (en la norma desacoplados)

- Procesos realizados antes, durante y después del desarrollo
- Antes (Pre-):
 - Exploración de Concepto
 - Asignación del Sistema:
(Funciones,
Arquitectura,
requisitos)
 - Importación de Software
- Durante:
 - Análisis de requisitos.
 - Diseño (arquitectura, BD,
interfaces, diseño detallado.)
 - Implementación: Código,
documentación, integración,
gestión de versiones
- Despues (Post-):
 - Instalación.
 - Operación y Soporte
 - Mantenimiento
 - Retirada.

IEEE 1074 -2006

Sección de grupos de actividades de soporte

- Actividades de evaluación (Actividades para asegurar la calidad)
- Gestión de la configuración del software
- Desarrollo de documentación
- Formación del cliente.

IEEE 1074 -2006

- Ver pag. 102/116 del pdf de la norma.

ISO 12207-1

PROCESOS PRINCIPALES

ADQUISICIÓN

SUMINISTRO

DESARROLLO

EXPLORACIÓN

MANTENIMIENTO

PROCESOS DE SOPORTE

DOCUMENTACIÓN

GESTIÓN DE CONFIGURACIÓN

ASEGURAMIENTO DE CALIDAD

VERIFICACIÓN

VALIDACIÓN

REVISIÓN CONJUNTA

AUDITORÍA

RESOLUCIÓN DE PROBLEMAS

PROCESOS DE LA ORGANIZACIÓN

GESTIÓN

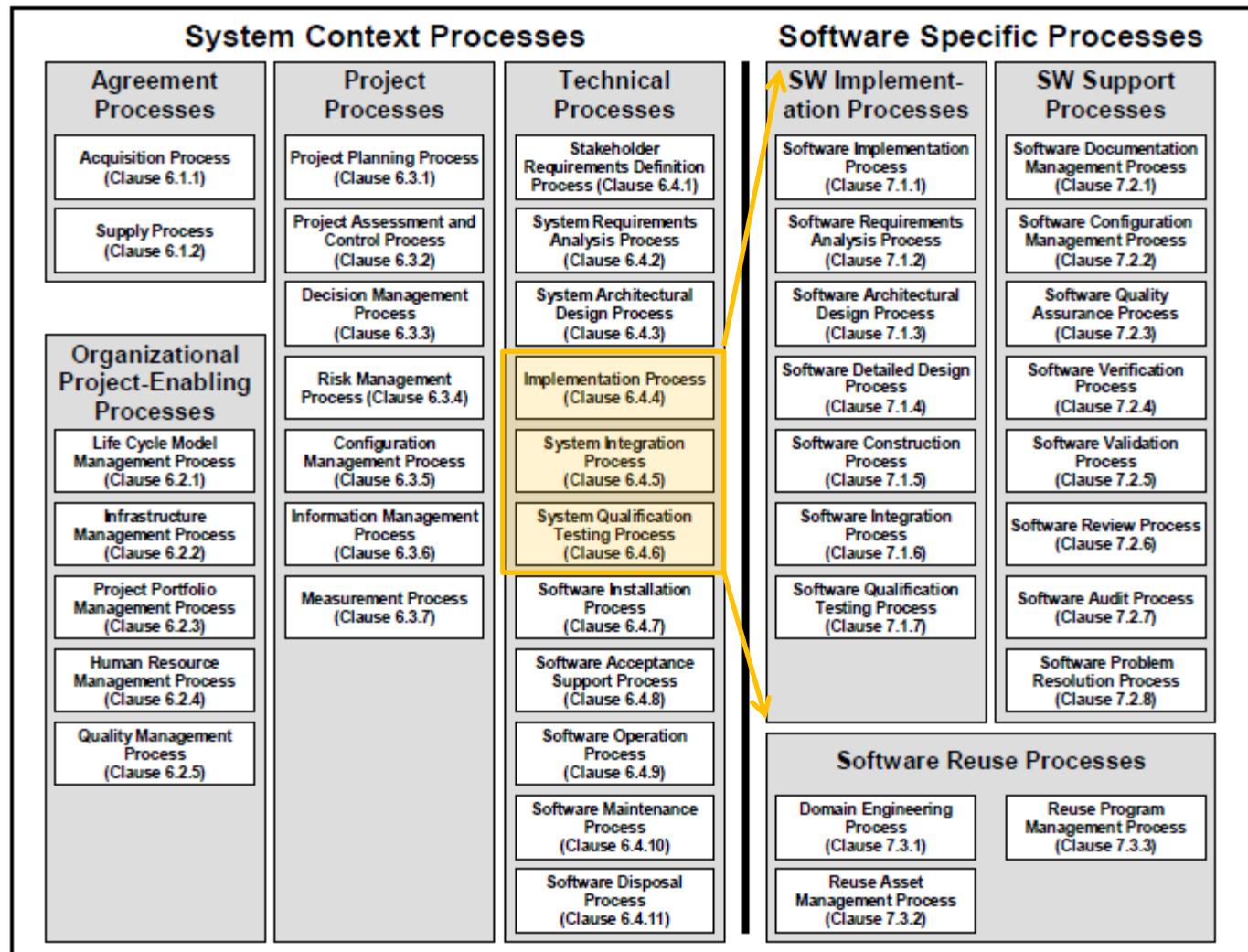
INFRAESTRUCTURA

MEJORA

FORMACIÓN

Tema 2.- El proceso del software

ISO 12207-2008



ISO 12207-1

PROCESOS PRINCIPALES

- Los procesos principales son aquellos que resultan útiles a las personas que inician o realizan el desarrollo, la explotación o el mantenimiento del software durante su ciclo de vida.
- Estas personas son:
 - Los compradores
 - Los suministradores
 - El personal de desarrollo
 - Los usuarios
 - El personal de mantenimiento de software

ISO 12207-1

PROCESOS PRINCIPALES

ADQUISICIÓN

- Actividades y tareas del comprador para
 - Preparación y publicación de solicitud de ofertas
 - Selección del suministrador
 - Gestión desde la adquisición a la recepción del producto

SUMINISTRO

- Actividades del suministrador para
 - Preparar una propuesta para responder a una solicitud
 - Identificar los Procedimientos y recursos para garantizar el éxito del proyecto.

ISO 12207-1

PROCESOS PRINCIPALES

ADQUISICIÓN

SUMINISTRO

DESARROLLO

- Análisis de requisitos del sistema.
- Diseño de arquitectura.
- Análisis de requisitos del software. Prueba Aceptación Soft.
- Diseño de la arquitectura software. Manuales, Plan integración
- Diseño detallado del software. Diseño y plan pruebas
- Codificación y prueba.
- Integración del Software.
- Prueba del software.
- Integración del sistema.
- Prueba del sistema.
- Instalación del Software.
- Soporte del proceso de aceptación.

ISO 12207-1

PROCESOS PRINCIPALES

ADQUISICIÓN

SUMINISTRO

DESARROLLO

EXPLOTACIÓN

- Operación y uso del software.
- Soporte operativo a los usuarios.
- Sus actividades y tareas se aplican al sistema completo.

MANTENIMIENTO

- Modificación del software o documentación por
 - Error o deficiencia
 - Mejora
 - Adaptación
 - Migración y retirada del software

ISO 12207-1

PROCESOS PRINCIPALES

ADQUISICIÓN

SUMINISTRO

DESARROLLO

EXPLOTACIÓN

MANTENIMIENTO

PROCESOS DE SOPORTE

- Sirven de apoyo al resto de procesos y se aplican en cualquier momento del ciclo de vida.

ISO 12207-1

PROCESOS PRINCIPALES

PROCESOS DE SOPORTE

DOCUMENTACIÓN

- Registra toda la información producida a lo largo de todo el ciclo de vida.
- Incluye todas las actividades
 - Planificar, diseñar, desarrollar, producir, editar, distribuir y mantener los propios documentos
- Para todas las personas involucradas
 - Directores, ingenieros, personal de desarrollo, usuarios...

ISO 12207-1

PROCESOS PRINCIPALES

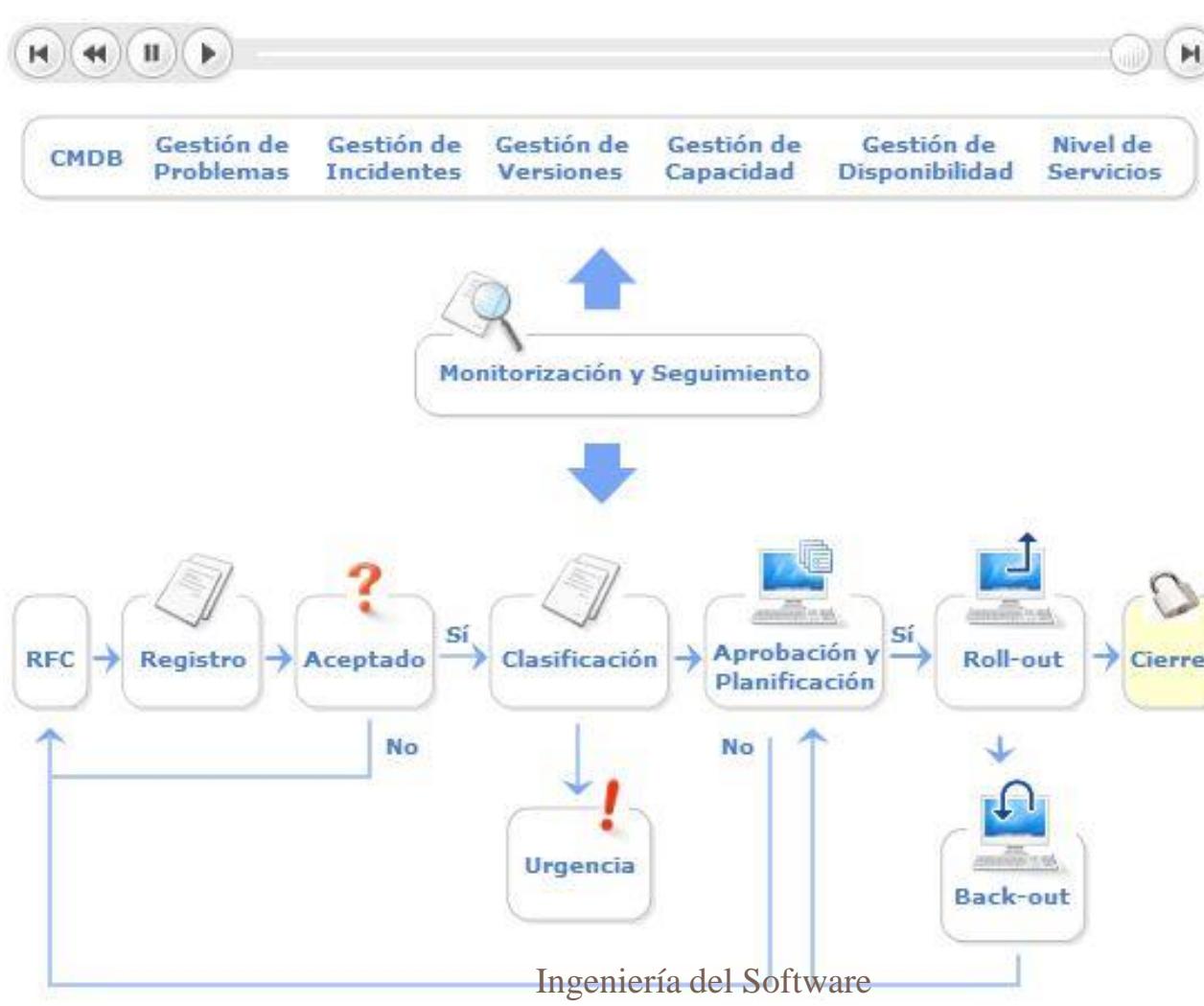
PROCESOS DE SOPORTE

DOCUMENTACIÓN

GESTIÓN DE LA CONFIGURACIÓN

- **Aplica procesos administrativos y técnicos durante todo el ciclo de vida para:**
 - Identificar, definir y establecer la línea de base de los elementos configurables del software del sistema.
 - Hacer el control de cambio de los elementos.
 - Registrar e informar del estado de elementos y sus peticiones de modificación
 - Asegurar sobre la compleción, consistencia y corrección
 - Controlar almacenamiento, manipulación y entrega.

Ejemplo: Diagrama flujo proceso Gestión del Cambio



ISO 12207-1

PROCESOS PRINCIPALES

PROCESOS DE SOPORTE

DOCUMENTACIÓN

GESTIÓN DE LA CONFIGURACIÓN

VERIFICACIÓN

- Determina si los requisitos están completos y son correctos
- Comprueba que los productos de cada fase cumplen los requisitos y condiciones impuestos sobre ellos en las anteriores.
 - Ej. Código coherente con el diseño.
- Interno / Externo

Verificación: Resultados

7.2.4.2 Outcomes

As a result of successful implementation of the Software Verification Process:

- a) a verification strategy is developed and implemented;
- b) criteria for verification of all required software work products is identified;
- c) required verification activities are performed;
- d) defects are identified and recorded; and
- e) results of the verification activities are made available to the customer and other involved parties.

- **7.2.4.3.2 Verification.** This activity consists of the following tasks:
 - **7.2.4.3.2.1 Requirements verification.**
 - **7.2.4.3.2.2 Design verification**
 - **7.2.4.3.2.3 Code verification**
 - **7.2.4.3.2.4 Integration verification**
 - **7.2.4.3.2.5 Documentation verification**

ISO 12207-1

PROCESOS PRINCIPALES

PROCESOS DE SOPORTE

DOCUMENTACIÓN

GESTIÓN DE LA CONFIGURACIÓN

VALIDACIÓN

- Determina si el software o sistema final cumple con los requisitos para su uso
- Externo / Interno

ISO 12207-1

PROCESOS PRINCIPALES

PROCESOS DE SOPORTE

DOCUMENTACIÓN

GESTIÓN DE LA CONFIGURACIÓN

REVISIÓN CONJUNTA

- Sirve para evaluar con los stakeholders el estado del desarrollo del software y sus productos en un determinado punto del proyecto.
- Nivel de gestión / técnico

ISO 12207-1

PROCESOS PRINCIPALES

PROCESOS DE SOPORTE

DOCUMENTACIÓN

GESTIÓN DE LA CONFIGURACIÓN

AUDITORÍA

- Permite determinar, en los hitos predeterminados, si se han cumplido los requisitos, los planes y el contrato.
- Puede ser interna o externa
- Sigue la rueda de Deming PDCA

ISO 12207-1

PROCESOS PRINCIPALES

PROCESOS DE SOPORTE

DOCUMENTACIÓN

GESTIÓN DE LA CONFIGURACIÓN

ASEGURAMIENTO DE LA CALIDAD

- Aporta la confianza en que los productos y procesos cumplen los requisitos y se ajustan a lo previsto.
- Interno / Externo
- Utiliza los resultados de otros procesos
 - Verificación, validación, revisiones conjuntas, auditoria y resolución de problemas.

Aseguramiento de la calidad: Resultados

- Outcomes:
 - a) a strategy for conducting quality assurance is developed;
 - b) evidence of software quality assurance is produced and maintained;
 - c) problems and/or non-conformance with requirements are identified and recorded;
 - d) adherence of products, processes and activities to the applicable standards, procedures and requirements are verified.

ISO 12207-1

PROCESOS PRINCIPALES

PROCESOS DE SOPORTE

DOCUMENTACIÓN

GESTIÓN DE LA CONFIGURACIÓN

ASEGURAMIENTO DE LA CALIDAD

VERIFICACIÓN

VALIDACIÓN

REVISIÓN CONJUNTA

AUDITORÍA

RESOLUCIÓN DE PROBLEMAS.

- Analizar y eliminar problemas descubiertos en el desarrollo o cualquier otro proceso.
- Objetivo: Aportar un medio que asegure que todos los problemas descubiertos se identifican, analizan, gestionan y controlan, para su resolución.

ISO 12207-1



ISO 12207-1

PROCESOS PRINCIPALES

PROCESOS DE SOPORTE

PROCESOS DE LA ORGANIZACIÓN

GESTIÓN

- Actividades
 - Planificación
 - Seguimiento
 - Control
 - Revisión
 - Evaluación

INFRAESTRUCTURA

- Dotar los procesos de infraestructura
 - Hardware
 - Software
 - Herramientas
 - Técnicas
 - Normas

MEJORA

- Permite establecer, valorar, medir, controlar y mejorar los procesos del ciclo de vida.

FORMACIÓN

- Mantener al personal formado lo que incluye materiales y planes de formación.

ISO 12207-1

PROCESOS PRINCIPALES

ADQUISICIÓN

SUMINISTRO

DESARROLLO

EXPLORACIÓN

MANTENIMIENTO

PROCESOS DE SOPORTE

DOCUMENTACIÓN

GESTIÓN DE CONFIGURACIÓN

ASEGURAMIENTO DE CALIDAD

VERIFICACIÓN

VALIDACIÓN

REVISIÓN CONJUNTA

AUDITORÍA

RESOLUCIÓN DE PROBLEMAS

PROCESOS DE LA ORGANIZACIÓN

GESTIÓN

INFRAESTRUCTURA

MEJORA

FORMACIÓN

ISO/IEC TR 15504-2 (1998)

Procesos principales

CUS. 1 Adquisición

CUS. 2 Suministro

CUS. 3 Obtención
de requisitos

CUS. 4 Explotación

ENG. 1 Desarrollo

ENG. 2 Mantenimiento del sistema y el
software

Procesos de soporte

SUP. 1 Documentación

SUP. 2 Gestión de configuración

SUP. 3 Aseguramiento de calidad

SUP. 4 Verificación

SUP. 5 Validación

SUP. 6 Revisión conjunta

SUP. 7 Auditoría

SUP. 8 Resolución de problemas

Procesos de la organización

MAN. 1 Gestión

MAN. 2 Gestión del proyecto

MAN. 3 Gestión de la calidad

MAN. 4 Gestión del riesgo

ORG. 1 Alineamiento con la organización

ORG. 2 Mejora

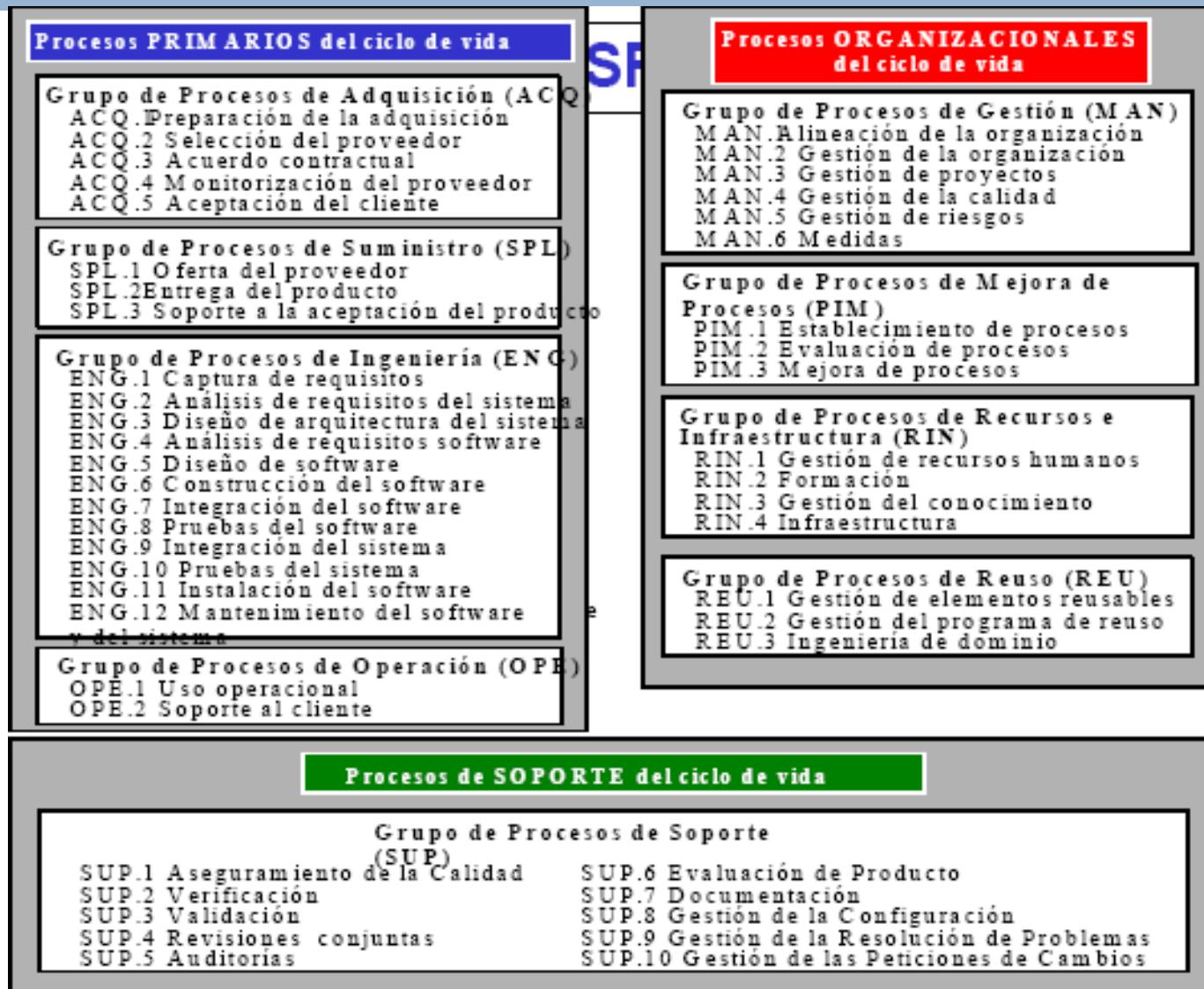
ORG. 3 Gestión de recursos humanos

ORG. 4 Infraestructura

ORG. 5 Medida

ORG. 6 Reutilización

ISO/IEC TR 15504-2 (2003)



Descripción de procesos en 12207: 2008.

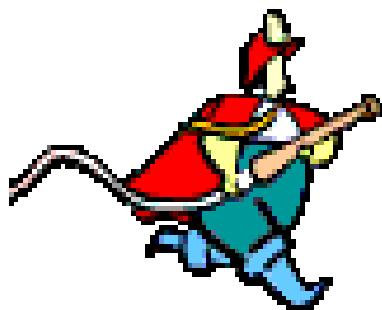
- **Identifier:** This identifies the process category and the sequential number.
- **Title:** A descriptive phrase. It conveys the scope of the process as a whole
- **Purpose:** A paragraph that states the purpose of the process. Describes the goals of performing the process
- **Outcomes:** A process outcome is an observable result
- **Activities and Tasks:**
 - **Activities** are a list of actions that are used to achieve the outcomes
 - **Task:** are requirements, recommendations, or permissible.

Áreas de Proceso por categoría en CMMI

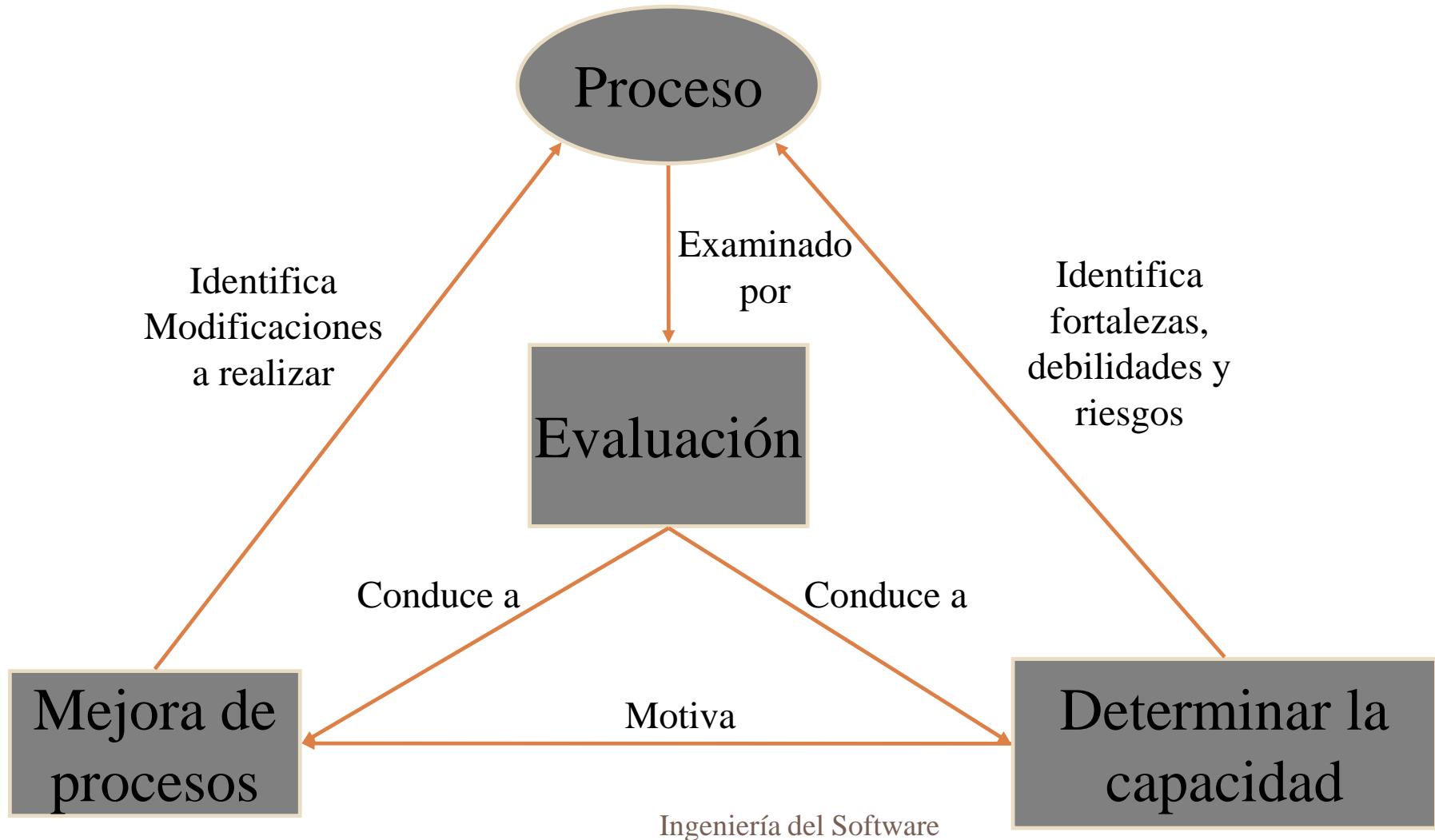
PA		The Process Management process areas:	PA		The Engineering process areas:
	OPF	Organizational Process Focus		RD	Requirements Development
	OPD	Organizational Process Definition		REQM	Requirements Management
	OT	Organizational Training		TS	Technical Solution
	OPP	Organizational Process Performance		PI	Product Integration
	OID	Organizational Innovation and Deployment		VER	Verification
PA		The Project Management process areas:		VAL	Validation
	PP	Project Planning	PA		The Support process areas:
	PMC	Project Monitoring and Control		CM	Configuration Management
	SAM	Supplier Agreement Management		PPQA	Process and Product Quality Assurance
	IPM	Integrated Project Management for IPPD		MA	Measurement and Analysis
	RSKM	Risk Management		DAR	Decision Analysis and Resolution
	QPM	Quantitative Project Management		CAR	Causal Analysis and Resolution

Evaluación de procesos software

- La calidad del proceso es relevante en la calidad del proceso final
- Cuestiones
 - ¿En que situación se encuentran los procesos de nuestra organización?
 - ¿Qué podemos hacer para mejorar?



Evaluación de procesos software



Beneficios del CMMI

Rendimientos medidos	Mejora Media
Coste	34%
Plazos	50%
Productividad	61%
Calidad	48%
Satisfacción del cliente	14%
Retorno de la inversión	4:1

Fuente: SEI a partir de un estudio sobre 30 organizaciones

Ingeniería del Software

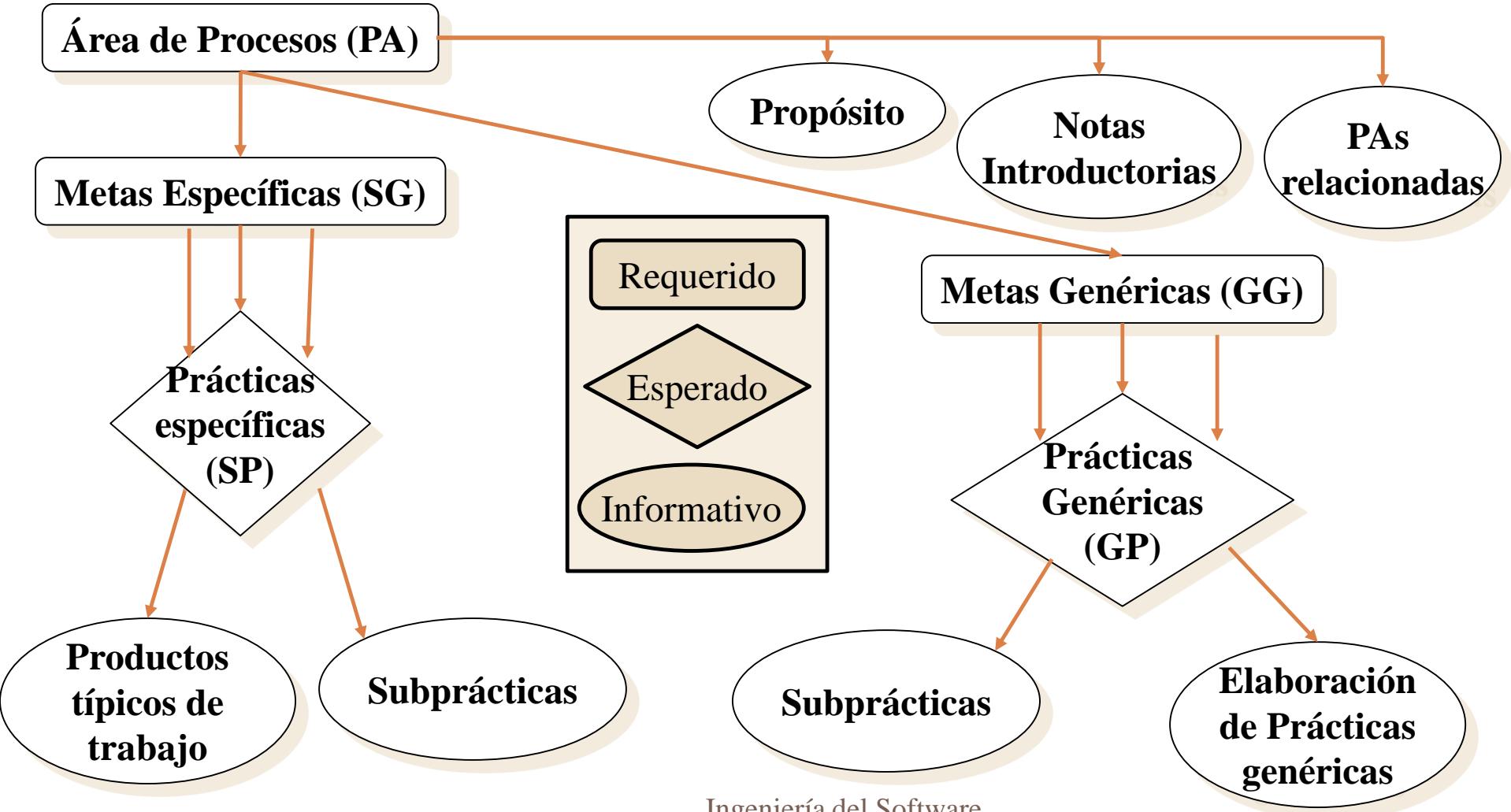
Beneficios del CMMI

- Mejora la satisfacción de los empleados
 - Evitar situaciones de crisis
 - Favorecen la “sobreasignación” del trabajador.
 - Procesos formalizados.
 - Sabemos que hay que hacer.
 - Se fijan responsabilidades.
 - Aseguramiento de la calidad
 - Se busca las causas de los fallos.
 - Asegurar la formación
 - Proporciona medios para garantizar que sabemos qué hay que hacer y cómo

Integración del Modelo de Capacidad de la Madurez (CMMI).

- Dos representaciones para cada modelo CMMI
 - Representación continua
 - Enfoque: Capacidad de los procesos
 - Las organizaciones eligen áreas del proceso en las que quieren incidir para la mejora continua.
 - Por etapas o discreta
 - Enfoque: Madurez de la organización.
 - Se sigue un camino predeterminado.

Componentes de las PA



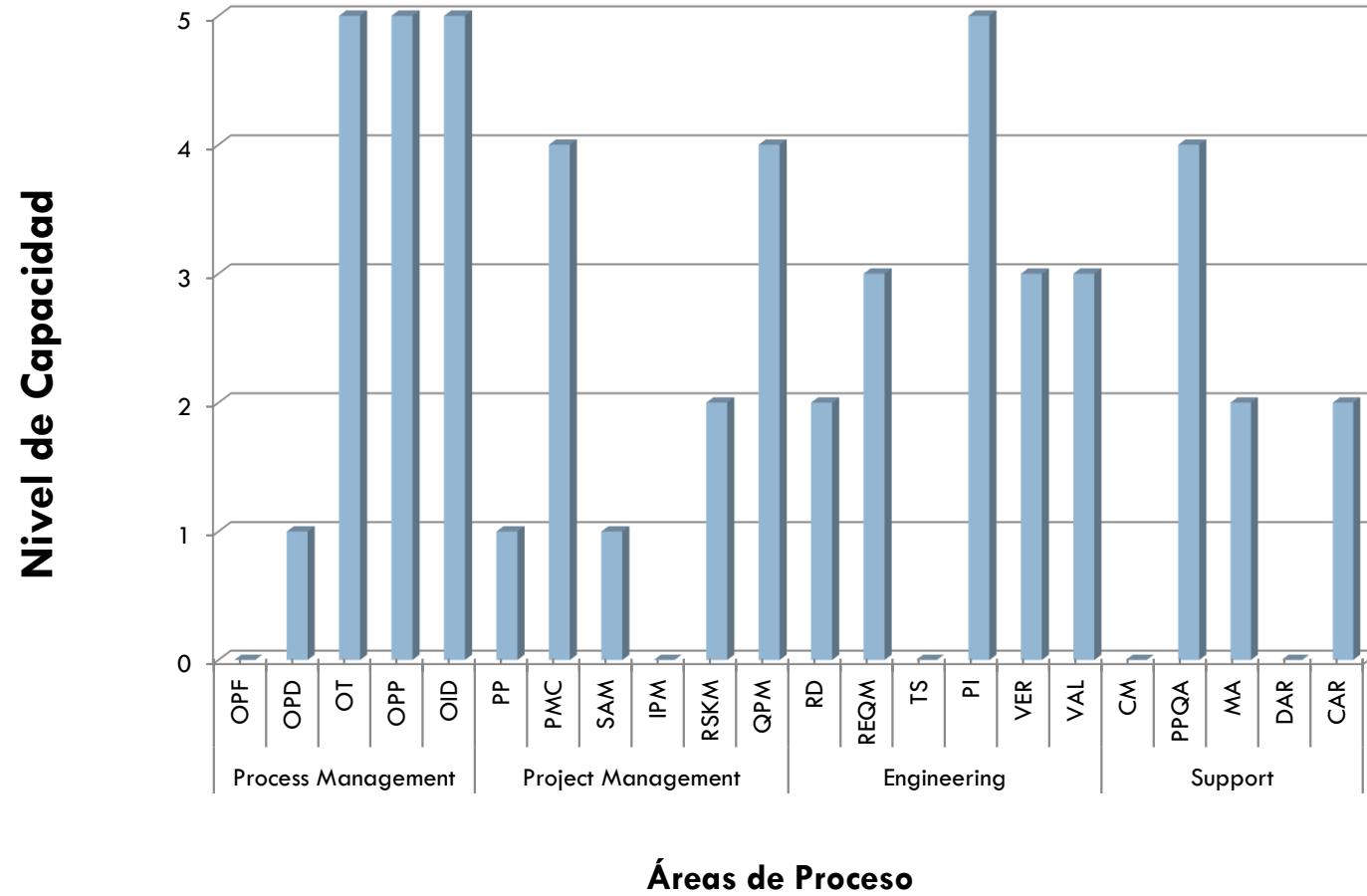
CMMI. Continuo

- Organiza en 4 categorías las 22 Áreas de Proceso (PA).
- Hay 5 Metas genéricas (GG).
 1. Alcanzar las metas específicas
 2. Institucionalizar un proceso gestionado
 3. Institucionalizar un proceso definido
 4. Institucionalizar un proceso cuantitativamente gestionado
 5. Institucionalizar un proceso en optimización

Áreas de Proceso por categoría en el CMMI continuo

PA		The Process Management process areas:	PA		The Engineering process areas:
	OPF	Organizational Process Focus		RD	Requirements Development
	OPD	Organizational Process Definition		REQM	Requirements Management
	OT	Organizational Training		TS	Technical Solution
	OPP	Organizational Process Performance		PI	Product Integration
	OID	Organizational Innovation and Deployment		VER	Verification
PA		The Project Management process areas:	PA		The Support process areas:
	PP	Project Planning		CM	Configuration Management
	PMC	Project Monitoring and Control		PPQA	Process and Product Quality Assurance
	SAM	Supplier Agreement Management		MA	Measurement and Analysis
	IPM	Integrated Project Management for IPPD		DAR	Decision Analysis and Resolution
	RSKM	Risk Management		CAR	Causal Analysis and Resolution
	QPM	Quantitative Project Management			

Modelo CMMI continuo.



CMMI continuo: Gestión de Requisitos REQM

Nivel de Capacidad 0: Incompleto

Prácticas específicas

SP1.1: Obtain an Understanding of Requirements

- Al menos una de las prácticas específicas no se realiza

SP1.2: Obtain Commitment to Requirements

SP1.3: Manage Requirements Changes

SP1.4: Maintain Bidirectional Traceability of Requirements

SP1.5: Identify Inconsistencies Between Project Work and Requirements

CMMI continuo: Gestión de Requisitos REQM

Nivel de Capacidad 1: Realizado

Prácticas específicas

SP1.1: Obtain an Understanding of Requirements

SP1.2: Obtain Commitment to Requirements

SP1.3: Manage Requirements Changes

SP1.4: Maintain Bidirectional Traceability of Requirements

SP1.5: Identify Inconsistencies Between Project Work and Requirements

Prácticas Generales

GP1.1: Perform Specific Practices

CMMI continuo. Gestión de Requisitos REQM

Nivel de Capacidad 2: Gestionado

Prácticas específicas

- SP1.1: Obtain an Understanding of Requirements
- SP1.2: Obtain Commitment to Requirements
- SP1.3: Manage Requirements Changes
- SP1.4: Maintain Bidireccional Traceability of Requirements
- SP1.5: Identify Inconsistencies Between Project Work and Requirements

Prácticas Generales

- GP2.1: Establish an Organizational Policy**
- GP2.2: Plan the Process**
- GP2.3: Provide Resources**
- GP2.4: Assign Responsibility**
- GP2.5: Train People**
- GP2.6: Manage Configurations**
- GP2.7: Identify and Involve Relevant Stakeholders**
- GP2.8: Monitor and Control the Process**
- GP2.9: Objectively Evaluate Adherence**
- GP2.10: Review Status with Higher Level Management**

CMMI continuo. Gestión de Requisitos REQM

Nivel de Capacidad 3: Definido

Prácticas específicas

- SP1.1: Obtain an Understanding of Requirements
- SP1.2: Obtain Commitment to Requirements
- SP1.3: Manage Requirements Changes
- SP1.4: Maintain Bidireccional Traceability of Requirements
- SP1.5: Identify Inconsistencies Between Project Work and Requirements

Prácticas Generales

- GP2.1: Establish an Organizational Policy
 - GP2.2: Plan the Process
 - GP2.3: Provide Resources
 - GP2.4: Assign Responsibility
 - GP2.5: Train People
 - GP2.6: Manage Configurations
 - GP2.7: Identify and Involve Relevant Stakeholders
 - GP2.8: Monitor and Control the Process
 - GP2.9: Objectively Evaluate Adherence
 - GP2.10: Review Status with Higher Level Management
- GP3.1: Establish a Defined Process**
- GP3.1: Collect Improvement Information**
- SP1.2: Ingeniería del Software

CMMI continuo. Gestión de Requisitos REQM

Nivel de Capacidad 4 & 5: GC y OPT

Prácticas específicas

Todas las prácticas Específicas

Prácticas Generales

Todas las prácticas generales de los niveles
1, 2 y 3 +

**GP4.1: Establish Quantitative Objectives
for the Process**

GP4.2: Stabilize Subprocess Performance

Prácticas específicas

Todas las prácticas Específicas

Prácticas Generales

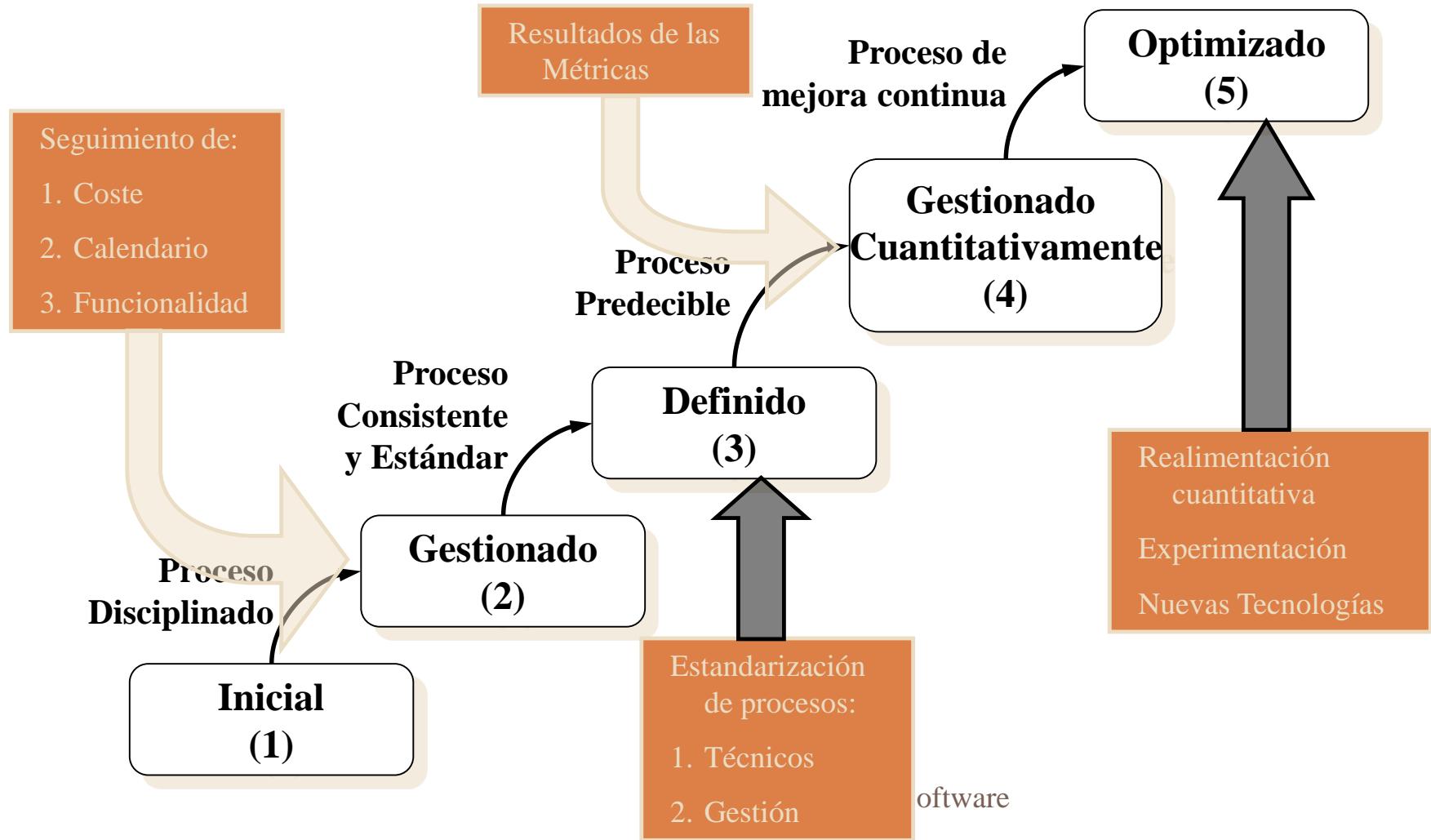
Todas las prácticas generales de los niveles
1, 2, 3 y 4+

**GP5.1: Ensure Continuous Process
Improvement**

GP5.2: Correct Root Causes of Problems

SP1.2: Ingeniería del Software

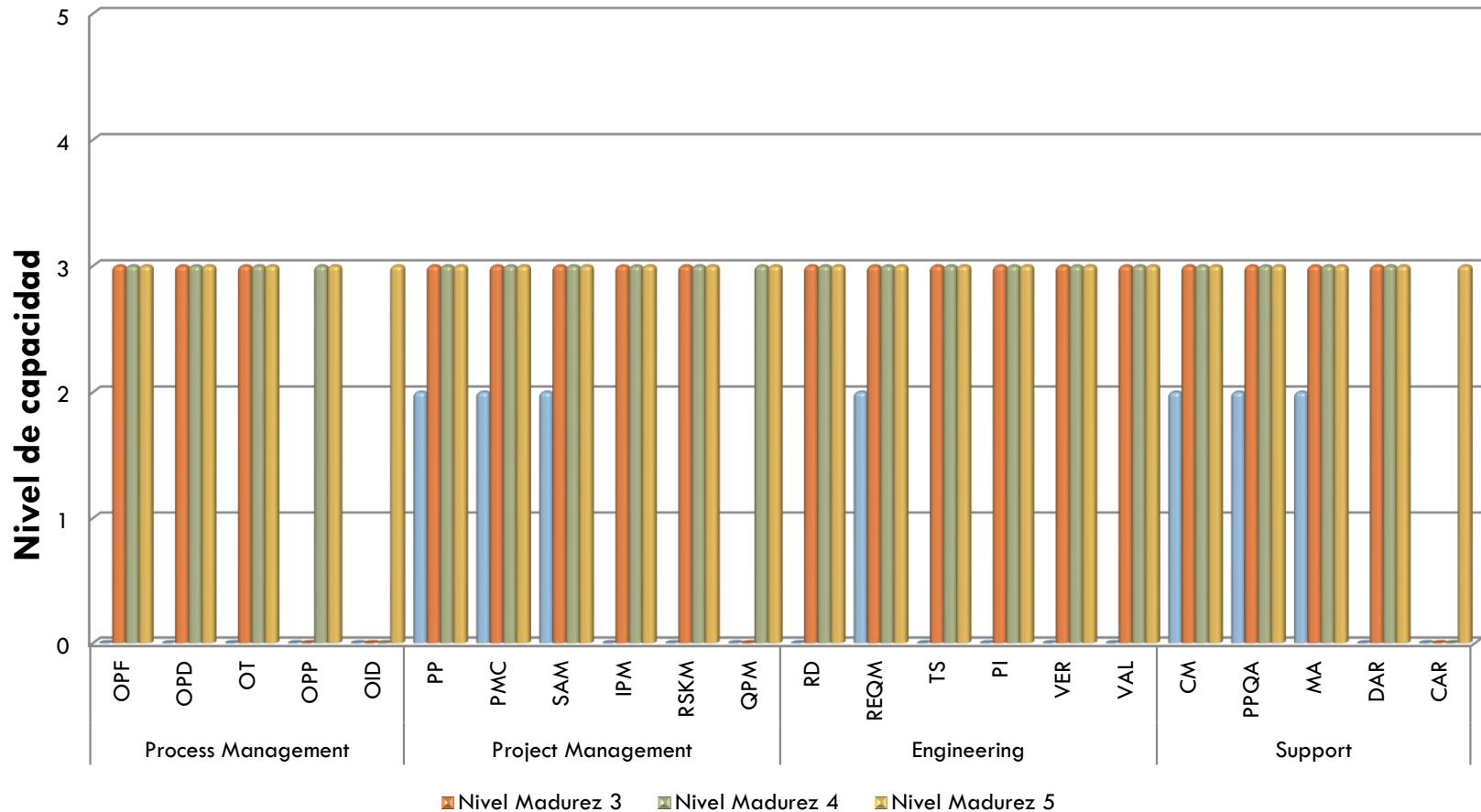
CMMI. Niveles de madurez



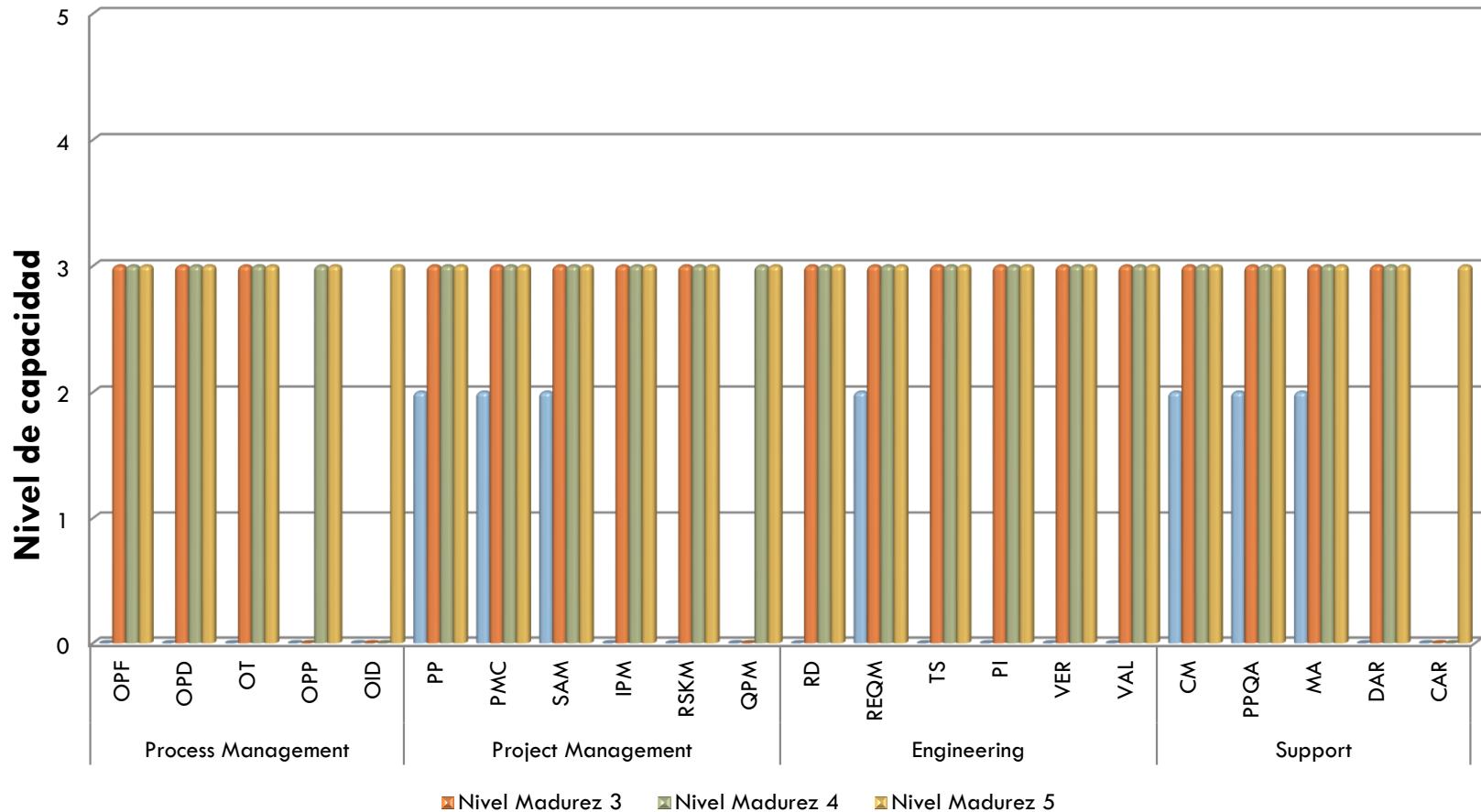
Representación por niveles

Nivel	Foco	Área de proceso	Metas Generales	Calidad/ Productivi- dad
5 En optimización	Mejora continua de procesos	OID, CAR	Todas las GG1 + Todas las GG2 + GG3: Institucionalizar un proceso definido	
4 Gestionado cuantitativamente	Gestión cuantitativa	OPP, QPM		
3 Definido	Estandarización de procesos	RD, TS, PI, VER, VAL, OPF, OPD, OT, IPM, RSKM,DAR		
2 Gestionado	Gestión básica de proyectos	REQM, PP, PCM, SAM, MA, PPQA, CM	GG1 + GG2: Institucionalizar un proceso gestionado	Retrabajo Riesgo
1 Inicial		Ingeniería del Software		

Modelo Continuo vs Discreto.

