1. Programa y aplicación informática.

Poner un programa en marcha es sinónimo de ejecutarlo. Cuando ejecutamos un programa sólo vemos los resultados pero no el guión seguido por el ordenador para conseguir esos resultados. Este guión es realmente el programa.

 Un programa es una serie de instrucciones dadas al ordenador en un lenguaje entendido por él para decirle exactamente lo que queremos que haga. Si el ordenador no entiende alguna instrucción nos lo comunicará mediante mensajes visualizados por pantalla.

Los pasos a seguir en la realización de un programa, son los siguientes:

* Editar el programa. (Escribir el código).
* Compilarlo.
* Ejecutarlo.
* Depurarlo

Un programa no es capaz de ejecutarse por sí mismo, sino que necesita de otros programas externos que lo complementen. Este complemento se llama librería, y puede ser un programa o simplemente un conjunto de código que realice tareas concretas

**Definiciones:**

* Un programa informático es un conjunto de instrucciones diseñadas y creadas a través del razonamiento lógico y que son almacenadas en ficheros de texto respetando la sintaxis de un determinado lenguaje de programación.
* Una aplicación informática es el conjunto de uno o más programas enlazados o relacionados entre sí, junto con la documentación generada durante el proceso de desarrollo de dicha aplicación.
* Un sistema es el conjunto de elementos relacionados entre sí para la consecución de un determinado fin.

1. Lenguajes de programación
   1. Definición de lenguaje

Un lenguaje de programación, está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones, permitiendo especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias.

Una característica relevante de los lenguajes de programación es, precisamente, que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos para realizar la construcción de un programa de forma colaborativa.

Al proceso por el cual se escribe, se prueba, se depura, se compila y se mantiene el código fuente de un programa informático se le llama programación.

* 1. Elementos de un lenguaje de programación

Un lenguaje de programación se compone de reglas sintácticas y semánticas que describen su estructura y significado respectivamente.

A la forma visible de un lenguaje de programación se le conoce como sintaxis. La mayoría de los lenguajes de programación son puramente textuales, es decir, utilizan secuencias de texto que incluyen palabras, números y puntuación, de manera similar a los lenguajes naturales escritos.

A la forma visible de un lenguaje de programación se le conoce como sintaxis. La mayoría de los lenguajes de programación son puramente textuales, es decir, utilizan secuencias de texto que incluyen palabras, números y puntuación, de manera similar a los lenguajes naturales escritos.

* 1. Tipos de lenguaje de programación
* Lenguajes interpretados: Un programa escrito en un lenguaje interpretado requiere de un programa auxiliar (el intérprete), que traduce los comandos de los programas según sea necesario.
* Lenguajes compilados: Un programa escrito en un lenguaje “compilado” se traduce a través de un programa anexo llamado compilador que, a su vez, crea un nuevo archivo independiente que no necesita ningún otro programa para ejecutarse a sí mismo. Este archivo se llama ejecutable.

Un programa escrito en un lenguaje compilado posee la ventaja de no necesitar un programa anexo para ser ejecutado una vez que ha sido compilado. Además, la ejecución se vuelve más rápida.

Sin embargo, no es tan flexible como un programa escrito en lenguaje interpretado, ya que cada modificación del archivo fuente, requiere de la compilación del programa para aplicar los cambios.

Un programa compilado tiene la ventaja de garantizar la seguridad del código fuente. Por otro lado, ciertas aplicaciones aseguradas necesitan confidencialidad de código para evitar las copias ilegales (transacciones bancarias, pagos en línea, comunicaciones seguras...).

Por último, algunos lenguajes pertenecen a ambas categorías dado que el programa escrito en estos lenguajes puede, en ciertos casos, sufrir una fase de compilación intermediaria, en un archivo escrito en un lenguaje ininteligible (por lo tanto diferente al archivo fuente) y no ejecutable (requeriría un intérprete).

Como veremos más adelante, éste es el caso de Java, en el que se realiza un doble proceso de compilación e interpretación.

* 1. Tipos de código generados por un lenguaje de programación

Código fuente es el código escrito por un programador en un determinado lenguaje de programación, definiendo el conjunto de instrucciones que debe realizar la computadora.

