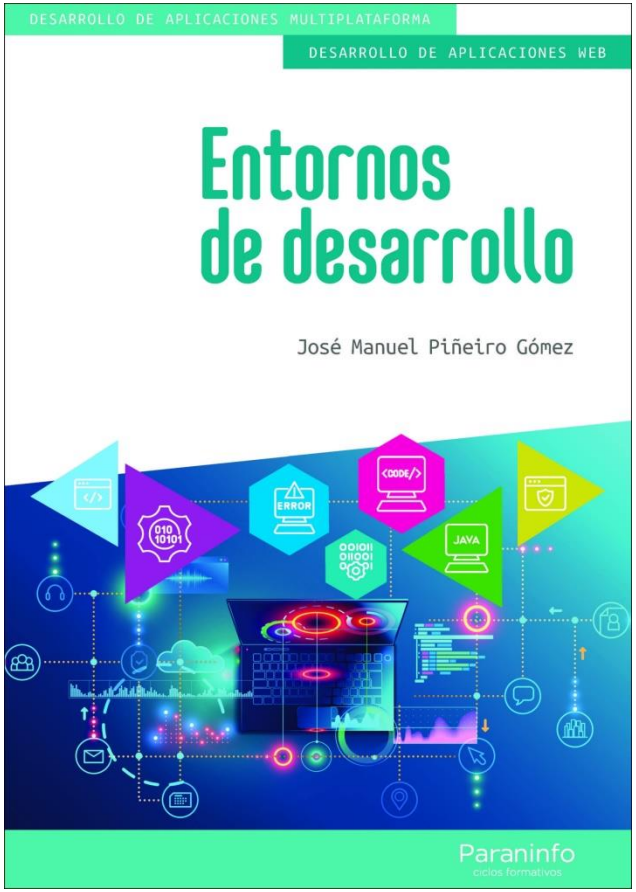
Paraninfo



UNIDAD 1

Desarrollo de software

Paraninfo

Contenidos

1.1. El software y su relación con otras partes del ordenador

Hardware y software

Software:

- Instrucciones.
- Estructuras de datos.
- Documentación.

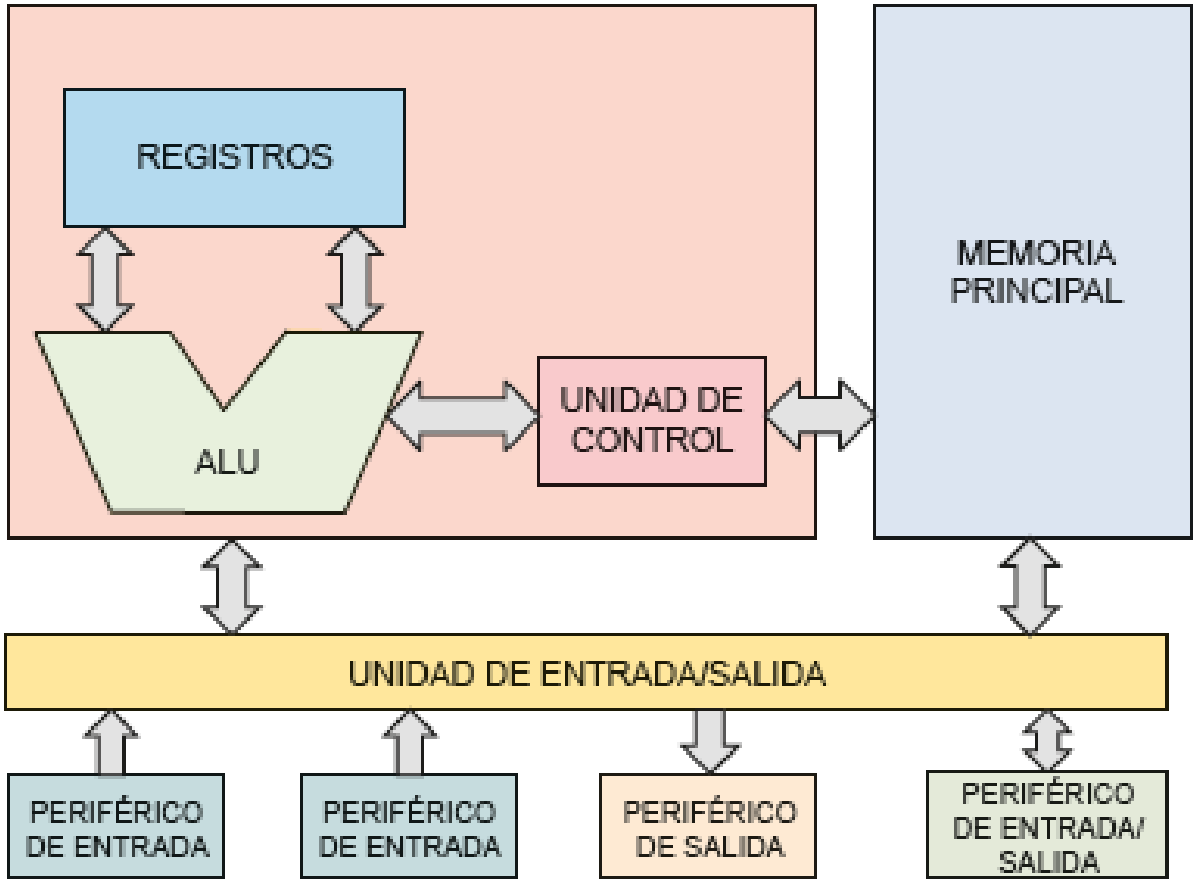


Figura 1.1. La estructura de un ordenador consta de una CPU, dividida en: unidad de control, unidad aritmético-lógica (ALU) y registros; la memoria principal o memoria RAM y la unidad de entrada/salida, que permite la comunicación del ordenador con el exterior gracias a la conexión de los periféricos con esta unidad. Los diversos elementos del hardware de un ordenador se conectan por medio de unas líneas llamadas buses por las que circulan los bits.

1.2. Lenguajes de programación. Tipos

Definición de lenguaje de programación

Elementos básicos

Identificadores	Constantes	Operadores	Instrucciones	Comentarios

Tipos de lenguajes según su cercanía al lenguaje natural

Lenguajes de bajo nivel o máquina	Lenguajes intermedios o ensambladores	Lenguajes de alto nivel o evolucionados

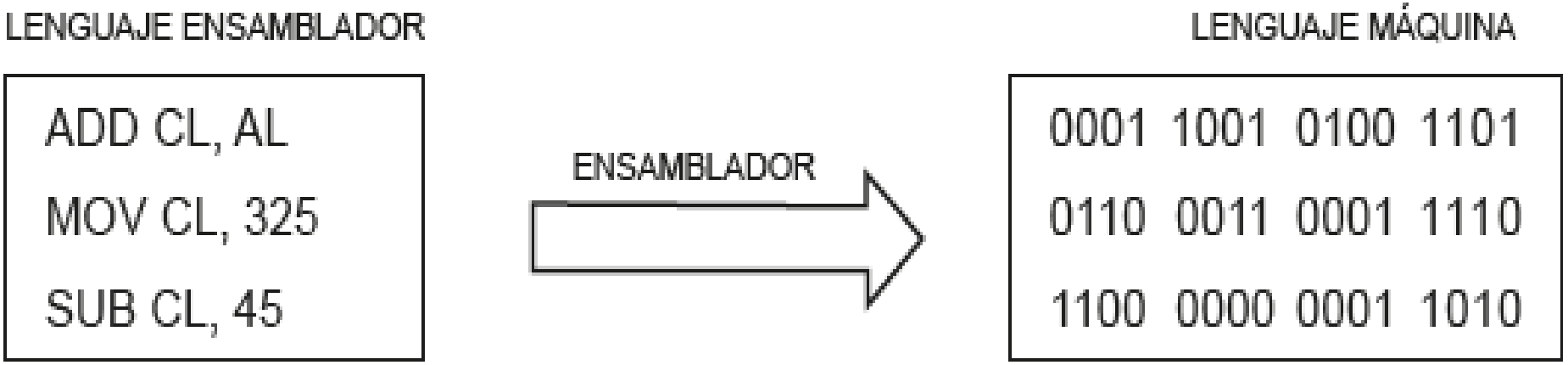


Figura 1.2. Las instrucciones en lenguaje ensamblador están escritas empleando códigos mnemotécnicos (*add* indica sumar; *mov*, mover; etc.) y cada instrucción corresponde a una instrucción en lenguaje máquina. Para pasar del lenguaje ensamblador al lenguaje máquina se usa un traductor llamado ensamblador.