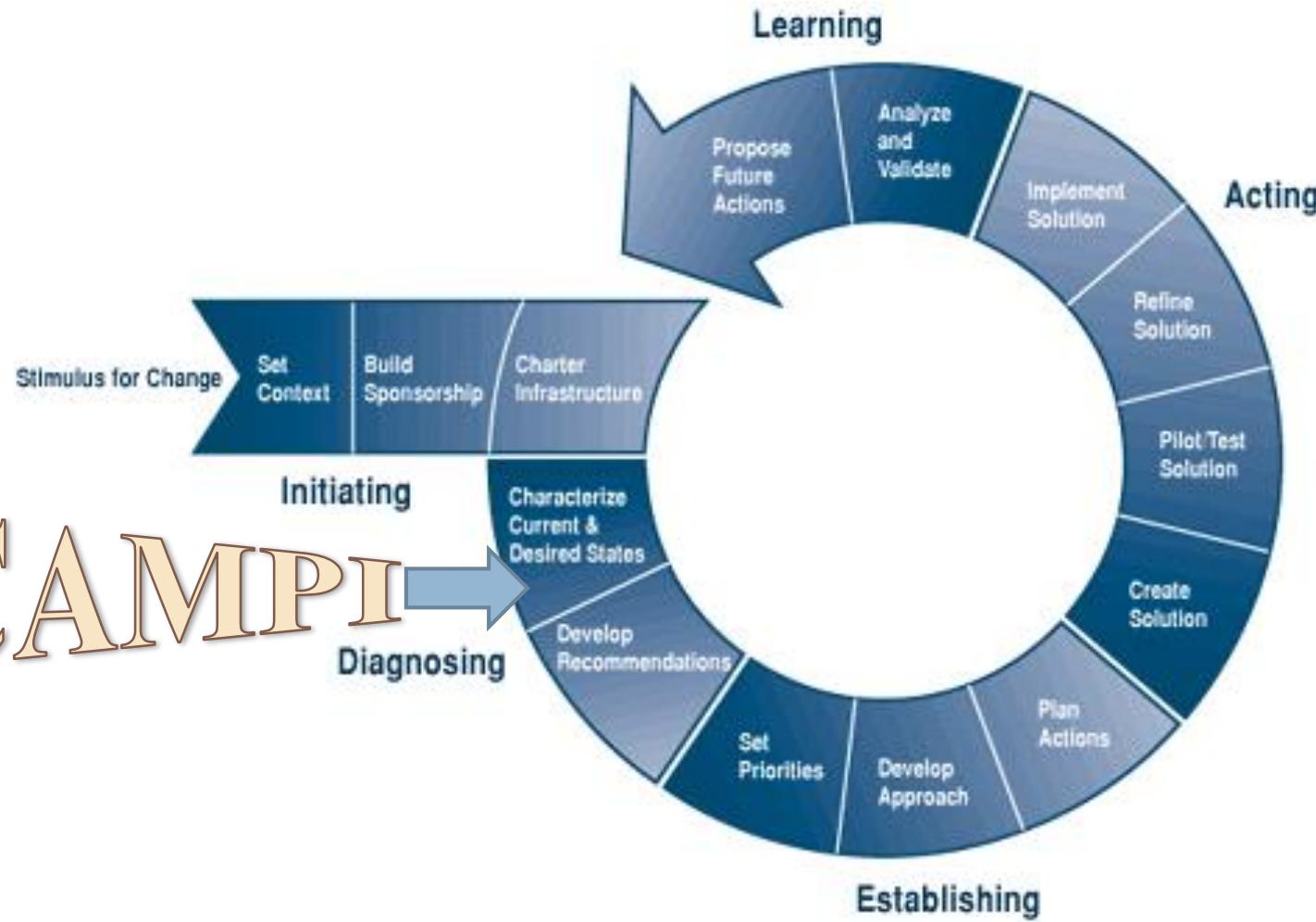


Modelo Continuo vs Discreto.

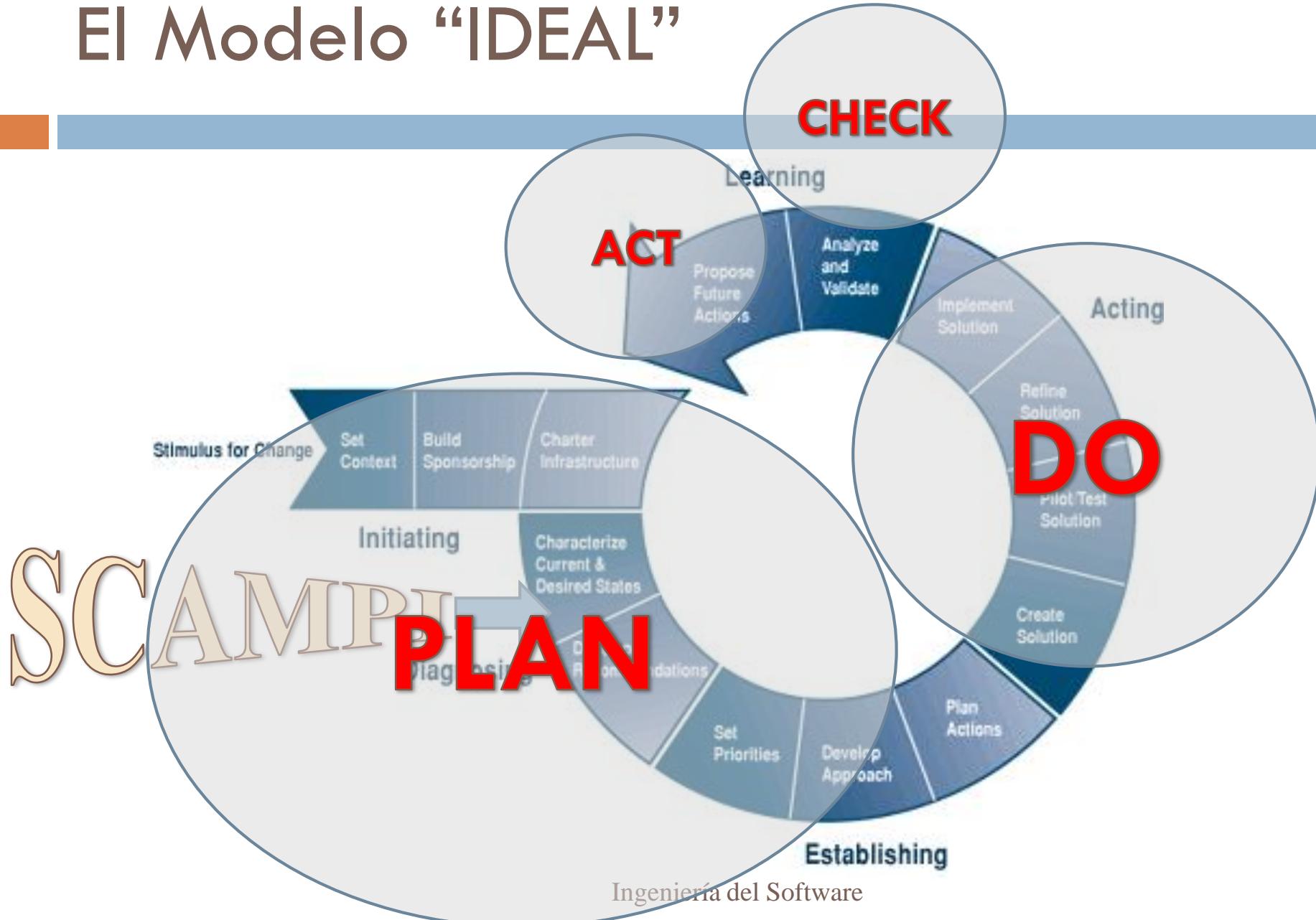


El Modelo “IDEAL”

SCAMPI



El Modelo “IDEAL”



Bibliografía

- 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
- Pressman, R.S.
 - Ingeniería del Software. Un enfoque práctico
 - 5^a Edición. 2002
 - 6^a Edición. 2005
- Sommersville, I.
 - Ingeniería de Software. 6^a Edición 2002
 - Ingeniería de Software. 9^a Edición 2011
- Weitzenfeld, A.
 - Ingeniería de Software. Orientada a objetos con UML. Java e Internet
- <http://www.sei.cmu.edu/cmmi/>

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.

Bibliografía

- Pressman, R.S.
 - Ingeniería del Software. Un enfoque práctico
 - 6^a Edición. 2005
 - 5^a Edición. 2002
- Sommersville, I
 - Ingeniería de Software. 6^a 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

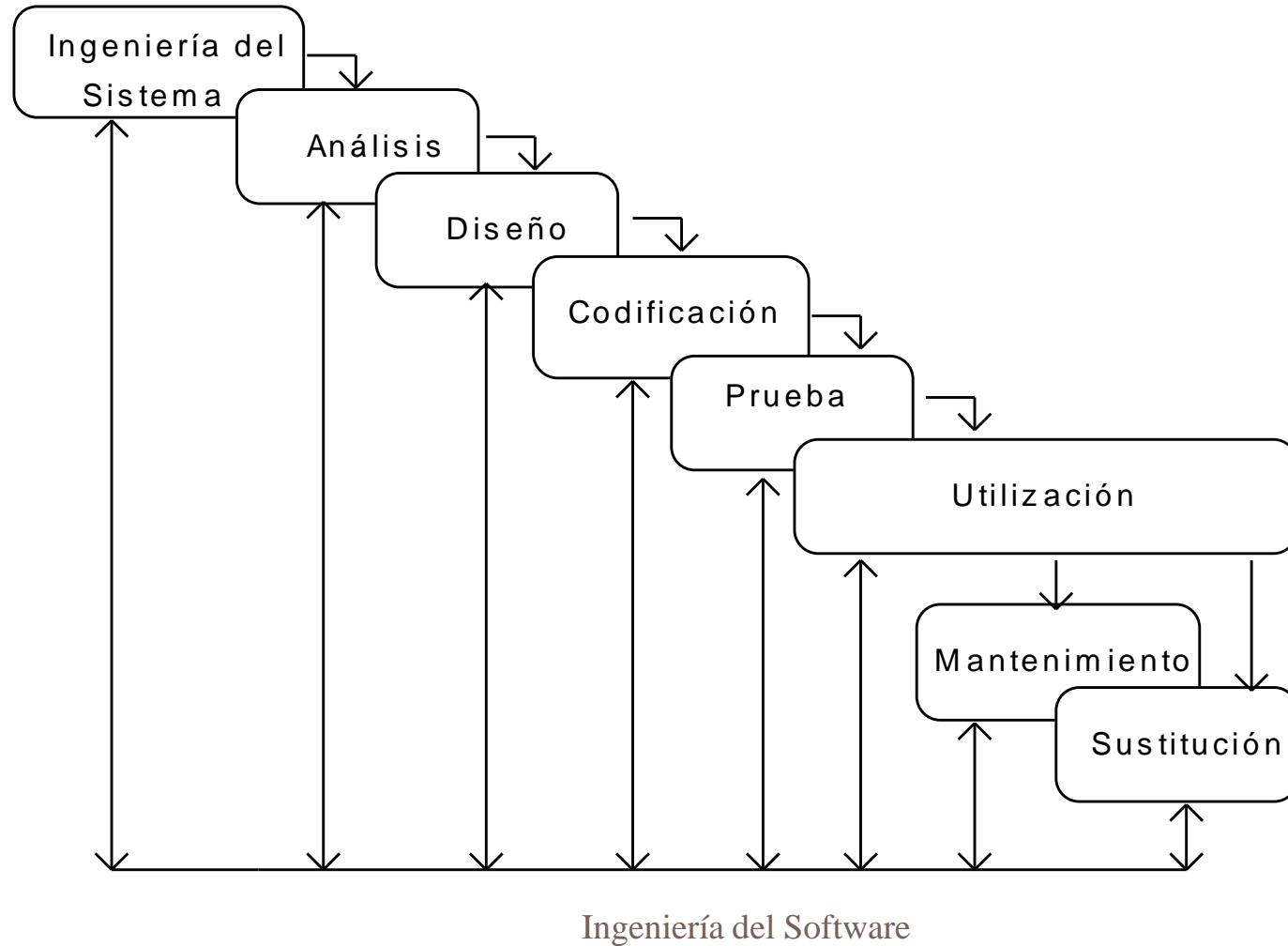
Ciclo de vida

- 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) y usado
- 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



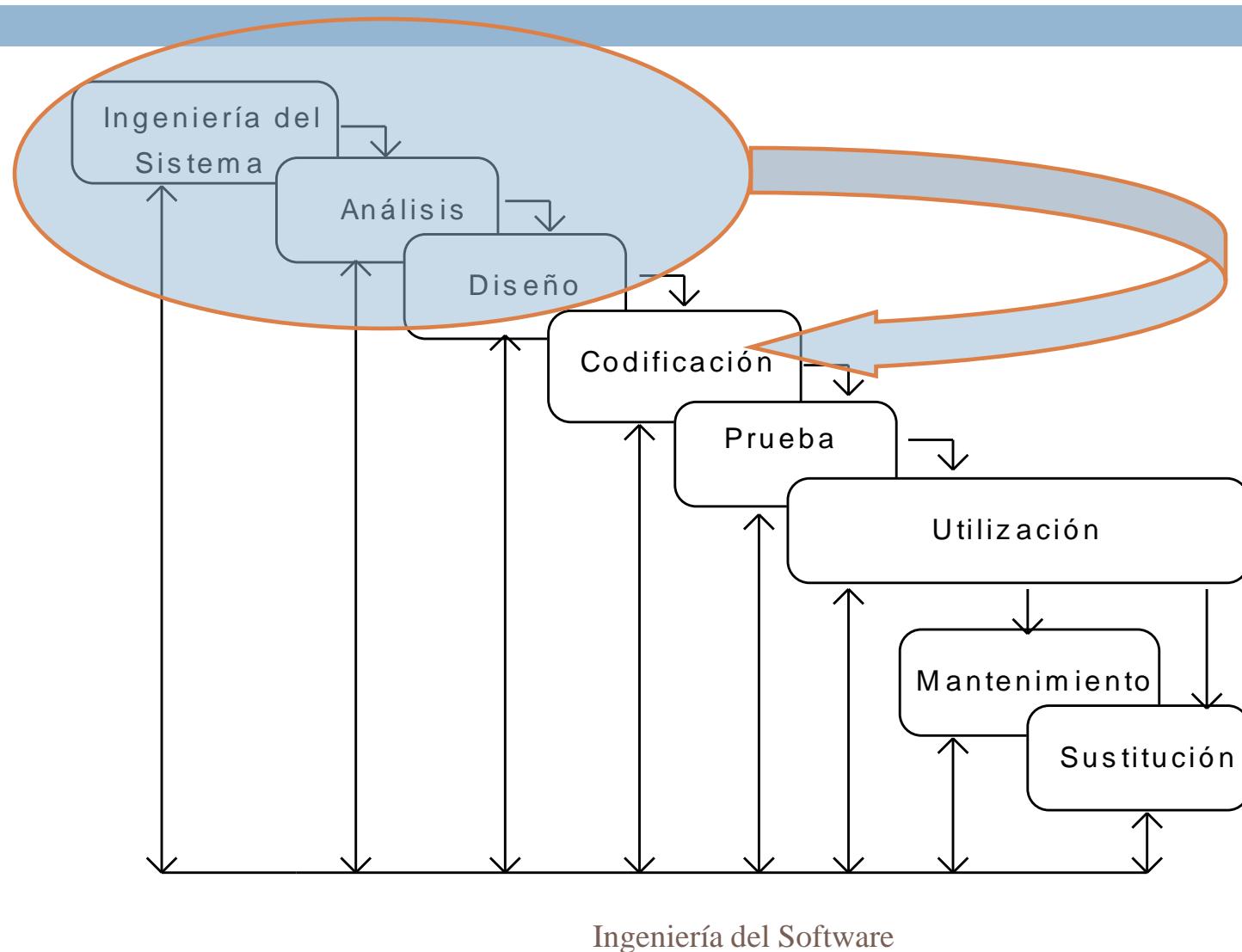
Documentos del Modelo en Cascada

Proceso	Documentos Producidos
Especificación del Sistema.	Especificación Funcional. Arquitectura del sistema.
Análisis de Requisitos	Documento de Requisitos
Definición de Requisitos	Documento de Requisitos.
Diseño de la Arquitectura del software	Especificación de la Arquitectura
Diseño de Interfaces	Especificación de la Interfaces.
Diseño Detallado	Especificación del diseño.
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 a desarrollar, interfaces, etc.
- Diseño:
 - Estructura de datos, arquitectura de aplicaciones, estructura interna de los programas y las interfaces.

Ciclo de vida en cascada

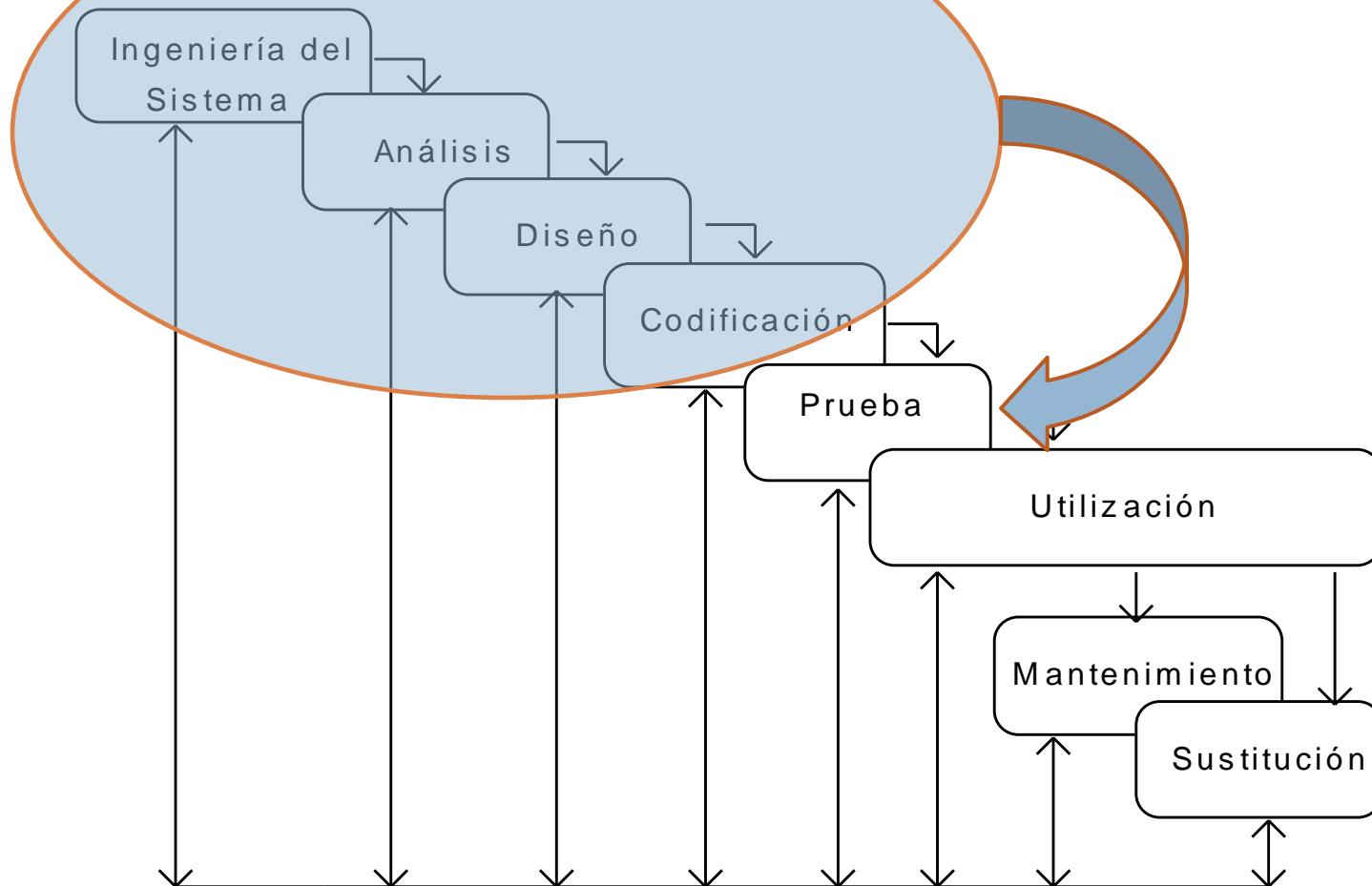


Ciclo de vida en cascada

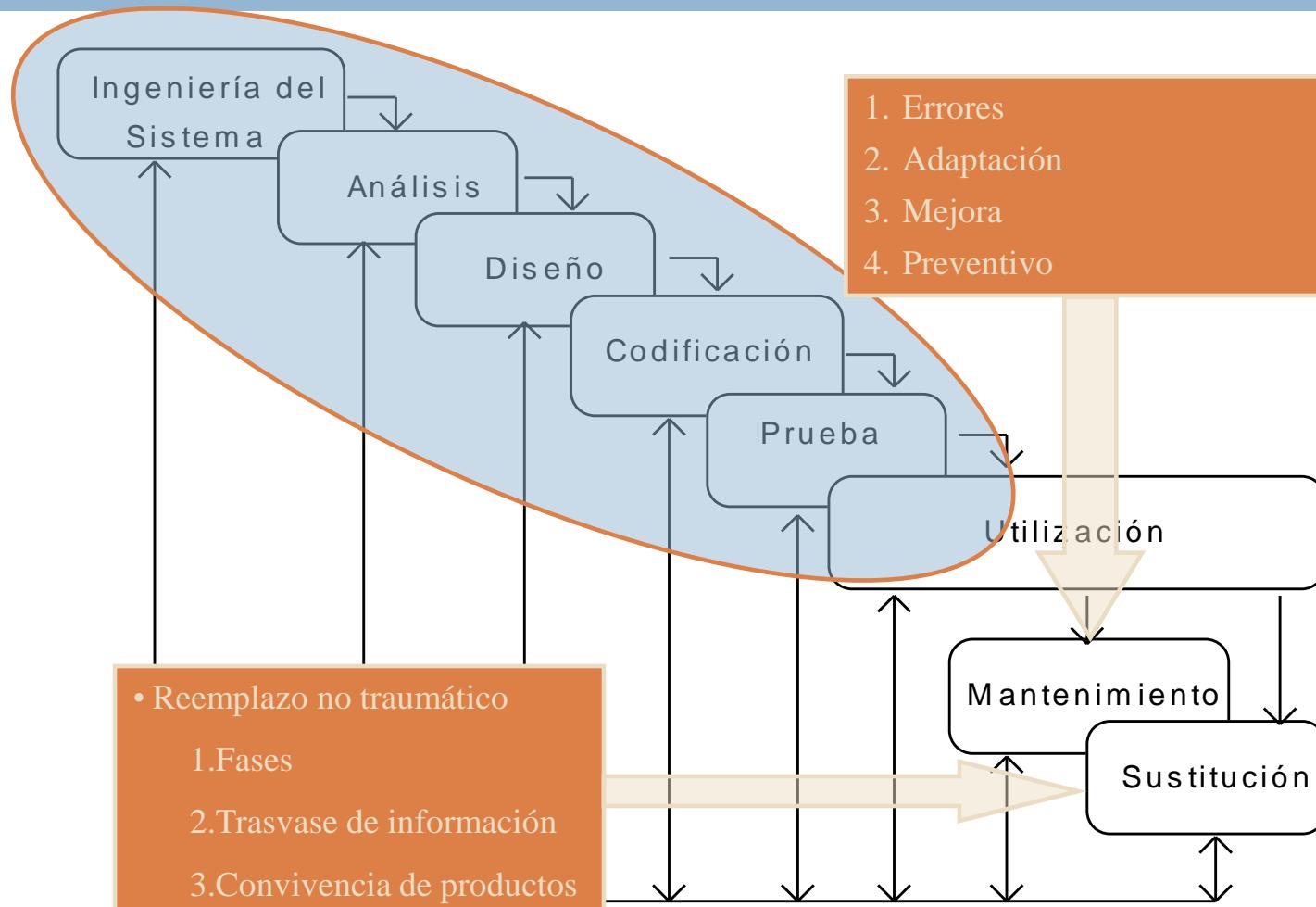
□ Codificación

- 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



Ciclo de vida en cascada



Ciclo de vida en cascada

□ Aportaciones

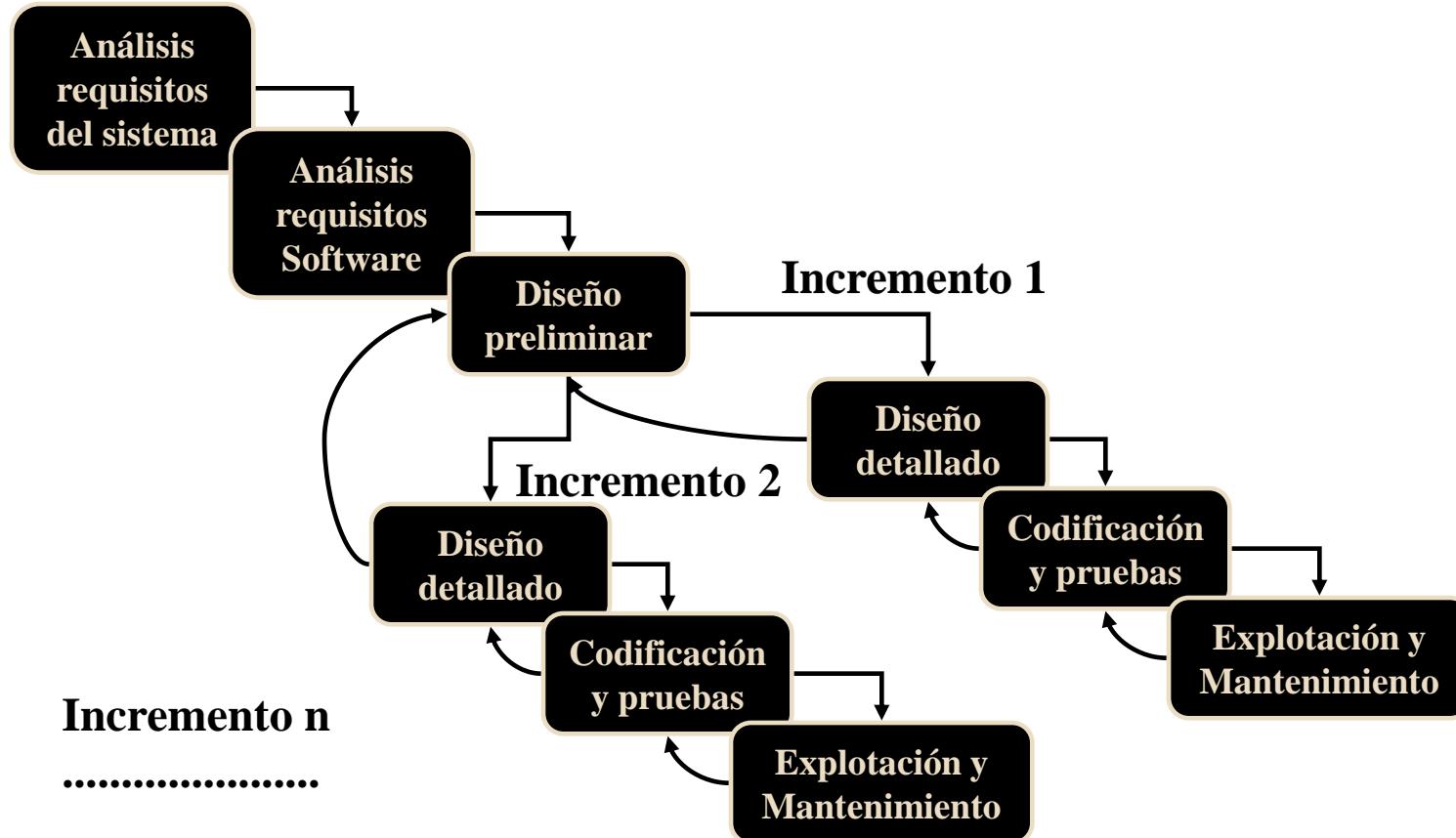
- Es el más simple, el más conocido y el más fácil de usar.
- Define una serie de fases o procesos a realizar que posteriormente se formalizaron en las normas.
- Permite generar software eficientemente y de acuerdo con las especificaciones.
 - Ayuda a prevenir que se sobrepasen las fechas de entrega 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**
- **En la realidad el ciclo de vida no es secuencial, sino que hay iteraciones, exige refinamiento.**
- 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
- 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 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

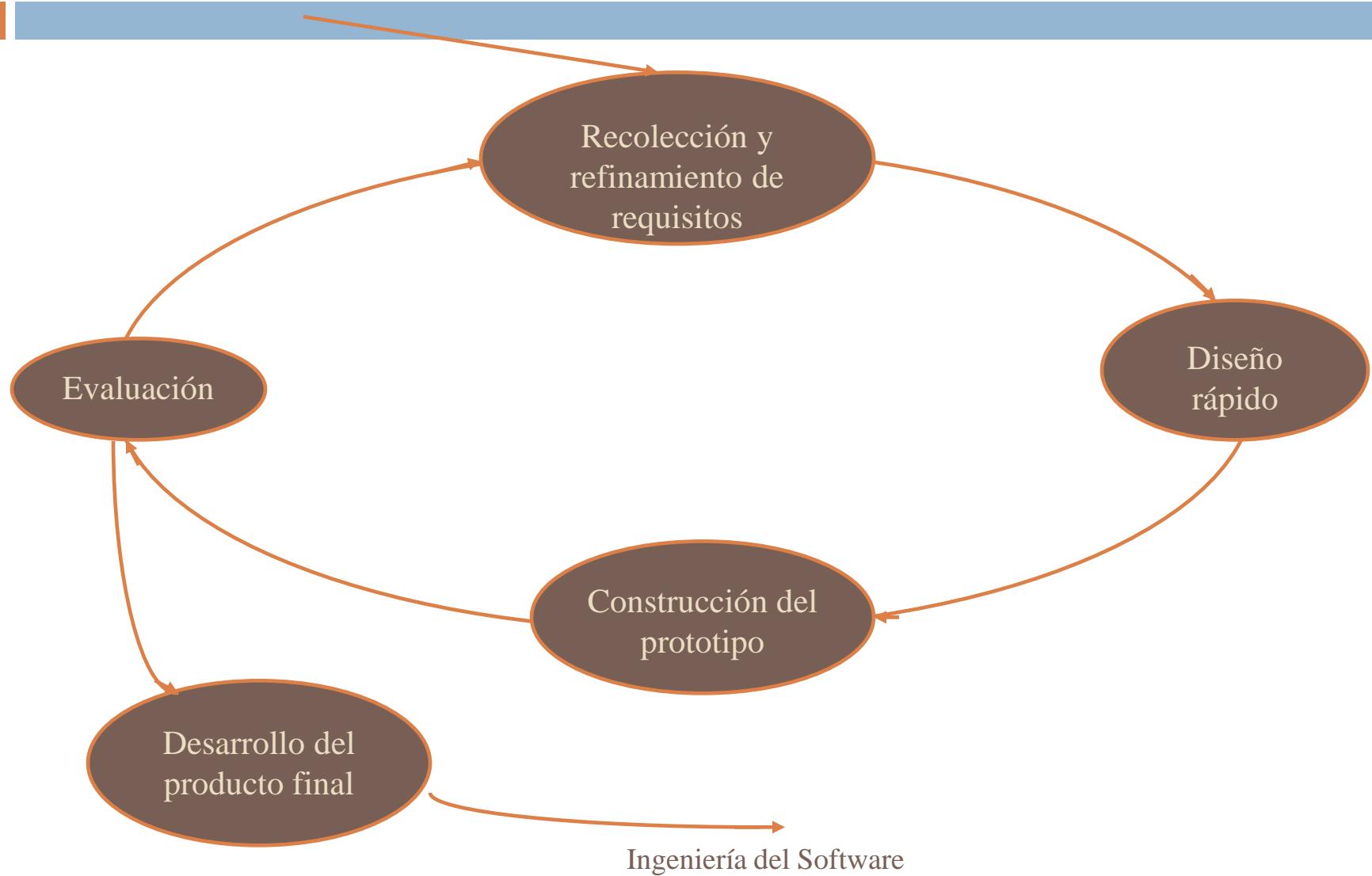
□ Tipos

- 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

- Elección
 - Nivel muy alto de incertidumbre
 - Mucha interacción con el usuario, algoritmos refinables
 - Disposición del cliente a probar los prototipos.
- 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

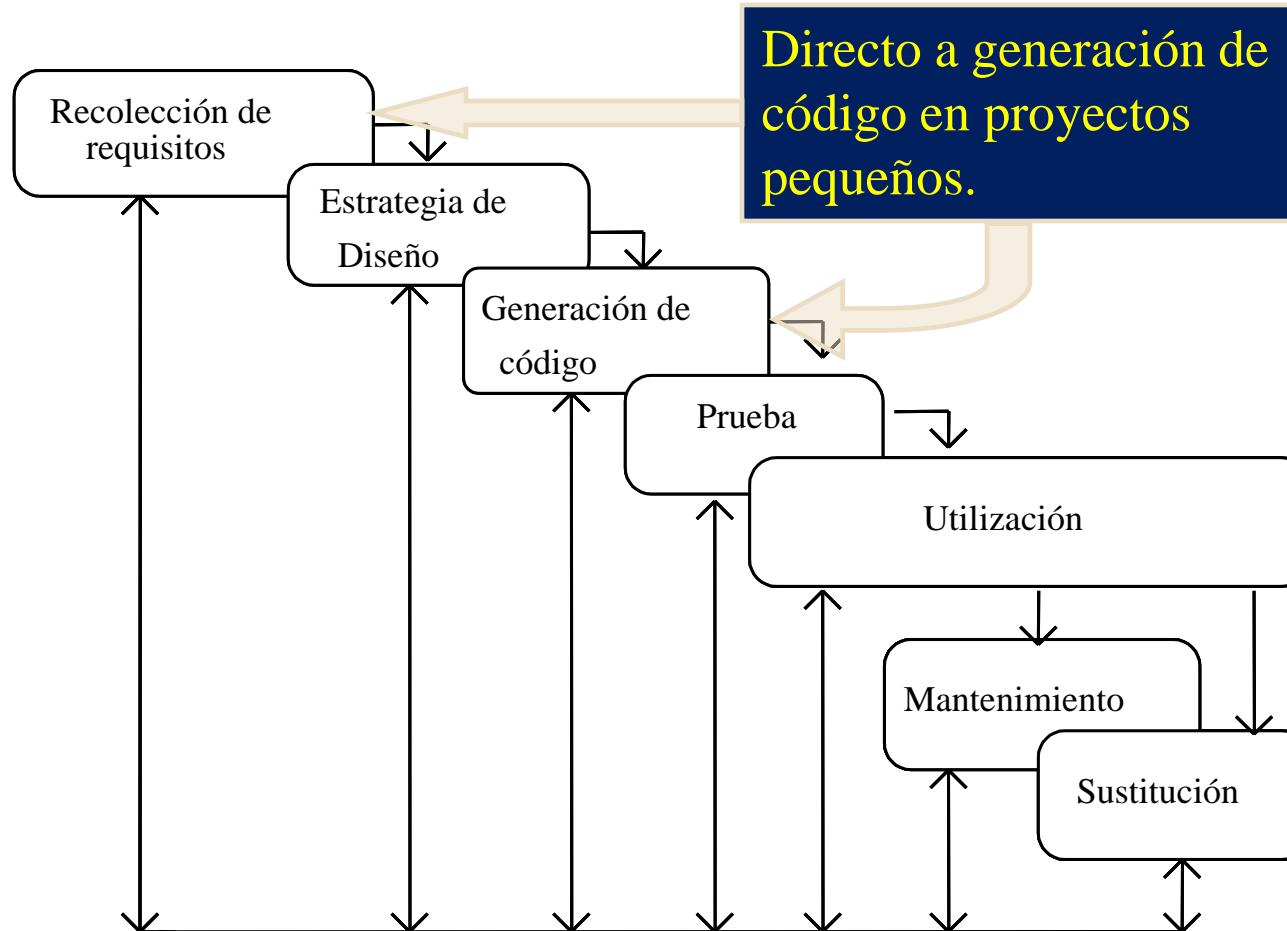
□ Desventajas:

- 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.
 - Baja eficiencia del desarrollo.
 - Poco fiable
 - Difícil de mantener

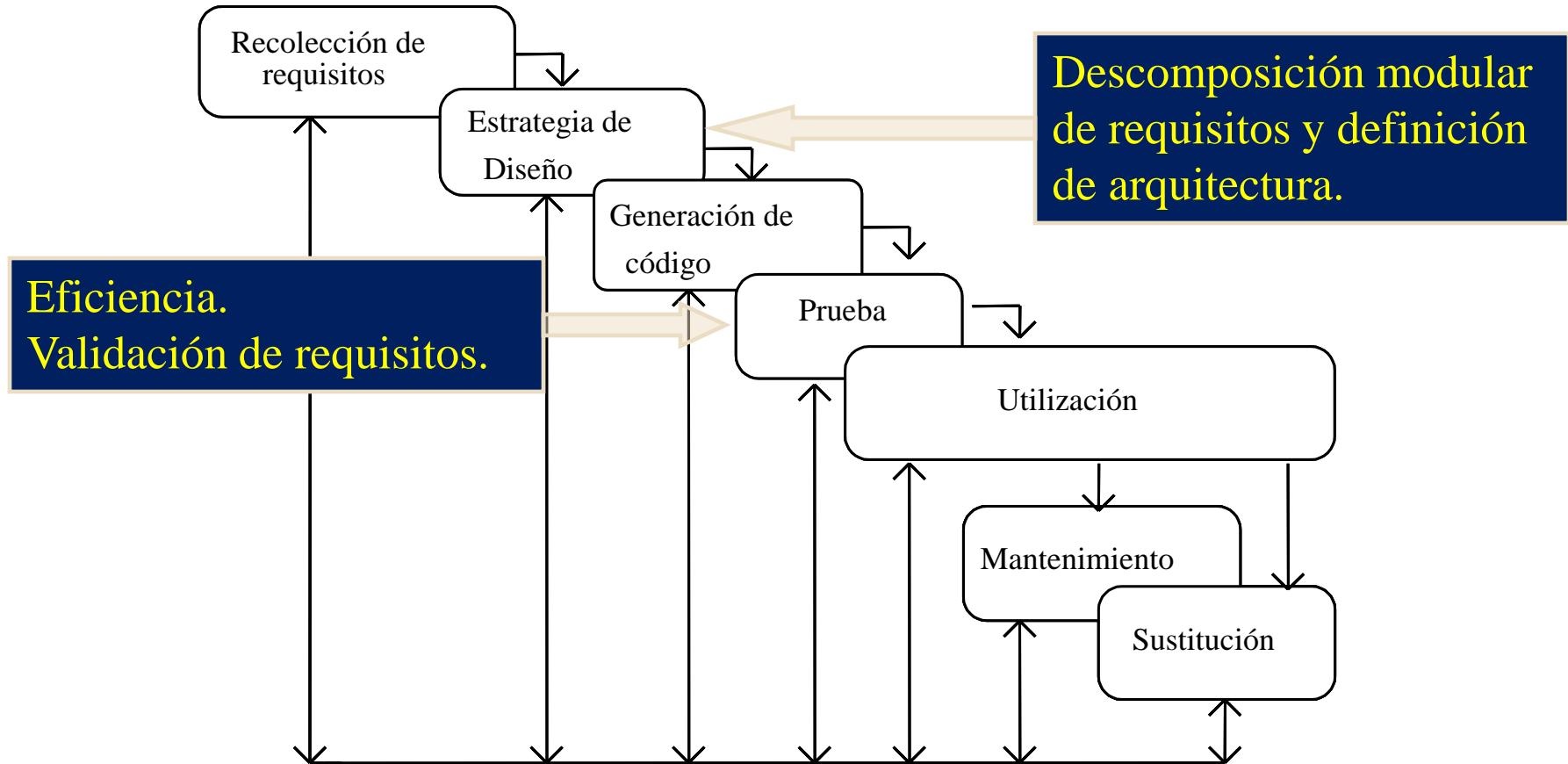
Técnicas de 4^a GENERACIÓN

- 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.
 - Manipulación de datos.
 - Capacidad de hojas de cálculo.
 - Capacidades gráficas de alto nivel.
 - Generación de código.

Técnicas de 4^a generación



Técnicas de 4^a generación



Técnicas de 4^a generación

□ Consideraciones:

- No se han obtenido los resultados esperados.
- Sin embargo consiguen reducir tiempo.
- Se utilizan sobre todo en software de gestión.

□ Críticas:

- No son más fáciles de usar.
- El código que generan es ineficiente.
- 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

Determinación:

- objetivos
- alternativas
- restricciones

Inicio

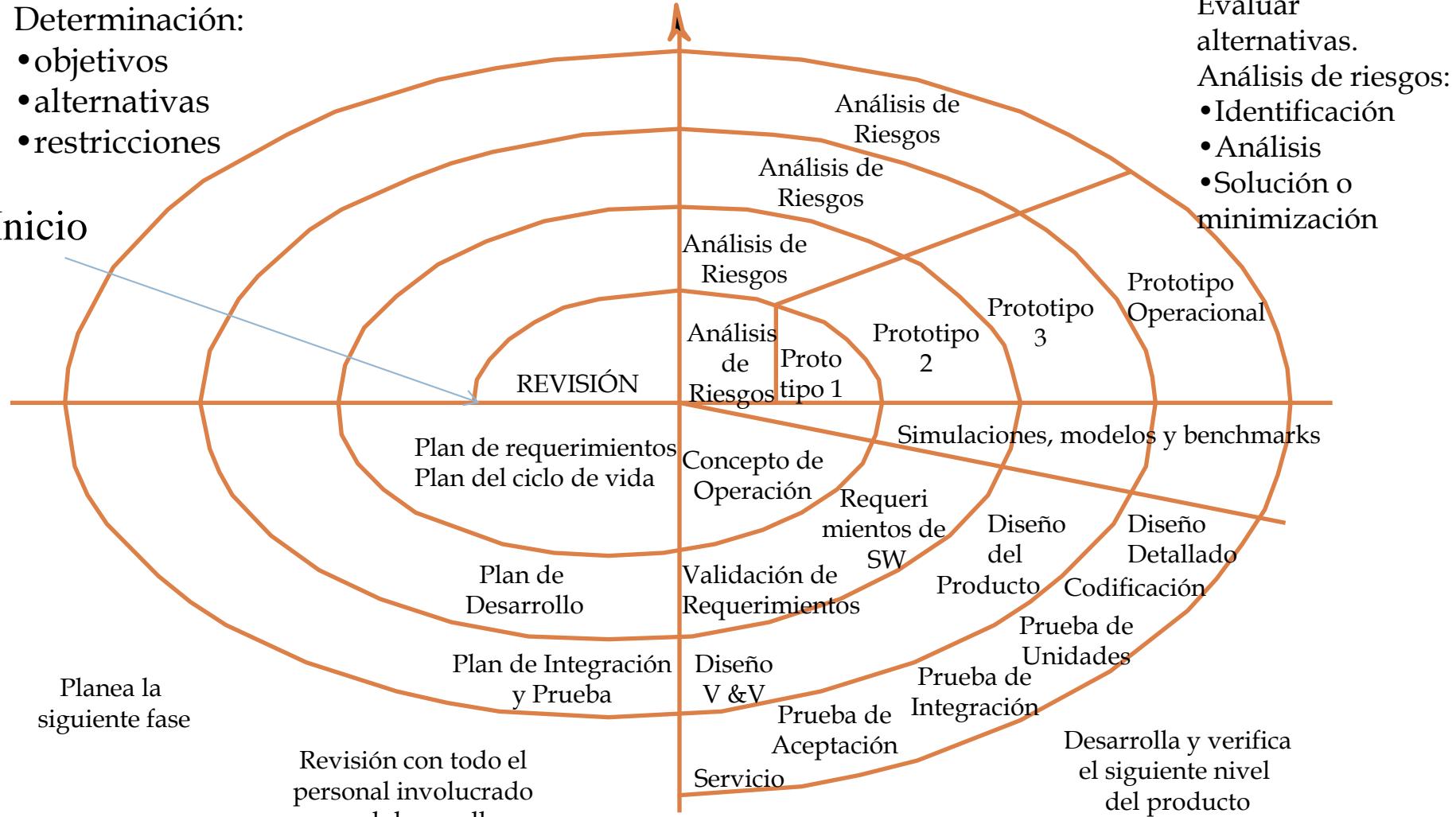
Planea la siguiente fase

Revisión con todo el personal involucrado en el desarrollo

Evaluar alternativas.
Análisis de riesgos:

- Identificación
- Análisis
- Solución o minimización

Ingeniería del Software



Modelo en Espiral

- Se divide en actividades estructurales o regiones neutras
 - Definición de objetivos
 - Objetivos específicos, Alternativas y Restricciones.
 - 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.

Ejemplo de Modelo en Espiral

Elaboración de un catálogo.

□ Objetivos

- Desarrollar un sistema amigable
- Sistema integrado
- Ayudar a todo el personal del proyecto
- Servir en todos los ciclos de vida

□ Restricciones

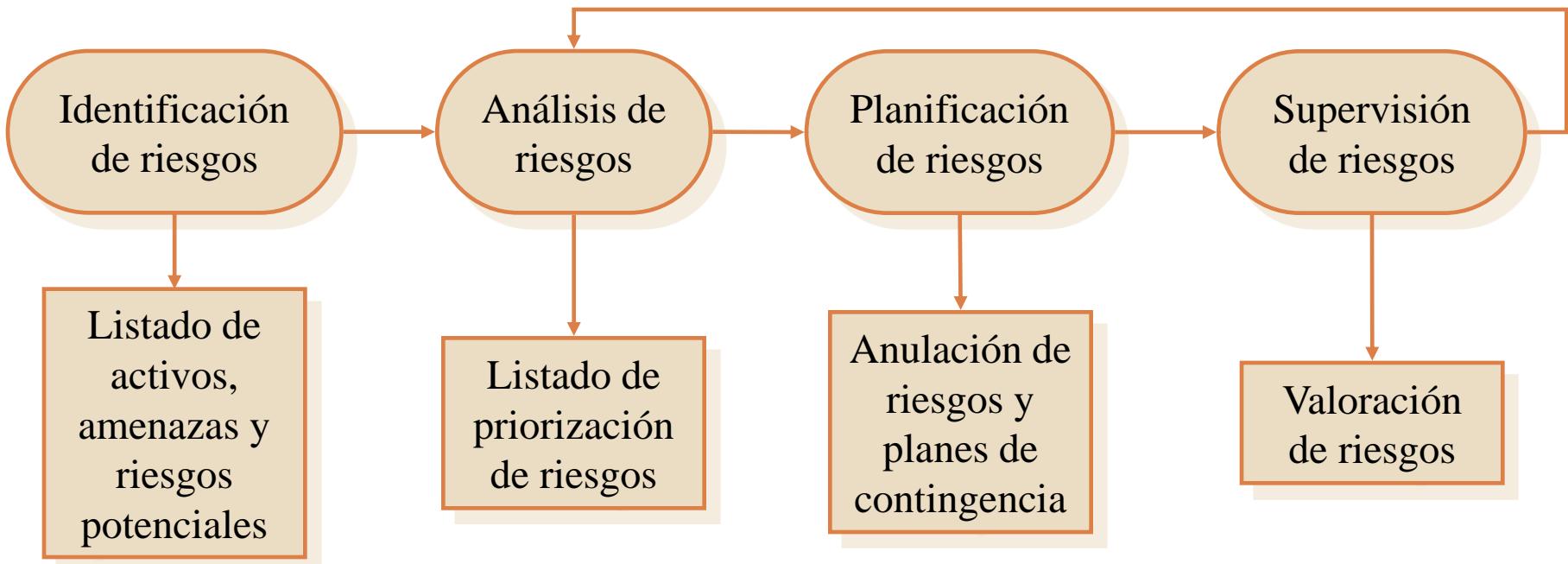
- Servicio confiable
- Portabilidad estable
-

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

Modelo en Espiral → Análisis de riesgos

□ Administración del riesgo:



Análisis de riesgos

DEFINICIONES (Magerit):

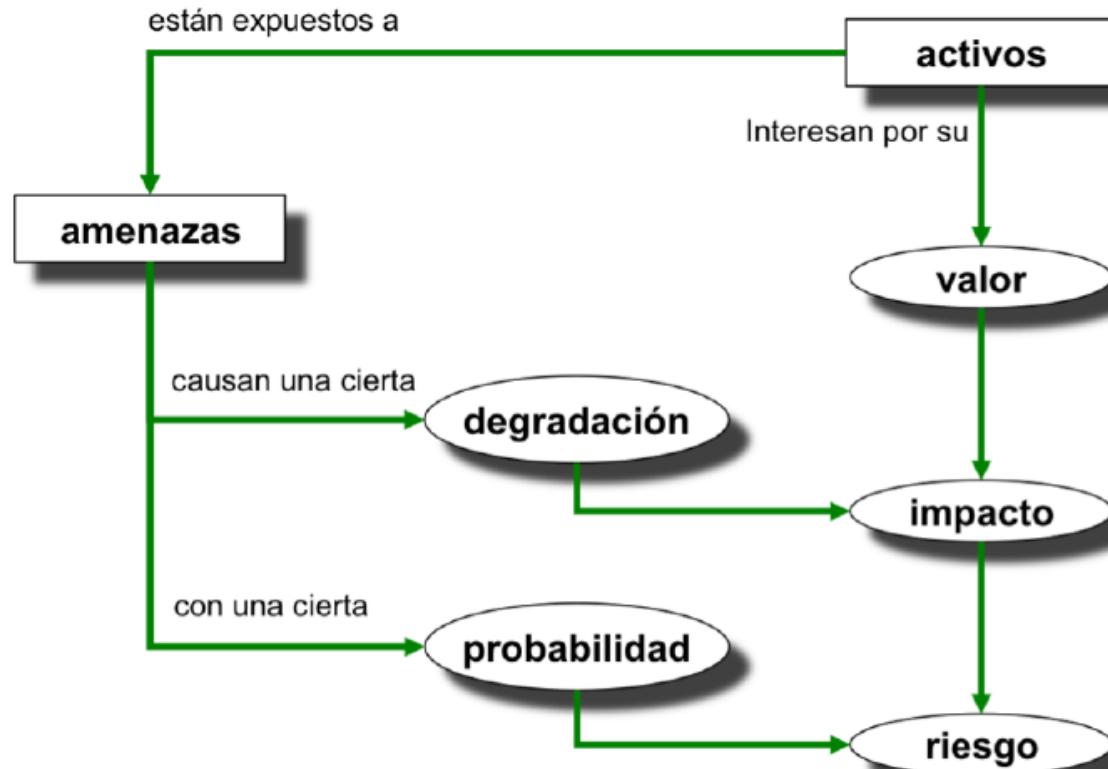
- **Activos**, son los elementos del sistema de información que soportan la misión de la Organización
- **Amenazas**, que son cosas que les pueden pasar a los activos causando un perjuicio a la Organización
- **Salvaguardas** (o contramedidas), que son medidas de protección desplegadas para que aquellas 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



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

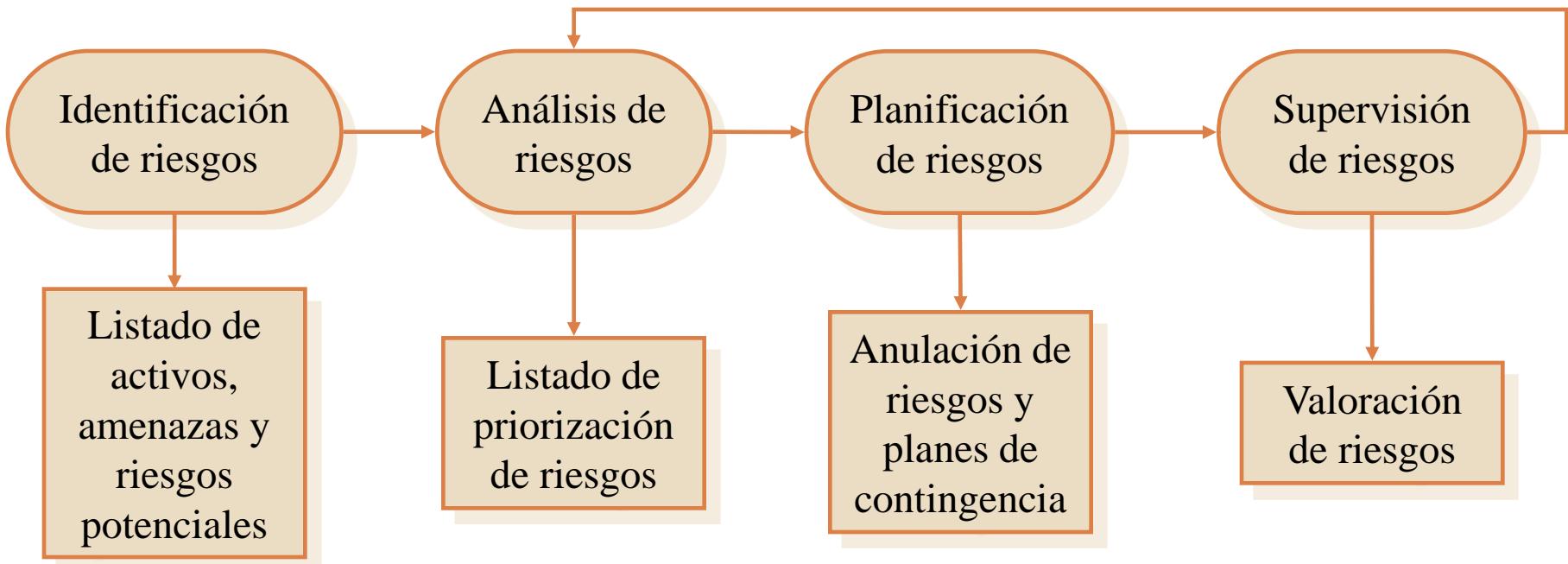
- Otras dimensiones :
 - la **autenticidad**: ¿qué perjuicio causaría no saber exactamente quien hace o ha hecho cada cosa?
 - la **trazabilidad** del uso del servicio: ¿qué daño causaría no saber a quién se le presta tal servicio? Es decir, ¿quién hace qué y cuándo?
 - la **trazabilidad** del acceso a los datos: ¿qué daño causaría no saber quién accede a qué datos y qué hace con ellos?