Este código fuente, debe ser traducido al lenguaje máquina del procesador. Este proceso de traducción se llama compilación y el resultado de dicho proceso es el código objeto.

Código objeto es el código máquina generado tras compilar un determinado código fuente.

Por lo general, una aplicación informática está formada por un conjunto de programas, cada uno de los cuales realiza una operación concreta además de una serie de librerías de código. Por tanto, para poder ejecutar una aplicación es necesario enlazar y combinar los códigos objeto de cada programa y las librerías necesarias, obteniendo un archivo ejecutable.

* 1. Paradigmas de programación

Un paradigma de programación determina la visión y métodos de un programador en la construcción de un programa o subprograma, se puede decir también que son diferentes formas de pensar la solución de problemas. Ejemplos de paradigmas de programación son los siguientes:

* Imperativo
* Declarativo
* Estructurado
* Orientado a objetos
* Funcional
* Lógico
  1. La programación estructurada

Es un estilo de programación con el cual el programador elabora programas, cuya estructura es la más clara posible, mediante el uso de tres estructuras básicas de control lógico:

* Secuencia
* Selección
* Iteración.

La programación estructurada tiene un teorema fundamental: Cualquier programa puede ser elaborado utilizando únicamente las tres estructuras básicas (secuencia, selección, iteración).

Entre las ventajas de la programación estructurada, se encuentran las siguientes:

* Programas más fáciles de entender
* Reducción del esfuerzo en las pruebas. Los errores se pueden detectar y reparar más fácilmente. Reducción de los costos de mantenimiento.
* Programas más sencillos y más rápidos.
* Aumento en la productividad del programador.
* Los programas quedan mejor documentados internamente.

**Inconvenientes.**

* Se obtiene un único bloque de programa, que cuando se hace demasiado grande puede resultar problemático en cuanto a su manejo, esto se resuelve empleando la programación modular.
  1. La programación orientada a objetos

Se caracteriza por la idea central, el objeto. Podemos decir que cualquier cosa, en serio, cualquier cosa, que nos rodea es un objeto

Las características fundamentales de este enfoque son las siguientes:

* Existe la posibilidad de representar directamente las entidades del mundo real en los escenarios informáticos, sin necesidad de deformarlas.
* Se facilita enormemente la reutilización y modificación del software.
* Se puede trabajar en equipo, desarrollando en paralelo módulos para una aplicación con un esfuerzo mínimo en coordinación.

Existen una serie de pasos que sirven de guía a la hora de modelar un sistema empleando orientación a objetos:

* Identificar los objetos que intervienen en él.
* Agrupar en clases a todos aquellos objetos que tengan características y comportamiento comunes. Identificar los datos y operaciones de cada una de las clases.
* Identificar las relaciones que puedan existir entre las clases.

La programación orientada a objetos presenta ciertas características que la hacen destacar y al mismo tiempo la diferencian de la programación estructurada o modular, alguna de las cuales se describen a continuación:

* Protección de la información: El encapsulamiento de datos y procedimientos junto con unos niveles de acceso a la información forman unos límites que proporcionan una gran protección a la información contenida en el objeto.
* Rápido desarrollo: La posibilidad de volver a usar objetos (reutilización objetos) ya definidos y probados en otros programas, disponibles en librerías, permite que el desarrollo de programas se realice con gran rapidez.
* Fácil mantenimiento: En el caso de una modificación o mejora de una aplicación desarrollada con objetos, las características de la programación orientada a objetos, permiten que se realice con una gran facilidad, pues se reduce a modificar los componentes de algunos objetos o a crear nuevos objetos aprovechando los ya existentes, haciendo que hereden sus características.

Otra característica fundamental de la POO es el polimorfismo y la herencia. Esta característica está basada en la posibilidad de definir objetos genéricos en base a una serie de propiedades comunes, permitiendo concretar más adelante dicho objeto genérico. Por ejemplo, se puede definir el objeto Vehículo y, a partir de éste, definir los objetos Coche, Barco, Tren, Bicicleta, Avión…

La base de la POO está en la definición de Clases y Objetos. Una clase es la definición genérica de un tipo de objeto, definiendo sus propiedades y su comportamiento. Mientras que un objeto es la realización concreta (o instancias) de una clase.

