
ANALYZE
DEVELOPMENT
DESIGN
IMPLEMENT

1. Introducción al desarrollo de software

Caso práctico En GoyoProg todos han vuelto ya de sus vacaciones. Les espera un septiembre agitado, pues acaban de recibir una petición por parte de un banco para desarrollar un proyecto software en banca online. Ada, la supervisora de proyectos de GoyoProg, se reúne con Juan y María (trabajadores de la empresa) para empezar a planificar el proyecto. Ana, cuya especialidad es el diseño gráfico de páginas web, acaba de terminar el Ciclo de Grado Medio en Sistemas Microinformáticos y Redes y realizó la FCT en GoyoProg. Trabaja en la empresa ayudando en los diseños, y aunque está contenta con su trabajo, le gustaría participar activamente en todas las fases en el proyecto. El problema es que carece de los conocimientos necesarios. Antonio comienza a estudiar el Ciclo de Grado Superior de Desarrollo de Aplicaciones Multiplataforma, y está dispuesto a hacerlo. (No tendría que dejar el trabajo). Después de todo… ¿qué pueden perder?

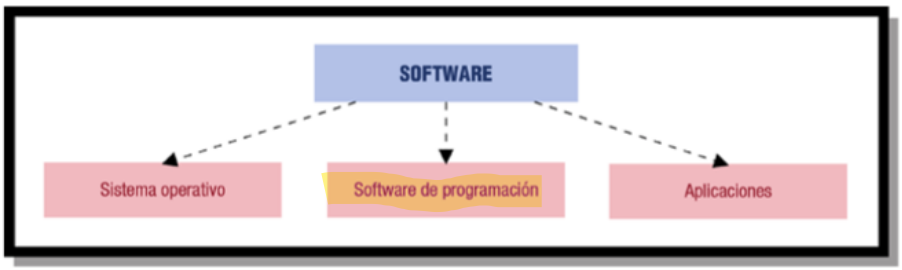
Todos en la empresa están entusiasmados con el proyecto que tienen entre manos. Saben que lo más importante es planificarlo todo de antemano y elegir el tipo de software más adecuado. Ana les escucha hablar y no llega a entender por qué hablan de "tipos de software". ¿Acaso el software no era la parte lógica del ordenador, sin más? ¿Cuáles son los tipos de software?

Después de saber ya diferenciar los distintos tipos de software, Ana se le plantea otra cuestión: El software, sea del tipo que sea, se ejecuta sobre los dispositivos físicos del ordenador. ¿Qué relación hay entre ellos?

En GoyoProg ya están manos a la obra. Ada reúne a toda su plantilla para desarrollar el nuevo proyecto. Ella sabe mejor que nadie que no será sencillo y que hay que pasar por una serie de etapas. Ana no quiere perderse la reunión y quiere descubrir por qué hay que tomar anotaciones y tantas molestias antes incluso de empezar.

1.1. Software y programa. Tipos de software

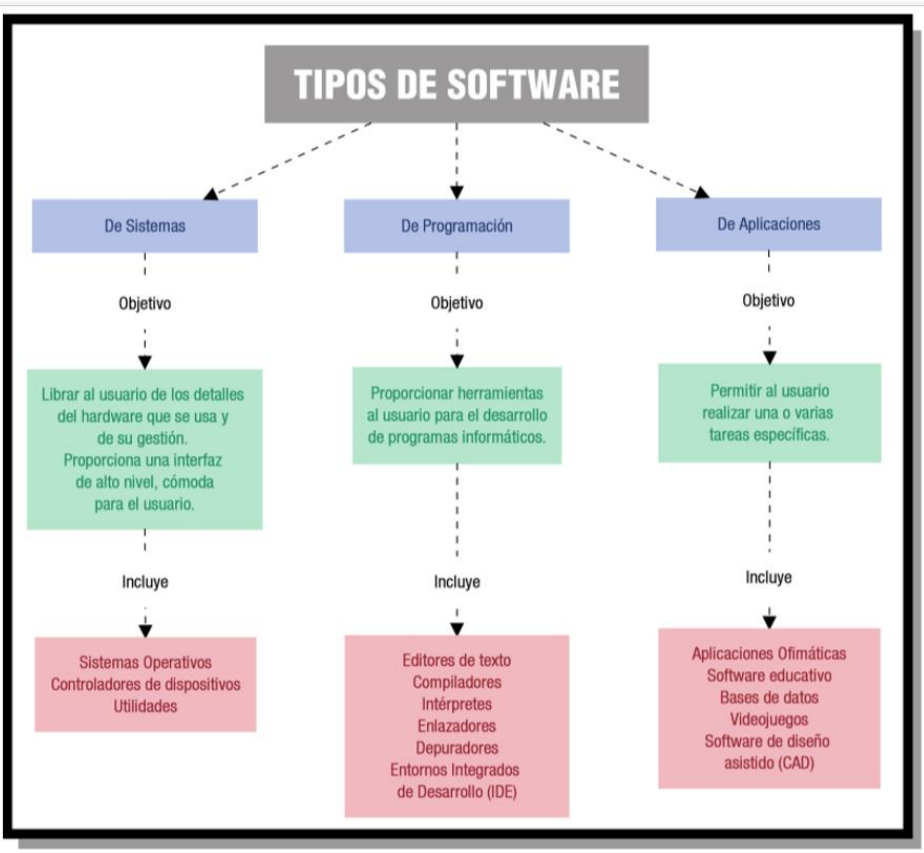
Es de sobra conocido que el ordenador se compone de dos partes bien diferenciadas: **hardware y software**. El **software** es el conjunto de programas informáticos que actúan sobre el hardware para ejecutar lo que el usuario desee.
Según su función **se distinguen tres tipos de software**: sistema operativo, software de programación y aplicaciones.



El **sistema operativo** es el software base que ha de estar instalado y configurado en nuestro ordenador para que las aplicaciones puedan ejecutarse y funcionar. Son ejemplos de sistemas operativos: Windows, Linux, Mac OS X , etc.

El **software de programación** es el conjunto de herramientas que nos permiten desarrollar programas informáticos, y **las aplicaciones informáticas** son un conjunto de programas que tienen una finalidad más o menos concreta. Son ejemplos de aplicaciones: un procesador de textos, una hoja de cálculo, el software para reproducir música, un videojuego, etc.

