
2.1. Presentación del capítulo

Este capítulo proporciona una visión de conjunto concisa del proceso unificado (UP). Los principiantes deberían empezar aprendiendo la historia de UP. Si ya la conoce, puede decidir saltar hasta el apartado correspondiente a la presentación de UP y el Proceso Racional Unificado (*Rational Unified Process* o RUP) o el apartado que trata cómo puede aplicar UP en su proyecto.

Nuestro interés en UP, lo que se refiere a este libro, es proporcionar un marco de trabajo del proceso dentro del que se pueden presentar las técnicas de análisis y diseño orientados a objetos. Encontrará una presentación completa de UP en [Jacobson 1] y sobre RUP en [Kroll 1], [Kruchten 2] y también en [Ambler 1], [Ambler 2] y [Ambler 3].

2.2. ¿Qué es UP?

Un proceso de ingeniería de software, también conocido como un proceso de desarrollo de software, define el quién, qué, cuándo y cómo del desarrollo de software. Como se ilustra en la figura 2.2, un proceso de ingeniería de software es el proceso en el que convertimos los requisitos de usuario en software.

El Proceso Unificado de Desarrollo de Software (*Unified Software Development Process*, USDP) es un proceso de ingeniería de software de los autores del UML.

Comúnmente se le conoce como Proceso Unificado o UP [Jacobson 1]. Utilizaremos el término UP a lo largo del libro.

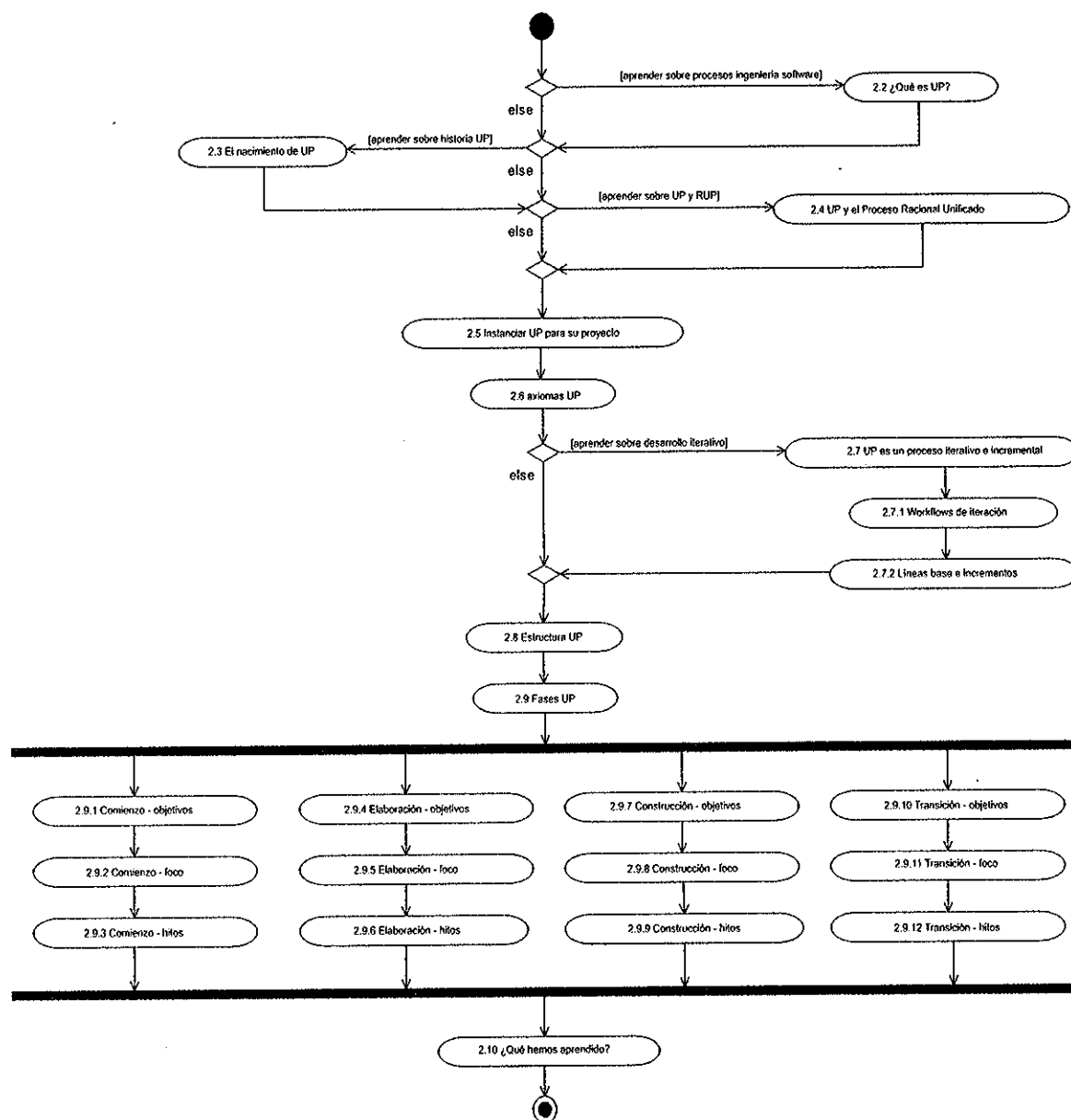
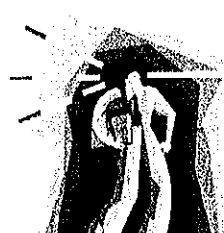
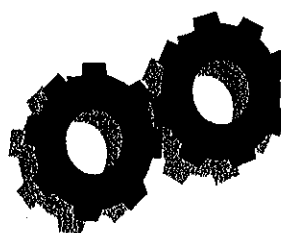


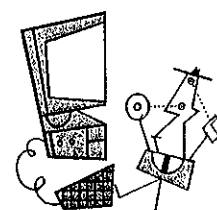
Figura 2.1.



Visión y
requisitos



Proceso de
ingeniería
de software



Software

Figura 2.2.

