Tabla de contenidos

1 Proceso de desarrollo de software

El objetivo de este capítulo es realizar una definición clara de proceso de desarrollo de software y los elementos que lo conforman, además identificar su papel dentro de la Ingeniería del software. Cuando termine de leer este capítulo, usted:

- conocerá cuáles son las principales problemáticas a las que se ha enfrentado la Ingeniería del software.
- conocerá el papel del proceso de desarrollo dentro de la Ingeniería del software
- conocerá qué es un proceso de desarrollo de software y qué elementos la conforman.
- conocerá los principales modelos de procesos.
- conocerá qué es metodología de desarrollo de software y las principales tendencias.

1.1 Ingeniería del Software y sus problemas

El uso de sistemas de software cada vez es más común; muchas áreas como medicina, comunicaciones, arte, educación, o comercio se apoyan en estos, permitiendo agilizar y economizar esfuerzos.

Muchos de nosotros no podríamos imaginarnos hoy día sin algunos instrumentos como una computadora, un celular, sin el uso de una tarjeta electrónica para realizar nuestros pagos; esto y más es posible gracias a las bondades que brindan algunos productos de software que permite el funcionamiento de estos instrumentos.

Cada día aumentan los requerimientos y las competencias de los productos de software. Podríamos pensar que la disciplina encargada de elaborar dichos productos ha evolucionado y madurado lo suficiente para satisfacer lo requerido.

E primer reconocimiento público de la existencia de problemas en la producción de software tuvo lugar en la conferencia organizada en 1968 por la Comisión de Ciencias de la OTAN en Garmisch (Alemania) con el objetivo de buscar soluciones a dicha situación problemática, que se denominó desde entonces como **crisis del software**. En esta conferencia, así como en la siguiente realizada en Roma en 1969, se manifestó el interés por los aspectos técnicos y administrativos en el desarrollo y mantenimiento de productos de software. Se pretendía hacer una **ingeniería de construcción de software**.

Según Fritz Bauer [Naur 1969] lo que se deseaba era "establecer y usar principios de ingeniería orientados a obtener software de manera económica, que sea confiable y funcione eficientemente sobre máquinas reales".

En [Pressman 1997] se comenta que en el reconocimiento anterior no se dice mucho sobre los aspectos técnicos de la calidad del software; no se enfrenta directamente con la necesidad de la satisfacción del cliente o de la entrega oportuna del producto; omite la mención de la importancia de mediciones y métricas; tampoco expresa la importancia de un proceso avanzado. Las

consideraciones anteriores solo nos proporcionan una línea base. ¿Cuáles son los principios robustos de la ingeniería aplicables al desarrollo de software de computadora? ¿Cómo construimos el software económicamente para que sea confiable? ¿Qué se necesita para crear programas de computadora que funcionen eficientemente no en una máquina si no en diferentes máquinas reales? Por lo tanto se seguían encontrando los siguientes problemas en los proyectos de software:

- Los sistemas no responden a las expectativas de los usuarios.
- Los programas "fallan" con cierta frecuencia.
- Los costos del software son difíciles de prever y normalmente superan las estimaciones.
- La modificación del software es una tarea difícil y costosa.
- El software se suele presentar fuera del plazo establecido y con menos características que las consideradas inicialmente.
- Normalmente, es difícil cambiar de entorno de hardware usando el mismo software.
- El aprovechamiento óptimo de los recursos (personas, tiempo, dinero, herramientas, etc.) no suele cumplirse.

Según el Centro Experimental de Ingeniería de Software (CEIS)[1], el estudio de mercado *The Chaos Report* realizado por *Standish Group International*, *Inc* [2] en 1994[3], concluyó que sólo un 16% de los proyectos de software son exitosos (terminan dentro de plazos y costos y cumplen los requerimientos acordados). Otro 53% sobrepasa costos y plazos y cumple parcialmente los requerimientos. El resto (31%), ni siquiera llega al término. Además señalaba las siguientes peores prácticas y prácticas ausentes del desarrollo de software:

- No existe medición del proceso ni registro de datos históricos.
- Rechazo a estimaciones imprevistas de plazos y costos.
- Mal uso de herramientas automatizadas para la planificación y estimación.
- Excesiva e irracional presión en los calendarios.
- Crecimiento excesivo de los requerimientos para un producto de software.
- Escaso o deficiente monitoreo en el avance del proceso de desarrollo.
- No se hace gestión de riesgos formalmente.
- No se realiza un proceso formal de pruebas.
- No se realizan revisiones técnicas formales e inspección de código.

Por otra parte, el proyecto Herramientas de Mejora de la Productividad para la Industria de Software en Chile (HMS) liderado por el Corporación de Investigación Tecnológica de Chile (INTEC)[4] entre los años 1995 y 1998, reveló que los riesgos más frecuentes en la construcción de un producto de software son:

- Los criterios de aceptación no son claros.

[1] Ver http://www.ceis.cl/Gestacion/Gestacion.htm (20.8.2005)
[2] Ver http://www.standishgroup.com/sample_research/chaos_1994_1.php (20.8.2005)
[3] Ver http://standishgroup.com/ (20.8.2005)
[4] Ver http://www.intec.cl/ (5.2.2003)

- La propiedad intelectual.
- No todos los involucrados comprenden los requerimientos del sistema.
- La poca reutilización de software existente.
- La interferencia excesiva del cliente.
- La escasa integración de software comercialmente disponible.
- La ausencia de planificación del proyecto.
- El control de calidad inadecuado.
- La gestión de configuración inadecuada.
- Los roles y responsabilidades no están claros.
- Especificaciones de contratos insuficientes, ambiguas e incompletas.
- El proyecto depende de muy pocos individuos.

El último estudio del Centro Experimental de Ingeniería de Software (CEIS), el estudio de mercado *The Chaos Report* realizado por *Standish Group International, Inc* en Marzo del 2003[5], señala una mejora en la cantidad de proyectos de desarrollo de software exitosos (terminan dentro de plazos y costos y cumplen los requerimientos acordados). En 1994 se señalaba que un 16% de los proyectos de software eran exitosos, en 2003 se señala este porcentaje como un 34%, con una mejora alrededor del 50% en esta categoría.

Esto es un avance significativo pero aún nos queda trabajo por realizar ya que solo el 34% del 100% terminan siendo proyectos exitosos. Además el estudio del 2003 muestra que han incrementado las problemáticas de sobrepaso de tiempos y costos, y que los productos no cumplen con las características y funcionalidades pactadas originalmente.

1.2 El papel del proceso de desarrollo dentro de la Ingeniería del software

El "*IEEE Standard Glossary of Software Engineering Terminology*" (Std. 610.12-1990) define la Ingeniería del software como [Pressman 1997]: "(1) La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento del software; es decir, la aplicación de ingeniería al software. (2) El estudio de enfoques en (1)".

Dada la definición anterior no podremos alcanzar un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento del software si no contamos con un marco de trabajo (proceso) definido y especificado que permita tener una base para la mejora continua.

Pressman [Pressman 1997] presenta la Ingeniería del Software como "**una tecnología multicapa**", ilustrada en la Figura 1.

[5] Ver http://www.standishgroup.com/press/article.php?id=2 (20.8.2005)

Figura 1: Capas de la Ingeniería del software.

Cada capa apoya o da soporte a las que se encuentran sobre ella. A continuación se explica cada una de ellas.

- Cualquier enfoque de ingeniería (incluida ingeniería del software) debe descansar sobre un empeño de organización de **calidad**. La gestión total de la calidad y las filosofías similares fomentan una cultura continua de mejoras de procesos que conduce al desarrollo de enfoques cada vez más robustos para la ingeniería del software.
- El fundamento de la Ingeniería del software es la **capa proceso**. El proceso define un marco de trabajo para un conjunto de áreas clave, las cuales forman la base del control de gestión de proyectos de software y establecen el contexto en el que se aplican los métodos técnicos, se producen resultados de trabajo, se establecen hitos, se asegura la calidad y el cambio se administra adecuadamente.
- Los **métodos** de la Ingeniería del software indican cómo construir técnicamente el software. Los métodos abarcan una gran gama de tareas que incluyen análisis de requerimientos, diseño y especificación (en diversos niveles), construcción de programas, pruebas y mantenimiento. Estos métodos dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.
- Las **herramientas** de la Ingeniería del software proporcionan un soporte automático o semi-automático para el proceso y los métodos, a estas se les llama herramientas CASE (C*omputer-Aided Software Engineering*).

Dado lo anterior, el objetivo de la Ingeniería del software es lograr productos o servicios de calidad aplicando **un adecuado proceso de desarrollo**, apoyado por métodos y herramientas.

A continuación se presenta una definición de proceso de desarrollo de software y los principales modelos de proceso.

1.3 Definición de proceso de desarrollo de software

Cuando se construye un producto o sistema es importante seguir una secuencia de pasos predecibles para generar resultados de alta calidad. Esta secuencia de pasos, en el caso de un producto o sistema de software, son llamados un **proceso de desarrollo de software**. Los requerimientos del usuario son la entrada del proceso de software, se sigue la secuencia de pasos planteados para obtener un producto como salida, como se ilustra en la Figura 2 [Jacobson 2000].

Figura 2: Un proceso de desarrollo de software

Sommerville define proceso de desarrollo de software como un conjunto de actividades y resultados asociados que conllevan a la creación de un producto de software. Es un proceso intensamente intelectual, afectado por la creatividad y el juicio de las personas involucradas [Sommerville 2002].

Existen muchas propuestas de procesos de software. No hay un proceso ideal o único ya que existen muchas variables que influyen en la efectividad de un proceso u otro. Debido a esta diversidad, es difícil automatizar todo un proceso de desarrollo de software.

A pesar de la diversidad de procesos, existe un conjunto de actividades fundamentales que se encuentran presentes en todos ellos [Sommerville 2002]:

1. **Especificación de software**: Se debe definir la funcionalidad y restricciones operacionales que debe cumplir el software.
2. **Diseño e Implementación**: Se diseña y construye el software de acuerdo con la especificación.
3. **Validación**: El software debe validarse, para asegurar que cumpla con lo que quiere el cliente.
4. **Evolución**: El software debe evolucionar, para adaptarse a las necesidades cambiantes del cliente.

Además de estas actividades fundamentales, en [Pressman 1997] se menciona un conjunto de "actividades protectoras":

- Planificación, seguimiento y control de proyectos de software.
- Revisiones técnicas formales.
- Aseguramiento de la calidad del software.
- Gestión de la configuración del software.
- Preparación y producción de documentos.
- Gestión de reutilización.
- Mediciones.
- Gestión de riesgos.

Las actividades de protección se aplican a lo largo de todo el proceso del software, permitiendo un mejor seguimiento y control sobre las actividades fundamentales.

Es importante contar con una buena definición de actividades fundamentales como base para incorporar la definición de actividades de protección dentro del proceso. Ver la Figura 3.

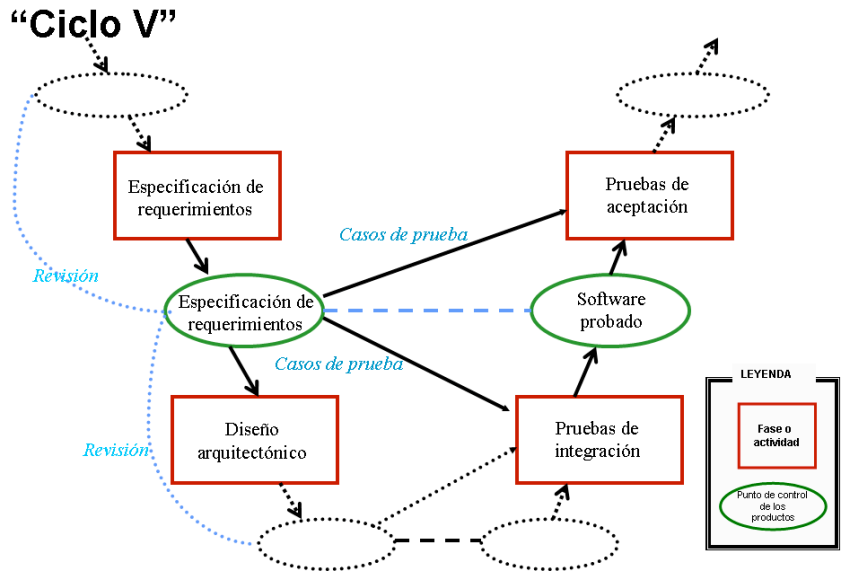


Figura 3: Incorporar actividades de protección relacionadas con Aseguramiento de calidad dentro de la definición de proceso.

Aunque no exista un proceso ideal de desarrollo, hay muchas técnicas que pueden mejorarlo, como la estandarización, la cual mejora la comunicación y permite introducir nuevos métodos, técnicas y buenas prácticas de ingeniería del software [Sommerville 2002].

Pressman [Pressman 1997] caracteriza un proceso de desarrollo de software como se muestra en la Figura 4.

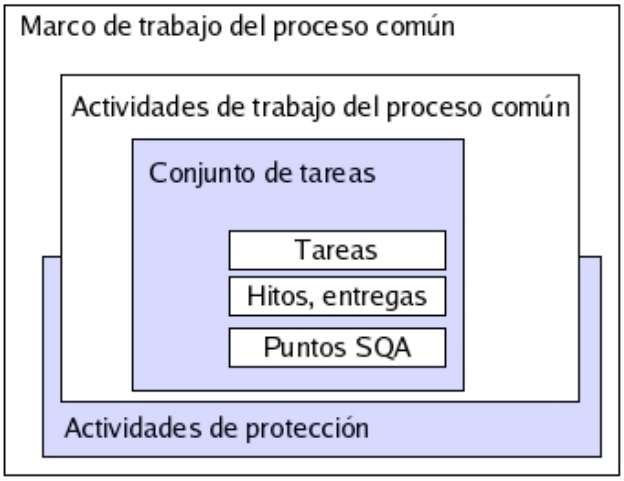


Figura 4: El proceso del software.

- **Un marco común del proceso**, que define un pequeño número de actividades del marco de trabajo que son aplicables a todos los proyectos de software, con independencia de su tamaño o complejidad.
- **Un conjunto de tareas** – cada uno es una colección de tareas de ingeniería del software, hitos de proyectos, posibles entregas y productos de trabajo del software, y puntos de garantía de calidad – que permiten que las actividades del marco de trabajo se adapten a las características del proyecto de software y los requerimientos del equipo del proyecto.
- **Las actividades de protección**, tales como aseguramiento de la calidad del software, gestión de configuración del software y medición, abarcan el modelo del proceso. Las actividades de protección son independientes de cualquier actividad del marco de trabajo y aparecen durante todo el proceso.

1.4 Elementos de un proceso de desarrollo de software

Al definir un marco de trabajo o proceso se debe contestar las siguientes interrogantes: **Quién** debe hacer **Qué**, **Cuándo y Cómo** debe hacerlo. En la Figura 5 se ilustran los elementos que definen un proceso de desarrollo de software.

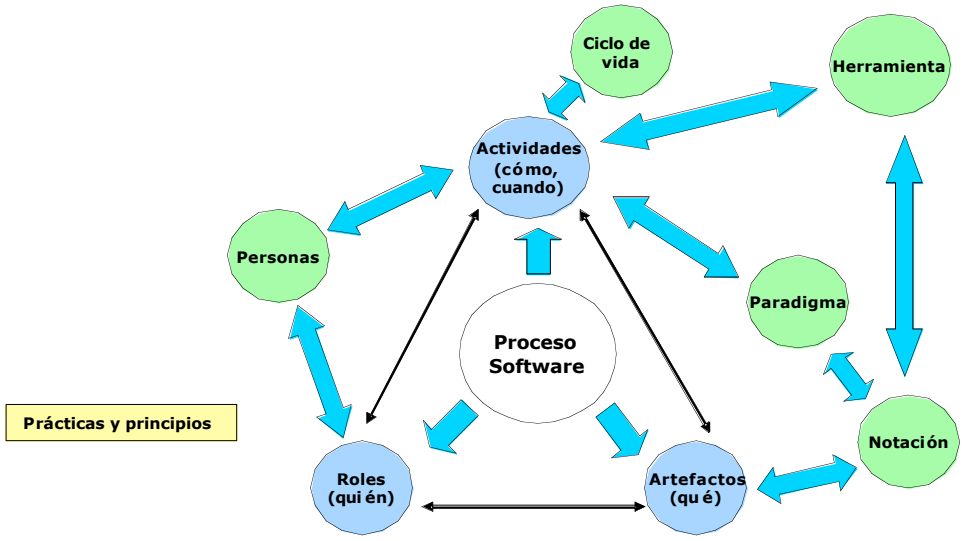
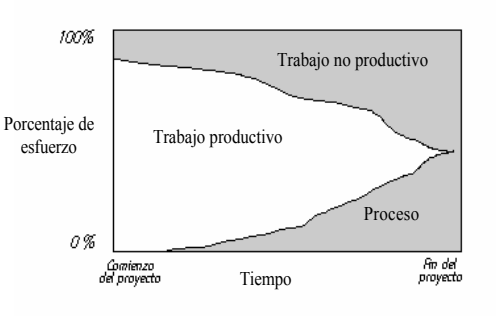


Figura 5: Proceso de desarrollo del software [Letelier 2002 A].

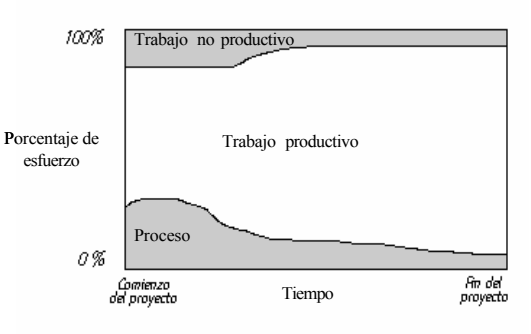
- **Quién:** Mediante los **Roles** que son asumidos por las **Personas** involucradas.
- **Qué**: Mediante los **Artefactos o Productos**, término general para cualquier tipo de información creada, producida, cambiada o utilizada por las Personas que han asumido los Roles. Los Artefactos llevan asociada una **Notación** dictada por el **Paradigma de desarrollo** utilizado.
- **Cómo y Cuándo**: Mediante las **Actividades**, serie de pasos que se deben realizar para generar un conjunto de Artefactos y su orden de ejecución lo indica el **Ciclo de vida** utilizado. Durante la Actividad se puede hacer uso de **Herramientas de apoyo** (por ejemplo: Herramientas CASE).

La estructuración y ejecución de las actividades debe realizarse a la luz de un conjunto de **Principios y Prácticas** que den lineamientos de cómo realizar estas actividades [Beck 2000]. Las Prácticas y Principios enfatizan el Cómo debe de realizarse una actividad (por ejemplo: Trabajo en grupo, uso de estándares, modelado visual, recodificación, etc.).

Una definición y especificación del proceso de desarrollo por ejecutar en un proyecto, permite que los miembros del equipo participante no pierdan tiempo en su organización y se concentren en las tareas por realizar. La especificación define qué productos deben de realizarse, quién es el responsable y en qué actividad o momento debe realizarse, además de la notación, estándares y herramientas en que debe apoyarse. El contar con esa información antes de empezar la ejecución del proyecto permite incrementar el tiempo productivo y disminuir el tiempo de organización o definición del proceso durante la ejecución, como se muestra en la Figura 6 (adaptada de [McConnell 1998]).



Ejecución de proyectos sin definición de proceso.



Ejecución de proyectos con definición de proceso.

Figura 6: Ejecución de proyectos sin definición de proceso vs. con definición de proceso.

1.5 Modelos de proceso de software

Sommerville [Sommerville 2002] define **modelo de proceso de software** como "Una representación simplificada de un proceso de software, representada desde una perspectiva específica. Por su naturaleza los modelos son simplificados, por lo tanto un modelo de procesos del software es una abstracción de un proceso real."

Los modelos genéricos no son descripciones definitivas de procesos de software, si bien son abstracciones útiles que pueden ser utilizadas para explicar diferentes enfoques del desarrollo de software. A continuación se discutirán algunos modelos genéricos:

- Codificar y Corregir
- Modelo de Cascada
- Desarrollo Evolutivo
- Desarrollo Formal de Sistemas
- Desarrollo basado en Reutilización
- Modelo de Desarrollo Incremental
- Modelo de Desarrollo en Espiral

1.5.1 Codificar y Corregir (*Code-and-Fix*)

Este es el modelo básico utilizado en los inicios del desarrollo de software. Contiene dos pasos:

- Escribir código.
- Corregir problemas en el código.

El orden de los pasos era elaborar algún código primero y luego pensar acerca de requerimientos, diseño, validación, y mantenimiento.

Este modelo tiene tres problemas principales [Boehm 1988]:

- Luego de cierta cantidad de correcciones, el código mal estructurado hace que los arreglos subsecuentes sean muy costosos. Esto provoca la necesidad de una fase de diseño antes de la codificación.
- Frecuentemente, aún el software bien diseñado no se ajusta a las necesidades del usuario, por lo que es rechazado o su reconstrucción es muy cara.
- El código es difícil de reparar ya que no posee una estructura que facilite probarlo y modificarlo.

1.5.2 Modelo de Cascada

Este fue el primer modelo de desarrollo de software que se publicó, derivado de otros procesos de ingeniería [Royce 1970]. Éste toma las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución y las representa como fases separadas del proceso.

El modelo de cascada consta de las siguientes fases:

1. Definición de los requerimientos: Los servicios, restricciones y objetivos son establecidos con los usuarios del sistema. Se busca hacer esta definición en detalle.
2. Diseño de software: Se particiona el sistema en sistemas de software o hardware. Se establece la arquitectura total del sistema. Se identifican y describen las abstracciones y relaciones de los componentes del sistema.
3. Implementación y prueba de unidades: Construcción de los módulos y unidades de software. Se realizan pruebas de cada unidad.
4. Integración y pruebas del sistema: Se integran todas las unidades. Se prueban en conjunto. Se entrega el conjunto probado al cliente.
5. Operación y mantenimiento: Generalmente es la fase más larga. El sistema es puesto en práctica y se realiza la corrección de errores descubiertos. Se realizan mejoras de implementación. Se identifican nuevos requerimientos.

La interacción entre fases puede observarse en la Figura 7. Cada fase tiene como resultado documentos que deben ser aprobados por el usuario.

Una fase no comienza hasta que termine la fase anterior y generalmente se incluye la corrección de los problemas encontrados en fases previas.

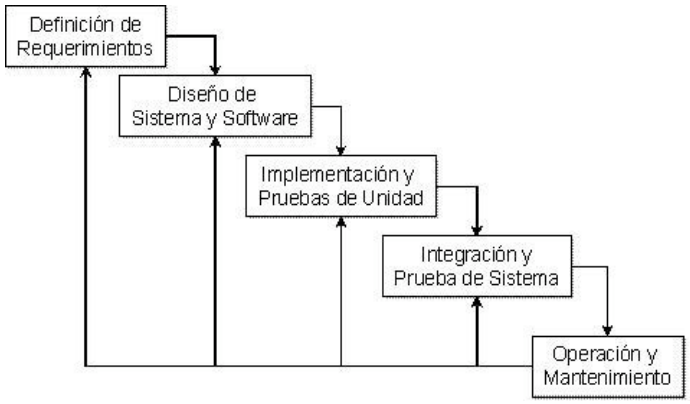


Figura 7: Modelo de desarrollo en cascada.

En la práctica, este modelo no es lineal, e involucra varias iteraciones e interacción entre las distintas fases de desarrollo.

Algunos problemas que se observan en el modelo de cascada son:

- Las iteraciones son costosas e implican rehacer trabajo debido a la producción y aprobación de documentos.
- Aunque son pocas iteraciones, es normal congelar parte del desarrollo y continuar con las siguientes fases.
- Los problemas se dejan para su posterior resolución, lo que lleva a que estos sean ignorados o corregidos de una forma poco elegante.
- Existe una alta probabilidad de que el software no cumpla oportunamente con los requerimientos del usuario por el largo tiempo de entrega del producto.
- Es inflexible a la hora de evolucionar para incorporar nuevos requerimientos.
- Es difícil responder a cambios en los requerimientos.

Este modelo sólo debe usarse si se entienden a plenitud los requerimientos; sin embargo, refleja la ingeniería del software y aún se utiliza como parte de proyectos grandes.

1.5.3 Modelo de Desarrollo Evolutivo

La idea detrás de este modelo es el desarrollo de una implantación del sistema inicial, exponerla a los comentarios del usuario, refinarla en n versiones hasta que se desarrolle el sistema adecuado. En la Figura 8 se observa cómo las actividades concurrentes especificación, desarrollo y validación se realizan durante el desarrollo de las versiones hasta llegar al producto final.

Una ventaja de este modelo es que se obtiene una rápida realimentación del usuario, ya que las actividades de especificación, desarrollo y pruebas se ejecutan en cada iteración.