Figura 1.3. Nombres de algunos lenguajes de programación de alto nivel.

Tipos de lenguajes de alto nivel según el paradigma de programación

- **Lenguajes estructurados.**

  - Una aplicación → conjunto de módulos (procedimientos o funciones) que se comunican mediante llamadas (diagramas de estructuras).

- **Lenguajes orientados a objetos.**

  - Una aplicación → conjunto de objetos (instancias de clases) que se envían mensajes. Un objeto está definido por su estado y su comportamiento.

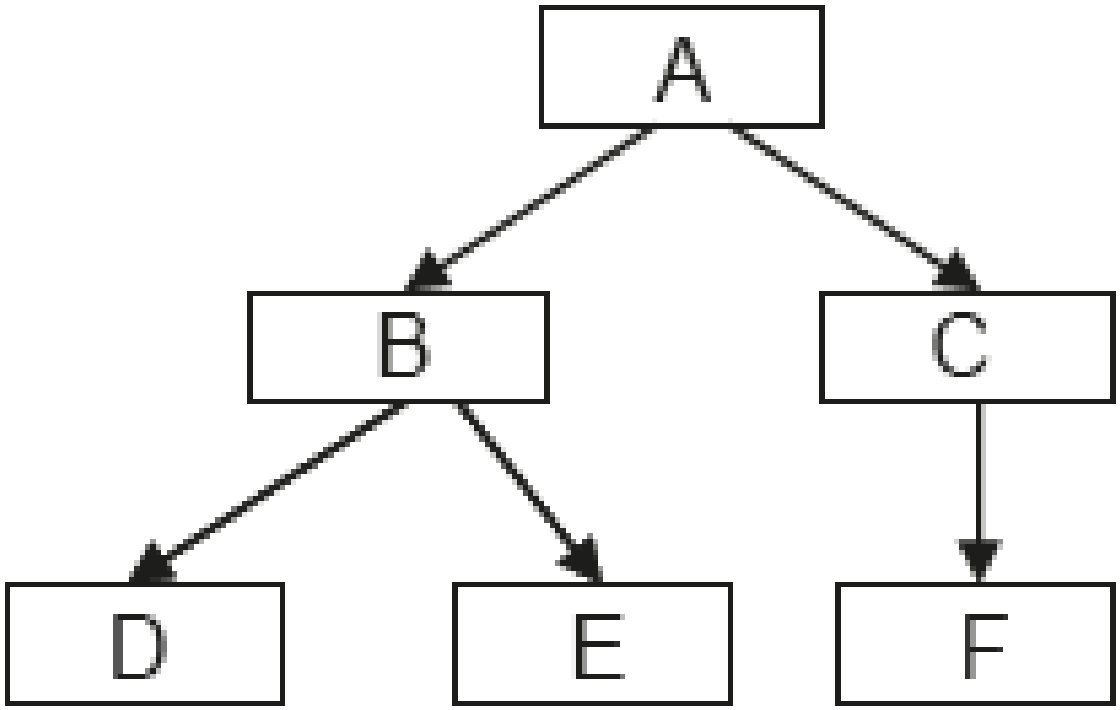
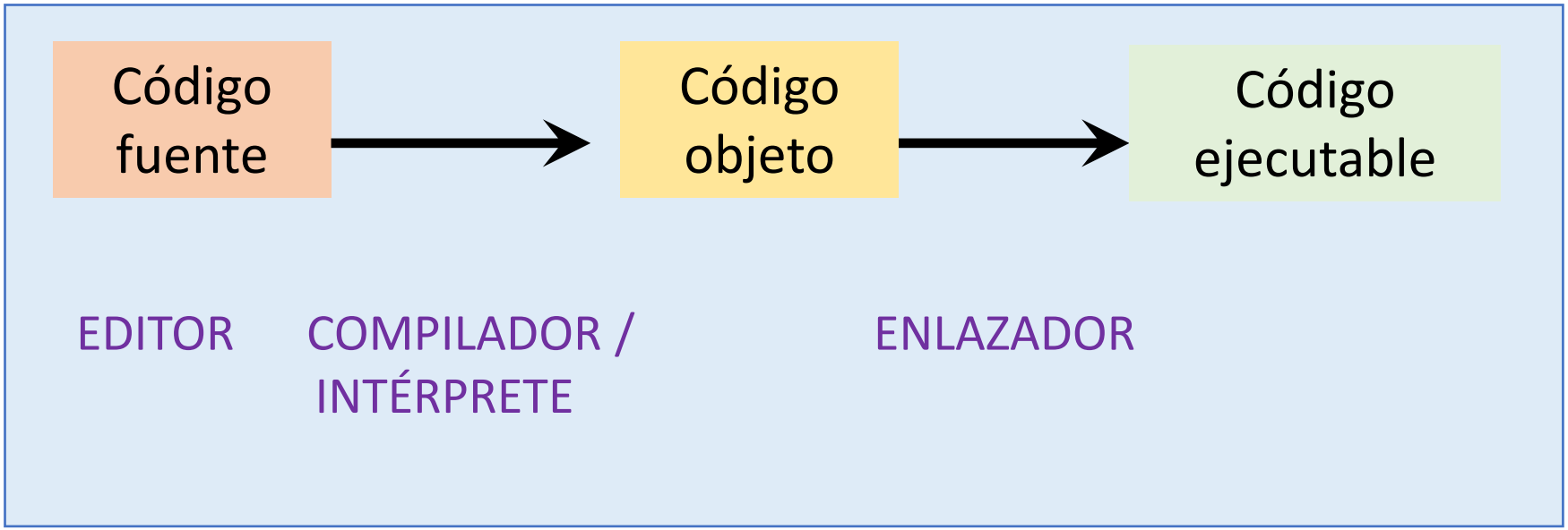


Figura 1.4. Diagrama de estructuras que muestra la arquitectura de una aplicación informática siguiendo la metodología estructurada.

1.3. Código fuente, código objeto y código ejecutable. Herramientas implicadas



Hoy en día, en lugar de emplear estas herramientas por separado, se emplean entornos de desarrollo integrados.

1.4. Máquinas virtuales

Máquina virtual de sistema.

Aplicaciones: VMware Workstation, Virtual Box.

Máquina virtual de proceso.

Máquina virtual de Java (JVM).