El proyecto UML se pensó para proporcionar un lenguaje visual y un proceso de ingeniería de software. Lo que conocemos hoy como UML es la parte del lenguaje visual del proyecto, UP es la parte del proceso. Sin embargo, merece la pena destacar que mientras que UML se ha estandarizado por el OMG, no así UP. Por lo tanto, no existe proceso de ingeniería de software estándar para complementar UML.

UP se basa en el trabajo de procesos llevado a cabo en Ericsson (el enfoque de Ericsson, 1967), Rational (el *Rational Objectory Process*, 1996 a 1997) y otras fuentes de buenas prácticas. Como tal, se trata de un método pragmático y probado para desarrollar software que incorpora buenas prácticas de sus predecesores.

2.3. El nacimiento de UP

Cuando examinamos la historia de UP, mostrada en la figura 2.3, es muy fácil decir que su desarrollo está íntimamente unido a la carrera de un hombre, Ivar Jacobson. De hecho, se piensa que Jacobson fue el padre de UP. Esto no pretende minimizar el trabajo del resto de personas (especialmente Booch) que han contribuido al desarrollo de UP; sino que lo que se pretende es enfatizar la contribución fundamental de Jacobson.

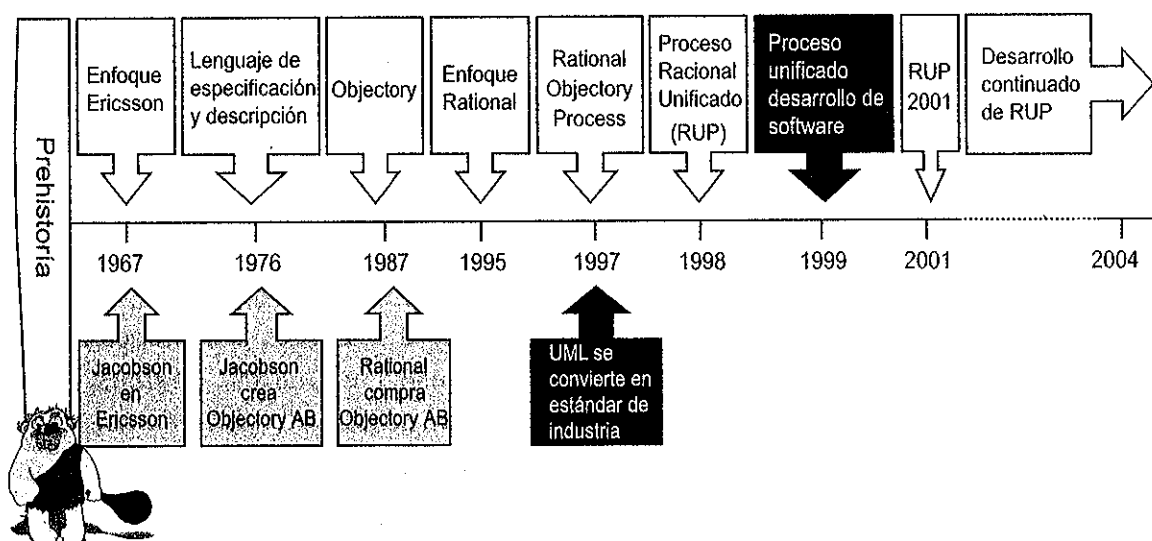


Figura 2.3.

UP se remonta a 1967 y el enfoque Ericsson, que dio el paso radical de modelar un sistema complejo como un conjunto de bloques interconectados. Los bloques pequeños estaban interconectados para formar bloques más grandes para construir un sistema completo. La base de este enfoque era "divide y vencerás" y fue el precursor de lo que se conoce hoy como desarrollo basado en componente.

Aunque un sistema completo podría ser incomprensible a cualquier persona que se acerca a él como un bloque, cuando se divide en pequeños bloques, puede tener sentido al entender los servicios que ofrece cada bloque (la interfaz al componente en terminología moderna) y cómo estos bloques encajan entre sí. En el lenguaje de

UML, los bloques grandes se denominan subsistemas, y cada subsistema se implementa en términos de bloques más pequeños denominados componentes.

Otra innovación de Ericsson fue una forma de identificar estos bloques al crear "casos de tráfico" que describían cómo se tenía que utilizar el sistema. Estos casos de tráfico han evolucionado con el tiempo y ahora se denominan casos de uso en UML. El resultado de este proceso fue una representación de arquitectura que describía todos los bloques y cómo encajaban entre sí. Esto fue el precursor del modelo estático de UML.

Además de la vista de requisitos (los casos de tráfico) y la vista estática (la descripción de arquitectura), Ericsson contaba con una vista dinámica que describía cómo todos los bloques se comunicaban entre sí con el tiempo. Esto consistía en diagramas de secuencia, comunicación y de estado de máquina que todavía se siguen encontrando en UML, aunque de forma mucho más elaborada.

El siguiente desarrollo importante en ingeniería de software orientada a objetos fue en 1980 con la aparición del Lenguaje de Especificación y Descripción (*Specification and Description Language*, SDL) del organismo internacional de estándares CCITT. SDL fue uno de los primeros lenguajes de modelado visual basado en objetos, y en 1992 se amplió para convertirse en orientado a objetos con clases y herencia. Este lenguaje se diseñó para capturar el comportamiento de sistemas de telecomunicación. Los sistemas se modelaron como un conjunto de bloques que se comunicaban al enviar señales entre sí. SDL-92 fue el primer estándar de modelado de objetos aceptado y se sigue utilizando en la actualidad.

En 1987 Jacobson fundó Objectory AB en Estocolmo. Esta empresa desarrolló y vendió un proceso de ingeniería de software, basado en el enfoque Ericsson, denominado Objectory (*Object Factory*). El proceso de ingeniería de software de Objectory constaba de un conjunto de documentación, una herramienta bastante idiosincrásica y mucha más consultoría de Objectory AB.