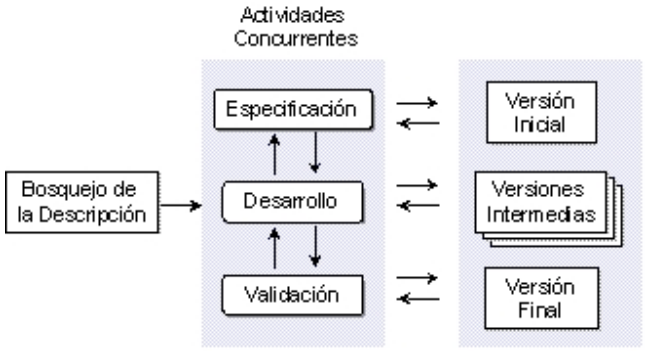


Figura 8: Modelo de desarrollo evolutivo.

Existen dos tipos de desarrollo evolutivo:

- Desarrollo Exploratorio: El objetivo de este enfoque es explorar con el usuario los requerimientos hasta llegar a un sistema final. El desarrollo comienza con las partes que se tienen más claras. El sistema evoluciona conforme se añaden nuevas características propuestas por el usuario.
- Enfoque utilizando prototipos: El objetivo es entender los requerimientos del usuario y trabajar para mejorar la calidad de los requerimientos. A diferencia del desarrollo exploratorio, se comienza por definir los requerimientos que no están claros para el usuario y se utiliza un prototipo para experimentar con ellos. El prototipo ayuda a terminar de definir estos requerimientos.

Entre los puntos favorables de este modelo están:

- La especificación puede desarrollarse de forma creciente.
- Los usuarios y desarrolladores logran un mejor entendimiento del sistema. Esto se refleja en una mejora de la calidad del software.
- Es más efectivo que el modelo de cascada, ya que cumple con las necesidades inmediatas del cliente.

Desde una perspectiva de ingeniería y administración se identifican los siguientes problemas:

- Proceso no visible: Los administradores necesitan entregas para medir el avance. Si el sistema se necesita desarrollar rápido, no es efectivo producir documentos que reflejen cada versión del sistema.
- Sistemas pobremente estructurados: Los cambios continuos pueden ser perjudiciales para la estructura del software, haciendo costoso el mantenimiento.
- Se requieren técnicas y herramientas: Para el desarrollo rápido se necesitan herramientas que pueden ser incompatibles con otras o que poca gente sabe utilizar.

Este modelo es efectivo en proyectos pequeños (menos de 100,000 líneas de código) o medianos (hasta 500,000 líneas de código) con poco tiempo para su desarrollo y sin generar documentación

para cada versión. Para proyectos grandes es mejor combinar lo mejor del modelo de cascada y evolutivo: se puede hacer un prototipo global del sistema y posteriormente reimplementarlo con un enfoque más estructurado. Los subsistemas con requerimientos bien definidos y estables se pueden programar utilizando cascada y la interfaz de usuario se puede especificar utilizando un enfoque exploratorio.

1.5.4 Desarrollo Formal de Sistemas

Este modelo se basa en pequeñas transformaciones formales de los requerimientos hasta llegar a un programa ejecutable. En la Figura 9 se observa las fases de este modelo.

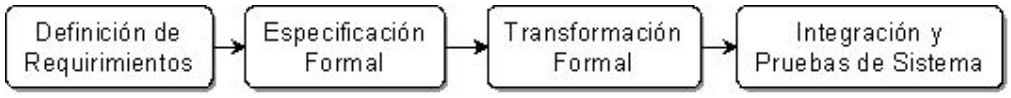


Figura 9: Modelo de desarrollo formal de sistemas.

Este tiene similitud con el modelo cascada, pero tiene dos diferencias sustanciales:

- Los requerimientos son definidos en una notación matemática formal.
- Las fases de diseño, implementación y pruebas son reemplazados por un proceso de transformación.

Observaciones sobre el desarrollo formal de sistemas:

- Permite demostrar la corrección del sistema durante el proceso de transformación.
- Es atractivo sobre todo para sistemas donde hay requerimientos de seguridad y confiabilidad importantes.
- No presenta desventajas ni en costo ni en calidad respecto de los otros modelos.
- Requiere desarrolladores especializados y experimentados en este proceso para llevarse a cabo eficientemente.

1.5.5 Desarrollo basado en reutilización

Como su nombre lo indica, es un modelo fuertemente orientado a la reutilización. Este modelo consta de 4 fases ilustradas en la Figura 10. A continuación se describe cada fase:

1. Análisis de componentes: Se determina qué componentes pueden ser utilizados para el sistema en cuestión. Casi siempre hay que hacer ajustes para adecuarlos.
2. Modificación de requerimientos: Se adaptan (en lo posible) los requerimientos para concordar con los componentes de la etapa anterior. Si no se puede realizar modificaciones en los requerimientos, hay que seguir buscando componentes más adecuados (fase 1).
3. Diseño del sistema con reutilización: Se diseña o reutiliza el marco de trabajo para el sistema. Se debe tener en cuenta los componentes localizados en la fase 2 para diseñar o

determinar este marco.

4. Desarrollo e integración: El software que no puede comprarse, se desarrolla. Se integran los componentes y subsistemas. La integración es parte del desarrollo en lugar de una actividad separada.

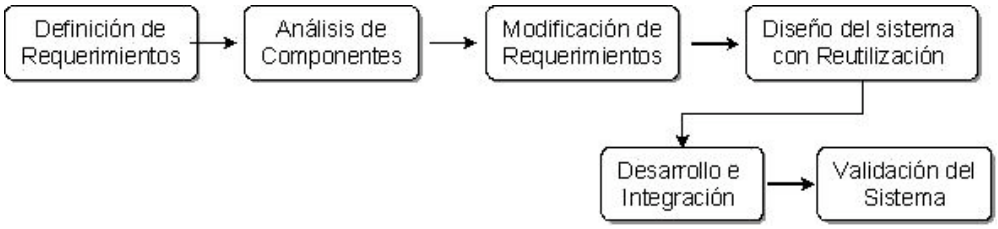


Figura 10: Modelo de desarrollo orientado a la reutilización.

Las ventajas de este modelo son:

- Disminuye el costo y esfuerzo de desarrollo.
- Reduce el tiempo de entrega.
- Disminuye los riesgos durante el desarrollo.
- Permite capitalizar experiencias y productos pasados, con posibilidad de aumentar confiabilidad y calidad de los desarrollos.

Desventajas de este modelo:

- Los acuerdos "de compromiso" en los requerimientos son inevitables, por lo cual puede que el software no cumpla las expectativas del cliente.
- Las actualizaciones de los componentes adquiridos no están en manos de los desarrolladores del sistema.

1.5.6 Procesos iterativos

A continuación se expondrán dos enfoques híbridos que están diseñados para el soporte de las iteraciones:

- Desarrollo Incremental.
- Desarrollo en Espiral.

Modelo de Desarrollo Incremental

Mills [Mills 1980] sugirió el enfoque incremental de desarrollo como una forma de reducir la repetición del trabajo en el proceso de desarrollo y dar oportunidad de posponer la toma de decisiones en los requerimientos hasta adquirir experiencia con el sistema (ver Figura 11). Es una combinación del Modelo de Cascada y Modelo Evolutivo.

Durante el desarrollo de cada incremento se puede utilizar el modelo de cascada o el evolutivo, dependiendo del conocimiento que se tenga sobre los requerimientos por implementar. Si se tiene un buen conocimiento de estos, se puede optar por cascada, si es dudoso, evolutivo.

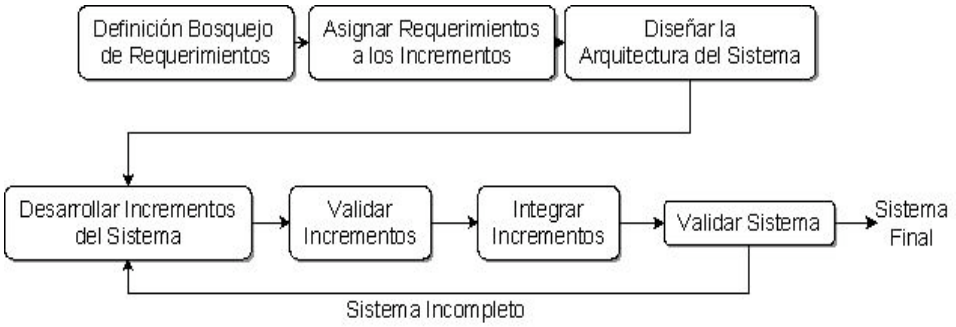


Figura 11: Modelo de desarrollo iterativo incremental.

Entre las ventajas del modelo incremental se encuentran:

- Los clientes no esperan hasta el fin del desarrollo para utilizar el sistema. Pueden empezar a usarlo desde el primer incremento.
- Los clientes pueden aclarar los requerimientos que no tengan claros conforme ven las entregas del sistema.
- Se disminuye el riesgo de fracaso de todo el proyecto, ya que se puede distribuir en cada incremento.
- Las partes más importantes del sistema son entregadas primero, por lo cual se realizan más pruebas en estos módulos y se disminuye el riesgo de fallos.

Algunas de las desventajas identificadas para este modelo son:

- Cada incremento debe ser pequeño para limitar el riesgo (menos de 20,000 líneas).
- Cada incremento debe aumentar la funcionalidad.
- Es difícil mapear los requerimientos contra los incrementos.
- Es difícil detectar las unidades o servicios genéricos para todo el sistema.

Modelo de Desarrollo en Espiral

El modelo de desarrollo en espiral (ver Figura 12) es actualmente uno de los más conocidos y fue propuesto por Barry Boehm [Boehm 1988]. El ciclo de desarrollo se representa como una espiral, en lugar de una serie de actividades sucesivas con retrospectiva de una actividad a otra.

Cada ciclo de desarrollo se divide en cuatro fases:

1. Definición de objetivos: Se definen los objetivos. Se definen las restricciones del proceso y

del producto. Se realiza un diseño detallado del plan administrativo. Se identifican los riesgos y se elaboran estrategias alternativas dependiendo de estos.

2. Evaluación y reducción de riesgos: Se realiza un análisis detallado de cada riesgo identificado. Pueden desarrollarse prototipos para disminuir el riesgo de requerimientos dudosos. Se llevan a cabo los pasos para reducir los riesgos.
3. Desarrollo y validación: Se escoge el modelo de desarrollo después de la evaluación del riesgo. El modelo que se utilizará (cascada, sistemas formales, evolutivo, etc.) depende del riesgo identificado para esa fase.
4. Planeación: Se determina si se continúa con otro ciclo. Se planea la siguiente fase del proyecto.

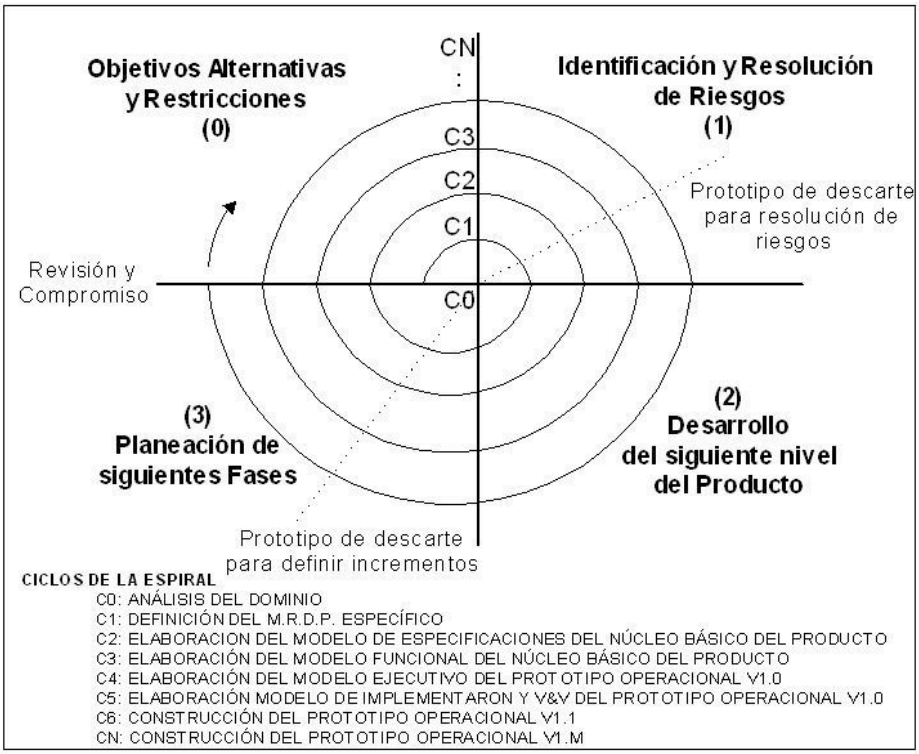


Figura 12: Una versión del Modelo de desarrollo en espiral de Boehm [Boehm 1988].

El transcurso del tiempo se representa en la espiral (ver Figura 12), alejándose del centro (punto C0).

Este modelo a diferencia de los otros, toma en consideración explícitamente el riesgo, que es un tema importante en la administración del proyecto.

El ciclo de vida inicia con la definición de los objetivos. De acuerdo con las restricciones se

determinan distintas alternativas. Se identifican los riesgos al sopesar los objetivos contra las alternativas. Se evalúan los riesgos con actividades como análisis detallado, simulación, prototipos, etc. Se desarrolla un poco el sistema. Se planea la siguiente fase.

1.5.7 ¿Cuál es el modelo de proceso más adecuado?

Cada proyecto de software requiere de una forma particular de abordar el problema. Las propuestas comerciales y académicas actuales hacen énfasis en procesos iterativos, donde en cada iteración puede utilizarse uno u otro modelo de proceso, tomando en cuenta un conjunto de criterios (Por ejemplo: grado de definición de requerimientos, tamaño del proyecto, riesgos identificados, presupuesto, entre otros).

En la Tabla 1 se expone un cuadro comparativo que toma en cuenta algunos criterios básicos para la selección de un modelo de proceso [Laboratorio Ing. Sof. 2002], la métrica utilizada indica el nivel de efectividad del modelo de proceso de acuerdo con el criterio (Por ejemplo: El modelo Cascada responde con un nivel de efectividad Bajo cuando los requerimientos y arquitectura no están predefinidos).

Modelo de proceso	Funciona con requerimientos y arquitectura no predefinidos	Produce software altamente fiable	Involucra gestión de riesgos	Permite correcciones sobre la marcha	Visión del avance por el Cliente y el Jefe del proyecto
Codificar y Corregir	Bajo	Bajo	Bajo	Alto	Medio
Cascada	Bajo	Bajo	Bajo	Bajo	Bajo
Evolutivo Prototipado	Alto	Medio	Medio	Alto	Alto
Desarrollo Formal de Sistemas	Bajo	Alto	Bajo a Medio	Bajo	Bajo
Desarrollo Orientado a Reutilización	Medio	Bajo a Alto	Bajo a Medio	Alto	Alto
Incremental	Medio o Alto	Alto	Alto	Medio o Alto	Medio o Alto
Espiral	Alto	Alto	Alto	Medio o Alto	Medio o Alto

Tabla 1: Comparación entre modelos de proceso de software.

1.6 Metodologías de desarrollo de software

Una metodología de desarrollo de software también es un modelo de proceso, pero no tan genérico como los casos anteriores ya que una metodología brinda más detalle no solo del flujo de actividades sino que define **Quién** debe hacer **Qué, Cuándo** y **Cómo** debe hacerlo.

Si tomamos como criterio los métodos de análisis y diseño utilizados en las metodologías (tanto comerciales como en el ámbito académico y de investigación), estas pueden agruparse en dos grandes corrientes: Metodologías Estructuradas y Metodologías Orientadas a Objetos.

1.6.1 Metodologías Estructuradas

Los métodos estructurados comenzaron a desarrollarse a fines de los 60's con la Programación Estructurada, luego a mediados de los 70's aparecieron técnicas para el Diseño primero y luego para el Análisis. Están enfocadas en implementaciones usando lenguajes de tercera generación.

Ejemplos de metodologías estructuradas de ámbito gubernamental: MERISE [6] (Francia), MÉTRICA 3 [7] (España), SSADM [8] (Reino Unido).

Ejemplos de propuestas de métodos estructurados en el ámbito académico y comercial: Gane & Sarson[9], Ward & Mellor[10], Yourdon & DeMarco[11] e Information Engineering[12].

1.6.2 Metodologías Orientadas a Objetos

Su historia va unida a la evolución de los lenguajes de programación orientada a objetos, los más representativos: a fines de los 60's SIMULA, a fines de los 70's Smalltalk-80, la primera versión de C++ por Bjarne Stroustrup en 1981 y actualmente Java [13] o C#. A fines de los 80's comenzaron a consolidarse algunos métodos de análisis y diseño Orientados a Objeto.

En 1995 Booch y Rumbaugh proponen el Método Unificado con la ambiciosa idea de conseguir una unificación de sus métodos y notaciones, que posteriormente se reorienta para dar lugar al *Unified Modeling Language* (UML)[14], la notación OO más popular en la actualidad, que incorpora importantes aportes de Jacobson y Odell.

Algunos métodos OO con notaciones predecesoras de UML son: OOAD (Booch), OOSE (Jacobson), Coad & Yourdon, Shaler & Mellor y OMT (Rumbaugh).

Algunas metodologías orientadas a objetos que utilizan la notación UML son: *Rational Unified Process* (RUP) [15], OPEN, MÉTRICA 3.

[6]Ver http://perso.club-internet.fr/brouardf/SGBDRmerise.htm (7.5.2005)
[7]Ver http://www.map.es/csi/metrica3/ (7.9.2005)
[8]Ver http://www.comp.glam.ac.uk/pages/staff/tdhutchings/chapter4.html (7.5.2005)
[9]Ver http://portal.newman.wa.edu.au/technology/12infsys/html/dfdnotes.doc (29.8.2005)
[10]Ver http://www.yourdon.com/books/coolbooks/notes/wardmellor.html (29.8.2005)
[11]Ver http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?Yourdon%2FDemarco (29.8.2005)
[12]Ver http://gantthead.com/Gantthead/process/processMain/1,1289,2-12009-2,00.html (29.8.2005)
[13]Ver http://java.sun.com/ (7.5.2005)
[14]Ver http://www.uml.org/ (7.5.2005)
[15]Ver http://www.rational.com/products/rup/index.jsp (7.5.2003)

1.6.3 Metodologías Ágiles

Por otro lado, actualmente podemos utilizar otro criterio para clasificar los procesos de desarrollo, observando aspectos tales como: grado de énfasis en actividades de modelado, el número de artefactos que utilizan, la cantidad y cualidades de los roles presentes en el proceso, número de iteraciones, entre otros. De acuerdo con estos y otros aspectos, han surgido dos categorías: Procesos Ágiles y Procesos No Ágiles (llamados peyorativamente "procesos pesados o ceremoniosos").

Un proceso es ágil cuando el desarrollo de software es **incremental** (entregas pequeñas de software, con ciclos rápidos), **cooperativo** (cliente y desarrolladores trabajan juntos constantemente con una cercana comunicación), **sencillo** (el método en sí mismo es fácil de aprender y modificar, bien documentado), y **adaptable** (permite realizar cambios de último momento) [Abrahamsson 2002].

Entre las metodologías ágiles identificadas en [Abrahamsson 2002] encontramos:

- *Extreme Programming* [Beck 2000].
- *Scrum* ([Schwaber 1995],[Schwaber 2002.]).
- Familia de Metodologías Crystal [Cockburn 2002].
- *Feature Driven Development* [Palmer 2002].
- *Proceso Unificado Rational*, una configuración ágil ([Kruchten 1996]).
- *Dynamic Systems Development Method* [Stapleton 1997].
- *Adaptive Software Development* [Highsmith 2000].
- *Open Source Software Development* [O'Reilly 1999].

Algunos procesos de desarrollado considerados no ágiles (de acuerdo a su configuración) son: *Proceso Unificado Rational* (RUP), METRICA 3, MERISE, SSADM.

En los siguientes capítulos se brinda una descripción de importantes metodologías como RUP y Métrica 3.0, que son metodologías robustas y de amplia difusión. Un futuro informe del Club de Investigación Tecnológica estará dedicado a las metodologías ágiles.

2 Rational Unified Process (RUP)

Este capítulo presenta un resumen con los conceptos básicos de *Rational Unified Process* o RUP Versión 2002.05.00, producto de proceso desarrollado y comercializado por *Rational Software Corporation*[16] (para mayores detalles consultar en [Kruchten 2000]). Cuando termine de leer este capítulo, usted:

- conocerá la evolución de la metodología RUP.
- conocerá las buenas prácticas de desarrollo de software en que se basa RUP.
- conocerá la estructura del proceso RUP en donde describe ciclo de vida, roles, actividades y productos.

RUP es un producto que define una metodología de desarrollo de software con un enfoque ingenieril; es una metodología robusta, que especifica la mayoría de elementos de un proceso de software (actividades, roles, productos, ciclo de vida, prácticas y principios, plantillas, herramientas, entre otros).

Su especificación permite realizar configuraciones de procesos según las características del proyecto tanto para proyectos grandes y riesgosos con fuerte captura de requerimientos, modelado, seguimiento y control, o para proyectos pequeños con configuraciones más livianas.

Ha sido tomada como base para la definición de metodologías más reducidas como la versión de Larman [Larman 2002] y versiones ágiles [Hirsch 2002].

Esta metodología captura muchas de las llamadas mejores prácticas (*best practices*), las cuales son prácticas modernas de desarrollo de software tales como desarrollo de software iterativo, manejo de requerimientos, arquitectura basada en componentes, modelado de software visual, verificación continua de la calidad y control de cambios. Estas serán explicadas con más detenimiento durante el desarrollo del presente capítulo.

2.1 Historia

RUP ha evolucionado y continúa evolucionando con el objetivo de acomodarse a las mejoras en las técnicas de desarrollo de software. La Figura 13 muestra los puntos importantes de su evolución.

[16] Ver http://www.rational.com/ (1.9.03)

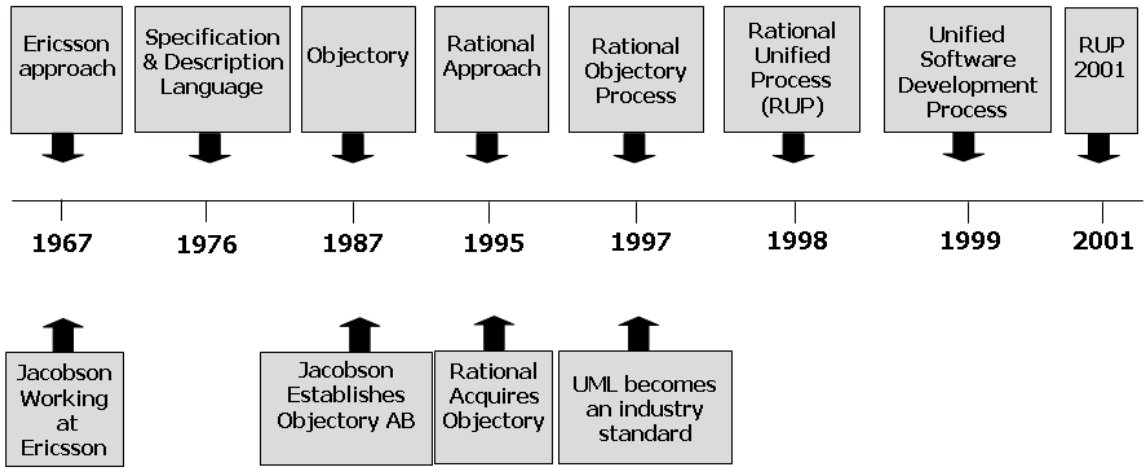


Figura 13: Evolución de RUP [Arlow 2002].

El antecedente más importante se ubica en 1967 con la Metodología Ericsson (*Ericsson Approach*). Ésta es una aproximación de desarrollo basada en componentes, que introdujo el concepto de caso de uso. En 1987 Jacobson fundó la compañía *Objectory AB* y lanza el proceso de desarrollo *Objectory* (abreviación de *Object Factory*).

Posteriormente en 1995 *Rational Software Corporation* adquiere *Objectory AB* y entre 1995 y 1997 se desarrolla *Rational Objectory Process* (ROP) fruto del encuentro y evolución de *Objectory* 3.8 y la Metodología Rational (*Rational Approach*) adoptando UML como lenguaje de modelado.

Desde ese entonces y con el liderazgo de Booch, Jacobson y Rumbaugh, Rational ha desarrollado e incorporado diversos elementos para expandir el ROP, destacándose especialmente el flujo de trabajo conocido como modelado del negocio, en junio del 1998 se lanza ***Rational Unified Process*** 5.0 evolucionado hasta el momento de elaboración de este documento bajo el nombre de RUP.