A su vez, un **programa** es un conjunto de instrucciones escritas en un lenguaje de programación. En definitiva, distinguimos los siguientes tipos de software:



En este tema, nuestro interés se centra en las **aplicaciones informáticas**: **cómo se desarrollan y cuáles son las fases por las que necesariamente han de pasar.** Necesitamos aprender los conceptos fundamentales de software y las fases del llamado **ciclo de vida de una aplicación informática**. También necesitamos **distinguir los diferentes lenguajes de programación y los procesos** que ocurren hasta que el programa funciona y realiza la acción deseada.

Como sabemos, **al conjunto de dispositivos físicos que conforman un ordenador** se le denomina **hardware.** Existe una relación indisoluble entre éste y el software, ya que necesitan estar instalados y configurados correctamente para que el equipo funcione. El software se ejecutará sobre los dispositivos físicos.

Esta relación software-hardware la podemos poner de manifiesto desde dos puntos de vista:

- ✓ **Desde el punto de vista del sistema operativo:** El sistema operativo es el encargado de **coordinar al hardware durante el funcionamiento del ordenador, actuando como intermediario entre éste y las aplicaciones que están corriendo en un momento dado.** Todas las aplicaciones necesitan recursos hardware durante su ejecución (tiempo de CPU, espacio en memoria RAM, tratamiento de interrupciones, gestión de los dispositivos de Entrada/Salida, etc.). Será siempre el sistema operativo el encargado de controlar todos estos aspectos de manera "oculta" para las aplicaciones (y para el usuario).
- ✓ **Desde el punto de vista de las aplicaciones:** Ya hemos dicho que una **aplicación es un conjunto de programas**, y que éstos están escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar.

Hay multitud de lenguajes de programación diferentes (como ya veremos en su momento). Sin embargo, todos tienen algo en común: estar escritos con sentencias de un idioma que el ser humano puede aprender y usar fácilmente. Por otra parte, el hardware de un ordenador sólo es capaz de interpretar señales eléctricas (ausencias o presencias de tensión) que, en informática, se traducen en secuencias de 0 y 1 (código binario).

Esto nos hace plantearnos una cuestión: **¿Cómo será capaz el ordenador de "entender" algo escrito en un lenguaje que no es el suyo?**

Como veremos a lo largo de este módulo, tendrá que pasar algo (un **proceso de traducción de código**) para que el ordenador ejecute las instrucciones escritas en un lenguaje de programación.

Para fabricar un programa informático ¿que se ejecuta en una computadora?

- ✓ Hay que escribir las instrucciones en código binario para que las entienda el hardware.
- ✓ Sólo es necesario escribir el programa en algún lenguaje de programación y se ejecuta directamente.
- ✓ Hay que escribir el programa en algún Lenguaje de Programación y contar con herramientas software que lo traduzcan a código binario.
- ✓ Los programas informáticos no se pueden escribir: forman parte de los sistemas operativos.

1.2 Desarrollo de software

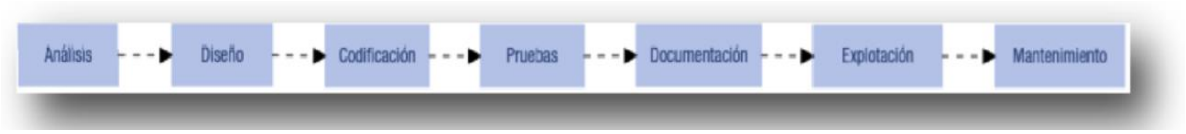
Entendemos por **Desarrollo de Software** desde que se concibe una idea hasta que un programa está implementado en el ordenador y funciona.

La Ingeniería del Software permite la aplicación de **principios de ingeniería al diseño de las aplicaciones**, de modo que su calidad y demás propiedades exigidas sean similares a las de cualquier producto de ingeniería.

Hay tres elementos sobre los que se asienta toda ingeniería; método, herramientas y procedimientos.

- ✓ **Los métodos:** hacen referencia a **como se ha de diseñar** el software.
- ✓ **Las herramientas:** se refieren a los **elementos informatizados** que permiten la automatización de las tareas a realizar en cada una de las fases del diseño del software.
- ✓ **Los procedimientos:** se refieren al **protocolo definido durante el diseño del software**, es decir las reglas, normas, especificaciones, los plazos, controles de calidad… etc.

El proceso de desarrollo que en principio puede parecer una tarea simple, consta de una serie de pasos de obligado cumplimiento, pues solo así podemos **garantizar que los programas creados son eficientes, fiables, seguros y responden a las necesidades de los usuarios finales** (aquellos que van a utilizar el programa)



El desarrollo de software es un proceso que conlleva una serie de pasos o etapas. Estas etapas son:

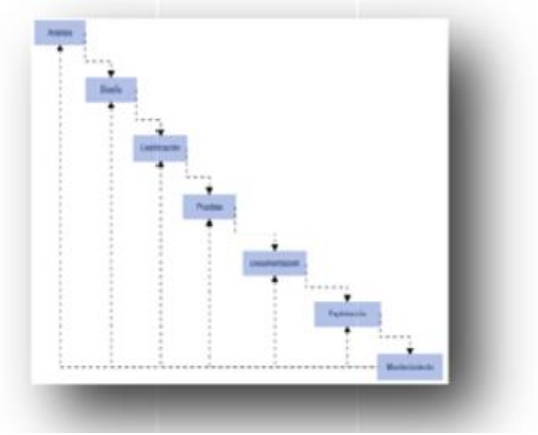
- ✓ **ANÁLISIS DE REQUISITOS:** Se especifican los requisitos funcionales y no funcionales del sistema.
- ✓ **DISEÑO:** Se divide el sistema en partes y se determina la función de cada una.
- ✓ **CODIFICACIÓN:** Se elige un Lenguajes de Programación y se codifican los programas.
- ✓ **PRUEBAS:** Se prueban los programas para detectar errores y se depuran.
- ✓ **DOCUMENTACIÓN:** De todas las etapas, se documenta y guarda toda la información.
- ✓ **EXPLOTACIÓN:** Instalamos, configuramos y probamos la aplicación en los equipos del cliente.
- ✓ **MANTENIMIENTO:** Se mantiene el contacto con el cliente para actualizar y modificar la aplicación el futuro.