Para finalizar, diremos que un modelo de objetos es más cercano a la realidad que un modelo funcional. Además, el modelo orientado a objetos facilita la integridad de módulos que han sido realizados por separado para no correr riesgos en el manejo de los datos.

1. El Lenguaje de programación Java
   1. Historial del lenguaje Java

El creador de Java es James Gosling.

James se encerró (Java se creó en 18 meses) en su oficina y creó un nuevo lenguaje que mejoraba las cosas que no le gustaban de C++

Inicialmente Java se llamó Oak (roble en inglés), aunque tuvo que cambiar de denominación, debido a que dicho nombre ya estaba registrado por otra empresa.

Java fue lanzado por Sun en el otoño de 1995. Tres de las principales razones que llevaron a crear Java son:

* Creciente necesidad de interfaces mucho más cómodas e intuitivas que los sistemas de ventanas que proliferaban hasta el momento.
* Fiabilidad del código y facilidad de desarrollo. Gosling observó que muchas de las características que ofrecían C o C++ aumentaban de forma alarmante el gran coste de pruebas y depuración. Por ello en los sus ratos libres creó un lenguaje de programación donde intentaba solucionar los fallos que encontraba en C++.
* Enorme diversidad de controladores electrónicos.

Por todo ello, en lugar de tratar únicamente de optimizar las técnicas de desarrollo y dar por sentada la utilización de C o C++, el equipo de Gosling se planteó que tal vez los lenguajes existentes eran demasiado complicados como para conseguir reducir de forma apreciable la complejidad de desarrollo asociada a ese campo. Por este motivo, su primera propuesta fue idear un nuevo lenguaje de programación lo más sencillo posible, con el objeto de que se pudiese adaptar con facilidad a cualquier entorno de ejecución.

Basándose en el conocimiento y estudio de gran cantidad de lenguajes, este grupo decidió recoger las características esenciales que debía tener un lenguaje de programación moderno y potente, pero eliminando todas aquellas funciones que no eran absolutamente imprescindibles.

En diciembre de 1998, Sun anuncia el Java 2 como marca comercial de la versión JDK 1.2. Sun, la empresa que desarrollaba Java fue adquirida por Oracle en 2010.

A lo largo de muchos años, el lanzamiento de nuevas versiones de Java se ha hecho en espacios largos de tiempo (entre 2 y 5 años).

En septiembre de 2017 se libera la versión 9 de Java, y a partir de ahí, se pasa a liberar nuevas versiones por calendario, y no por funcionalidades. Cada 6 meses se libera una nueva versión de Java, siendo alguna de ellas una versión LTS (long term support) para garantizar compatibilidad durante un período aceptable de tiempo.

A día de hoy, la última versión pendiente de liberar es Java 20, y la última LTS liberada es la 17. La próxima versión LTS prevista es la 21 y su lanzamiento está planificado para Septiembre de 2023.

* 1. Características del lenguaje Java
* Lenguaje Simple: Java posee una curva de aprendizaje muy rápida. Resulta relativamente sencillo escribir programas interesantes desde el principio.
* Orientado a objetos: Java fue diseñado como un lenguaje orientado a objetos desde el principio.
* Distribuido: Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.
* Compilado e interpretado: Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina,. Por otra parte, es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real (runtime).
* Robusto: Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución.
* Seguro: Dada la naturaleza distribuida de Java, donde las applets se bajan desde cualquier punto de la Red, la seguridad se impuso como una necesidad de vital importancia. A nadie le gustaría ejecutar en su ordenador programas con acceso total a su sistema, procedentes de fuentes desconocidas. Así que se implementaron barreras de seguridad en el lenguaje y en el sistema de ejecución en tiempo real.
* Independiente a la arquitectura: Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos.
* Multihilo: Hoy en día ya se ven como terriblemente limitadas las aplicaciones que sólo pueden ejecutar una acción a la vez. Java soporta sincronización de múltiples hilos de ejecución (multithreading) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas. Así, mientras un hilo se encarga de la comunicación, otro puede interactuar con el usuario mientras otro presenta una animación en pantalla y otro realiza cálculos.
* Dinámico: El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.
  1. La máquina virtual de Java

El código fuente en Java se escribe en ficheros que tienen la extensión .java. Por lo general, se escribe un fichero de código fuente por cada clase definida para el programa.