Análisis de riesgos

- 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 la organización
 - 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 2005, Technical Reports SEI*
 - Del proyecto afectan a la calendarización o recursos.
 - Del producto afectan a la calidad o desempeño del software
 - Del negocio afectan a la organización responsable del desarrollo
 - Análisis de riesgos
 - Planificación del riesgo
 - Gestión de riesgos

Análisis de riesgos

- Administración del riesgo:
 - ▣ Identificar los riesgos.
 - ▣ Análisis de riesgos
 - Para cada riesgo, evaluamos la probabilidad de que ocurra (baja, moderada, alta...) y sus consecuencias (impacto o degradación) (Catastrófico, serio, tolerable...)
 - Definición de escalas.
 - ¿Los atendemos a todos.?
 - ▣ Planificación del riesgo
 - Estrategias evitar y aceptar
 - Estrategias para la administración de riesgos: Estrategias de prevención, Estrategias de minimización, transferencia y planes de contingencia
 - ▣ Gestión 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

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.

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:

■ Controlar el cambio.

- Limitar los cambios permitidos.
- Limitar el momento del cambio.

□ Ligeras

■ Acelerar al máximo las entregas de software.

■ Hacer participar 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
 2. Organiza el proceso
 3. Organiza el plan de trabajo

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 refleja la forma en que se puede volver más efectivo; entonces su comportamiento se ajusta y adecua en concordancia.

Desarrollo ágil

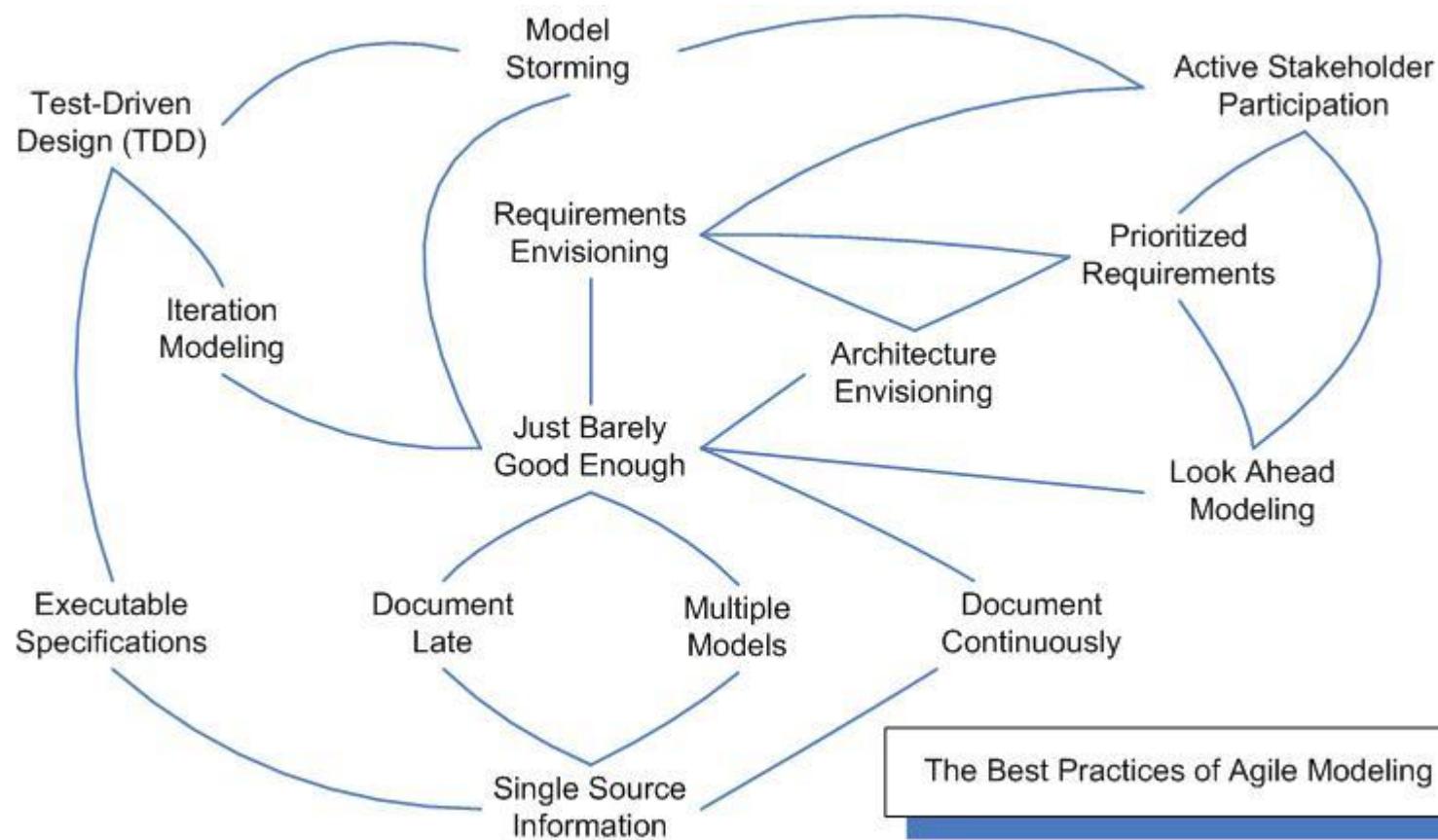
- Modelos de desarrollo ágil
 - Desarrollo Adaptativo de Software (DAS)
 - Método de desarrollo de sistemas dinámicos (MDSD)
 - Melé (Scrum)
 - Cristal
 - Desarrollo conducido por características (DCC)
 - **Modelado Ágil (MA)**
 - **Programación Extrema (PE)**

Modelado Ágil (MA)

- Agile Modeling (AM) is a practice-based methodology for effective modeling and documentation of Software Programs
- At a high level AM is a collection of best practices, depicted in the pattern language map next slide
- At a more detailed level AM is a collection of values, principles, and practices for modeling software.

<http://www.agilemodeling.com/>

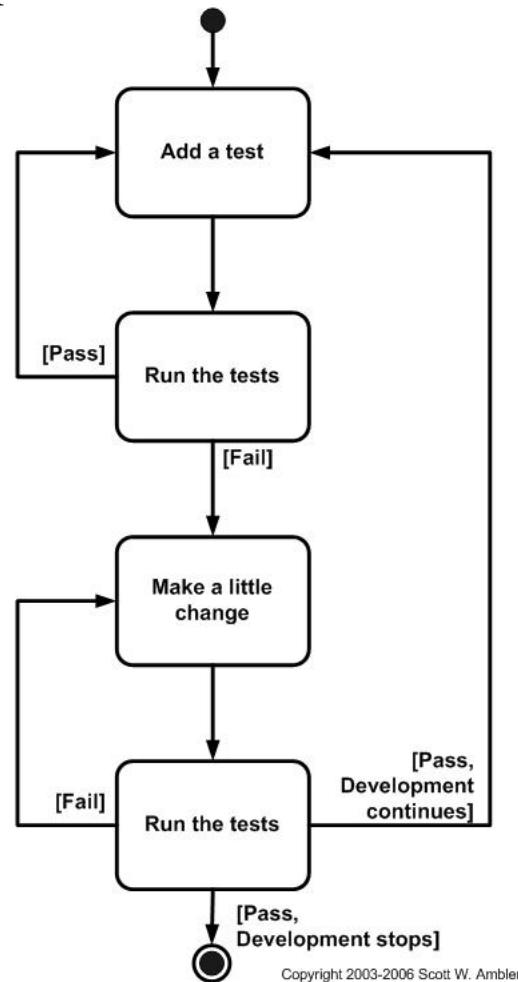
Modelado Ágil (MA)



Copyright 2005-2011 Scott W. Ambler

Modelado Ágil (MA)

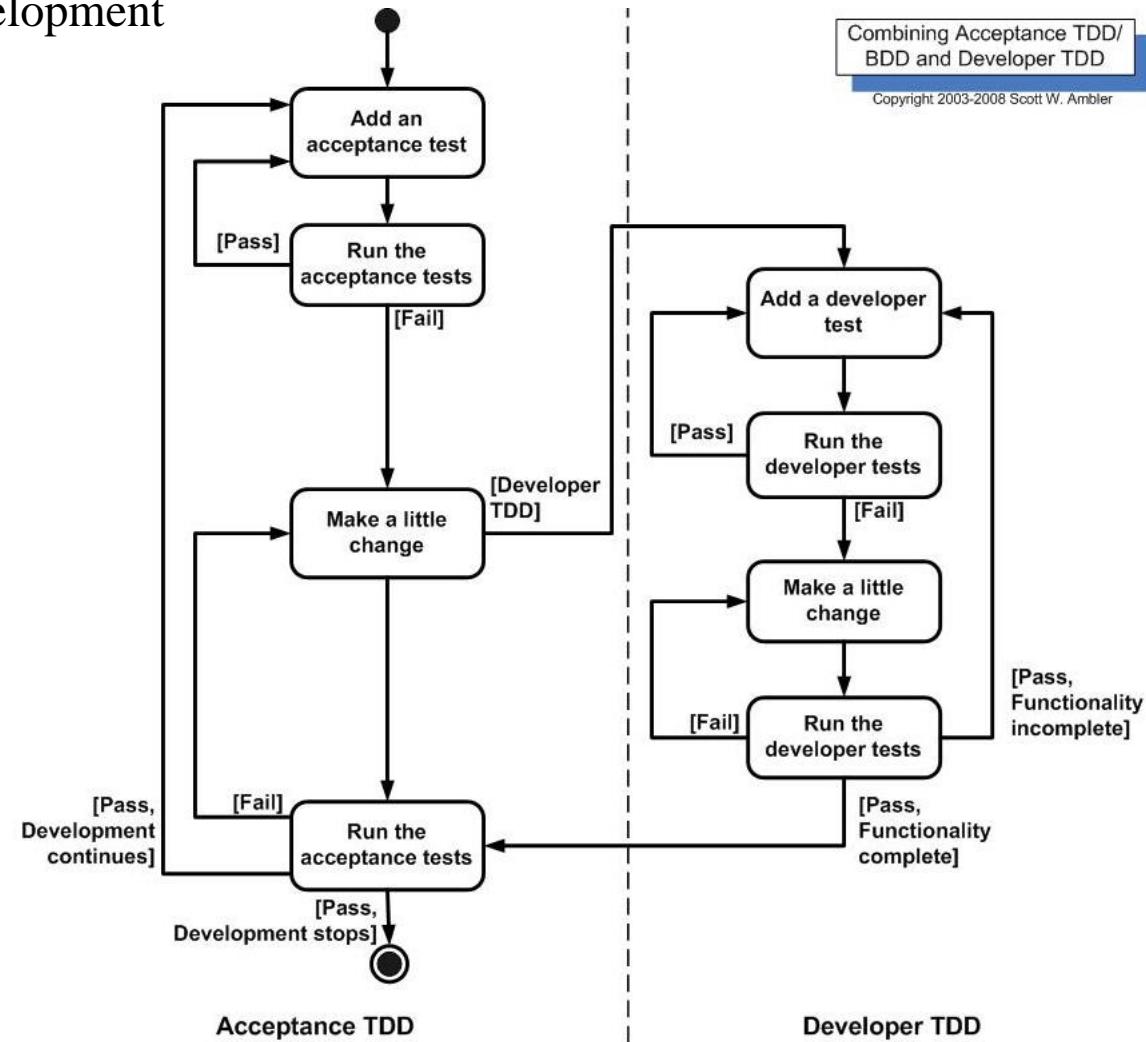
TFD: Test First Development



TDD: Refactorización + TFD

Modelado Ágil (MA)

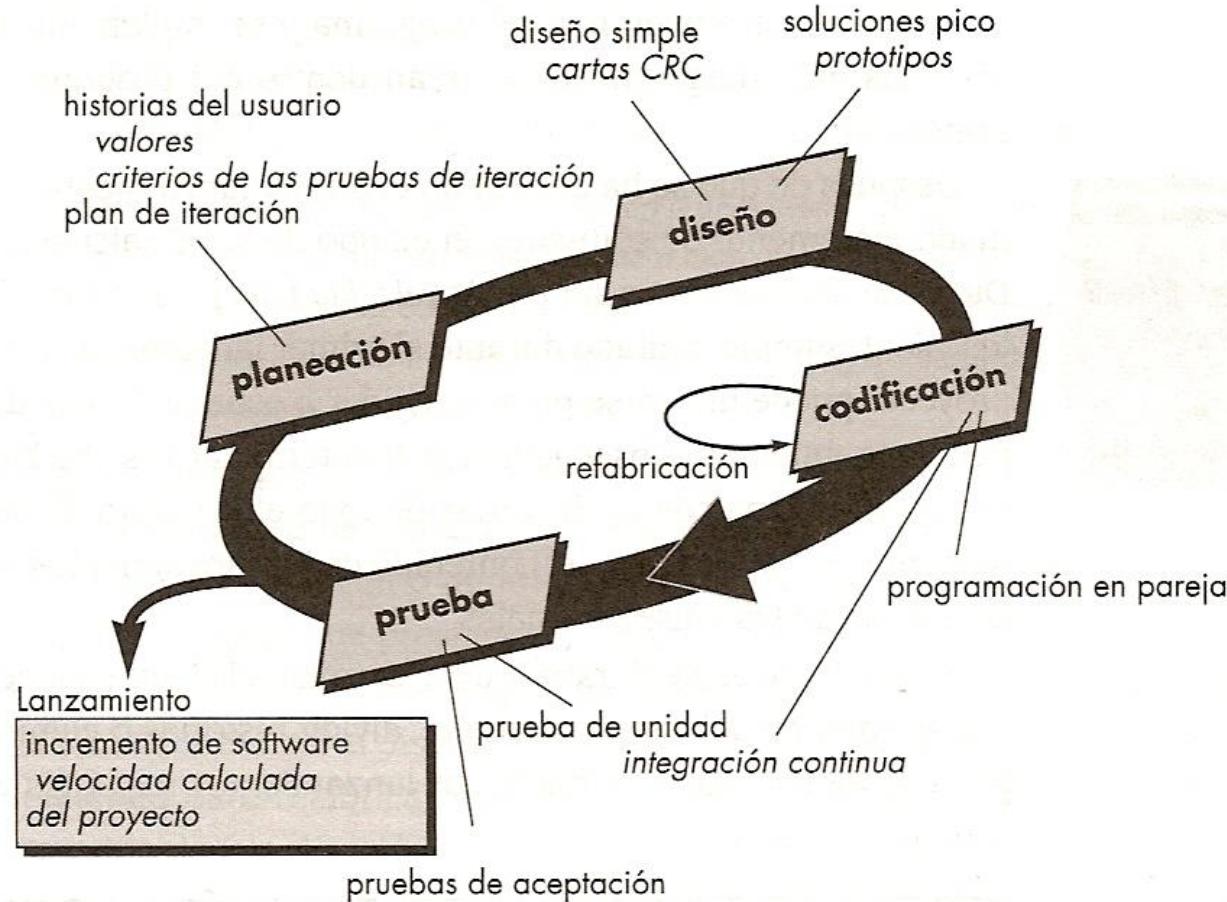
TDD: Test-Driven Development



Modelado Ágil (MA)

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

Programación Extrema (PE)



Programación Extrema (PE): Planificación

Historia de Usuario

Número: 1	ID: 5
Nombre historia:	Cómo un libro, presta
Prioridad en neg:	Alta
Puntos estimados:	Estimacion
Programadorres:	Prioridad
Descripción:	Quiero cambiar la
Validación:	* Intro que se * Intro compr * Intro compr * Intro númer númer
<input type="checkbox"/> P	<input type="checkbox"/> Se estima

Historias de usuario

Como [cliente habitual], **quiero** [ver productos relacionados] **para** [ver si hay otros productos que me puedan interesar]

Condiciones de completitud

- Los productos estarán ordenados por valoración y margen de beneficio.
- Cuando el usuario haga clic en un producto, se desplegará el detalle.
- Etc.

Prioridad
70

Coste
5

http://farm1.static.flickr.com/55/1478/4576_8a453070f3.jpg

www.agile-spain.com

viernes 18 de enero de 13

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 y acuerdan las fechas de entrega del incremento. (Cuántas, cuáles y con qué fechas).
 - 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 implementaran 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

- 1^{er} principio. MANTENERLO SIMPLE
- Refabricación del código.
 - Puede aumentar radicalmente con el tamaño de la aplicación.
- 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.

□ Productos

- Tarjetas CRC (Clase-Responsabilidad- Colaborador)
- Soluciones de Pico (prototipos para problemas de difícil diseño).

Programación Extrema (PE)

Diseño

Refabricación 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 refabricar, 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)

Prueba

- Pruebas diarias. Refabricación.
 - ¿Cómo?
 - Organizamos las pruebas individuales en un conjunto universal de pruebas
 - Pruebas automáticas (junit)
 - ¿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.

Glosario

- El modelo en espiral
- Definiciones
- Activos
- Amenazas
- Plan de riesgos

Modelo en Espiral

Determinación:

- objetivos
- alternativas
- restricciones

Inicio

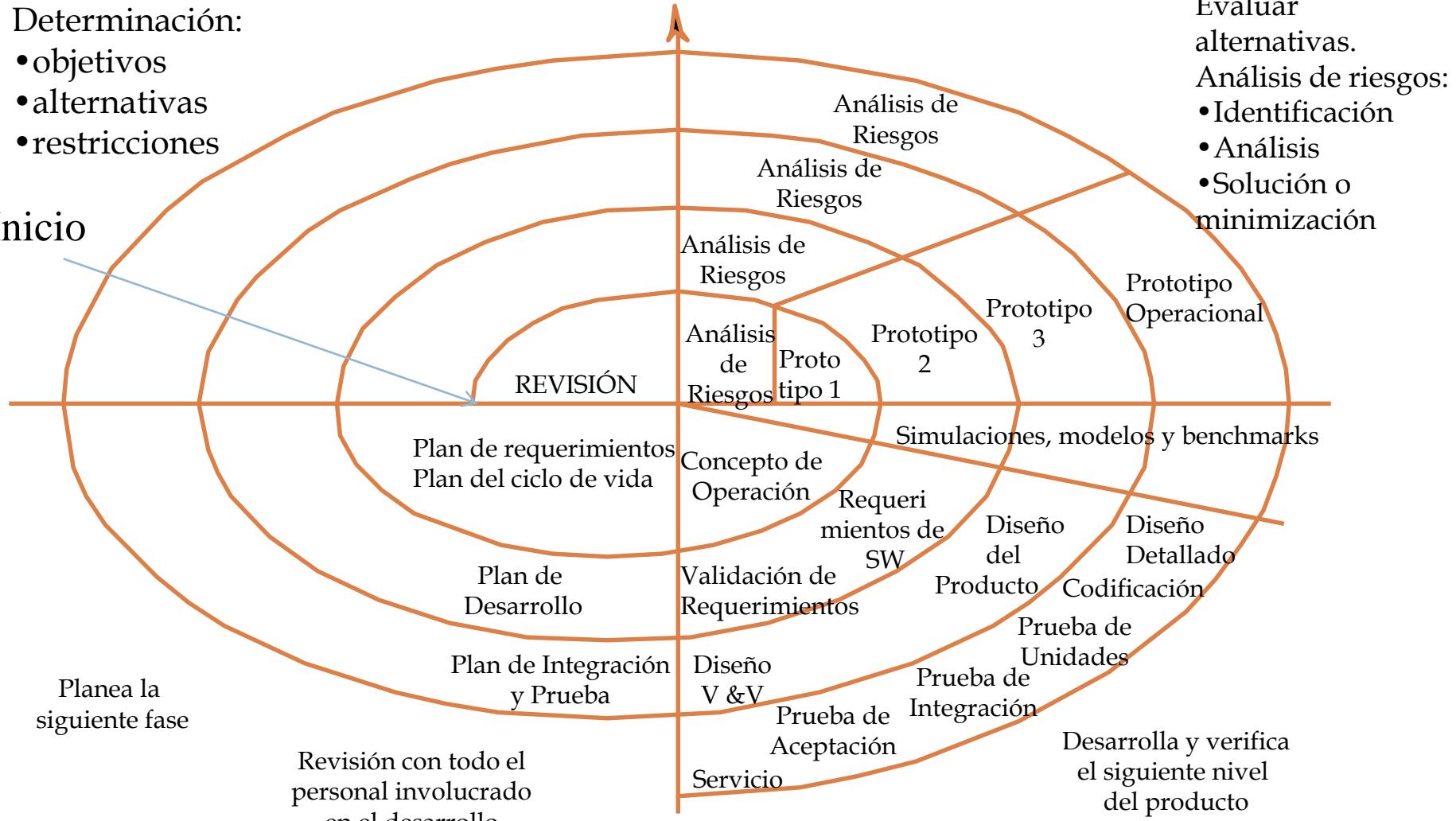
Planea la siguiente fase

Revisión con todo el personal involucrado en el desarrollo

Evaluar alternativas.
Análisis de riesgos:

- Identificación
- Análisis
- Solución o minimización

Ingeniería del Software



Análisis de riesgos: Definiciones

DEFINICIONES:

- **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

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: Definiciones



Análisis de riesgos: Activos

□ ACTIVOS

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

Análisis de riesgos: Activos

- 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 y prestar servicios
 - **Las instalaciones** que acogen equipos informáticos y de comunicaciones.
 - **Las personas** que explotan u operan todos los elementos anteriormente citados.

Análisis de riesgos: Activos

- Dimensiones de la información:
 - 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: Activos

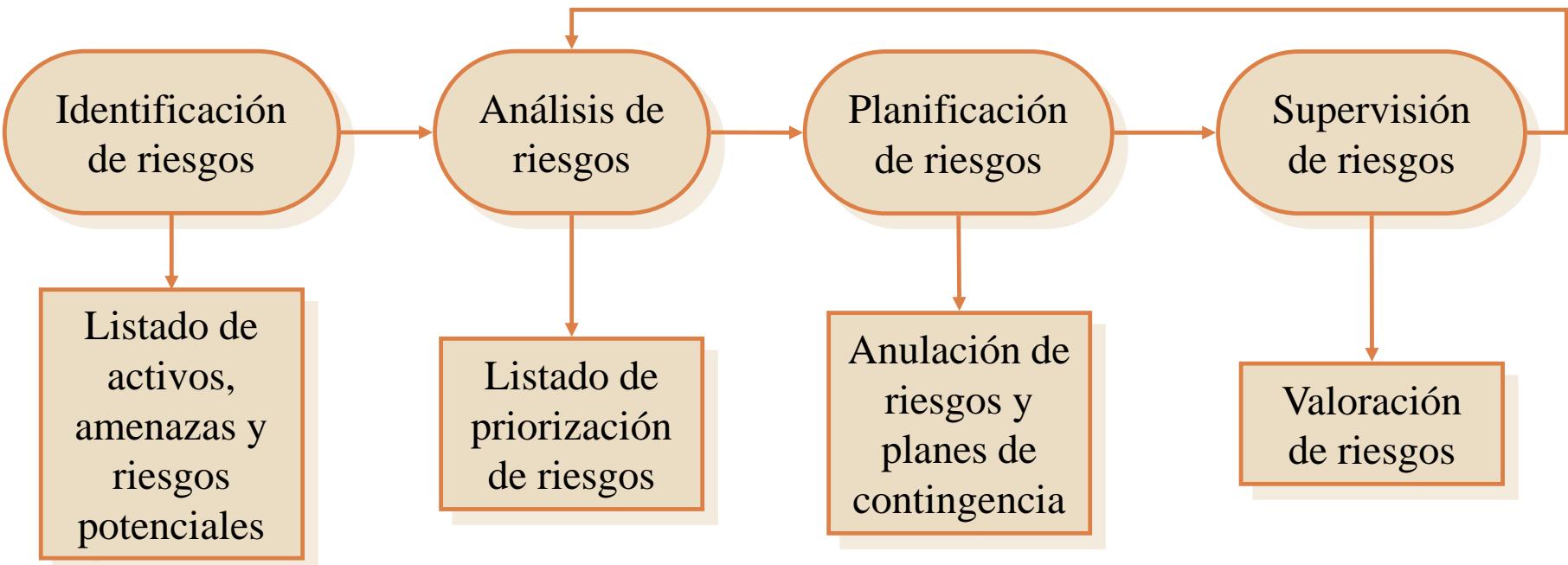
- Dimensiones de los servicios:
 - la **autenticidad**: ¿qué perjuicio causaría no saber exactamente quién hace o ha hecho cada cosa?
 - la **trazabilidad** del uso del servicio: ¿qué daño causaría no saber a quién se le presta tal servicio? Es decir, ¿quién hace qué y cuándo?
 - la **trazabilidad** del acceso a los datos: ¿qué daño causaría no saber quién accede a qué datos y qué hace con ellos?

Análisis de riesgos: Amenazas

- **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: Plan de riesgos

□ Administración del riesgo:



Análisis de riesgos: Plan de riesgos

Administración del riesgo:

Identificar los riesgos.

- Determinar los activos relevantes para la organización
- Determinar a que amenazas están expuestos dichos activos
- Definir el riesgo como la probabilidad de un activo esté expuesto a una amenaza.
- Clasificación en taxonomías. Sommerville, Technical Reports SEI
 - Del negocio: afectan a la organización responsable del desarrollo
 - Del proyecto: afectan a la calendarización o recursos.
 - Del producto: afectan a la calidad o desempeño del software
- Tecnológicos
- Personales
- De la organización
- En las herramientas
- En los requisitos
- En la estimación

- Análisis de riesgos
- Planificación del riesgo
- Supervisión 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
- Su Impacto, consecuencias, o degradación: Catastrófico, Grave, Tolerable, Insignificante.

Matriz de riesgos

- Planificación
- Supervisión

	TOLERABLE	SERIO	CRÍTICO
BAJA		R4, R19, R26, R32, R33	R5, R8, R13, R34
MODERADA	R6, R15,R16, R18, R28, R29,R30,	R2, R3, R12, R17, R21, R23, R25, R27,R31	R1, R7, R9, R11
ALTA	R24, R10	R14, R20, R22	

Análisis de riesgos: Plan de riesgos

- Administración del riesgo:
 - **Identificar los riesgos.**
 - **Planificación del riesgo**
 - **Prevención, o evitación:** reducimos la probabilidad de aparición
 - **Minimización:** reducimos el impacto de la amenaza
 - **Planes de contingencia:** tratamos de llevar a nuestra organización al estado anterior a la ocurrencia del riesgo, siguiendo un plan preconcebido.
 - **Transferencia:** subcontratamos expertos que nos gestionen (y se responsabilicen) de la gestión de un riesgo (porque son expertos en ello)
 - **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 laboral	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.			

GLOSARIO

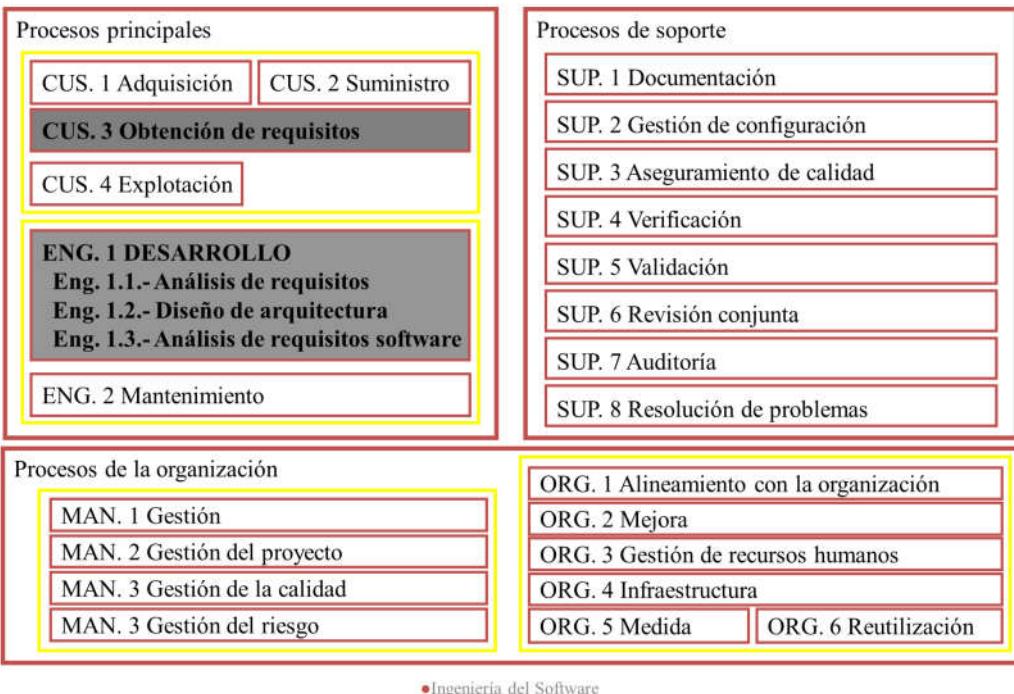
4.- Ingeniería de requisitos

- 4.0.- Introducción.
- 4.1.- Primeras cuestiones.
- 4.2.- Obtención de requisitos
- 4.3.- Análisis de requisitos
 - 4.3.1.- Del Sistema
 - 4.3.2.- Del Software
- 4.4.- Especificación.
- 4.5.- Validación de requisitos.
- 4.6.- Administración de requisitos.

•Ingeniería del Software

Tema 4.- *Análisis de requisitos*

ISO/IEC TR 15504-2



•Ingeniería del Software

Introducción

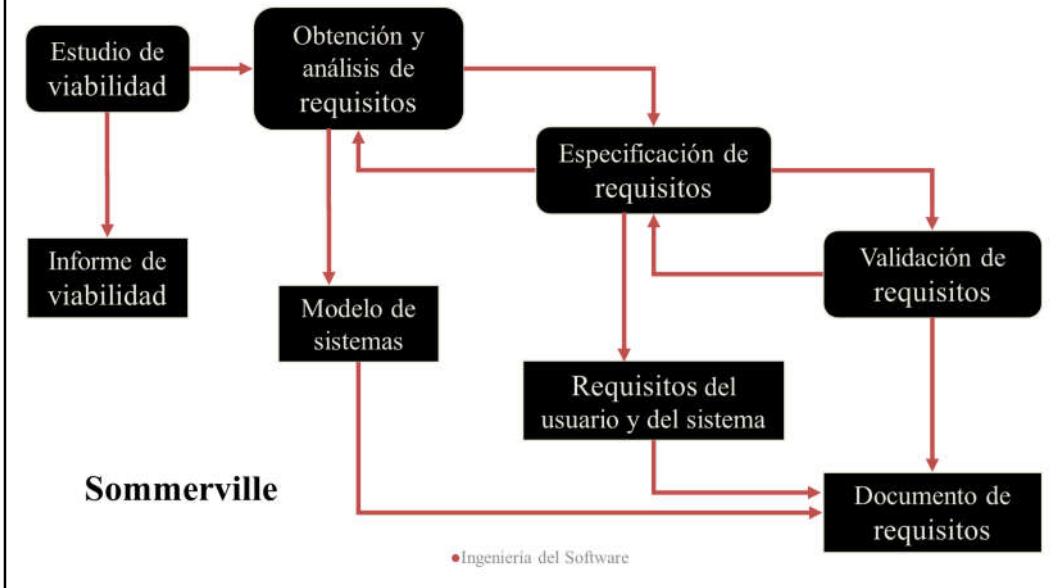
- Todos los modelos de ciclo de vida introducen una etapa de Análisis de Requisitos.
- Requisito. Definición:
 - Una condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado
 - Por extensión las condiciones o capacidades que debe cumplir o poseer un sistema o uno de sus componentes para satisfacer un contrato, una norma o una especificación.
- Análisis de requisitos IEEE Std. 610. Definición:
 - El proceso del estudio de las necesidades de los usuarios para llegar a una definición de los requisitos del sistema, de hardware o de software, así como, el proceso de refinamiento de dichos requisitos.

•Ingeniería del Software

<https://segoldmine.ppi-int.com/content/standard-ieee-std-610-ieee-standard-glossary-software-engineering-terminology>

IEEE Std 610.12 is a revision and redesignation of IEEE Std 729. This standard contains definitions for more than 1000 terms, establishing the basic vocabulary of software engineering. Building on a foundation of American National Standards Institute (ANSI) and International Organization for Standardization (ISO) terms, it promotes clarity and consistency in the vocabulary of software engineering and associated fields.

Introducción



El estudio de viabilidad se recoge en el ciclo de vida en espiral.

Puede estar muy relacionado con el desarrollo que prototipos

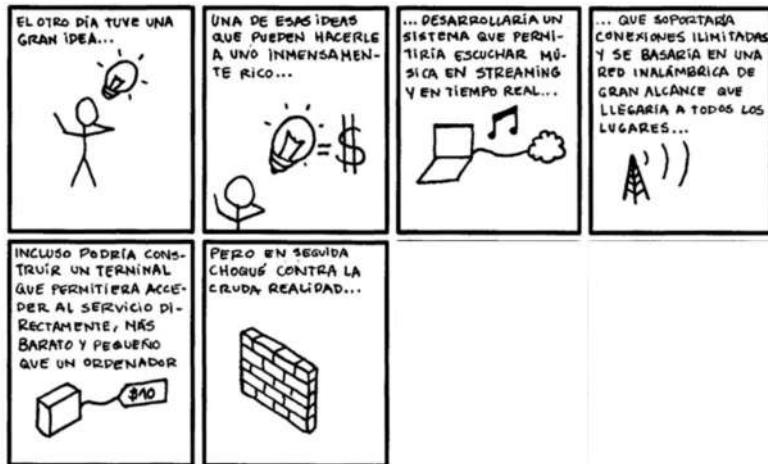
Introducción

1. Identificar las fuentes
 - ¿quién está involucrado en el desarrollo del sistema?
2. Realizar las preguntas adecuadas
 - Técnicas de recogida de información
3. Analizar la información
 - Técnicas de análisis de requisitos
4. Confirmar con los usuarios
 - Validación/Verificación de requisitos
5. Sintetizar los requisitos
 - Especificar formalmente

Raghavan

•Ingeniería del Software

Tema 4.- Análisis de requisitos



•Ingeniería del Software

Introducción

- Análisis y Obtención de requisitos
 - Determinación de requisitos, determinación del problema, estudio del problema, definición de requisitos
- Síntesis. Especificación de requisitos:
 - Describe el problema analizado
 - Muestra la estructura de la solución propuesta

Características comunes de los métodos de análisis

- Realizan una **abstracción de las características del sistema**, es decir, consisten en **desarrollar un modelo** del mismo.
- Representan el **sistema de forma jerárquica**, basándose en mecanismos de partición del problema y estableciendo varios **niveles de detalle**.
- Definen cuidadosamente las **interfaces del sistema**, tanto las interfaces externas, que relacionan el sistema con su entorno, como de las internas, las que se establecen entre los distintos módulos definidos.
- Sirven de **base para las etapas posteriores** de diseño y de implementación.

Características comunes de los métodos de análisis

- Deben proporcionar **distintos niveles de especificación** del sistema. (Requisitos de usuario, esenciales, de sistema, de implementación.)
- No prestan demasiada atención a la representación de las restricciones o de criterios de validación (exceptuando los métodos formales).

Requisitos: Clasificación.

Tema 6 Sommerville

- **Funcionales:** Son declaraciones de los servicios o funcionalidades que implementará el sistema.
- **No funcionales:** Son restricciones sobre los servicios (de tiempo, ajuste a estándares,...)
- **De dominio:** Son los requisitos que provienen del dominio (entorno) en el que se realiza la aplicación y pueden ser funcionales o no funcionales.

•Ingeniería del Software

Realmente no son conjuntos disjuntos. Un requisito No Funcional de ajuste a estándares en una interfaz, por ejemplo usar DICOM o HL7 en medicina, , puede ser perfectamente un requisito de dominio.

GLOSARIO

4.- Ingeniería de requisitos

4.0.- Introducción.

4.1.- Primeras cuestiones.

4.2.- Obtención de requisitos

4.3.- Análisis de requisitos

 4.3.1.- Del Sistema

 4.3.2.- Del Software

4.4.- Especificación.

4.5.- Validación de requisitos.

4.6.- Administración de requisitos.

•Ingeniería del Software

Requisitos: Funcionales

- **Funcionales:**

- Ejemplos

- El usuario deberá tener la posibilidad de buscar en el conjunto de toda la base de datos o seleccionar un subconjunto de ella.
 - El sistema deberá proveer visores adecuados para que el usuario lea documentos en el almacén de documentos.

- Consecuencias de incumplimiento

- Degrada el sistema

Requisitos No Funcionales

- **No funcionales:**

- Ejemplos

- El sistema deberá resolver 300 transacciones por segundo.
 - El tiempo máximo de respuesta del *airbag* ha de ser de 30 milisegundos

- Consecuencias de incumplimiento

- Inutilizan el sistema

- Deberían expresarse de forma cuantitativa

Requisitos No Funcionales

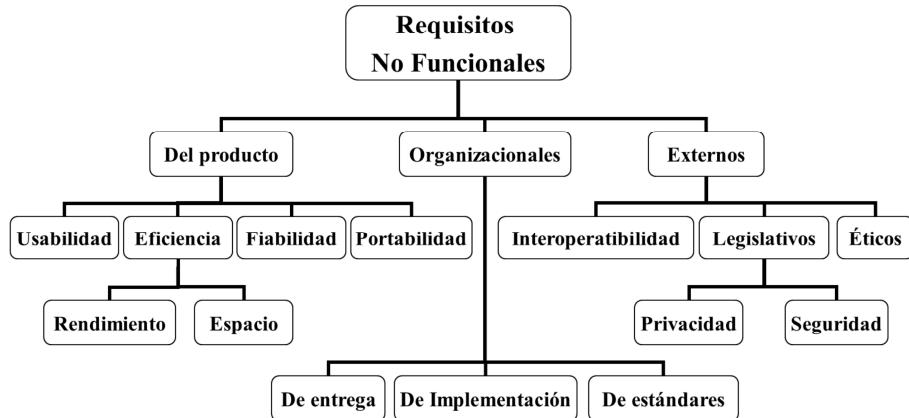
– Deberían expresarse de forma cuantitativa

Propiedad	Medida
Rapidez	Nº transacciones, tiempo de respuesta, tiempo de actualización
Tamaño	Kbytes, Nº de chips de RAM
Facilidad de uso	Tiempo de formación, Nº de cuadros de ayuda
Fiabilidad	Tiempo medio entre fallos, Tasa de ocurrencia de fallos, probabilidad de disponibilidad
Robustez	Tiempo de reinicio después de fallo, % de eventos que provocan fallos
Portabilidad	Nº de sistemas objetivo, % de declaraciones dependientes

•Ingeniería del Software

Requisitos No Funcionales

- Taxonomía



•Ingeniería del Software

Requisitos de Dominio

- De dominio:

- Ejemplos

- Deberá existir una Interface de usuario estándar para acceder a todas las bases de datos que tomen como referencia el estándar Z39.50. (En un dominio de biblioteca.)
 - La desaceleración del tren con alarma de luz roja se calculará como: $D_{min}=D_{control} + D_{gradiente}$; donde la $D_{gradiente}$ es conocida y distinta según los tipos de trenes (Dominio de control ferroviario)

- Consecuencias de incumplimiento

- Es imposible que el sistema trabaje satisfactoriamente

•Ingeniería del Software

Primeras cuestiones

- ¿Cuánto tiempo debe dedicarse al análisis del sistema?
 - Regla 40-20-40.
 - *"Es tu peor pesadilla. Un cliente entra en tu oficina, se sienta, te mira a los ojos y dice: Yo sé que usted piensa que entiende lo que digo, pero lo que usted no entiende es que lo que digo no es realmente lo que quiero decir. Esto sucede de manera invariable cuando el proyecto está avanzado, después de que se han realizado los compromisos relativos al tiempo de entrega. Las reputaciones están en juego y el dinero en serio peligro"*

•Ingeniería del Software

Pressman 6, pp 155

Primeras cuestiones

- ¿Quién está involucrado?
 - El analista de sistemas. (Comprometido)
 - Este debe ser una persona con buena formación técnica y con experiencia. Ha de estar en contacto con las demás partes.
 - Los Stakeholders. (Involucrados o comprometidos)
 - Cualquier persona que tiene influencia directa o indirecta sobre los requisitos del sistema.

•Ingeniería del Software

Involucrado, afectado, implicado. En una tortilla con chorizo la gallina se ve involucrada, el cerdo se ve comprometido.

Personal: El analista.

- Comunicación entre Stakeholders y desarrolladores de software.
 - los técnicos no conocen los detalles del trabajo de la empresa para la cual se va a desarrollar.
 - Los stakeholders no saben que información es necesaria para el desarrollo de una aplicación.
- Es el puente entre los desarrolladores y los stakeholders. Está presente en todas las reuniones y revisiones que se hagan a lo largo del desarrollo.
- Características de un buen analista:
 - Conocimiento de Ingeniería del Software
 - Conocimientos técnicos y experiencia en soluciones informáticas.
 - Facilidad de comunicación
 - Conocimientos sobre el campo de aplicación
 - Clarividencia de ideas

•Ingeniería del Software

Personal involucrado: Los Stakeholders.

- Usuarios finales que interactúan con el sistema, en la medida de que influyen en los requisitos
- Ingenieros que desarrollan, prueban y/o mantienen sistemas relacionados
- Administradores de negocio
- Expertos en el dominio del sistema

Primeras cuestiones

•¿Por qué es una tarea tan difícil?

- Porque consiste en la traducción de unas ideas vagas de necesidades de software en un conjunto concreto de funciones y restricciones.
- Porque el lenguaje natural puede ser ambiguo
 - Sé que cree que entendió lo que piensa que dije, pero no estoy seguro de que se dé cuenta de que lo que escuchó no es lo que yo quise decir.

Anónimo

Primeras cuestiones

- ¿Por qué es una tarea tan difícil?
 - Los stakeholders no saben lo que quieren del sistema.
 - Los stakeholders expresan los requisitos con sus propios términos. (Requisitos de dominio).
 - Los stakeholders tienen requisitos distintos que pueden entrar en conflicto
 - Puede haber condicionantes políticos y de la organización.
 - Los requisitos cambian durante el análisis.
 - <https://www.youtube.com/watch?v=BKorP55Aqvg>

•Ingeniería del Software

Junta a tres clientes en una sala para que te den una idea del sistema a construir y tendrás 4.

<https://www.youtube.com/watch?v=BKorP55Aqvg>

Video con subtítulos en castellano de una parodia sobre una reunión de requisitos

“The Chaos Report” (Standish Group)

Fracaso en proyectos de informática

- Según este estudio del total de proyectos evaluados:
 - El 39% son completados con el alcance esperado, en el tiempo planificado y dentro del presupuesto asignado.
 - El 43% de los proyectos son completados con menor alcance, y/o sobrecosto y/o fuera de término.
 - El 18% de los proyectos son cancelados antes de terminar.
- Del total de proyectos que se completan:
 - El 70% de los proyectos terminan fuera de plazo.
 - El 54% de los proyectos sufren sobrecostos.
 - El 66% de los proyectos no son considerados exitosos.

Principales factores de fracaso

1. Requerimientos incompletos **13.1 %**
2. Falta de involucramiento del usuario **12.4 %**
3. Falta de recursos **10.6 %**
4. Expectativas no realistas **9.9 %**
5. Falta de apoyo de la Gerencia **8.7 %**
6. Requerimientos cambiantes **8.1**

TOTAL: 21.2 %

•Ingeniería del Software

Realmente la comprensión de los requisitos supone un problema

Chaos Manifesto 2013.pdf para los datos del primer punto.

Los otros son de años anteriores y no se recogen en el de 2013, por lo que no pueden ser actualizados.

Desde el punto de vista conceptual siguen siendo válidos.

GLOSARIO

4.- Ingeniería de requisitos

4.0.- Introducción.

4.1.- Primeras cuestiones.

4.2.- Obtención de requisitos

4.3.- Análisis de requisitos

 4.3.1.- Del Sistema

 4.3.2.- Del Software

4.4.- Especificación.

4.5.- Validación de requisitos.

4.6.- Administración de requisitos.

•Ingeniería del Software

Técnicas de recogida de información

Tema 6. Piattini, 2003

- Entrevistas
- Cuestionarios
- Prototipado
- Observación, Estudio de documentación
- Escenarios/Casos de uso
- Tormenta de ideas (Brainstorming)
- Desarrollo conjunto de aplicaciones (JAD: Joint Application Design)
 - Técnicas para facilitar las especificaciones de la aplicación (TFEA)

•Ingeniería del Software

Piattini 2003, pp 182

Técnicas de recogida de información TFEA

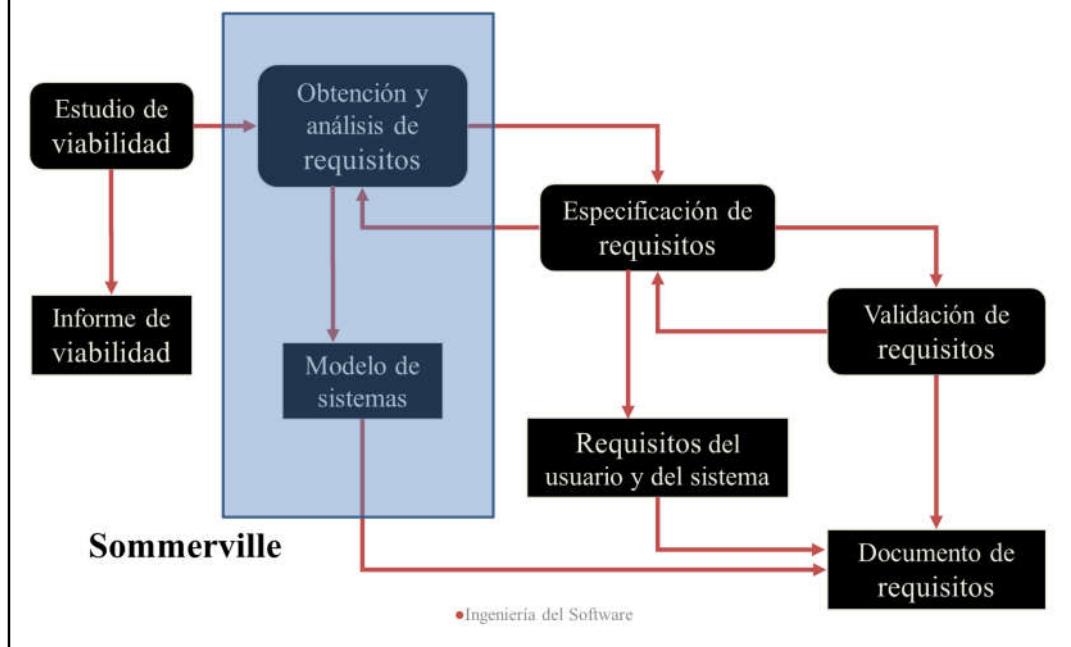
- Redacción de un documento inicial: (analista y cliente)
 - Descripción del problema.
 - Objetivos.
 - Propuesta de solución (si es posible).
(Grupo TFEA: analista, cliente, moderador, ...)
- Creación de listas individuales de:
 - Objetos: del sistema y del entorno
 - Operaciones: funcionalidad, procesos, relación entre los objetos
 - Restricciones (Costes, Tamaño, Peso...)
 - Rendimiento (Velocidad, Precisión, ...)
- Creación de una lista conjunta
- Discusión de la lista conjunta
- Redacción de miniespecificaciones
- Redacción de un borrador de la especificación (analista)

•Ingeniería del Software

Pressman 5, pp. 184

Pressman 6, pp.166-170 Nota: en esta versión no se llama al proceso TFEA (es un tipo de JAD)

Introducción



GLOSARIO

4.- Ingeniería de requisitos

- 4.0.- Introducción.
- 4.1.- Primeras cuestiones.
- 4.2.- Obtención de requisitos
- 4.3.- Análisis de requisitos**
 - 4.3.1.- Del Sistema**
 - 4.3.2.- Del Software**
- 4.4.- Especificación.
- 4.5.- Validación de requisitos.
- 4.6.- Administración de requisitos.

•Ingeniería del Software

Análisis de requisitos del sistema

OBJETIVO:

Representar el sistema en su totalidad

Incluyendo: Hardware, Software, Personas

Sin entrar en la estructura interna de los diversos componentes

Fases del AR del sistema

- Identificación de las necesidades del cliente.
- Análisis de alternativas.
- Asignación de funciones a cada uno de los elementos del sistema.
- Evaluación de la viabilidad del sistema.
 - Económica.
 - Técnica.
 - Legal.
 - Operativa.
 - Plazos Temporales.
- Obtención de una definición del sistema que sea la base del trabajo posterior.

Fases del AR del sistema

- **Identificación de las necesidades del cliente.**
- Análisis de alternativas.
- Asignación de funciones a cada uno de los elementos del sistema.
- Evaluación de la viabilidad del sistema.
 - Económica.
 - Técnica.
 - Legal.
 - Operativa.
 - Plazos Temporales.
- Obtención de una definición del sistema que sea la base del trabajo posterior.

Fases ARS: Identificación de necesidades

- Técnicas de recogida de información:
 - Objetivos, funciones, rendimiento, restricciones, etc.
- Distinguiremos entre:
 - Lo que el cliente necesita (imprescindible)
 - Lo que el cliente quiere (útil, pero no imprescindible)
- Despliegue de función de calidad (QFD) [Pressman, 6^a, 171]
 - Requisitos Normales (Vitales en REM)
 - Requisitos esperados (Importantes en REM)
 - Requisitos estimulantes (Quedaría bien en REM)

•Ingeniería del Software

Requisitos Normales: El cliente pide y necesita.