Quizás la innovación más importante durante este tiempo fue que el proceso de ingeniería de software de Objectory se vio como un sistema por derecho propio. Los workflows, o flujos de trabajo, del proceso (requisitos, análisis, diseño, implementación y prueba) se expresaban en un conjunto de diagramas. En otras palabras, el proceso Objectory se modeló y desarrolló como un sistema de software. Esto preparó el camino para el futuro desarrollo del proceso. Objectory, como UP, fue también un marco de trabajo de un proceso y necesitaba personalización antes de que se pudiera aplicar a cualquier proyecto específico. El producto del proceso Objectory apareció con algunas plantillas para varios tipos de proyecto de desarrollo de software, pero invariablemente se tenía que personalizar aún más. Jacobson reconoció que todos los proyectos de desarrollo de software son diferentes y por lo tanto un proceso de ingeniería de software del tipo "sirve para todos" no era ni posible ni deseable.

Cuando Rational adquirió Objectory AB en 1995, Jacobson se puso a trabajar unificando el proceso Objectory con la amplia cantidad de trabajo relacionado con el proceso que ya se había realizado en Rational. Se desarrolló una vista 4+1 de arquitectura basada en cuatro vistas distintas (lógica, proceso, física y desarrollo) y una vista de caso de uso unificadora. Esto todavía sigue formando la base del

enfoque UP para la arquitectura de sistema. Además, el desarrollo iterativo se formalizó en una secuencia de fases (Comienzo, Elaboración, Construcción y Transición) que combinaba la disciplina del ciclo de vida de cascada con el grado de reacción dinámico del desarrollo iterativo e incremental. Los principales participantes en este trabajo fueron Walter Royce, Rich Reitmann, Grady Booch (inventor del método Booch) y Philippe Kruchten. En particular, la experiencia de Booch y sus ideas sobre arquitectura se incorporaron en el enfoque Rational (véase [Booch 1] para una excelente presentación de sus ideas).

El Rational Objectory Process (ROP) fue el resultado de la unificación de Objectory con el trabajo del proceso de Rational. En particular, ROP mejora áreas en las que Objectory no era tan fuerte, requisitos, implementación, prueba, gestión de proyectos, despliegue, gestión de configuración y entorno de desarrollo. El riesgo se introdujo en ROP y la arquitectura de definió y formalizó como una "descripción de arquitectura" distribuable. Durante este período Booch, Jacobson y Rumbaugh desarrollaron UML en Rational. Éste se convirtió en el lenguaje en el que se expresaron los modelos ROP y el propio ROP.

Desde 1997 en adelante, Rational adquirió muchas más empresas que aportaban su experiencia en captura de requisitos, gestión de configuración, pruebas, etc. Esto llevó a la aparición del Rational Unified Process o RUP (Proceso Racional Unificado) en 1998. Desde entonces, ha habido muchas versiones de RUP, cada una de ellas mejores que la anterior. Consulte www.rational.com y [Kruchten 1] para más detalles.

En 1999, vimos la publicación de un libro importante, *The Unified Software Development Process* [Jacobson 1] que describe el proceso unificado.

Mientras que RUP es un producto de proceso Rational, UP es un proceso de ingeniería de software abierto de los autores de UML. No sorprendentemente, UP y RUP están íntimamente relacionados. Hemos decidido utilizar UP en lugar de RUP en este libro ya que un proceso de ingeniería de software abierto, accesible a todos y no está unido a ningún producto o vendedor específico.

2.4. UP y el Proceso Racional Unificado

El Proceso Racional Unificado (*Rational Unified Process*, RUP) es una versión comercial de UP de IBM que adquirió Rational Corporation en 2003. Proporciona todos los estándares, herramientas y otras necesidades que no están incluidas en UP y que tendría que proporcionar de cualquier modo. También se distribuye con un entorno rico basado en Web que incluye documentación completa del proceso y "mentores de herramientas" para cada una de las herramientas.

En 1999 RUP era una implementación bastante sencilla de UP. Sin embargo, RUP ha cambiado mucho desde entonces y ahora amplía UP en muchas formas importantes. En la actualidad, deberíamos ver UP como el caso general abierto y RUP como una subclase comercial específica que extiende y anula las características de UP. Pero, RUP y UP siguen siendo mucho más similares que diferentes. Las diferen-

cias principales son las que se refieren a la integridad y detalle en lugar de diferencias semánticas o ideológicas. Los workflows básicos del análisis y diseño orientados a objetos son suficientemente similares que una descripción desde una perspectiva UP sería de igual utilidad para los usuarios RUP. Al elegir utilizar UP en este libro, hacemos que el texto se adapte a la mayoría de los analistas y diseñadores orientados a objetos que no utilizan RUP y también para la minoría creciente que son.

UP y RUP modelan el quién, cuándo y qué del proceso de desarrollo de software, pero lo hacen de forma ligeramente diferente. La última versión de RUP tiene algunas diferencias terminológicas y sintácticas de UP, aunque la semántica de los elementos del proceso permanece igual.

La figura 2.4 muestra cómo los iconos del proceso RUP se mapean con los iconos UP que utilizamos en este libro. Observe también que existe una relación <<trace>> entre el icono RUP y el icono UP original. En UML, una relación <<trace>> es un tipo especial de dependencia entre elementos de modelo que indica que el elemento al principio de la relación <<trace>> es un desarrollo histórico del elemento apuntado por la flecha. Esto describe la relación entre los elementos del modelo UP y RUP perfectamente.








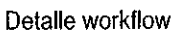
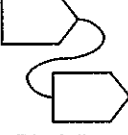
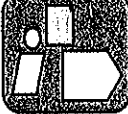
UP	RUP	Semántica
 Trabajador	 Rol	Quién: Un rol en el proyecto desempeñado por un persona o equipo
 Actividad  Artefacto	 Actividad  Artefacto	Qué: Una unidad de trabajo realizada por un trabajador (rol) o un artefacto producido en el proyecto
 Workflow  Detalle workflow	 Disciplina  Detalle workflow	Cuándo: Una secuencia de actividades relacionadas que aportan valor al proyecto

Figura 2.4.