2.2 Características esenciales

Existen tres prácticas esenciales en las que se basa RUP, estas son proceso dirigido por casos de uso, proceso centrado en la arquitectura y proceso iterativo e incremental.

2.2.1 Proceso dirigido por casos de uso

Según [Jacobson 2000], los casos de uso son una técnica de captura de requerimientos que fuerza a pensar en términos de importancia para el usuario y no sólo en términos de funciones que sería bueno contemplar.

Se define un caso de uso como un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante; estos representan los requerimientos funcionales del sistema [Jacobson 2000].

En RUP los casos de uso no son sólo una herramienta para especificar los requerimientos del sistema. También guían su diseño, implementación y prueba, es decir, existe una integración y una guía del trabajo por medio de los casos de uso, como se muestra en la Figura 14.

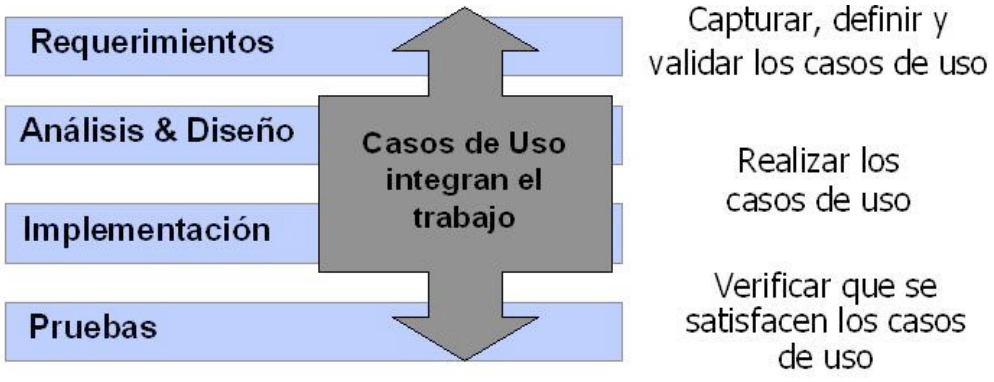


Figura 14: Los casos de uso integran y dirigen el trabajo [Letelier 2002 A].

Los casos de uso no sólo inician el proceso de desarrollo sino que proporcionan un hilo conductor [Jacobson 2000], esto permite una fuerte rastreabilidad entre los artefactos que son generados en las diferentes etapas del proceso de desarrollo a partir de los caso de uso.

Como se muestra en la Figura 15, basándose en los casos de uso se crean los modelos de análisis y diseño, luego la implementación que los lleva a cabo, y se verifica que efectivamente el producto implemente adecuadamente cada caso de uso. Todos los modelos deben estar acordes con el modelo de casos de uso.

Estos artefactos poseen una rastreabilidad que parte desde el reconocimiento del caso de uso, a la realización de los modelos de análisis y diseño, la implementación, las pruebas, y por último el mantenimiento del caso de uso.

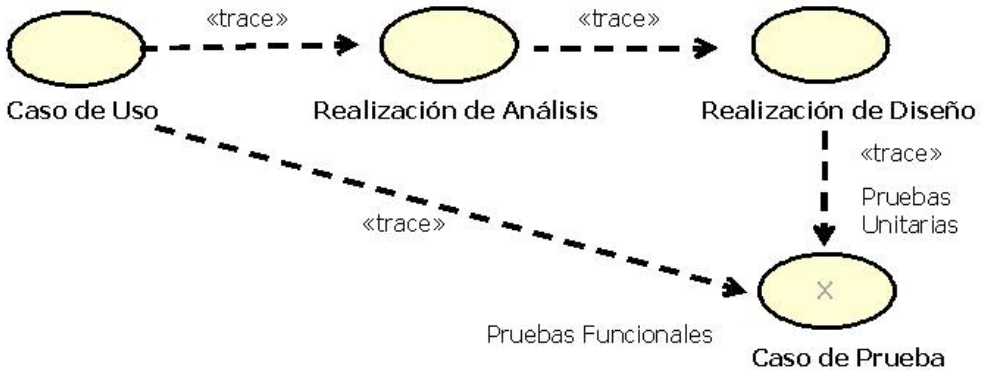


Figura 15: Rastreabilidad por medio de los casos de uso.

2.2.2 Proceso centrado en la arquitectura

La arquitectura de un sistema es la organización o estructura de sus partes más relevantes, lo que permite tener una visión común entre todos los involucrados (desarrolladores y otros usuarios) y una perspectiva clara del sistema completo, necesaria para controlar el desarrollo [Jacobson 2000].

La arquitectura involucra los aspectos estáticos y dinámicos más significativos del sistema, está relacionada con la toma de decisiones que indican cómo tiene que ser construido el sistema y ayuda a determinar en qué orden, además debe tomar en consideración elementos de calidad del sistema, rendimiento, reutilización y capacidad de evolución por lo que debe ser flexible durante todo el proceso de desarrollo.

La arquitectura se ve influida por la plataforma de software, sistema operativo, gestor de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados y requerimientos no funcionales.

En el caso del RUP, además de utilizar los casos de uso, para guiar el proceso es indispensable la arquitectura. Cada producto tiene tanto una función como una forma. La función corresponde a la funcionalidad reflejada en los casos de uso y la forma la proporciona la arquitectura. Existe una interacción entre los casos de uso y la arquitectura; los casos de uso deben encajar en la arquitectura cuando se llevan a cabo y la arquitectura debe permitir el desarrollo de todos los casos de uso requeridos, actualmente y en el futuro. Esto provoca que tanto arquitectura como casos de uso deban evolucionar en paralelo durante todo el proceso de desarrollo de software [Jacobson 2000].

Es conveniente ver el sistema desde diferentes perspectivas para comprender mejor el diseño. Por ello la arquitectura se representa mediante varias vistas que se centran en aspectos concretos del sistema, abstrayéndose de lo demás. Todas las vistas juntas forman el llamado modelo 4+1 de la arquitectura, el cual recibe este nombre porque lo forman las vistas lógica, de implementación, de proceso y de despliegue, más la de casos de uso que es la que da cohesión a todas. Ver Figura 16.

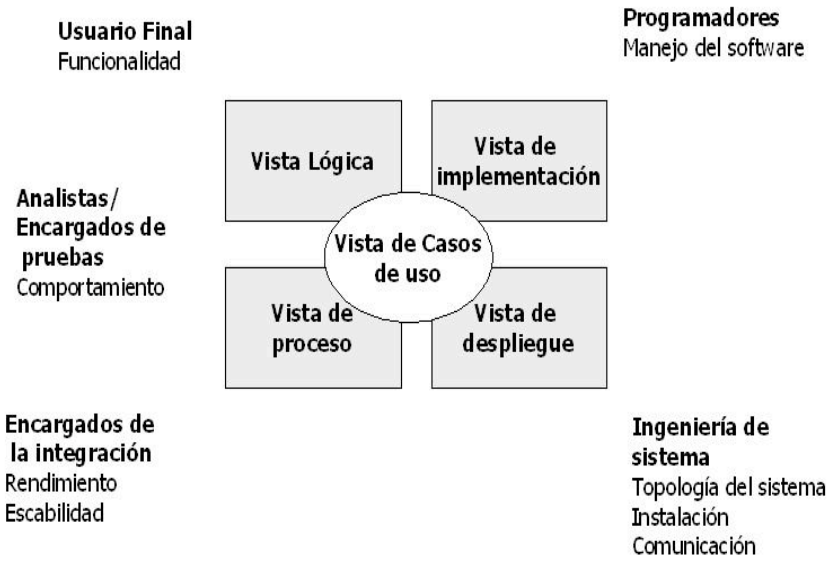


Figura 16: **Modelo de arquitectura 4+1 vistas [Kruchten 2000].**

Vista lógica: esta es la vista de la arquitectura que guía los requerimientos funcionales del sistema. Es una abstracción del modelo de diseño que identifica paquetes de diseño, subsistemas y clases.

Vista de implementación: esta vista describe la organización de los módulos de software estático (código fuente, archivos de datos, componentes, ejecutables entre otros) en el ambiente de desarrollo, en términos de empaquetamiento y capas y en términos de manejo de la configuración.

Vista de proceso: esta vista maneja el aspecto concurrente del sistema en tiempo de ejecución: tareas, hilos y procesos. Esto maneja asuntos como concurrencia y paralelismo, tolerancia a fallas y distribución de objetos.

Vista de despliegue: esta vista define la arquitectura física del sistema por medio de nodos interconectados, muestra cómo varios ejecutables y otros componentes son configurados según la plataforma.

Vista de casos de uso: esta vista desempeña un rol especial, contiene escenarios clave o casos de uso que son usados para descubrir y diseñar la arquitectura en las fases iniciales del proceso.

2.2.3 Proceso iterativo e incremental

Según [Jacobson 2000] el equilibrio correcto entre los casos de uso y la arquitectura es algo muy parecido al equilibrio de la forma y la función en el desarrollo del producto, lo cual se consigue con el tiempo.

La solución que se propone en RUP es tener un proceso iterativo e incremental en donde el trabajo se divide en partes más pequeñas o miniproyectos. Esto permite que el equilibrio entre casos de uso y arquitectura se vaya logrando durante cada miniproyecto, así durante todo el proceso de desarrollo [Jacobson 2000]. Cada mini proyecto se puede ver como una iteración (un recorrido más o menos completo a lo largo de todos los flujos de trabajo fundamentales) del cual se obtiene un incremento que produce un crecimiento en el producto, generalmente en términos de funcionalidad extendida.

Una iteración puede realizarse por medio de una cascada como se muestra en la Figura 17. Se pasa por los flujos fundamentales (Requerimientos, Análisis, Diseño, Implementación y Pruebas), también existe una planificación de la iteración, un análisis de la iteración y algunas actividades específicas de la iteración. Al finalizar se realiza una integración de los resultados con lo obtenido de las iteraciones anteriores.

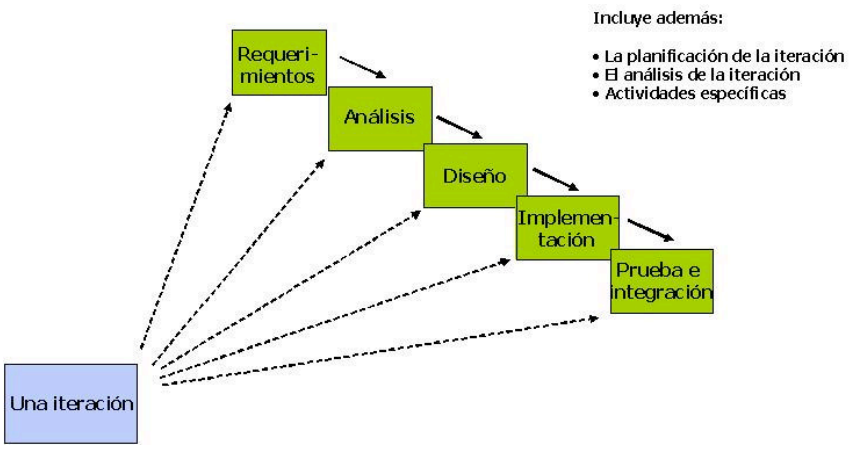


Figura 17: Una Iteración.

El proceso iterativo e incremental tiene una secuencia de iteraciones (como se muestra en la Figura 18) por lo que las iteraciones deben estar controladas, esto es, deben seleccionarse y ejecutarse de una forma planificada.

Cada iteración contempla una parte de la funcionalidad total, se pasa por los flujos de trabajo relevantes y se madura la arquitectura, esto permite tener bases más firmes para las siguientes iteraciones.

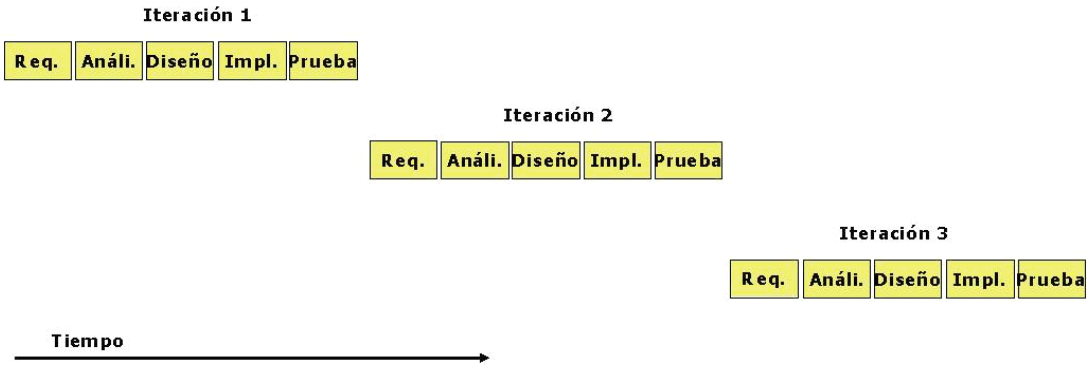


Figura 18: Secuencia de iteraciones.

Cada iteración se analiza cuando termina. Se puede determinar si han aparecido nuevos requerimientos o si han cambiado los existentes, afectando a las iteraciones siguientes. Durante la planificación de los detalles de la siguiente iteración, el equipo también examina cómo afectarán los riesgos que aún quedan afectarán al trabajo en curso. Toda la retroalimentación de la iteración

pasada permite reajustar los objetivos para las siguientes iteraciones.

Si una iteración cumple con los objetivos marcados se procede con la siguiente iteración. Cuando una iteración no cumple con sus objetivos, se debe analizar las decisiones previas y probar con otro enfoque. Se continúa con esta dinámica hasta que se haya finalizado por completo con la versión actual del producto.

En concreto, RUP divide el proceso en cuatro fases: Inicio, Elaboración, Construcción y Transición. En cada fase se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en los distintas actividades. En la Figura 19 se muestra cómo varía el esfuerzo asociado a las disciplinas según la fase en la que se encuentre el proyecto RUP.

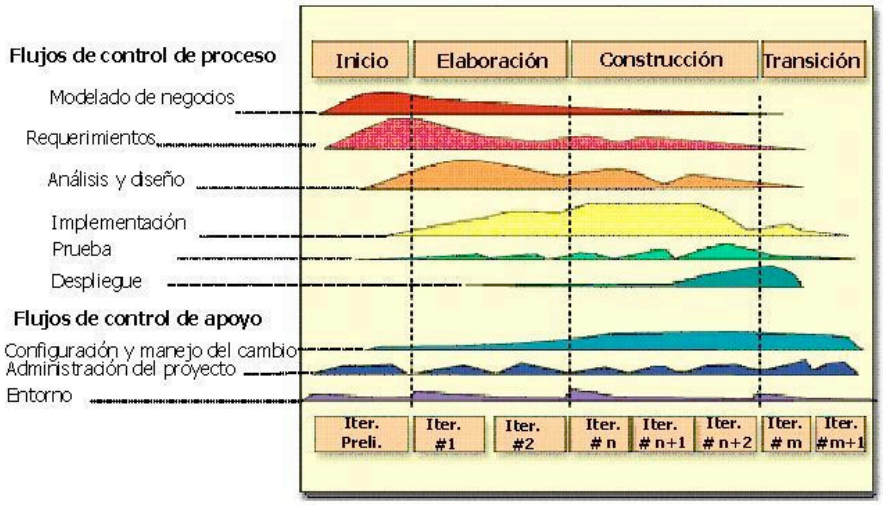


Figura 19: Fases e iteraciones de RUP.

Las primeras iteraciones (Inicio y Elaboración) tratan en su mayor parte la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de los riesgos críticos, y la creación de la línea base de la arquitectura.

Como se observa en el dibujo en la **fase de inicio**, las iteraciones se centran por producir un análisis del negocio y una determinación inicial de los requerimientos.

En la **fase de elaboración**, las iteraciones se orientan al desarrollo de la línea base de la arquitectura (esto permite definir una propuesta de arquitectura y probarla antes de entrar a la fase de construcción), abarcan además los flujos de trabajo de requerimientos, modelo de negocios

(refinamiento), análisis, diseño y una parte de implementación.

En la **fase de construcción**, se lleva a cabo la construcción del producto (programación, definición de bases de datos, pruebas e integración) por medio de una serie de iteraciones. Cada iteración genera un incremento en el producto final.

Para cada iteración se selecciona algunos casos de uso, se refina su análisis y diseño y se procede a su implementación y pruebas. Se realiza una pequeña cascada para cada ciclo. Se realizan tantas iteraciones como las que se requieran para implementar la nueva versión del producto.

En la **fase de transición** se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios. Esto incluye actividades como terminar los instaladores, manuales de usuario, guías de capacitación, etc.

Como se puede observar, en cada fase participan todas las disciplinas pero, dependiendo de la fase, el esfuerzo dedicado a una disciplina varía.

2.3 Buenas prácticas

RUP identifica 6 mejores prácticas (*best practices*) con las que define una forma efectiva de trabajar para los equipos de desarrollo de software.

2.3.1 Manejo de requerimientos

RUP brinda una guía para encontrar, organizar, documentar, y controlar los cambios de los requerimientos funcionales y restricciones. Utiliza una notación de casos de uso y escenarios para representar los requerimientos.

2.3.2 Desarrollo basado en componentes

La creación de sistemas intensivos en software requiere dividir el sistema en componentes con interfaces bien definidas, que posteriormente serán ensamblados para generar el sistema. Esta característica en un proceso de desarrollo permite que el sistema se vaya creando a medida que se obtienen o que se desarrollen y maduren sus componentes.

2.3.3 Desarrollo de software iterativo

Esto se describió en el apartado Proceso iterativo e incremental.

2.3.4 Modelado visual

UML es adoptado como único lenguaje de modelado para el desarrollo de todos los modelos. UML define un lenguaje gráfico para presentar modelos y define la semántica para cada elemento

gráfico. Utilizar herramientas de modelado visual facilita la gestión de dichos modelos, permitiendo ocultar o exponer detalles cuando sea necesario. El modelado visual también ayuda a mantener la consistencia entre los artefactos del sistema: requerimientos, diseños e implementaciones. En resumen, el modelado visual ayuda a mejorar la capacidad del equipo para gestionar la complejidad del software, así como para comunicarse entre sí y con otros interlocutores. Ver el informe del club de Investigación Tecnológica sobre UML 1.0 [Luna 2000]

2.3.5 Verificación continua de la calidad

Los problemas del software son de 100 a 1000 veces más caros de encontrar y corregir en la etapa de mantenimiento. Por ello es importante verificar la calidad del software durante todo su ciclo de vida.

La calidad de todos los productos debe ser asegurada en varios puntos del ciclo de vida. Un producto debe ser evaluado en la actividad que lo elabora y en la conclusión de la iteración donde fue elaborado.

En RUP el manejo de la calidad es implementado en todos sus flujos de trabajo, fases e iteraciones. En general manejar la calidad a través del ciclo de vida significa implementar, medir y asegurar la calidad del proceso y de los productos.

2.3.6 Control de cambios en el software

Cuando se está desarrollando software en donde se tiene múltiples desarrolladores, diferentes equipos de trabajo posiblemente organizados en varios lugares, trabajando en múltiples iteraciones, productos y plataformas, es indispensable contar con una disciplina de control que permita coordinar las actividades y productos, coordinar las iteraciones y controlar los cambios.

Los cambios deben ser aceptados y dárseles un seguimiento, por lo que el proceso de desarrollo debe controlar, seguir y monitorear los cambios en el software.

2.4 Estructura del proceso

El proceso puede ser descrito en dos dimensiones o ejes:

Eje horizontal: Representa el tiempo y es considerado el eje de los aspectos dinámicos del proceso. Indica las características del ciclo de vida del proceso expresado en términos de fases, iteraciones e hitos. Se puede observar en la Figura 20 que RUP consta de cuatro fases: **Inicio**, **Elaboración, Construcción** y **Transición**. Como se mencionó anteriormente cada fase se subdivide a la vez en iteraciones.

Eje vertical: Representa los aspectos estáticos del proceso. Describe el proceso en términos de componentes de proceso, disciplinas, flujos de trabajo, actividades, artefactos y roles.

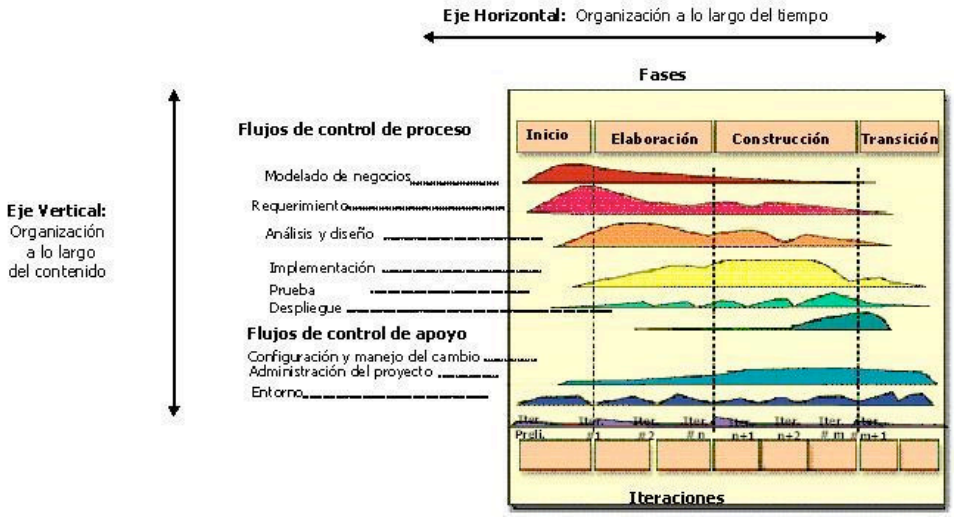


Figura 20: Dimensiones de RUP.

2.4.1 Estructura dinámica del proceso. Fases e iteraciones

RUP se repite a lo largo de una serie de ciclos que constituyen la vida de un producto. Cada ciclo concluye con una versión del producto para los clientes.

Cada ciclo consta de cuatro fases mencionadas anteriormente: **Inicio**, **Elaboración**, **Construcción** y **Transición**. Cada fase se subdivide a la vez en iteraciones, cuyo número no es fijo (ver Figura 21).

Cada fase se concluye con un hito bien definido, un punto en el tiempo en el cual se deben tomar ciertas decisiones críticas y alcanzar las metas clave antes de pasar a la siguiente fase. Ese **hito principal** de cada fase se compone de **hitos menores** que podrían ser los criterios aplicables a cada iteración.

Los hitos principales para cada una de las fases son: Inicio – **Objetivos del ciclo de vida** (*Lifecycle Objectives*), Elaboración – **Arquitectura del ciclo de vida** (*Lifecycle Architecture*), Construcción – **Capacidad operativa inicial** (*Initial Operational Capability*), Transición – **Liberación del producto** (*Product Release*).

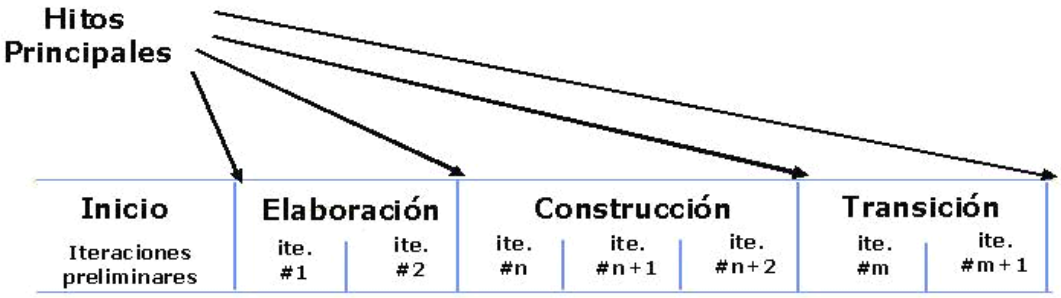


Figura 21: Fases, iteraciones e hitos.

A continuación se explica cada una de estas fases.

Inicio

Durante la fase de inicio se define el modelo del negocio y el alcance del proyecto. Se identifican todos los actores y casos de uso, y se diseñan los casos de uso más esenciales (típicamente el 20% del modelo completo). Se desarrolla un plan de negocio para determinar qué recursos deben ser asignados al proyecto [Kruchten 2000].

Los objetivos de la fase de inicio son [Kruchten 2000]:

- Establecer el ámbito del proyecto y sus límites.

- Encontrar los casos de uso críticos del sistema, los escenarios básicos que definen la funcionalidad.
- Mostrar al menos una arquitectura candidata para los escenarios principales.
- Estimar el costo en recursos y tiempo de todo el proyecto.
- Estimar los riesgos, las fuentes de incertidumbre.

Los resultados de la fase de inicio deben ser [Rational1998]:

- Un documento de visión: Una visión general de los requerimientos del proyecto, características clave y restricciones principales.
- Modelo inicial de casos de uso (10-20% completado).
- Un glosario inicial: Terminología clave del dominio.
- El caso de negocio.
- Lista de riesgos y plan de contingencia.
- Plan del proyecto, mostrando fases e iteraciones.
- Modelo de negocio, si es necesario
- Prototipos exploratorios para probar conceptos o la arquitectura candidata.

