Metodología

Un objetivo de décadas ha sido el encontrar procesos y [metodologías](https://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software), que sean sistemáticas, predecibles y repetibles, a fin de mejorar la productividad en el desarrollo y la calidad del producto *software*, en pocas palabras, determina los pasos a seguir y como realizarlos para finalizar una tarea.

**Etapas del proceso**

La ingeniería de *software* requiere llevar a cabo numerosas tareas agrupadas en etapas, al conjunto de estas etapas se le denomina [ciclo de vida](https://es.wikipedia.org/w/index.php?title=Proceso_para_el_desarrollo_de_%27%27software%27%27&action=edit&redlink=1). Las etapas comunes a casi todos los modelos de ciclo de vida son las siguientes:

**Obtención de los requisitos**

Se debe identificar sobre qué se está trabajando, es decir, el tema principal que motiva el inicio del estudio y creación del nuevo *software* o modificación de uno ya existente. A su vez identificar los recursos que se tienen, en esto entra el conocer los recursos humanos y materiales que participan en el desarrollo de las actividades. Es importante entender el contexto del negocio para identificar adecuadamente los requisitos.

Se tiene que tener dominio de la información de un problema, lo cual incluye los datos fuera del *software* (usuarios finales, otros sistemas o dispositivos externos), los datos que del sistema (por la interfaz de usuario, interfaces de red, reportes, gráficas y otros medios) y los almacenamientos de datos que recaban y organizan objetos persistentes de datos (por ejemplo, aquellos que se conservan de manera permanente).

También hay que ver los puntos críticos, lo que significa tener de una manera clara los aspectos que entorpecen y limitan el buen funcionamiento de los procedimientos actuales, los problemas más comunes y relevantes que se presentan, los motivos que crean insatisfacción y aquellos que deben ser cubiertos a plenitud. Por ejemplo: ¿El contenido de los reportes generados, satisface realmente las necesidades del usuario? ¿Los tiempos de respuesta ofrecidos, son oportunos?, etc.

Hay que definir las funciones que realizará el *software* ya que estas ayudan al usuario final y al funcionamiento del mismo programa.

Se tiene que tener en cuenta cómo será el comportamiento del *software* ante situaciones inesperadas como lo son por ejemplo una gran cantidad de usuarios usando el *software* o una gran cantidad de datos entre otros.

**Análisis de requisitos**

Extraer los requisitos de un producto *software* es la primera etapa para crearlo. Durante la fase de análisis, el cliente plantea las necesidades que se presenta e intenta explicar lo que debería hacer el *software* o producto final para satisfacer dicha necesidad mientras que el desarrollador actúa como interrogador, como la persona que resuelve problemas. Con este análisis, el ingeniero de sistemas puede elegir la función que debe realizar el *software* y establecer o indicar cuál es la interfaz más adecuada para el mismo.

El análisis de requisitos puede parecer una tarea sencilla, pero no lo es debido a que muchas veces los clientes piensan que saben todo lo que el *software* necesita para su buen funcionamiento, sin embargo se requiere la habilidad y experiencia de algún especialista para reconocer requisitos incompletos, ambiguos o contradictorios. Estos requisitos se determinan tomando en cuenta las necesidades del usuario final, introduciendo técnicas que nos permitan mejorar la calidad de los sistemas sobre los que se trabaja.

El resultado del análisis de requisitos con el cliente se plasma en el documento ERS (especificación de requisitos del sistema), cuya estructura puede venir definida por varios estándares, tales como [CMMI](https://es.wikipedia.org/wiki/CMMI). Asimismo, se define un [diagrama de entidad/relación](https://es.wikipedia.org/wiki/Modelo_entidad-relaci%C3%B3n), en el que se plasman las principales entidades que participarán en el desarrollo del *software*.

La captura, análisis y especificación de requisitos (incluso pruebas de ellos), es una parte crucial; de esta etapa depende en gran medida el logro de los objetivos finales. Se han ideado modelos y diversos procesos metódicos de trabajo para estos fines. Aunque aún no está formalizada, ya se habla de la [ingeniería de requisitos](https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_requisitos).

La IEEE Std. 830-1998 normaliza la creación de las especificaciones de requisitos de *software* (Software Requirements Specification).