Según el orden y la forma en que se lleven a cabo las etapas hablaremos de diferentes **ciclos de vida del software.**

La construcción del software es un proceso que puede llegar a ser muy complejo y que exige gran coordinación y disciplina del grupo de trabajo que lo desarrolla.

1.2.1 Ciclo de vida del software: Metodologías tradicionales vs Metodologías ágiles

La serie de pasos a seguir para desarrollar un programa es lo que se conoce como **Ciclo de Vida del Software**.



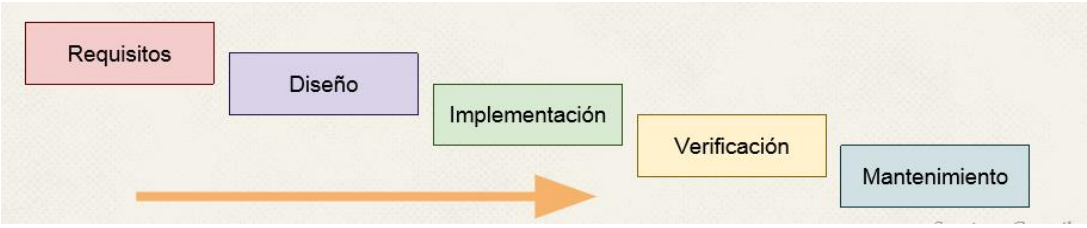
Una metodología de desarrollo es una forma de gestionar un proyecto de software. Esto incluye aspectos como seleccionar las características que se van a incluir en una nueva versión, el momento en el que el software se entrega, quién trabaja en qué parte y el testeo que se realiza. Según el tipo de proyecto, será más beneficiosa una metodología u otra, y también dependerá de la cultura del equipo de desarrollo.

1.2.1.1 Metodologías tradicionales

Las **metodologías tradicionales** centran su atención en llevar una **documentación exhaustiva de todo el proyecto y en cumplir con un plan de proyecto**, definido en su fase inicial del desarrollo. Se focalizan en documentación, planificación y procesos. (Plantillas, técnicas de administración, revisiones, etc).

Existen diferentes ciclos de vida para las **diversas metodologías** según distintos autores, pero los más conocidos son los siguientes:

- ✓ **Metodología en cascada (waterfall).** Es una de las metodologías más tradicionales. Tiene un flujo lineal que hace gran énfasis en **recoger los requisitos y realizar la arquitectura antes de comenzar la programación y las pruebas**. **Se centra mucho en la documentación y calidad del código.**
  - **Ventajas:** una ventaja es que el proyecto está bien planificado, lo que minimiza costes a medio plazo por tener los requisitos bien fijados, y que el proyecto tiende a quedar muy bien documentado.
  - **Desventajas:** no se adapta bien a cambios de requisitos no esperados, y no puede verse ni siquiera una parte del resultado final hasta muy avanzado el proyecto.



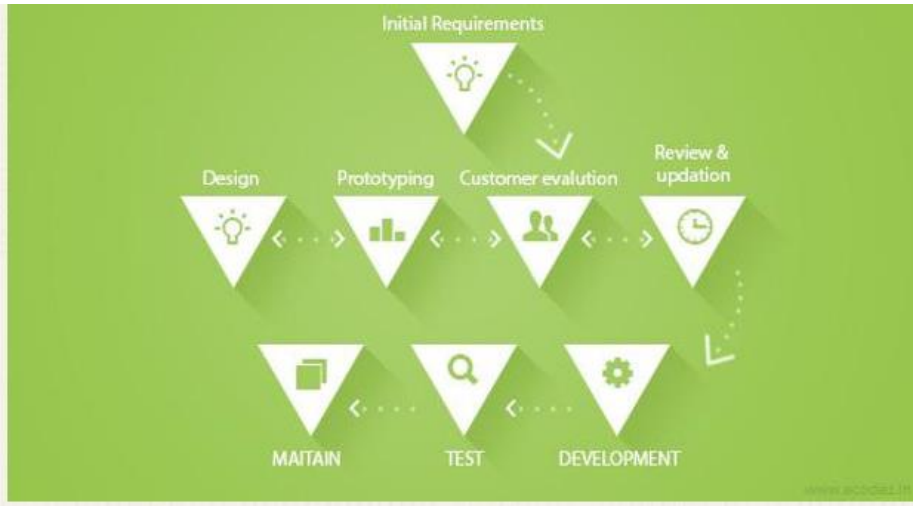
Situaciones adecuadas:

- Proyecto con objetivos y soluciones muy claros
- No hay presión por una implementación inmediata
- Los requisitos son muy estables
- En algunos proyectos largos y complejos puede acortar costes a medio plazo

Situaciones poco apropiadas:

- Proyectos susceptibles de frecuentes cambios inesperados
- El cliente (inexperto) quiere comprender el progreso del proyecto
- El proyecto es evolutivo y de continuo desarrollo

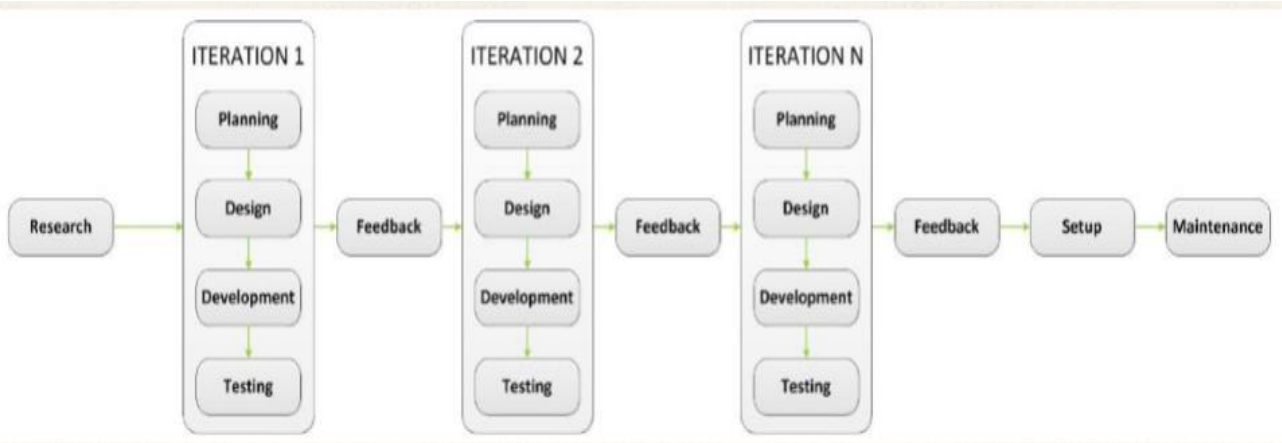
✓ **Prototipado.** Se centra en el gran problema de que **el cliente pueda comprender mejor el sistema en base a prototipos aproximados de manera que los requisitos queden más claros y definidos.** Parte de una investigación inicial del proyecto (requisitos) para pasar a realizar iteraciones sobre la elaboración de un prototipo de todo o parte/s del sistema, que se discute con el cliente para refinarlo. Tras estos ciclos, se procede a construir el sistema final con los requisitos aclarados.