Requisitos esperados: El cliente no pide y necesita.

Requisitos estimulantes: El cliente no pide pero le resultan satisfactorios.

Análisis del sistema

Definir los elementos del S. Informático

- Identificar funciones
- Análisis de alternativas
- Asignarlas a componentes (soft, hard, rrhh)

Partimos de:

OBJETIVOS
ÁMBITO
RESTRICCIONES



Representamos:

Funcionalidad del sistema
La información que maneja
Comportamiento
Interfaces
Rendimiento
Restricciones

•Ingeniería del Software

Alcance (PMI) ≡ Requisitos. ¿Ámbito ≡ Contexto de operación?

Se trata de pasar de una descripción vaga (no necesariamente) a una descripción precisa

Sommerville habla de pasar de Requisitos de usuario a Requisitos del sistema.

Ejemplo: Clasificador de paquetes

- Especificación informal
- Funcionalidad del sistema
- Información que se maneja
- Comportamiento
- Interfaces del sistema
- Rendimiento
- Restricciones

•Ingeniería del Software

Ejemplo: Clasificador de paquetes

Especificación informal del sistema de clasificación de paquetes.

El sistema de clasificación de paquetes debe realizarse de forma que los paquetes que se mueven a lo largo de una cinta transportadora sean identificados (para lo cual van provistos de un código numérico) y clasificados en alguno de los seis contenedores situados al final de la cinta. Los paquetes de cada tipo aparecen en la cinta siguiendo una distribución aleatoria y están espaciados de manera uniforme. La velocidad de la cinta debe ser tan alta como sea posible; como mínimo el sistema debe ser capaz de clasificar 10 paquetes por minuto. La carga de paquetes al principio de la cinta puede realizarse a una velocidad máxima de 20 paquetes por minuto. El sistema debe funcionar las 24 horas del día, siete días a la semana.

•Ingeniería del Software

Ejemplo: Clasificador de paquetes

Funcionalidad del sistema

Realizar la clasificación de paquetes que llegan por una cinta transportadora en seis compartimentos distintos, dependiendo del tipo de cada paquete.

Información que se maneja

Los paquetes disponen de un código numérico de identificación.

Debe existir una tabla o algoritmo que asigne a cada número de paquete el contenedor donde debe ser clasificado.

Ejemplo: Clasificador de paquetes

Comportamiento

El sistema realiza el proceso de identificación cuando detecta al paquete en la zona de identificación (por tiempo o sensor)

El sistema activa el clasificador cuando se ha reconocido el tipo de paquete y éste entre en la zona de clasificación (por tiempo o sensor).

Ejemplo: Clasificador de paquetes

Interfaces del sistema.

El sistema de clasificación se relaciona con otros dos sistemas:

La cinta transportadora. Parece conveniente que el sistema de clasificación pueda parar el funcionamiento de la cinta o del sistema de carga de paquetes en la cinta, en caso de que no pueda realizar la clasificación (por ejemplo si se produce una avería).

(TBD - To Be Determined, PD Por Determinar)

- **TBD:** Existe interfaz con el sistema de sustitución de contenedores. ¿Qué ocurre si se llenan?
 - **CAUSA:** El documento no recoge nada al respecto
 - **SOLUCIÓN:** Hablar con el cliente / Observar el funcionamiento actual

•Ingeniería del Software

TBD: Existe interface con el sistema de sustitución de contenedores.
¿Qué ocurre si se llenan?

causa: El documento no recoge nada al respecto

solución: Hablar con el cliente / Observar el funcionamiento actual

Ejemplo: Clasificador de paquetes

Rendimiento.

El sistema debe ser capaz de clasificar al menos 10 paquetes por minuto. No es necesario que el sistema sea capaz de clasificar más de 20 paquetes por minuto.

Restricciones.

El sistema debe tener un funcionamiento continuo. Por tanto, debemos evitar la parada del sistema incluso en el caso de que para alguno de los componentes del mismo se averíen.

- **TBD:** ¿Cuál es el porcentaje máximo que se puede tolerar de paquetes que pueden ser clasificados de forma errónea?
 - **CAUSA:** El documento no indica restricciones sobre la eficacia del sistema
 - **SOLUCIÓN:** Estos detalles también deben ser aclarados con el cliente.

•Ingeniería del Software

TBD: No hay restricciones de eficacia. No se indican valores cuantitativos del % de clasificaciones erróneas aceptables

causa: No se detalla en la especificación inicial

solución: discutir con el cliente / Comparar con competencia.

Fases del AR del sistema

- Identificación de las necesidades del cliente.
- **Análisis de alternativas.**
- Evaluación de la viabilidad del sistema.
 - Económica.
 - Técnica.
 - Legal.
 - Operativa.
 - Plazos Temporales.
- Obtención de una definición del sistema que sea la base del trabajo posterior.

Ejemplo: Clasificador de paquetes

Análisis de alternativas

Asignación 1: solución manual para implementar el sistema.

- Los recursos que utiliza son básicamente personas, y se requiere además algo de documentación, definiendo las características del puesto de trabajo y del sistema de turnos y una tabla que sirva al operador para relacionar los códigos de identificación de los paquetes con el contenedor donde deben ser depositados.
- Poca inversión para poner en marcha este sistema.
- Costes de funcionamiento serán elevados. Gran cantidad de mano de obra (turnos de trabajo)
- Ventaja: Cantera de personal, no es necesario personal cualificado.
- Problema: Trabajo rutinario -> fallos sistemáticos, absentismo laboral

•Ingeniería del Software

Ejemplo: Clasificador de paquetes

Análisis de alternativas

Asignación 2: automatizamos con un mecanismo de distribución

- *Los recursos que utiliza son : hardware (el lector de códigos de barras, el **mecanismo de distribución**), software (para el lector de códigos de barras, el controlador, y una base de datos que permita asignar a cada código su contenedor)*
- *Inversión relativa para comprar o desarrollar los componentes.*
- *Costes de funcionamiento bajos.*
- *Ventaja: Mantenimiento simple, dispositivos sencillos.*
- *Problema: Costes de mantenimiento de los dispositivos mecánicos y paradas por avería o mantenimiento con cierta frecuencia*

•Ingeniería del Software

Ejemplo: Clasificador de paquetes

Análisis de alternativas

Asignación 3: automatizamos con un brazo robot

- Los recursos que utiliza son : hardware (el lector de códigos de barras, uno o varios brazos robóticos), software (para el lector de códigos de barras, el controlador del brazo, y una tabla o algoritmo de clasificación)
- Inversión inicial muy alta.
- Costes de funcionamiento bajos.
- Costes de mantenimiento del robot, y mantenimiento especializado.
- Ventaja: Mayor velocidad de funcionamiento, menor tasa de errores.
- Problema: Viabilidad técnica (¿robot?)

•Ingeniería del Software

Ejemplo: Clasificador de paquetes

Análisis de alternativas

Asignación 4: ¿Existe una solución estándar en el mercado?

- Los recursos que utiliza son : Descripción de acuerdo al fabricante.
- Inversión inicial: Coste de la aplicación.
- Costes de funcionamiento: Se calcularán de acuerdo a la descripción del fabricante.
- Costes de mantenimiento: Consultar fabricante.
- Ventaja: Estimación precisa de costes y calidad.
- Problemas: Dependencia de una tecnología que no es propia. Estabilidad de la empresa proveedora. Calidad de los productos de la empresa...

Fases ARS: Análisis de Alternativas

Análisis paramétrico: Fijamos unos parámetros de evaluación para la elección de la aproximación final al problema

Sistema de clasificación de paquetes. Criterios de evaluación.

	Alternativa 1	Alternativa 2	Alternativa 3
Inversión inicial	Nula	Moderada	Grande
Coste funcionamiento	Grande	Moderado	Moderado
Tiempo amortización	Nulo	Moderado	Grande
Fiabilidad	Moderada	Pequeña	Grande
Mantenimiento	Nulo	Grande	Moderado
Flexibilidad	Alta	Nula	Alta

•Ingeniería del Software

Análisis de alternativas

- DIAGRAMA DAFO

Debilidades	Amenazas
Negativo Presente/Internas	Negativo Futuro/Externas
Positivo Presente/Internas	Positivo Futuro/Externas
Fortalezas	Oportunidades

•Ingeniería del Software

Análisis de alternativas

- DIAGRAMA DAFO: Solución manual

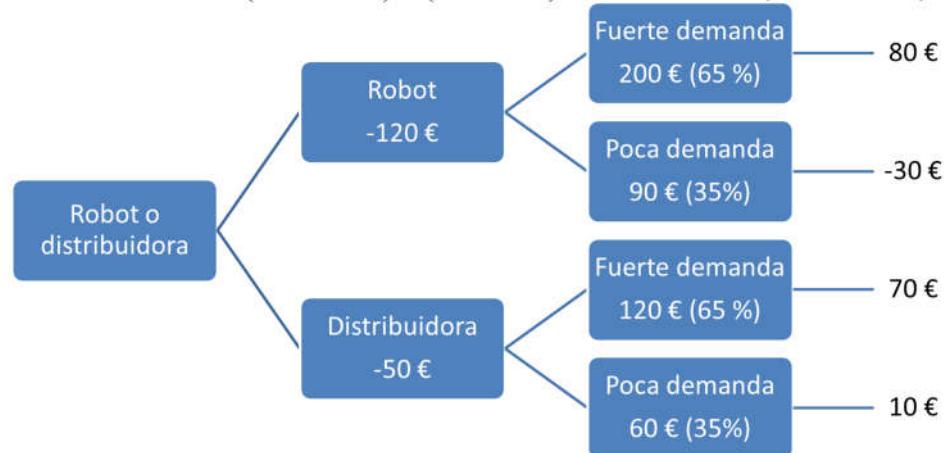
Debilidades	Amenazas
Costes de funcionamiento elevados	Absentismo laboral
Bajo coste de puesta en marcha	Possible uso del personal como cantera de la empresa
Fortalezas	Oportunidades

•Ingeniería del Software

Análisis de alternativas

- Valor monetario esperado (EMV)

$$\text{Robot} = (200 * 65\%) + (90 * 35\%) - 120 = 130 + 31,5 - 120 = 41,5$$



$$\text{Distribuidora} = (120 * 65\%) + (60 * 35\%) - 50 = 78 + 21 - 50 = 49$$

Adaptado del ejemplo en PMBOK

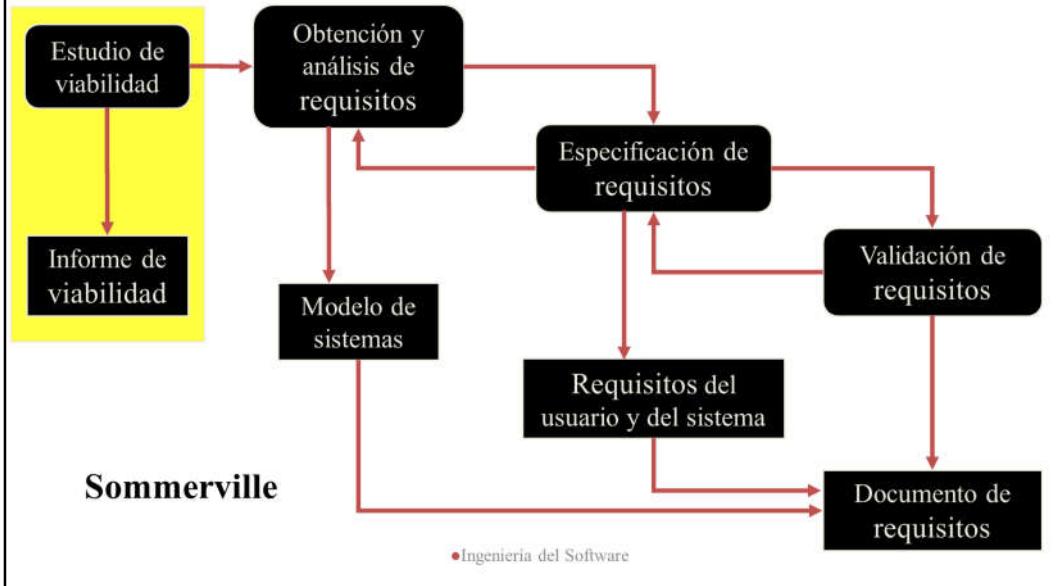
•Ingeniería del Software

Extraido del PMBok, 257-8,348

Fases del AR del sistema

- Identificación de las necesidades del cliente.
- Análisis de alternativas.
- **Evaluación de la viabilidad del sistema.**
 - Económica.
 - Técnica.
 - Legal.
 - Operativa.
 - Plazos Temporales.
- Obtención de una definición del sistema que sea la base del trabajo posterior.

Análisis de requisitos



Estudio de viabilidad

- Un estudio breve (2 ó 3 semanas)
- Utiliza la información recogida
- Esta orientado a determinar
 - Si el sistema contribuye a los objetivos de la organización
 - Si el sistema puede implantarse cumpliendo las restricciones de coste y tiempo con la tecnología actual
 - El sistema puede integrarse con los recursos ya disponibles por la organización
- OBJETIVO: Establecer si vale la pena o no desarrollar el sistema

•Ingeniería del Software

Sin limitaciones de recursos (coste, tiempo) cualquier alcance es posible con cualquier calidad.

PMI: triple restricción tiempo, coste y alcance (con la calidad como tercera dimensión, triángulo 3D)

Estudio de viabilidad

- ¿Cómo se las arreglaría la organización si no se lleva a cabo el sistema?
- ¿Cuáles son los problemas con los procesos actuales y como ayudaría el nuevo sistema a resolverlos?
- ¿Cuál es la contribución directa que haría el sistema a los objetivos del negocio?
- ¿La información se puede obtener y transferir a otros sistemas de la organización?
- ¿El sistema requiere de tecnología que no se ha utilizado previamente en la organización?
- ¿A qué debe ayudar el sistema y a qué no necesita ayudar?

Sommerville

•Ingeniería del Software

Sommerville 7, 131

Fases ARS: Estudio de viabilidad

- **Viabilidad económica:** beneficios .vs. costes de desarrollo
- **Viabilidad técnica:** Posibilidad de desarrollo en función de los recursos humanos y técnicos disponibles
- **Viabilidad legal:** ¿Infringe nuestro proyecto alguna disposición legal sobre el derecho a la intimidad, normas de seguridad, leyes de Copyright, etc.?
- **Viabilidad Operativa:** Determinar si se puede implantar de manera efectiva en la empresa. ¿Puede combinarse con los métodos existentes? ¿Encaja en la filosofía de la empresa y en la cultura del personal?
- **Viabilidad de los plazos:** Las técnicas de calendarización nos deberían permitir confirmar la viabilidad de una fecha de entrega.

•Ingeniería del Software

Fases ARS: Estudio de viabilidad

Análisis económico:

Evaluación de costes/beneficios

Dificultad para cuantificar los beneficios (mayor satisfacción, incremento en la capacidad de producción, etc.)

Costes y beneficios directos o indirectos

Ahorro en la empresa en función de:

- < tiempo para realizar tareas
- < mano de obra para realizar el mismo trabajo
- > productividad con igual mano de obra

•Ingeniería del Software

Coste del desarrollo vs Ahorro una vez implementado

Fases ARS: Estudio de viabilidad

Análisis económico:

Sistema manual actual:

7 operadores: $7 \times 15.000 \text{ €/año} \Rightarrow 105.000 \text{ €/año}$

Sistema con brazo robot:

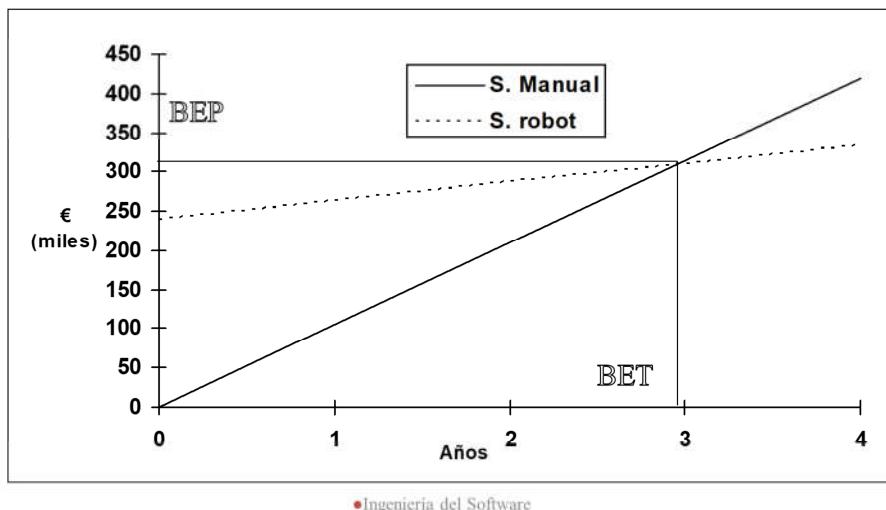
Compra y puesta en marcha:	240.000,00 €
Mantenimiento:	6.000,00 €/año
Funcionamiento:	12.000,00 €/año
Tiempo útil al año	95 %

Amortizado en 2.9 años

•Ingeniería del Software

Fases ARS: Estudio de viabilidad

Análisis económico:



BET: Break-Event Time

BEP: Break-Event Point

Habría que sumar los intereses de una posible amortización del crédito inicial necesario para poner en marcha el S. Robot.

Fases ARS: Estudio de viabilidad

Análisis técnico:

Estudio de las características:

Capacidad	¿Son realizables?
Rendimiento	¿Tenemos la tecnología?
Fiabilidad	¿Tenemos el personal formado?
Seguridad	

Riesgo técnico alto → cancelación del proyecto

•Ingeniería del Software

Para poder realizarlo debemos comprender la función y ser capaces de predecir sus características. Lo comprendemos a través de modelados, simulaciones, prototipos.

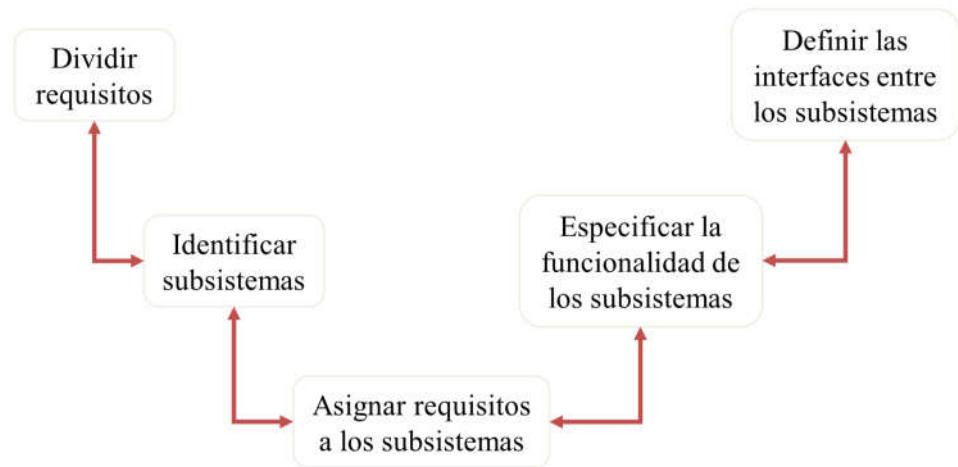
Fases del AR del sistema

- Identificación de las necesidades del cliente.
- Análisis de alternativas.
- Asignación de funciones a cada uno de los elementos del sistema.
- Evaluación de la viabilidad del sistema.
 - Económica.
 - Técnica.
 - Legal.
 - Operativa.
 - Plazos Temporales.
- **Obtención de una definición del sistema que sea la base del trabajo posterior. (Especificación del sistema)**

•Ingeniería del Software

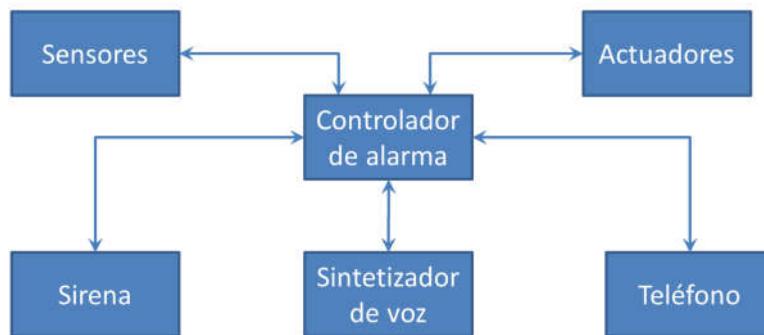
Reordenar el análisis de viabilidad como punto final.

Diseño del sistema



Representación de la arquitectura

- Hemos de hacer un diagrama del sistema:
 - 1.- Representa las relaciones entre sus elementos
 - 2.- Servirá de base para el trabajo posterior



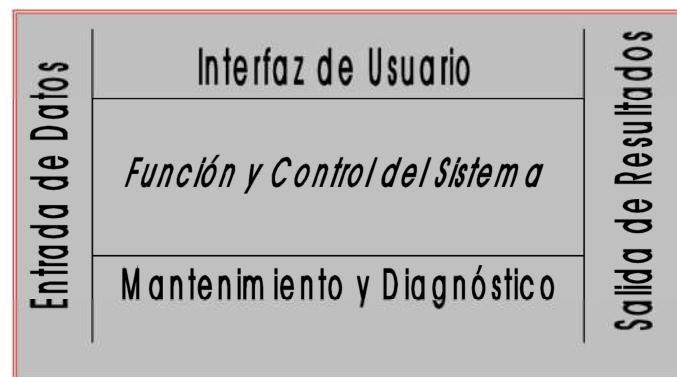
•Ingeniería del Software

Representación de la arquitectura

- Hemos de hacer un diagrama del sistema:
 - 1.- Representa las relaciones entre sus elementos
 - 2.- Servirá de base para el trabajo posterior
- En sistemas complejos utilizaremos diagramas de arquitectura en jerarquía de niveles:
 - 1.- Diagramas de contexto
 - 2.- Diagramas de flujo de arquitectura

Representación de la arquitectura

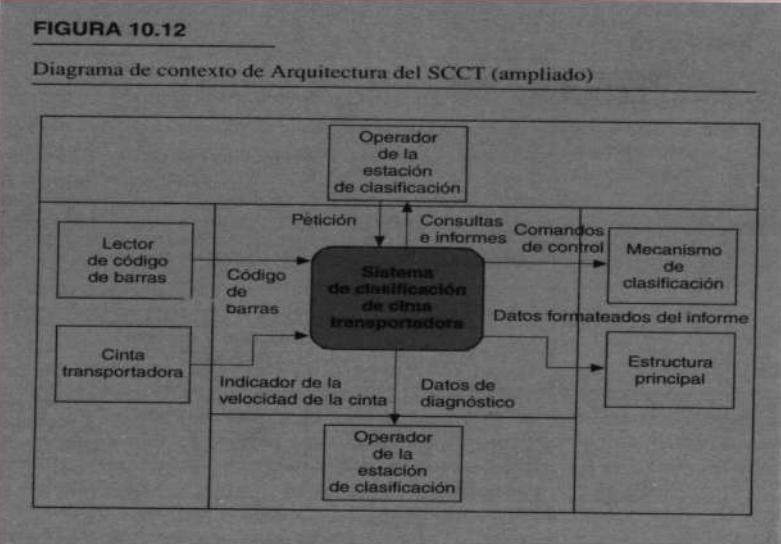
Plantilla estándar de diagramas de arquitectura



Fuente: Pressman 5^a pp175. Hatley y Pirbhai

•Ingeniería del Software

Representación de la arquitectura



•Ingeniería del Software

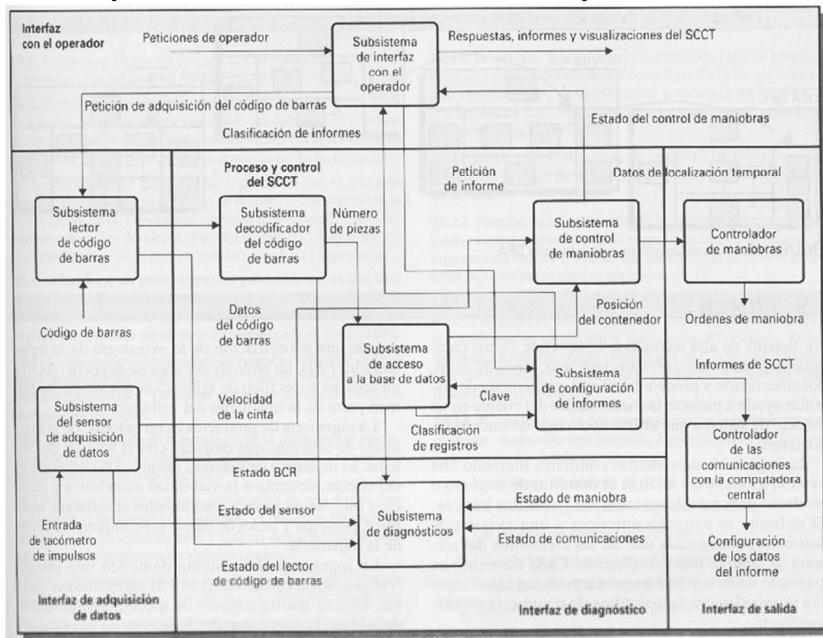
Representación de la arquitectura

2.- Diagramas de flujo de arquitectura

Expandimos el diagrama de contexto en mayor grado a través de diagramas de flujo convencionales determinando el flujo de información entre los distintos subsistemas.

Tema 4.- Análisis de requisitos - 4.3.- Análisis de requisitos

Representación de la arquitectura



•Ingeniería del Software

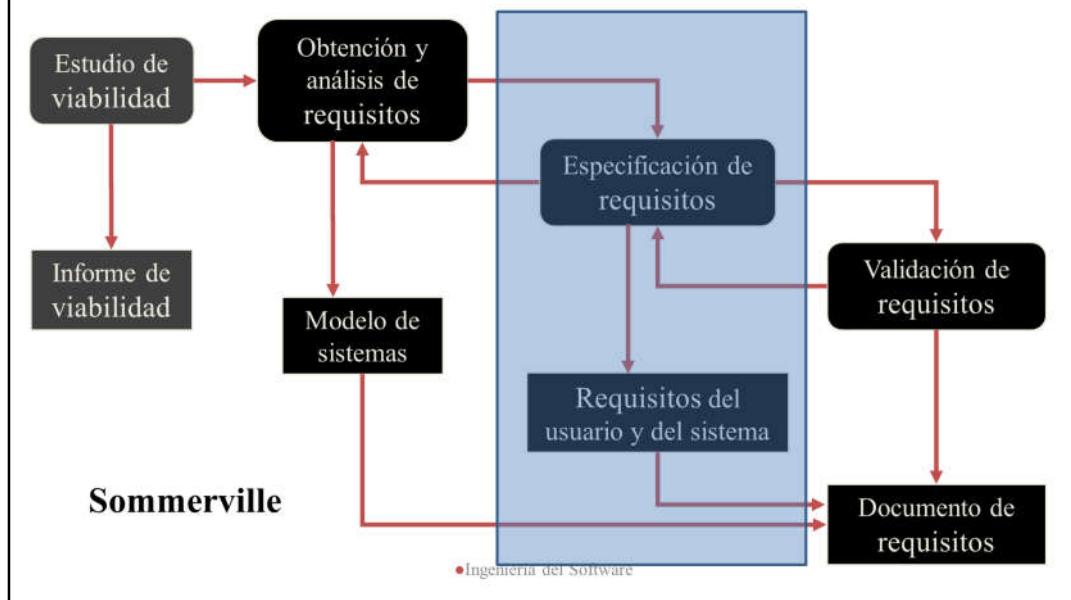
GLOSARIO

4.- Ingeniería de requisitos

- 4.0.- Introducción.
- 4.1.- Primeras cuestiones.
- 4.2.- Obtención de requisitos
- 4.3.- Análisis de requisitos
 - 4.3.1.- Del Sistema
 - 4.3.2.- Del Software
- 4.4.- Especificación.**
- 4.5.- Validación de requisitos.
- 4.6.- Administración de requisitos.

•Ingeniería del Software

Análisis de requisitos



Especificación del sistema

Es un documento que describe la función, rendimiento y restricciones que debe de cumplir el sistema y limita cada uno de sus componentes.

El analista y el cliente deben de revisar este documento para determinar si:

Especificación del sistema

Evaluación inicial de la especificación

- Se ha delimitado correctamente el ámbito del proyecto.
- Se ha definido correctamente la funcionalidad, el comportamiento, las interfaces y el rendimiento.
- Las necesidades del usuario y el análisis de viabilidad justifican el desarrollo del proyecto.
- Cliente y analista tienen la misma percepción de los objetivos del proyecto.

Especificación del sistema

Evaluación técnica de la especificación

- *Las estimaciones de riesgos, coste y agenda se corresponden con la complejidad del proyecto.*
- *Todos los detalles técnicos (asignación de funciones, interfaces, rendimientos) están bien definidos.*
- *La especificación del sistema sirve de base para las fases siguientes (en concreto para la ingeniería de requisitos del software).*

•Ingeniería del Software

Especificación de requisitos del sistema

I. Introducción.

- A. Ámbito y propósito del documento.
- B. Descripción general.
 - 1. *Objetivos.*
 - 2. *Restricciones.*

II. Descripción funcional y de datos.

- A. Diagrama de contexto de arquitectura.
- B. Descripción del DCA.

III. Descripción de los subsistemas.

- A. Especificación del diagrama de arquitectura para el subsistema i .
 - 1. *Diagrama de flujo de arquitectura.*
 - 2. *Narrativa del módulo del sistema.*
 - 3. *Rendimiento.*
 - 4. *Restricciones de diseño.*
 - 5. *Asignación de componentes.*
- B. Diccionario de la arquitectura.

IV. Resultados de la simulación del sistema.

- A. Modelo usado para la simulación.
- B. Resultados de la simulación.
- C. Aspectos especiales de rendimiento.

V. Aspectos del proyecto.

- A. Costes del proyecto.
- B. Agenda.
- C. Análisis de Viabilidad.

VI. Apéndices.

La especificación del Software.

Definición

La especificación es un documento que define, de forma completa, precisa y verificable, los requisitos, el diseño, el comportamiento u otras características de un sistema o componente de un sistema.

IEEE

Objetivo

Representar los requisitos de forma que conduzcan a una correcta implementación del sistema

Especificación del Software

- Es el documento que culmina las labores del análisis de requisitos software.
- Debe contener
 - Descripción detallada del ámbito de información.
 - Funciones y comportamiento asignados al software.
 - Requisitos de rendimiento.
 - Restricciones de diseño.
 - Pruebas de aceptación.

Principios de la especificación.

- La especificación debe modelar el dominio del problema.
- Es necesario separar funcionalidad de implementación.
- El lenguaje de especificación debe estar orientado al proceso.
- La especificación debe abarcar todo el sistema del que el software es parte.
- La especificación debe abarcar también el entorno del sistema.
- La especificación debe ser operativa.

•Ingeniería del Software

Especificación del Software

- Para que sea útil deben alcanzar las siguientes propiedades.
 - **No ambigua**
 - Completa.
 - Fácil de verificar
 - Consistente
 - Fácil de modificar
 - Facilidad para identificar el origen y las consecuencias de cada requisito.
 - Facilidad de utilización durante la fase de explotación y mantenimiento.

Especificación del Software

- Completa
 - Incluye todos los requisitos significativos
 - Define la respuesta a todas las entradas
 - Está conforme con cualquier estándar de especificación
 - Están etiquetadas todas las figuras, tablas y diagramas y definidos todos los términos y unidades.
- TBD (PD): To be Determinated (Por Determinar)
 - Condiciones que lo han causado
 - Descripción de que hay que hacer para resolverlo

Especificación del Software

- Para que sea útil debe poseer las siguientes propiedades.
 - No ambigua
 - Completa.
 - **Fácil de verificar**
 - Consistente
 - Fácil de modificar
 - Facilidad para identificar el origen y las consecuencias de cada requisito.
 - Facilidad de utilización durante la fase de explotación y mantenimiento.

Especificación del Software

- Consistente. Los requisitos no entran en conflicto
- Problemas:
 - Dos o más requisitos pueden definir el mismo objeto real pero utilizar distintos términos
 - Las características de objetos reales pueden estar en conflicto.
 - Ej. Luces rojas en un requisito son verdes en otro.
 - Conflicto lógico o temporal entre dos acciones
 - Ej. Un requisito pide que se sumen dos valores que otro indica que deben multiplicarse.

Especificación del Software

- Para que sea útil debe poseer las siguientes propiedades.
 - No ambigua
 - Completa.
 - Fácil de verificar
 - Consistente
 - **Fácil de modificar** (Estructurada, no redundante)
 - **Facilidad para identificar el origen y las consecuencias de cada requisito.** (Trazabilidad de los requisitos)
 - **Facilidad de utilización durante la fase de explotación y mantenimiento.**
 - ERS modificable
 - ERS prever cambios sobre requisitos
 - Critico / Durabilidad / Origen
 - Necesidad que genera la función. Requisitos de usuario

•Ingeniería del Software

Especificación de requisitos del software

I. Introducción.

- A. Referencia del sistema.
- B. Descripción general.
 - 1. Objetivos.
 - 2. Restricciones.

II. Descripción de la información.

- A. Representación del flujo de la información.
 - 1. Diagramas de flujo de datos.
 - 2. Diagramas de flujo de control.
- B. Representación del contenido de la información.

III. Descripción funcional.

- A. Narrativa de procesamiento.
- B. Restricciones.
- C. Requisitos de rendimiento.
- D. Restricciones de diseño.

IV. Descripción del comportamiento.

- A. Especificaciones de control.
- B. Estados del sistema.
- C. Eventos y acciones.
- D. Restricciones de diseño.

V. Criterios de validación.

- A. Descripción de las pruebas.
- B. Respuesta esperada del software.
- C. Consideraciones especiales.

VI. Apéndices.

Especificación de requisitos

IEEE/ANSI 830-1993

1. Introducción.

- 1.1. Objetivo.
- 1.2. Ámbito. Alcance del producto
- 1.3. Definiciones, siglas y abreviaturas
- 1.4. Referencias
- 1.5. Glosario del documento

2. Descripción general.

- 2.1. Perspectiva del producto.
- 2.2. Funciones del producto
- 2.3. Características del usuario
- 2.4. Restricciones generales
- 2.5. Suposiciones y dependencias

3. Requisitos específicos.

Fuertemente dependiente de la organización.

Ejemplo ⇒

Apéndices.

Índice.

3. Requisitos específicos.

- 3.1. Requisitos funcionales.
 - 3.1.1. Requisito funcional 1
 - Introducción
 - Entradas
 - Procesamiento
 - Salidas
 - 3.1.2. Requisito funcional 2
 -
 - 3.1.n. Requisito funcional n
 -
- 3.2. Requisitos no funcionales
 - Desglosar según taxonomía
- 3.3. Requisitos de dominio
 - 3.3.1. Requisitos funcionales
 - 3.3.2. Requisitos no funcionales
- 3.4. Requisitos de interfaz

•Ingeniería del Software

Especificación de requisitos: Detalle de estructura

- **Identificador:** Código único que identifica al requisito y su tipo.
- **Título:** Título del requisito
- **Descripción:** Breve descripción del requisito.
- **Importancia:** cada requisito pertenecerá a una de estas tres clases:
 - **Normales (Vital):** Reflejan los objetivos y metas establecidos para un producto. Si los requisitos están presentes, el cliente estará satisfecho.
 - **Esperados (Importante):** Están implícitos en el producto o sistema y pueden parecer tan obvios, aunque son fundamentales, que el cliente no los establece. Su ausencia causaría insatisfacción
 - **Estimulantes (Quedaría bien):** Reflejan características que van más allá de las expectativas del cliente. Su presencia sería muy satisfactoria.
- **Estabilidad: Baja, Media, Alta**
- **Trazabilidad (Rastreabilidad):**
 - **Desde este requisito:** Requisitos que tendrán que ser modificados/revisados si este requisito sufre alguna modificación
 - **Hacia este requisito:** Requisitos cuya modificación pueden influir en la definición del que estamos especificando
- **Criterio de validación:** Describe cómo comprobar el grado de cumplimiento del requisito.

Las etiquetas entre paréntesis es cómo clasifica REM

El criterio de validación en REM lo introduciremos en el campo de observaciones

Especificación de requisitos: Ejemplo (Norma IN830A)

Requisito RF1

Título: Activación de la alarma por el sensor de ruido

Descripción: El sensor de ruido situado en la vivienda activará una alarma y mandará una incidencia al sistema central si detecta suficiente ruido cuando el sistema de protección esté activado.

Importancia: Normal

Criterio de validación: Se considerará que el requisito se cumple si el sensor activa la alarma y manda un aviso cuando el ruido capturado sea mayor que 50db y la respuesta sea <1sg.

Ejemplo de requisito especificado con REM

FRQ-0009	Captura de audio de un micrófono
Versión	1.0 (06/11/2013)
Autores	• Manel Cotos
Fuentes	?
Dependencias	<ul style="list-style-type: none">• [FRQ-0001] Conexión de datos GPIO• [FRQ-0002] Conexión de datos Zwave• [FRQ-0007] Gestión de ficheros
Descripción	El sistema deberá permitir el envío de un fichero de texto al altavoz del dispositivo, como por ejemplo, resultado de una alarma
Importancia	quedaría bien
Urgencia	puede esperar
Estado	pendiente de verificación
Estabilidad	baja
Comentarios	Realmente este es un caso particular de FRQ-007, pero se hace explícito por haber salido en conversación enconcreto

•Ingeniería del Software

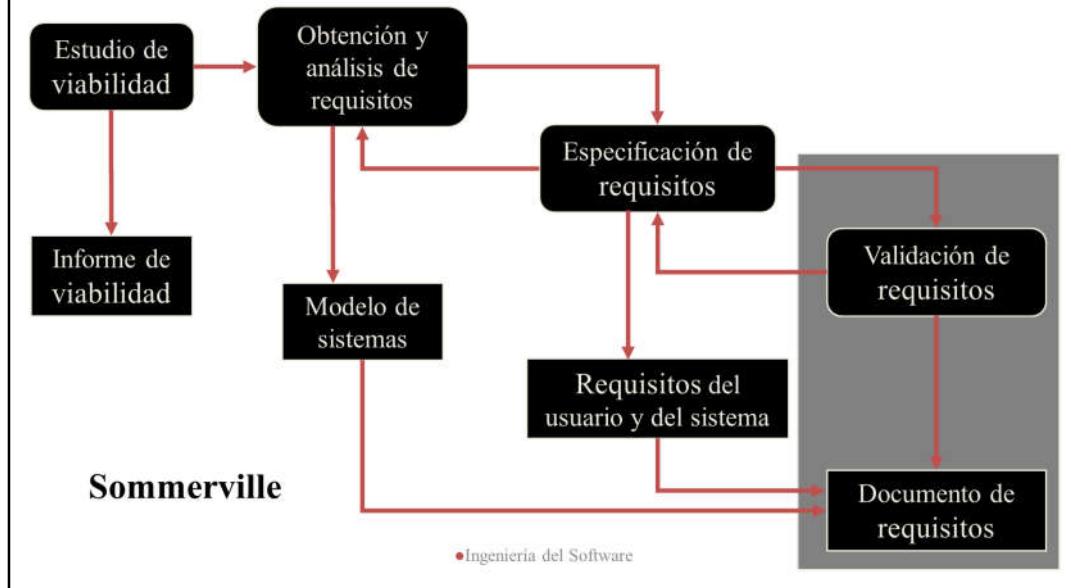
GLOSARIO

4.- Ingeniería de requisitos

- 4.0.- Introducción.
- 4.1.- Primeras cuestiones.
- 4.2.- Obtención de requisitos
- 4.3.- Análisis de requisitos
 - 4.3.1.- Del Sistema
 - 4.3.2.- Del Software
- 4.4.- Especificación.
- 4.5.- Validación de requisitos.**
- 4.6.- Administración de requisitos.

•Ingeniería del Software

Introducción



Verificación & Validación de Requisitos

Sommerville

1. Comprobar la validez:
 1. ¿Es válida mi especificación?
 2. ¿Responde a todas las necesidades?
2. Comprobar la consistencia
 1. ¿Hay incongruencias entre requisitos?
3. Comprobar la totalidad
 1. ¿Nos falta algo por especificar?
4. Comprobar el realismo
 1. Es posible implementar la funcionalidad requerida
5. Comprobar si es validable
 1. ¿Soy capaz de crear pruebas asociadas a la validación de los requisitos?

•Ingeniería del Software

Sommerville 9, p. 110

Verificación & Validación de Requisitos Estrategias

- Análisis de consistencia
- Generación de casos de uso/prueba
- Construcción de prototipos
- Revisión de Requisitos

•Ingeniería del Software

Sommerville 9, p. 110

Verificación & Validación de Requisitos

Características Especificación. **Validación y revisión.**

Piattini

1. No ambigua
2. Completa.
3. Fácil de verificar
4. Consistente
5. Fácil de modificar
6. Facilidad para identificar el origen y las consecuencias de cada requisito.
7. Facilidad de utilización durante la fase de explotación y mantenimiento.

Sommerville

1. Comprensible
2. Verificar la integridad
3. Verificabilidad
4. Verificar la consistencia
5. Adaptable
6. Rastreabilidad
7. Verificar la validez
8. Verificar el realismo

•Ingeniería del Software

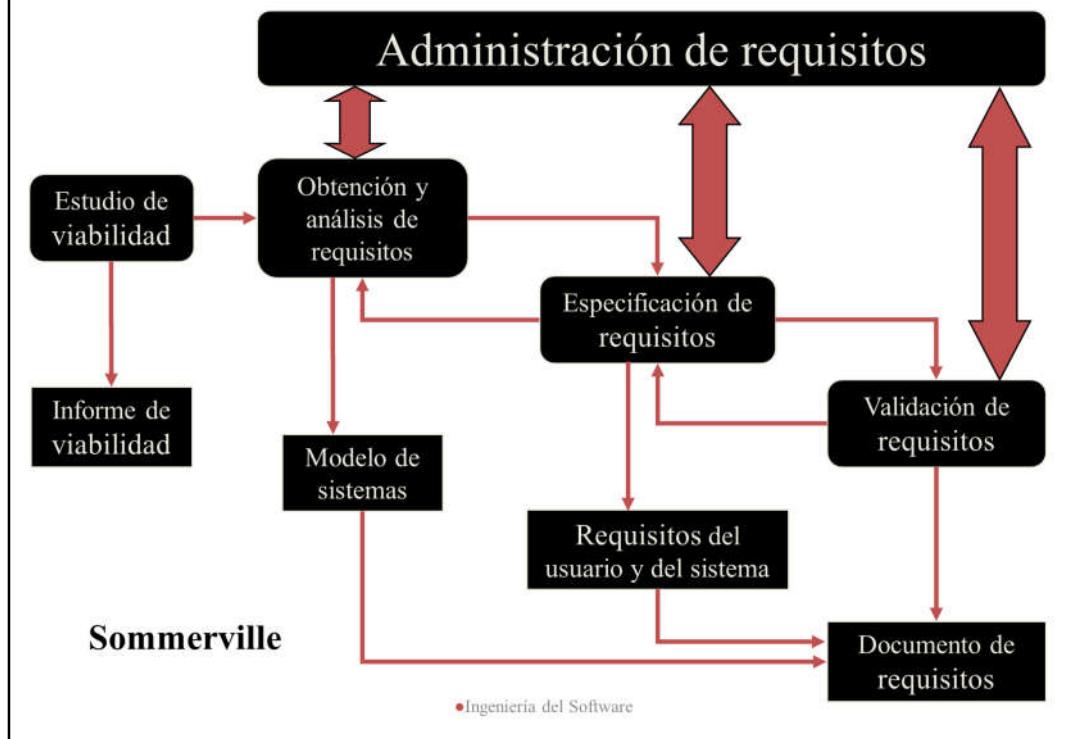
GLOSARIO

4.- Ingeniería de requisitos

- 4.0.- Introducción.
- 4.1.- Primeras cuestiones.
- 4.2.- Obtención de requisitos
- 4.3.- Análisis de requisitos
 - 4.3.1.- Del Sistema
 - 4.3.2.- Del Software
- 4.4.- Especificación.
- 4.5.- Validación de requisitos.
- 4.6.- Administración de requisitos.**

•Ingeniería del Software

Tema 4.- Análisis de requisitos - Análisis de requisitos del software



Administración de Requisitos

- La ERS es un proceso iterativo
 - Los requisitos no deben descuidarse ni ser imprecisos
 - Gestión formal de cambio
 - Rastro preciso de las modificaciones
 - Examen de la situación actual y los reemplazos
- Requisitos (Estabilidad: Baja, Media, Alta)
 - Duraderos
 - Volátiles
 - Cambiantes
 - De compatibilidad
 - Emergentes
 - Consecutivos

•Ingeniería del Software

Pressman 6, pag 162

Sommerville 9, 111

<https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/Requirements/EnduringReq.html>

Administración de Requisitos

- Tiene dos etapas
 - Planificación
 - Administración del cambio

Administración de Requisitos

Planificación

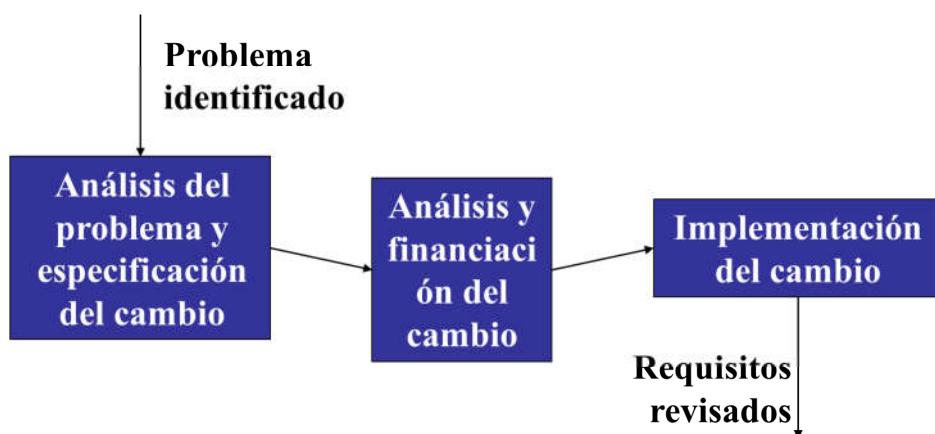
- Elegir nivel de detalle ⇒ Decidir
 - Identificación de requisitos
 - Un Proceso de administración del cambio
 - Políticas de rastreo
 - Requisito – Fuente
 - Requisito – Requisito
 - Requisito – Diseño
 - Herramientas CASE
 - Almacenar requisitos
 - Administrar el cambio
 - Administrar el rastreo

Req. Id.	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		U	R					
1.2			U			R		U
1.3	R			R				
2.1			R		U			U
2.2								U
2.3		R		U				
3.1							R	
3.2							R	

•Ingeniería del Software

Pressman 6, pag 162

Administración de Requisitos. Administración del cambio



•Ingeniería del Software

Revisar CMMI.

COMPROBAR: La administración del cambio es un proceso a parte para todos los artefactos pero el primero sobre el que se aplica son los requisitos.

CASO PRÁCTICO

CASO PRÁCTICO (3 puntos)

Desenvolvemento do software asociado á programación da entrada de datos e display (reducido) dun forno de microondas programable. Tódalas partes mecánicas do microondas (motor de xiro e xerador de microondas cando menos) serán consideradas entidades externas ao noso software. O forno disporá dun display reducido para visualizar, cando menos, a hora, o tempo de quecemento e a potencia de quecemento. Fai as suposicións que precisas para facer o exercicio, sempre e cando non contradigan os enunciados.

Detalla formalmente as funcionalidades asociadas aos seguintes casos de uso (segundo o modelo empregado nas prácticas):

CAU1: Quentar un vaso de leite

CAU2: Establecer a hora no display. (O forno ten función de reloxoio cando esta en standby)

CASO PRÁCTICO: Requisitos erróneos

ID	RF1
Título	Calentar un vaso de leche
Descripción	El sistema deberá permitir calentar un vaso de leche en base a la potencia deseada por el usuario y al tiempo deseado
Importancia	Normal
Estabilidad	Alta
Criterio de Validación	El usuario puede especificar una potencia y un tiempo de funcionamiento del microondas

•Ingeniería del Software

Importancia: Normal, Esperado, Estimulante

Estabilidad: Media, Baja, Alta

Error1: El caso de uso es un ejemplo. El requisito es calentar un líquido, o simplemente calentar. ¿Qué pasa si pongo agua?

Error2: A pesar del error anterior, hay una cierta ambigüedad en la redacción: se calienta el vaso, o el contenido del vaso

CASO PRÁCTICO: Requisitos erróneos

ID	RF2
Título	Calentar
Descripción	El sistema deberá permitir calentar el contenido de un recipiente (plato, vaso o similar no metálico) a una temperatura dada en base a una potencia y un tiempo, establecidos por el usuario
Importancia	Normal
Estabilidad	Alta
Criterio de Validación	El usuario puede especificar una potencia y un tiempo de funcionamiento del microondas

Pista: Pensad en el sistema, no en el software

•Ingeniería del Software

Importancia: Normal, Esperado, Estimulante

Estabilidad: Media, Baja, Alta

Error: Análisis de inviabilidad técnica: Imposibilidad de implementación en el sistema (no en el software). No podemos medir la temperatura de lo que contiene el recipiente

CASO PRÁCTICO: Requisitos erróneos

ID	RF2
Título	Calentar
Descripción	El sistema deberá permitir calentar el contenido de un recipiente (plato, vaso o similar no metálico) en base a una potencia y un tiempo, establecidos por el usuario. La potencia estará entre 0 y 800 , y el tiempo entre 1 y 15.
Importancia	Normal
Estabilidad	Alta
Criterio de Validación	El usuario puede especificar una potencia y un tiempo de funcionamiento del microondas

•Ingeniería del Software

Importancia: Normal, Esperado, Estimulante

Estabilidad: Media, Baja, Alta

Error: en este caso el error viene de la falta de unidades. Podríamos asumir que fuesen Watios y minutos, pero aún así el software podría servir para una gama de modelos en los que la potencia máxima no fuese 800.

CASO PRÁCTICO: Requisitos erróneos

ID	RF3
Título	Establecimiento de hora en display
Descripción	El sistema deberá permitir al usuario establecer la hora en un display
Importancia	Normal
Estabilidad	Alta
Criterio de Validación	Se puede ver la hora y modificarla

•Ingeniería del Software

Importancia: Normal, Esperado, Estimulante

Estabilidad: Media, Baja, Alta

Error: Qué es la hora? Necesitaríamos un formato de hora

CASO PRÁCTICO: Requisitos erróneos

ID	RF3
Título	Establecimiento de hora en display
Descripción	El sistema deberá permitir al usuario establecer la hora en un display, en formato HH:MM:SS
Importancia	Normal
Estabilidad	Alta
Criterio de Validación	Se puede ver la hora y modificarla

•Ingeniería del Software

Importancia: Normal, Esperado, Estimulante

Estabilidad: Media, Baja, Alta

Error: No es suficiente. Este requisito nos permitiría introducir las 25:67:91

CASO PRÁCTICO: Requisitos erróneos

RF3: Correctamente especificado

ID	RF3
Título	Establecimiento de hora en display
Descripción	El sistema deberá permitir al usuario establecer la hora en un display, en formato HH:MM:SS
Importancia	Normal
Estabilidad	Alta
Criterio de Validación	La hora establecida por el usuario está en formato HH:MM:SS Y además la visualización de HH está en el rango [00-23], y la de MM y SS en el [00-59]

•Ingeniería del Software

Importancia: Normal, Esperado, Estimulante

Estabilidad: Media, Baja, Alta

Error: No es suficiente. Este requisito nos permitiría introducir las 25:67:91

CASO PRÁCTICO: Requisitos erróneos

ID	RF3
Título	Establecimiento de hora en display
Descripción	El sistema deberá permitir al usuario establecer la hora en un display, en formato HH:MM:SS, de forma que con un botón el usuario pudiese elegir entre HH, MM y SS, y con una rueda giratoria moverse entre cada uno de los valores
Importancia	Normal
Estabilidad	Alta
Criterio de Validación	La hora establecida por el usuario está en formato HH:MM:SS Y además la visualización de HH está en el rango [00-23], y la de MM y SS en el [00-59]

•Ingeniería del Software

Importancia: Normal, Esperado, Estimulante

Estabilidad: Media, Baja, Alta

Error: A pesar de estar bien especificado en general, entra en detalles de implementación del sistema. ¿Por qué una rueda giratoria?