Este código fuente es compilado para obtener código objeto en ficheros con la extensión .class (se obtiene un fichero .class por cada .java). Este código objeto está escrito en un lenguaje cercano al código máquina, llamado bytecode. La principal ventaja de este paso es que el bytecode obtenido para una clase es siempre igual, independientemente de la máquina en la que se pretenda ejecutar.

La portabilidad e independencia de máquina o sistemas operativos, se obtiene a través de la máquina virtual de java (Java Virtual Machine, JVM). Esta “máquina” es un software necesario para poder ejecutar cualquier aplicación escrita en java y es el encargado de traducir el bytecode en lenguaje máquina, ya adaptado a la máquina concreta sobre la que se está ejecutando. Por tanto, será necesario instalar la JVM apropiada al S.O. del dispositivo.

1. Fases en el desarrollo de una aplicación.
   1. Metodología para el desarrollo de software

El proceso de construcción del software requiere, como cualquier otra ingeniería, identificar las tareas que se han de realizar sobre el software y aplicar esas tareas de una forma ordenada y efectiva. Adicionalmente, el desarrollo del software se debe realizar por un conjunto coordinado de personas simultáneamente, y por lo tanto sus esfuerzos deben estar dirigidos por una misma metodología que permita estructurar las diferentes fases del desarrollo.

Características de las metodologías:

* Define como se divide un proyecto en fases y las tareas a realizar en cada una.
* Para cada una de las fases está especificado cuales son las entradas que reciben y las salidas que producen.
* Tienen alguna forma de gestionar el proyecto.

Teniendo esto en cuenta establecemos la siguiente definición:

Una metodología es un modo sistemático de producir software.

Existen multitud de metodologías, resultando cada una de ellas más adecuada a un tipo u otro de proyectos. Una de las metodologías que han sido más utilizadas en España es Métrica v3, definida por el Ministerio de Administraciones Públicas para el desarrollo de aplicaciones informáticas en el ámbito de las administraciones públicas, aunque hoy en día se utiliza en otros muchos ámbitos. Con todo, a día de hoy se están imponiendo las conocidas como metodologías ágiles, centradas en que el cliente obtenga resultados iterativos lo más pronto posible. Algunas de ellas son la programación extrema (XP), Scrum, Kanban, ...

* 1. Ciclo de vida del software.

Al igual que en otros sistemas de ingeniería, los sistemas de software requieren un tiempo y esfuerzo considerable para su desarrollo y deben permanecer en uso por un periodo mucho mayor.

Durante este tiempo de desarrollo y uso, desde que se detecta la necesidad de construir un sistema de software hasta que este es retirado, se identifican varias etapas que en conjunto se denominan el ciclo de vida del software.

**Usualmente se consideran las etapas:**

* Especificación y análisis de requisitos
* Diseño del sistema
* Implementación del software
* Aplicación y pruebas
* Entrega y mantenimiento.

Cada fase estará definida por un conjunto de tareas a realizar en ella y por el producto generado antes de pasar a la siguiente etapa. Este producto podrá ser considerado como un entregable o no, en función de si es para entregarlo al cliente como parte del proyecto o forma parte de la documentación interna.

Un aspecto esencial dentro de las tareas del desarrollo del software es la documentación de todos los elementos y especificaciones en cada fase. Cada etapa tiene como entrada uno o varios documentos procedentes de las etapas anteriores y produce otros documentos de salida.

Después de cada etapa se realiza una revisión para comprobar si se puede pasar a la siguiente.

* 1. Análisis

Durante esta fase, se estudia a fondo el problema que pretende resolverse, tratando de comprender necesidad del cliente y de definir las especificaciones del producto que pretende desarrollarse.

En un primer lugar, se obtendrá una descripción grosso modo, de lo que se desea obtener, definiendo los distintos usuarios (o roles) que utilizarán el sistema. Además, se describirán a grandes rasgos las principales operaciones que realizará la aplicación. Esta descripción no sigue ningún estándar y se hace de manera que el cliente pueda ir entendiendo, desde las primeras reuniones, la solución propuesta.