Para modelar el "quién" del proceso de ingeniería de software, UP introduce el concepto del recurso (trabajador). Esto describe un rol desempeñado por una per-

sona o equipo dentro del proyecto. Cada recurso puede realizarse por muchas personas o equipos y cada persona o equipo puede realizarse como muchos diferentes recursos. En RUP se denominan "roles", pero la semántica sigue siendo la misma.

UP modela el "qué" como actividades y artefactos. Las actividades son tareas que realizarán personas o equipos en el proyecto. Estas personas o equipos adoptarán roles específicos cuando realizan ciertas actividades y, por lo tanto, para cualquier actividad, UP (y RUP) nos pueden decir los roles que participan en esa actividad. Las actividades se pueden desglosar en niveles de detalle según sea necesario. Los artefactos son elementos como entradas y salidas del proyecto; pueden ser código fuente, programas ejecutables, estándares, documentación, etc. Pueden tener muchos iconos diferentes dependiendo de lo que sean, y en la figura 2.4 los mostramos con un icono genérico de documento.

UP modela el "cuándo" como workflows. Éstos son secuencias de actividades relacionadas que realizan los roles. En RUP, a los workflows de alto nivel, como los Requisitos o Prueba, se les asigna un nombre especial, disciplinas. Los workflows se pueden desglosar en uno o más detalles de workflow que describen las actividades, roles y artefactos implicados en el workflow. Estos detalles de workflow solamente se refieren por nombre en UP pero se les ha asignado su propio icono en RUP.

2.5. Instanciar UP para su proyecto

UP es un proceso de desarrollo de software genérico que se tiene que instanciar para una empresa y luego para cada proyecto en particular. Esto reconoce que todos los proyectos de software tienden a ser diferentes y que ese enfoque de "uno vale para todos" del proceso de ingeniería de software no funciona. El proceso de instanciación implica definir e incorporar:

- Estándares propios.
- Plantillas de documento.
- Herramientas, compiladores, herramientas de gestión de configuración, etc.
- Bases de datos, seguimiento de errores, seguimiento del proyecto, etc.
- Modificaciones del ciclo de vida, por ejemplo, control de calidad más sofisticado para sistemas críticos en seguridad.

Los detalles de este proceso de personalización están fuera del alcance de este libro, pero se describen en [Rumbaugh 1].

Aunque RUP es mucho más completo que UP, todavía se tiene que personalizar e instanciar de forma similar. Sin embargo, la cantidad de trabajo que se tiene que hacer es mucho menos que empezar a partir de UP sin formato. De hecho, con cualquier proceso de ingeniería de software puede esperar invertir cierta cantidad

de tiempo y dinero en la instanciación y necesitará presupuestar consultoría con el vendedor del proceso de ingeniería de software para ayudarle con esto.

2.6. Axiomas de UP

UP tiene tres axiomas básicos:

- Está dirigido por requisitos y riesgo.
- Está centrado en arquitectura.
- Es iterativo e incremental.

Examinaremos los casos de uso en gran detalle en el capítulo 4, pero por ahora, simplemente digamos que son una forma de capturar requisitos, por lo que podemos decir que UP está dirigido por requisitos.

UP también está dirigido por riesgo porque si no se enfrenta a los riesgos de forma activa, ellos le atacarán a usted. Cualquiera que haya trabajado en un proyecto de desarrollo de software estará de acuerdo con esta afirmación, y UP aborda esto al basar la construcción de software sobre el análisis de riesgo. Sin embargo, éste es un trabajo para el director del proyecto y el arquitecto, por lo que no lo trataremos en mucho detalle en este libro.

El enfoque UP para desarrollar sistemas de software es desarrollar y evolucionar una arquitectura de sistema robusta. La arquitectura describe los aspectos estratégicos de cómo el sistema se desglosa en componentes, y como estos componentes interactúan y se despliegan en hardware. Claramente, una arquitectura de sistema de calidad llevará a un sistema de calidad, en lugar de a una colección de código fuente ad hoc que se ha recortado con poca previsión.

Por último, UP es iterativo e incremental. El aspecto iterativo de UP significa que desglosamos el proyecto en pequeños subproyectos (las iteraciones) que distribuyen funcionalidad del sistema en bloques, o incrementos, que llevan a un sistema totalmente funcional. En otras palabras, creamos software por un proceso de mejora paso a paso. Éste es un enfoque muy diferente a la construcción de software comparado con el antiguo ciclo de vida en cascada de análisis, diseño y creación que ocurre en una secuencia más o menos estricta. De hecho, regresamos a los workflows clave de UP varias veces durante el transcurso del proyecto.

2.7. UP es un proceso iterativo e incremental

Para entender UP necesitamos entender las iteraciones. La idea es fundamentalmente muy sencilla; la historia nos muestra que las personas encuentran más fácil resolver los pequeños problemas que los grandes problemas. Por lo tanto, dividimos un gran proyecto de desarrollo de software en una serie de "mini proyectos"

más pequeños que son más fáciles de gestionar y de completar con éxito. Cada uno de estos "mini proyectos" es una iteración. El punto clave es que cada iteración contiene todos los elementos de un proyecto normal de desarrollo de software:

- Planificación.
- Análisis y diseño.
- Construcción.
- Integración y prueba.
- Versión interna y externa.

Cada iteración genera una línea base que engloba una versión parcialmente completa del sistema final y cualquier documentación asociada del proyecto. Las líneas base se crean sobre siguientes iteraciones sucesivas hasta que se consigue el sistema definitivo.

La diferencia entre dos líneas base consecutivas se conoce como un incremento. Ésta es la razón por la que UP se conoce como un ciclo de vida iterativo e incremental.

Como verá en el siguiente apartado, las iteraciones se agrupan en fases. Las fases proporcionan la macroestructura de UP.

2.7.1. Workflows de iteración

En cada iteración, cinco workflows principales especifican lo que se necesita hacer y qué habilidades se necesitan para realizarlo. Al igual que los cinco workflows centrales, habrá otros como planificación, valoración y cualquier otro específico para esa iteración en particular. Sin embargo, esto no se trata en UP.