Al terminar la fase de inicio se deben comprobar los criterios de evaluación para continuar [Rational1998]:

- Todos los interesados en el proyecto coinciden en la definición del ámbito del sistema y las estimaciones de calendario.
- Entendimiento de los requerimientos, como evidencia de la fidelidad de los casos de uso principales.
- Las estimaciones de tiempo, costo y riesgo son creíbles.
- Comprensión total de cualquier prototipo de la arquitectura desarrollado.
- Los gastos hasta el momento se asemejan a los planeados.

Si el proyecto no pasa estos criterios hay que plantearse abandonarlo o repensarlo profundamente.

Elaboración

El propósito de la fase de elaboración es analizar el dominio del problema, establecer los cimientos de la arquitectura, desarrollar el plan del proyecto y eliminar los mayores riesgos [Kruchten 2000].

En esta fase se construye un prototipo de la arquitectura, que debe evolucionar en iteraciones sucesivas hasta convertirse en el sistema final. Este prototipo debe contener los casos de uso críticos identificados en la fase de inicio. También debe demostrarse que se han evitado los riesgos más graves.

Los objetivos de esta fase son [Kruchten 2000]:

- Definir, validar y cimentar la arquitectura.
- Completar la visión.
- Crear un plan fiable para la fase de construcción. Este plan puede evolucionar en sucesivas iteraciones. Debe incluir los costos si procede.
- Demostrar que la arquitectura propuesta soportará la visión con un costo razonable y en un tiempo razonable.

Al terminar la fase de elaboración deben obtenerse los siguientes resultados [Rational1998]:

- Un modelo de casos de uso completo al menos hasta el 80%: todos los casos y actores identificados, la mayoría de los casos desarrollados.
- Requerimientos adicionales que capturan los requerimientos no funcionales y cualquier requerimiento no asociado con un caso de uso específico.
- Descripción de la arquitectura del software.
- Un prototipo ejecutable de la arquitectura.
- Lista de riesgos y caso de negocio revisados.
- Plan de desarrollo para el proyecto.
- Un ejemplo de desarrollo actualizado que especifica el proceso por seguir.
- Un manual de usuario preliminar (opcional).

En esta fase se debe tratar de abarcar todo el proyecto con la profundidad mínima. Sólo se profundiza en los puntos críticos de la arquitectura o riesgos importantes.

En la fase de elaboración se actualizan todos los productos de la fase de inicio.

Los criterios de evaluación de esta fase son los siguientes [Rational1998]:

- La visión del producto es estable.
- La arquitectura es estable.
- Se ha demostrado mediante la ejecución del prototipo que los principales elementos de riesgo han sido abordados y resueltos.
- El plan para la fase de construcción es detallado y preciso. Las estimaciones son creíbles.
- Todos los interesados coinciden en que la visión actual será alcanzada si se siguen los planes actuales en el contexto de la arquitectura actual.
- Los gastos hasta ahora son aceptables, comparados con los previstos.

Si no se superan los criterios de evaluación quizá sea necesario abandonar el proyecto o replanteárselo considerablemente.

Construcción

La finalidad principal de esta fase es alcanzar la capacidad operacional del producto de forma incremental a través de las sucesivas iteraciones. Durante esta fase todos los componentes, características y requerimientos deben ser implementados, integrados y probados en su totalidad, obteniendo una versión aceptable del producto.

Los objetivos concretos según [Kruchten 2000] incluyen:

- Minimizar los costos de desarrollo mediante la optimización de recursos y evitando el tener que rehacer un trabajo o incluso desecharlo.
- Conseguir una calidad adecuada tan rápido como sea práctico.
- Conseguir versiones funcionales (alfa, beta, y otras versiones de prueba) tan rápido como sea práctico.

Los resultados de la fase de construcción según [Rational1998] deben ser:

- Modelos completos (Casos de Uso, Análisis, Diseño, Despliegue e Implementación).
- Arquitectura íntegra (mantenida y mínimamente actualizada).
- Riesgos presentes mitigados.

- Plan del proyecto para la Fase de Transición.
- Manual inicial de usuario (con suficiente detalle).
- Prototipo operacional – beta.
- Caso del negocio actualizado.

Los criterios de evaluación de esta fase son los siguientes [Rational1998]:

- El producto es estable y maduro como para ser entregado a la comunidad de usuarios para ser probado.
- Todos los usuarios expertos están listos para la transición en la comunidad de usuarios.
- Son aceptables los gastos reales versus los gastos planeados.

Transición

La finalidad de la fase de transición es poner el producto en manos de los usuarios finales, para lo que se requiere desarrollar nuevas versiones actualizadas del producto, completar la documentación, entrenar al usuario en el manejo del producto y, en general, tareas relacionadas con el ajuste, configuración, instalación y usabilidad del producto.

En [Kruchten 2000] se citan algunas de las cosas que puede incluir esta fase:

- Prueba de la versión beta para validar el nuevo sistema frente a las expectativas de los usuarios.
- Funcionamiento paralelo con los sistemas legados que están siendo sustituidos por el producto de nuestro proyecto.
- Conversión de las bases de datos operacionales.
- Entrenamiento de los usuarios y técnicos de mantenimiento.
- Si el proyecto es a lo interno se entrega a los responsables de la producción y operación, de lo contrario se hace el traspaso del producto a los equipos de mercadeo, distribución y venta.

Los principales objetivos de esta fase son:

- Conseguir que el usuario se valga por sí mismo.
- Un producto final que cumpla los requerimientos esperados, que funcione y satisfaga suficientemente al usuario.

Los resultados de la Fase de Transición según [Rational1998] son:

- Prototipo operacional.
- Documentos legales.
- Caso del negocio completo.
- Línea de base del producto completa y corregida que incluye todos los modelos del sistema.
- Descripción de la arquitectura completa y corregida.
- Las iteraciones de esta fase irán dirigidas normalmente a conseguir una nueva versión.

Los criterios de evaluación de esta fase son los siguientes [Rational1998]:

- El usuario se encuentra satisfecho.
- Son aceptables los gastos reales versus los gastos planeados.

2.4.2 Estructura Estática del proceso. Roles, actividades, productos y flujos de trabajo

Un proceso de desarrollo de software define quién hace qué, cómo y cuándo. RUP define cuatro elementos para resolver esto: los **roles**, que responden a la pregunta **¿Quién?**, las **actividades** que responden a la pregunta **¿Cómo?**, los **productos**, que responden a la pregunta **¿Qué?** y los **flujos de trabajo** de las disciplinas que responde a la pregunta **¿Cuándo?** (ver Figura 22) [Rational1998].

Figura 22: Roles, actividades y productos.

Roles

Un rol define el comportamiento y las responsabilidades de un individuo, o de un grupo de individuos trabajando juntos como un equipo. Una persona puede desempeñar diversos roles, así como un mismo rol puede ser representado por varias personas.

Las responsabilidades de un rol son tanto el llevar a cabo un conjunto de actividades como el ser el dueño de un conjunto de artefactos.

RUP define grupos de roles, agrupados por su participación en actividades relacionadas. Estos grupos son [Rational2002]: Grupo Analista, Grupo Desarrollador, Grupo Especialista en pruebas, Grupo Gestor, Grupo Otros roles.

Actividades

Una actividad en concreto es una unidad de trabajo cuya realización puede solicitarse a una persona que desempeña un rol. Las actividades tienen un objetivo concreto, normalmente expresado en términos de crear o actualizar algún producto.

Productos

Un producto o artefacto es un trozo de información que es producido, modificado o usado por un proceso. Los productos son los resultados tangibles del proyecto, las cosas que va creando y usando hasta obtener el producto final.

Un artefacto puede ser cualquiera de los siguientes [Rational2002]:

- Un documento, como el documento de la arquitectura del software.
- Un modelo, como el modelo de casos de uso o el modelo de diseño.
- Un elemento del modelo, un elemento que pertenece a un modelo como una clase, un caso de uso o un subsistema.

Flujos de trabajo

Con la enumeración de roles, actividades y artefactos no se define un proceso, necesitamos contar con una secuencia de actividades realizadas por los diferentes roles, así como la relación entre ellos. Un flujo de trabajo es una relación de actividades que nos producen unos resultados observables. A continuación se dará una breve explicación de cada flujo de trabajo.

Modelado de negocios

Con este flujo de trabajo pretendemos llegar a un mejor entendimiento de la organización donde se va a implantar el producto. A continuación se muestra un diagrama de actividad que muestra la secuencia de ejecución de este flujo.

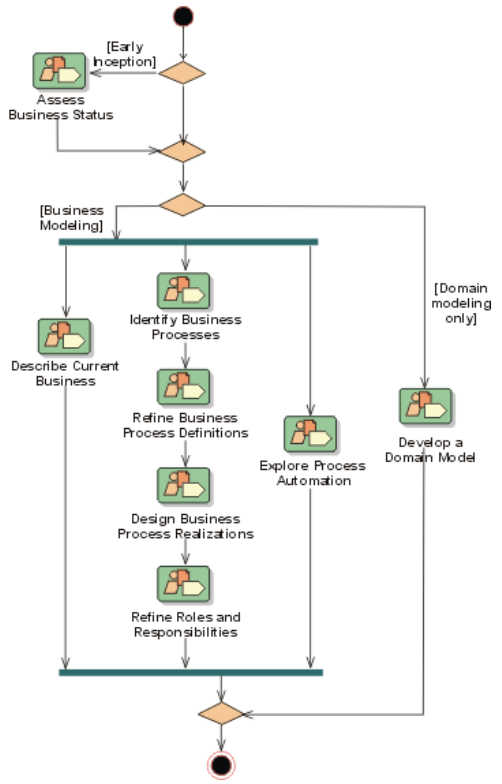


Figura 23: Secuencia de ejecución del Flujo de Trabajo de Modelado de negocios.

Los objetivos del modelado de negocio son [Rational2002]:

- Entender la estructura y la dinámica de la organización para la cual el sistema va ser desarrollado (organización objetivo).
- Entender el problema actual en la organización objetivo e identificar potenciales mejoras.
- Asegurar que clientes, usuarios finales y desarrolladores tengan un entendimiento común de la organización objetivo.
- Derivar los requerimientos del sistema necesarios para apoyar a la organización objetivo.

Para lograr estos objetivos, el modelo de negocio describe cómo desarrollar una visión de la nueva organización, con base en esta visión se definen procesos, roles y responsabilidades de la organización por medio de un modelo de casos de uso del negocio y un modelo de objetos del negocio.

Complementario a esos modelos, se desarrollan una Especificación suplementaria del negocio y un

Glosario.

Requerimientos

Este es uno de los flujos de trabajo más importantes, porque en él se establece qué tiene que hacer exactamente el sistema que construyamos. En esta línea los requerimientos son el contrato que se debe cumplir, de modo que los usuarios finales tienen que comprender y aceptar los requerimientos que especifiquemos. A continuación se muestra un diagrama de actividad que muestra la secuencia de ejecución.

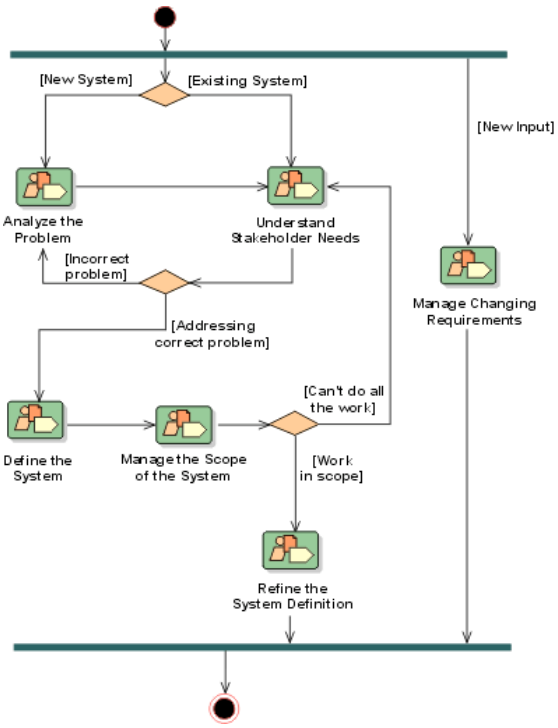


Figura 24: Secuencia de ejecución del Flujo de Trabajo de Requerimientos.

Los objetivos del flujo de trabajo de requerimientos son [Rational2002]:

- Establecer y mantener un acuerdo entre clientes y otros interesados (*stakeholders*) sobre lo que el sistema podría hacer.
- Proveer a los desarrolladores un mejor entendimiento de los requerimientos del sistema.
- Definir el ámbito del sistema.

- Proveer una base para la planeación de los contenidos técnicos de las iteraciones.
- Proveer una base para estimar costos y tiempo de desarrollo del sistema.
- Definir una interfaz de usuarios para el sistema, enfocada a las necesidades y metas del usuario.

Los requerimientos se dividen en dos grupos: los requerimientos funcionales y requerimientos no funcionales.

Los requerimientos funcionales representan la funcionalidad del sistema. Se modelan mediante diagramas de casos de uso.

Los requerimientos no funcionales representan aquellos atributos que debe exhibir el sistema, pero que no son una funcionalidad específica; por ejemplo requerimientos de usabilidad, fiabilidad, eficiencia, portabilidad, etc.

Para capturar los requerimientos es preciso entrevistar a todos los interesados en el proyecto, no sólo a los usuarios finales, y anotar todas sus peticiones. A partir de ellas hay que descubrir lo que necesitan y expresarlo en forma de requerimientos.

Análisis y Diseño

El objetivo de este flujo de trabajo es traducir los requerimientos a una especificación que describe cómo implementar el sistema.

Los objetivos del análisis y diseño son [Rational2002]:

- Transformar los requerimientos al diseño del futuro sistema.
- Desarrollar una arquitectura para el sistema.
- Adaptar el diseño para que sea consistente con el entorno de implementación, diseñando para el rendimiento.

A continuación se muestra un diagrama de actividad que muestra la secuencia de ejecución.

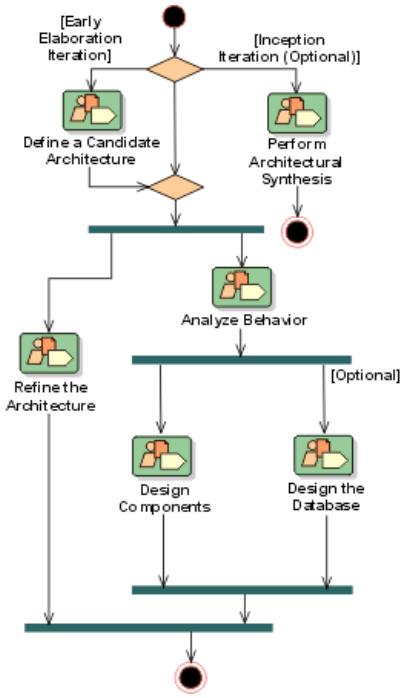


Figura 25: Secuencia de ejecución del Flujo de Trabajo de Análisis y diseño.

El análisis consiste en obtener una visión del sistema que se preocupa por qué hace éste, de modo que sólo se interesa por los requerimientos funcionales. Por otro lado el diseño es un refinamiento del análisis que tiene en cuenta los requerimientos no funcionales, en definitiva cómo cumple el sistema sus objetivos.

Al principio de la fase de elaboración hay que definir una arquitectura candidata: crear un esquema inicial de la arquitectura del sistema, identificar clases de análisis y actualizar las realizaciones de los casos de uso con las interacciones de las clases de análisis. Durante la fase de elaboración se va refinando esta arquitectura hasta llegar a su forma definitiva.

En cada iteración hay que analizar el comportamiento para diseñar componentes. Además si el sistema usará una base de datos, habrá que diseñarla también, obteniendo un modelo de datos.

El resultado final más importante de este flujo de trabajo será el modelo de diseño. Consiste en colaboraciones de clases, que pueden ser agregadas en paquetes y subsistemas.

Otro producto importante de este flujo es la documentación de la arquitectura de software, que captura varias visiones arquitectónicas del sistema.

Implementación

En este flujo de trabajo se implementan las clases y objetos en archivos fuente, binarios, ejecutables y demás. Además se deben hacer las pruebas de unidad: cada desarrollador es responsable de probar las unidades que produzca. El resultado final de este flujo de trabajo es un sistema ejecutable. A continuación se muestra un diagrama de actividad que muestra la secuencia de ejecución.

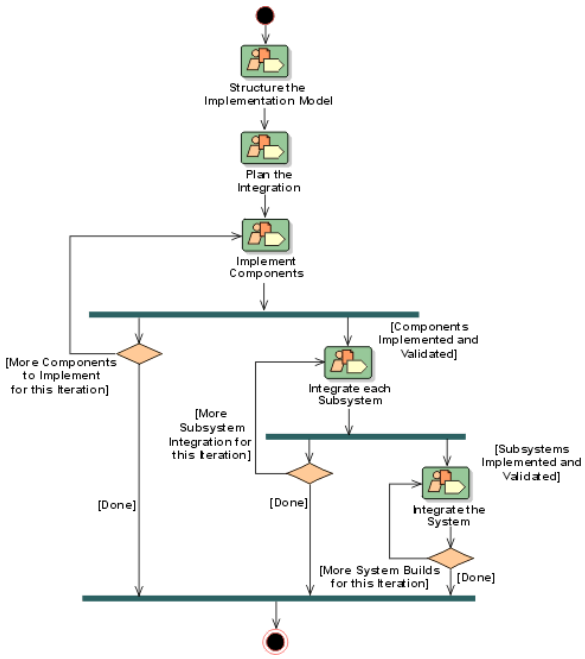


Figura 26: Secuencia de ejecución del Flujo de Trabajo de Implementación

En cada iteración habrá que hacer lo siguiente:

- Planear qué subsistemas deben ser implementados y en qué orden deben ser integrados, formando el Plan de integración.
- Cada desarrollador decide en qué orden implementa los elementos del subsistema.
- Si encuentra errores de diseño, los notifica.
- Se prueban los subsistemas individualmente.
- Se integra el sistema siguiendo el plan.

La estructura de todos los elementos implementados forma el modelo de implementación. La

integración debe ser incremental, es decir, en cada momento sólo se añade un elemento. De este modo es más fácil localizar fallos y los componentes se prueban más a fondo. En fases tempranas del proceso se pueden implementar prototipos para reducir el riesgo. Su utilidad puede ir desde ver si el sistema es viable desde el principio, probar tecnologías o diseñar la interfaz de usuario. Los prototipos pueden ser exploratorios (desechables) o evolutivos. Estos últimos llegan a transformarse en el sistema final.

Pruebas

Este flujo de trabajo es el encargado de evaluar la calidad del producto que estamos desarrollando, pero no para aceptar o rechazar el producto al final del proceso de desarrollo, sino integrando las actividades de pruebas en todo el ciclo de vida. A continuación se muestra un diagrama de actividad que muestra la secuencia de ejecución.

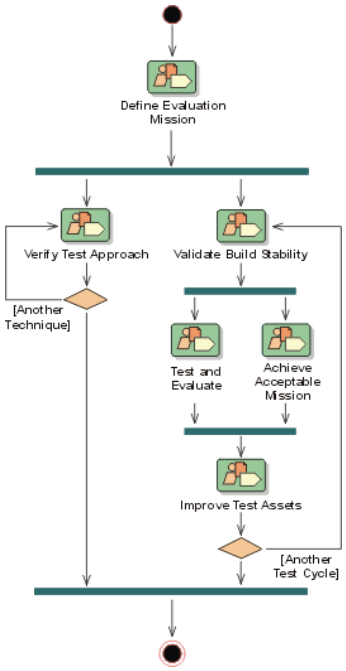


Figura 27: Secuencia de ejecución del Flujo de Trabajo de Pruebas.

Esta disciplina brinda soporte a las otras disciplinas. Sus objetivos son [Rational2002]:

- Encontrar y documentar defectos en la calidad del software.
- Asesorar sobre la calidad del software percibida.
- Provee la validación de los supuestos realizados en el diseño y especificación de

requerimientos por medio de demostraciones concretas.

- Verificar las funciones del producto de software según lo diseñado.
- Verificar que los requerimientos tengan su apropiada implementación.

Las actividades de este flujo comienzan temprano en el proyecto con el plan de pruebas (el cual contiene información sobre los objetivos generales y específicos de las pruebas en el proyecto, así como las estrategias y recursos con que se dotará a esta tarea), o incluso antes con alguna evaluación durante la fase de inicio, y continuará durante todo el proyecto.

El desarrollo del flujo de trabajo consistirá en planificar qué es lo que hay que probar, diseñar cómo se va a hacer, implementar lo necesario para llevar a cabo los procedimientos y casos de prueba, ejecutarlos en los niveles necesarios y obtener los resultados, de forma que la información obtenida nos sirva para ir refinando el producto por desarrollar.

Despliegue

El objetivo de este flujo de trabajo es producir con éxito distribuciones del producto y hacer llegar a los usuarios. A continuación se muestra un diagrama de actividad que muestra la secuencia de ejecución.

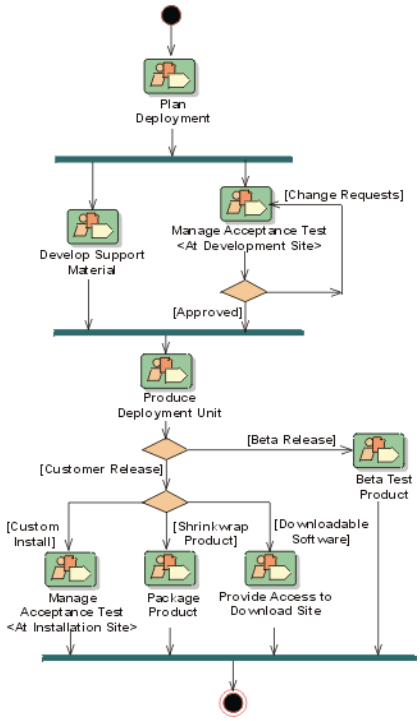


Figura 28: Secuencia de ejecución del Flujo de Trabajo de Despliegue.

Las actividades implicadas incluyen:

- Probar el producto en su entorno de ejecución final.
- Empaquetar el software para su distribución.
- Distribuir el software.
- Instalar el software.
- Proveer asistencia y ayuda a los usuarios.
- Formar a los usuarios y al cuerpo de ventas.
- Migrar el software existente o convertir bases de datos.
- Integrar el software con la plataforma operativa y otras aplicaciones.

Este flujo de trabajo se desarrolla con mayor intensidad en la fase de transición, ya que el propósito del flujo es asegurar una aceptación y adaptación sin complicaciones del software por parte de los usuarios. Su ejecución inicia en fases anteriores, para preparar el camino, sobre todo con actividades de planificación, en la elaboración del manual de usuario y tutoriales.

Administración del proyecto

La Gestión del proyecto es el arte de lograr un balance al gestionar objetivos, riesgos y restricciones para desarrollar un producto que sea acorde a los requerimientos de los clientes y los usuarios.

Los objetivos de este flujo de trabajo son:

- Proveer un marco de trabajo para la gestión de proyectos intensivos en software.
- Proveer guías prácticas realizar planeación, contratar personal, ejecutar y monitorear el proyecto.
- Proveer un marco de trabajo para gestionar riesgos.

La planeación de un proyecto posee dos niveles de abstracción: un plan para las fases y un plan para cada iteración.

A continuación se muestra un diagrama de actividad que muestra la secuencia de ejecución.