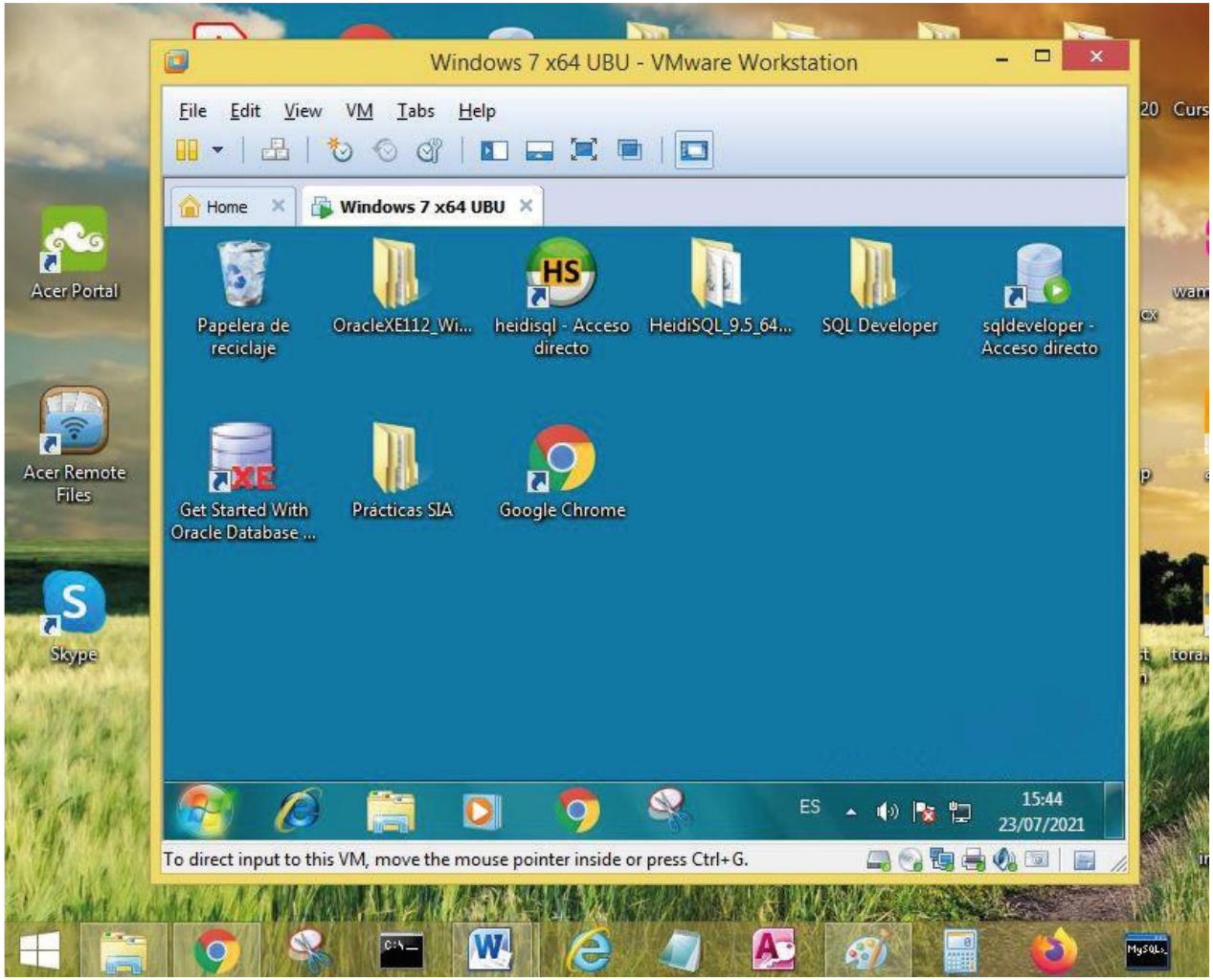


Figura 1.5. Uso del programa VMware Workstation para poder utilizar, dentro de un equipo, otro equipo que tiene incorporado su propio sistema operativo y disco duro, entre otras tecnologías.

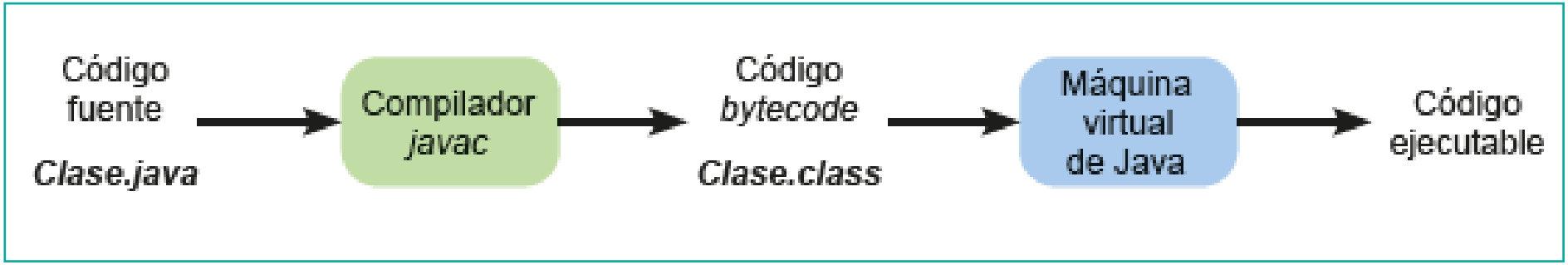


Figura 1.6. Proceso de obtención del código ejecutable de un programa escrito en Java: se parte del código fuente, que se plasma en un archivo con extensión .java por cada clase; con el compilador javac se transforma en un archivo con extensión .class escrito en código bytecode; finalmente, la JVM transforma este archivo en código ejecutable.

1.5. La ingeniería del software

Evolución:

- Trabajo artesanal.
- Década de 1970 → Crisis del software: problemas con:
  - Planificación y estimación de costes.
  - Productividad.
  - Calidad del software.
- Solución: dar un enfoque de ingeniería al desarrollo de software.
- Nacimiento de la ingeniería del software.

Figura 1.7. La ingeniería del software incluye una serie de tareas de desarrollo como la planificación, el análisis, diseño, programación, pruebas e implantación. Aquí también se muestran otros conceptos relacionados con el desarrollo de software, como el propio desarrollo y la validación y verificación de este.

1.6. Fases del desarrollo de una aplicación informática

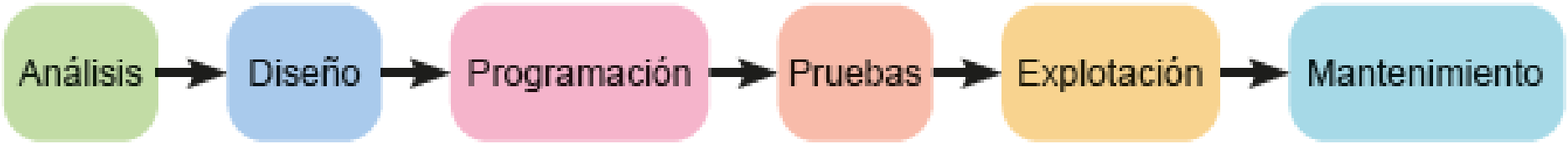


Figura 1.8. Fases del desarrollo de una aplicación informática.

Actividades sombrilla:

- ✓ Seguimiento y control del proyecto.
- ✓ Administración del riesgo.
- ✓ Aseguramiento de la calidad.
- ✓ Revisiones técnicas.
- ✓ Medición.
- ✓ Administración de la configuración.
- ✓ Administración de la reutilización.
- ✓ Preparación y producción del producto del trabajo.

1.6.1. Análisis

Objetivo

Comprensión de los requisitos funcionales y no funcionales de la aplicación (fiabilidad, escalabilidad, extensibilidad, seguridad, mantenibilidad).