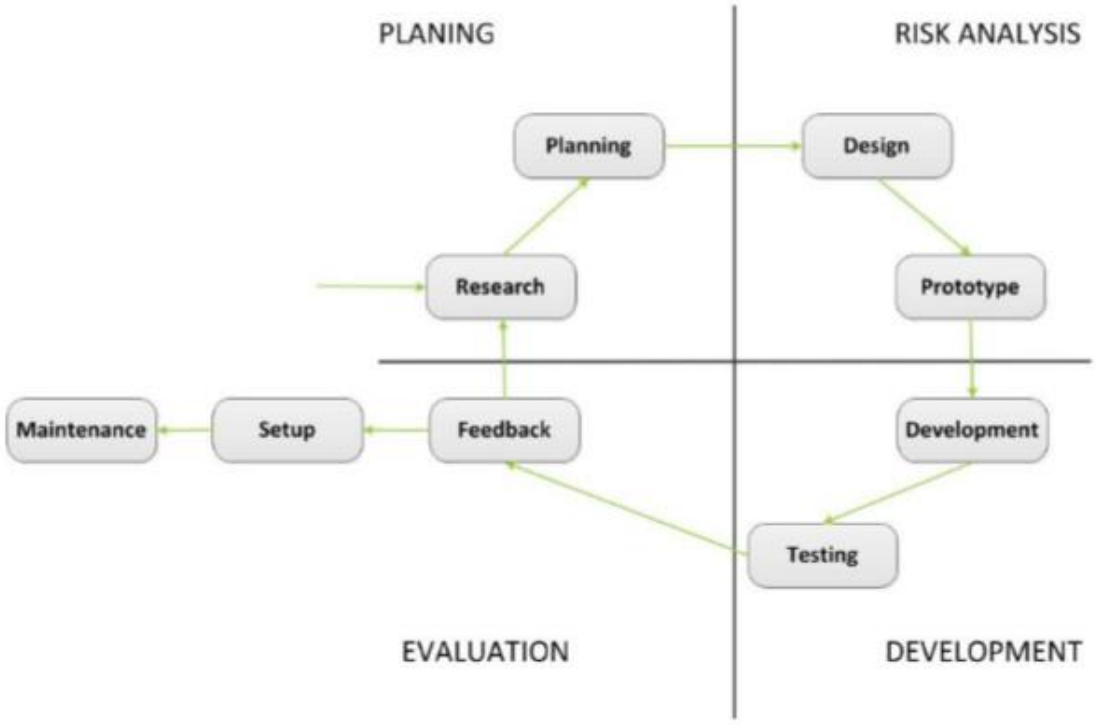
- **Ventajas:** Mejora la participación del cliente y la comunicación del proyecto. Ayuda a aclarar requisitos confusos. Promueve la innovación. Situaciones adecuadas:
  - ○ Proyectos complejos con muchos actores y requisitos.
  - ○ Objetivos del proyecto poco claros
  - ○ Presión por entregas inmediatas
  - ○ La composición del equipo de trabajo es estable (contratos/despidos)

- **Desventajas:** Se pueden dar en los prototipos soluciones incompletas o inadecuadas para el producto final. Los prototipos pueden dar lugar a arquitecturas poco pensadas e inflexibles (o exige desarrolladores expertos). Situaciones poco adecuadas:
  - ○ La escalabilidad del diseño del sistema es fundamental
  - ○ Equipo de trabajo cambiante
  - ○ Los objetivos del proyecto están muy claros / bajo riesgo

- ✓ **Iterativa e Incremental.** Las metodologías Iterativa e Incremental existen de forma separada y también de forma conjunta, como vemos en la imagen. Todos **los requisitos del proyecto se dividen, y se realiza una serie de desarrollos mini-Waterfall que llevan a cabo parte de los requisitos**. Estos mini-Waterfall se encadenan uno tras otro hasta que se han completado todos los requisitos.
  - **Ventajas:** El proyecto evoluciona por etapas y se reducen algunos de los problemas de rigidez del modelo Waterfall.
  - **Desventajas:** pérdida de visión global; arrastrar problemas complejos hacia el final.



- ✓ **Espiral.** Está **centrado en la evaluación de riesgos**, abordando los más grandes primero. Se organiza por ciclos, cada uno de los cuales se siguen los mismos pasos sobre una parte diferente del sistema:
  1) Determinación de objetivos y restricciones
  2) Evaluar alternativas y riesgos
  3) Desarrollar y verificar entregables
  4) Planear la siguiente iteración