Los cinco workflows principales son:

- **Requisitos:** Capturan lo que el sistema debería hacer.
- **Análisis:** Mejora y estructura los requisitos.
- **Diseño:** Realiza los requisitos en la arquitectura del sistema.
- **Implementación:** Crea el software.
- **Prueba:** Verifica que la implementación funciona según se desea.

Algunos workflows posibles para una iteración se ilustran en la figura 2.5. Examinaremos los workflows de requisitos, análisis, diseño e implementación con más detalle más adelante en el libro.

Aunque cada iteración puede contener los cinco workflows principales, el énfasis en un workflow en particular depende de dónde ocurra la iteración en el ciclo del proyecto.

Dividir el proyecto en una serie de iteraciones permite un enfoque flexible en la planificación del proyecto. El enfoque más sencillo es una secuencia de iteraciones ordenadas en tiempo, donde cada una de ellas conduce a la siguiente. Sin embargo,

a menudo es posible programar iteraciones en paralelo. Esto implica entender las dependencias entre los artefactos de cada iteración y requiere un enfoque al desarrollo de software basado en la arquitectura y el modelado. La ventaja de las iteraciones paralelas es un mejor *time-to-market* y quizá una mejor utilización del equipo, pero es esencial una adecuada planificación.

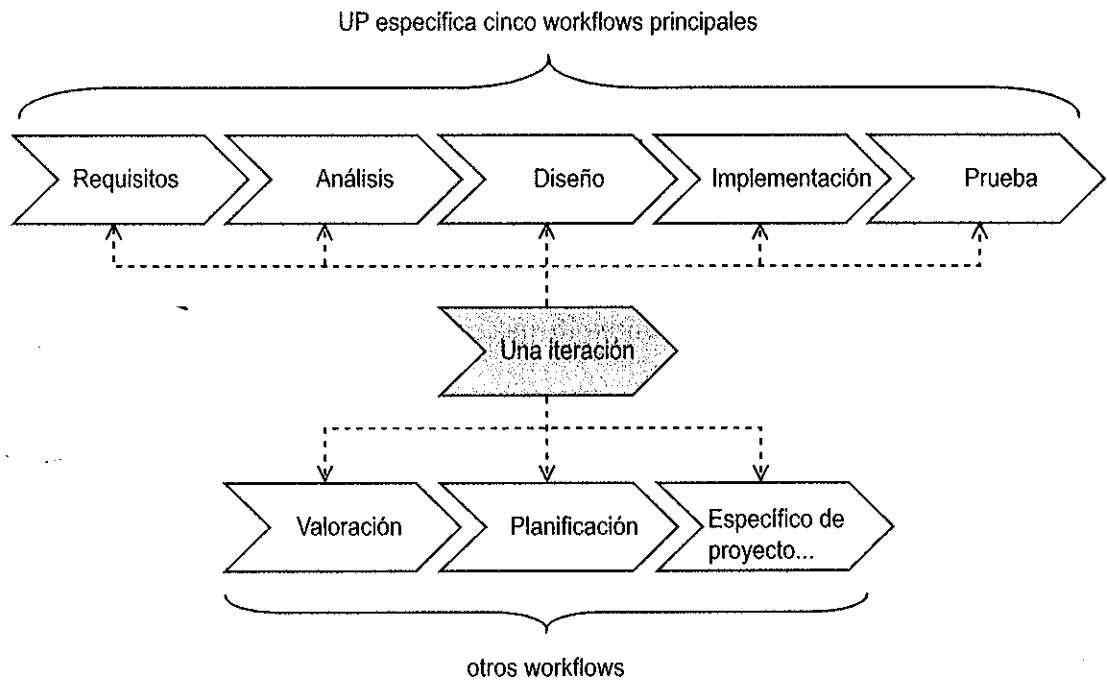


Figura 2.5.

2.7.2. Líneas base e incrementos

Toda iteración UP genera una línea base. Esto es una versión interna (o externa) del conjunto de artefactos revisados y aprobados generados por esa iteración. Cada línea base:

- Proporciona una base acordada para mayor revisión y desarrollo.
- Se puede cambiar solamente por medio de procedimientos formales de configuración y gestión del cambio.

Sin embargo, los incrementos son simplemente la diferencia entre una línea base y la siguiente. Constituyen un paso hacia el sistema final.

2.8. Estructura de UP

La figura 2.6 muestra la estructura de UP. El ciclo de vida del proyecto se divide en cuatro fases: Comienzo, Elaboración, Construcción y Transición, cada una de las cuales termina con un hito importante. Dentro de cada fase, podemos tener una o

más iteraciones y en cada iteración ejecutamos los cinco workflows principales y cualquier workflow adicional. El número exacto de iteraciones por fase depende del tamaño del proyecto, pero cada iteración debería durar no más de dos a tres meses. El ejemplo es típico de un proyecto que dura unos 18 meses y es de tamaño medio.

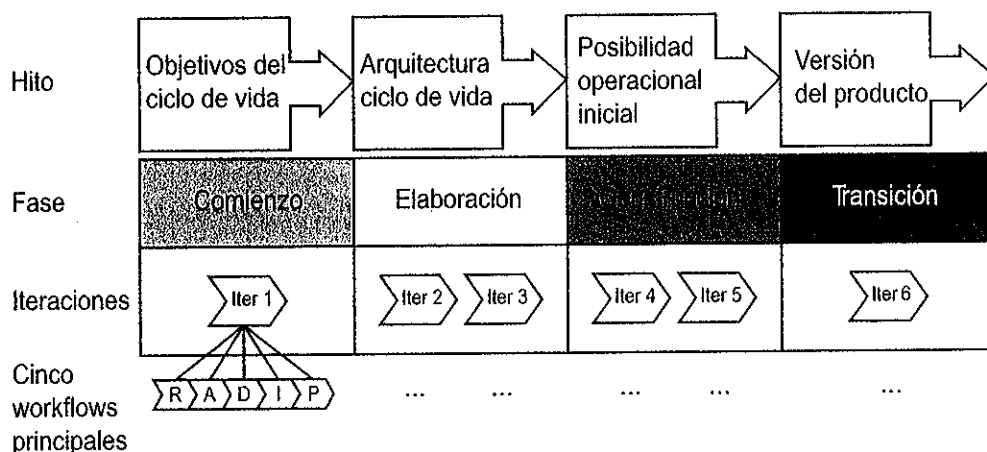


Figura 2.6.