Pasos

- Comunicación con el cliente. Técnicas:
  - ✓ Entrevistas.
  - ✓ Desarrollo conjunto de aplicaciones.
  - ✓ Desarrollo de un prototipo.
- Creación de modelos gráficos con apoyos textuales.

Resultado

Especificación de requisitos del software (ERS).

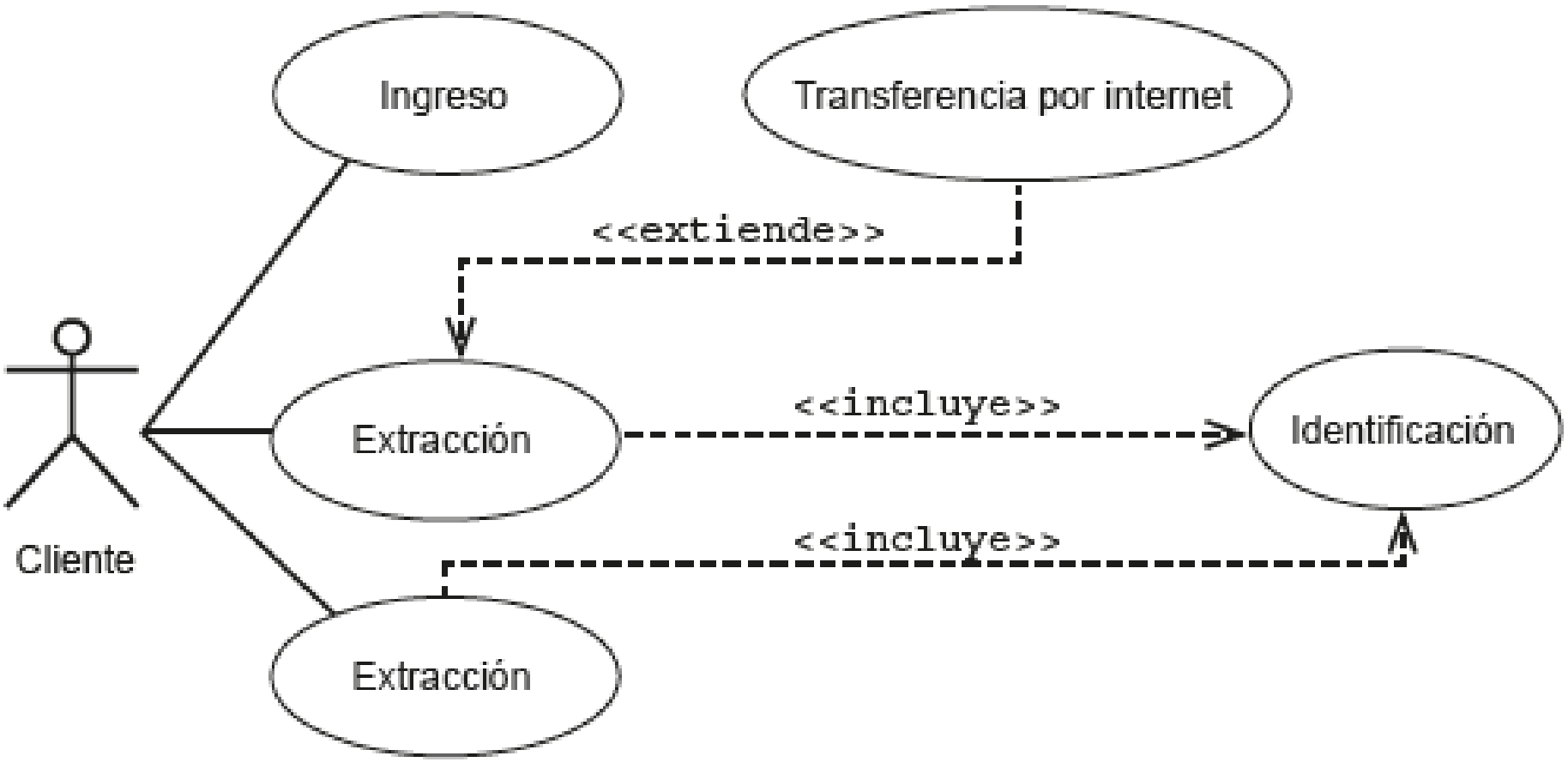


Figura 1.9. Este diagrama de casos de uso refleja los requisitos funcionales que debe satisfacer una aplicación para una entidad bancaria en la que los usuarios pueden realizar tres tipos de operaciones con su cuenta bancaria: ingresar dinero, extraer dinero o realizar una transferencia. Para los dos últimos tipos de operaciones, los clientes se tienen que identificar.

1.6.2. Diseño

Objetivo: se parte de la ERS y se obtienen modelos más detallados que indican los pasos que se han de dar para responder a los requisitos.

1.6.3. Programación

Objetivo: escribir el código fuente a partir del resultado de la etapa de diseño empleando un lenguaje de programación y siguiendo ciertas normas en cuanto a:

- Comentarios.
- Declaraciones de variables y parámetros de métodos.
- Nombres de clases, atributos, métodos, variables, constantes, etc.
- Líneas en blanco entre clases, métodos, etc.
- Sangrados.

1.6.4. Pruebas

Objetivo

- Detectar errores en el software antes de que se entregue al cliente:

  ✓ Verificación.
  ✓ Validación.

Dos tipos de técnicas de pruebas

- Pruebas de caja blanca.
- Pruebas de caja negra.

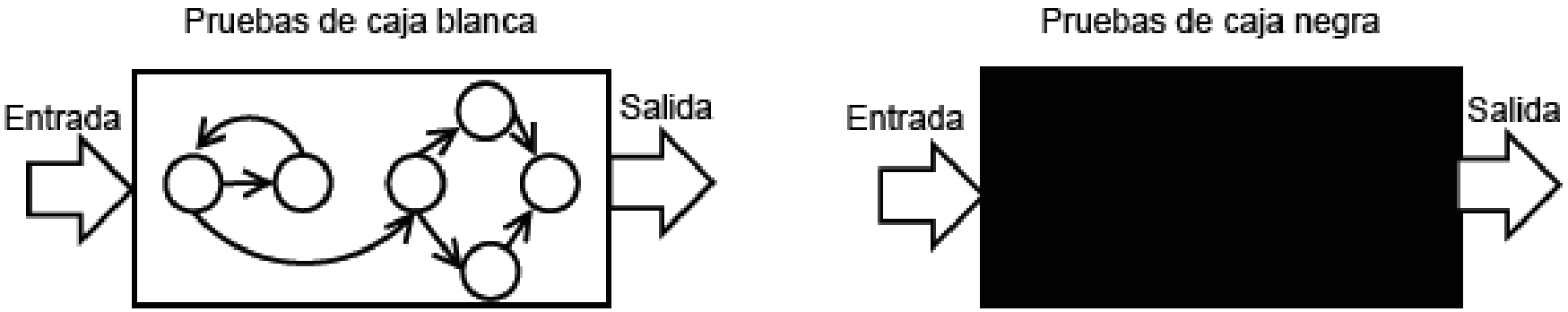


Figura 1.10. En las pruebas de caja blanca es necesario conocer el código fuente para realizar las pruebas, mientras que en las de caja negra, solo es necesario conocer las funciones que realiza el software para comprobar si la salida obtenida coincide con la esperada.

1.6.5. Explotación

Objetivo:

- Asegurarse de que el software está preparado para su uso por parte del cliente en sus instalaciones.