Finalidades del análisis de requisitos:

* Brindar al usuario todo lo necesario para que pueda trabajar en conjunto con el *software* desarrollado obteniendo los mejores resultados posibles.
* Tener un control más completo en la etapa creación del *software*, en cuanto a tiempo de desarrollo y costos.
* Utilización de métodos más eficientes que permitan el mejor aprovechamiento del *software* según sea la finalidad de uso del mismo.
* Aumentar la calidad del *software* desarrollado al disminuir los riesgos de mal funcionamiento.

No siempre en la etapa de "análisis de requisitos" las distintas metodologías de desarrollo llevan asociado un estudio de viabilidad y/o estimación de costes. El más conocido de los modelos de estimación de coste del *software* es el modelo [COCOMO](https://es.wikipedia.org/wiki/COCOMO)

**Limitaciones**

Los *software* tienen la capacidad de emular inteligencia creando un modelo de ciertas características de la inteligencia humana pero solo posee funciones predefinidas que abarcan un conjunto de soluciones que en algunos campos llega a ser limitado. Aun cuando tiene la capacidad de imitar ciertos comportamientos humanos no es capaz de emular el pensamiento humano porque actúa bajo condiciones.

Otro aspecto limitante de los software proviene del proceso totalmente mecánico que requiere de un mayor esfuerzo y tiempos elevados de ejecución lo que lleva a tener que implementar el *software* en una máquina de mayor capacidad.

**Especificación**

La especificación de requisitos describe el comportamiento esperado en el *software* una vez desarrollado. Gran parte del éxito de un proyecto de *software* radicará en la identificación de las necesidades del negocio (definidas por la alta dirección), así como la interacción con los usuarios funcionales para la recolección, clasificación, identificación, priorización y especificación de los requisitos del *software*.

Entre las técnicas utilizadas para la especificación de requisitos se encuentran:

* [Caso de uso](https://es.wikipedia.org/wiki/Caso_de_uso)
* [Historias de usuario](https://es.wikipedia.org/wiki/Historias_de_usuario)

Siendo los primeros más rigurosas y formales, los segundas más ágiles e informales.

**Arquitectura**

La integración de infraestructura, desarrollo de aplicaciones, bases de datos y herramientas gerenciales, requieren de capacidad y liderazgo para poder ser conceptualizados y proyectados a futuro, solucionando los problemas de hoy. El rol en el cual se delegan todas estas actividades es el del Arquitecto.

El arquitecto de *software* es la persona que añade valor a los procesos de negocios gracias a su valioso aporte de soluciones tecnológicas.

La arquitectura de sistemas en general, es una actividad de planeación, ya sea a nivel de infraestructura de red y *hardware*, o de *software*.

Lo principal en este punto es poner en claro los aspectos lógicos y físicos de las salidas, modelos de organización y representación de datos, entradas y procesos que componen el sistema, considerando las bondades y limitaciones de los recursos disponibles en la satisfacción de las pacificaciones brindadas para el análisis.

Hay que tener en consideración la arquitectura del sistema en la cual se va a trabajar, elaborar un plan de trabajo viendo la prioridad de tiempo y recursos disponibles. En los diseños de salidas entra los que es la interpretación de requerimientos lo cual es el dominio de información del problema, las funciones visibles para el usuario, el comportamiento del sistema y un conjunto de clases de requerimientos que agrupa los objetos del negocio con los métodos que les dan servicio.

La arquitectura de *software* consiste en el diseño de componentes de una aplicación (entidades del negocio), generalmente utilizando patrones de arquitectura. El diseño arquitectónico debe permitir visualizar la interacción entre las entidades del negocio y además poder ser validado, por ejemplo por medio de diagramas de secuencia. Un diseño arquitectónico describe en general el *cómo* se construirá una aplicación de *software*. Para ello se documenta utilizando diagramas, por ejemplo:

* [Diagrama de clases](https://es.wikipedia.org/wiki/Diagrama_de_clases)
* [Diagrama de base de datos](https://es.wikipedia.org/w/index.php?title=Diagrama_de_base_de_datos&action=edit&redlink=1)
* [Diagrama de despliegue](https://es.wikipedia.org/wiki/Diagrama_de_despliegue)
* [Diagrama de secuencia](https://es.wikipedia.org/wiki/Diagrama_de_secuencia)

Los diagramas de clases y de base de datos son los mínimos necesarios para describir la arquitectura de un proyecto que iniciará a ser codificado. Dependiendo del alcance del proyecto, complejidad y necesidades, el arquitecto elegirá cuales de los diagramas se requiere elaborar.