Como podemos ver por la figura 2.6, UP consta de una secuencia de cuatro fases, cada una de las cuales termina con un hito importante:

- **Comienzo:** Objetivos del ciclo de vida.
- **Elaboración:** Arquitectura del ciclo de vida.
- **Construcción:** Posibilidad operacional inicial.
- **Transición:** Versión del producto.

A medida que el proyecto pasa por las fases del UP, la cantidad de trabajo que se realiza en cada uno de los cinco workflows principales cambia.

La figura 2.7 es la clave para entender cómo funciona UP. En la parte superior tenemos las fases. En la parte izquierda tenemos los cinco workflows principales. En la parte inferior, tenemos algunas iteraciones. Las curvas muestran la cantidad relativa de trabajo realizado en cada una de los cinco workflows principales a medida que el proyecto progresa por las fases.

Como muestra la figura 2.7, en la fase del comienzo la mayor parte del trabajo se realiza en los requisitos y análisis. En la elaboración el énfasis pasa a requisitos, análisis y algo de diseño. En la construcción, el énfasis está claramente en el diseño e implementación. Por último, en la transición el énfasis se encuentra en la implementación y prueba.

Una de las estupendas características de UP es que es un proceso basado en el objetivo en lugar de un proceso basado en entregables. Cada fase termina con un hito que consta de un conjunto de condiciones de satisfacción y estas condiciones pueden implicar la creación de un entregable particular, o no, dependiendo de las necesidades específicas de su proyecto.

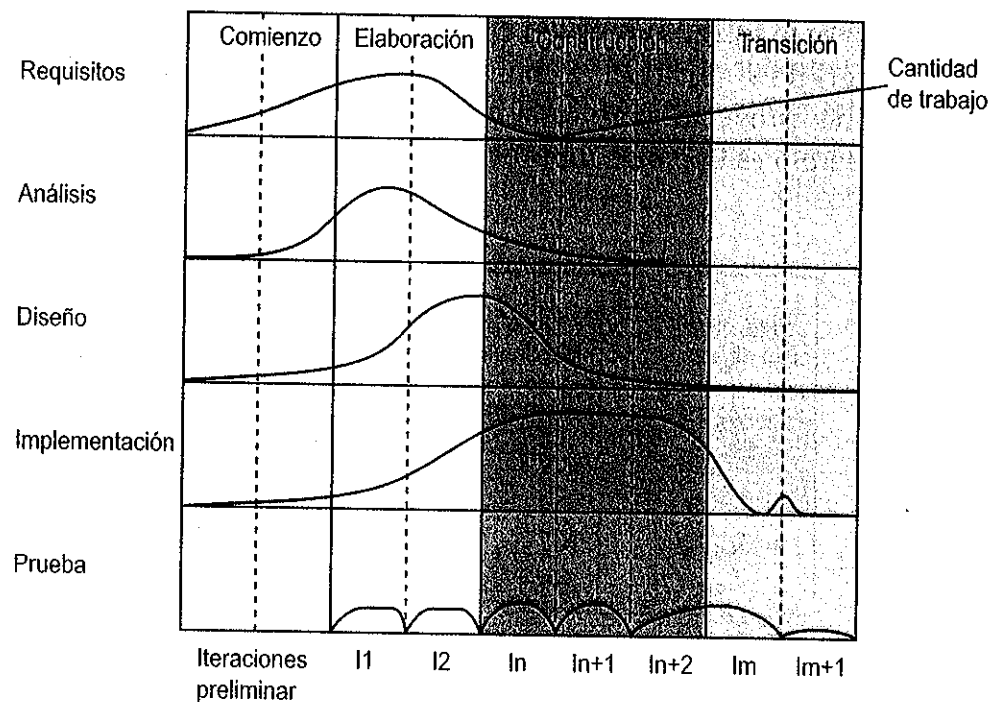


Figura 2.7. Adaptada de la figura 1.5 [Jacobson 1] con permiso de Addison-Wesley.

En el resto de este capítulo proporcionamos una breve visión de conjunto de cada una de las fases de UP.

2.9. Fases de UP

Cada fase tiene un objetivo, un foco de actividad con uno o más workflows principales enfatizados y un hito. Éste será nuestro marco de trabajo para investigar las fases.

2.9.1. Comienzo: objetivos

El objetivo de esta fase es "lanzar el proyecto". El comienzo implica:

- Establecer la viabilidad; esto puede implicar algún prototipo técnico para validar decisiones de tecnología o prueba del concepto para validar requisitos de negocio.
- Crear un business case para demostrar que el proyecto proporcionará beneficios cuantificables.
- Capturar requisitos esenciales para ayudar a definir el ámbito de aplicación del sistema.
- Identificar riesgos críticos.

Los recursos principales de esta fase es el gestor de proyecto y el responsable del sistema.

2.9.2. Comienzo: foco

El énfasis principal de esta fase se encuentra en los workflows de requisitos y análisis. Sin embargo, también se podría realizar algo de diseño e implementación si se decide crear un prototipo técnico o prueba de concepto. El workflow de prueba no es aplicable por lo general a esta fase, ya que los únicos artefactos de software son prototipos que se van a desechar.

2.9.3. Comienzo: hito. Objetivos del ciclo de vida

Mientras que muchos procesos de ingeniería de software se centran en la creación de artefactos clave, UP adopta un enfoque diferente que está orientado a objetivos. Cada hito establece ciertos objetivos que se deben alcanzar antes de que ese hito se considere alcanzado. Algunos de estos objetivos pueden ser la producción de ciertos artefactos y algunos podrían no ser eso. El hito para el comienzo son los objetivos del ciclo de vida. Las condiciones que se deben cumplir para que este hito sea alcanzable se proporcionan en la tabla 2.1. También sugerimos un conjunto de entregables que puede necesitar crear para realizar estas condiciones. Sin embargo, recuerde que solamente crea un entregable cuando añade valor a su proyecto.