Tras esa primera descripción, el producto a desarrollar se define a través de requisitos que se van obteniendo a partir de las sucesivas reuniones con el cliente.

Un requisito puede definirse como la capacidad que el sistema debe tener porque el cliente lo ha pedido explícita o implícitamente.

**La obtención de requisitos es difícil por varias razones:**

* La naturaleza de los requisitos es cambiante.
* Surgen nuevos requisitos en cualquier momento.
* El cliente puede no tenerlos claros.
* Pueden existir malos entendidos debidos a:
  + Falta de conocimientos por parte del equipo desarrollador sobre el problema.
  + Falta de conocimientos técnicos (informáticos) por parte del cliente para expresarse con claridad.

**Los requisitos se dividen en dos tipos:**

* Requisitos funcionales: dicen qué debe hacer el sistema, en el sentido de servicios proporcionados al usuario.
* Requisitos no funcionales: hablan de características del sistema, como pueden ser la fiabilidad, la mantenibilidad, el sistema operativo, la plataforma hardware, el aspecto gráfico, etc.

Otros elementos importantes en la fase de análisis son los casos de uso, que constituyen otra forma de especificar los requisitos de un sistema. Un caso de uso:

Describe la interacción entre un actor externo al sistema y el sistema con texto en lenguaje natural. Representa los requisitos funcionales desde el punto de vista del usuario y por lo tanto produce un resultado observable por él.

Es iniciado por un único actor.

Realiza una funcionalidad concreta.

Mediante un diagrama de casos de uso podemos representar los diferentes casos de uso, las relaciones entre ellos y los actores participantes.

Si un caso de uso resulta demasiado complejo, podemos elaborar una ficha que nos sirva para estudiar, paso a paso, cuál es la interacción entre los diferentes actores y el sistema.

Un proceso clave en la fase de análisis y toma de requisitos son las entrevistas con el cliente, a través de las cuales se van comprendiendo sus necesidades que, posteriormente, serán traducidas a requisitos y casos de uso. El proceso habitual suele ser:

En una reunión final, se dará por cerrado el análisis, aprobando el Documento de Análisis que contiene todos los requisitos y casos de uso, así como las especificaciones del sistema. Este documento se entiende como un contrato por el que el equipo de desarrollo se compromete a desarrollar una aplicación que realice exactamente las tareas que se recogen en dicho documento.

Dicho de otro modo, cualquier modificación que quiera hacerse en el proyecto deberá encontrarse dentro del alcance del documento de análisis. En caso contrario, se entenderá fuera del contrato y, por tanto, no se realizará (o se renegociará el contrato).

* 1. Diseño

Para dejar constancia de los diseños se deben utilizar lenguajes lo más formales posible, como tablas, diagramas y pseudocódigo.

UML son las siglas de lenguaje unificado de modelado (unified modeling language). Se trata de un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema (en nuestro caso, un sistema software).

Por lo general, el producto de esta fase es el Documento de Diseño que, entre otros elementos, suele contener:

* Una descripción y un boceto del interfaz de la aplicación. En ocasiones se construyen maquetas sin funcionalidad.
* Diagrama UML de Componentes, indicando la arquitectura del sistema y las relaciones entre los distintos subsistemas (base de datos, interfaz, etc.)
* Diagrama UML de Clases, mostrando las distintas entidades que aparecen en el modelo de datos. Si la aplicación se va a realizar siguiendo la POO, se indican las clases, con sus atributos, métodos y relaciones entre ellas.
* Diagrama UML de Secuencias, mostrando el flujo de la aplicación en los distintos casos de uso. Descripción de cada una de las clases que conformarán el modelo de datos.
  1. Implementación o codificación

En esta fase el programador recibe las especificaciones del diseño y transforma esas especificaciones, que pueden estar en diversos formatos, en un programa o módulo que las efectúe. Esta tarea de codificación puede estar semi-automatizada o realizarse de forma completamente manual.