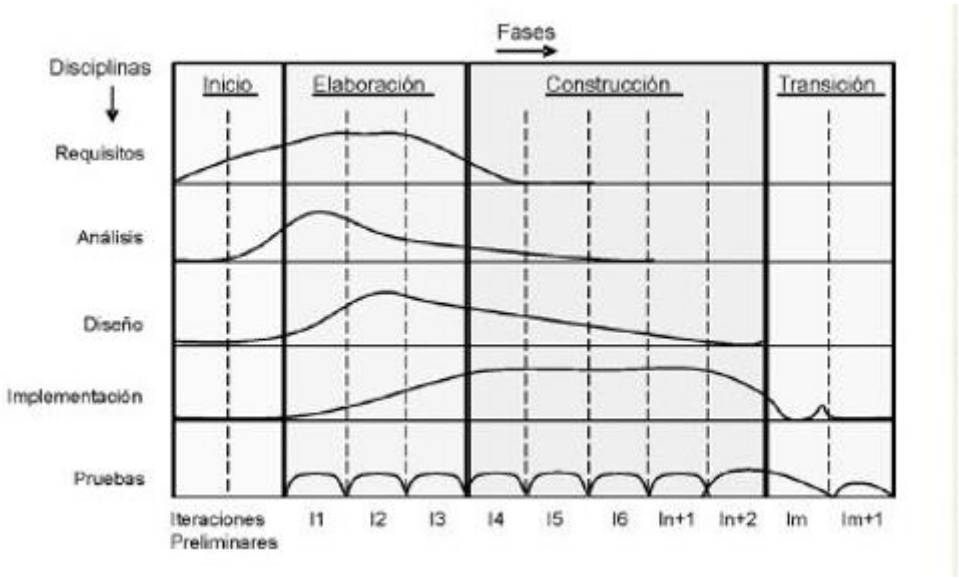


- **Ventajas:** Promueve reducir riesgos. Permite elegir una metodología diferente (Waterfall, Prototipado, Iterativa…) en cada iteración que se adecúe a sus riesgos. Situaciones adecuadas:
    - Los riesgos son importantes en el proyecto
    - Se requiere un fuerte control en la aprobación de requisitos
    - El proyecto se puede beneficiar de diferentes metodologías
    - El software es de uso interno y se conocen bien los riesgos

- **Desventajas:** Se requiere un manager experimentado para evaluar riesgos dada la complejidad de su análisis. No hay deadlines claros, por lo que el proyecto se puede prolongar en exceso. Riesgo de acabar siendo Waterfall. Situaciones poco adecuadas: contrarias a las anteriores.

✓ **Proceso unificado.** Esta metodología como introducción a UML, ya que están íntimamente relacionados:

- Se centra en casos de uso como herramienta para definir requisitos funcionales
- Es iterativa e incremental, definiendo 4 ciclos: Inicio, Elaboración, Construcción y Transición.
- Se centra en la definición de una arquitectura sólida y estable
- Sigue un análisis de riesgos



Si queremos construir una aplicación pequeña, y se prevé que no sufrirá grandes cambios durante su vida, ¿sería el modelo de ciclo de vida en espiral el más recomendable?

✓ Sí.
✓ No.

1.2.1.2 Metodologías ágile

No existe una metodología llamada Ágil, ni establece una forma de trabajar concreta. Ágil es un conjunto de valores y principios definidos en el **Manifiesto por el Desarrollo Ágil de Software**. Se caracterizan por tener las siguientes cualidades:

- Desarrollo evolutivo y flexible
- Autonomía de los equipos
- Planificación
- Comunicación

Por ejemplo: Si me encuentro con que necesito almacenar y recuperar unos números de una base de datos: ○ preveo que pueda utilizar esa funcionalidad en el futuro y construyo una capa de persistencia con una arquitectura sólida. ○ me las apaño para tener esos números funcionando por el momento. (visión ágil)

Metodología agile, en Informática, se usa para describir un método alternativo de gestión de proyectos. Es un proceso que **ayuda a los equipos a proporcionar respuestas rápidas a los cambios que se reciben sobre su proyecto**. ¿Esto qué significa? Pues que crea oportunidades para evaluar la dirección de un proyecto durante el ciclo de desarrollo del mismo. Los equipos evalúan el proyecto en reuniones regulares llamadas **sprints o iteraciones**. Los 11 principios del manifiesto ágil son:

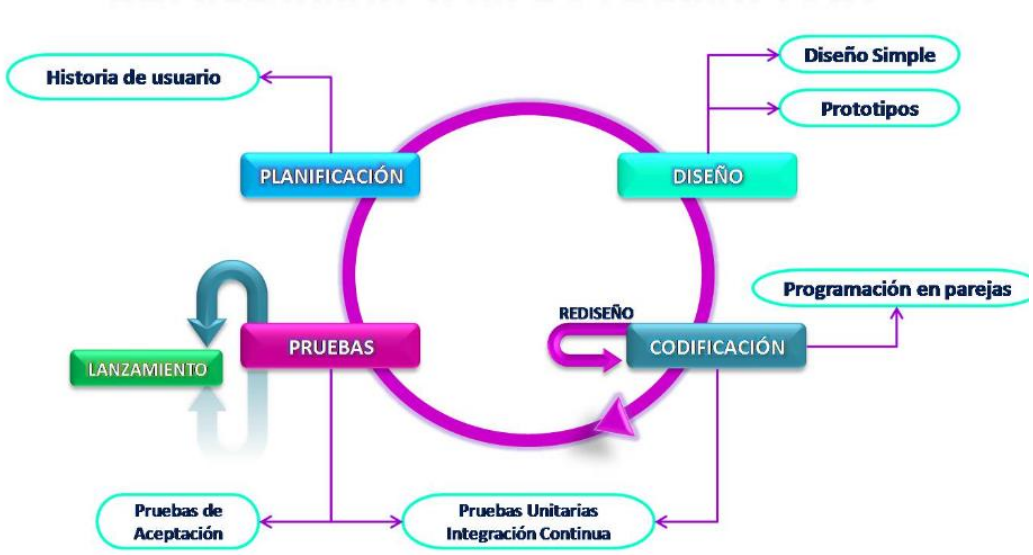
1. Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua del software de valor.
2. Son bienvenidos los requisitos cambiantes, aun llegando tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
3. Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses.
4. Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.
5. Construcción de proyectos en torno a individuos motivados.
6. La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
7. El software/producto/servicio que funcione es la principal medida de progreso.
8. Los procesos ágiles promueven el desarrollo sostenido. Los desarrolladores, patrocinadores, y usuarios han de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica ensalza la agilidad.
10. La simplicidad como arte de maximizar la cantidad de trabajo que se hace, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos que se auto organizan.

Algunos tipos de metodología ágil son:

- ✓ **Programación Extrema (XP).** Metodología ágil **centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo del software**, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen clima de trabajo. XP se basa en retroalimentación continua entre cliente y el equipo de desarrollo. XP es especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes.