CASO PRÁCTICO: Requisitos erróneos

ID	RF3
Título	Establecimiento de hora en display
Descripción	El sistema deberá permitir al usuario establecer la hora en un display, en formato HH:MM:SS, de forma que los valores se almacenen en una clase previamente definida, que tenga atributos para cada parámetro de tipo integer.
Importancia	Normal
Estabilidad	Alta
Criterio de Validación	La hora establecida por el usuario está en formato HH:MM:SS Y además la visualización de HH está en el rango [00-23], y la de MM y SS en el [00-59]

•Ingeniería del Software

Importancia: Normal, Esperado, Estimulante

Estabilidad: Media, Baja, Alta

Error: A pesar de estar bien especificado en general, entra en detalles de implementación del sistema: Vamos a tener clases? Se va a hacer en JAVA, o en C, con software embebido?

GLOSARIO

Tema 5. Modelado del Análisis estructurado.

5.1. Introducción.

5.2. Técnicas.

5.2.1. Diagramas de flujo de datos.

5.2.2. Especificaciones de proceso

5.2.3. Diagramas de flujo de control.

5.2.4. Especificaciones de control

5.2.5. Diagramas de estados.

5.2.6. Redes de Petri

5.2.7. Diagramas Entidad/Relación.

5.2.8. Diccionario de datos.

5.2.9. Consistencia entre modelos. Técnicas matriciales.

5.4. Metodología del análisis estructurado.

5.4.1. Fases.

5.5. Modelos del sistema: esencial y de implementación.

5.6. Ejemplos.

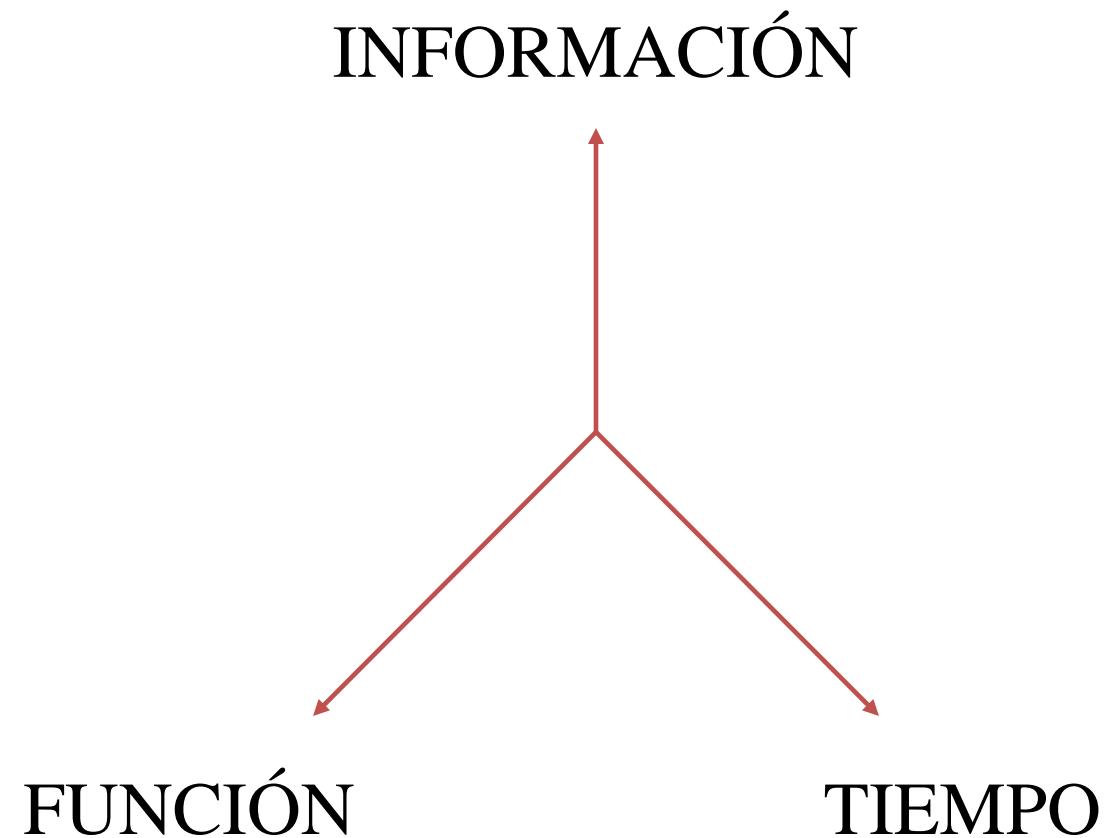
Introducción: Construcción de modelos

- Análisis Clásico basado en documentos:
 - Eran monolíticos: había que leerlos de principio a fin.
 - Eran redundantes → inconsistencias en los cambios
 - Eran ambiguos → estaban en lenguaje natural
 - Eran imposibles de mantener o modificar
- Nuevos métodos de análisis:
 - Gráficas: Colección de diagramas comentados
 - Material de referencia. No es el cuerpo principal.
 - Particionadas: Se puede trabajar sobre partes individuales
 - Mínimamente redundantes
 - Transparentes: Fáciles de leer y entender

Introducción: Construcción de modelos

- Ayudan a entender y corregir
 - Permite centrarse en determinadas características del sistema, dejando de lado otras menos significativas.
 - Permite realizar cambios y correcciones en los requisitos a bajo coste y sin correr ningún riesgo.
- Ayudan a representar y transmitir
 - Permite verificar que el ingeniero del software ha entendido correctamente las necesidades del usuario.
 - Se usan para describir el sistema a los desarrolladores.
- Ayudan en el proceso de reflexión. (Feedback)
 - Comprobamos que cada fase verifica los modelos.

Introducción: Análisis estructurado



Introducción: Análisis estructurado

Punto de vista de los datos.

Se centra en la información que utiliza el sistema. Se describen los datos y sus relaciones.

Para ello utilizaremos:

- Diagramas de Estructura de datos. (DED)
- Diagramas Entidad/Relación. (DER)

TODOS LOS MODELOS DESCRIBEN EL MISMO
SISTEMA

Introducción: Análisis estructurado

Punto de vista del proceso.

Se centra en la función del sistema. Se describe que flujos de datos recibe cada operación como ésta transforma la información y que flujos de datos genera. Para describir el sistema desde este punto de vista utilizaremos:

- **Diagramas de Flujo de Datos**
- **Especificaciones de procesos.**

**TODOS LOS MODELOS DESCRIBEN EL MISMO
SISTEMA**

Introducción: Análisis estructurado

Punto de vista del comportamiento.

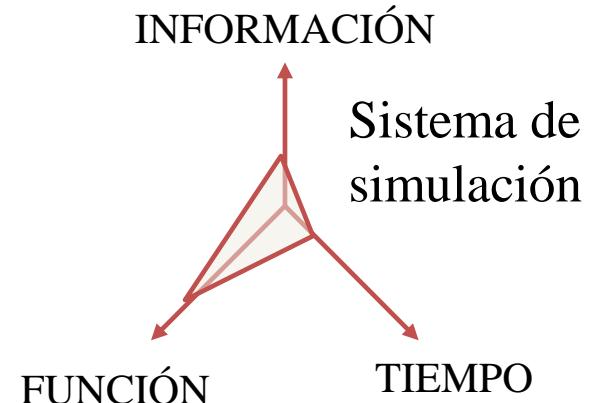
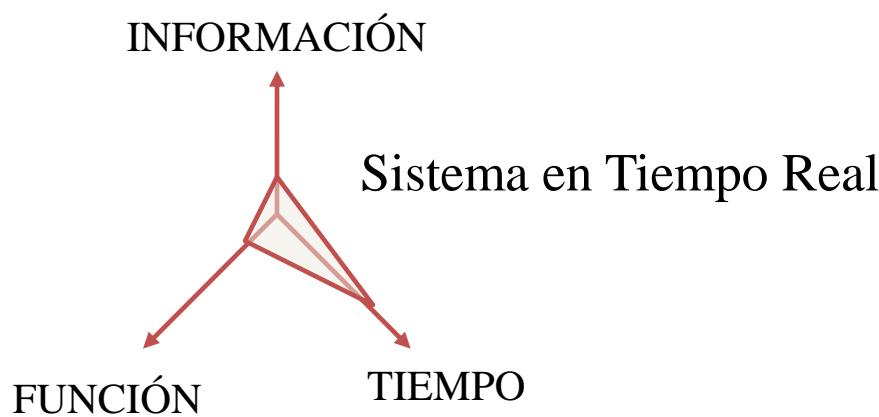
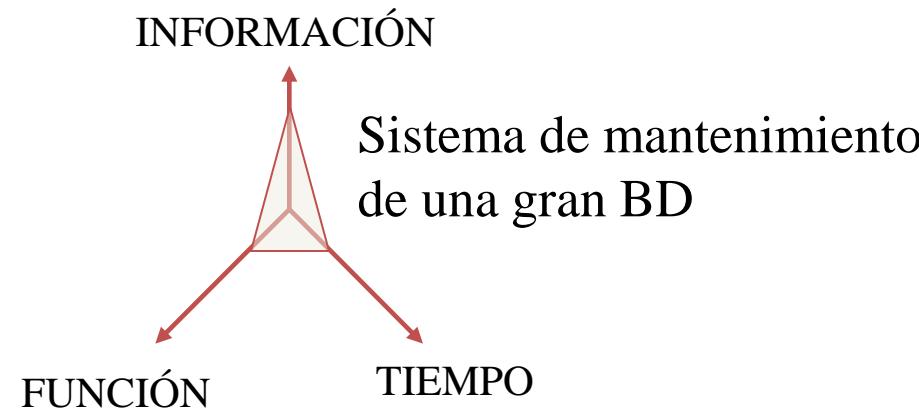
Describiremos el sistema como una sucesión de estados o modos de funcionamiento. Indicaremos también cuáles son las condiciones o eventos que hacen que el sistema pase de un modo a otro.

Utilizaremos:

- **Diagramas de Flujo de Control.**
- **Especificaciones de Control.**
- **Diagramas de Estados.**
- **Redes de Petri**

TODOS LOS MODELOS DESCRIBEN EL MISMO SISTEMA

Introducción: Análisis estructurado



Introducción: Análisis estructurado

La relación entre los diagramas de cada dimensión se establece con las técnicas que representan los planos formados por cada dos dimensiones.

**TODOS LOS MODELOS DESCRIBEN EL MISMO
SISTEMA**

Introducción: Análisis estructurado

Diferentes técnicas de modelado y especificación.

	Información	Función	Tiempo
Información	Diagramas Entidad-Relación (ER). Diagrama de estructura de datos (DED). Matriz Entidad/entidad Diagramas de clases.		
Función	Diagramas de Flujo de datos (DFD). Matriz Función/Entidad Diagrama de Clases Diagramas de colaboración	Diagramas de Flujo de datos (DFD). Diagramas de casos de uso Diagrama de estructura Tarjetas CRC Diagramas de Componentes Diagramas de Despliegue Diagramas de actividad	
Tiempo	Diagramas de Historia y vida de entidad. Digrama de transición de estados Diagramas de secuencia	Redes de Petri Diagramas de transición de estados Diagramas de Actividad Diagramas de secuencia	Diagramas de transición de estados Diagramas de flujo de control

GLOSARIO

Tema 5. Modelado del Análisis estructurado.

5.1. Introducción.

5.2. Técnicas.

5.2.1. Diagramas de flujo de datos.

5.2.2. Especificaciones de proceso

5.2.3. Diagramas de flujo de control.

5.2.4. Especificaciones de control

5.2.5. Diagramas de estados.

5.2.6. Redes de Petri

5.2.7. Diagramas Entidad/Relación.

5.2.8. Diccionario de datos.

5.2.9. Consistencia entre modelos. Técnicas matriciales.

5.4. Metodología del análisis estructurado.

5.4.1. Fases.

5.5. Modelos del sistema: esencial y de implementación.

5.6. Ejemplos.

Diagramas de Flujo de Datos. DFD

- Qué funciones son las que realiza el sistema.
- Qué interacción se produce entre estas funciones.
- Qué transformaciones de datos realiza el sistema. Qué datos de entrada se transforman en qué datos de salida.

Diagramas de Flujo de Datos. DFD

ELEMENTOS

I. Proceso

Procesos. Representan elementos software que transforman información.

Deben verificar las reglas:

- **De Conservación de Datos:** El proceso recibe todos los datos necesarios para generar su salida
- **De Pérdida de Información:** Todas las entradas del proceso tienen que utilizarse para calcular alguna de sus salidas.

Diagramas de Flujo de Datos. DFD

ELEMENTOS

Procesos. Representan elementos software que transforman información.

Entidades externas. Representan elementos del sistema informático o de otros sistemas adyacentes que producen información que va a ser transformada por el software o que consumen información transformada por el software.

Almacenes de datos. Representan información almacenada que puede ser utilizada por el software. En la mayoría de los casos, utilizaremos almacenes de datos cuando dos procesos intercambian información pero no ocurren o se ejecutan simultáneamente.

Flujos de datos. Representan datos o colecciones de datos que fluyen a través del sistema. La flecha indica el sentido de flujo. Posiblemente en los diagramas de nivel mayor existan *par de diálogo* y *flujos múltiples*. Pueden ser discretos o continuos.

I. Proceso

Entidad
Externa

Almacén
de Datos

Flujo

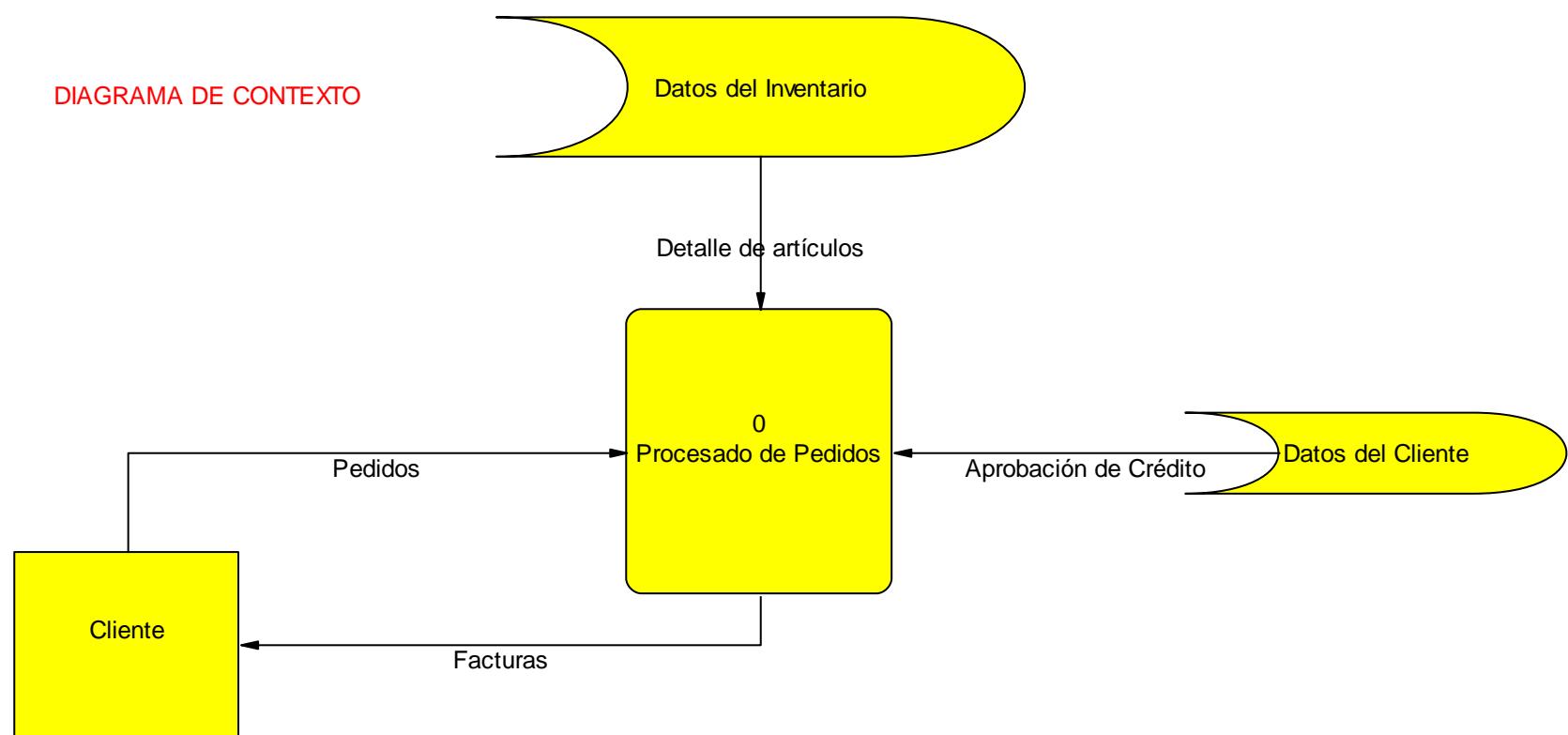
Diagramas de Flujo de Datos. DFD

Otras Notaciones

	Yourdon, DeMarco	Gane y Sarson	SSADM MÉTRICA
Flujos de Datos	→	→	→
Procesos	○		
Almacenes de datos	— —		
Entidades externas	□	□□	○

Diagramas de Flujo de Datos. DFD

EJEMPLO



Diagramas de Flujo de Datos. DFD

- Los diagramas de Flujo de Datos sólo dicen que hace el sistema.
- Los diagramas de Flujo de Datos no representan el comportamiento.

NO DICEN:

- Cuando se hace.
- En qué secuencia se hace.

Diagramas de Flujo de Datos. DFD

DESCOMPOSICIÓN EN NIVELES DE UN DFD

- Los sistemas grandes (su DFD no cabe en una página) se descomponen por niveles en una aproximación top-down
 - Cada proceso de un DFD “explota” en un DFD de nivel superior.
- Cada nivel no debe contener más de 7 ± 2 procesos
- No es conveniente usar más de 7 u 8 niveles
- Los niveles superiores dan una visión más detallada de los procesos de niveles inferiores. Cabe distinguir
 - Nivel 0: Diagrama de Contexto
 - Nivel I: Diagrama 0 o de Sistema
 - Niveles intermedios
 - Procesos primitivos

Diagramas de Flujo de Datos. DFD

DESCOMPOSICIÓN EN NIVELES DE UN DFD

- Diagrama de contexto – Diagrama de Nivel 0
 - Primer diagrama de la jerarquía. Nivel 0.
 - Resume el requisito principal del sistema.
 - Todo el sistema se representa como un proceso, “caja negra”. PROCESO 0
 - Se representan todas las entidades externas con las que se relaciona el sistema
 - Su objetivo es delimitar la frontera entre el sistema y el mundo exterior y definir sus interfaces. Flujos de información con las entidades externas.

Diagramas de Flujo de Datos. DFD

DESCOMPOSICIÓN EN NIVELES DE UN DFD

- Diagrama del sistema.
 - Es el diagrama que descompone el proceso 0 del diagrama de contexto. Por esto se denomina Diagrama 0, (no confundir con diagrama de Nivel 0)
 - Debe representar las funciones principales que realiza el sistema.
 - Es interesante que estas funciones sean independientes entre sí.

Diagramas de Flujo de Datos. DFD

DESCOMPOSICIÓN EN NIVELES DE UN DFD

- Procesos primitivos
 - Son aquellos aquellos procesos que ya no explotan en nuevos niveles de DFDs.
 - Cada Función primitiva debe describirse en una especificación.
 - Cuando detenerse
 - Cuando la función puede expresarse en una página
 - Cuando los procesos tienen pocos flujos de entrada/salida
 - Cuando descomponer implica perder el significado de la función. Se generan diagramas demasiado sencillos que complican la comprensión global.

Diagramas de Flujo de Datos. DFD

ESTRATEGIA DE CREACIÓN

- Diagrama de contexto
 - Se debe localizar todas las entidades y definir sus flujos con precisión (Interfaces).
- Diagrama de sistema.
 - Se deben seleccionar las funciones principales
 - Definir los flujos entre estas funciones. Normalmente a través de almacenes
 - Recoger los flujos del diagrama de contexto. Normalmente cada uno entrará en un proceso distinto. No conviene dividir estos flujos múltiples.
- Resto de diagramas.
 - No descomponer al máximo
 - Subfunciones principales de cada proceso
 - Interfaces entre sus procesos
 - Se recogen los interfaces (flujos) de nivel superior y se asignan a alguno de los procesos.
 - Se pueden desglosar flujos múltiples
- Ojo
 - Interfaces complejas.
 - Redes desconectadas.
 - Particionamiento desigual.
 - Posición de almacenes.

Diagramas de Flujo de Datos. DFD

REGLAS DE CONSTRUCCIÓN

- Todos los elementos del DFD tienen que tener **nombre**
 - Deben ser únicos.
 - Debe mantenerse su consistencia entre los DFDs. DD
 - Deben ser breves.
 - Lo más representativos posibles:
 - Los procesos deben definir la función.
 - Los flujos la información que transportan.
 - Los almacenes la información que contienen.
 - Las entidades externas a la entidad que representan.
 - No deben ser genéricos o inespecíficos. “Realizar operación”
- Excepción
 - Flujos que entren o salgan de almacenes simples, en cuyo caso la estructura de estos flujos es la misma que la del almacén.

Diagramas de Flujo de Datos. DFD

REGLAS DE CONSTRUCCIÓN

- Convenciones de **numeración**
 - El proceso del diagrama de contexto es siempre numerado como 0
 - Los procesos del diagrama de sistema se enumeran por un entero empezando en el 1.
 - Cada diagrama recibe el número y el nombre del proceso padre que descompone.
 - Los restantes niveles tienen sus procesos numerados con la concatenación del número del diagrama en el que están más un punto y un entero que lo identifica en el diagrama.

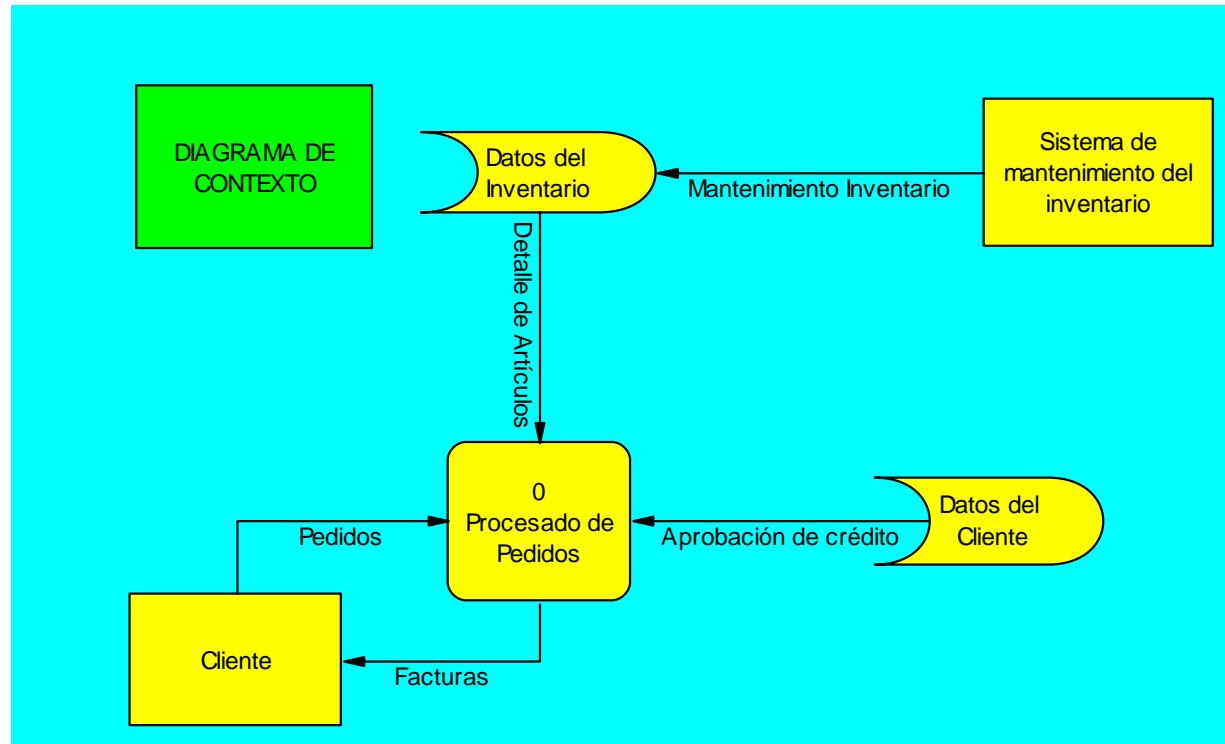
Diagramas de Flujo de Datos. DFD

REGLAS DE CONSTRUCCIÓN

- **Los almacenes**
 - Se pueden representar varias veces en un DFD si con ello se mejora su legibilidad
 - Se sitúan en el nivel más alto en el que sirven de interconexión entre varios procesos
 - Si sólo se comunica con un proceso entonces es local y debe representarse en el nivel siguiente.
 - No puede haber comunicación entre almacenes.
- **Las entidades externas**
 - Pueden igualmente representarse varias veces en un DFD
 - Normalmente sólo aparecerán en el diagrama de contexto.
 - No se representará la comunicación entre entidades externas
- La comunicación entre almacenes y entidades externas sólo se representará excepcionalmente cuando el almacén haga de interface con la entidad externa.

Diagramas de Flujo de Datos. DFD

EJEMPLO



Diagramas de Flujo de Datos. DFD

REGLAS DE CONSTRUCCIÓN

- **Flujos.**
 - Sencillos, contienen uno o varios datos
 - Múltiples, Representan como un flujo un conjunto de ellos.
- Regla del balanceo
 - Los flujos de datos que entran o salen de un proceso padre deben aparecer en el hijo manteniendo el nombre y el sentido.
 - En el diagrama hijo estos flujos se mostrarán con un extremo libre.
 - Excepción
 - Los flujos múltiples en el padre pueden descomponerse en sus componentes en el hijo por lo que haremos simultáneamente una descomposición de procesos y flujos.

Especificación de procesos (PSPEC)

Procesos primitivos

- Es un documento breve. < Página
- Complementa DFD
- Debe definir de forma más o menos formal cómo se obtienen los flujos de salida a partir de los de entrada más una información local.
- Alternativas
 - Lenguaje Natural
 - Diagramas de flujo
 - Lenguaje estructurado
 - **Diagramas de acción**
 - Árboles de decisión
 - Tablas de decisión
- Deben incluir
 - Precondiciones
 - Postcondiciones

Especificación de procesos (PSPEC)

	Lenguaje estructurado	Diagramas de acción
Secuencia	Conjunto de acciones sencillas o una estructura de las demás	<pre>[]</pre>
Alternativa	SI condición bloque FINSI	<pre>[] SI condición bloque FINSI</pre>
	SI condición bloque SI NO bloque FINSI	<pre>[] SI condición bloque SI NO bloque FINSI</pre>
Repetitiva	MIENTRAS condición bloque FIN MIENTRAS	<pre>[] MIENTRAS condición bloque FIN MIENTRAS</pre>
	REPETIR condición bloque HASTA condición	<pre>[] REPETIR condición bloque HASTA condición</pre>

Especificación de procesos (PSPEC)

Procesos primitivos

- Es un documento breve. < Página
- Complementa DFD
- Debe definir de forma más o menos formal cómo se obtienen los flujos de salida a partir de los de entrada más una información local.
- Alternativas
 - Lenguaje Natural
 - Diagramas de flujo
 - Lenguaje estructurado
 - Diagramas de acción
 - Árboles de decisión
 - Tablas de decisión
- Deben incluir
 - Precondiciones
 - Postcondiciones

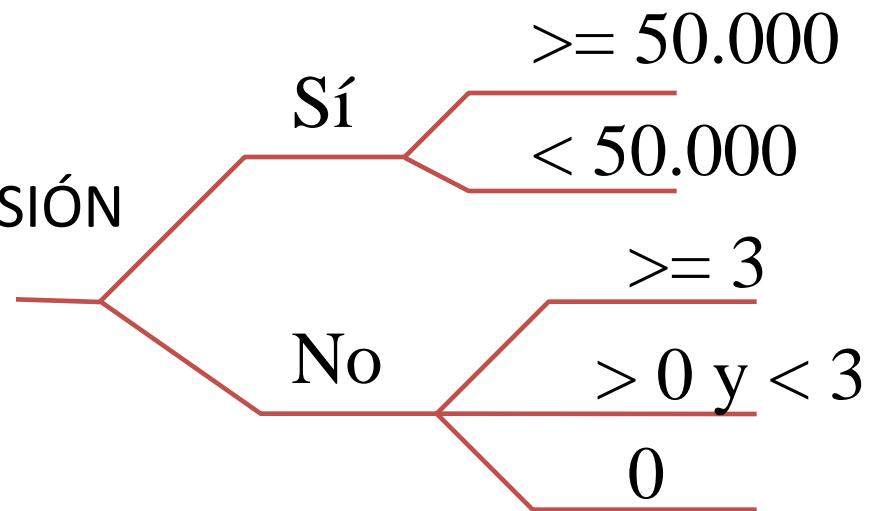
Especificación de procesos (PSPEC)

CONDICIONES	Sí	Sí	No	No	No
Condición 1	Sí	Sí	No	No	No
Condición 2	Sí	-	-	-	-
Condición 3	-	Sí	-	-	-
Condición 4	-	-	Sí	-	-
Condición 5	-	-	-	Sí	-
Condición 6	-	-	-	-	Sí

ACCIONES	X	X	X	X	X
Acción 1	X				
Acción 2		X			
Acción 3				X	

- TABLA DE DECISIÓN

- ÁRBOL DE DECISIÓN



Especificación de procesos (PSPEC)

Procesos primitivos

- Es un documento breve. < Página
- Complementa DFD
- Debe definir de forma más o menos formal cómo se obtienen los flujos de salida a partir de los de entrada más una información local.
- Alternativas
 - Lenguaje Natural
 - Diagramas de flujo
 - Lenguaje estructurado
 - Diagramas de acción
 - Árboles de decisión
 - Tablas de decisión
- Deben incluir
 - Precondiciones
 - Postcondiciones

GLOSARIO

Tema 5. Modelado del Análisis estructurado.

5.1. Introducción.

5.2. Técnicas.

5.2.1. Diagramas de flujo de datos.

5.2.2. Especificaciones de proceso

5.2.3. Diagramas de flujo de control.

5.2.4. Especificaciones de control

5.2.5. Diagramas de estados.

5.2.6. Redes de Petri

5.2.7. Diagramas Entidad/Relación.

5.2.8. Diccionario de datos.

5.2.9. Consistencia entre modelos. Técnicas matriciales.

5.4. Metodología del análisis estructurado.

5.4.1. Fases.

5.5. Modelos del sistema: esencial y de implementación.

5.6. Ejemplos.

Diagrama de Flujo de Control DFC

1. Los procesos que figuran en el DFD están activos siempre. En este caso no necesitamos especificar el control del sistema.
2. Los procesos se activan cuando llegan datos a través de sus flujos de entrada, transforman estos datos y emiten los resultados a través de los flujos de salida, permaneciendo entonces inactivos hasta la llegada de nuevos datos. Este comportamiento está implícito en la notación usada para los DFDs por lo que tampoco será necesario especificar el control.
3. Cada proceso pasa por períodos de actividad e inactividad. Un proceso se activará cuando se produzca determinada situación o suceso en el sistema y permanecerá activo hasta que se produzca otra situación. Necesitamos especificar el control.

EJEMPLO TDC

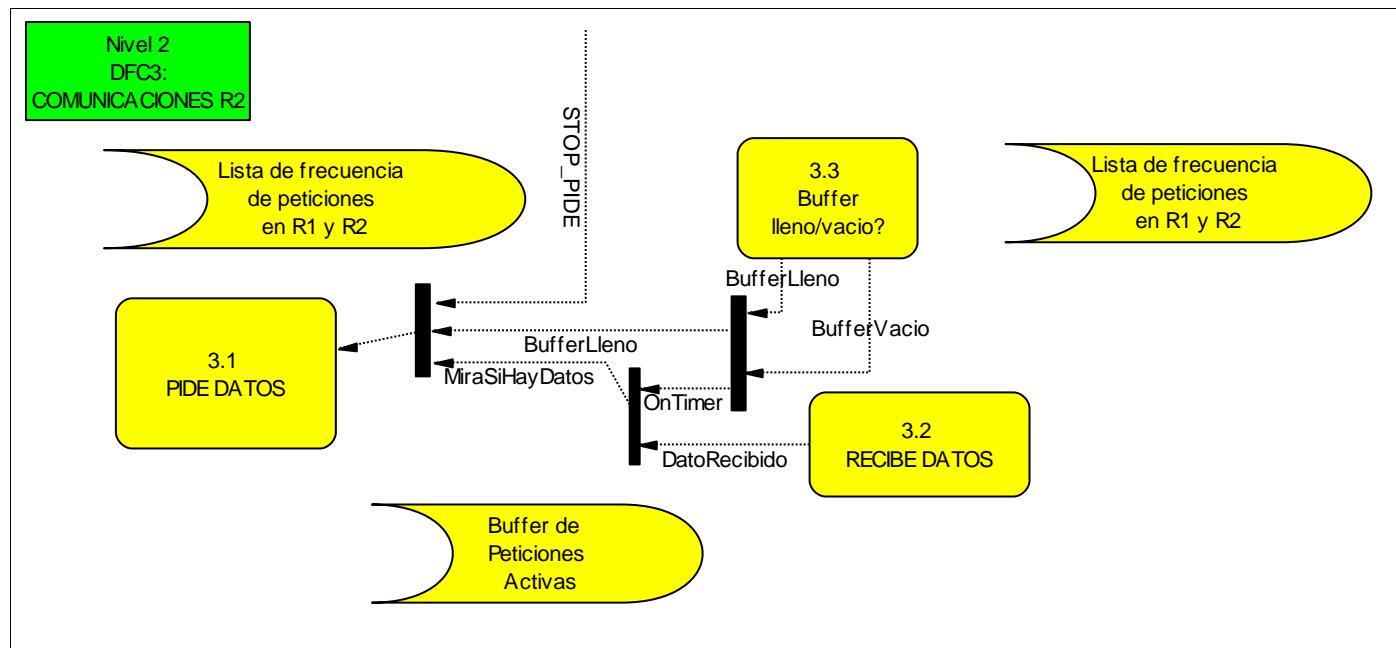
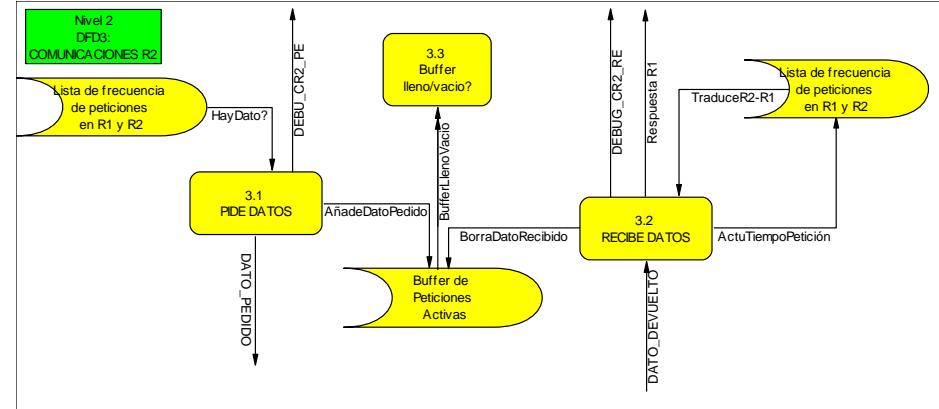


Diagrama de Flujo de Control DFC

Procesos, entidades externas y almacenes de datos. Serán los mismos y tendrán el mismo significado que en el DFD al que corresponden.

NEW

Flujos de control. Se representan mediante trazos discontinuos y modelan el flujo de información de control en el sistema. Habrá procesos o entidades externas que generen información de control y otras que la consuman.

NEW

Almacenes de control. Se representan igual que los almacenes de datos pero con trazos discontinuos. Permiten almacenar información de control, para ser utilizada posteriormente.

NEW

Ventanas a especificaciones de control. Se representan mediante barras. Estas ventanas reciben y emiten flujos de control y representan la transformación de flujos de control en el sistema.

Diagrama de Flujo de Control DFC

CONSIDERACIONES:

1. Los procesos de un DFC simplemente representan a los mismos procesos de los DFDs, y lo que indica el DFC es como fluyen los flujos de control a través de estos procesos.
 - a) No representan los estados del sistema (que se representan en los DEs).
 - b) Tampoco representan procesamiento ni transformación de los flujos de control (lo que se hace en las CSPECs)
2. Un flujo de control que entra en un proceso refleja:
 - a) Si activa o desactiva el proceso se indica en la CSPEC
 - b) Si es utilizado por algún proceso en los que éste se descompone. Se verá en el DFC y/o las CSPEC del siguiente nivel.
 - c) Que va a ser utilizado como un dato más para que el proceso funcione. Se describe en su DFD/PSPEC

Diagrama de Flujo de Control DFC

- ¿Para qué sirven los DFC?

Solo reflejan la información de control que existe en el sistema y que procesos y entidades las producen y consumen

Hemos de combinarlos con las Especificaciones de Control (CESPC) para reflejar el COMPORTAMIENTO del sistema

- ¿Cómo separar datos y control?

Señales que des/activen procesos de forma no trivial: control

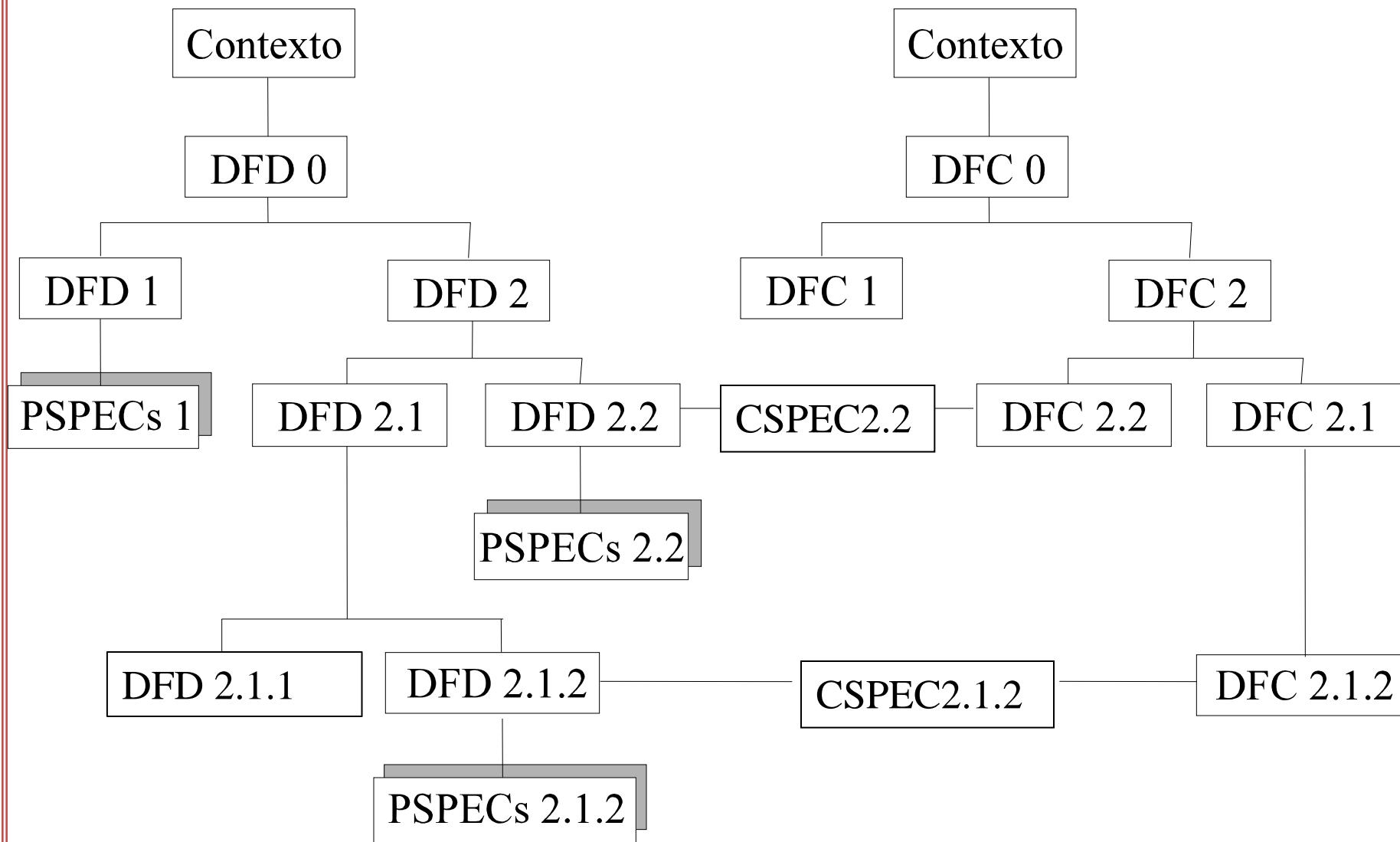
El resto: datos

Diagrama de Flujo de Control DFC

- Construcción de un DFC
 - Construimos una jerarquía de DFC's paralela a la de DFD's
 - Cada par DFC/DFD representa los mismos procesos y las mismas entidades externas
 - Sólo introducimos las señales de control que no estén implícitas en el DFD.
 - Cada DFC se desarrolla en un CSPEC

DFDs

DFCs



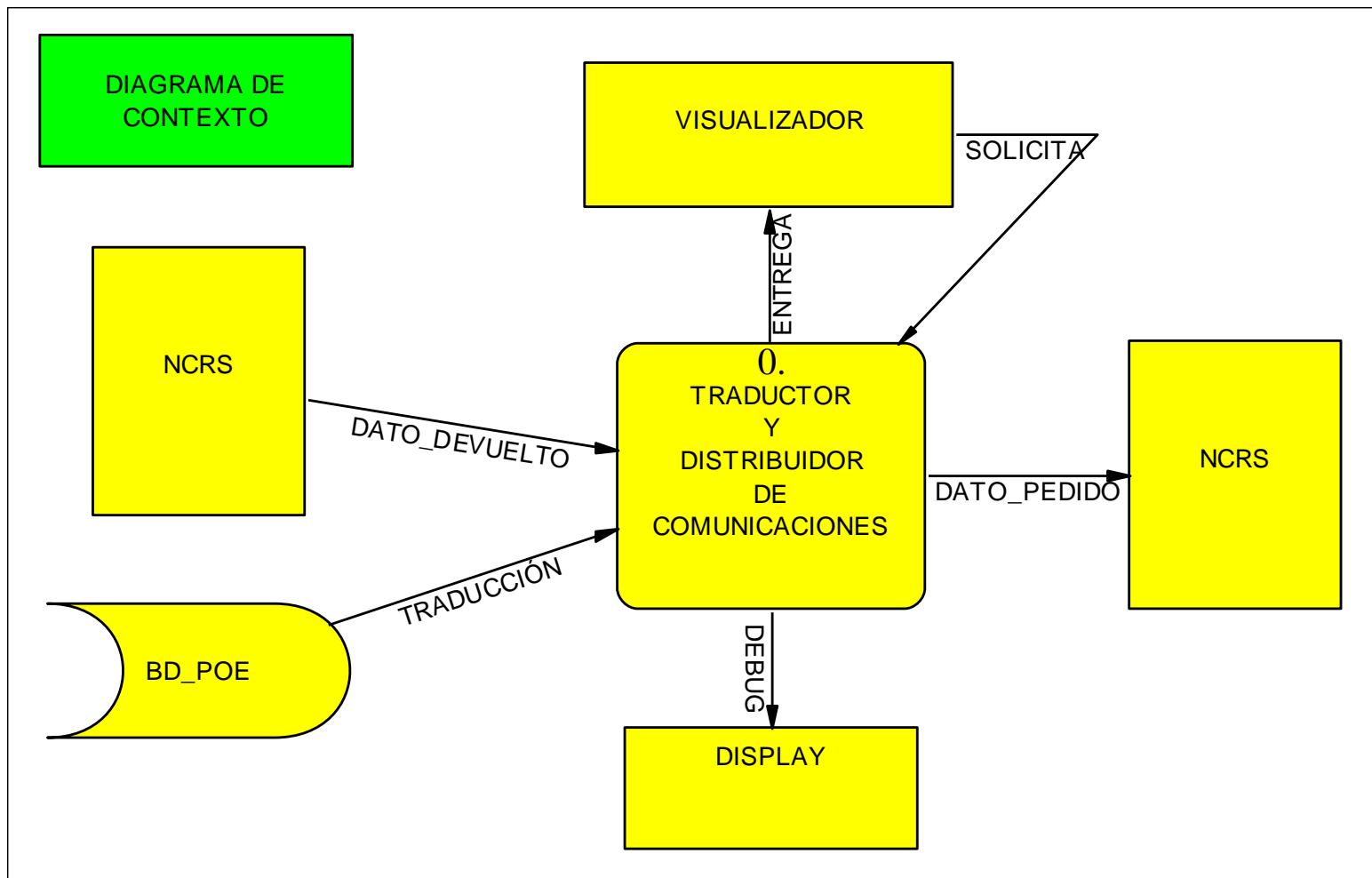
Especificaciones de Control. CSPEC

- Es un documento.
- Define detalles procedimentales de cómo se realiza el procesamiento de los flujos de control de E/S
- Se especifican con
 - a) Lenguaje Estructurado
 - b) Tablas de Activación de Procesos (combinacional)
 - c) Tablas de Decisión (combinacional)
 - d) Diagramas de estados (secuencial)
 - e) Redes de Petri (secuencial)

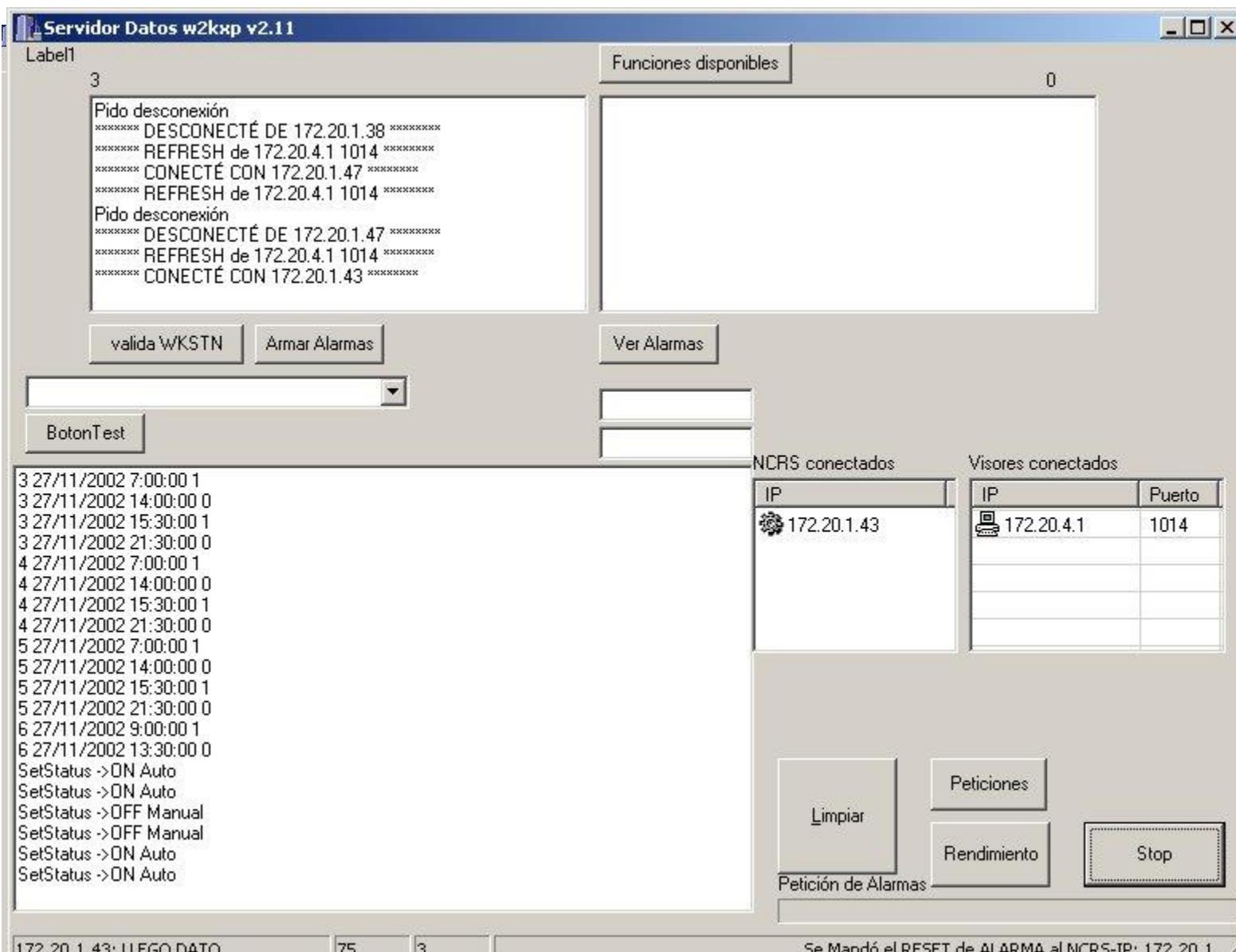
Especificaciones de Control. CSPEC

- Condiciones de datos: Flujos de control generados por un proceso.
 - Ejemplo: *Comprobar_Saldo* procesa *Número_de_cuenta* e *Importe* y genera dos flujos de control, *Aceptar* o *Rechazar Operación*
- Activadores: Señales de control especiales que sólo toman valor on/off y activan procesos. Las activations siguen la jerarquía de los modelos.
- Ventanas de Control: Indican el procesamiento de señales de control.