Las herramientas para el diseño y modelado de *software* se denominan [CASE](https://es.wikipedia.org/wiki/Herramienta_CASE) (*Computer Aided Software Engineering*) entre las cuales se encuentran:

* Enterprise Architect
* Microsoft Visio for Enterprise Architects

**Programación**

Implementar un diseño en código puede ser la parte más obvia del trabajo de ingeniería de *software*, pero no necesariamente es la que demanda mayor trabajo y ni la más complicada. La complejidad y la duración de esta etapa está íntimamente relacionada al o a los [lenguajes de programación](https://es.wikipedia.org/wiki/Lenguajes_de_programaci%C3%B3n) utilizados, así como al diseño previamente realizado.

**Desarrollo de la aplicación**

Para el desarrollo de la aplicación es necesario considerar cinco fases para tener una aplicación o programa eficiente, estas son:

* **Desarrollo de la infraestructura:**Esta fase permite el desarrollo y la organización de los elementos que formaran la infraestructura de la aplicación, con el propósito de finalizar la aplicación eficientemente.
* **Adaptación del paquete:**El objetivo principal de esta fase es entender de una manera detallada el funcionamiento del paquete, esto tiene como finalidad garantizar que el paquete pueda ser utilizado en su máximo rendimiento, tanto para negocios o recursos. Todos los elementos que componen el paquete son inspeccionados de manera detallada para evitar errores y entender mejor todas las características del paquete.
* **Desarrollo de unidades de diseño de interactivas:**En esta fase se realizan los procedimientos que se ejecutan por un diálogo usuario-sistema. Los procedimientos de esta fase tienen como objetivo principal:

1. Establecer específicamente las acciones que debe efectuar la unidad de diseño.
2. La creación de componentes para sus procedimientos.
3. Ejecutar pruebas unitarias y de integración en la unidad de diseño.

* **Desarrollo de unidades de diseño batch:**En esta fase se utilizan una serie de combinación de técnicas, como [diagrama de flujo](https://es.wikipedia.org/wiki/Diagrama_de_flujo), diagramas de estructuras, tablas de decisiones, etc. Cualquiera a utilizar será beneficioso para plasmar de manera clara y objetiva las especificaciones y que así el programador tenga mayor comprensión a la hora de programar y probar los programas que le corresponden.
* **Desarrollo de unidades de diseño manuales:**En esta fase el objetivo central es proyectar todos los procedimientos administrativos que desarrollarán en torno a la utilización de los componentes computarizados.

**Pruebas de *software***

Consiste en comprobar que el *software* realice correctamente las tareas indicadas en la especificación del problema. Una técnica es probar por separado cada módulo del *software* ([prueba unitaria](https://es.wikipedia.org/wiki/Prueba_unitaria)), y luego probarlo de manera integral ([pruebas de integración](https://es.wikipedia.org/wiki/Pruebas_de_integraci%C3%B3n)), para así llegar al objetivo. Se considera una buena práctica el que las pruebas sean efectuadas por alguien distinto al desarrollador que la programó, idealmente un área de pruebas; sin perjuicio de lo anterior el programador debe hacer sus propias pruebas. En general hay dos grandes maneras de organizar un área de pruebas, la primera es que esté compuesta por personal inexperto y que desconozca el tema de pruebas, de esta manera se evalúa que la documentación entregada sea de calidad, que los procesos descritos son tan claros que cualquiera puede entenderlos y el *software* hace las cosas tal y como están descritas. El segundo enfoque es tener un área de pruebas conformada por programadores con experiencia, personas que saben sin mayores indicaciones en qué condiciones puede fallar una aplicación y que pueden poner atención en detalles que personal inexperto no consideraría.