Tabla 2.1.

Condiciones de satisfacción	Entregable
Los grupos de decisión se han puesto de acuerdo en los objetivos del proyecto.	Un documento que indique los requisitos principales, características y restricciones del proyecto.
El ámbito de aplicación del sistema se ha definido y se ha acordado con los grupos de decisión.	Un modelo inicial de caso de uso (solamente completo del 10 al 20 por ciento).
Se han capturado los requisitos clave y se han acordado con los grupos de decisión.	Un glosario del proyecto.
Se ha acordado con los grupos de decisión el coste y la estimación del calendario.	Un plan de proyecto inicial.
Se ha lanzado un business case por el gestor de proyecto.	Un business case.
El gestor de proyecto ha realizado un análisis de riesgo.	Un documento de análisis de riesgo o base de datos.
La viabilidad se ha confirmado por medio de estudios técnicos y/o la creación de prototipos.	Uno o más prototipos desechables.
Se ha esbozado la arquitectura.	Un documento inicial de arquitectura.

2.9.4. Elaboración: objetivos

Los objetivos de la elaboración se pueden resumir de la siguiente forma:

- Crear una línea base ejecutable de la arquitectura.
- Mejorar el análisis de riesgos.
- Definir atributos de calidad.
- Capturar casos de uso hasta el 80 por ciento de los requisitos funcionales (verá exactamente a qué se refiere en los capítulos 3 y 4).
- Crear un plan detallado de la fase de construcción.
- Formular una oferta que incluya recursos, tiempo, equipamiento, personal y coste.

El objetivo principal de la elaboración es crear una línea base ejecutable de la arquitectura. Éste es un sistema real, ejecutable que se crea de acuerdo a la arquitectura especificada. No es un prototipo (que es desechable), sino el primer modelo del sistema deseado. Esta línea base ejecutable de la arquitectura aumentará a medida que progresa el proyecto y evolucionará hacia el sistema final durante las fases de construcción y transición. Puesto que las fases futuras se basan en los resultados de la elaboración, ésta es posiblemente la fase más crítica. De hecho, este libro se centra mucho en las actividades de elaboración.

2.9.5. Elaboración: foco

En la fase de elaboración el foco en cada uno de los workflows principales es el siguiente:

- **Requisitos:** Mejora el ámbito de aplicación del sistema y requisitos.
- **Análisis:** Establece lo que se va a crear.
- **Diseño:** Crea una arquitectura estable.
- **Implementación:** Crea la línea base de la arquitectura.
- **Prueba:** Prueba la línea base de la arquitectura.

El foco en elaboración está claramente en los workflows de requisitos, análisis y diseño con la implementación siendo muy importante al final de la fase cuando se produce la línea base ejecutable de la arquitectura.

2.9.6. Elaboración: hito. Arquitectura del ciclo de vida

El hito es la arquitectura del ciclo de vida. Las condiciones de satisfacción para este hito se resumen en la tabla 2.2.

Tabla 2.2.

Condiciones de satisfacción	Entregable
Se ha creado una línea base ejecutable de la arquitectura.	La línea base ejecutable de la arquitectura.
La línea base ejecutable de la arquitectura demuestra que se han identificado riesgos importantes y se han resuelto.	Modelo UML estático, modelo UML dinámico, modelo UML de caso de uso.
La visión del producto se ha estabilizado.	Documento de visión.
El análisis de riesgos se ha revisado.	Análisis de riesgo actualizado.
El business case se ha revisado y acordado con los grupos de decisión.	Business case actualizado.
Se ha creado un plan de proyecto con suficiente detalle para permitir formular una oferta realista de tiempo, dinero y recursos en siguientes fases.	Plan de proyecto actualizado.
Los grupos de decisión están de acuerdo con el plan de proyecto.	
El business case se ha verificado con el plan de proyecto.	Business case.
Se llega a un acuerdo con los grupos de decisión para continuar el proyecto.	Documento firmado.

2.9.7. Construcción: objetivos

El objetivo de la construcción es completar todos los requisitos, análisis y diseño y evolucionar la línea base de arquitectura generada en la elaboración hacia el sistema final. Un tema clave en la construcción es mantener la integridad de la arquitectura del sistema. Es bastante común que una vez que se presenta la presión de la entrega, se empieza a atajar en la codificación lo que lleva a un sistema final de baja calidad y costes altos de mantenimiento. Claramente, este resultado se debería evitar.

2.9.8. Construcción: foco

El énfasis en esta fase está en el workflow de implementación. Ya se realiza suficiente trabajo en los otros workflows para completar la captura de requisitos, análisis y diseño. Las pruebas también se hacen más importantes, puesto que cada nuevo incremento se crea sobre el siguiente, las pruebas de unidad e integración

son necesarias. Podemos resumir el tipo de trabajo realizado en cada workflow de la siguiente manera:

- **Requisitos:** Descubrir cualquier requisito que se haya pasado por alto.
- **Análisis:** Finalizar el modelo de análisis.
- **Diseño:** Finalizar el modelo de diseño.
- **Implementación:** Crear la capacidad operativa inicial.
- **Prueba:** Probar la capacidad operativa inicial.

2.9.9. Construcción: hito. Capacidad operativa inicial

Este hito es muy sencillo, el sistema de software está terminado para una prueba beta en el sitio del usuario. Las condiciones de satisfacción de este hito se proporcionan en la tabla 2.3.

Tabla 2.3.

Condiciones de satisfacción	Entregable
El producto de software es suficientemente estable y de suficiente calidad para desplegarse en la comunidad de usuarios.	El producto de software, el modelo UML, la suite de prueba.
Los grupos de decisión se han puesto de acuerdo y están preparados para la transición del software a su entorno.	Manuales de usuario, descripción de esta versión.
Los gastos reales vs. los gastos planificados son aceptables.	Plan de proyecto.

2.9.10. Transición: objetivos

La fase de transición empieza cuando la prueba beta se completa y el sistema está finalmente desplegado. Esto implica ajustar cualquier defecto que se encuentre en la prueba beta y prepararse para presentar el software en todos los sitios del usuario.