Se caracteriza por:

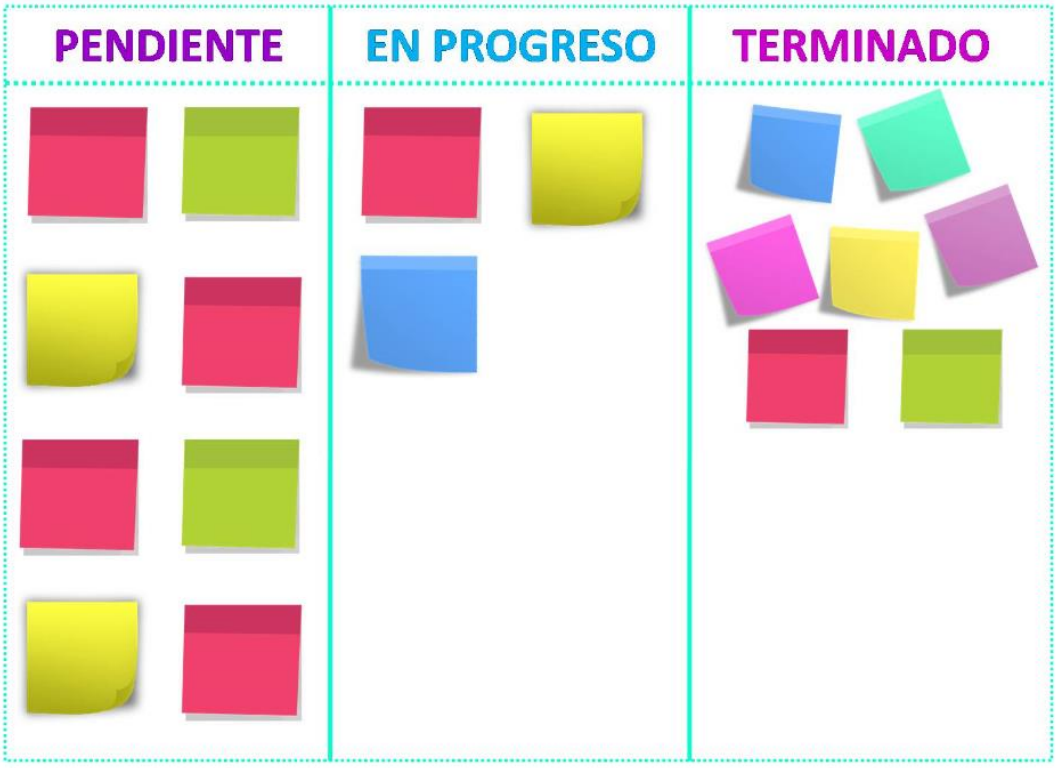
- Desarrollo iterativo e incremental
- Programación en parejas
- Pruebas unitarias continuas
- Corrección periódica de errores
- Integración del equipo de programación con el cliente
- Simplicidad, propiedad del código compartida y refactorización del código



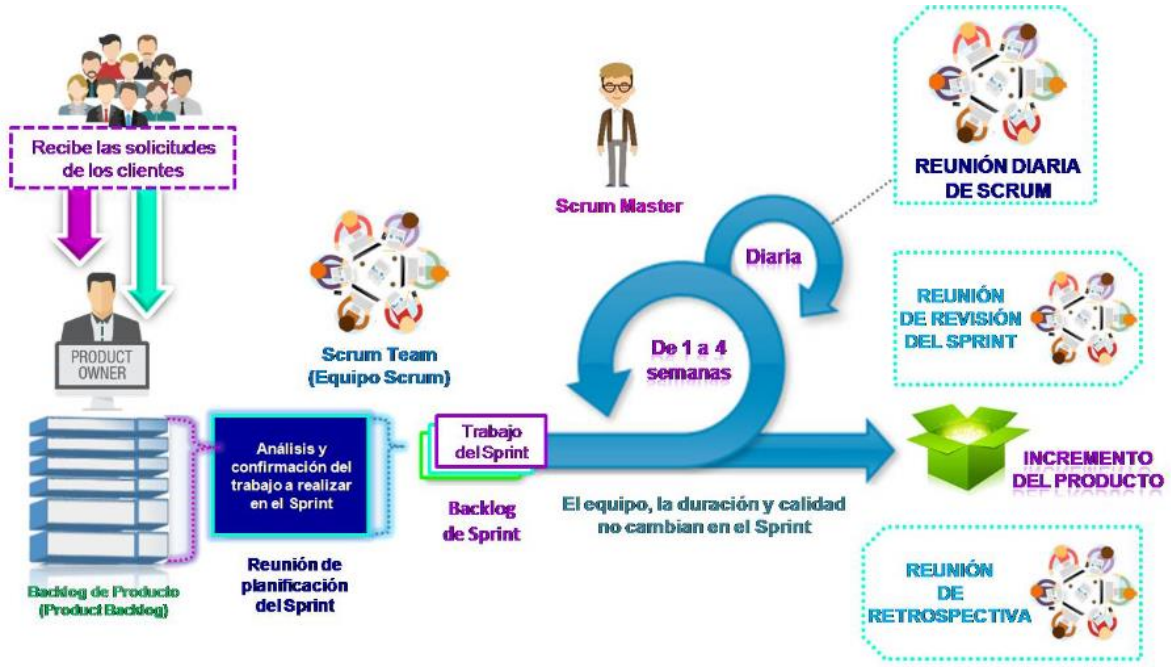
La **programación extrema** optimiza los tiempos y se adapta al desarrollo de sistemas grandes y pequeños sin mayor documentación, el código es claro y simple, así mismo complementa los conocimientos entre los miembros del equipo, gracias a la programación en parejas. Sin embargo, una desventaja de esta metodología ágil, es que después de cada entrega el sistema puede ir creciendo según sean las peticiones del cliente.