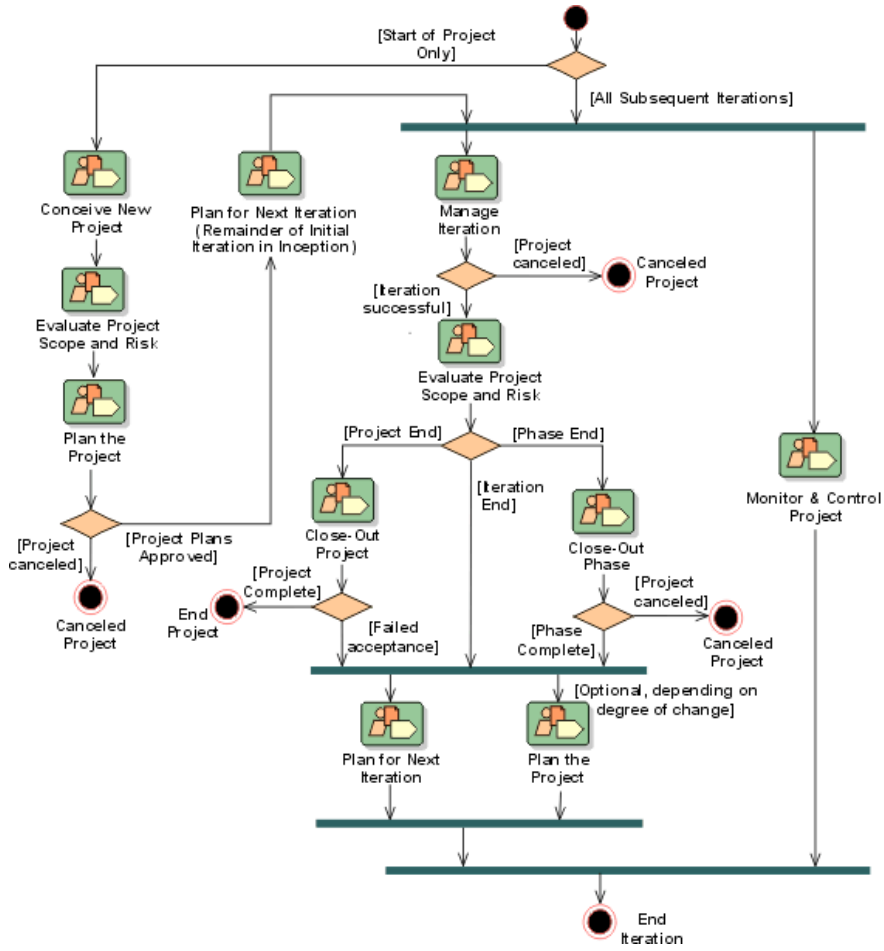


Figura 29: Secuencia de ejecución del Flujo de Trabajo de Administración del proyecto.

Configuración y control de cambios

La finalidad de este flujo de trabajo es mantener la integridad de todos los artefactos que se crean en el proceso, así como de mantener información del proceso evolutivo que han seguido. A continuación se muestra un diagrama de actividad que muestra la secuencia de ejecución.

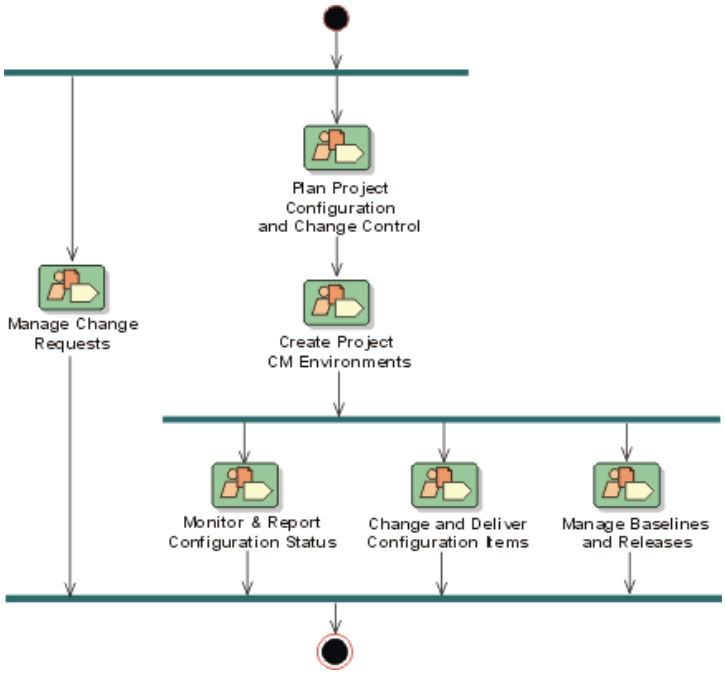


Figura 30: Secuencia de ejecución del Flujo de Trabajo Configuración y control de cambios.

Entorno

La finalidad de este flujo de trabajo es dar soporte al proyecto con las adecuadas herramientas, procesos y métodos. Brinda una especificación de las herramientas que se van a necesitar en cada momento, así como definir la instancia concreta del proceso que se va a seguir. . A continuación se muestra un diagrama de actividad que muestra la secuencia de ejecución.

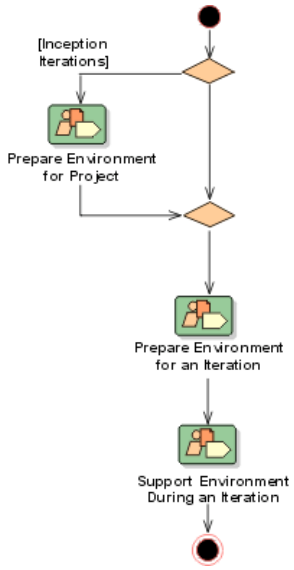


Figura 31: Secuencia de ejecución del Flujo de Trabajo Entorno.

En concreto las responsabilidades de este flujo de trabajo incluyen:

- Selección y adquisición de herramientas
- Establecer y configurar las herramientas para que se ajusten a la organización.
- Configuración del proceso.
- Mejora del proceso.
- Servicios técnicos.

El principal artefacto que se usa en este flujo de trabajo es el caso de desarrollo que especifica para el proyecto actual en concreto, cómo se aplicará el proceso, qué productos se van a utilizar y cómo van a ser utilizados. Además se tendrán que definir las guías para los distintos aspectos del proceso, como pueden ser el modelado del negocio y los casos de uso, para la interfaz de usuario, el diseño, la programación, el manual de usuario.

3 MÉTRICA Versión 3.0

El objetivo de este capítulo es realizar un resumen de la Metodología de Planificación, Desarrollo y Mantenimiento de sistemas de información (MÉTRICA) versión 3, promovida por el Consejo Superior de Informática, órgano colegiado encargado de la elaboración y desarrollo de la política informática del Gobierno Español, basado en la documentación oficial de la metodología [Ministerio de Administraciones Públicas 2001 A][Ministerio de Administraciones Públicas 2001 B][Ministerio de Administraciones Públicas 2001 C] . Cuando termine de leer este capítulo, usted:

- conocerá las características principales en que se basa MÉTRICA 3.0.
- conocerá la definición de proceso brindada en MÉTRICA 3.0, constituida por procesos principales, roles, actividades y productos.

Métrica 3.0 es la última versión de la metodología utilizada por el gobierno español, es una metodología basada en estándares de importancia en la ingeniería de software y recolecta la experiencia de las versiones anteriores. La especificación de esta metodología está disponible para el público[17].

3.1 Características de MÉTRICA 3.0

3.1.1 Basado en estándares y metodologías de Ingeniería del software

En la elaboración de MÉTRICA 3.0 se han tenido en cuenta una serie de estándares y metodologías como se muestra en la Figura 32, ellos son [Ministerio de Administraciones Públicas 2001 C]:

- El Modelo de Ciclo de Vida de Desarrollo propuesto en la norma ISO 12207 "*Information technology –Software life cycle processes*" [ISO/IEC/JTC1 1995].
- Las normas ISO/IEC TR 15.504/SPICE "*Software Process Improvement and Assurance Standards Capability Determination*" [ISO/IEC 1998].
- UNE-EN-ISO 9001:2000 Sistemas de Gestión de la Calidad.
- UNE-EN-ISO 9000:2000 Sistemas de Gestión de la Calidad.
- IEEE 610.12-1.990 "*Standard Glossary of Software Engineering Terminology*" [IEEE 1993].

[17] Ver http://www.csi.map.es/csi/metrica3/index.html.

- SSADM[18], MERISE[19] [Tardieu 1986], MAGERIT[20] y EUROMÉTODO [Euromethod Project 1996][21].

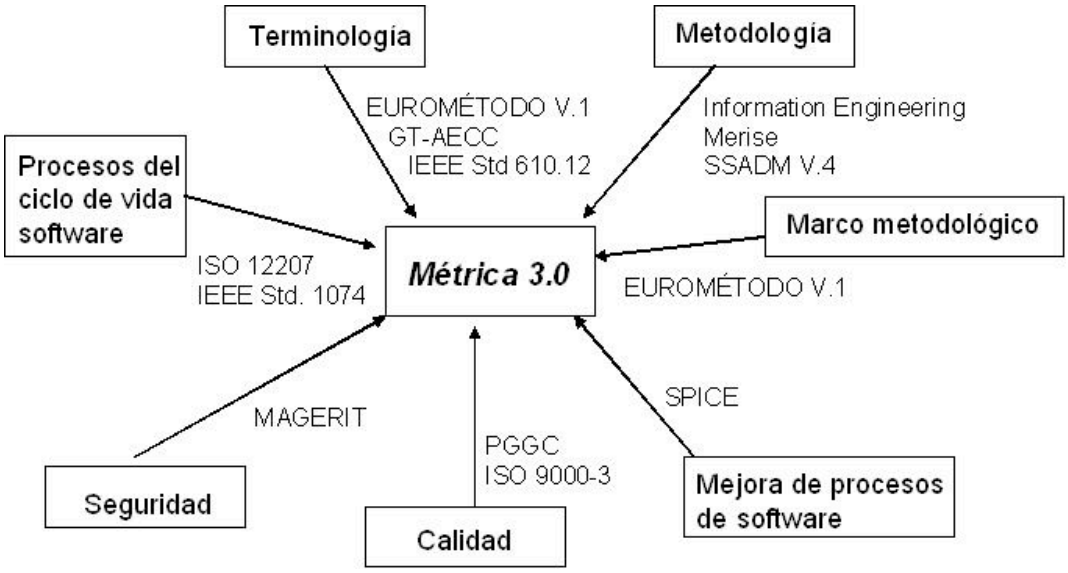


Figura 32: Estándares usados en MÉTRICA 3.0 [Herrero 2003].

3.1.2 Enfoque orientado al proceso

MÉTRICA 3.0 tiene un enfoque orientado al proceso, debido a que se ha enmarcado dentro de la norma ISO 12207, que se centra en la clasificación y definición de los procesos del ciclo de vida del software. Como punto de partida y atendiendo a dicha norma, cubre el Proceso de Desarrollo y el Proceso de Mantenimiento de Sistemas de Información.

3.1.3 Puede ser estructurado u orientado a objetos

En una única estructura la metodología MÉTRICA 3.0 cubre distintos tipos de desarrollo: estructurado y orientado a objetos, facilitando a través de interfaces la realización de los procesos de apoyo u organizativos: Gestión de Proyectos, Gestión de Configuración, Aseguramiento de Calidad y Seguridad.

[18]SSADM es el estándar británico para desarrollo de sistemas de información.
[19]MERISE es el estándar francés para desarrollo de sistemas de información.
[20]Ver http://www.csi.map.es/csi/pg5m20.htm (3.8.05).
[21]Ver http://projekte.fast.de/Euromethod/ (3.8.05).

3.1.4 Herramientas de apoyo

La automatización de las actividades propuestas en la estructura de MÉTRICA 3.0 es posible ya que sus técnicas están soportadas por una amplia variedad de herramientas del mercado de ayuda al desarrollo.

MÉTRICA ofrece una herramienta software, Gestor Metodológico[22], de ayuda a la aplicación de la metodología en cada proyecto concreto, que permite adaptar la estructura de acuerdo a las características del proyecto, facilitando el seguimiento y control de sus actividades y tareas realizadas por distintos perfiles de usuario asignados a los participantes por el jefe de proyecto y un software Selector de Herramientas[23], que ayuda a seleccionar entre las CASE del mercado la que mejor se adapta a las necesidades de cada proyecto teniendo en cuenta las características de cada organización. Estas herramientas no han sido probadas con rigurosidad y se considera que aún son herramientas inmaduras.

Existe un curso de autoformación[24], que sirve para aprender todos los conceptos y elementos de la metodología contemplando varios niveles y diversos perfiles.

3.1.5 Integración con interfaces

La estructura de MÉTRICA 3.0 incluye también un conjunto de interfaces que definen una serie de actividades de tipo organizativo o de soporte al proceso de desarrollo y a los productos. La aplicación proporciona sistemas con calidad y seguridad, no obstante puede ser necesario en función de las características del sistema un refuerzo especial en estos aspectos, el cual se obtendría aplicando la interfaz. Las interfaces descritas en la metodología son:

- Gestión de Proyectos (GP)
- Seguridad (SEG)
- Aseguramiento de la Calidad (CAL)
- Gestión de la Configuración (GC)

3.2 Estructura del proceso

MÉTRICA 3.0 posee un enfoque orientado al proceso. Ha sido concebida para abarcar el desarrollo completo de Sistemas de Información sea cual sea su complejidad y magnitud, por lo

[22]Ver http://www.csi.map.es/csi/metrica3/gesmet.htm (1.8.05).
[23]Ver http://www.csi.map.es/csi/metrica3/selector.htm (1.8.05).
[24]Ver http://www.csi.map.es/csi/metrica3/autoformacion.htm (1.8.05).

cual su estructura responde a desarrollos *máximos* y deberá adaptarse y dimensionarse en cada momento de acuerdo con las características particulares de cada proyecto [Ministerio de Administraciones Públicas 2001 C].

La metodología descompone cada uno de los procesos en actividades, y éstas a su vez en tareas. Para cada tarea se describe su contenido haciendo referencia a sus principales acciones, productos, técnicas, prácticas y participantes. MÉTRICA 3.0 se compone de procesos principales e interfaces.

3.2.1 Procesos Principales

Los procesos principales son los siguientes (ver Figura 33):

- Planificación de sistemas de información.
- Desarrollo de sistemas de información.
- Mantenimiento de sistemas de información.

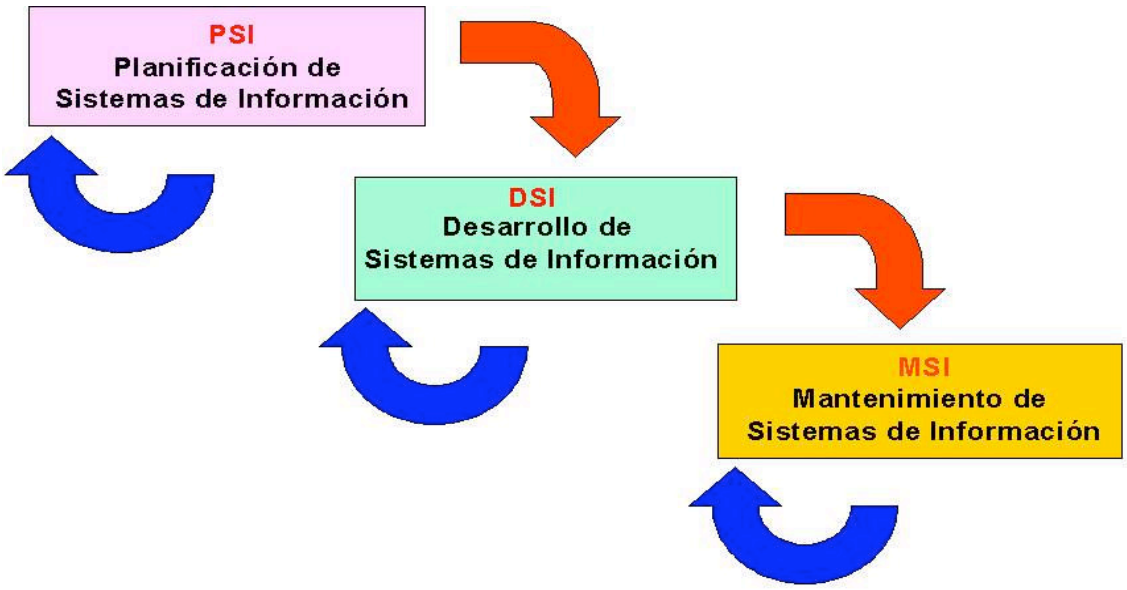


Figura 33: Procesos Principales de MÉTRICA 3.0 [Letelier 2002 B.].

Proceso de Planificación de Sistemas de Información

El Plan de Sistemas de Información tiene como objetivo la obtención de un marco de referencia para el desarrollo de sistemas de información que responda a los objetivos estratégicos de la organización. Este marco de referencia consta de:

- Una descripción de la situación actual, que constituirá el punto de partida del Plan de Sistemas de Información. Dicha descripción incluirá un análisis técnico de puntos fuertes y riesgos, así como el análisis del servicio a los objetivos de la organización.
- Un conjunto de modelos que constituya la arquitectura de información.
- Una propuesta de proyectos a desarrollar en los próximos años, así como la prioridad de realización de cada proyecto.
- Una propuesta de calendario para la ejecución de dichos proyectos.
- La evaluación de los recursos necesarios para los proyectos por desarrollar en el próximo año, con el objetivo de considerarlos en los presupuestos. Para el resto de proyectos, bastará con una estimación de alto nivel.
- Un plan de seguimiento y cumplimiento de todo lo propuesto mediante unos mecanismos de evaluación adecuados.

Actividades

El proceso **Planificación de Sistemas de Información** contiene las siguientes actividades (ver Figura 34):

- PSI 1. Inicio del Plan de Sistemas de Información.
- PSI 2. Definición y organización del PSI.
- PSI 3. Estudio de la información relevante.
- PSI 4. Identificación de requerimientos[25].
- PSI 5. Estudio de los sistemas de información actuales.
- PSI 6. Diseño del modelo de sistemas de información.
- PSI 7. Definición de la arquitectura tecnológica.
- PSI 8. Definición del plan de acción.
- PSI 9. Revisión y aprobación del PSI.

[25] En España se utiliza el término `requisitos´ para `requerimientos`.

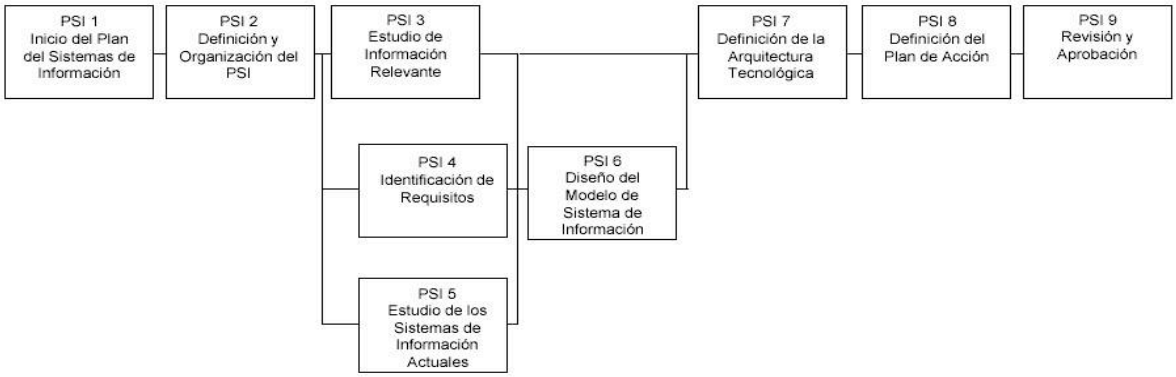


Figura 34: Actividades del PSI [Ministerio de Administraciones Públicas 2001 B].

Productos

En el proceso de **Planificación de Sistemas de Información** se generan varios productos que sirven como insumo del proceso **Estudio de Viabilidad del Sistema** (ver Figura 35). Estos productos son:

- Requerimientos del PSI.
- Arquitectura de información que incluye el Modelo de Información, Modelo de Sistemas de Información, Arquitectura Tecnológica.
- Plan de acción que incluye Plan de Proyectos, Plan de Mantenimiento.

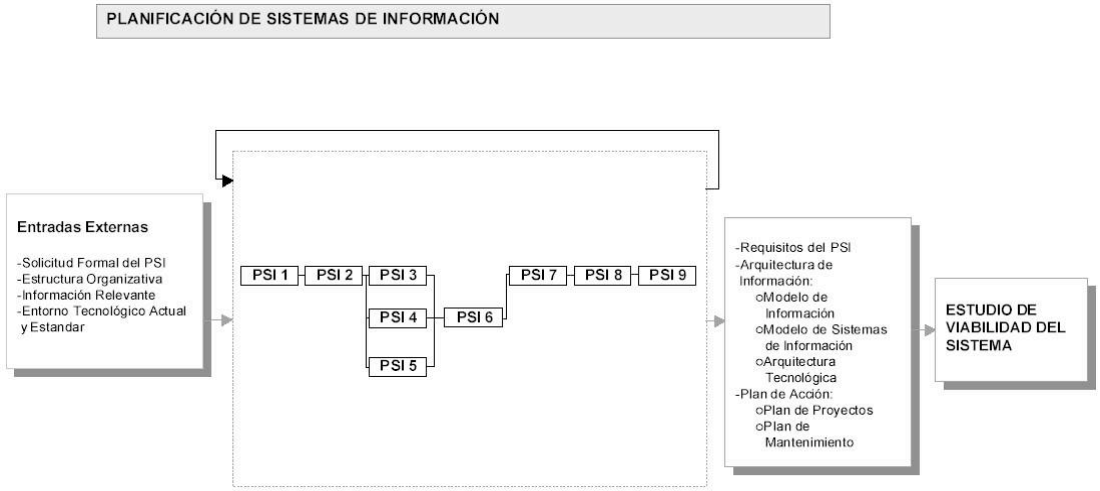


Figura 35: Planificación de Sistemas de Información [Ministerio de Administraciones Públicas 2001 B].

Proceso de Desarrollo de sistemas de información

El proceso de **Desarrollo de Sistemas de Información**, se ha subdividido en los siguientes cinco procesos (ver Figura 36):

- Estudio de Viabilidad del sistema (EVS).
- Análisis del sistema de información (ASI).
- Diseño del sistema de información (DSI).
- Construcción del sistema de información (CSI).
- Implantación y aceptación del sistema (IAS).

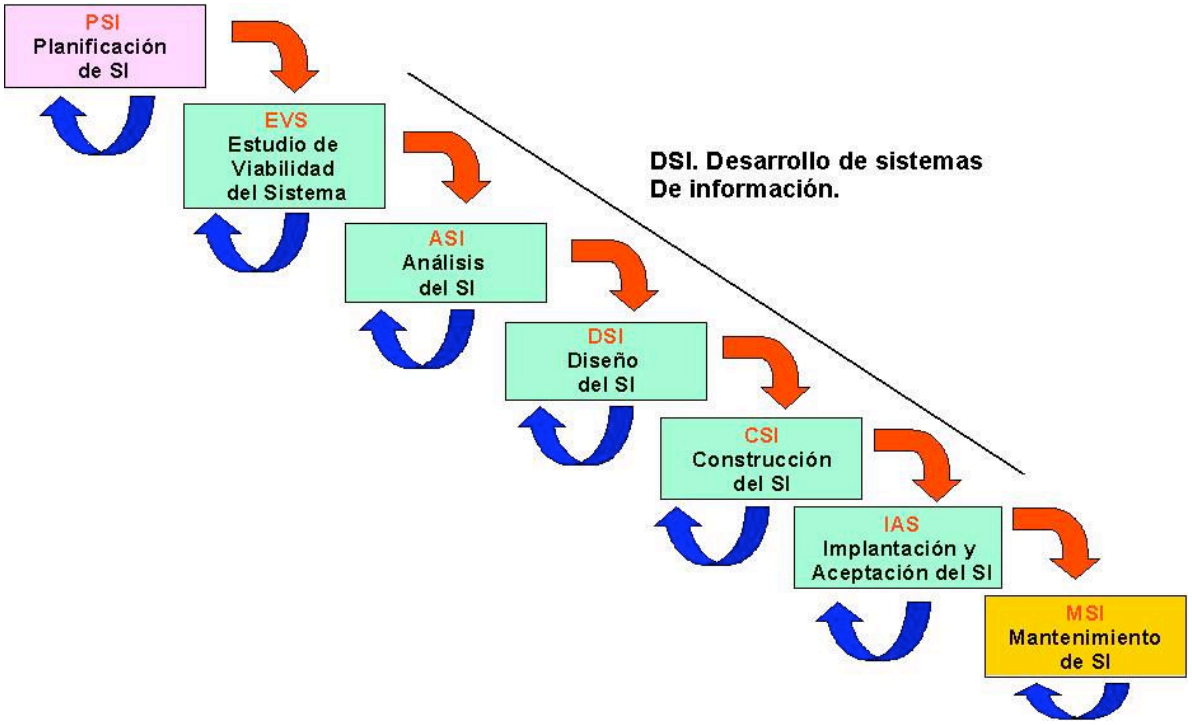


Figura 36: Subdivisión de Procesos de Desarrollo de sistemas de información [Letelier 2002 B.].

Estudio de Viabilidad del Sistema

El objetivo del **Estudio de Viabilidad del Sistema** es el análisis de un conjunto concreto de necesidades para proponer una solución a corto plazo, que tenga en cuenta restricciones económicas, técnicas, legales y operativas. La solución obtenida como resultado del estudio puede ser la definición de uno o varios proyectos que afecten a uno o varios sistemas de información ya

existentes o nuevos. Para ello, se identifican los requerimientos que se ha de satisfacer y se estudia, si procede, la situación actual.

Actividades

El proceso **Estudio de Viabilidad del Sistema** contiene las siguientes actividades (ver Figura 37):

- EV1. Establecimiento del alcance del sistema.
- EV2. Estudio de la situación actual.
- EV3. Definición de requerimientos del sistema.
- EV4. Estudio de alternativas de solución.
- EV5. Valoración de las alternativas.
- EV6. Selección de la solución.