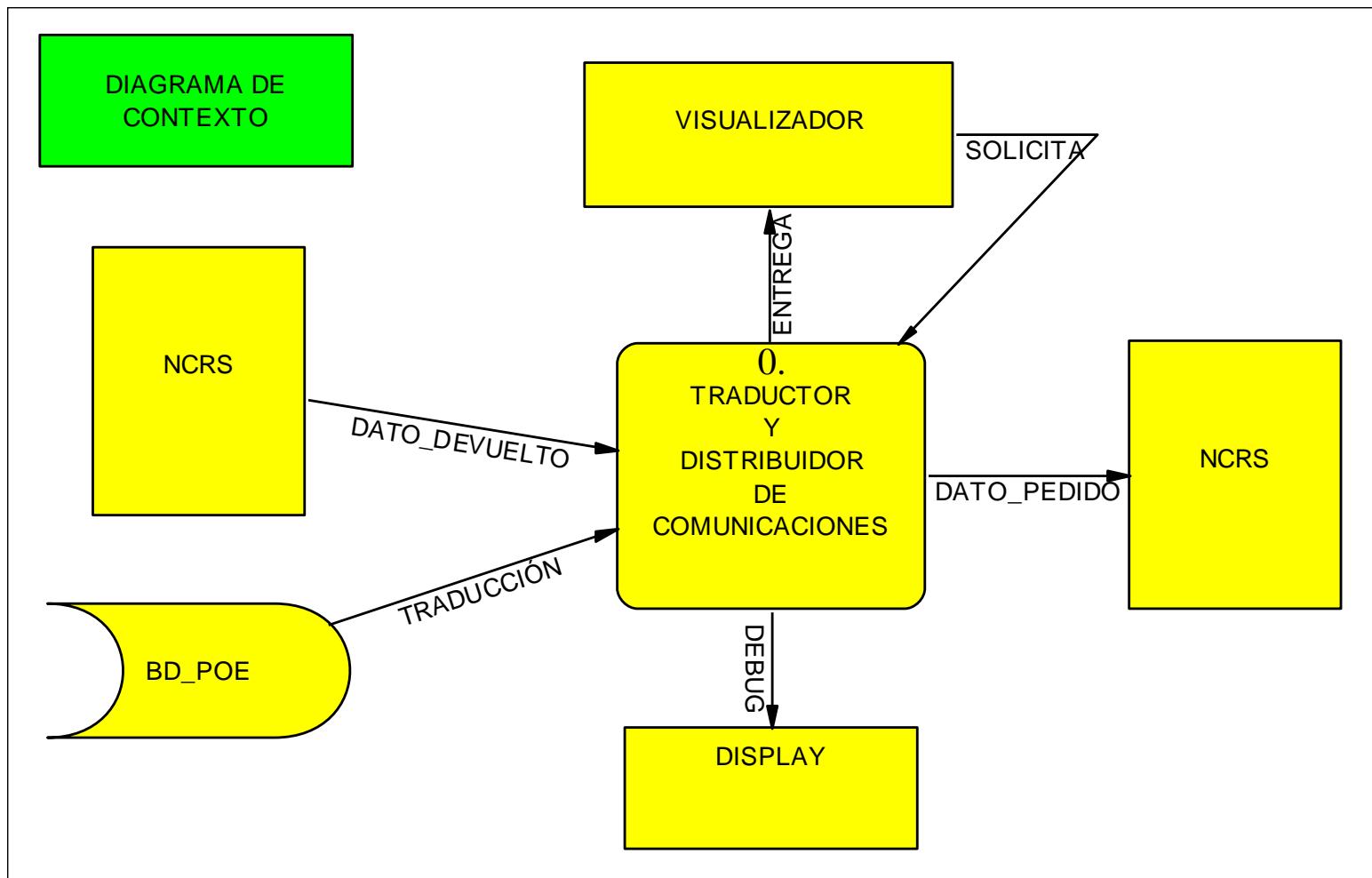
EJEMPLO TDC



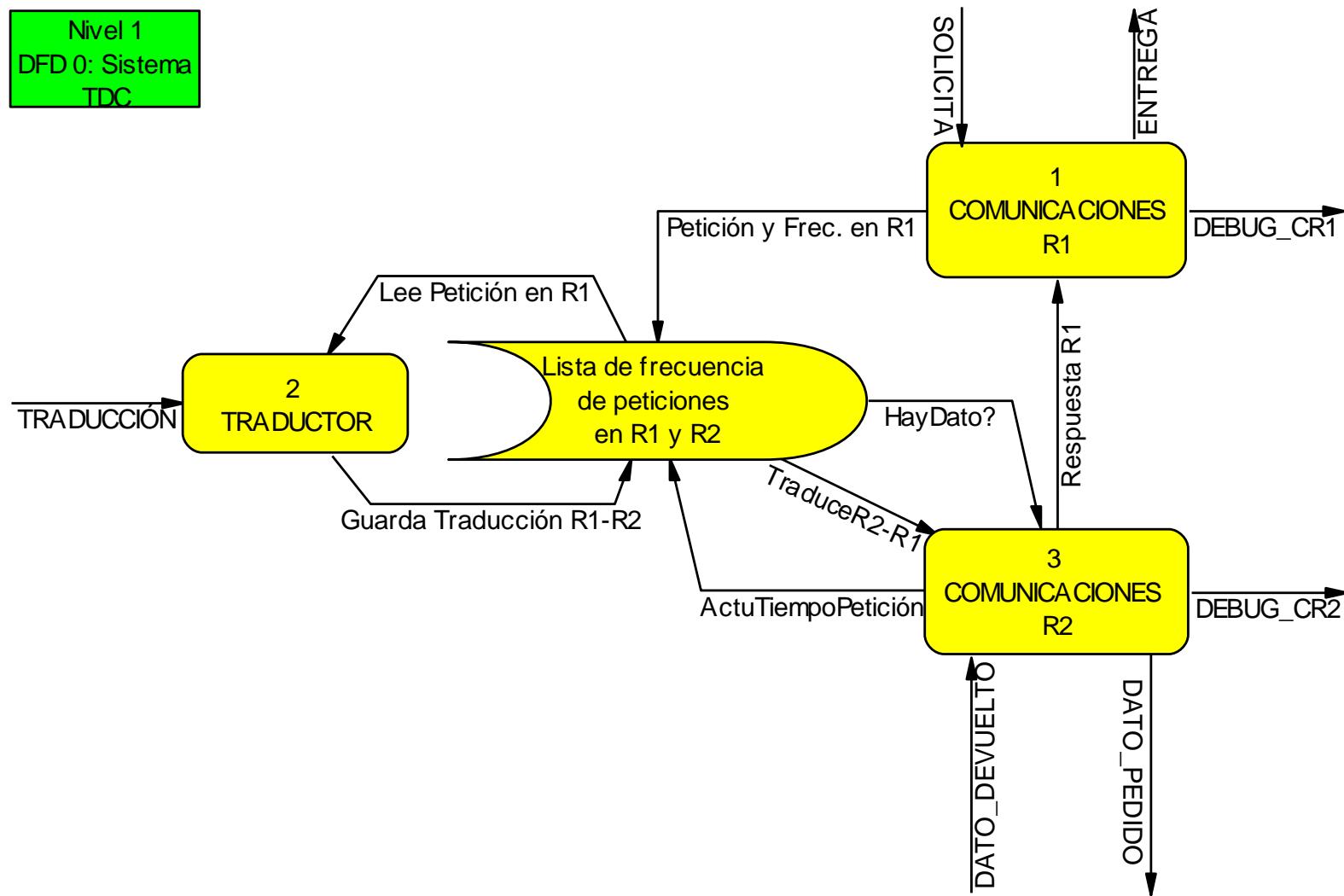
EJEMPLO TDC



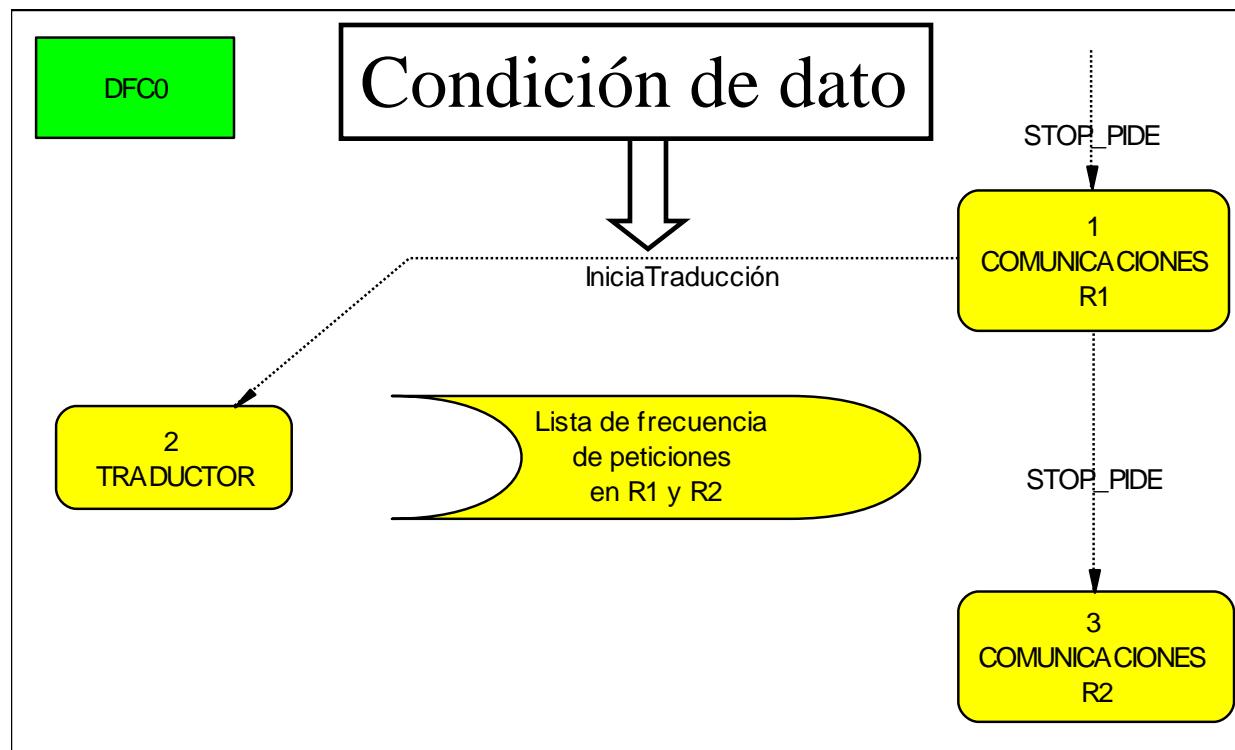
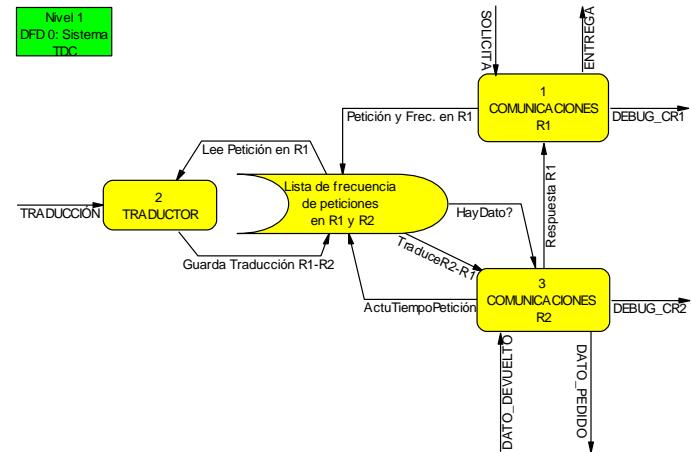
EJEMPLO TDC



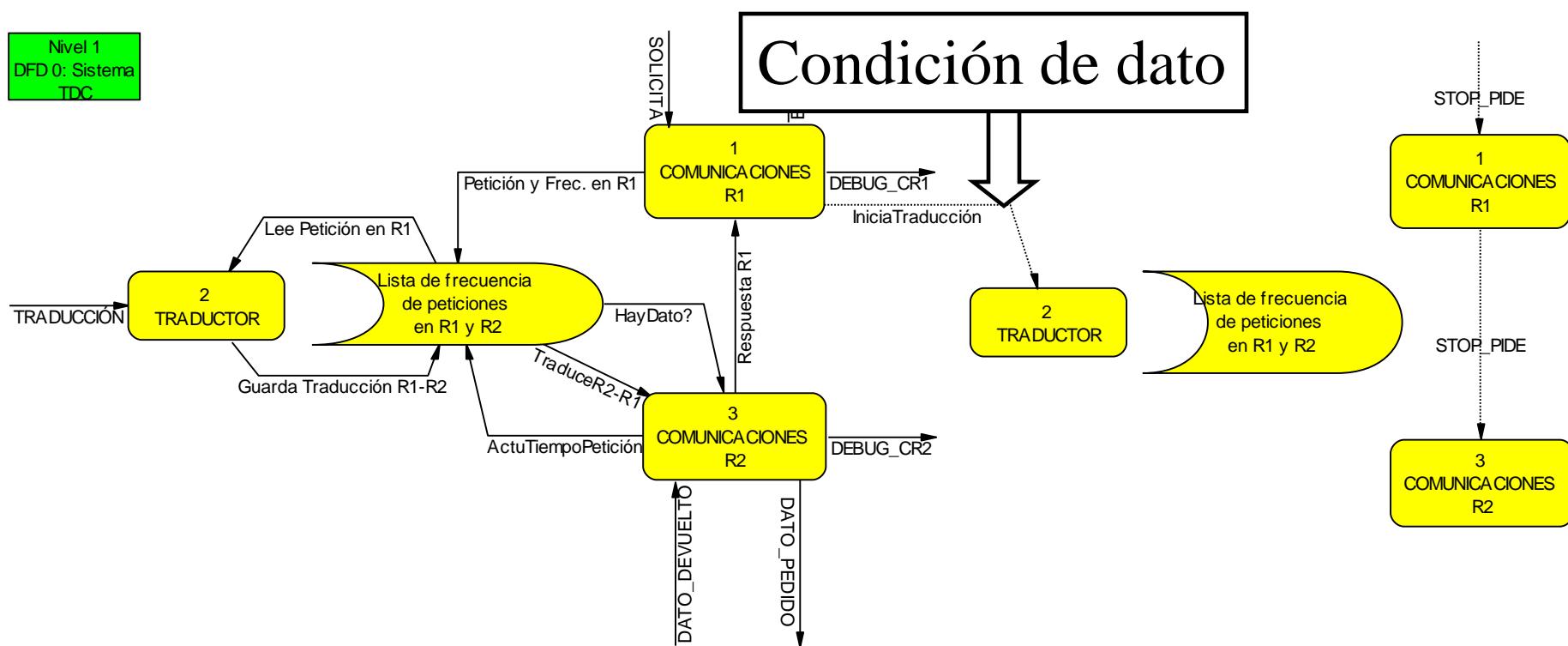
EJEMPLO TDC



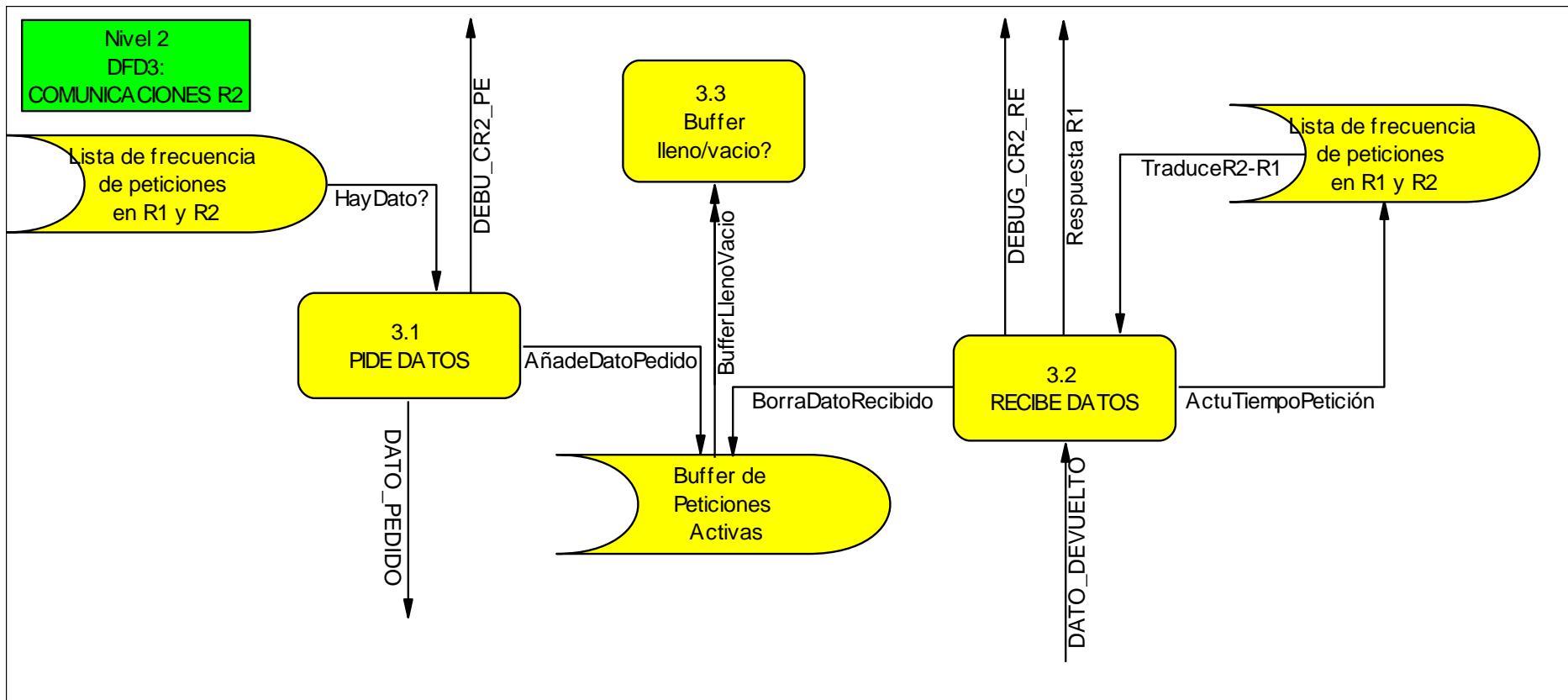
EJEMPLO TDC



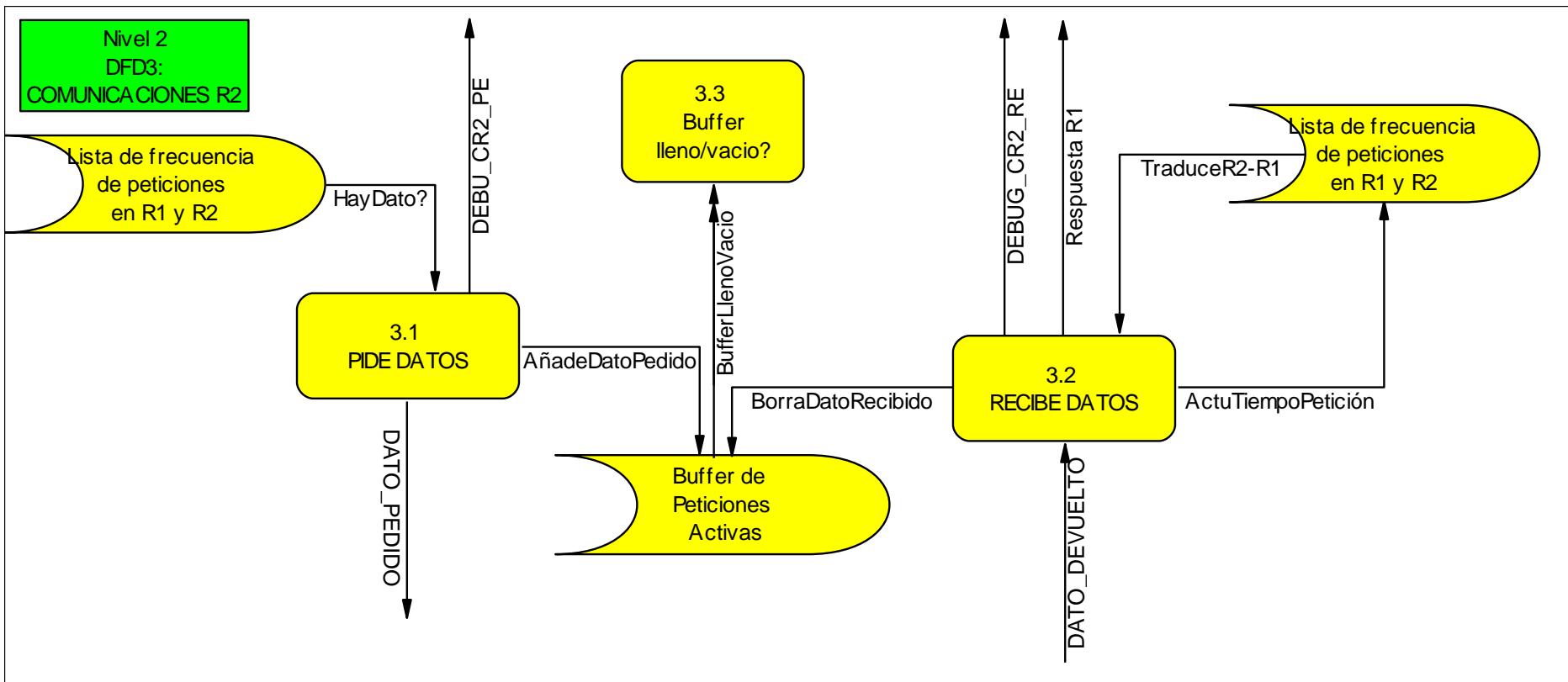
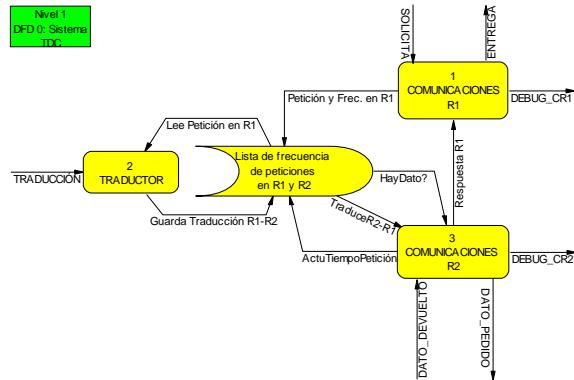
EJEMPLO TDC



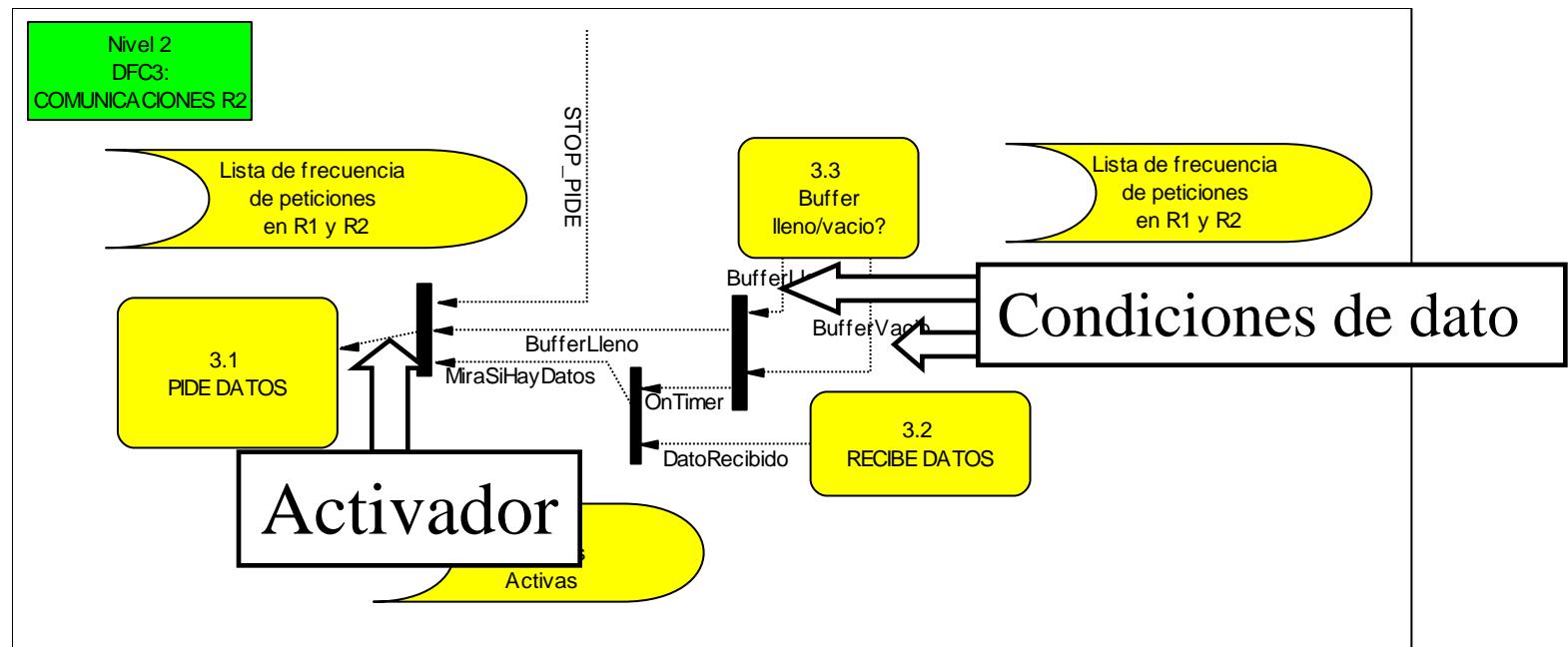
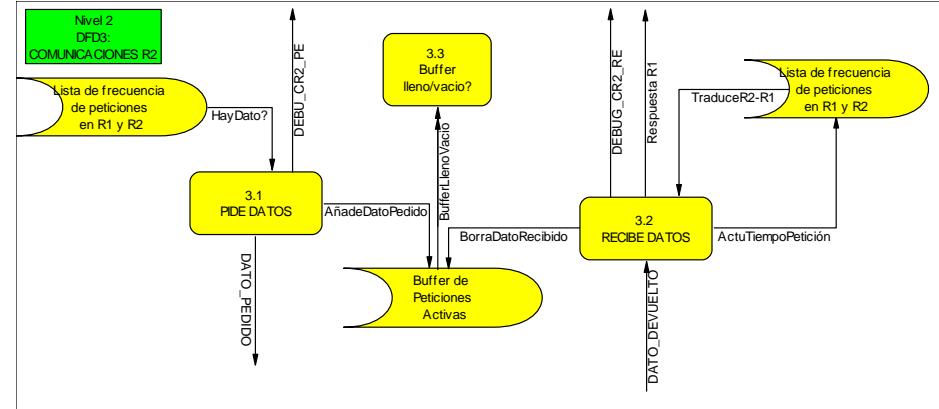
EJEMPLO TDC



EJEMPLO TDC



EJEMPLO TDC



Especificaciones de Control. CSPEC

- Es un documento.
- Define detalles procedimentales de cómo se realiza el procesamiento de los flujos de control de E/S
- Se especifican con
 - a) Lenguaje Estructurado
 - b) Tablas de Activación de Procesos (combinacional)
 - c) Tablas de Decisión (combinacional)
 - d) Diagramas de estados (secuencial)
 - e) Redes de Petri (secuencial)

Especificaciones de Control. CSPEC

- Tablas de activación de procesos: Tabla que indica bajo qué señales de control se activa un proceso dado. Cada proceso se indica en una fila y las señales de control se indican en las columnas.
- Lenguaje Estructurado: Conjunto de palabra clave elegidas en cualquier idioma que permiten definir, secuencias, condiciones, bucles y acciones a realizar
- Tablas de decisión: Tabla en la que se expresan condiciones y acciones a realizar como dos lista en una única columna. Cada columna a la derecha de ésta indicará que condiciones están activas y que acción se realiza en esas condiciones.

Especificaciones de Control. CSPEC

TABLA DE ACTIVACIÓN

	MiraSiHayDatos
3.1	1
3.2	0
3.3	0

TABLA DE DESACTIVACIÓN

	STOP_PIDE	BufferLleno
3.1	1	1
3.2	0	0
3.3	0	0

Especificaciones de Control. CSPEC

- Descripción en lenguaje estructurado
 - ACTIVACIÓN DE *MiraSiHayDatos*
SI OnTimer **O** DatoRecibido
 MiraSiHayDatos
 - ACTIVACIÓN DE *OnTimer*
SI BufferVacio
 MIENTRAS NO BufferLleno
 ESPERA Retardo.
 OnTimer

Especificación de control (CSPEC)

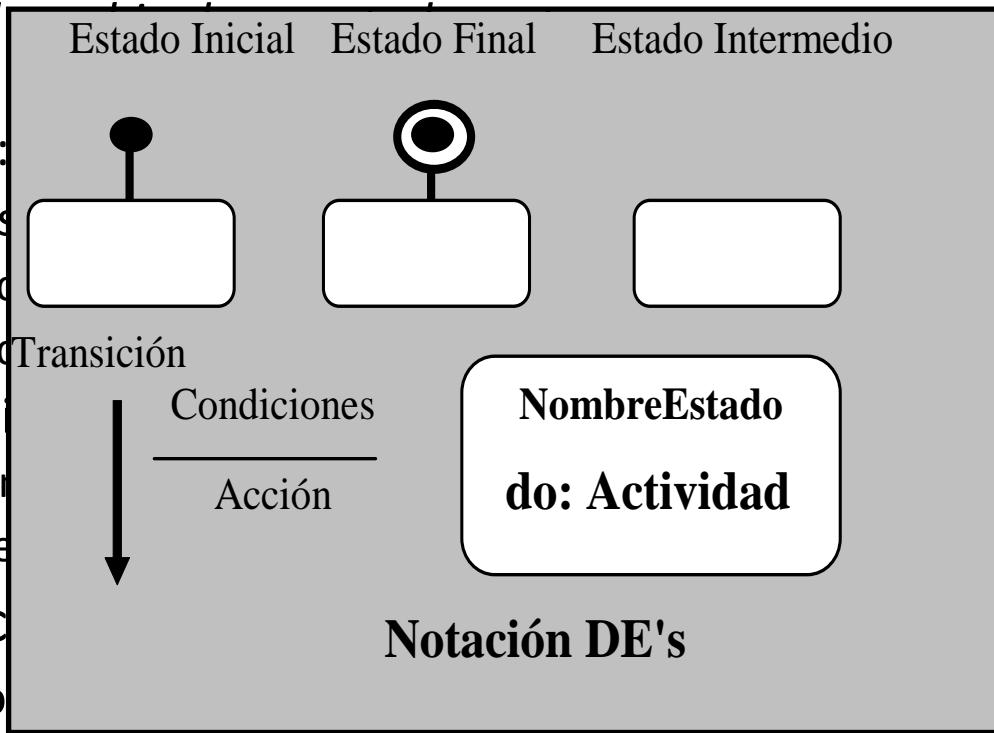
- TABLA DE DECISIÓN: Transformación de señales.

CONDICIONES					
Condición 1	Sí	Sí	No	No	No
Condición 2	Sí	-	-	-	-
Condición 3	-	Sí	-	-	-
Condición 4	-	-	Sí	-	-
Condición 5	-	-	-	Sí	-
Condición 6	-	-	-	-	Sí
ACCIONES	X	X	X	X	X
Acción 1					
Acción 2					
Acción 3					

Especificaciones de Control. CSPEC

- Diagramas de Transición de Estados (DTE)

- *Diagrama que representa los estados que puede tomar un componente o un sistema y que, además, muestra los eventos o circunstancias que implican el cambio entre estos estados.*



IEEE / Std 610

- Elementos:

- Estados:

- Inicio

- Acción

- Transición:

- Condición

- Evento

- Estados compuestos:

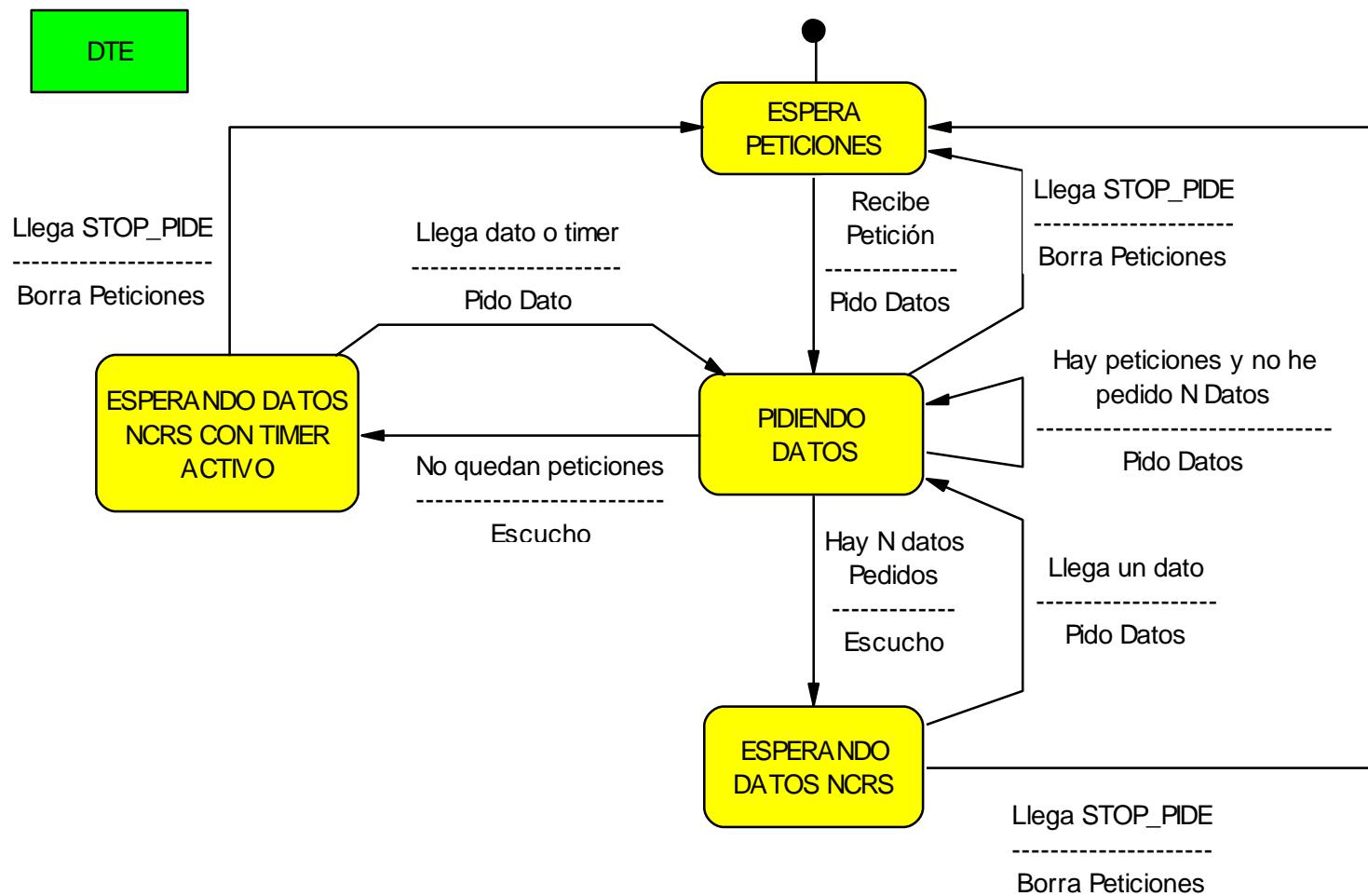
- Componente

- Anidamiento (POO: generalización).

(estados diferidos)

npo)

Especificaciones de Control. CSPEC

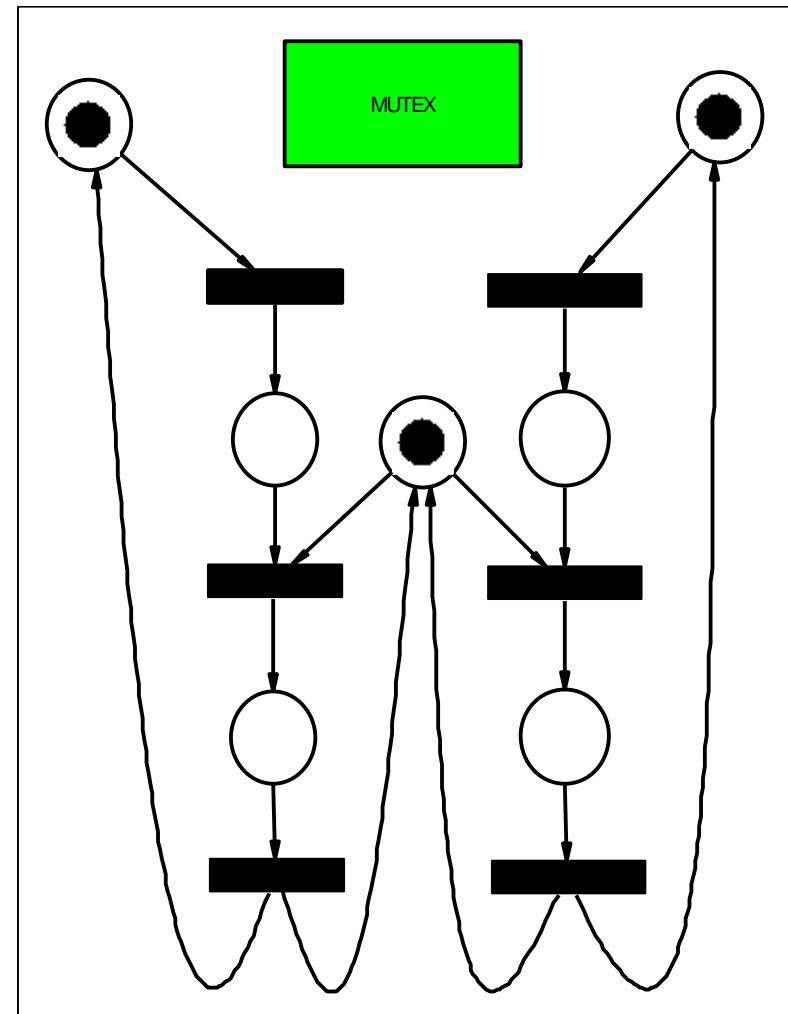


Especificaciones de Control. CSPEC

- Redes de Petri
 - Creadas por C. A. Petri en 1963
 - Técnica muy apropiada para la descripción del control de sistemas de comportamiento asíncrono concurrente
- Elementos
 - Lugares: Representados por círculos
 - Transiciones: representadas por segmentos
 - Conexiones o arcos: flechas uniendo lugares y transiciones o viceversa.
- Estado
 - Depende del marcado. Tokens presentes en cada lugar

Especificaciones de Control. CSPEC

Red de Petri. Estado inicial



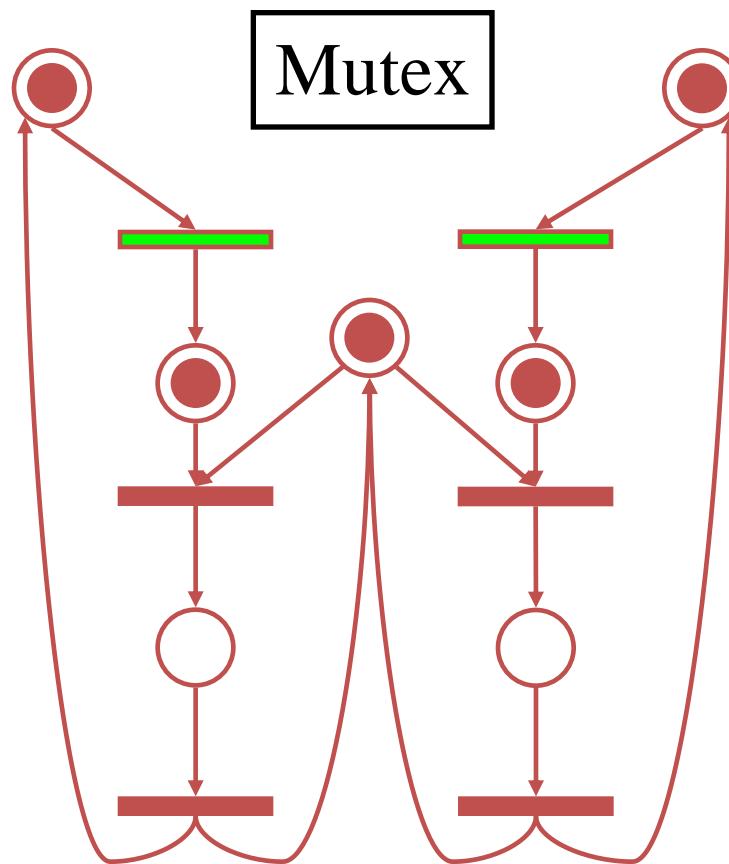
Especificaciones de Control. CSPEC

Red de Petri.

- Reglas de disparo
 - Una transición está habilitada cuando existe un token en cada lugar de entrada de esa transición.
 - Si una transición habilitada se dispara se consumen los tokens en los lugares de entrada y se genera un token en los de salida.

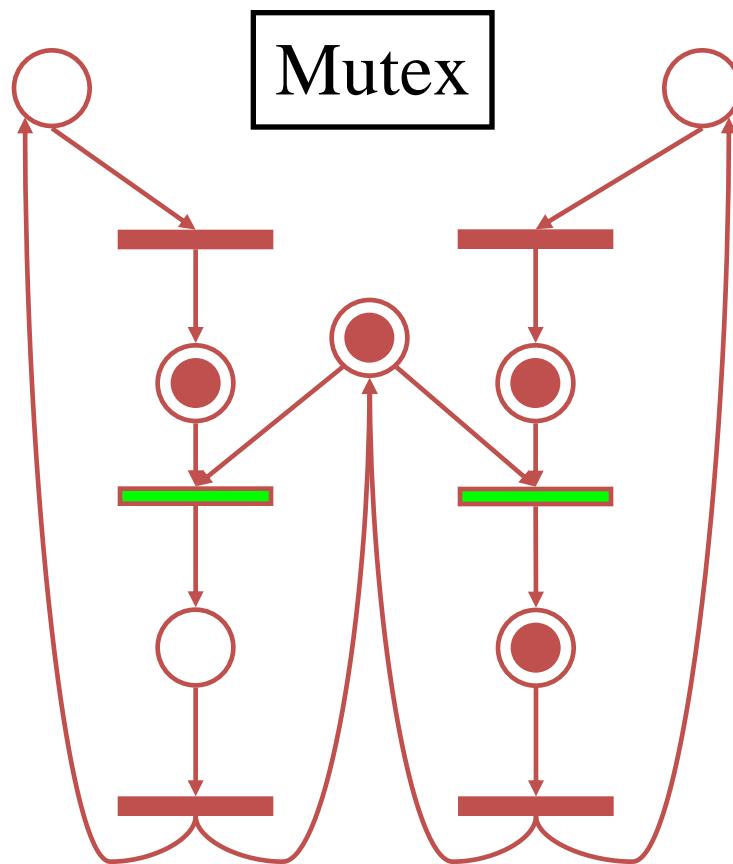
Especificaciones de Control. CSPEC

Red de Petri



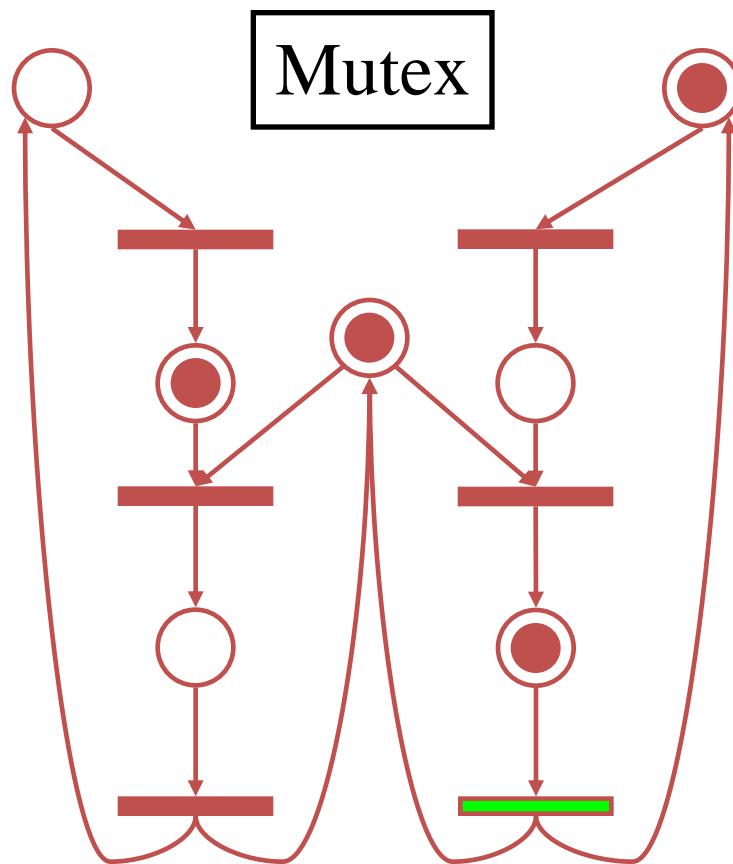
Especificaciones de Control. CSPEC

Red de Petri



Especificaciones de Control. CSPEC

Red de Petri

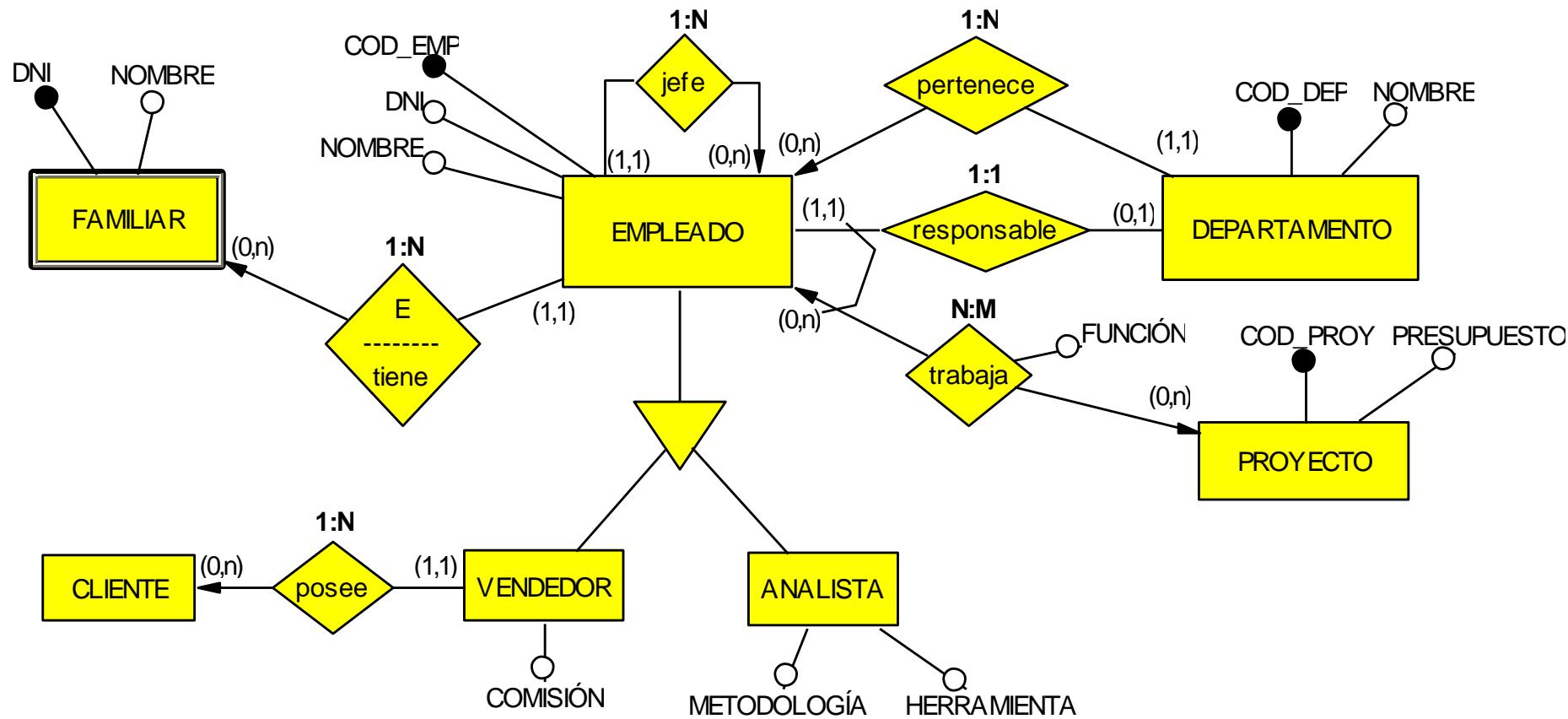


Especificaciones de Control. CSPEC

Red de Petri

- Variantes
 - Valores en arcos
 - Identificación de marcas
 - Tiempos en transición o lugares
- Propiedades
 - Limitación
 - Vivacidad

Diagrama E/R



Comprobaciones

- Complección
 - Los modelos y la especificación es completa
- Integridad
 - No existen contradicciones ni inconsistencias
- Exactitud
 - Mira si los modelos cumplen los requisitos del usuario
- Calidad
 - Estilo, legibilidad y facilidad de mantenimiento

Tema 5.- Análisis estructurado

	PREGUNTA	S	N
C	Todos los componentes tienen nombres		
C	Todos los procesos tienen números		
C	Todos los procesos primitivos tienen una especificación de proceso asociado		
C	Todos los flujos están definidos en el DD		
C	Todos los elementos de datos están definidos		
I	Hay elementos definidos en el DFD no incluidos en el DD		
I	Los almacenes de datos representados en los DFD están definidos en el DD		
I	Los elementos de datos referenciados en las especificaciones de proceso están definidos en el DD		
I	Los flujos de datos de entrada y salida de un proceso primitivo se corresponden con las entradas y salidas de la especificación de proceso		
I	Hay errores de balanceo		
I	Hay procesos que tienen sólo entradas o sólo salidas		
I	Por cada proceso se cumple la regla de conservación de datos		
I	Hay flujos de entrada superfluos a un proceso		
I	Hay flujos de control o flujos de datos como activadores de procesos		
I	Los procesos pueden generar los flujos de salida a partir de los de entrada más una información local al proceso		
I	Hay pérdida de información en los procesos		
I	Hay almacenes sólo con entradas o sólo con salidas		
I	Hay conexiones incorrectas entre los elementos del DFD		
I	Hay almacenes locales		
I	Es correcta la dirección de las flechas de los DFD		
I	Existen redes desconectadas		
E	Cada requisito funcional del usuario tiene asociado uno o más procesos primitivos en los DFD		
CA	El diagrama es claro (posición correcta de las etiquetas, existencia de cruces de línea, etc.)		
CA	Hay nombres de componentes con poca significación		
CA	Hay muchos flujos de entrada y salida (complejidad de interfaz alta) en procesos primitivos		

CONSISTENCIA ENTRE MODELOS

- Técnicas Matriciales

	FUNCIÓN	INFORMACIÓN	TIEMPO
FUNCIÓN			
INFORMACIÓN	Matriz Entidad/Función	Matriz Entidad/Entidad	
TIEMPO		Matriz Evento/Entidad	

- Otras Matrices
 - SSADM (**Structured systems analysis and design method**)
 - Papeles de usuario/Función => Diseño de Interfaces
 - POO
 - Casos de Uso/Clases

CONSISTENCIA ENTRE MODELOS

- Matriz Entidad/Función
 - Filas: Entidades, relaciones, subtipos
 - Columnas: Funciones primitivas o de alto nivel del sistema.
 - Celdas: Acciones de la función en la entidad
 - Insertar I
 - Leer L
 - Modificar o Actualizar M
 - Borrar B

Entidades \ Funciones	Gestionar presupuesto cliente	Gestionar Cliente	...
CLIENTE	L	I, M, B	
PRESUPUESTO	I, M, B		
...			

CONSISTENCIA ENTRE MODELOS

- Matriz Entidad/Entidad
 - Especialmente indicada con un Número alto de entidades.
 - Filas Columnas: Entidades.
 - Celdas: Una X o el nombre de la relación entre entidades

Entidades Entidades	CLIENTE	PRESUPUESTO	...
CLIENTE		Tiene	
PRESUPUESTO			
...			

CONSISTENCIA ENTRE MODELOS

- Matriz Evento/Entidad
 - Filas: Eventos
 - Columnas: Entidades, relaciones, subtipos.
 - Celdas: Alteración causada por el evento en la entidad
 - Insertar I
 - Leer L
 - Modificar o Actualizar M
 - Borrar B

Entidades Eventos	CLIENTE	PRESUPUESTO	...
Datos del Cliente	I, M, B		
Datos del presupuesto	I	I, M, B	
...			

Metodología de análisis estructurado

- Fases
 - Creación del modelo de procesos. DFD y PSPEC.
 - Revisión de documentación
 - Análisis gramatical
 - DFD de contexto y continuar reglas de construcción.
 - Acoplamiento mínimo y máxima cohesión.
 - Evitar detalles de implementación.
 - Según incluyamos elementos en el DFD \Rightarrow DD
 - Evitar ocultar almacenes en las PSPEC. Sólo variables locales.

Metodología de análisis estructurado

- Fases
 - Creación del modelo de procesos. DFD y PSPEC.
 - Creación del modelo de control.
 - No siempre son necesarios. Evitar los detalles de implementación
 - Seguiremos las reglas generales de construcción de DFC, CSPEC.
 - Cada par DFD/DFC contiene los mismos elementos y en la misma posición. Se define una CSPEC para un par DFD/DFC.
 - Las CSPEC pueden ser combinacional, secuencial o compuestas. Elegiremos siempre la más sencilla.
 - En los DE tener muy presente todas las posibles transiciones, en particular, las producidas por comportamientos incorrectas del sistema o el usuario.

Metodología de análisis estructurado

- Fases
 - Creación del modelo de procesos. DFD y PSPEC.
 - Creación del modelo de control.
 - Creación del modelo de datos
 - Si los datos manejados por el sistema son complejos los modelaremos en un DER.

Metodología de análisis estructurado

- Fases
 - Creación del modelo de procesos. DFD y PSPEC.
 - Creación del modelo de control.
 - Creación del modelo de datos
 - Consistencia entre los modelos
 - Aplicar todas las estrategias analizadas que sean necesarias.