1.6.6. Mantenimiento

Tipos:

- Mantenimiento correctivo.
- Mantenimiento adaptativo.
- Mantenimiento perfectivo.

1.7. Roles que intervienen en el proceso de desarrollo de software

Jefe de proyecto	Expertos del dominio	Analista	Arquitecto
Diseñador	Programador	Probador	Encargado de implantación

Figura 1.11. El trabajo conjunto y cohesionado de todos los miembros del equipo de desarrollo bajo el mando de un líder (jefe de proyecto) es un aspecto fundamental para el éxito del proyecto.

1.8. Modelos de ciclo de vida del software

Orden en el que se deben llevar a cabo las tareas del proyecto y criterios para el paso de una tarea a la siguiente.

Clasificación:

- Modelo en cascada.
- Modelos de proceso incremental.
- Modelos de proceso evolutivo:
  - Construcción de prototipos.
  - Modelo en espiral.

1.8.1. Modelo en cascada

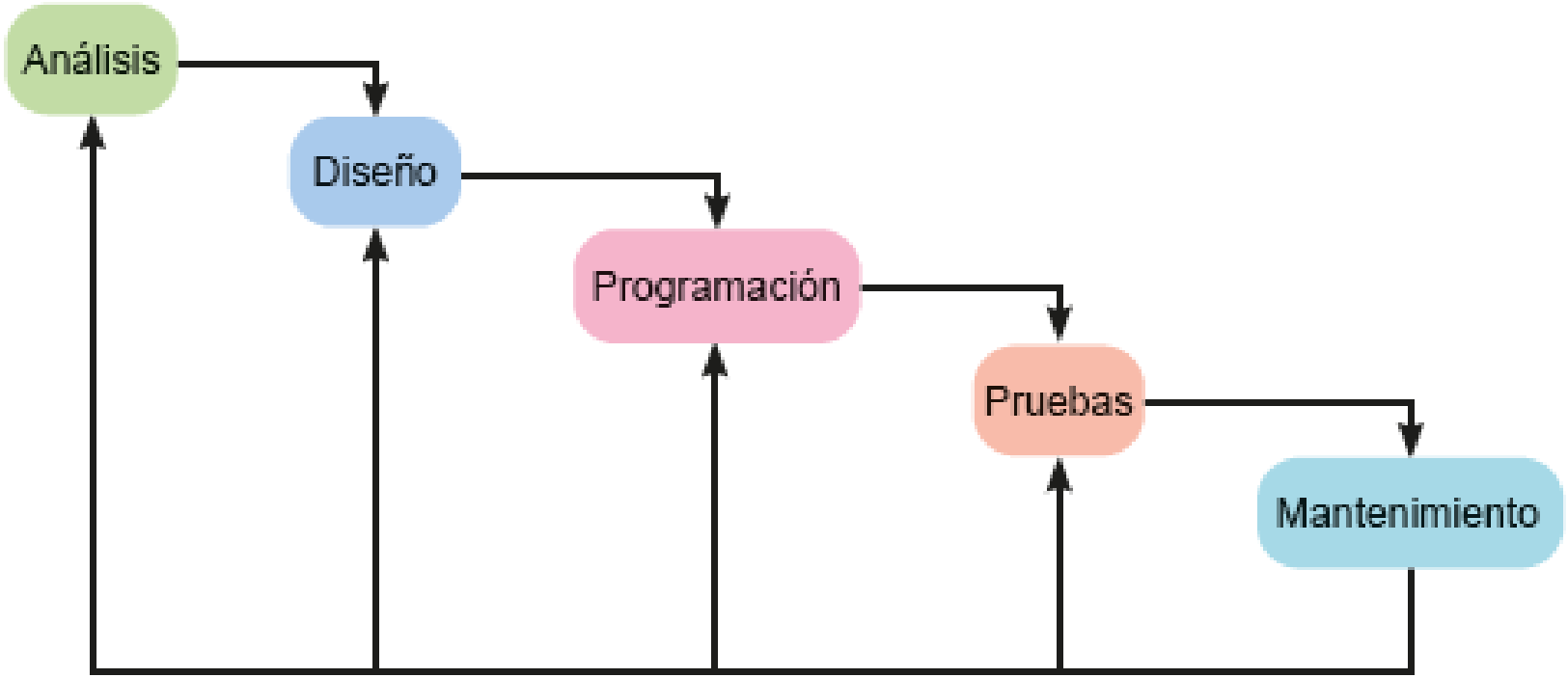


Figura 1.12. Representación del modelo en cascada o ciclo de vida clásico. Las cinco actividades que se muestran se deben ejecutar en secuencia, si bien se permite volver hacia atrás cuando se detecta la necesidad de realizar algún cambio.