Podemos resumir los objetivos de esta fase de la siguiente manera:

- Corregir defectos.
- Preparar los sitios del usuario para el nuevo software.
- Adaptar el software para que funcione en los sitios del usuario.

- Modificar el software si surgen problemas imprevistos.
- Crear manuales de usuario y otra documentación.
- Proporcionar consultoría al usuario.
- Realizar una revisión después del proyecto.

2.9.11. Transición: foco

El énfasis está en los workflows de implementación y prueba. Se ha realizado suficiente diseño para corregir cualquier error de diseño encontrado en la prueba beta. En este punto del ciclo de vida del proyecto, debería haber poco trabajo en los workflows de requisitos y análisis. Si éste no es el caso, el proyecto está en problemas.

- **Requisitos:** No aplicable.
- **Análisis:** No aplicable.
- **Diseño:** Modifique el diseño si aparecen problemas en la prueba beta.
- **Implementación:** Adapte el software para el sitio del usuario y corrija problemas descubiertos en la prueba beta.
- **Prueba:** Prueba beta y prueba de aceptación en el sitio del usuario.

2.9.12. Transición: hito. Versión del producto

Éste es el último hito: prueba beta, prueba de aceptación y reparación de defectos está terminado y se lanza el producto en la comunidad de usuarios. Las condiciones de satisfacción para este hito se proporcionan en la tabla 2.4.

Tabla 2.4.

Condiciones de satisfacción	Entregable
Se ha completado la prueba beta, se han realizado los cambios necesarios y los usuarios están de acuerdo en que el sistema se ha desplegado con éxito.	El producto de software.
La comunidad de usuarios está utilizando activamente el producto.	
Se han acordado estrategias de soporte del producto con los usuarios y se han implementado.	Plan de soporte de usuario, manuales de usuario actualizados.

2.10. ¿Qué hemos aprendido?

- Un proceso de ingeniería de software convierte los requisitos de usuario en software al especificar quién hace qué, cuándo.
- El proceso unificado (UP) lleva en desarrollo desde 1967. Se trata de un proceso de ingeniería de software abierto y maduro de los autores de UML.
- El Proceso Racional Unificado (RUP) es una extensión comercial de UP. Es totalmente compatible con UP pero es más completo y detallado.
- UP (y RUP) se debe instanciar para cualquier proyecto específico al añadir estándares internos, etc.
- UP es un proceso de ingeniería de software moderno que:
 - Está dirigido por riesgo y caso de uso (requisitos).
 - Está centrado en la arquitectura.
 - Es iterativo e incremental.
- El software UP se crea en iteraciones:
 - Cada iteración es como un "mini proyecto" que entrega una parte del sistema.
 - Las iteraciones se crean sobre otras para crear el sistema final.
- Cada iteración tiene cinco workflows principales:
 - Requisitos: Capturar lo que el sistema debería hacer.
 - Análisis: Mejorar y estructurar los requisitos.
 - Diseño: Realizar los requisitos en la arquitectura del sistema (cómo el sistema lo hace).
 - Implementación: Crear el software.
 - Prueba: Verificar que la implementación funciona como era de desear.
- UP tiene cuatro fases, cada una de las cuales termina con un hito importante:
 - Comienzo: Lanzar el proyecto: objetivos del ciclo de vida.
 - Elaboración: Evolucionar la arquitectura del sistema: arquitectura del ciclo de vida.
 - Construcción: Crear el software: capacidad operativa inicial.
 - Transición: Desplegar el software en el entorno del usuario: versión del producto.

Parte 2

Requisitos

ario en

de un
UML.

UP. Es

añadir

rte del

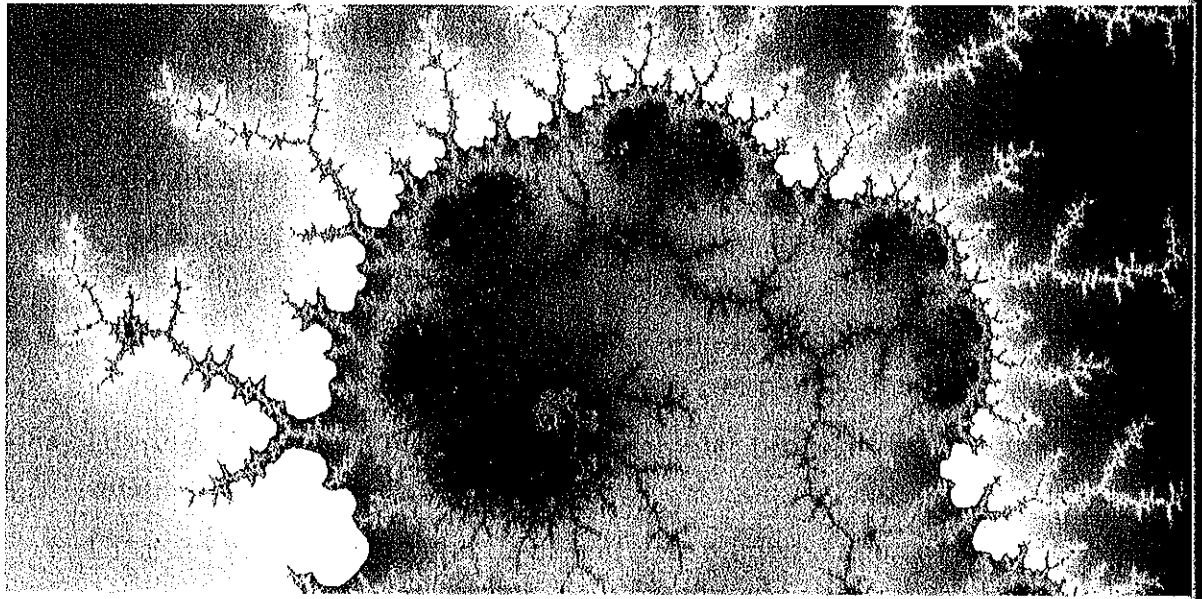
ómo el

ear.

rtante:

del ci-

ón del



El workflow de requisitos