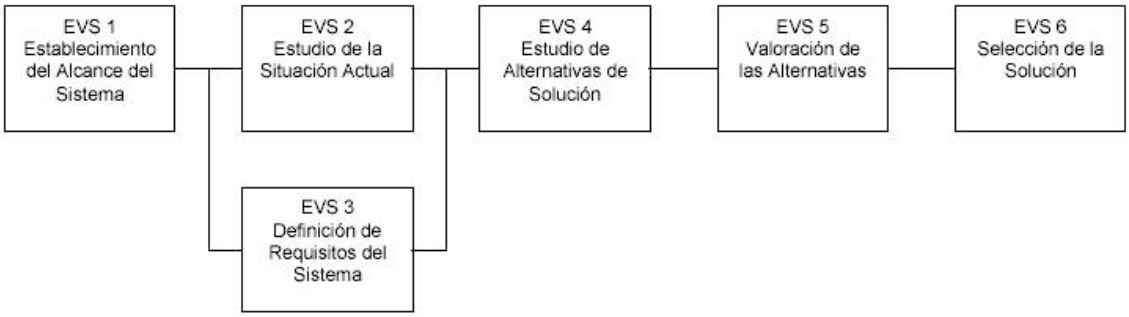


Figura 37: Actividades del Estudio de Viabilidad del Sistema (EVS) [Ministerio de Administraciones Públicas 2001 B].

Productos

En el proceso **Estudio de viabilidad del sistema** se genera una serie de productos que sirven como insumo del proceso Análisis del sistema de información (ver Figura 38). Estos productos son:

- Situación Actual.
- Catálogo de requerimientos y objetivos.

- Alternativas de solución que incluye Contexto del sistema, Impacto y Coste (Costo)[26]-Beneficio, Valoración de riesgos y Plan de trabajo.
- Solución propuesta.

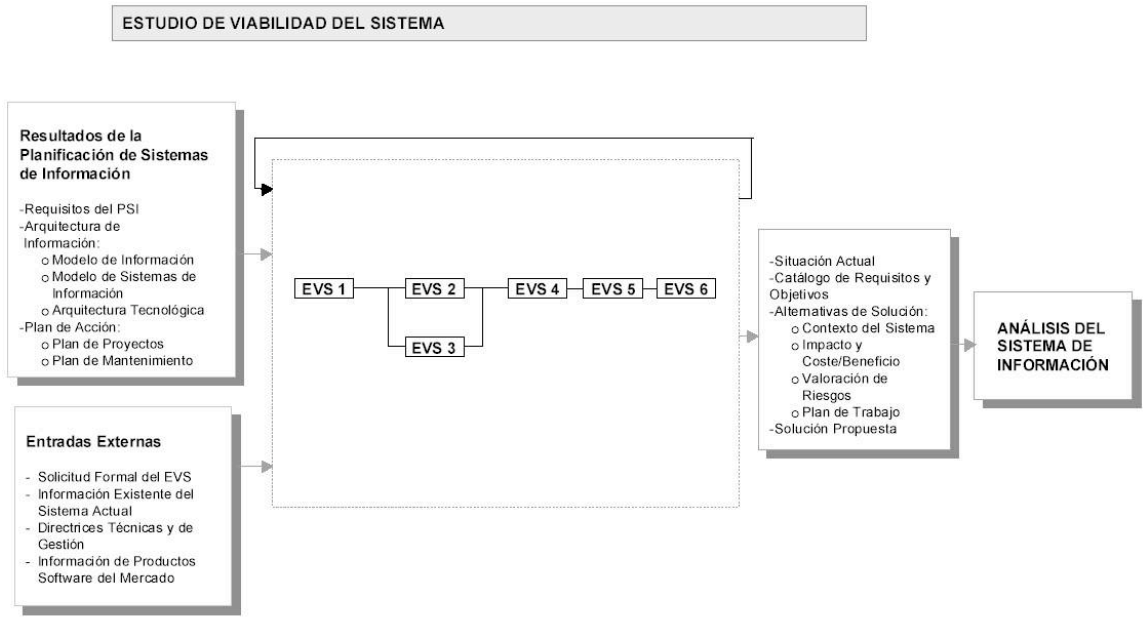


Figura 38: Productos del EVS [Ministerio de Administraciones Públicas 2001 B].

Análisis del Sistema de Información

El objetivo de este proceso es la obtención de una especificación detallada del sistema de información que satisfaga las necesidades de información de los usuarios y sirva de base para el posterior diseño del sistema.

Aunque en la práctica se aplica alternativamente, sea el enfoque orientado a objetos o el enfoque estructurado, las actividades de ambos están integradas en una estructura común.

Actividades

El proceso **Análisis del Sistema de Información** contiene actividades que se distribuyen según el enfoque orientado a objetos o estructurado por utilizar, como se muestra en la Figura 39:

- ASI 1.Definición del sistema.
- ASI 2.Establecimiento de requerimientos.

[26]En España se utiliza el término ´coste` para ´costo`

- ASI 3.Identificación de subsistemas de análisis.
- ASI 4.Análisis de casos de uso.
- ASI 5.Análisis de clases.
- ASI 6.Elaboración del modelo de datos.
- ASI 7.Elaboración del modelo de datos.
- ASI 8.Definición de interfaces de usuario.
- ASI 9.Análisis de consistencia.
- ASI 10.Especificación del plan de pruebas.
- ASI 11.Presentación y aprobación análisis sistemas de información.

En la Figura 39 se muestra la relación de actividades del proceso Análisis del Sistema de Información, tanto para desarrollos estructurados como para desarrollos orientados a objetos, distinguiendo las que se pueden realizar en paralelo de aquellas que han de realizarse secuencialmente.

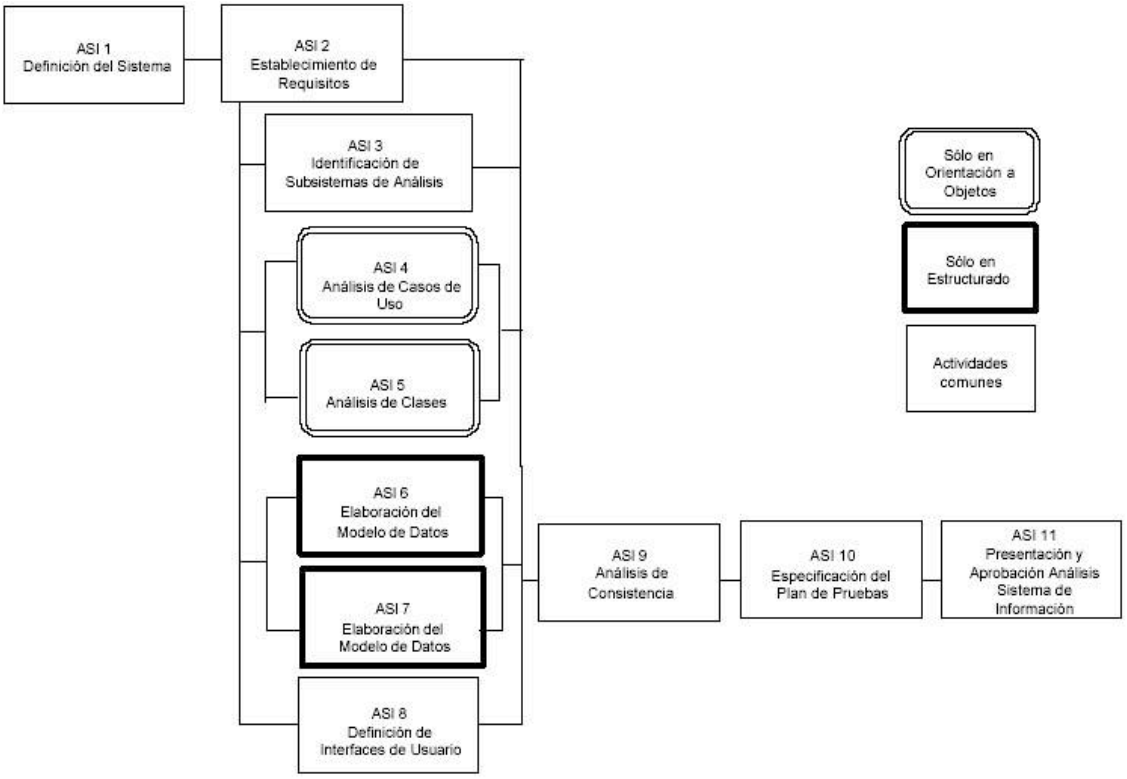


Figura 39: Actividades del Análisis del Sistema de Información (ASI) [Ministerio de Administraciones Públicas 2001 B].

Productos

Si se realiza un Análisis del sistema de información estructurado se generan los siguientes productos: (ver Figura 40).

- Catálogo de requerimientos.
- Glosario.
- Contexto del sistema.
- Modelo de datos.
- Modelo de procesos.
- Modelo de casos de uso (opcional).
- Descripción de subsistemas.

- Resultado del Análisis de consistencia.
- Interfaz de usuario.
- Especificación de requerimientos de software (ERS).

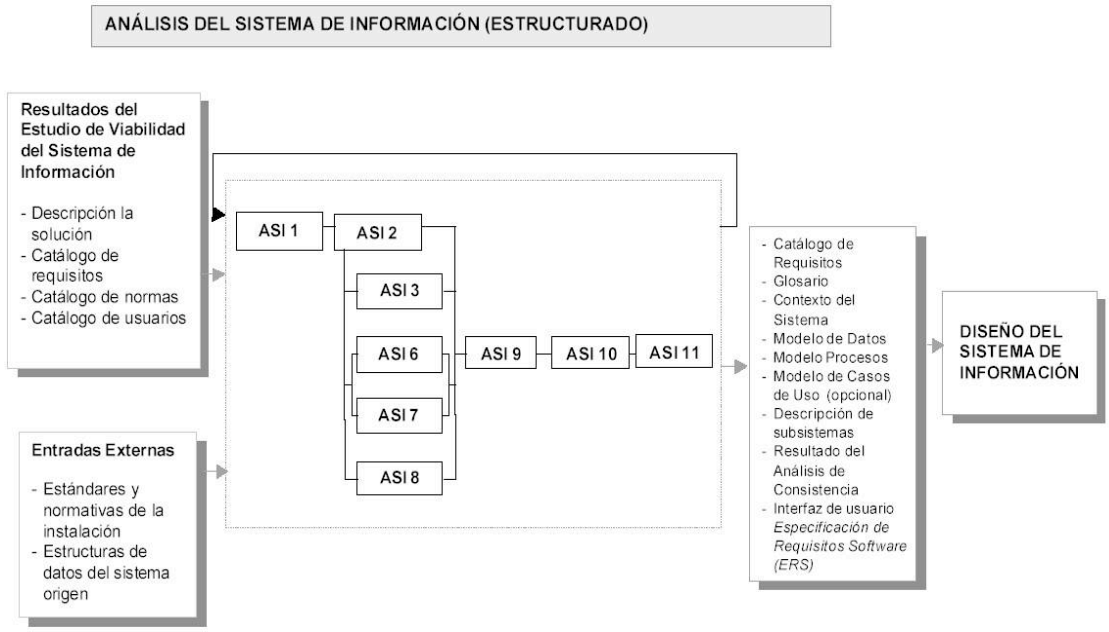


Figura 40: Productos de ASI (Estructurado) [Ministerio de Administraciones Públicas 2001 B].

Si se realiza un análisis del sistema de información orientado a objetos se generan los siguientes productos (ver Figura 41):

- Catálogo de requerimientos.
- Glosario.
- Contexto del sistema.
- Modelo del negocio.
- Modelo de dominio.
- Modelo de casos de uso.
- Descripción de subsistemas.
- Resultado del Análisis de consistencia.
- Interfaz de usuario

- Especificación de requerimientos software (ERS).

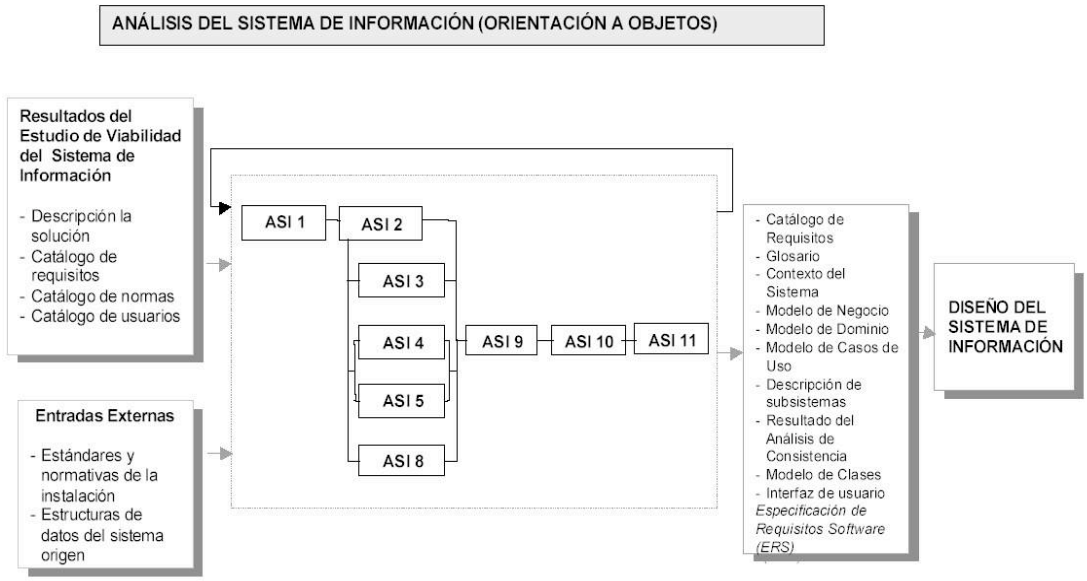


Figura 41: Productos de ASI (Orientado a objetos) [Ministerio de Administraciones Públicas 2001 B].

Diseño del Sistema de Información

El objetivo del proceso de **Diseño del Sistema de Información** (DSI) es la definición de la arquitectura del sistema y del entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes del sistema de información.

Aunque en la práctica se aplica alternativamente, el enfoque orientado a objetos con el enfoque estructurado, las actividades de ambos están integradas en una estructura común.

Actividades

El proceso de **Diseño del sistema de información** contiene actividades que se distribuyen según el enfoque orientado a objetos o estructurado por utilizar, como se muestra en la Figura 42:

- DSI 1.Definición de la arquitectura del sistema.
- DSI 2.Diseño de la arquitectura de soporte.
- DSI 3.Diseño de casos de uso reales.
- DSI 4.Diseño de clases.
- DSI 5.Diseño de la arquitectura de módulos del sistema.

- DSI 6.Diseño físico de datos.
- DSI 7.Verificación y aceptación de la arquitectura del sistema.
- DSI 8.Generación de especificaciones de construcción.
- DSI 9.Diseño de migración y carga inicial de datos.
- DSI 10.Especificación técnica del plan de pruebas.
- DSI 11.Establecimiento de requerimientos de implantación.
- DSI 12.Aprobación del diseño del sistema de información.

En la Figura 42 se muestra la relación de actividades del proceso Diseño del Sistema de Información (DSI), tanto para Desarrollos Estructurados como para Desarrollos Orientados a Objetos.

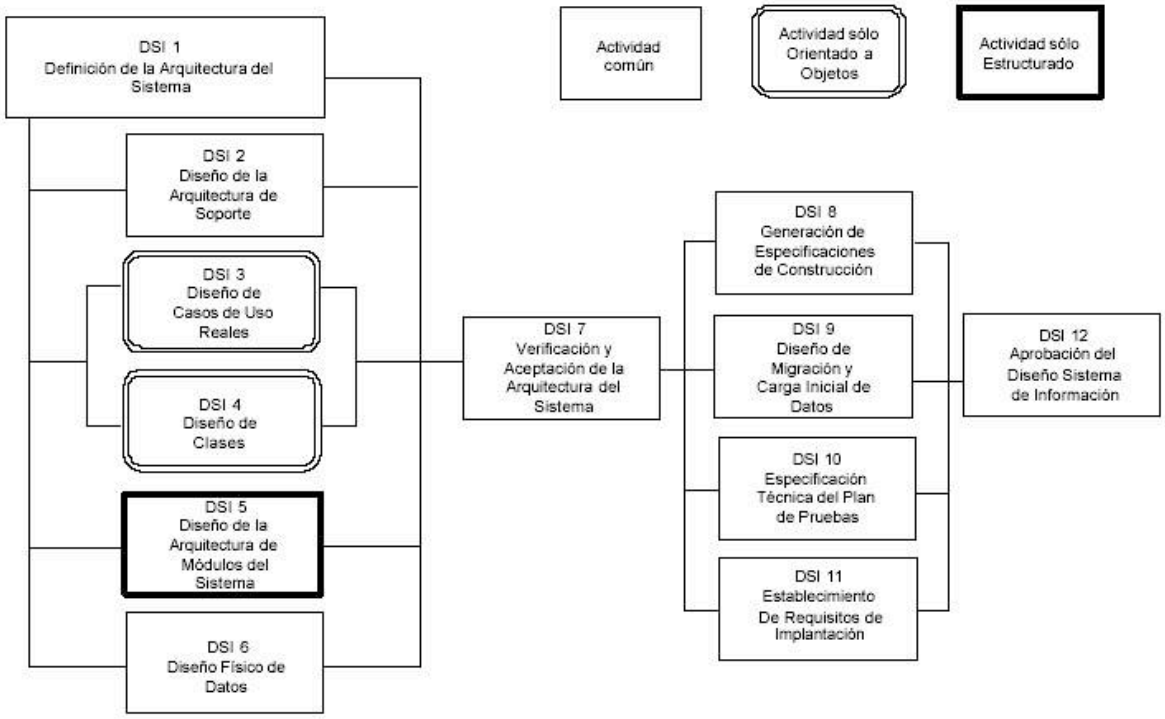


Figura 42: Actividades del Diseño del Sistema de Información (DSI) [Ministerio de Administraciones Públicas 2001 B].

Productos

Si se realiza un diseño del sistema de información estructurado se generan los siguientes productos (ver Figura 43):

- Diseño de la arquitectura del sistema.
- Entorno tecnológico, seguridad, operación y administración.
- Diseño de la arquitectura modular e interfaz de usuario.
- Modelo físico de datos.
- Resultado de análisis de consistencia.
- Especificaciones de construcción.
- Plan de migración y carga inicial.
- Especificación de entorno, niveles y planificación de las pruebas.
- Requerimientos de implantación.

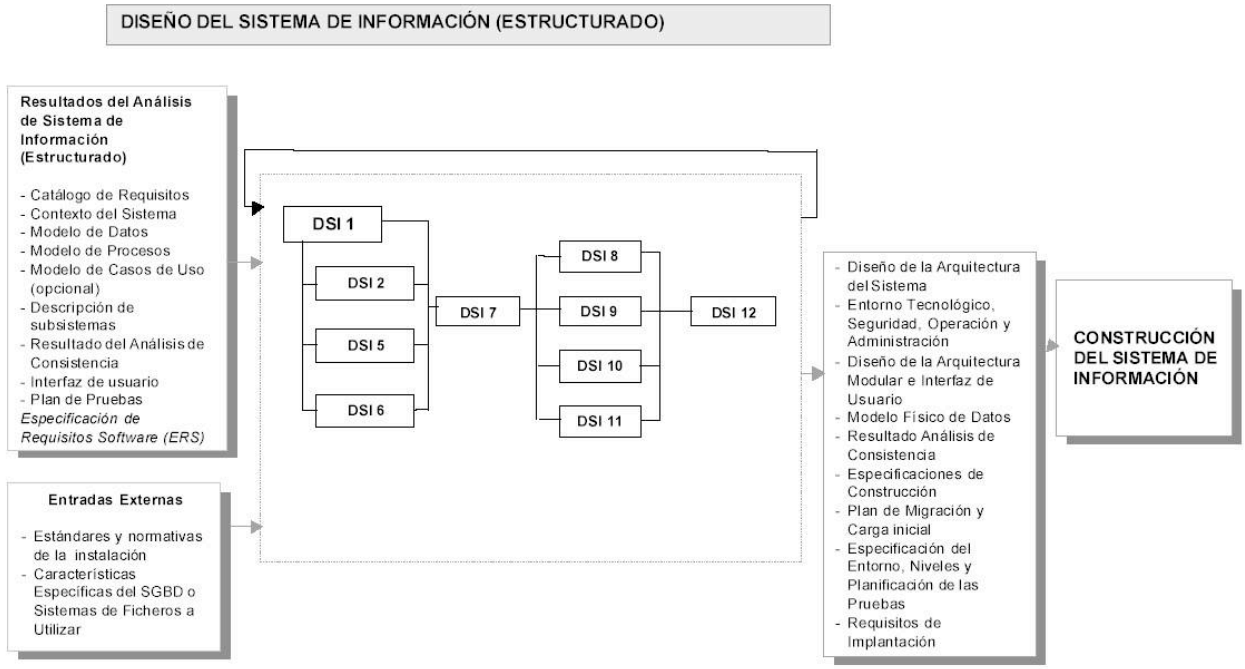


Figura 43: Productos de DSI Estructurado [Ministerio de Administraciones Públicas 2001 B].

Si se realiza un diseño del sistema de información orientado a objetos se generan los siguientes productos (ver Figura 44):

- Diseño de la arquitectura del sistema.
- Entorno tecnológico, seguridad, operación y administración.
- Diseño detallado de subsistemas.
- Diseño de la realización de casos de uso.
- Diseño de la interfaz de usuario.
- Modelos de clases de diseño.
- Modelo físico de datos.
- Resultado de análisis de consistencia.
- Especificaciones de construcción.
- Plan de migración y carga inicial.
- Especificación de entorno, niveles y planificación de las pruebas.
- Requerimientos de implantación.

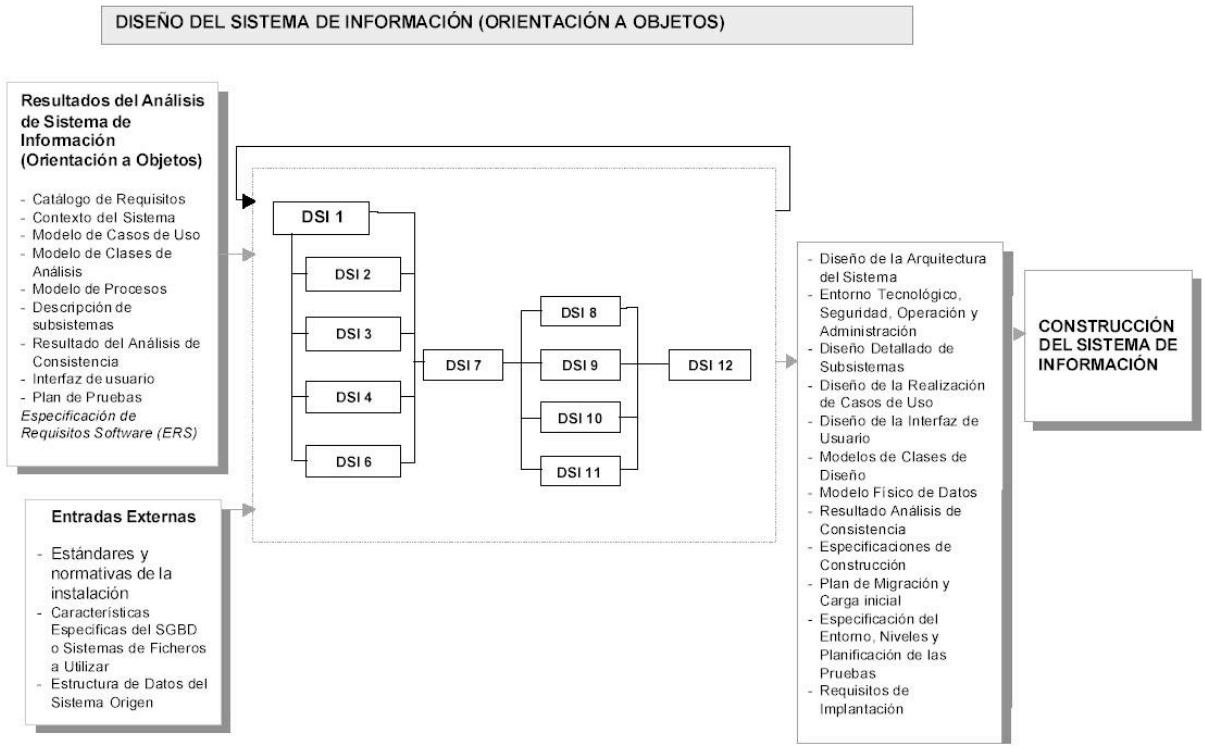


Figura 44: Productos de DSI Orientado a objetos [Ministerio de Administraciones Publicas 2001 B].