1.8.2. Modelos de proceso incremental

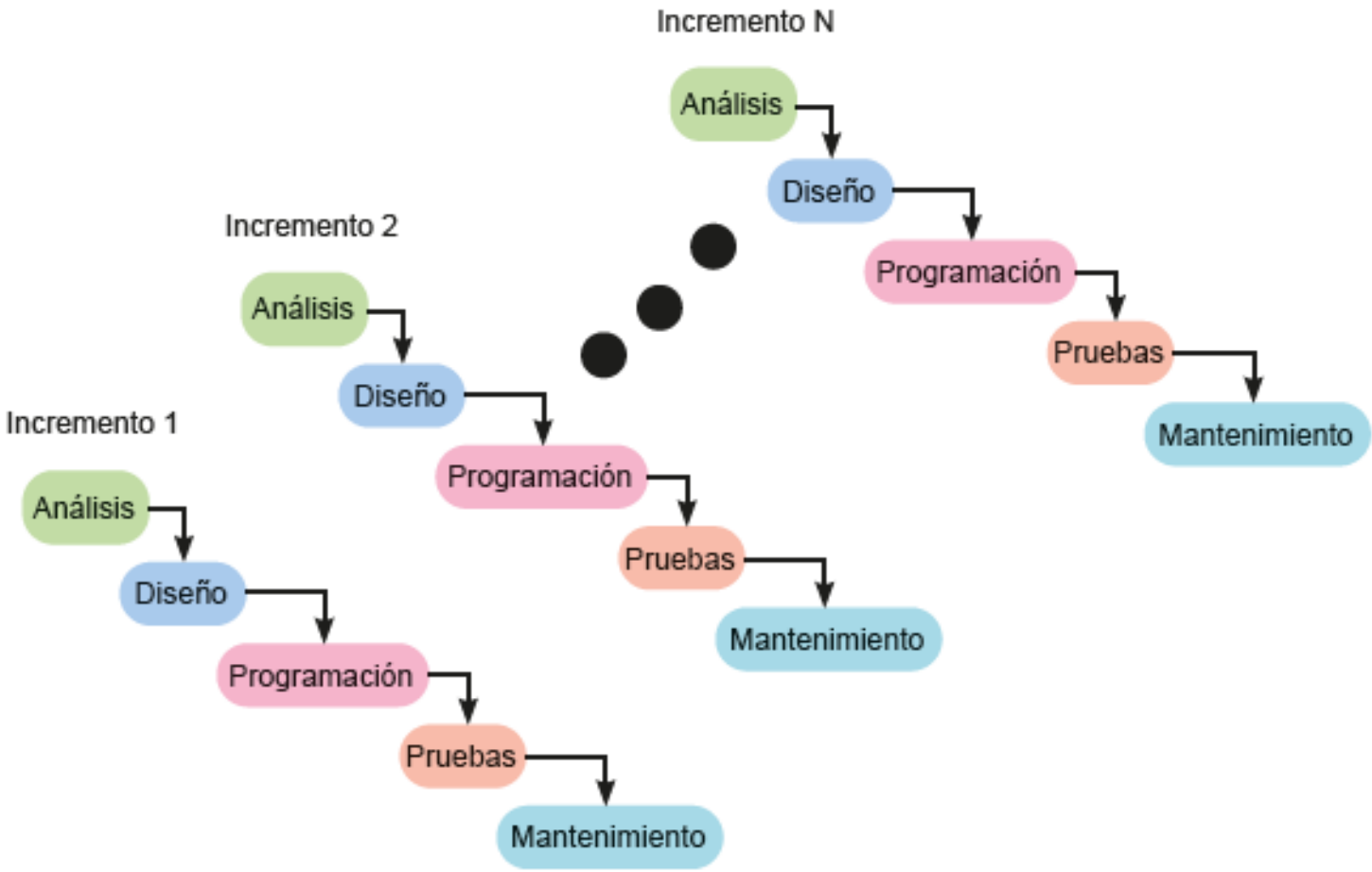


Figura 1.13. Representación del modelo incremental de desarrollo de software. El software se divide en una serie de incrementos, en cada uno de los cuales se aplican los pasos del ciclo de vida clásico.

1.8.3. Modelos de proceso evolutivo

- Se construye el producto en varias iteraciones.
- En cada iteración se desarrollan versiones cada vez más completas del software.

Construcción de prototipos

- Es útil cuando hay incertidumbre en relación con los requisitos.
- Se construye un prototipo que simula la interfaz hombre-máquina o que implementa parte de la funcionalidad requerida.

Figura 1.14. Actividades del modelo de construcción de prototipos.

Modelo en espiral

- Incorpora el análisis de riesgo en cada iteración para prever los problemas que se pueden presentar y tomar medidas.

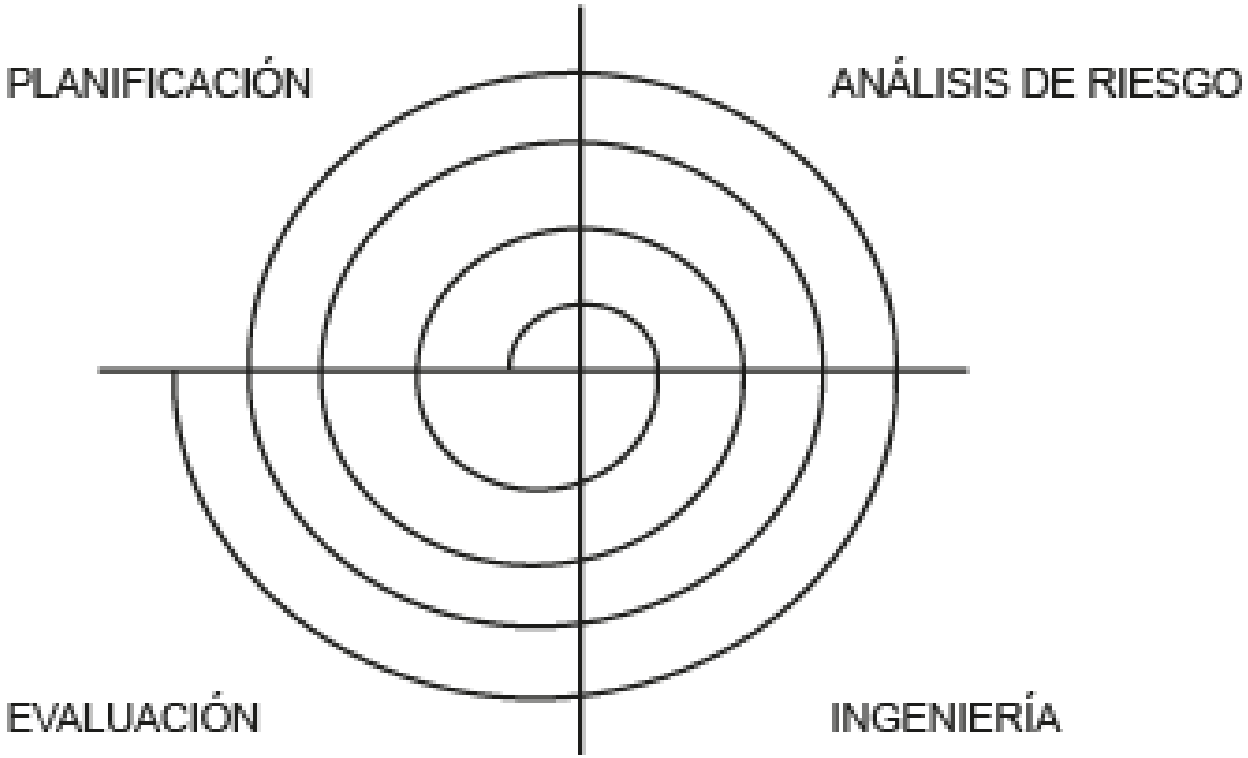
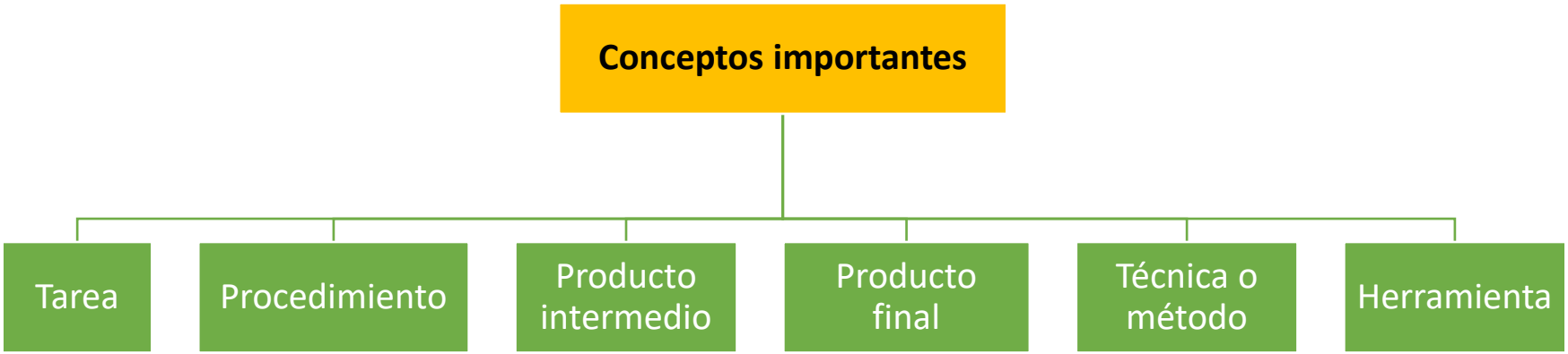


Figura 1.15. Representación del modelo en espiral, indicando las actividades que se llevan a cabo en cada cuadrante.

1.9. Metodologías de desarrollo de software

- Definición: conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores.
- Una metodología sigue uno o varios modelos de ciclo de vida.



Evolución cronológica de las metodologías

- Desarrollo convencional.
- Metodologías estructuradas.
- Metodologías orientadas a objetos: cambio de paradigma en relación con las metodologías estructuradas.

En la actualidad

- Metodología de referencia: proceso unificado de desarrollo de software (RUP).
- Metodologías ágiles.

1.9.1. El proceso unificado de Rational (RUP)

Dimensión temporal: un ciclo se compone de cuatro fases

- Comienzo.
- Elaboración.
- Construcción.
- Transición.

Dimensión estática

- Perfil.
- Actividad.
- Producto intermedio.
- Flujo de trabajo.