Modelos del sistema: esencial y de implementación

- Modelo Esencial o modelo lógico
 - Modelo abstracto para el que disponemos de tecnología perfecta.
- Errores típicos suponen incluir en éste:
 - Secuenciación de procesos
 - Utilizar ficheros temporales o de backup
 - Utilizar información redundante

Modelos del sistema: esencial y de implementación

- **Modelo de implementación**
 - Elección de dispositivos de entrada-salida
 - Elección de los dispositivos de almacenamiento
 - Formato de entradas y salidas (definición de interfaces)
 - Secuencia de las operaciones de Entrada salida
 - Volumen de datos
 - Tiempo de respuesta
 - Copias de seguridad y descarga de datos del sistema
 - Seguridad.

Temario

- 1. Introducción a la Ingeniería del Software**
- 2. Ingeniería del software: Los procesos**
- 3. Ingeniería del software: Los ciclos de vida**
- 4. Análisis de requisitos**
- 5. Modelado del Análisis Estructurado**
- 6. Pruebas del software**

Tema 6. Pruebas del software

6.1.- Introducción.

6.2.- Pruebas Estructurales

6.3.- Prueba Funcional

6.4.- Enfoque práctico recomendado para el diseño de casos

6.5.- Documentación del diseño de las pruebas

6.6.- Ejecución de las pruebas

6.7.- Estrategia de aplicación de las pruebas

6.8.- Pruebas en desarrollos orientados a objetos

INTRODUCCIÓN

- **VERIFICACIÓN:**
 - El proceso de evaluación de un sistema o de uno de sus componentes para determinar si los productos de una fase dada satisfacen las condiciones impuestas al principio de dicha fase (IEEE)
 - ¿Estamos construyendo correctamente el producto? (Boehm)
- **VALIDACIÓN**
 - El proceso de evaluación del sistema o de uno de sus componentes durante o al final del desarrollo para determinar si satisface los requisitos especificados. (IEEE)
 - ¿Estamos construyendo el producto correcto? (Boehm)

Filosofía de las pruebas

- Es un proceso muy difícil
 - No es físico, no hay leyes de comportamiento, es complejo
 - La prueba exhaustiva es imposible.
 - Prejuicios y poco tiempo
- Cambio de mentalidad:
 - Desarrollo de software (Constructivo)
 - Pruebas para “demoler” lo construido (Destructivo)
 - No son destructivas en el sentido estricto del vocablo
 - No hay culpabilidad en el fallo
 - Prueba clínica
 - Una prueba tiene éxito si descubre un error nuevo

OBJETIVO

- **Objetivo**
 - Proyecto. Detectar errores en la menor cantidad de tiempo y con el menor número de recursos posibles.
 - Proceso. Diseño de técnicas que permitan un desarrollo sistemático de pruebas que garanticen dicho objetivo.
- **Ventajas Secundarias**
 - Demuestra hasta qué punto se verifican los requisitos
 - Se generan datos de prueba que informan sobre la fiabilidad
- **No aseguran la ausencia de defectos.**

DEFINICIONES

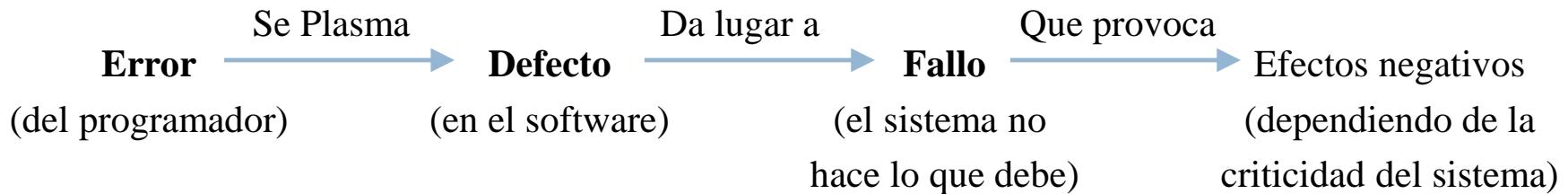
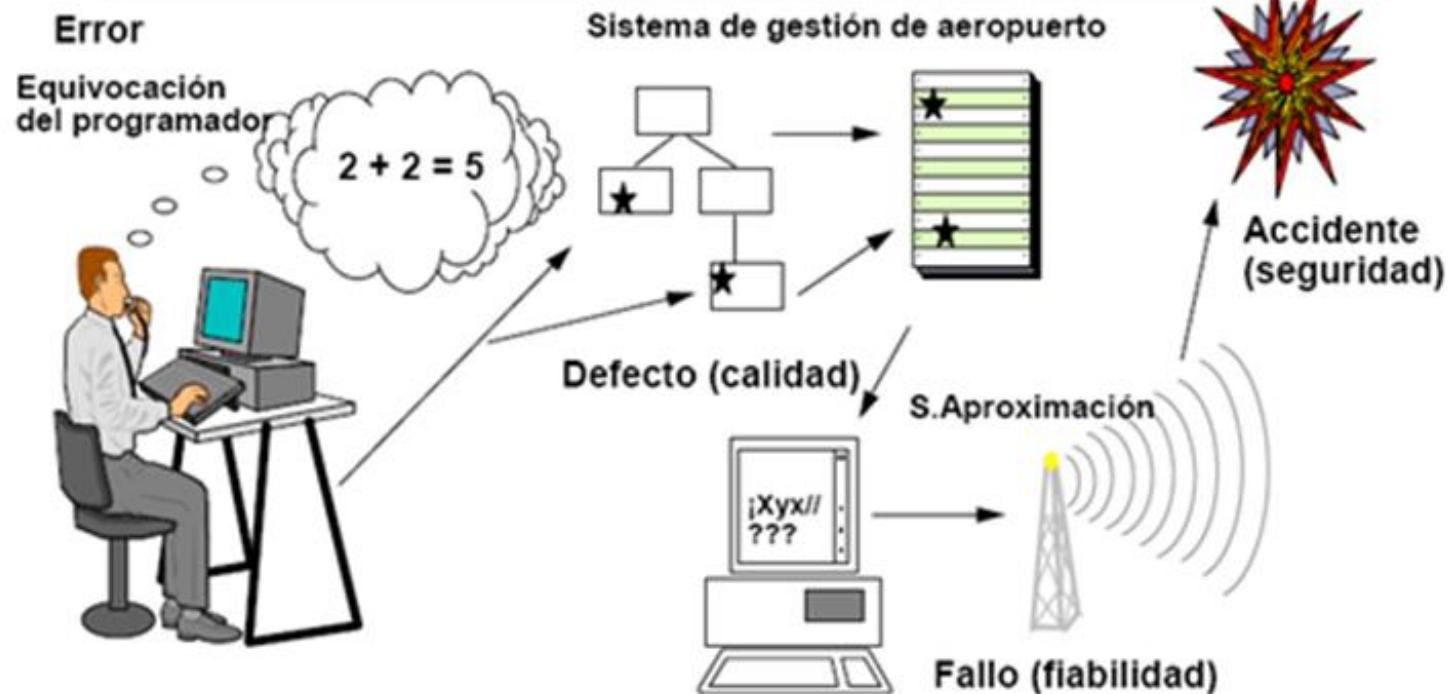
- **Pruebas:** Una actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan y registran se realiza una evaluación de algún aspecto.
 - Probar es el proceso de ejecutar un programa con el fin de encontrar errores. (MYERS)
- **Caso de prueba:** Un conjunto de entradas, condiciones de ejecución, y resultados desarrollados para un objetivo particular como, por ejemplo, ejercitar un camino concreto de un programa o verificar el cumplimiento de un requisito.
 - Un buen caso de prueba es aquel que tiene una alta probabilidad de detectar un error.
 - **1 Prueba ⇒ N Casos_de_Prueba**

DEFINICIONES

- **Fallo:** La incapacidad de un sistema para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados.
- **Defecto:** Es una incorrección en el software que genera un fallo, como por ejemplo, un proceso, una definición de datos o un paso de procesamiento incorrectos en un programa.
- **Error:** Varias acepciones
 1. Diferencia entre un valor calculado, observado o medido y el verdadero, especificado o teóricamente correcto.
 2. Un resultado incorrecto del software
 3. Un defecto en el software
 4. Una acción humana que conduce a un resultado incorrecto.

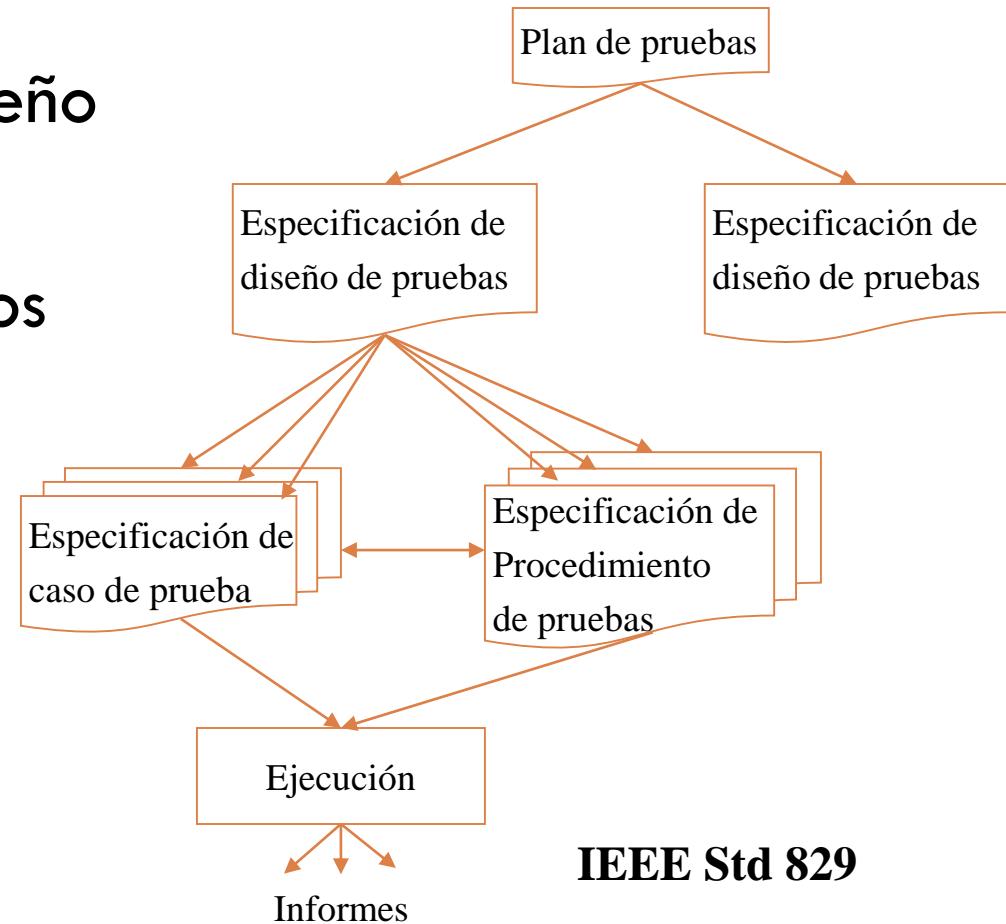
DEFINICIONES

RELACION ENTRE ERROR, DEFECTO Y FALLO



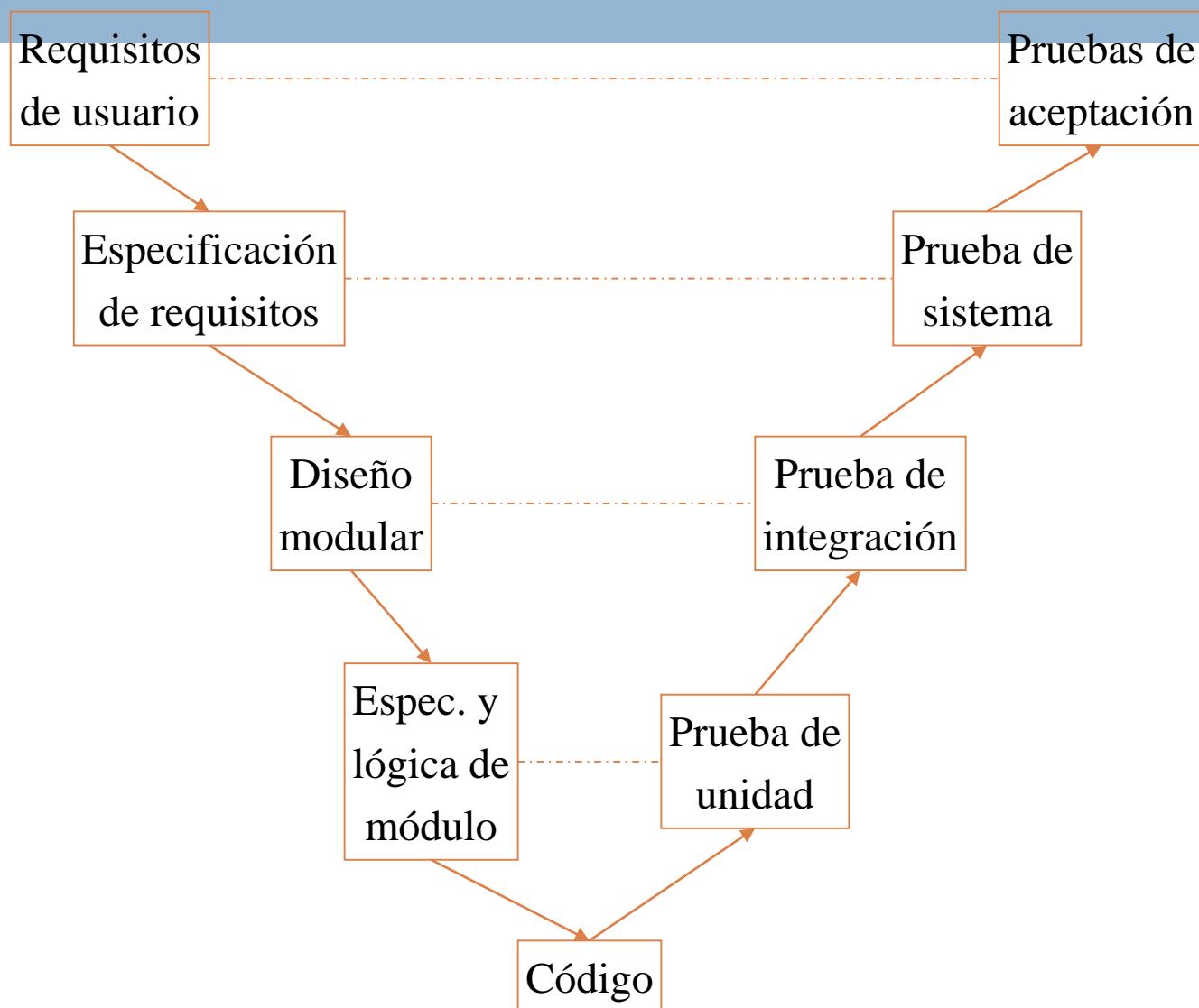
Documentación de diseño de pruebas

- Plan de pruebas
- Especificación del diseño de prueba
- Especificación de casos de prueba
- Especificación de procedimiento de pruebas
- Ejecución.

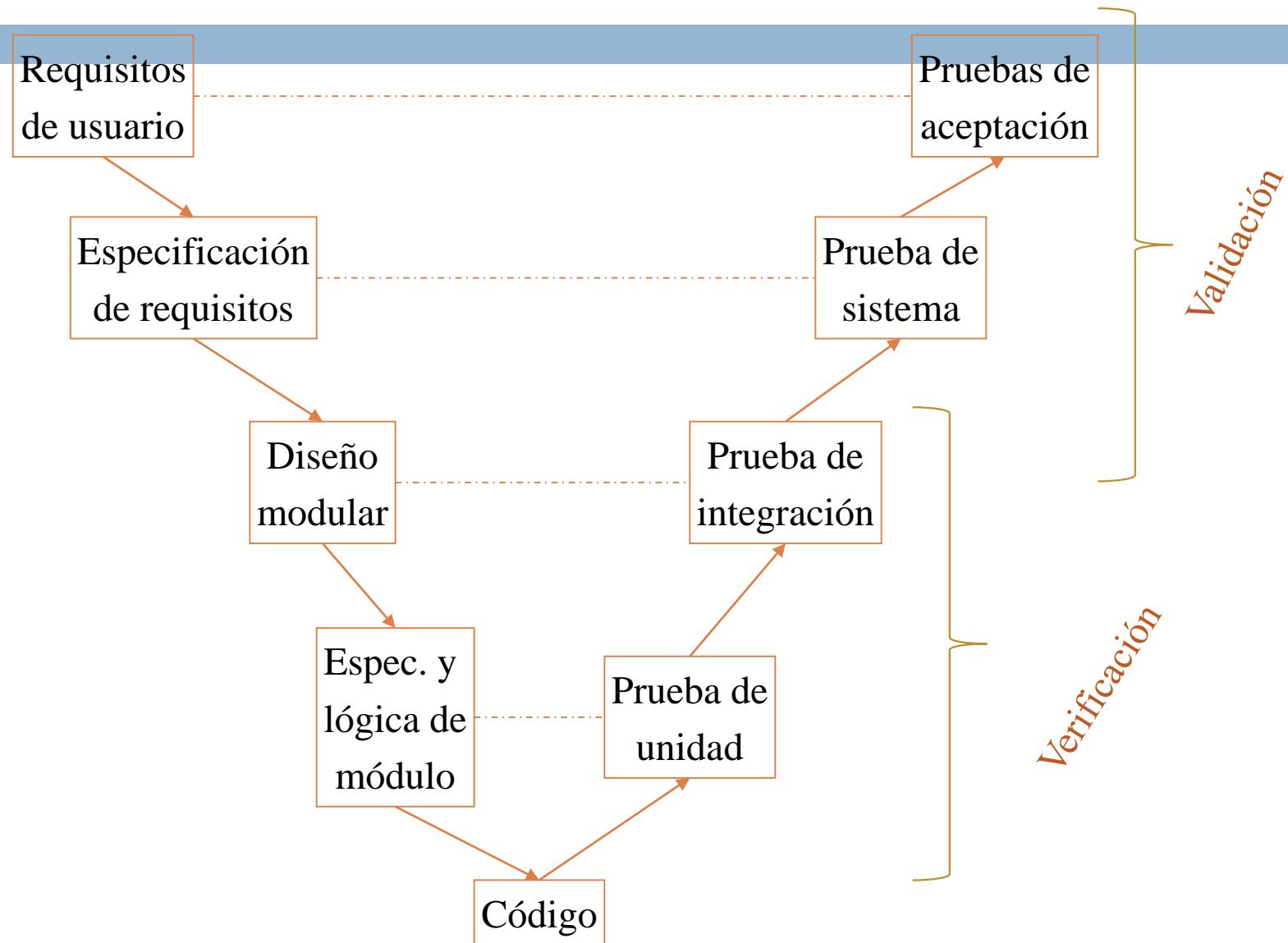


IEEE Std 829

Estrategia de aplicación de pruebas



Estrategia de aplicación de pruebas



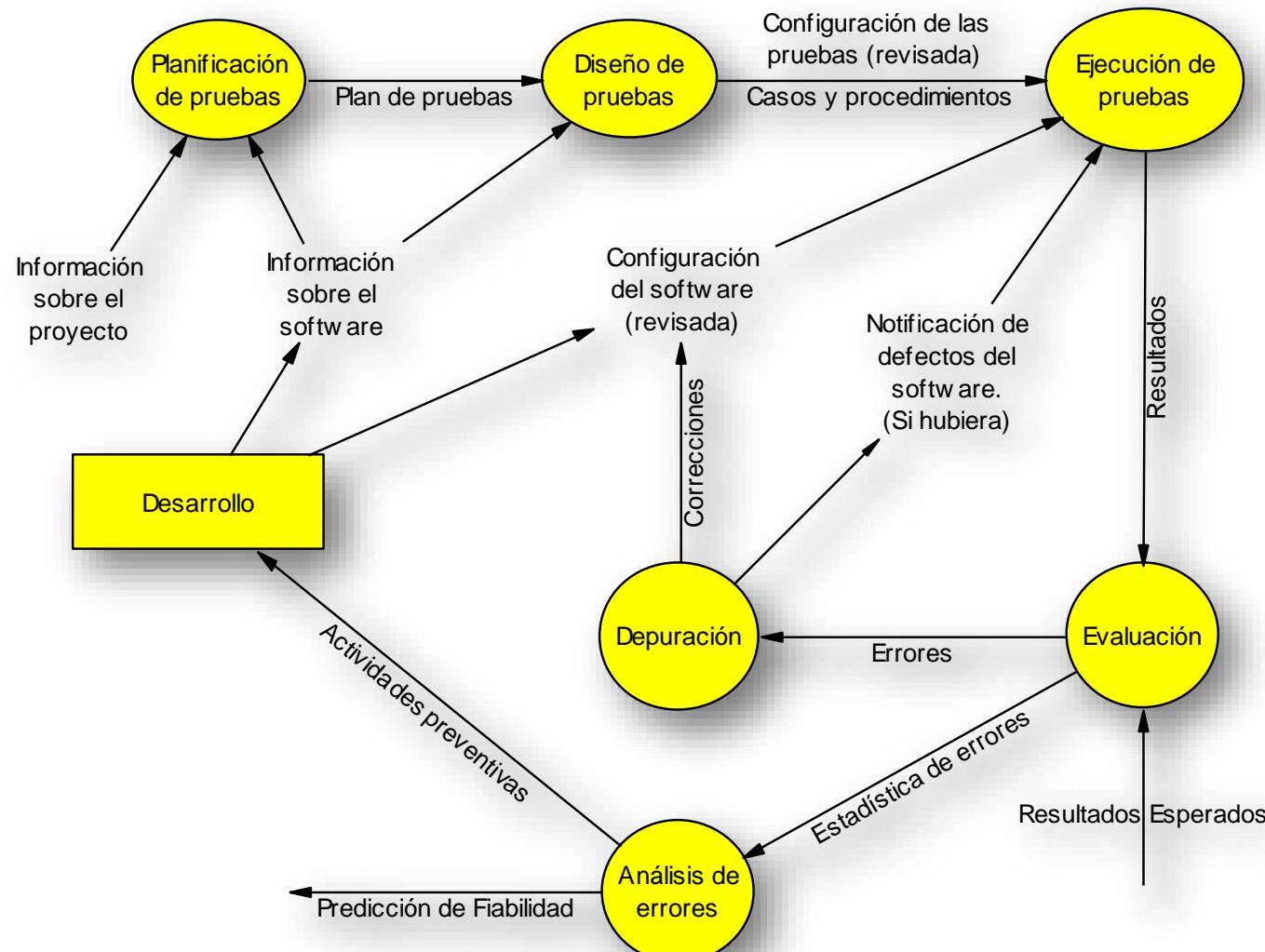
PRINCIPIOS

1. A todas las pruebas se les debería poder hacer un seguimiento hasta los requisitos del cliente
2. Las pruebas deberían planificarse mucho antes de que empiecen.
3. El 80% de los errores surgen al hacer un seguimiento del 20% de los módulos del software (Principio de Pareto)
4. Las pruebas tendrían que hacerse de lo pequeño hacia lo grande
5. No son posibles pruebas exhaustivas
6. Las pruebas, para ser más efectivas, deberían de ser realizadas por un equipo independiente del equipo de desarrollo del software

PRINCIPIOS

1. Cada caso de prueba debe definir el resultado de salida esperado.
2. Se debe inspeccionar a conciencia el resultado de cada prueba para poder descubrir síntomas de defectos
3. Al generar casos de prueba se deben incluir datos válidos y no válidos
4. Las pruebas deben tener dos objetivos
 1. Probar si el software no hace lo que debe
 2. Probar si el software hace lo que no debe
5. Se debe evitar los casos desecharables (No documentados)
6. No se deben hacer planes asumiendo que no habrá fallos.
7. Las pruebas son una tarea creativa.
8. Añade además el Principio de Pareto e independencia del equipo de pruebas (3 y 6 de Davis)

El Proceso de Prueba



TÉCNICAS DE DISEÑO DE PRUEBAS

ES IMPOSIBLE LA PRUEBA EXHAUSTIVA

- Equilibrio entre confianza en el software y recursos consumidos
- Construir los mejores casos de prueba posibles.
 - ¿Cómo puede fallar el software?.
 - Casos no redundantes.
 - Ni demasiado sencillo ni demasiado complejo.

MÉTODOS DE PRUEBA

- PRUEBAS DE CAJA NEGRA.
 - No nos importa como está implementada la aplicación
 - Conocemos exactamente que función debe realizar
 - Pruebas que demuestren que cada función es completamente operativa
 - Se llevarán a cabo pruebas sobre la(s) interface(s)
- PRUEBAS DE CAJA BLANCA
 - Conocemos la estructura interna del producto.
 - Pruebas que aseguren que todos los módulos o componentes internos funcionan bien y encajan correctamente
 - Minucioso examen de los detalles procedimentales.
 - Probaremos los caminos lógicos del software, con casos que ejerciten conjuntos específicos de condiciones y/o bucles.

MÉTODOS DE PRUEBA

- Prueba exhaustiva de **Caja Negra** impracticable. (Ej. Piattini)
 - Programa que sume 2 números de 2 cifras.
 - 10.000 posibles combinaciones
 - Faltan posibles errores.
 - Números negativos
 - Números mayores que 100
 - Introducir letras
 - ...
- Prueba exhaustiva de **Caja Blanca** impracticable. (Ej. Pressman)
 - Programa con dos bucles anidados de 0 a 20 en función de las entradas y 4 if-then-else en el bucle interior
 - Hay $10^{14} \Rightarrow 1$ prueba por 1 ms. $\Rightarrow 3170$ años.
- Los enfoques no son excluyentes.
 - SON COMPLEMENTARIOS.

Tema 6. Pruebas del software

6.1.- Introducción.

6.2.- Pruebas Estructurales

6.3.- Prueba Funcional

6.4.- Enfoque práctico recomendado para el diseño
de casos

6.5.- Documentación del diseño de las pruebas

6.6.- Ejecución de las pruebas

6.7.- Estrategia de aplicación de las pruebas

6.8.- Pruebas en desarrollos orientados a objetos

Pruebas Estructurales. Caja Blanca

- ¿Por qué hacer pruebas de caja blanca y no solo de caja negra?
 - Los errores lógicos y las suposiciones incorrectas son inversamente proporcionales a la probabilidad de que se ejecute un camino del programa
 - Los errores tipográficos son aleatorios
 - Se suele creer que un determinado flujo es poco probable cuando, de hecho, puede ejecutarse regularmente

Pruebas Estructurales. Caja Blanca

- Se trata de elegir casos de prueba que ofrezcan una seguridad aceptable de descubrir defectos.
 - Caso de prueba ≡ Camino lógico.
 - CRITERIOS DE COBERTURA LÓGICA
 - Aunque no son imprescindibles se suele utilizar el método gráfico denominado Grafo de flujo.



Construcción de grafos

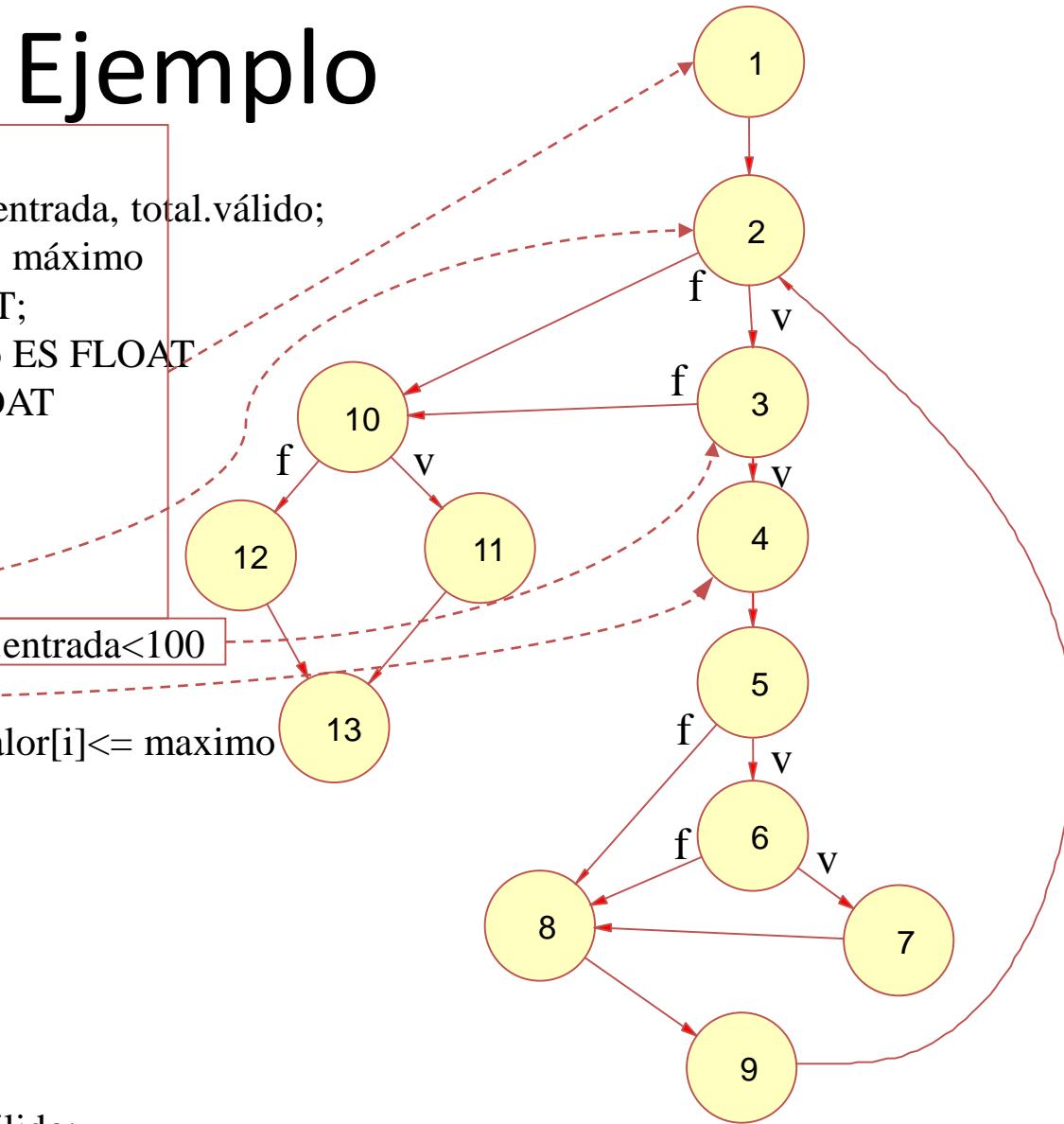
- Señalar sobre el código la condición de cada decisión.
 - IF, SWITCH, DO, WHILE
- Agrupar el resto de las sentencias situadas entre cada dos condiciones según los siguientes esquemas
- Numerar cada nodo del grafo e identificar los nodos condición con una letra indicando en cada arista si ésta se debe a que la condición es verdadera o falsa.

Ejemplo

```

PROCEDURE media;
INTERFACE DEVUELVE media, total.entrada, total.válido;
INTERFACE ACEPTE valor, mínimo, máximo
TYPE valor[1:100] ES ARRAY FLOAT;
TYPE media, total.entrada, total.válido ES FLOAT
TYPE mínimo, máximo, suma ES FLOAT
TYPE i ES ENTERO
i=1;
total.entrada=total.válido=0;
suma=0;
DO WHILE valor[i]<>-999 AND total.entrada<100
    total.entrada+=1;
    IF valor[i]>=mínimo and valor[i]<= maximo
        THEN total.válido+=1;
        suma+=valor[i];
    ELSE ignorar
    ENDIF
    i+=1;
END DO
IF total.válido >0
    THEN media=suma/total.válido;
    ELSE media=-999;
ENDIF

```

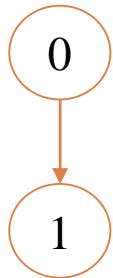


Construcción de grafos

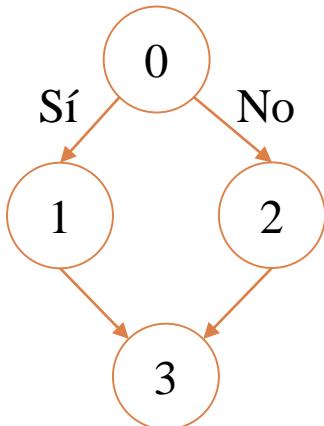
- Señalar sobre el código la condición de cada decisión.
 - IF, SWITCH, DO, WHILE
- Agrupar el resto de las sentencias situadas entre cada dos condiciones según los siguientes esquemas
- Numerar cada nodo del grafo e identificar los nodos condición con una letra indicando en cada arista si ésta se debe a que la condición es verdadera o falsa.

Construcción de grafos

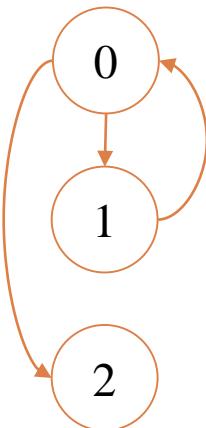
Secuencia



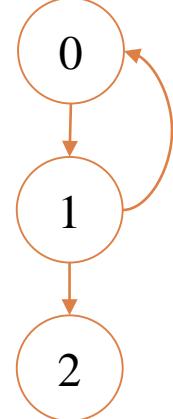
Salto condicional



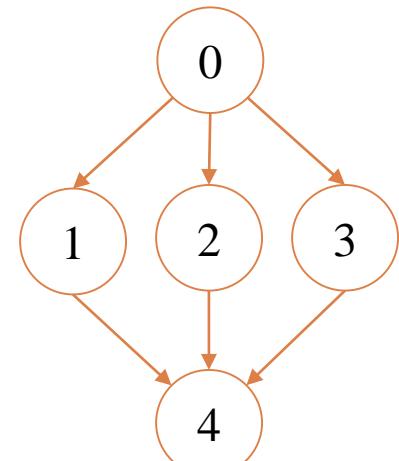
Mientras que



Repetir hasta que



Según sea
(Switch Case)



Construcción de grafos

- Señalar sobre el código la condición de cada decisión.
 - IF, SWITCH, DO, WHILE
- Agrupar el resto de las sentencias situadas entre cada dos condiciones según los siguientes esquemas
- Numerar cada nodo del grafo e identificar los nodos condición con una letra indicando en cada arista si ésta se debe a que la condición es verdadera o falsa.

T.7.- Prueba del software

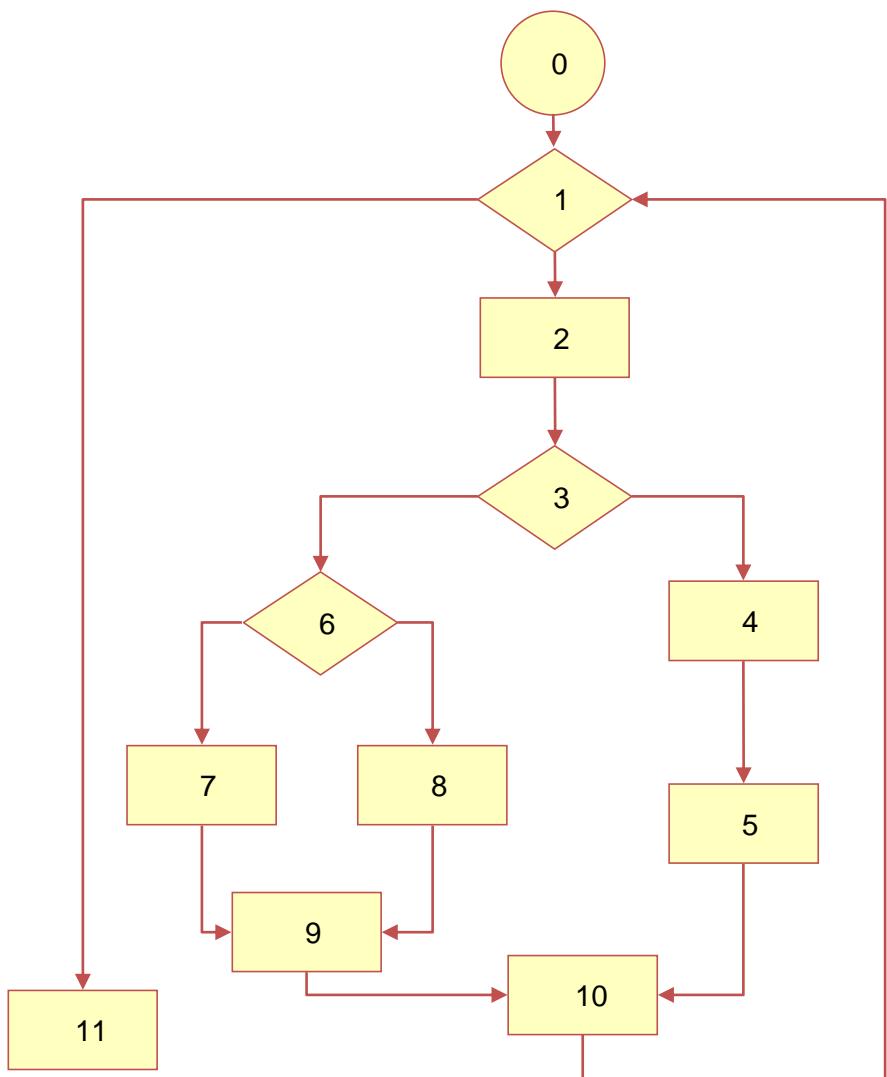
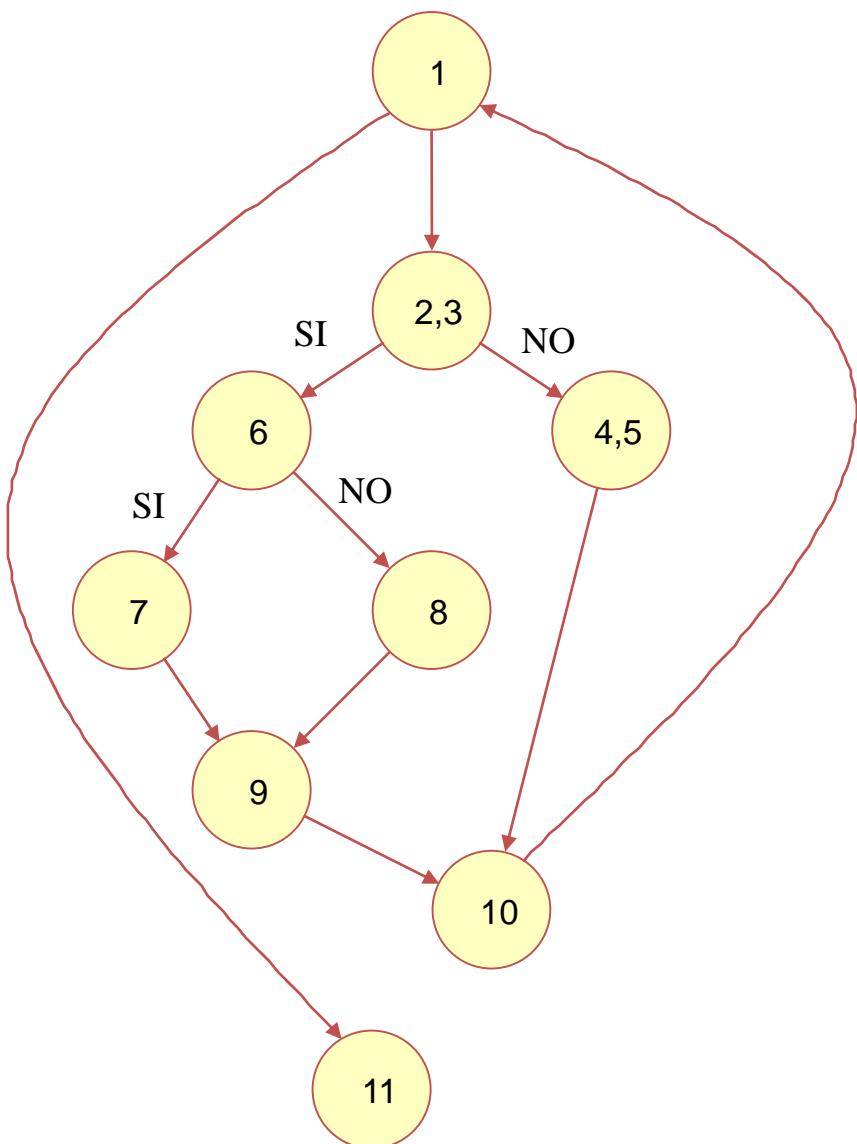


Diagrama de flujo



Grafo de flujo

Criterios de cobertura lógica

- Cobertura de Sentencias:** Se trata de generar CP para que cada sentencia se ejecute una vez
- Cobertura de Decisiones:** Existen suficientes CP como para que cada decisión tenga, al menos una vez, un resultado verdadero y otro falso. En general, garantiza la cobertura de Sentencia
- Cobertura de Condición:** Cada condición adopta al menos una vez un resultado verdadero y otro falso. No garantiza la cobertura de decisión
- Cobertura de Decisión/Condición:** Consiste en exigir los dos criterios anteriores simultáneamente.
- Criterio de Condición múltiple:** Descompone cada decisión múltiple en una secuencia de condiciones unicondicionales y luego se exige que cada combinación posible de resultados en cada condición se ejecute al menos una vez.

Descomposición de condiciones compuestas

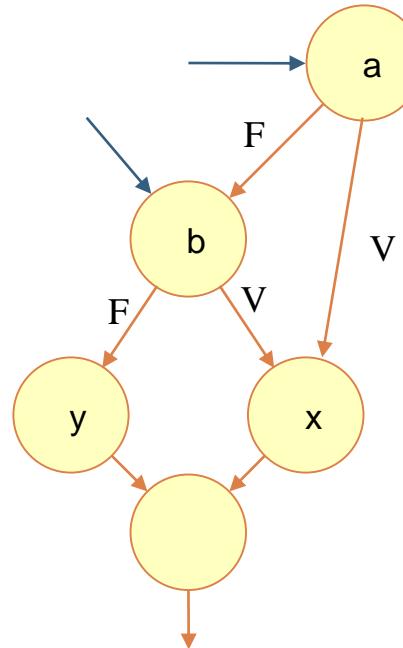
Condiciones Compuestas

IF a OR b

Then Procedimiento X

Else Procedimiento Y

Endif



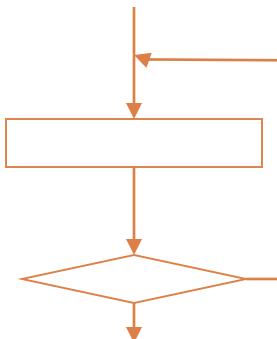
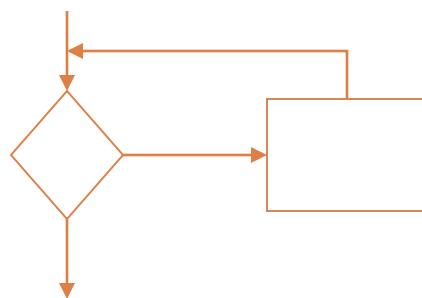
- **Nodo Predicado:** Contiene una condición y se caracteriza porque dos o más aristas parten de él.

Criterios de cobertura lógica

- **Cobertura de caminos:** Es el criterio de cobertura más riguroso. Exige que cada camino del programa se ejecute al menos una vez.
- Problema: Los bucles.
 - Los bucles generan el mayor número de problemas con la cobertura de caminos. Bucles anidados o con condiciones que varían el número de repeticiones.
 - **Camino de prueba:** Un camino que atraviesa, como máximo, una vez el interior de cada bucle que se encuentra.
 - Camino de prueba insuficiente. Los bucles se deben recorrer 0, 1, 2 veces. [HUANG]

Tratamiento de Bucles

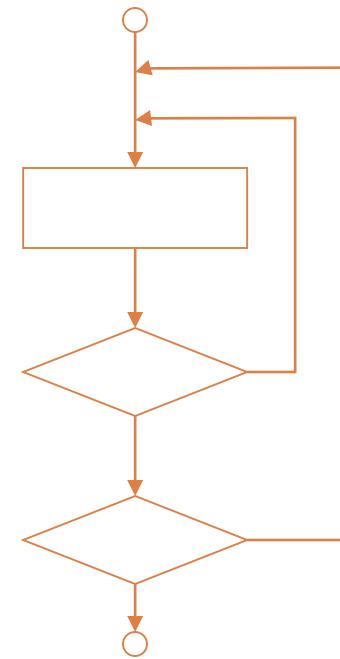
- **Bucles simples:** n es el número máximo de pasos permitidos para el bucle
 - 1. Pasar totalmente por alto el bucle
 - 2. Pasar una sola vez por el bucle
 - 3. Pasar dos veces por el bucle
 - 4. Hacer m pasos con $m < n$
 - 5. Hacer $n-1$, n y $n+1$ pasos por el bucle.



Tratamiento de Bucles

- **Bucles Anidados:** No es posible extender la prueba del bucle simple

1. Comenzar por el bucle más interior con los otros en sus valores mínimos
2. Llevar a cabo la prueba de bucle simple al más interior
3. Progresar hacia fuera llevando a cabo pruebas para el siguiente bucle y manteniendo los exteriores en sus valores mínimos y los interiores en sus valores típicos.
4. Continuar hasta probar todos los bucles.



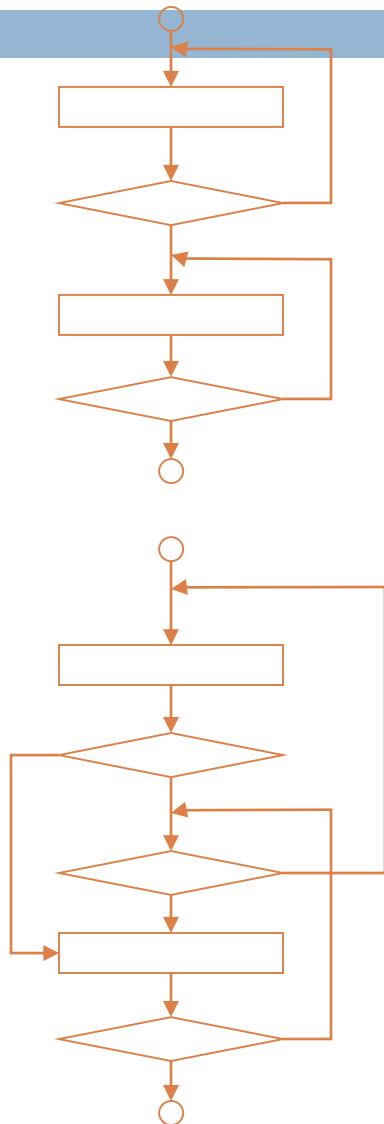
Tratamiento de Bucles

□ **Bucles Concatenados:**

- Como los bucles simples si son independientes
- Como los bucles anidados si los bucles no son independientes, por ejemplo, los valores del bucle 1 se usan como iniciales para el bucle 2.

□ **Bucles no estructurados**

- Siempre que sea posible deben reconstruirse como estructurados.



Complejidad Ciclomática de McCabe

- Dado un grafo cuantos caminos son precisos para probarlo.
 - McCabe propone una métrica que indica el número de caminos independientes que existen en un grafo.
 - Esta métrica nos proporciona una medida cuantitativa de la complejidad lógica del módulo software que estamos probando
 - Propone como un buen criterio de prueba la ejecución de un conjunto de caminos independientes cuyo número indica la métrica.
 - Nos permitirá establecer un conjunto básico de caminos de ejecución que aseguren que todo el código se ejecuta, al menos una vez
 - Este criterio se propone como equivalente a la cobertura de decisión

Complejidad Ciclomática de McCabe

□ Cálculo de la complejidad ciclomática

- $V(G) = a - n + 2$, siendo a el número de arcos o aristas del grafo y n el número de nodos
- $V(G) = r$, siendo r el número de regiones cerradas del grafo
- $V(G) = c + 1$, siendo c el número de nodos de condición. (una condición de n arcos de salida se contabiliza como $n-1$)

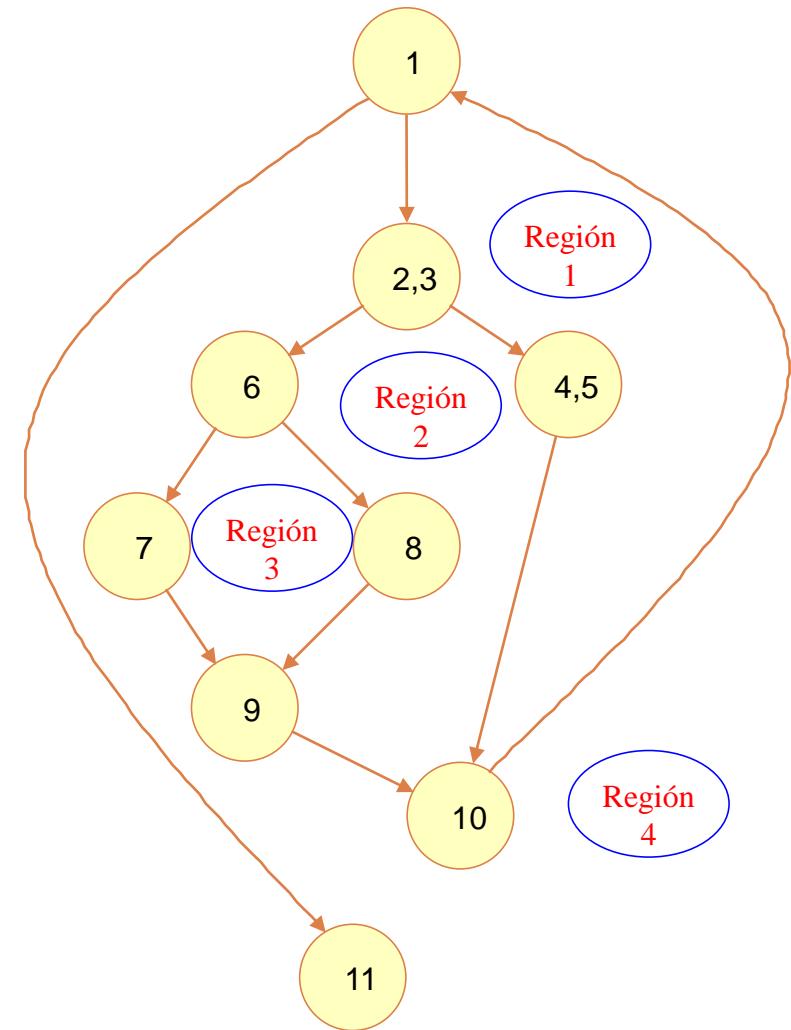
Complejidad Ciclomática de McCabe

$$V(G) = A - N + 2 = 11 - 9 + 2 = 4$$

$$V(G) = R = 4$$

$$V(G) = P + 1 = 3 + 1 = 4$$

En el número de regiones hay que tener presente que las fórmulas de McCabe sólo pueden aplicarse a grafos fuertemente conexos para los cuales siempre existe un camino entre cualesquiera dos nodos que elijamos. Esto no se verifica en los programas que tienen un nodo de inicio y otro de final por lo que para calcular las regiones debemos unir estos nodos o como alternativa contabilizar la región externa.



Complejidad Ciclomática de McCabe

- El criterio de prueba de McCabe implica elegir tantos caminos de un grafo como caminos independientes haya.
- $V(G)$ constituye un límite superior que asegura la cobertura de sentencia y sería equivalente a la cobertura de decisiones.
- Cuando $V(G)$ es mayor que 10 la probabilidad de defectos en el módulo es muy alta si no se debe a sentencias case-of

Complejidad Ciclomática de McCabe

- Método del camino básico facilita la selección de los caminos independientes presentes en un grafo.
 - Selección de un camino de prueba típico o básico
 - Crear variaciones sobre este camino de forma que cada variación se distinga en al menos una arista de las demás.
 - Conviene tener presente que algunos caminos no se pueden ejecutar solos y necesitan de la ejecución de algún otro
- Seleccionados los caminos debemos analizar el código para determinar las entradas que los fuerzan
 - Es posible que estas no existan encontrándonos ante un camino imposible que debe ser sustituido por otro que también permita satisfacer el criterio de McCabe
- A partir de las entradas debemos revisar la especificación para predecir las salidas.