Construcción del Sistema de Información

En este proceso se genera el código de los componentes del Sistema de Información, se desarrollan todos los procedimientos de operación y seguridad y se elaboran todos los manuales de usuario final con el objetivo de asegurar el correcto funcionamiento del Sistema para su posterior implantación.

Actividades

El proceso **Construcción del sistema de información** contiene las siguientes actividades (ver Figura 45):

- CSI 1.Preparación del entorno de generación y construcción.
- CSI 2.Generación del código de los componentes y procedimientos.
- CSI 3.Ejecución de las pruebas unitarias.
- CSI 4.Ejecución de las pruebas de integración.
- CSI 5.Ejecución de las pruebas del sistema.
- CSI 6.Elaboración de los manuales de usuario.

- CSI 7.Definición de la formación (capacitación) de usuarios finales.
- CSI 8.Construcción componentes y procedimientos de migración y carga inicial de datos.
- CSI 9. Aprobación del sistema de información.

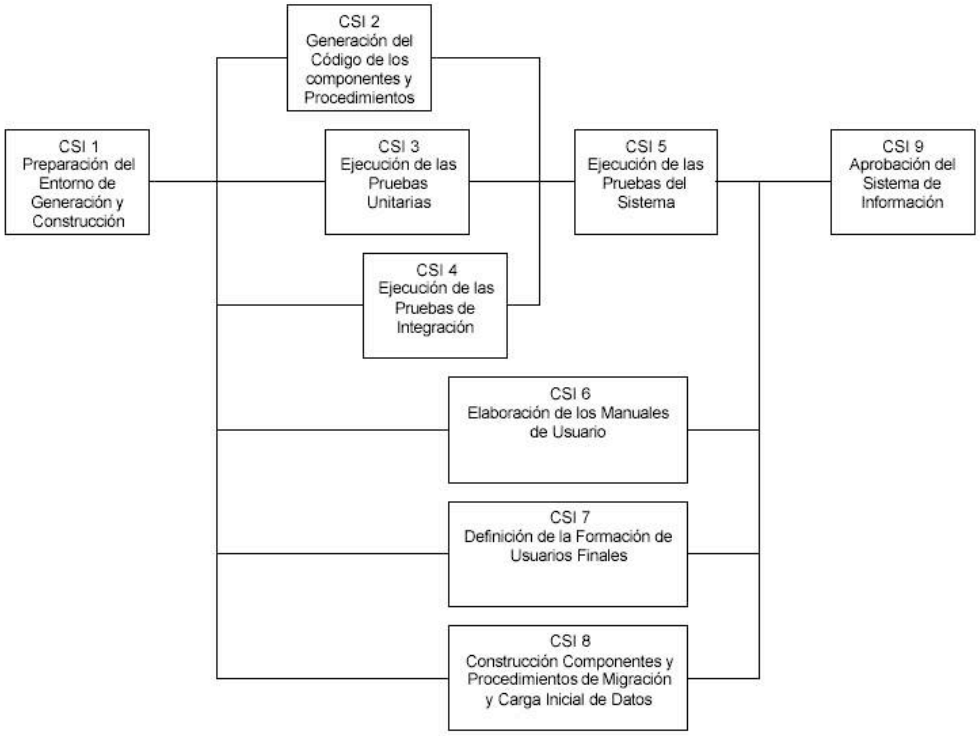


Figura 45: Actividades de Construcción del sistemas de información [Ministerio de Administraciones Públicas 2001 B].

Productos

En el proceso **Construcción del sistema de información** se generan una serie de productos que sirven como insumo del proceso Implantación y aceptación del sistema (ver Figura 46). Estos productos son:

- Bases de datos o sistema de ficheros[27] (archivos).
- Código fuente de los componentes.
- Entorno de construcción y pruebas.
- Evaluación y resultado de las pruebas.

[27]En España el término archivo se llama fichero

- Esquema de formación[28] (capacitación).
- Manuales de usuario.
- Materiales y entornos de formación (capacitación).
- Procedimientos de operación y administración del sistema, seguridad y control de acceso.

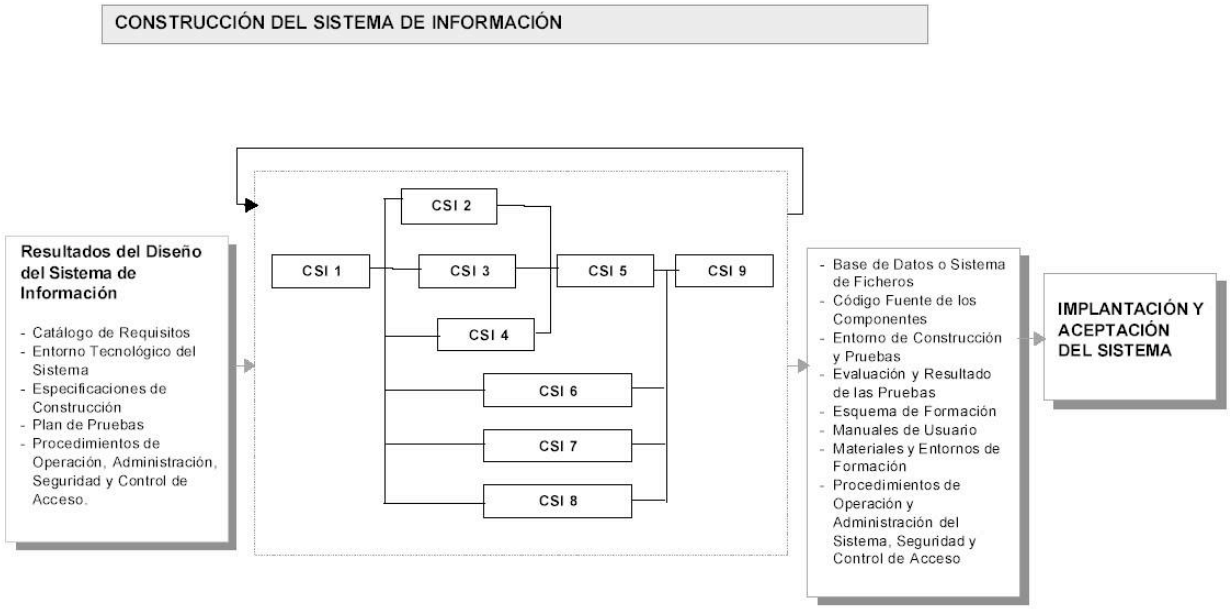


Figura 46: Productos de CSI [Ministerio de Administraciones Públicas 2001 B].

Implantación y aceptación del sistema

Este proceso tiene como objetivo principal la entrega y aceptación del sistema en su totalidad, y la realización de todas las actividades necesarias para el paso a producción.

Actividades

El proceso **Implantación y aceptación del sistema** contiene las siguientes actividades (ver 47):

- IAS1. Establecimiento del plan de implantación.
- IAS2. Formación necesaria para la implantación.
- IAS3. Incorporación del sistema a entorno de operación.
- IAS4. Carga de datos al entorno de operación.
- IAS5. Pruebas de implantación del sistema.

[28]Se utiliza el término formación para capacitación de usuarios

- IAS6. Pruebas de aceptación del sistema.
- IAS7. Preparación del mantenimiento
- IAS8. Establecimiento del acuerdo de nivel de servicio.
- IAS9. Presentación y aprobación del sistema.
- IAS10. Paso a producción.

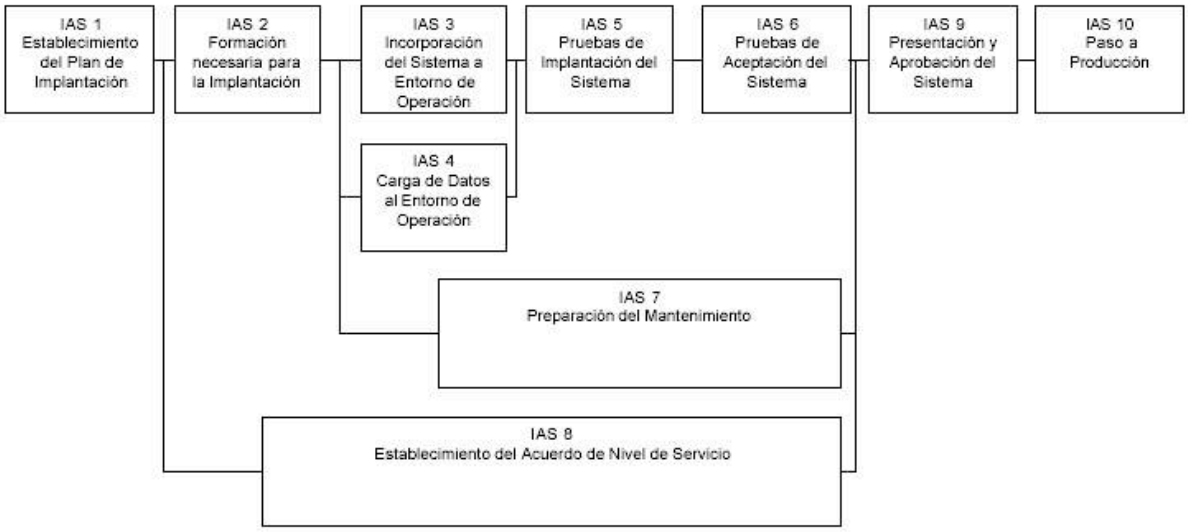
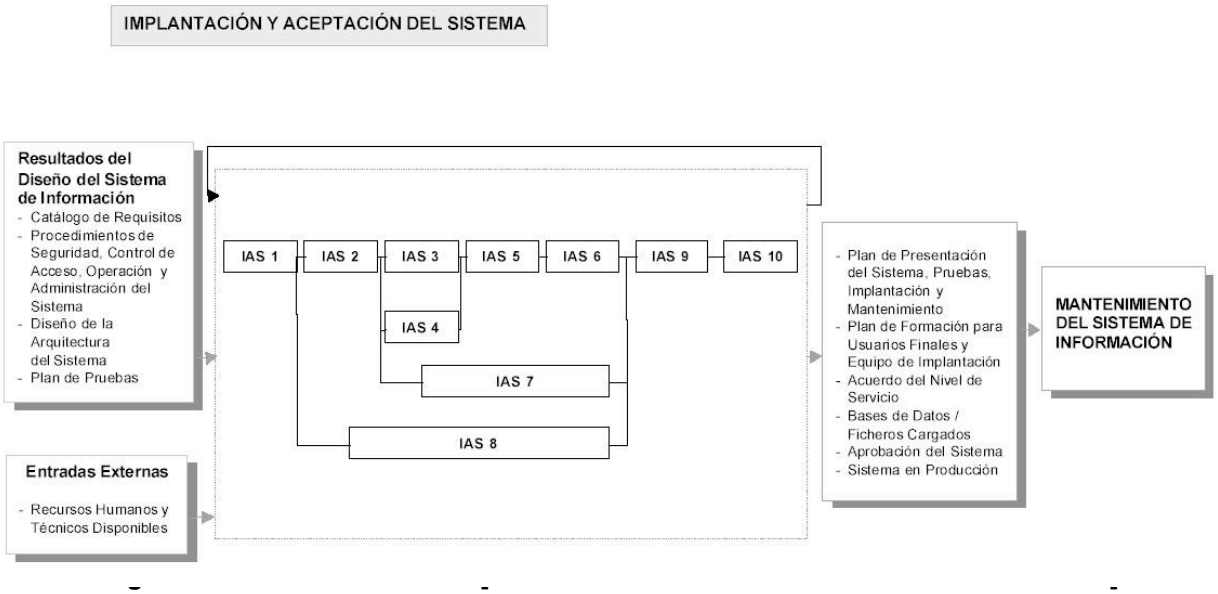


Figura 47: Actividades de Implantación de Sistemas de Información (IAS) [Ministerio de Administraciones Públicas 2001 B].

Productos

En el proceso **Implantación y aceptación del sistema** se generan productos que sirven como insumo del proceso **Mantenimiento del sistema de información** (ver Figura 48). Estos productos son:

- Plan de presentación del sistema, pruebas, implantación y mantenimiento.
- Plan de formación para usuarios finales y equipo de implantación.
- Acuerdo del nivel de servicio.
- Bases de datos-Ficheros (archivos) cargados.
- Aprobación del sistema.
- Sistema en producción.



Mantenimiento de sistemas de información

El objetivo de este proceso es la obtención de una nueva versión de un sistema de información desarrollado con MÉTRICA Versión 3 ó Versión 2, a partir de las peticiones de mantenimiento que los usuarios realizan con motivo de un problema detectado en el sistema, o por la necesidad de una mejora de este.

Actividades

El proceso **Mantenimiento del sistema de información** contiene las siguientes actividades (ver Figura 49):

- MSI 1.Registro de la petición.
- MSI 2.Análisis de la petición.
- MSI 3.Preparación de la implementación de la modificación.
- MSI 4. Seguimiento y evaluación de los cambios hasta la aceptación.

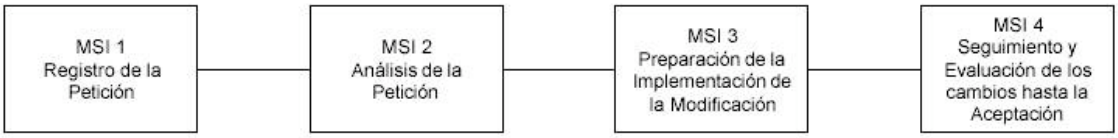


Figura 49 Actividades del Mantenimiento de sistemas de información (MSI) [Ministerio de Administraciones Públicas, 2001 B]

Productos

En el proceso **Mantenimiento del sistema de información** se generan varios productos. (Ver Figura 50). Estos son:

- Catálogo de peticiones.
- Propuesta de solución.
- Análisis de impacto de los cambios.
- Plan de acción.
- Plan de pruebas de regresión.
- Evaluación del cambio.
- Resultado de las pruebas.

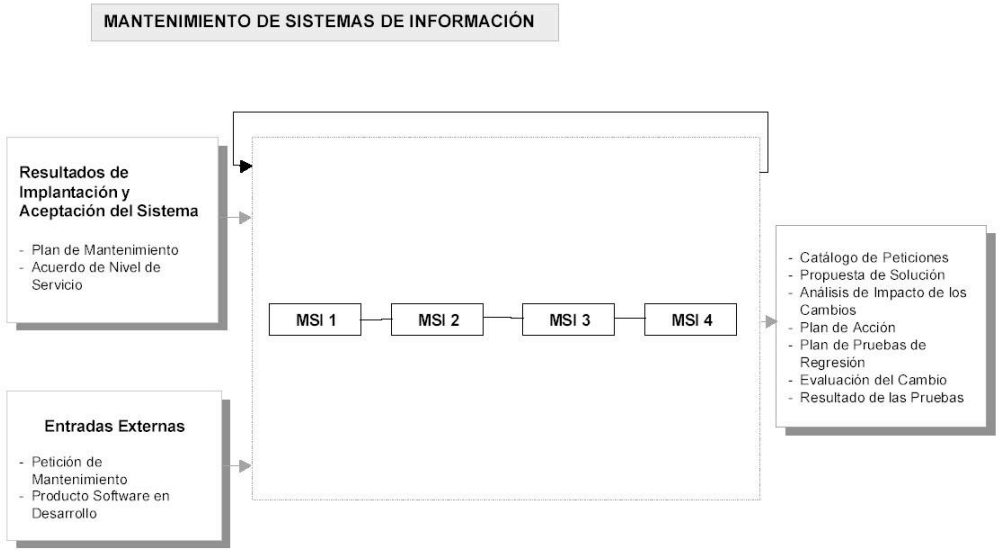


Figura 50. Productos de MSI [Ministerio de Administraciones Públicas 2001 B].

3.2.2 Interfaces

Como se mencionó en el capítulo 1, los procesos de desarrollo y mantenimiento requieren de actividades de apoyo o protección que permitan un mejor control y seguimiento sobre su ejecución y los productos que generen. En MÉTRICA 3.0 se conocen como interfaces.

A continuación se muestran las interfaces que brindan soporte al proceso de desarrollo [Ministerio de Administraciones Públicas 2001 A]:

- Gestión de proyectos.
- Aseguramiento de la calidad.
- Seguridad
- Gestión de la configuración.

Gestión de Proyectos

La **Gestión de Proyectos** tiene como finalidad principal la planificación, el seguimiento y el control de las actividades, del personal y de los recursos materiales que intervienen en el desarrollo de un Sistema de Información. Como consecuencia de este control es posible conocer en todo momento qué problemas se producen y resolverlos de manera inmediata.

Las actividades de la Interfaz de Gestión de Proyectos se presentan en el siguiente esquema, en el que se aprecian las áreas que cubre. Se distinguen tres grupos de actividades:

- Actividades de Inicio del Proyecto (GPI). Al principio del proyecto, al concluir el proceso Estudio de Viabilidad del Sistema, se realizarán las actividades de Estimación de Esfuerzo y Planificación del proyecto.
- Actividades de Seguimiento y Control (GPS). Comprenden desde la asignación de las tareas hasta su aceptación interna por parte del equipo de proyecto, incluyendo la gestión de incidencias y cambios en los requerimientos que puedan presentarse y afectar a la planificación del proyecto. El Seguimiento y Control del proyecto se realizan durante los procesos de Análisis, Diseño, Construcción, Implantación y Aceptación, y Mantenimiento del Sistema de Información, para vigilar el correcto desarrollo de las actividades y tareas establecidas en la planificación.
- Actividades de Finalización del Proyecto (GPF). Por último, al concluir el proyecto se realizan las tareas propias de Cierre del Proyecto y Registro de la Documentación de Gestión.

Actividades de inicio del proyecto

Contiene 2 actividades:

- GPI1. Estimación del esfuerzo.

- GPI2. Planificación.

Actividades de seguimiento y control

Contiene 13 actividades:

- GPS1.Asignación detallada de tareas.
- GPS2.Comunicación al equipo del proyecto.
- GPS3.Seguimiento de tareas.
- GPS4.Análisis y registro de la incidencia.
- GPS5.Petición de cambio de requerimientos.
- GPS6.Análisis de la petición de cambio de requerimientos.
- GPS7.Aprobación de la solución.
- GPS8.Estimación del esfuerzo y planificación de la solución.
- GPS9.Registro del cambio de requerimientos.
- GPS10.Finalización de la tarea.
- GPS11.Actualización de la planificación.
- GPS12.Reuniones de seguimiento.
- GPS13.Aceptación.

Actividades de finalización

Contiene 1 actividad:

- GPF1. Cierre del proyecto.

Aseguramiento de la calidad

El objetivo de la interfaz de Aseguramiento de la Calidad de MÉTRICA Versión 3 es proporcionar un marco común de referencia para la definición y puesta en marcha de planes específicos de aseguramiento de calidad aplicables a proyectos concretos.

Aseguramiento de la calidad en Estudio de Viabilidad del Sistema (EVS)

Contiene 3 actividades:

- EVS-CAL 1.Identificación de las propiedades de calidad para el sistema.
- EVS-CAL 2.Establecimiento del plan de aseguramiento de calidad.
- EVS-CAL 3.Adecuación del plan de aseguramiento de calidad a la solución.

Aseguramiento de la calidad en Análisis del Sistema de información (ASI)

Contiene 5 actividades:

- ASI-CAL 1.Especificación inicial del plan de aseguramiento de calidad.
- ASI-CAL 2.Especificación detallada del plan de aseguramiento de calidad.
- ASI-CAL 3.Revisión del análisis de consistencia.
- ASI-CAL 4.Revisión del plan de pruebas.
- ASI-CAL 5.Registro de la aprobación del análisis del sistema.

Aseguramiento de la calidad en Diseño del Sistema de Información (DSI)

Contiene 4 actividades:

- DSI-CAL 1.Revisión de la verificación de la arquitectura del sistema.
- DSI-CAL 2.Revisión de la especificación técnica del plan de pruebas.
- DSI-CAL 3.Revisión de los requerimientos de implementación.
- DSI-CAL 4.Registro de la aprobación del diseño del sistema de información.

Aseguramiento de la calidad en Construcción del Sistema de Información (CSI)

Contiene 5 actividades:

- CSI-CAL 1.Revisión del código de componentes y procedimientos.
- DSI-CAL 2.Revisión de las pruebas unitarias, integración y del sistema.
- DSI-CAL 3.Revisión de los manuales de usuario.
- DSI-CAL 4.Revisión de la formación de usuarios finales.
- DSI-CAL 5. Registro de la aprobación del sistema de información.

Aseguramiento de la calidad en Implantación y Aceptación del sistema (IAS)

Contiene 5 actividades:

- IAS-CAL 1.Revisión del plan de implantación del sistema.
- IAS-CAL 2.Revisión de las pruebas de implantación del sistema.
- IAS-CAL 3.Revisión de las pruebas de aceptación del sistema.
- IAS-CAL 4.Revisión del plan de mantenimiento del sistema.
- IAS-CAL 5.Registro de la aprobación de la implantación del sistema.

Aseguramiento de la calidad en Mantenimiento de Sistemas de Información (MSI)

Contiene 3 actividades:

- MSI-CAL 1.Revisión del mantenimiento del sistema de información.
- MSI-CAL 2.Revisión del plan de pruebas de regresión.
- MSI-CAL 3.Revisión de la realización de las pruebas de regresión.

Seguridad

El objetivo de la interfaz de seguridad de MÉTRICA Versión 3 es incorporar en los sistemas de información mecanismos de seguridad adicionales.

La seguridad del sistema de información ya se considera en MÉTRICA Versión 3 como requerimiento (ASI 2.1), es decir previamente al desarrollo del sistema. La interfaz de Seguridad hace posible incorporar durante la fase de desarrollo las funciones y mecanismos que refuerzan la seguridad del nuevo sistema y del propio proceso de desarrollo, asegurando su consistencia y seguridad.

Seguridad en Planificación de Sistemas de Información (PSI)

Contiene 4 actividades:

- PSI-SEG 1.Planificación en la seguridad requerida en el proceso Planificación de sistemas de información.
- PSI-SEG 2.Evaluación del riesgo para la arquitectura.
- PSI-SEG 3.Determinación de la seguridad en el plan de acción.
- PSI-SEG 4.Catalogación de los productos generados durante el proceso de planificación de sistemas de información.

Seguridad en Estudio de Viabilidad del Sistema (EVS)

Contiene 6 actividades:

- EVS-SEG 1.Estudio de la seguridad requerida en el proceso Estudio de viabilidad del sistema.
- EVS-SEG 2.Selección del equipo de seguridad.
- EVS-SEG 3.Recomendaciones adicionales de seguridad para el sistema de información.
- EVS-SEG 4.Evaluación de la seguridad de las alternativas de solución.
- EVS-SEG 5.Evaluación detallada de la seguridad de la solución propuesta.
- EVS-SEG 6.Catalogación de los productos generados durante el proceso de estudio de viabilidad del sistema.

Seguridad en Análisis del Sistema de Información (ASI)

Contiene 4 actividades:

- ASI-SEG 1.Estudio de la seguridad requerida en el proceso de análisis de sistemas de información.
- ASI-SEG 2.Descripción de las funciones y mecanismos de seguridad.
- ASI-SEG 3.Definición de los criterios de aceptación de la seguridad.
- ASI-SEG 4. Catalogación de los productos generados durante el proceso de análisis de sistemas de información.

Seguridad en Diseño en Sistemas de Información (DSI)

Contiene 5 actividades:

- DSI-SEG 1.Estudio de la seguridad requerida en el proceso de diseño del sistema de información.
- DSI-SEG 2.Especificación de requerimientos de seguridad del entorno tecnológico.
- DSI-SEG 3.Requerimientos de seguridad del entorno de construcción.
- DSI-SEG 4.Diseño de pruebas de seguridad.
- DSI-SEG 5.Catalogación de los productos generados durante el proceso de diseño de sistemas de información.

Seguridad en Construcción del Sistema de Información (CSI)

Contiene 4 actividades:

- CSI-SEG 1.Estudio de la seguridad requerida en el proceso de construcción del sistema de información.
- DSI-SEG 2.Evaluación de los resultados de pruebas de seguridad.
- DSI-SEG 3.Elaboración del plan de formación de la seguridad.
- DSI-SEG 4.Catalogación de los productos generados durante el proceso de construcción del sistema de información.

Seguridad en Implantación y Aceptación del sistema (IAS)

Contiene 5 actividades:

- IAS-SEG 1.Estudio de la seguridad requerida en el proceso de implantación y aceptación del sistema.
- IAS-SEG 2.Revisión de medidas de seguridad del entorno de operación.
- IAS-SEG 3.Evaluación de resultados de pruebas de seguridad de implantación del sistema.

- IAS- SEG 4.Catalogación de los productos generados durante el proceso de implantación del sistema.
- IAS-SEG 5.Revisión de medidas de seguridad en el entorno de producción.