Flujos de trabajo

Flujos de ingeniería

- Modelado del negocio.
- Requisitos.
- Análisis y diseño.
- Implementación.
- Pruebas.
- Implantación.

Flujos de apoyo

- Gestión de la configuración.
- Gestión de proyecto.
- Entorno.

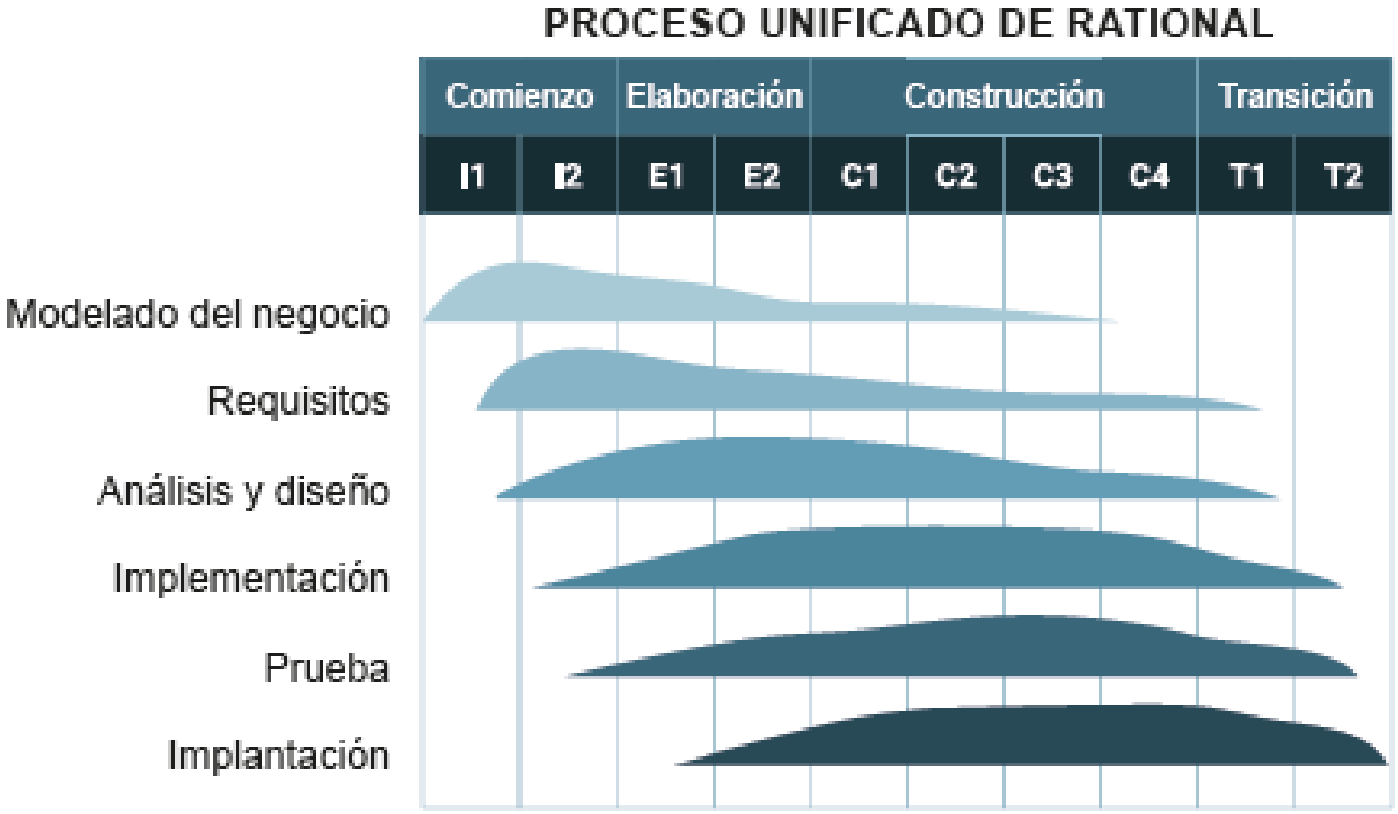


Figura 1.16. Ejemplo de evolución de un proyecto de desarrollo empleando la metodología RUP. Se puede observar cómo en las diferentes fases (comienzo, elaboración, construcción y transición) se van realizando tareas correspondientes a los distintos flujos de ingeniería (modelado del negocio, requisitos, análisis y diseño, implementación, pruebas e implantación), así como el volumen de trabajo que supone cada uno de estos flujos de ingeniería en cada una de las iteraciones.

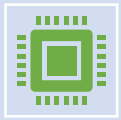
1.9.2. Modelos de desarrollo ágil

Manifiesto por el desarrollo ágil de software de la Alianza Ágil: https://agilemanifesto.org

Características:

- Entrega rápida al cliente de software incremental.
- Equipos pequeños y bien motivados.
- Mayor facilidad para incorporar cambios y menor coste de los cambios.

12 principios de agilidad: https://agilemanifesto.org/principles.html



Modelos de desarrollo ágil:

- Programación extrema (XP).
- Scrum.