Ejemplo

CAMINOS INDEPENDIENTES

Camino 1: 1-11

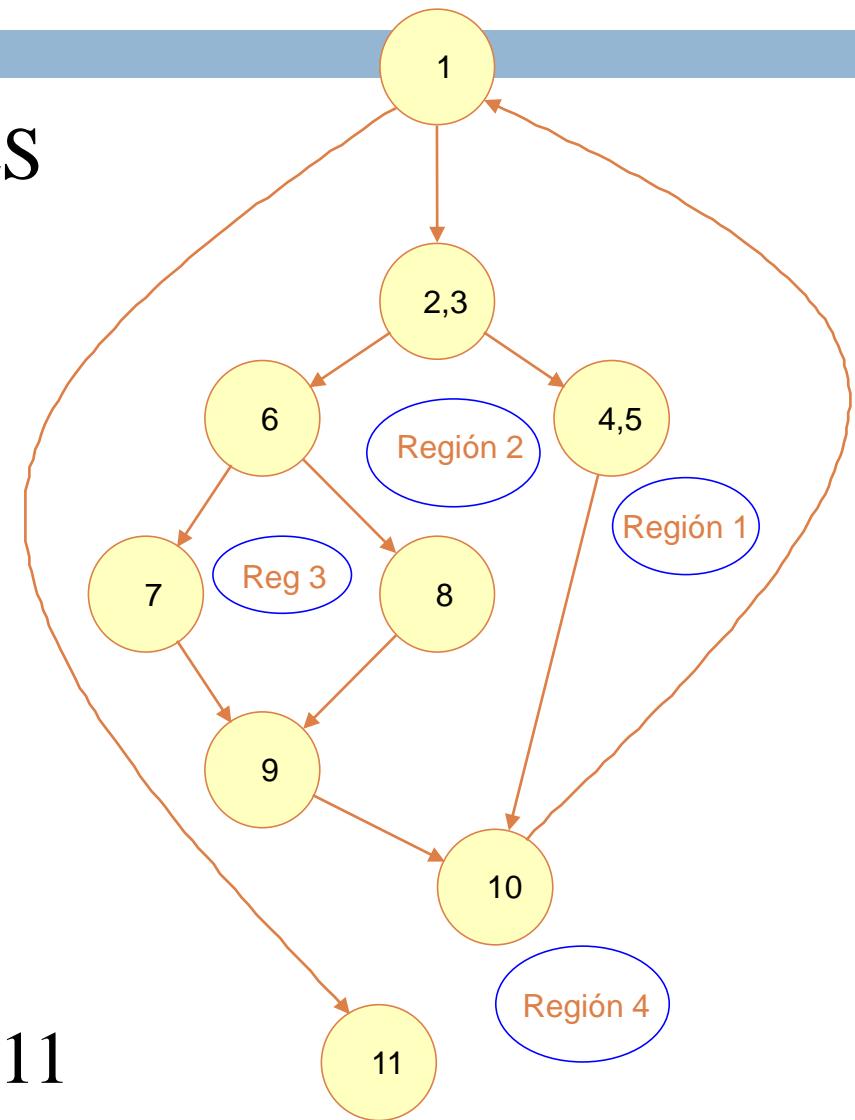
Camino 2: 1-2-3-4-5-10-1-11

Camino 3: 1-2-3-6-8-9-10-1-11

Camino 4: 1-2-3-6-7-9-10-1-11

OTROS CAMINOS:

1-2-3-4-5-10-1-2-3-6-8-9-10-1-11



Ejemplo completo

```
PROCEDURE media;
INTERFACE DEVUELVE media, total.entrada, total.válido;
INTERFACE ACEPTE valor, mínimo, máximo
TYPE valor[1:100] ES ARRAY FLOAT;
TYPE media, total.entrada, total.válido ES FLOAT
TYPE mínimo, máximo, suma ES FLOAT
TYPE i ES ENTERO
i=1;
total.entrada=total.válido=0;
suma=0;
DO WHILE valor[i]<>-999 AND total.entrada<100
    total.entrada+=1;
    IF valor[i]>=mínimo and valor[i]<= máximo
        THEN total.válido+=1;
            suma+=valor[i];
        ELSE ignorar
    ENDIF
    i+=1;
END DO
IF total.válido >0
    THEN media=suma/total.válido;
    ELSE media=-999;
ENDIF
```

Ejemplo completo

```
PROCEDURE media;
INTERFACE DEVUELVE media, total.entrada, total.válido;
INTERFACE ACEPTE valor, mínimo, máximo
TYPE valor[1:100] ES ARRAY FLOAT;
TYPE media, total.entrada, total.válido ES FLOAT
TYPE mínimo, máximo, suma ES FLOAT
TYPE i ES ENTERO
i=1;
total.entrada=total.válido=0;
suma=0;1
DO WHILE valor[i]<>-999 2 AND total.entrada<100 3
    total.entrada+=1; 4
    IF valor[i]>=mínimo 5 and valor[i]<= maximo 6
        THEN total.válido+=1;
            suma+=valor[i]; 7
        ELSE ignorar
    ENDIF
    i+=1; 8
END DO 9
IF total.válido >0 10
    THEN media=suma/total.válido; 11
    ELSE media=-999; 12
ENDIF 13
```

Ejemplo completo

```

PROCEDURE media;
INTERFACE DEVUELVE media, total.entrada, total.válido;
INTERFACE ACEPTE valor, mínimo, máximo
TYPE valor[1:100] ES ARRAY FLOAT;
TYPE media, total.entrada, total.válido ES FLOAT
TYPE mínimo, máximo, suma ES FLOAT
TYPE i ES ENTERO
i=1;
total.entrada=total.válido=0;
suma=0;1

```

```

DO WHILE valor[i]<>-999 2 AND total.entrada<100 3
    total.entrada+=1; 4
    IF valor[i]>=mínimo 5 and valor[i]<= maximo 6
        THEN total.válido+=1;
            suma+=valor[i]; 7
        ELSE ignorar
    ENDIF
    i+=1; 8

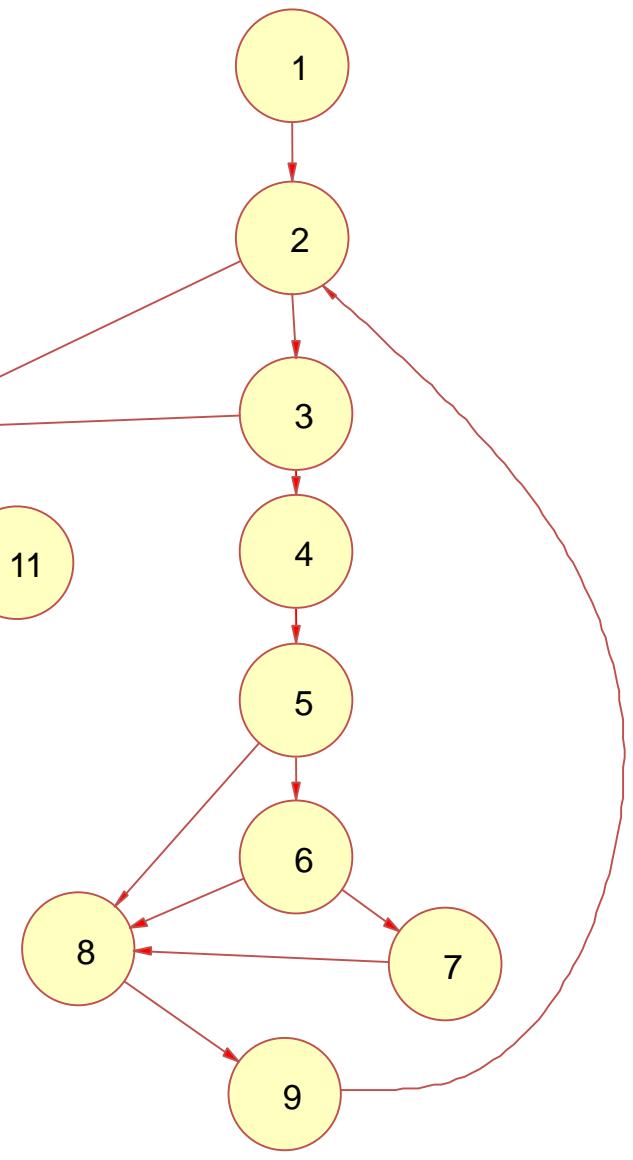
```

```
END DO 9
```

```

IF total.válido >0 10
    THEN media=suma/total.válido; 11
    ELSE media=-999; 12
ENDIF 13

```



Ejemplo completo

COMPLEJIDAD CICLOMÁTICA:

$$V(G) = a - n + 2 = 17 - 13 + 2 = 6$$

$$V(G) = r = 6$$

$$V(G) = p + 1 = 5 + 1 = 6$$

CONJUNTO BÁSICO DE CAMINOS:

1: 1-2-10-11-13

2: 1-2-10-12-13

3: 1-2-3-10-11-13

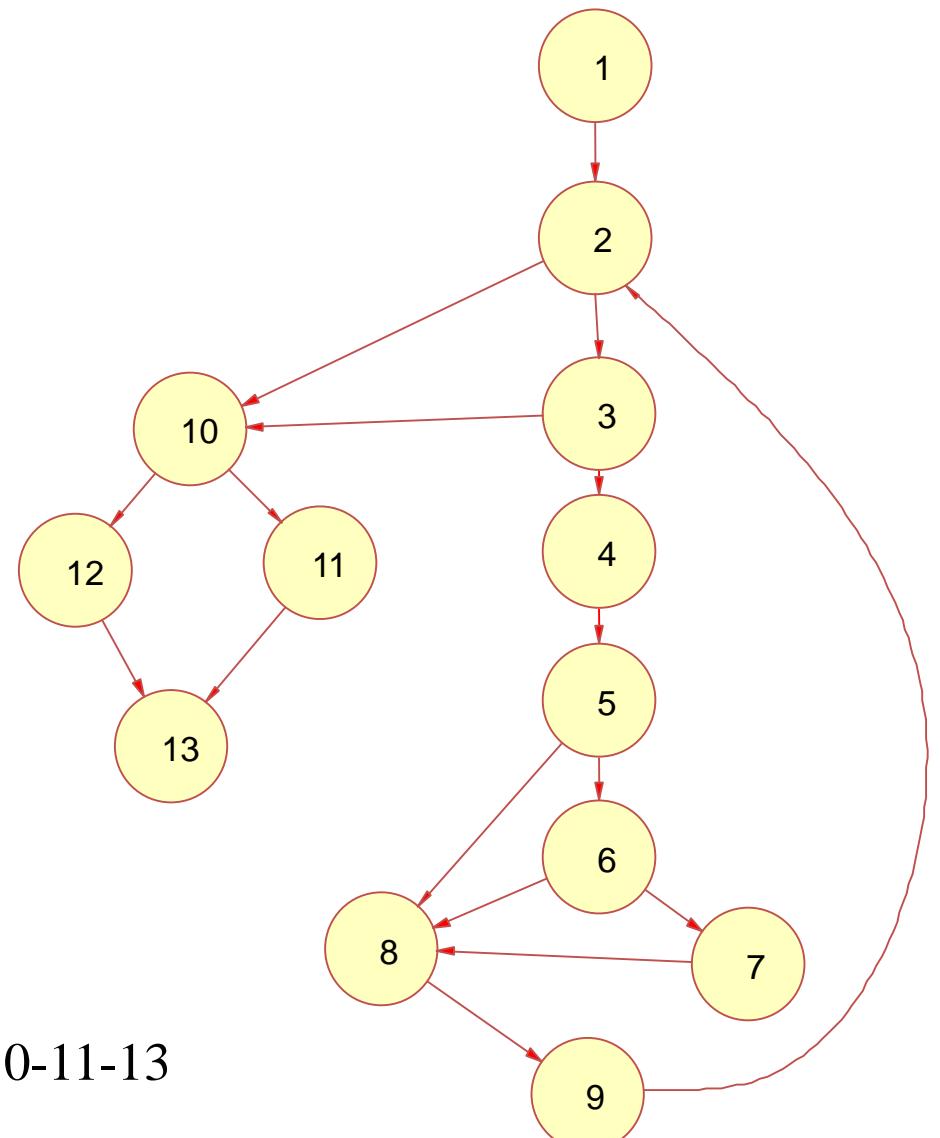
4: 1-2-3-4-5-8-9-2...

5: 1-2-3-4-5-6-8-9-2...

6: 1-2-3-4-5-6-7-8-9-2...

OTROS CAMINOS

7: 1-2-3-4-5-6-7-8-9-2-3-4-5-6-7-8-9-2-10-11-13



Ejemplo completo

```
PROCEDURE media;
INTERFACE DEVUELVE media, total.entrada, total.válido;
INTERFACE ACEPTE valor, mínimo, máximo
TYPE valor[1:100] ES ARRAY FLOAT;
TYPE media, total.entrada, total.válido ES FLOAT
TYPE mínimo, máximo, suma ES FLOAT
TYPE i ES ENTERO
i=1;
total.entrada=total.válido=0;
suma=0;1
DO WHILE valor[i]<>-999 2 AND total.entrada<100 3
    total.entrada+=1; 4
    IF valor[i]>=mínimo 5 and valor[i]<= maximo 6
        THEN total.válido+=1;
            suma+=valor[i]; 7
        ELSE ignorar
    ENDIF
    i+=1; 8
END DO 9
IF total.válido >0 10
    THEN media=suma/total.válido; 11
    ELSE media=-999; 12
ENDIF 13
```

Camino 1: 1-2-10-11-13

Valor(1) = dato válido

Valor(i) = -999, $2 \leq i \leq 100$

Media = Valor(1)

Camino 1, alternativo:

1-2-3-4-5-6-7-8-9-2-10-11-13

Caso de Prueba:

Estado:

mínimo = 5

máximo = 20

Entrada:

Valor(1) = 10.

Valor(i) = -999, $2 \leq i \leq 100$

Salida:

Media = 10

Ejemplo completo

```
PROCEDURE media;
INTERFACE DEVUELVE media, total.entrada, total.válido;
INTERFACE ACEPTE valor, mínimo, máximo
TYPE valor[1:100] ES ARRAY FLOAT;
TYPE media, total.entrada, total.válido ES FLOAT
TYPE mínimo, máximo, suma ES FLOAT
TYPE i ES ENTERO
i=1;
total.entrada=total.válido=0;
suma=0;1
DO WHILE valor[i]<>-999 2 AND total.entrada<100 3
    total.entrada+=1; 4
    IF valor[i]>=mínimo 5 and valor[i]<= maximo 6
        THEN total.válido+=1;
            suma+=valor[i]; 7
        ELSE ignorar
    ENDIF
    i+=1; 8
END DO 9
IF total.válido >0 10
    THEN media=suma/total.válido; 11
    ELSE media=-999; 12
ENDIF 13
```

Camino 1: 1-2-10-11-13

Valor(1) = dato válido

Valor(i) = -999, $2 \leq i \leq 100$

Media = Valor(1)

Camino 2: 1-2-10-12-13

Valor(1) = -999

Media = -999

Caso de Prueba:

Estado:

mínimo = 5

máximo = 20

Entrada:

Valor(i) = -999, $1 \leq i \leq 100$

Salida:

Media = -999

Ejemplo completo

```
PROCEDURE media;
INTERFACE DEVUELVE media, total.entrada, total.válido;
INTERFACE ACEPTE valor, mínimo, máximo
TYPE valor[1:100] ES ARRAY FLOAT;
TYPE media, total.entrada, total.válido ES FLOAT
TYPE mínimo, máximo, suma ES FLOAT
TYPE i ES ENTERO
i=1;
total.entrada=total.válido=0;
suma=0;1
DO WHILE valor[i]<>-999 2 AND total.entrada<100 3
    total.entrada+=1; 4
    IF valor[i]>=mínimo 5 and valor[i]<= maximo 6
        THEN total.válido+=1;
            suma+=valor[i]; 7
        ELSE ignorar
    ENDIF
    i+=1; 8
END DO 9
IF total.válido >0 10
    THEN media=suma/total.válido; 11
    ELSE media=-999; 12
ENDIF 13
```

Camino 1: 1-2-10-11-13

Valor(1) = dato válido

Valor(i) = -999, $2 \leq i \leq 100$

Media = Valor(1)

Camino 2: 1-2-10-12-13

Valor(1) = -999

Media = -999

Camino 3: 1-2-3-10-11-13

Intentamos procesar 100 valores, con los 100 primeros $\neq -999$

Media = Valor medio de los 99

Camino 4: 1-2-3-4-5-8-9-2...

Valor(i) = dato válido, $i < 100$

Valor(k) < mínimo, $k < i$

Media: Valor correcto sobre los datos $>$ mínimo

Ejemplo completo

```

PROCEDURE media;
INTERFACE DEVUELVE media, total.entrada, total.válido;
INTERFACE ACEPTE valor, mínimo, máximo
TYPE valor[1:100] ES ARRAY FLOAT;
TYPE media, total.entrada, total.válido ES FLOAT
TYPE mínimo, máximo, suma ES FLOAT
TYPE i ES ENTERO
i=1;
total.entrada=total.válido=0;
suma=0;1
DO WHILE valor[i]<>-999 2 AND total.entrada<100 3
    total.entrada+=1; 4
    IF valor[i]>=mínimo 5 and valor[i]<= maximo 6
        THEN total.válido+=1;
            suma+=valor[i]; 7
        ELSE ignorar
    ENDIF
    i+=1; 8
END DO 9
IF total.válido >0 10
    THEN media=suma/total.válido; 11
    ELSE media=-999; 12
ENDIF 13

```

Camino 5: 1-2-3-4-5-6-8-9-2...

Valor(i)= dato válido, $i < 100$

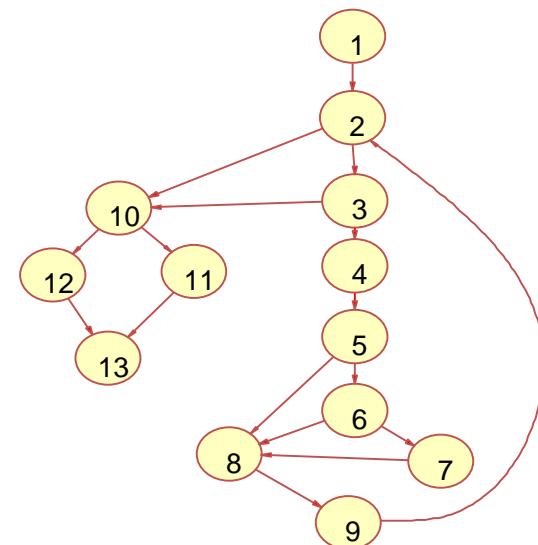
Valor(k) >máximo, $k < i$

Media: Valor correcto sobre los datos menores que máximo

Camino 6: 1-2-3-4-5-6-7-8-9-2...

Valor(i)= dato válido, $i < 100$

Media: Valor correcto sobre los n valores totales y adecuados



Tema 6. Pruebas del software

6.1.- Introducción.

6.2.- Pruebas Estructurales

6.3.- Prueba Funcional

6.4.- Enfoque práctico recomendado para el diseño
de casos

6.5.- Documentación del diseño de las pruebas

6.6.- Ejecución de las pruebas

6.7.- Estrategia de aplicación de las pruebas

6.8.- Pruebas en desarrollos orientados a objetos

Prueba Funcional. Caja Negra.

- Trataremos de encontrar errores en las siguientes categorías:
 1. Funciones incorrectas o ausentes
 2. Errores de interfaz
 3. Errores en estructuras de datos o acceso a BD
 4. Errores de rendimiento
 5. Errores de inicialización y terminación

Prueba funcional. Métodos de Caja Negra.

- La prueba funcional o de caja negra se centra en el estudio de las especificaciones.
 - Conocidas las entradas que salidas esperamos.
- No es posible la prueba exhaustiva.

Prueba funcional. Métodos de Caja Negra.

- Selección de casos de prueba
 - Enfoque sistemático
 - Búsqueda de buenos casos de prueba
 - Un buen caso de prueba es aquel que tienen una alta probabilidad de detectar un nuevo error
 - El caso ejecute el máximo número de posibilidades de entrada diferentes para así reducir el total de casos.
 - Cada entrada cubre un conjunto extenso de otras
 - Enfoque aleatorio.
 - Pruebas aleatorias

Prueba funcional. Métodos de Caja Negra.

- Selección de casos de prueba
 - Enfoque sistemático
 - Partición o clases de equivalencia
 - Análisis de valores límite AVL
 - Conjetura de Errores
 - Enfoque aleatorio.
 - Pruebas aleatorias

Enfoque Sistemático

- Busca utilizar casos de prueba bien elegidos.

Definiciones de Myers:

- El que reduce el número de casos necesarios para que la prueba sea razonable. Explota al máximo las posibilidades de combinaciones de entradas.
- El que cubre un conjunto extenso de casos posibles. Da información sobre la ausencia o presencia de defectos con las entradas probadas pero también sobre un conjunto de otras entradas similares no probadas.

Partición o clases de equivalencia

- Dividimos el dominio de valores de entradas en un número finito de clases que cumplan:
 - La prueba de un valor representativo de una clase permite suponer razonablemente que el resultado obtenido será el mismo que el obtenido probando cualquier otro valor de la clase

Partición o clases de equivalencia

- Método de diseño de casos consiste:
 - Identificar las clases de equivalencia
 - Crear los casos de prueba correspondientes

Partición o clases de equivalencia

- Identificar las clases de equivalencia
 - Identificación de las condiciones de las entradas del programa. (Restricciones de formato y valores)
 - Identificación de las clases de equivalencia:
 - Datos válidos
 - Datos no válidos o erróneos

Partición o clases de equivalencia

- Reglas de identificación de clases
 - R1.- Rango $5 < N < 7$
 - Una clase válida y dos no válidas
 - R2.- Lista de valores de tamaño variable. Ej. Titulares de una Cuenta Bancaria (es posible: 1, 2, 3 ó 4)
 - Una clase válida y dos no válidas
 - R3.- Situación del tipo debe ser o booleana. Edad es número
 - Una clase válida y una no válida
 - R4.- Valores admitidos con comportamientos distintos. Pago con tarjeta
 - Una clase válida por cada uno de ellos, y otra no válida
 - R5.- Si es necesario se subdividen las clases

Partición o clases de equivalencia

- Método de diseño de casos:
 - Identificar las clases de equivalencia
 - Crear los casos de prueba correspondientes.
 - Pasos a seguir:
 1. Numeramos las clases de equivalencia
 2. Hasta que todas las C.E. Válidas hallan sido cubiertas, especificar casos de prueba que cubran el mayor número posible de clases válidas no cubiertas
 3. Hasta que todas las C.E. No Válidas hallan sido cubiertas, especificar casos de prueba únicos por cada CENV sin cubrir.

Partición o clases de equivalencia

- Pasos a seguir:

3.- *Hasta que todas las C.E. No Válidas hallan sido cubiertas, especificar casos de prueba únicos por cada CENV sin cubrir.*

Contraejemplo: Introducir un valor entre 5 y 25

200A

El primer error enmascararía el segundo, por lo que un solo caso de prueba no llegaría

Partición o clases de equivalencia

□ Ejemplo

□ Leemos de fichero dos tipos de registros:

■ Registro_Participantes:

- Tipo_Registro: Definido en un dominio numérico sin signo, sólo toma el valor 0
- Ganador: Definido en el dominio lógico, toma el valor Verdadero si el participante es ganador y Falso en caso contrario
- Nombre: Cadena alfanumérica

■ Registro_Preguntas:

- Tipo_Registro: Definido en un dominio numérico sin signo, sólo toma el valor 1
- Pregunta: Definido en el dominio numérico, toma valores entre 1 y 99
- Respuesta: Respuesta a la pregunta, dominio alfanumérico

Partición o clases de equivalencia

- Reglas de identificación de clases
 - R1.- Si se especifica un **rango** de valores para los datos de entrada, se creará una clase válida y dos no válidas
 - R2.- Si se especifica un **número de valores** (alternativas) posibles (por ejemplo el número de titulares de una cuenta bancaria ha de ser mayor que cero y menor que seis), se establece una clase válida y dos no válidas.
 - R3.- Si se especifica una situación del tipo debe ser o **booleana**, se identifica una clase válida y una no válida
 - R4.- Si se especifica un **conjunto de valores** admitidos, y el software trata de forma diferente cada uno de ellos, se establece una clase válida por cada uno de ellos, y otra no válida
 - R5.- En cualquier caso, si es necesario se **subdivide** cada clase válida en otras menores

Partición o clases de equivalencia

- Ejemplo

Información	Tipo de dato	Regla	Clase Válida		Clase No válida	
			Id	Dominio	Id	Dominio
Tipo_Registro	Valores Válidos	4	1	0		
			2	1	3	>1
	Numérico sin signo	5,3		>=0	4	No es número positivo
Ganador	Booleana	4	5	True	6	No existe
			7	False		
Participante	Txt	3	8	Existe	9	No existe
Pregunta	1-99	1	10	1-99	11	<1
					12	>99
Respuesta	Txt	3	13	Existe	14	No existe

Partición o clases de equivalencia

□ Ejemplo

Casos de Prueba Válidos				
	0	TRUE	Pepe	1, 5, 8
	0	FALSE	Pepe	1, 7, 8
	1	50	Mi casa	2, 10, 13
Casos de Prueba No Válidos				
	2	FALSE	Pepe	3, 5, 8
	a	FALSE	Pepe	4, 5, 8
	0		Pepe	1, 6, 8
	0	FALSE		1, 5, 9
	1	-1	Mi casa	2, 11, 13
	1	222	Mi casa	2, 12, 13
	1	50		2, 10, 14

Análisis de valores Límite (AVL)

- Técnica que complementa la de Particiones en Clases de Equivalencia con dos diferencias:
 - Seleccionamos elementos de la frontera de cada clase
 - Examinamos también el dominio de salida

Análisis de valores Límite (AVL)

□ Reglas de construcción

- R1.- Si una condición de entrada especifica un intervalo cerrado de valores [-1.0,1.0], examinaremos exactamente los valores extremos del intervalo -1.0 y 1.0, y los casos no válidos justo fuera del intervalo, esto es -1.1 y 1.1 (si sólo se admite un decimal)
- R2.- Si la condición especifica un número de valores de entrada (i.e. El fichero tendrá de 1 a 255 registros), diseñaremos casos con los valores máximo y mínimo y uno más que el máximo y uno menos que el mínimo (i.e. 0, 1, 255, 256)

Análisis de valores Límite (AVL)

□ Reglas de Construcción

- R3.- Usar la regla 1 para la condición de salida. Por ejemplo: el descuento aplicable será del 6% al 50%; Intentaremos obtener descuentos para 5.99%, 6%, 50% y 50.01%
- R4.- Usar la regla 2 para cada condición de salida. Por ejemplo: El programa puede generar de 1 a 4 informes; trataríamos de generar 0, 1, 4 y 5 informes.
- R5.- Si la entrada o salida del programa es una colección ordenada de objetos (Tabla, fichero secuencial, etc.), nos centraríamos en el primer y último elemento.

Análisis de valores Límite (AVL)

- Consideraciones sobre las reglas de salida R3 y R4:
 - 1.- Los valores límite de entrada no siempre nos dan valores límites de salida.
 $y = \operatorname{seno}(x) \quad x \in [0, 2\pi] \quad y \in [-1, 1]$
 - 2.- No siempre se pueden obtener resultados fuera del rango de salida, pero es interesante considerar esta posibilidad

Conjetura de errores

- Se trata de hacer una lista de equivocaciones que pueda cometer un programador y diseñar un caso de prueba para cada elemento de la lista.
 - Valor 0 (entrada/salida)
 - En listas de valores concentrarse en el caso de que no haya valores, que haya 1 o que todos sean iguales
 - Intentar imaginar que se puede malinterpretar en las especificaciones
 - Prever que el usuario no es muy hábil o es malintencionado.

Pruebas aleatorias

- El sistema funciona correctamente con datos válidos
 - Eventualmente se deberían crear todas las posibles entradas al sistema. Podemos apoyarnos en métodos estadísticos para conseguir la misma distribución de entrada o en reglas de comportamiento que deban seguir los datos.
- Test de esfuerzo
 - Se trata de simular la entrada al programa en la secuencia y la frecuencia con la que pueden aparecer en realidad los datos de forma continua y sin parar.

Métodos de Caja Negra basados en grafos

- Identificación de nodos y sus atributos
 - Es muy útil identificar los nodos de entrada y de salida.
- Establecer enlaces y sus pesos (p ej. Tiempo para la generación de un informe)
- Utilidad. Facilitan la identificación de casos de prueba para:
 - Identificación de bucles a probar. Prueba de Bucle.
 - Cobertura de nodo. Que no se han omitido nodos
 - Cobertura de enlace. Todas las relaciones se prueban separadamente basándose en sus propiedades.
 - Reflexiva.
 - Simetría.
 - Transitividad.

Métodos de Caja Negra basados en grafos

- Son posibles distintos modelos
 - Modelo de Estados Finitos
 - Modelo de Transacción
 - Modelo de Flujo de Datos.
- Todos son modelos de Caja Negra
 - No modelan como está programado el software
 - No son los Estados del software, ni sus transacciones programadas ni sus flujos de datos.
 - Modelan su comportamiento visible, pantallas por las que pasa, o el contexto del problema, cómo se hace una reserva (con software o sin el).

Métodos de Caja Negra basados en grafos

- Modelado de estado finito:
 - Los nodos son estados del software observables por el usuario y los enlaces representan las transiciones que ocurren para moverse de un estado a otro.
 - Ejemplo: Al realizar una determinada función, el usuario ve una serie de PANTALLAS en las que puede realizar distintas acciones que llevan a distintas pantallas.
 - Los nodos reflejan PANTALLAS
 - Las transiciones reflejan ACCIONES que cambian la pantalla

Métodos de Caja Negra basados en grafos

- Modelado del flujo de transacción:
 - Los nodos representan los pasos de alguna transacción y los enlaces son las conexiones lógicas entre ellos. Podríamos partir del DFD
 - Ejemplo: Los pasos requeridos para hacer una reserva en un hotel o una aerolínea. Son pasos del proceso, se use un software o no.

Métodos de Caja Negra basados en grafos

- Modelado del flujo de datos:
 - Los nodos son objetos de datos y los enlaces son las transformaciones que sufren para pasar de ser un objeto a ser otro.
 - Ejemplo: Cálculo de hipoteca. A partir del valor de hipoteca y plazo de amortización resultan unos gastos, mensualidades e intereses que pueden llevar a cambiar el valor o el plazo

Tema 6. Pruebas del software

6.1.- Introducción.

6.2.- Pruebas Estructurales

6.3.- Prueba Funcional

6.4.- Enfoque práctico recomendado para el diseño de casos

6.5.- Documentación del diseño de las pruebas

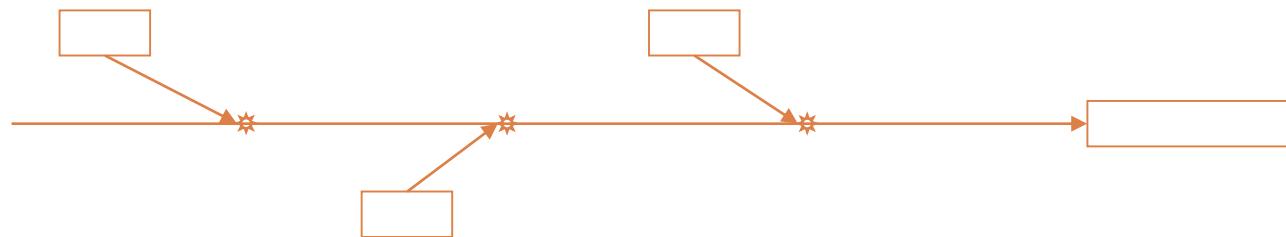
6.6.- Ejecución de las pruebas

6.7.- Estrategia de aplicación de las pruebas

6.8.- Pruebas en desarrollos orientados a objetos

Enfoque recomendado

1. Si la especificación contiene combinaciones de condiciones de entrada, comenzar formando grafos causa-efecto



2. Identificar clases de equivalencia válidas y no validas para la entrada y la salida
3. En todos los casos usar AVL para añadir casos de prueba
4. Utilizar conjetura de errores para añadir nuevos casos
5. Ejecutar los casos generados hasta el momento y analizar la cobertura obtenida
6. Elegir casos precisos de caja blanca si en 5 no se ha cumplido el criterio de cobertura lógica elegido.

Tema 6. Pruebas del software

6.1.- Introducción.

6.2.- Pruebas Estructurales

6.3.- Prueba Funcional

6.4.- Enfoque práctico recomendado para el diseño
de casos

6.5.- Documentación del diseño de las pruebas

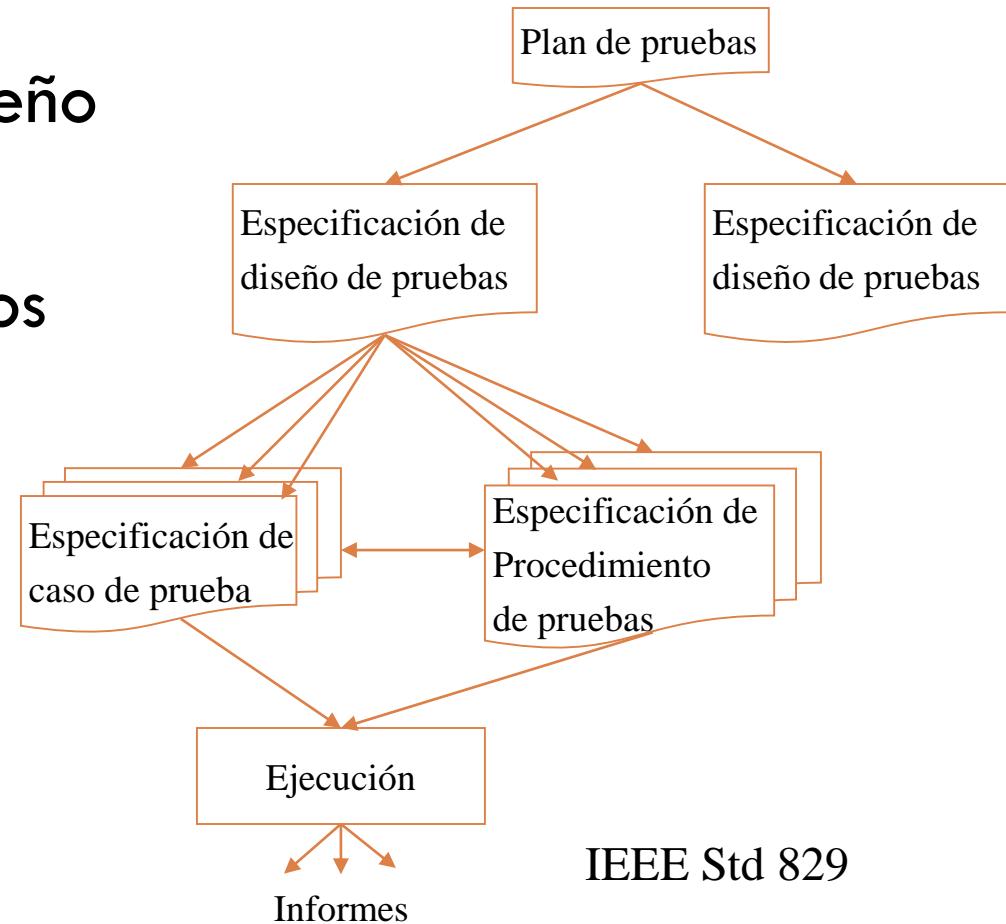
6.6.- Ejecución de las pruebas

6.7.- Estrategia de aplicación de las pruebas

6.8.- Pruebas en desarrollos orientados a objetos

Documentación de diseño de pruebas

- Plan de pruebas
- Especificación del diseño de prueba
- Especificación de casos de prueba
- Especificación de procedimiento de pruebas
- Ejecución.



IEEE Std 829

Documentación de diseño de pruebas

- Plan de pruebas: Planificación general del esfuerzo a realizar
 - Señalar el enfoque, los recursos y el esquema de las actividades de prueba
 - Los elementos y características a probar y a no probar
 - Las actividades de prueba,
 - El personal responsable
 - Los riesgos asociados
- Especificación del diseño de prueba
- Especificación de casos de prueba
- Especificación de procedimiento de pruebas
- Ejecución.

Documentación de diseño de pruebas

- Plan de pruebas
- Especificación del diseño de prueba: Detalla el anterior plan de pruebas
 - Características de los elementos a probar
 - Detalles como Técnicas y métodos de análisis de resultados.
 - Identificación de pruebas (un Id para cada prueba)
 - Describe los casos de prueba a utilizar (un Id para cada CP)
 - Procedimientos a seguir
 - Criterios de paso/fallo de la prueba
- Especificación de casos de prueba
- Especificación de procedimiento de pruebas
- Ejecución.

Documentación de diseño de pruebas

- Plan de pruebas
- Especificación del diseño de prueba
- Especificación de casos de prueba: Define con detalle los casos de prueba mencionados en el punto anterior.
 - Elementos Software y características a probar
 - Especificación de entradas requeridas, y su sincronización)
 - Especificación de las salidas (y sus características: tiempo de respuesta)
 - Necesidades del entorno (Hardware, software, personal...)
 - Requisitos especiales
 - Dependencias entre casos.
- Especificación de procedimiento de pruebas
- Ejecución.

Documentación de diseño de pruebas

- Plan de pruebas
- Especificación del diseño de prueba
- Especificación de casos de prueba
- Especificación de procedimiento de pruebas: Indican como proceder en detalle a la ejecución de los casos
 - Objetivos y lista de casos para evaluar un elemento
 - Requisitos especiales
 - Pasos: Formas de registrar resultados e incidencias
 - Secuencias necesarias para preparar la ejecución
 - Acciones para empezar y continuar la ejecución
 - Cómo realizar las medidas
 - Cómo tratar las incidencias y restaurar el entorno
- Ejecución.

Tema 6. Pruebas del software

6.1.- Introducción.

6.2.- Pruebas Estructurales

6.3.- Prueba Funcional

6.4.- Enfoque práctico recomendado para el diseño
de casos

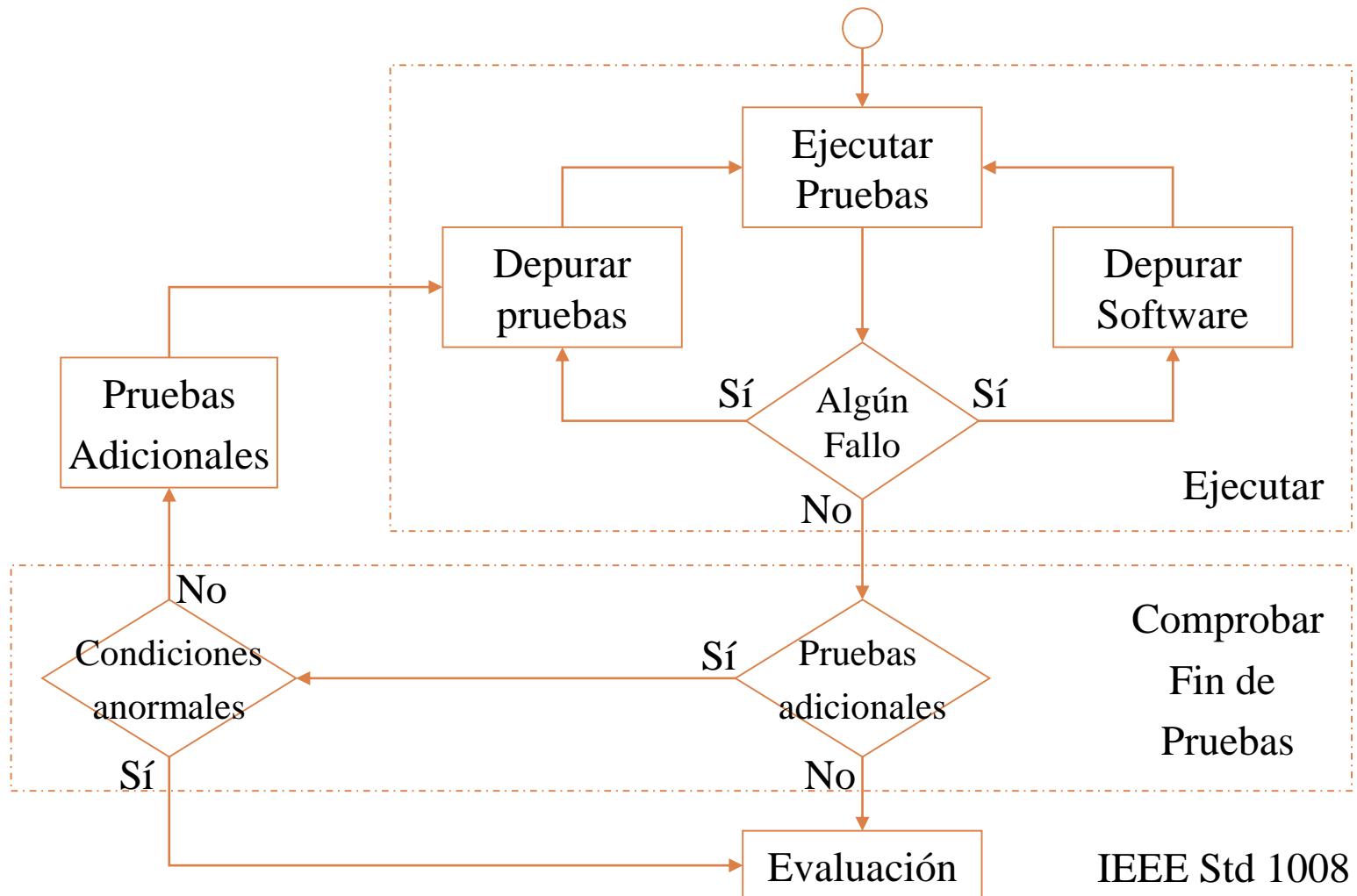
6.5.- Documentación del diseño de las pruebas

6.6.- Ejecución de las pruebas

6.7.- Estrategia de aplicación de las pruebas

6.8.- Pruebas en desarrollos orientados a objetos

Ejecución de las pruebas



Documentación de la ejecución

- Histórico de pruebas
 - Registro cronológico de la ejecución.
 - Elementos probados y su entorno
 - Fecha
 - Referencia al informe de incidencia si lo hay
- Informe de Incidencia
 - Resumen del incidente y analiza su impacto sobre las pruebas.
- Informe resumen
 - Resume los resultados de las pruebas y aporta una evaluación del software basado en dichos resultados.
 - Variaciones del software y las pruebas con respecto a su especificación o diseño.
 - Valoración de la cobertura lógica alcanzada.
 - Resumen de las actividades (Detalles de recursos).

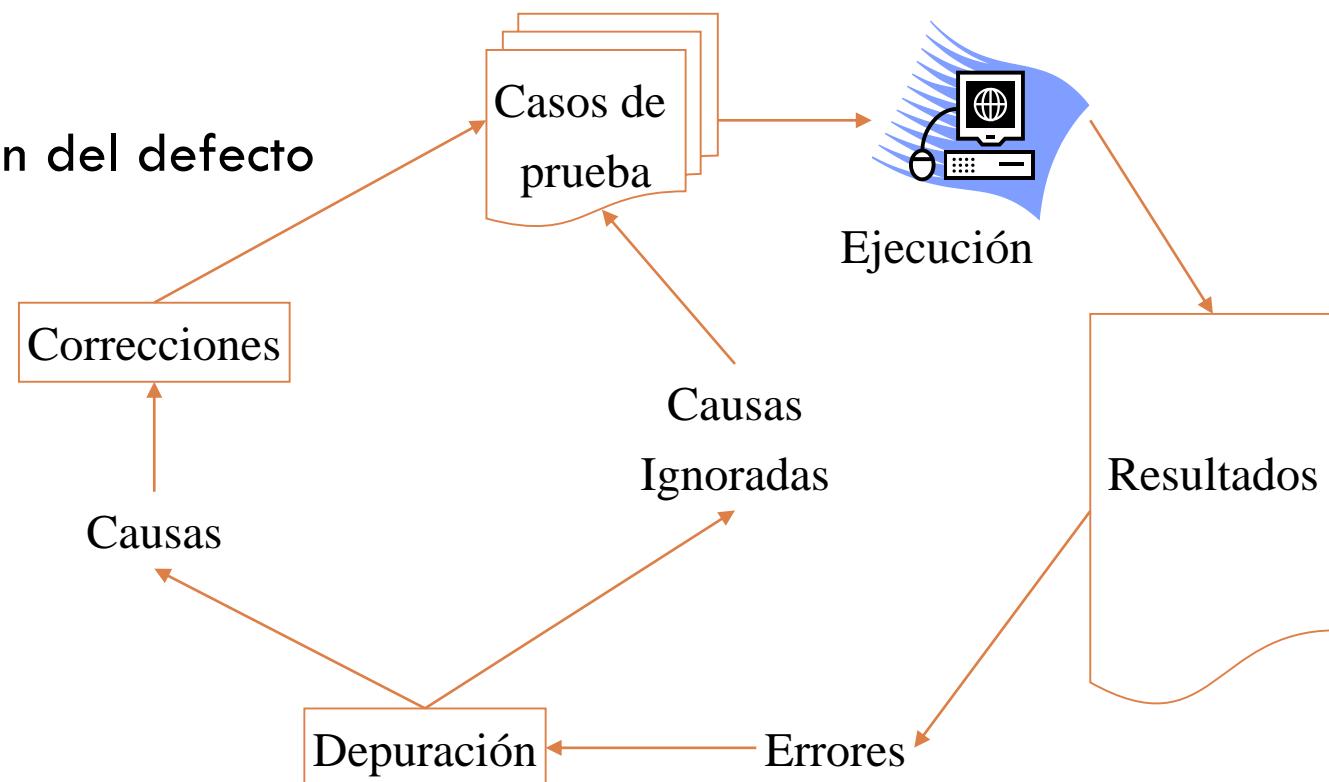
Depuración

Objetivos

- Encontrar la causa del error, analizarla y corregirla
- Si no se encuentra la causa generar nuevos casos de prueba

Etapas

- Localización del defecto
- Corrección



Consejos de depuración

- Localización del error
 - Proceso mental de solución de un problema.
 - Analizar la información.
 - No explorar aleatoriamente.
 - No experimentar cambiando el programa
 - Usar herramientas de depuración sólo como recurso secundario
 - Al llegar a un punto muerto
 - pasar a otra cosa
 - Describir el problema a otra persona
 - Se deben atacar los errores individualmente
 - Se debe fijar la atención también en los datos y no sólo en la lógica del proceso.

Consejos de depuración

- Corrección del error
 - Donde hay un defecto suele haber más
 - Debe corregirse el defecto no sus síntomas
 - La probabilidad de corregir un defecto perfectamente no es del 100%
 - Cuidado con crear nuevos defectos
 - La corrección debe situarnos temporalmente en la fase de diseño

Tema 6. Pruebas del software

6.1.- Introducción.

6.2.- Pruebas Estructurales

6.3.- Prueba Funcional

6.4.- Enfoque práctico recomendado para el diseño
de casos

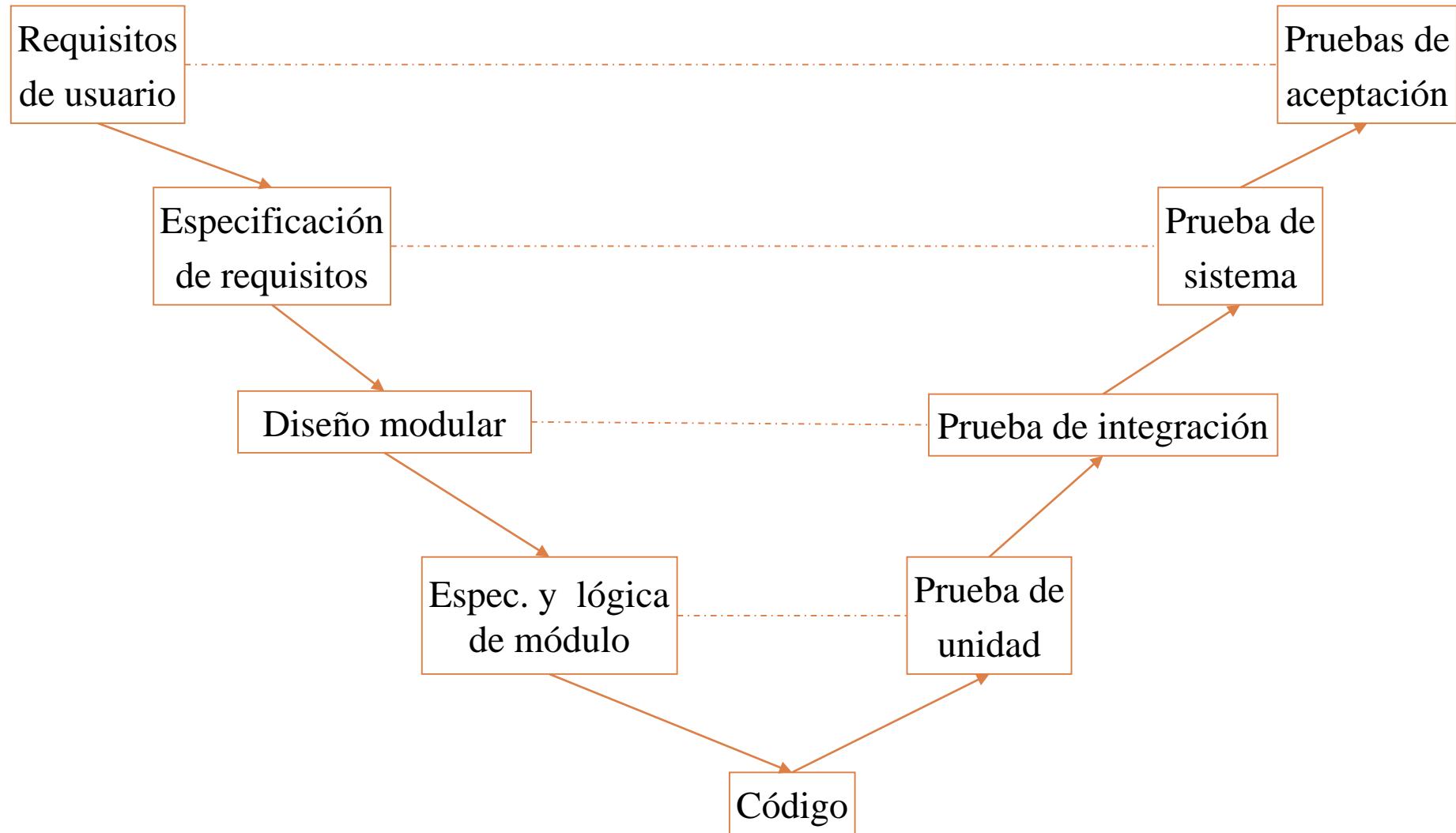
6.5.- Documentación del diseño de las pruebas

6.6.- Ejecución de las pruebas

6.7.- Estrategia de aplicación de las pruebas

6.8.- Pruebas en desarrollos orientados a objetos

Estrategia de aplicación de pruebas



Tema 6. Pruebas del software

6.1.- Introducción.

6.2.- Pruebas Estructurales

6.3.- Prueba Funcional

6.4.- Enfoque práctico recomendado para el diseño
de casos

6.5.- Documentación del diseño de las pruebas

6.6.- Ejecución de las pruebas

6.7.- Estrategia de aplicación de las pruebas

6.8.- Pruebas en desarrollos orientados a objetos

Pruebas en desarrollos OO

- Técnicas de caja negra: Totalmente válidos
 - Análisis de valores límite, tratamiento de combinaciones de entrada, conjetura de errores.
 - Diseñar los casos de prueba basándose
 - Datos y eventos de los escenarios de los casos de uso.
 - Flujos alternativos, tratamientos de error y excepciones.
- Técnicas de caja blanca.
 - Quedan confinadas a su aplicación en los métodos de las clases.

Pruebas en desarrollos OO

- Diseño de pruebas: Distinguir criterios según nivel
 - Pruebas de unidad:
 - Análisis por clase de equivalencia y AVL para entradas y salidas de métodos de clase
 - Pruebas de integración
 - Clases independientes o con un funcionamiento dependiente de pocas clases para ir añadiendo el resto
 - Probar hilos de clases que colaboran en una función
 - La filosofía de la programación orientado a objetos cambian:
 - Herencia: probar un método en la clase padre no garantiza su funcionamiento en clases hijas
 - Polimorfismo: un único método puede tener implementaciones diferentes en función de la clase en la que se use.