Seguridad en Mantenimiento del Sistema de Información (MSI)

Contiene 3 actividades:

- MSI-SEG 1.Estudio de la seguridad requerida en el proceso de Mantenimiento de sistemas de información.
- MSI-SEG 2.Especificación e identificación de las funciones y mecanismos de seguridad.
- MSI-SEG 3.Catalogación de los productos generados durante el proceso de Mantenimiento de sistemas de información.

Gestión de la configuración

En el desarrollo de software los cambios, debidos principalmente a modificaciones de requerimientos y fallos, son inevitables. Normalmente se trabaja en equipo, por lo que es preciso llevar un control y registro de los cambios con el fin de reducir errores, aumentar la calidad y la productividad y evitar los problemas que puede acarrear una incorrecta sincronización en dichos cambios, al afectar a otros elementos del sistema o a las tareas realizadas por otros miembros del equipo de proyecto.

El objetivo de la gestión de la configuración es mantener la integridad de los productos que se obtienen a lo largo del desarrollo de los sistemas de información, garantizando que no se realizan cambios incontrolados y que todos los participantes en el desarrollo del sistema disponen de la versión adecuada de los productos que manejan. Así, entre los elementos de configuración software, se encuentran no únicamente los ejecutables y el código fuente, sino también los modelos de datos, modelos de procesos, especificaciones de requerimientos, pruebas, etc.

La gestión de configuración se realiza durante todas las actividades asociadas al desarrollo del sistema, y continúa registrando los cambios hasta que éste deja de utilizarse.

Gestión de la configuración en Estudio de Viabilidad del Sistema (EVS)

Contiene 2 actividades:

- EVS-GC 1.Definición de los requerimientos de gestión de configuración.
- EVS-GC 2.Establecimiento del plan de gestión de la configuración.

Gestión de la configuración en Análisis, Diseño, Construcción e Implantación y Aceptación del sistema de información.

Contiene 2 actividades:

- GC 1.Identificación y registro de productos.
- GC 2.Identificación y registro del producto global.

Gestión de la configuración en Mantenimiento de Sistemas de Información (MSI).

Contiene 1 actividad:

- MSI-GC 1.Registro del cambio en el sistema de gestión de la configuración.

3.2.3 Participantes (Roles)

MÉTRICA 3.0 establece una serie de perfiles en los que se encuadran la totalidad de los participantes. Los perfiles establecidos son:

Directivo:

- Comité de dirección.
- Comité de seguimiento.
- Directores de usuarios.
- Usuarios expertos.

Jefe de Proyecto:

- Jefe de proyecto.
- Responsable de implantación.
- Responsable de mantenimiento.
- Responsable de operación.
- Responsable de Sistemas.
- Responsable de Seguridad.
- Responsable de Calidad.

Consultor:

- Consultor.
- Consultor Informático.
- Consultor de las Tecnologías de la Información.
- Consultor de Sistemas de Información.
- Especialista en Comunicaciones.
- Técnico de Sistemas.
- Técnico de Comunicaciones.

Analista:

- Analista.
- Administrador de Bases de Datos.
- Equipo de Arquitectura.

- Equipo de Formación.
- Equipo de Implantación.
- Equipo de Operación.
- Equipo de Seguridad.
- Equipo de Soporte Técnico.
- Equipo de Proyecto.
- Grupo de Aseguramiento de la Calidad.

Programador:

- Programador.

3.2.4 Técnicas y prácticas

En el proceso de Desarrollo de Sistemas de Información se incluyen tanto las técnicas propias de un desarrollo orientado a objetos como uno estructurado, ya que las actividades de ambos enfoques están integradas en una estructura común.

Se hace una distinción entre técnicas y prácticas en función del propósito al que respondan. Se considera técnica al conjunto de heurísticas y procedimientos que se apoyan en estándares, es decir, que utilizan una o varias notaciones específicas en términos de sintaxis y semántica y cumplen unos criterios de calidad en cuanto a la forma de obtención del producto asociado. Las prácticas representan un medio para la consecución de unos objetivos específicos de manera rápida, segura y precisa, sin necesidad de cumplir unos criterios o reglas preestablecidas. MÉTRICA 3.0 ofrece técnicas tanto de desarrollo como de gestión de proyectos.

Técnicas de desarrollo

Las técnicas de desarrollo son un conjunto de procedimientos que se basan en reglas y notaciones específicas en términos de sintaxis, semántica y gráficos, orientadas a la obtención de productos en el desarrollo de un sistema de información. En desarrollos del tipo estructurado o de orientación a objetos merecen especial atención las técnicas gráficas, que proponen símbolos y notaciones estándares para una mejor comprensión de los sistemas o sus componentes.

- Análisis Costo-Beneficio[29].
- Casos de uso.
- Diagrama de clases.
- Diagrama de componentes.
- Diagrama de descomposición.

[29]Puede ser considerada una técnica de gestión de proyectos.

- Diagrama de despliegue.
- Diagrama de estructura.
- Diagrama de flujos de datos (DFD).
- Diagramas de interacción (diagrama de secuencia, diagrama de colaboración).
- Diagrama de paquetes.
- Diagrama de transición de estados.
- Modelo de procesos de la organización.
- Modelo entidad relación-extendido
- Normalización.
- Optimización.
- Reglas de obtención del modelo físico a partir del lógico.
- Reglas de transformación.
- Técnicas matriciales.

Técnicas de gestión de proyectos

La Gestión de Proyectos es un conjunto de actividades específicas que se emplean para la administración del proyecto. Estas actividades comprenden diversos aspectos:

- Estimación del esfuerzo necesario para el desarrollo de un Sistema de Información.
- Planificación de tareas y recursos.
- Control de tareas.
- Seguimiento del proyecto.
- Control de las incidencias.
- Control de cambios.

Estas técnicas son:

- Técnicas de estimación (Método *Albrecht* para el análisis de los puntos de función, Método *MARKII* para el análisis de los puntos de función).
- *STAFFING SIZE* (orientación a objetos).
- Planificación.
- *Program Evaluation & Review Technique* – PERT.
- Diagrama de *Gantt.*
- Estructura de descomposición de trabajo.

- Diagrama de extrapolación.

Prácticas

A continuación se listan las prácticas propuestas por MÉTRICA 3.0:

- Análisis de impacto.
- Catalogación.
- Cálculos de accesos.
- Caminos de acceso.
- Diagrama de presentación.
- Factores críticos de éxito.
- Impacto de la organización.
- Presentaciones.
- Prototipado.
- Pruebas[30] (unitarias, integración, sistema, implantación, aceptación y regresión)
- Revisión formal.
- Revisión técnica.
- Sesiones de trabajo.

[30]Puede ser considerada técnica de desarrollo.

4 Reflexiones

Este capítulo se presenta un análisis sobre los siguientes puntos:

- definición de proceso brindado por *RUP* y Métrica 3.0.
- beneficios que brinda el basarse en una definición de proceso durante el desarrollo de software.

Es importante recalcar que el análisis del presente capítulo está sujeto a la investigación y la experiencia de la autora del informe.

4.1 Definición y especificación de los procesos de software RUP y Métrica 3.0

En esta sección se presenta un marco elaborado para el análisis para una o varias metodologías de desarrollo de software. Para efectos de este informe, el marco de análisis es aplicado a las metodologías RUP y MÉTRICA 3.0. El marco de análisis se compone de los siguientes apartados:

- Terminología
- Estado de la metodología
- Estructura
- Características claves
- Soporte
- Reflexiones

4.1.1 Terminología

Para iniciar, se unifica la terminología usada por las metodologías.

Término por utilizar	RUP	Métrica 3.0
Actividad	Actividad	Actividad
Tarea	Paso	Tarea
Rol	Rol	Participante
Producto	Artefacto	Producto
Guía	Directrices de trabajo o Plantillas	No define
Práctica/ Técnica	Práctica	Práctica/ Técnica
Ciclo de vida	Ciclo de vida	Estrategia de desarrollo
Flujo de trabajo	Flujo de trabajo	Proceso

4.1.2 Estado

A continuación se evaluar el estado de madurez o de aplicación de las metodologías. Para ello se considerarán los siguientes criterios:

- **En construcción**: La metodología se encuentra en proceso de definición y experimentación por parte de sus creadores, se ha liberado alguna información pero no oficialmente. No se cuenta con experiencia en el mercado.

- **En experimentación**: La metodología ha sido liberada oficialmente, ha sido sometida a experimentos por medio de sus creadores y está siendo implantada en el mercado sin resultados visibles aún. No existen suficientes experiencias en el mercado para garantizar el éxito de la metodología.
- **Activa**: La metodología ha sido probada y aceptada en el mercado. Se cuenta con experiencias de éxito y fracaso de la metodología. Existen recomendaciones de uso con base en la experiencia. A lo mejor existe refinamiento de la metodología basada en la experiencia, que ha provocado nuevas versiones.

Metodología	Estado	Año en que se liberó la primera propuesta oficial
RUP	Activa	1998
MÉTRICA 3.0	Activa	1994

4.1.3 Estructura

En esta sección se presenta un análisis de la incorporación y definición de los elementos de un proceso de software. El análisis se compone de la valoración de los siguientes criterios:

- **Elementos fundamentales**: Actividades, roles, productos y sus relaciones con otros elementos como ciclo de vida, técnicas, guías, herramientas, entre otros.
- **Incorporación de diversas disciplinas**: Actividades fundamentales y de apoyo.

Elementos fundamentales de una metodología: Actividad, Rol, Producto y sus relaciones

Para la valoración de la identificación y definición de los elementos fundamentales se utilizarán los siguientes criterios:

- **Sí**: Cuenta con una robusta definición del elemento.
- **No**: No cuenta con la definición del elemento por completo.
- **Limitada**: Cuenta con la definición del elemento, pero de manera ambigua o incompleta.

Elemento	RUP	Métrica 3.0
– Actividad –		
Identificación de actividades.	**Sí**	**Sí**
Secuencia de tareas para una actividad.	**Sí**	**Sí**
Descripción del ciclo de vida que organiza las actividades.	**Sí**	**Limitada**
Guías para la ejecución de actividades.	**Sí**	**Limitada**
Identificación de las técnicas y prácticas - reflejadas en la ejecución de las actividades -.	**Sí**	**Sí**
Guías para el uso de las técnicas y prácticas.	**Limitada**	**No**
– Rol –		
Identificación de los roles.	**Sí**	**Sí**
Responsabilidades de cada rol.	**Sí**	**Limitada**
Habilidades requeridas para la ejecución de cada rol.	**Sí**	**No**

Elemento	RUP	Métrica 3.0
– Producto–		
Identificación de los productos por elaborar.	**Sí**	**Sí**
Plantillas para la elaboración del producto.	**Sí**	**No**
Ejemplos para la elaboración de productos.	**Sí**	**No**
Propone herramientas para la elaboración de productos.	**Sí**	**No**
Identifica la notación para la elaboración de modelos – productos específicos –.	**Sí**	**Sí**
Guías para la elaboración de productos.	**Sí**	**No**
Describe la rastreabilidad o hilo conductor entre productos.	**Sí**	**Limitada**
Calendarización del producto - momento de la creación -.	**Sí**	**Sí**
– Relaciones Actividad, Rol y Producto –		
Establece la relación entre actividad y rol - responsable(s) de la ejecución de la actividad -.	**Sí**	**Sí**
Establece la relación entre actividad y producto de entrada – insumos de la actividad -.	**Sí**	**Sí**
Establece la relación entre actividad y producto de salida – resultados de la actividad -.	**Sí**	**Sí**
Establece la relación entre rol y producto - responsable(s) de la creación -.	**Sí**	**Limitada**

Flujos de trabajo. Fundamentales y de protección

Para la valoración de la identificación y definición de las disciplinas se utilizarán los siguientes criterios:

- **Sí**: cuenta con una robusta definición de la disciplina.
- **No**: No cuenta con la definición de la disciplina.
- **Limitada**: Cuenta con la definición de la disciplina, pero de manera ambigua o incompleta.

Disciplina – Actividades fundamentales –	RUP	Métrica 3.0
Modelado de negocio	**Sí**	**Limitada**
Requerimientos	**Sí**	**Sí**
Análisis y diseño	**Sí**	**Sí**
Implementación	**Sí**	**Sí**
Implantación	**Sí**	**Sí**
Mantenimiento	**Limitada**	**Sí**
Permite varios paradigmas de análisis, diseño y desarrollo	**No**	**Si**

Disciplina – Actividades de apoyo –	RUP	Métrica 3.0
Administración de proyectos	**Sí**	**Sí**
Configuración del proceso	**Sí**	**Sí**
Aseguramiento de la calidad	**Sí**	**Sí**
Control de versiones	**Sí**	**Sí**
Control de cambios	**Sí**	**Sí**
Control de riesgos	**Sí**	**Sí**
Seguridad	**No**	**Sí**

4.1.4 Características claves

En esta sección se presentan los puntos clave y los puntos débiles de cada metodología.

Metodología	Puntos clave	Puntos débiles
RUP	• Desarrollo del software iterativo e incremental. • Desarrollo del software orientado al control de riesgos y cambios. • Promueve el uso de estándares. • Incorporación de actividades de apoyo (calidad, planeación y otros). • Robusta definición de las relaciones de los elementos roles, actividades y productos. • Productos orientados al modelado gráfico. • Jerarquía de trabajo robusta. • Adecuado soporte: herramientas, guías, plantillas, ejemplos, otros. • Proceso configurable.	• Basada en un único paradigma (orientado a objetos). • No contempla consideraciones especiales para el mantenimiento del software. • No ofrece consideraciones particulares para la seguridad.
Métrica 3.0	• Flexibilidad en la elección del paradigma de análisis, diseño e implementación, ya que define las actividades para ambos. • Incorporación de actividades de apoyo (calidad, planeación y otros). • Incorporación de actividades relacionadas a seguridad. • Productos orientados a la documentación. • Jerarquía de trabajo robusta. • Proceso configurable. • Contempla el mantenimiento de software.	• No es orientada al uso de estándares. • Poco soporte: herramientas, guías, plantillas, ejemplos, otros. • No cuenta con una clara definición de ciclo de vida; aunque la estructura de las actividades coincide con un cascada, no define explícitamente ése como su ciclo de vida.

4.1.5 Soporte

A continuación se muestran los recursos que ofrecen las metodologías como ayuda o soporte a quienes las utilizan.

Recurso	RUP	Métrica 3.0
Documentación	La documentación oficial ofrece un documento en hipertexto que facilita la navegabilidad de la metodología. Además, cuenta con una gran gama de libros e información en la Web de diversas fuentes.	La documentación oficial está disponible en formato pdf en la Web. También existe documentación preparada por profesores de diversas universidades españolas, y disponible en la Web.

Licencia para la documentación oficial de la metodología	Requiere adquirir una licencia para su uso, la cual tiene un valor dependiendo de diversos factores como por ejemplo tamaño de la empresa. Existen algunas versiones para evaluación (en la Web).	No requiere licencia, es gratuita.
Página oficial	http://www-306.ibm.com/software/awdtools/rup/support/	http://www.csi.map.es/csi/metrica3/index.html
Ejemplos	La documentación oficial cuenta con ejemplos. Además hay una gran variedad de ejemplos en diferentes idiomas, tanto en libros y como en la Web.	La documentación oficial no cuenta con ejemplos. Existen algunos ejemplos elaborados por profesores de las universidades españolas (en la Web).
Experiencia en su uso e interés	RUP es una metodología muy utilizada mundialmente. Se han realizados y utilizado muchas adaptaciones de RUP y se ha escrito sobre sus fortalezas y en qué casos es conveniente y en qué casos no. En Costa Rica cada vez tiene más difusión.	MÉTRICA 3.0 es la metodología oficial del gobierno español; está basada en versiones anteriores que han sido probadas y mejoradas gracias a la experiencia de su uso. Es muy difundida en España, en el nivel de universidad y es obligatoria para desarrollos internos del gobierno o para las contrataciones externas que realiza el gobierno. Algunos países de Latinoamérica han mostrado interés en ella debido a su naturaleza gratuita y su idioma.
Herramientas	El trabajo con RUP puede ser facilitado por una serie de herramientas propietarias que apoyan la mayoría de flujos de trabajo. Existen herramientas de código abierto y uso libre apoyan algunos flujos de trabajo o la notación UML.	Métrica no es apoyado por herramientas propietarias.

4.1.6 Reflexiones

- Dado el análisis anterior se puede observar que RUP y Métrica son metodologías de amplia trayectoria, que han marcado camino en la definición de procesos de desarrollo de software tanto para la industria como para el sector educativo.
- Al ser RUP una metodología de interés comercial, ha contado con una plataforma de inversión más amplia que MÉTRICA, esto ha provocado que el soporte que brinda RUP sea más amplio y sólido, incorporando herramientas especializadas, guías, ejemplos y más.
- Ambas metodologías son muy amplias en su definición de actividades y productos, esto puede aturdir a quienes la estudian o desean implantarlas, haciendo incierto o confuso por dónde iniciar.

4.2 Beneficios de una definición de proceso de software

A continuación se discute la importancia de contar con una adecuada definición de proceso de software, al trabajar en proyectos de sistemas intensivos en software.

4.2.1 Estructura definida

1. Contar con una definición de proceso de software que describa elementos como actividades, roles, productos, notaciones, entre otros, permite que los miembros del equipo se concentren en la tarea y no tengan que preocuparse o les genere incertidumbre el organizarse o definir la tarea durante la ejecución.
2. La base de la mejora continua en la construcción de software es contar con una definición de proceso que permita ejecutarlo e identificar claramente sus puntos débiles, con el propósito de mejorarlos.
3. La base para incorporar actividades de protección o flujos de trabajo relacionados con aseguramiento de la calidad, control de versiones, planeación y seguimiento o medición es contar con una definición de proceso que describa detalladamente las actividades fundamentales de requerimientos, análisis, diseño, construcción e implementación. Esta definición permite identificar los puntos donde es posible (y deseable) incorporar las actividades de protección y buscar las técnicas más convenientes.
4. El proceso de software de una empresa determinada refleja sus prácticas y principios, al ser estas incorporadas como hábitos de los miembros de los equipos de trabajo. Por ello es importante analizar con detenimiento si las prácticas y principios corresponden con los objetivos de la empresa.

4.2.2 Uso de estándares y notación

5. El uso de estándares permite a los miembros del equipo desarrollar y a los usuarios hablar en los mismos términos. Esto cada día es más necesario, debido a que los equipos se han vuelto multidisciplinarios y podrían involucrar a varias empresas.
6. El proceso de desarrollo de software es una secuencia de actividades relacionadas con los flujos de trabajo de requerimientos, análisis, diseño, implementación y pruebas. Cada flujo realiza una abstracción o representación de la necesidad; al pasar de un flujo a otro se corre el riesgo de representar la abstracción en una notación que pierda datos de interés, por ello es importante contar con notaciones que permitan dar continuidad sin pérdida de información importante y faciliten a los miembros del equipo entender la notación en que se expresan los artefactos provenientes de flujos anteriores a su participación.
7. Si se realiza mantenimiento o ampliación del software, el uso de estándares y una notación permitirá a nuevos encargados familiarizarse más rápidamente con lo desarrollado previamente y continuar con el mismo lineamiento.
8. Por medio de estándares y una notación se definen de antemano los criterios que deben cumplir los productos para satisfacer los objetivos de calidad. Esto facilita el desarrollo y la revisión de productos durante la ejecución del proceso.

Bibliografía

[Abrahamsson 2002] Abrahamsson, P.; Salo, O.; Ronkainen J., Agile Software Development Methods. Review and Analysis, VVT Electronics Publications 478, 2002.

[Arlow 2002] Arlow, J.; Neustadt, I., UML and the Unified Process, Addison Wesley. 2002.

[Beck 2000] Beck, K., Una explicación de la Programación Extrema. Aceptar el cambio, Pearson Educación, 2000.

[Boehm 1988] Boehm, B., A Spiral Model of Software Development and Enhancement, IEEE Computer. 21(5) 61 – 72, 1988.

[Cockburn 2002] Cockburn, A., Agile Software Development, Addison Wesley, 2002.

[Euromethod Project 1996] Euromethod Project, Euromethod version 1, X Congreso Mundial de Tecnologías de la Información, 1996.

[Herrero 2003] Herrero, M., Metodologías de desarrollo. Apuntes curso Metodologías de desarrollo, Universidad Politécnica de Valencia, España, 2003.

[Highsmith 2000] Highsmith, J., Adaptive Software Development: A Collaboration Approach, Dorset House, 2000.

[Hirsch 2002] Hirsch, M., Making RUP Agile, Conference on Object Oriented Programming Systems, Languages and Applications. ACM, OOPSLA Practitioners Reports, 2002.

[IEEE 1993] Institute of Electrical and Electronic Engineers. IEEE Standard 610: IEEE Standard Glossary of Software Engineering Terminology, IEEE Standard Collection: software Engineering 610.12-1990, revisado en 2002.

[ISO/IEC 1998] ISO/IEC, ISO/IEC TR 15504: Information Technology-Software process assessment, 1998.

[ISO/IEC/JTC1 1995] ISO/IEC/JTC1, ISO/IEC 12207: Information technology- Software life cycle processes, 1995.

[Jacobson 2000] Jacobson, I.; Booch, G.; Rumbaugh, J., El Proceso Unificado de Desarrollo de Software, Pearson Educación, 2000.

[Kruchten 1996] Kruchten, P., A Rational Development Process, Crosstalk, 9 (7), pp.11-16, July 1996.

[Kruchten 2000] Kruchten, P., The Rational Unified Process: An Introduction, Addison Wesley, 2000.

[Laboratorio Ing. Sof. 2002] Laboratorio Ing. Software, Apuntes Curso Ingeniería de Software 2, Departamento de Informática del Laboratorio de Ing. en sistemas, Madrid, España, 2002.

[Larman 2002] Larman, C., Applying UML and patterns, Prentice Hall, 2002.

[Letelier 2002 A] Letelier, P., Introducción al Rational Unified Process, Apuntes curso Laboratorio de sistemas, Universidad Politécnica de Valencia, España, 2002.

[Letelier 2002 B.] Letelier, P., Introducción a MÉTRICA 3.0. Curso Metodología MÉTRICA 3.0, Universidad Politécnica de Valencia, España, 2002.

[Luna 2000] Luna, A.; Trejos, I., El modelo de objetos: Lenguaje de modelado Unificado (UML). Informe número 29, Club de Investigación Tecnológica, Costa Rica, 2000.

[McConnell 1998] McConnell, Steve. Software project survival guide. Microsoft Press, 1998.

[Mills 1980] Mills H.; O'Neil D., The Management of Software Engineering, 24(2), p. 414-477, 1980.

[Ministerio de Administraciones Públicas 2001 A] Ministerio de Administraciones Públicas, Métrica Versión 3. Interfaces Técnicas y Prácticas, Ministerio de Administraciones Públicas, 2001.

[Ministerio de Administraciones Públicas 2001 B]: Ministerio de Administraciones Públicas, Métrica Versión 3. Guía de usuario, Ministerio de Administraciones Públicas, 2001.

[Ministerio de Administraciones Públicas 2001 C]: Ministerio de Administraciones Públicas, Métrica Versión 3 Guía de Referencia, Ministerio de Administraciones Públicas, 2001.

[Naur 1969] Naur, P.; Randell B., Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee, 1969.

[O'Reilly 1999] O'Reilly, T., Lessons from Open Source Software Development, ACM Press, 1999.

[Palmer 2002] Palmer, S.; Felsing J., A Practical Guide to Feature Driven Development, Prentice Hall, 2002.

[Pressman 1997] Pressman, R., Ingeniería del software: Un enfoque práctico, McGraw Hill, 1997.

[Pressman 2001] Pressman, R., Software Engineering. A Practitioner' s Approach,McGraw Hill, 2001.

[Rational1998] Rational Software Corporation, Rational Unified Process. Best Practices for Software Development Teams, Rational Software White Paper TP026B, Rev 11/01, 1998.

[Rational2002] Rational Software Corporation, Product: Rational Unified Process, Rational Software Corporation, 2002.

[Royce 1970] Royce, W., Managing the development of large software systems: concepts and techniques, IEEE WESTCON, 1970.

[Schwaber 1995] Schwaber, K., Scrum Development Process. Workshop on Business Object Design and Implementation, 1995.

[Schwaber 2002.] Schwaber, K.; Beedle M., Agile Software Development with Scrum, Prentice-Hall, 2002.

[Sommerville 2002] Sommerville, I., Ingeniería de software, Pearson Educación, 2002.

[Stapleton 1997] Stapleton, J., Dynamic Systems Development Method-The Method in Practice, Addison Wesley, 1997.

[Tardieu 1986] Tardieu, H.; Rochfeld A.; Coletti R., La Méthode Merise. Tome 1: Principes et Outils, Editions d'Organisation, 1986.