✓ **Kanban**. Kanban es una palabra japonesa que significa "**tarjetas visuales**" (kan significa visual, y ban tarjeta). Esta técnica se creó en Toyota, y **se utiliza para controlar el avance del trabajo en el contexto de una línea de producción**. Para el desarrollo de software, gracias a su sencillez KANBAN, simplifica la planificación y la asignación de responsabilidades, en **un tablero se representan los procesos del flujo de trabajo**, cómo mínimo deben existir tres columnas (**Pendiente, En Progreso, Terminado**), la cantidad de tarjetas en estatus pendiente forma parte de lo solicitado por el cliente, aquellas colocadas en progreso dependerán de la capacidad del equipo de trabajo. Se caracteriza por:

- Planificación de tareas
- Tiempos de ciclos reducidos
- Rendimiento del equipo de trabajo
- Métricas visuales
- Menos cuellos de botella
- Entrega continua



✓ **SCRUM:** Es un framework de trabajo que se sigue por los principios ágiles. **El producto se va desarrollando incrementalmente por sprints**, en cada uno de los cuales se construye un **Potentially Shippable Product**, un sistema terminado que sólo tiene ciertas funcionalidades. Todas las entregas regulares y parciales (sprint) del producto final se hacen con una prioridad previamente establecida que nace según el beneficio que aporten al cliente, minimizando los riesgos que pueden surgir de desarrollos extremadamente largos. Es por tal motivo, que **Scrum** está especialmente **indicado para proyectos en entornos complejos**, donde se necesitan obtener resultados de manera inmediata y donde son fundamentales los siguientes aspectos: la innovación, la productividad, la flexibilidad y la competitividad.



El equipo Scrum (Scrum role)

- **Stakeholder:** Es el cliente, su responsabilidad radica en definir los requerimientos (**Product Backlog**), recibir el producto al final de cada iteración y proporcionar el feedback correspondiente.

- **Product Owner:** Es el intermediario de la comunicación entre el cliente (**stakeholder**) y el equipo de desarrollo. Este debe priorizar los requerimientos según sean las necesidades de la solicitud.

- **Scrum Master:** Actúa como facilitador ante todo el equipo de desarrollo, elimina todos aquellos impedimentos que identifique durante el proceso, así mismo se encarga de que el equipo siga los valores y los principios ágiles, las reglas y los procesos de Scrum, incentivando al grupo de trabajo.

- **Scrum Team (Equipo de desarrollo):** Se encarga de desarrollar los casos de uso definidos en el Product Backlog, es un equipo auto gestionado lo que quiere decir que no existe un de jefe de equipo, motivo por el cual todos los miembros se deben de

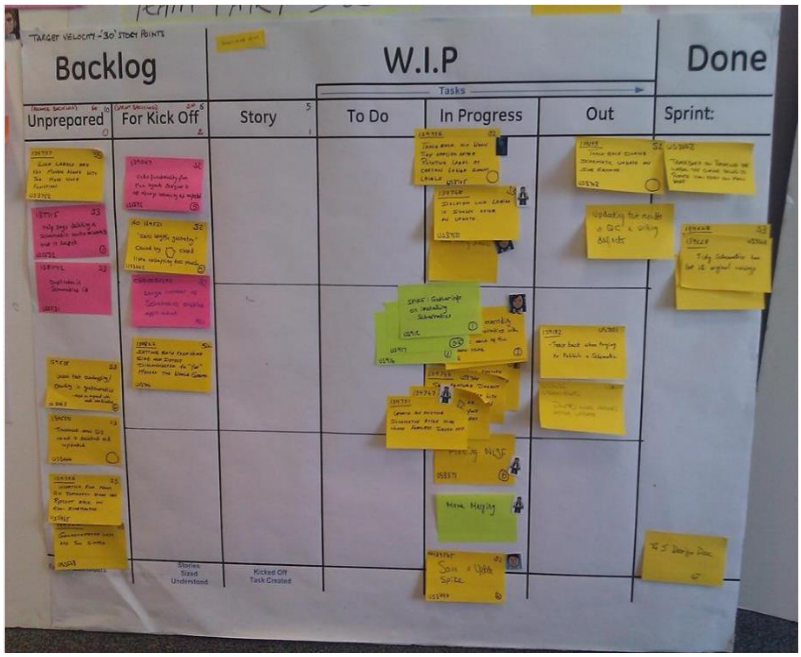
encargar de realizar las estimaciones y en base a la velocidad obtenida en las iteraciones irán construyendo el **Sprint Backlog**.

Scrum- Artefactos

- **Product Backlog:** es una lista priorizada de todas las **User Stories** (características/requisitos del sistema). Cambia con cada sprint.
- **User Story:** se enuncia brevemente con cierta informalidad. Es común que tenga el siguiente formato, que suele ayudar a adecuar el nivel de detalle:

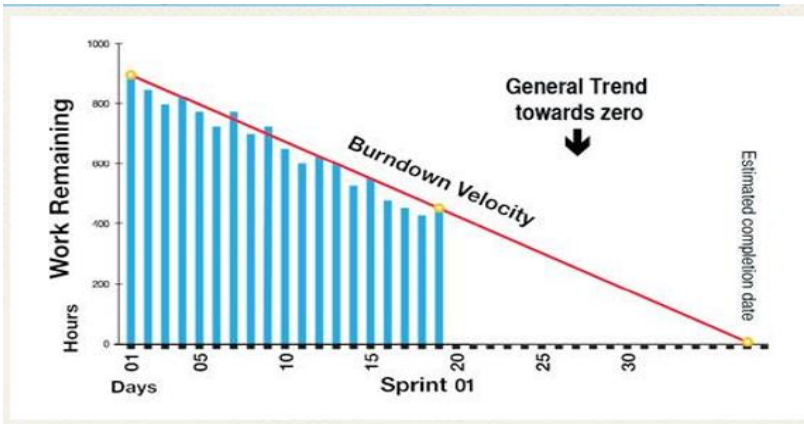
```
Como <tipo_usuario>   As a _________
quiero <feature>      I need _______
para <razón>          so that _____
```

- **Sprint Planning/Backlog:** es el conjunto de User Stories de mayor prioridad que se deciden implementar en el sprint actual. Los backlogs se suelen llevar en notas adhesivas o en herramientas digitales especializadas.