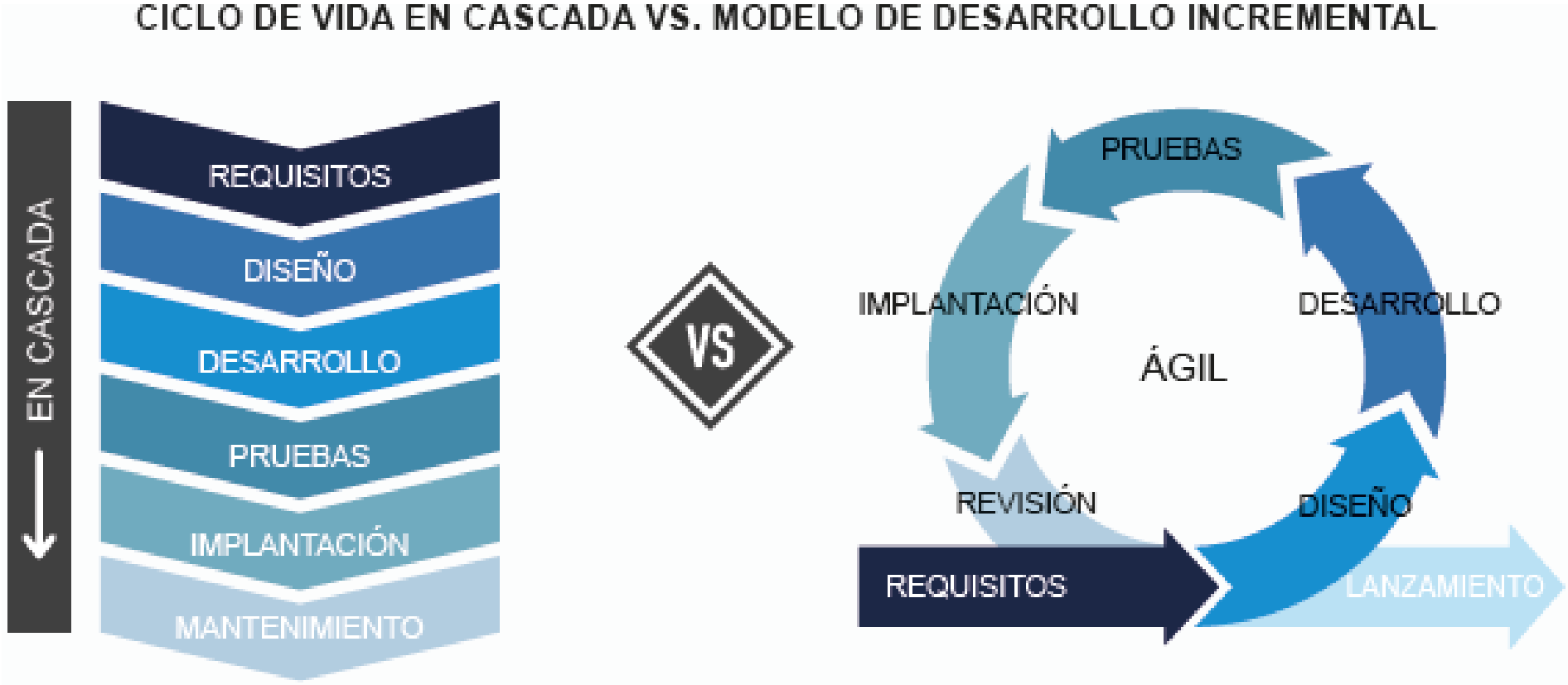


Figura 1.17. La agilidad, propia del modelo de desarrollo incremental, frente al modelo en cascada. En este modelo, se llevan a cabo las mismas tareas que en el modelo en cascada, pero se aplican sobre entregas pequeñas o incrementos de software.

Figura 1.18. Cada uno de los incrementos de software se obtiene mediante una iteración, que en los modelos ágiles suele recibir el nombre de sprint. En este caso, se desarrolla el software en tres *sprints*.

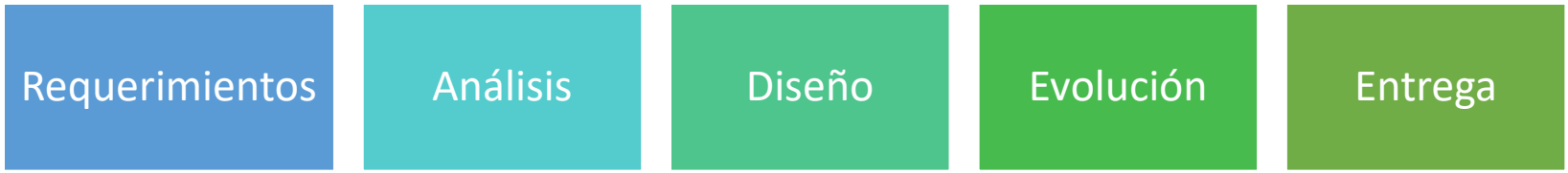
Programación extrema (XP)

Es el método de desarrollo ágil más destacado.

Valores	Actividades estructurales
• Simplicidad. • Comunicación. • Retroalimentación. • Valentía. • Respeto.	• Planificación (historias de usuario). • Diseño (sencillo). • Codificación (programación por parejas). • Pruebas.

Scrum

- **Objetivo:** entrega de valor (productos) al cliente en cortos periodos de tiempo.
- **Principios:**
  - Transparencia.
  - Inspección.
  - Adaptación.
- **Actividades estructurales:**



- **Sprint:** periodo de tiempo utilizado para realizar el trabajo requerido para entregar valor al cliente.

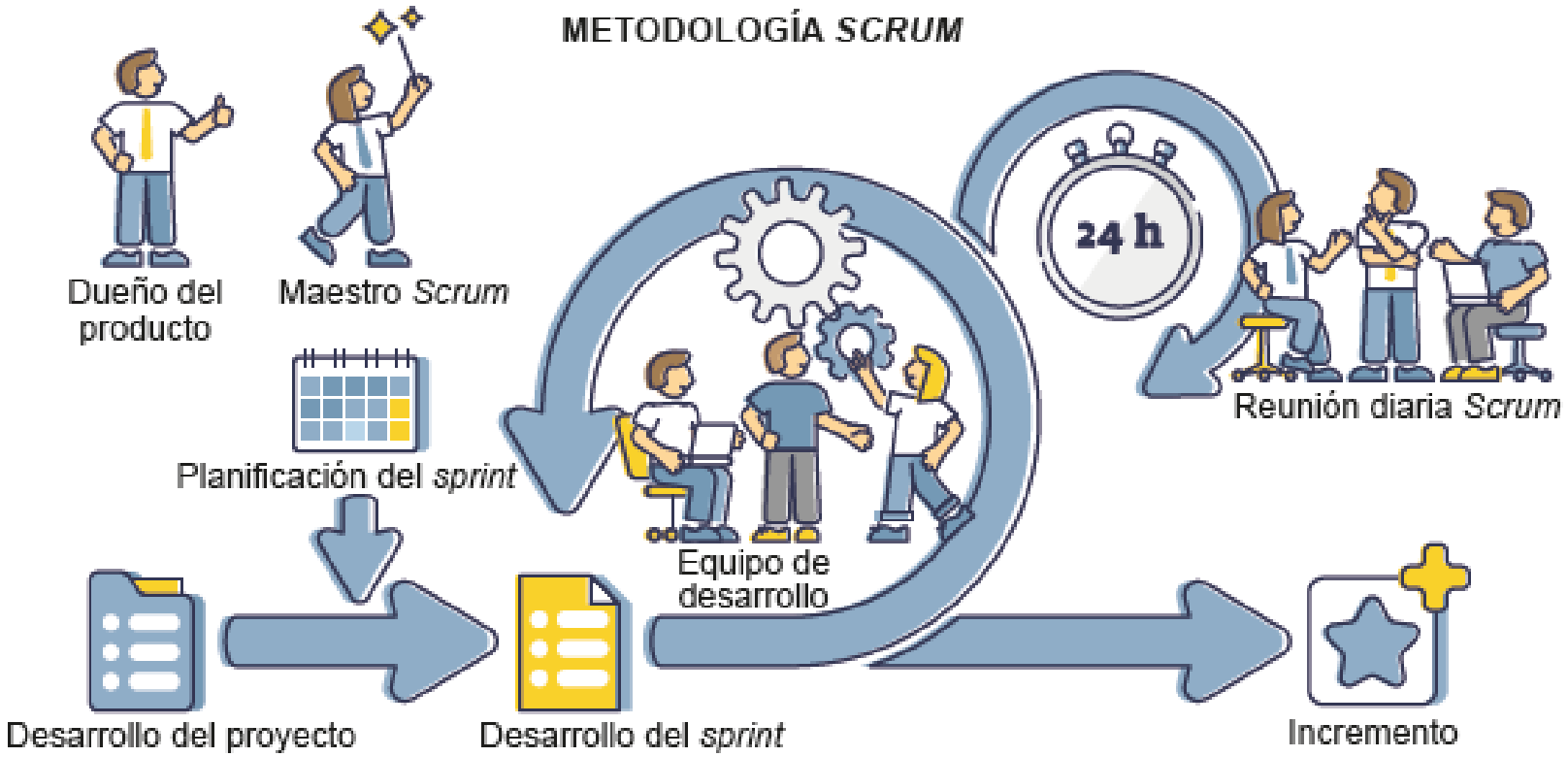


Figura 1.19. En la metodología *Scrum* todo el trabajo que hay que realizar para el desarrollo del proyecto (*product backlog*) se divide en una serie de *sprints*. A partir de la planificación del *sprint*, se crea una lista de tareas (*sprint backlog*) que irá realizando el equipo asignado (*Scrum team*). Cada día de trabajo, se lleva a cabo una reunión para analizar la marcha del sprint (*Scrum meeting*) y, al finalizar el *sprint*, se realiza una presentación al cliente (*sprint review*) del producto obtenido hasta el momento (incremento, en inglés '*product increment*').