De acuerdo con [Roger S. Pressman](https://es.wikipedia.org/w/index.php?title=Roger_S._Pressman&action=edit&redlink=1), el proceso de pruebas se centra en los procesos lógicos internos del *software*, asegurando que todas las sentencias se han comprobado, y en los procesos externos funcionales, es decir, la realización de pruebas para la detección de errores. Se requiere poder probar el *software* con sujetos reales que puedan evaluar el comportamiento del *software* con el fin de proporcionar [realimentación](https://es.wikipedia.org/wiki/Realimentaci%C3%B3n) a los desarrolladores. Es importante que durante el proceso de desarrollo del *software* no se pierda contacto con los interesados o solicitantes del desarrollo de *software*, de esta manera los objetivos del proyecto se mantendrán vigentes y se tendrá una idea clara de los aspectos que tienen que probarse durante el período de pruebas.

**Implementación**

Una implementación es la realización de una especificación técnica o algoritmos con un programa, componente *software*, u otro sistema de cómputo. Muchas especificaciones son dadas según a su especificación o un estándar. Las especificaciones recomendadas según el [World Wide Web Consortium](https://es.wikipedia.org/wiki/World_Wide_Web_Consortium" \o "World Wide Web Consortium), y las herramientas de desarrollo del *software* contienen implementaciones de lenguajes de programación. El modelo de implementación es una colección de componentes y los subsistemas que contienen. Componentes tales como: ficheros ejecutables, ficheros de código fuente y todo otro tipo de ficheros que sean necesarios para la implementación y despliegue del sistema.

La etapa de implementación del diseño de *software* es el proceso de convertir una especificación del sistema en un sistema ejecutable. Siempre implica los procesos de diseño y programación de *software*, pero, si se utiliza un enfoque evolutivo de desarrollo, también puede implicar un refinamiento de la especificación del *software*. Esta etapa es una descripción de la estructura del *software* que se va a implementar, los datos que son parte del sistema, las interfaces entre los componentes del sistema, y algunas veces los algoritmos utilizados.

**Documentación**

Es todo lo concerniente a la documentación del propio desarrollo del *software* y de la gestión del proyecto, pasando por modelaciones ([UML](https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado)), diagramas de casos de uso, pruebas, manuales de usuario, manuales técnicos, etc; todo con el propósito de eventuales correcciones, usabilidad, mantenimiento futuro y ampliaciones al sistema.

**Mantenimiento**

*Artículo principal:*[Mantenimiento de software](https://es.wikipedia.org/wiki/Mantenimiento_de_software)

Fase dedicada a mantener y mejorar el *software* para corregir errores descubiertos e incorporar nuevos requisitos. Esto puede llevar más tiempo incluso que el desarrollo del *software* inicial. Alrededor de 2/3 del tiempo de ciclo de vida de un proyecto está dedicado a su mantenimiento. Una pequeña parte de este trabajo consiste eliminar errores *(bugs)*; siendo que la mayor parte reside en extender el sistema para incorporarle nuevas funcionalidades y hacer frente a su [evolución](https://en.wikipedia.org/wiki/Software_evolution).

El [mantenimiento de software](https://es.wikipedia.org/wiki/Mantenimiento_de_software), según la investigación de [Lehman](https://es.wikipedia.org/wiki/Meir_M._Lehman), el 80% del mantenimiento no son acciones correctivas. Son mejoras de funcionalidad (e incorporación de nuevos requisitos), según [Lehman](https://es.wikipedia.org/wiki/Meir_M._Lehman) el software evoluciona con el tiempo. Teniendo en cuenta eso, la etapa de mantenimiento en sí misma puede incluir las etapas anteriores entre el despliegue de cada nueva versión, reutilizando el software ya existente, reformandolo y adaptandolo.

**Ventajas**

**Desde el punto de vista de gestión**

* Facilitar la tarea de seguimiento del proyecto
* Optimizar el uso de recursos
* Facilitar la comunicación entre usuarios y desarrolladores
* Facilitar la evaluación de resultados y cumplimiento de objetivos

**Desde el punto de vista de los ingenieros de *software***

* Ayudar a comprender el problema
* Permitir la reutilización
* Facilitar el mantenimiento del producto final
* Optimizar el conjunto y cada una de las fases del proceso de desarrollo

**Desde el punto de vista de cliente o usuario final**

* Garantizar el nivel de calidad del producto final
* Obtener el ciclo de vida adecuado para el proyecto
* Confianza en los plazos del tiempo mostrados en la definición del proyecto