Para realizar el trabajo, este debe ser dividido en pequeños módulos que luego se reparten entre individuos o pequeños grupos. Este reparto se realiza atendiendo a un gráfico de dependencias entre tareas, siempre con la idea en mente de paralelizar tanto trabajo como sea posible.

El producto de la fase de codificación es el programa propiamente dicho, listo para ser testeado en la fase de pruebas. Otros productos de esta fase son:

* Los manuales de configuración e instalación
* El manual de usuario
* La documentación del código, donde se recoge la funcionalidad de cada clase.
  1. Pruebas

Tras la codificación llega el momento de realizar la mayor parte de las pruebas a nuestro software.

Las pruebas consumen mucho tiempo, pero deben hacerse de un modo sistemático para asegurar que el resultado cumple con el grado de calidad exigido, siguiendo alguna de las métricas existentes.

En esta parte del desarrollo se trata de encontrar errores, no sólo de codificación, sino también los relativos a la especificación o el diseño.

Las pruebas que se van haciendo durante el ciclo de vida son:

* Pruebas unitarias
* Pruebas de integración
* Pruebas de sistema
* Pruebas de tensión
* Pruebas de desempeño

Inevitablemente también hay que añadir la correspondiente documentación de las pruebas realizadas, en la que se indica tanto el plan de pruebas que se va a seguir como los resultados de la ejecución del plan.

* 1. Documentación

En esta fase es necesario reunir todos los documentos generados y clasificarlos según el nivel técnico de sus descripciones. Es muy importante distinguir entre:

La documentación orientada a futuros desarrollos o modificaciones de mantenimiento.

La documentación de uso y aplicación.

Una vez que se ha finalizado el proyecto se debe tener una documentación útil para el mantenimiento posterior y para la operación normal por parte de los usuarios.

En un proyecto no es necesario que estén todos, sobre todo en proyectos pequeños, pero los documentos habituales son:

* Documentos previos de Ingeniería:
  + - Estudio de viabilidad
    - Análisis de riesgo
    - Análisis costo-beneficio
    - Informe de decisión de sistemas
    - Plan de proyecto
* Documentos de Desarrollo del Proyecto:
  + - Requisitos funcionales
    - Requisitos de datos
    - Especificaciones de sistema/subsistema
    - Especificaciones del programa
    - Especificaciones de la base de datos
    - Plan de Pruebas
* Documentos de usuarios y de explotación:
  + - Manual de usuario
* Manual de operación
  + - Manual de mantenimiento
    - Plan de instalación
  1. Explotación y Mantenimiento

Como etapa final en el ciclo de vida del software se debe realizar la entrega de la primera versión al cliente y considerar las posibles posteriores modificaciones de mantenimiento.

Dentro del mantenimiento se deben incluir no solamente las correcciones de errores detectados posteriormente por el cliente, sino también las modificaciones necesarias para actualización, e incluso las peticiones de cambios por parte del cliente.

1. Metodologías ágiles.

El desarrollo ágil de software refiere a métodos basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan mediante la colaboración de grupos auto organizados y multidisciplinarios.

Otra característica fundamental de este tipo de metodologías es que la mayoría desarrolla el software en lapsos cortos, lo que minimiza los riesgos. El software desarrollado en una unidad de tiempo es llamado una iteración, la cual debe durar poco tiempo (de una a cuatro semanas).

Cada iteración del ciclo de vida incluye todas las fases tradicionales vistas anteriormente, pero realizadas de manera simultánea y, evidentemente, más ligera, ya que sólo debe realizarse una pequeña parte del proyecto.

Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, sino que la meta es tener una «demo» (sin errores) al final de cada iteración. Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto.

Los métodos ágiles enfatizan las comunicaciones cara a cara en vez de la documentación. Los métodos ágiles también enfatizan que el software funcional es la primera medida del progreso. Combinado con la preferencia por las comunicaciones cara a cara, generalmente los métodos ágiles son criticados y tratados como "indisciplinados" por la falta de documentación técnica (“planificación por post-it”). La metodología ágil más implantada hoy en día es Scrum.

Otra técnica relacionada es la técnica Pomodoro, para gestionar el tiempo, cuyo objetivo es dividir todo el trabajo en tareas que puedan realizarse en 25 minutos.