Burndown Chart (Trabajo pendiente)

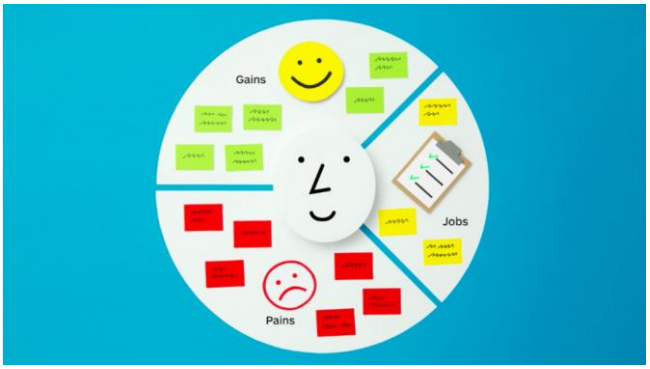
Gráfico **Burndown Chart o Trabajo Pendiente**. Representa gráficamente el trabajo que queda por hacer según avanza el tiempo. Idealmente, bajo una correcta estimación inicial, la gráfica es lineal y las estimaciones del tiempo de cada User Story se cumple, aunque esta predicción es muy compleja. Este gráfico tiene la ventaja de permitir predecir la fecha concreta de finalización del sprint, y permite al Scrum Master analizar el proyecto, ver posibles retrasos o sugerir User Stories para el siguiente sprint.



Scrum Ceremonies (Reuniones)

- **Sprint Planning:** el PO, Scrum Master y Team se reúnen para discutir las User Stories y estimar sus tamaños relativos.
- **Daily Scrum:** es una stand-up meeting (se realiza de pie generalmente), breve (~20 min) y diaria donde el Team discute lo que han completado desde la reunión previa, en lo que están trabajando, y dificultades que pueden estar encontrando.
- **Retrospectiva (Sprint Review**): ocurre al final de un sprint. El Team se reúne con el PO para informar del trabajo completado, y se discute cómo mejorar el proceso en los siguientes sprints.

✓ **Agile Inception:** Empezamos cada proyecto con **Inception**, un ejercicio de descubrimiento que crea el escenario para un producto y produce un backlog inicial. Los **inceptions** es una manera de poner sobre la mesa todos los elementos que componen un producto. Claro hasta ese momento no hay software que funcione, pero se tiene una idea clara de por qué queremos construir algo.



Típicamente el proceso toma un día completo, y la asistencia incluye a todo el equipo (desarrolladores, gerentes de producto, diseñadores, analistas de soporte, UX, ingenieros, testers, etc.).

Un facilitador (**agile coach / scrum master**) dirige la reunión para ayudar a establecer expectativas sobre cómo trabajamos y facilitar la transferencia de conocimiento entre los equipos.

Quiero añadir que hay muchos tipos de inceptions y que es probable que dependiendo de las características del proyecto/producto, se pueden mezclar los eventos y artefactos de cada uno.

- ✓ **Design Sprint:** la metodología de Google. En cualquier organización, la estrategia de negocios es lo más importante. Esta metodología viene de la mano de Google Ventures, un servicio del gigante tecnológico para la innovación y promoción de startups tecnológicas. Se trata de un proceso que dura 5 días en el que el negocio tiene que resolver todas las cuestiones relacionadas con diseño, prototipado, testeo de clientes. La idea es que el trabajo se elabora en etapas de sprints en las que meses de trabajo se pueden reducir en pocas semanas, en vez de esperar a lanzar un producto para entender si la idea es buena, el prototipo proporciona antes la información para evitar posibles errores.

1.2.1.3 ÁGILE vs TRADICIONALES

METODOLOGÍA ÁGILE	METODOLOGÍAS TRADICIONALES
– Flexibilidad ante los cambios del proyecto de forma moderada a rápida – Los clientes hacen parte del equipo de desarrollo – Grupos pequeños (promedio 10 participantes in situ) en el mismo lugar. – Menor dependencia de la arquitectura de software – Continuo Feedback acortando el tiempo de entrega – Diversidad de roles – Basadas en heurísticas a partir de prácticas de producción de código – Procesos menos controlados, pocas políticas y normas – Capacidad de respuesta ante los cambios	– Rigidez ante los cambios, de manera lentos o moderada – Los clientes interactúan con el equipo de desarrollo mediante reuniones – Grupos de gran tamaño y varias veces distribuidos en diferentes sitios – Dependencia de la arquitectura de software mediante modelos – Poco Feedback lo que extiende el tiempo de entrega – Mínimos roles – Basadas en normas de estándares de desarrollo – Procesos muy controlados por políticas y normas – Seguimiento estricto del plan inicial de desarrollo

1.2.2 Herramientas de apoyo al desarrollo del software

En la práctica, para llevar a cabo varias de las etapas vistas en el punto anterior contamos con herramientas informáticas, cuya finalidad principal es automatizar las tareas y ganar fiabilidad y tiempo.

Esto nos va a permitir centrarnos en los **requerimientos del sistema y el análisis del mismo**, que son las causas principales de los fallos del software.

Las **herramientas CASE** son un conjunto de aplicaciones que se utilizan en el desarrollo de software con el objetivo de reducir costes y tiempo del proceso, mejorando por tanto la productividad del proceso.

¿En qué fases del proceso nos pueden ayudar?

En el diseño del proyecto, en la codificación de nuestro diseño a partir de su apariencia visual, detección de errores…

El desarrollo rápido de aplicaciones o RAD es un proceso de desarrollo de software que **comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE**. Hoy en día se suele utilizar para referirnos al desarrollo rápido de interfaces gráficas de usuario o entornos de desarrollo integrado completos.

La tecnología CASE trata de automatizar las fases del desarrollo de software para que mejore la calidad del proceso y del resultado final. En concreto, estas herramientas permiten:

- ✓ Mejorar la planificación del proyecto.

- ✓ Darle agilidad al proceso.
- ✓ Poder reutilizar partes del software en proyectos futuros.
- ✓ Hacer que las aplicaciones respondan a estándares.
- ✓ Mejorar la tarea del mantenimiento de los programas.
- ✓ Mejorar el proceso de desarrollo, al permitir visualizar las fases de forma gráfica.

Normalmente, las herramientas CASE se clasifican en función de las fases del ciclo de vida del software en la que ofrecen ayuda:

- ✓ **U-CASE:** ofrece ayuda en las fases de **planificación** y **análisis de requisitos**.
- ✓ **M-CASE:** ofrece ayuda en **análisis** y **diseño**.
- ✓ **L-CASE:** ayuda en la **programación del software, detección de errores del código, depuración de programas y pruebas y en la generación de la documentación** del proyecto.

Algunos ejemplos de herramientas CASE son ERwin, EasyCASE, OracleDesigner, PowerDesigner, JUnit, StartUML, REM, NetBeans,Eclipse, Spring Tools, Basamiq Mockups, etc