**El Software**

El software se refiere atodo componente intangible (y no físico) que forma parte de dispositivos como computadoras, teléfonos móviles o tabletas y que permite su funcionamiento. es un conjunto de reglas o programas que dan instrucciones a un ordenador para que realice tareas específicas. El software puede utilizarse para gestionar datos, automatizar procesos y crear aplicaciones o productos informáticos.

**Cualidades de un Software**

1. **Correctitud** (*Correctness*): Se refiere a que un programa se comporta de acuerdo a la especificación de las funciones que debería proveer. Esto implica que el software cumple con los requerimientos funcionales establecidos.
2. **Confiabilidad** (*Reliability*): Se refiere a la capacidad del software para realizar sus funciones de manera consistente y sin errores. Un software confiable es aquel que puede funcionar correctamente durante un período prolongado de tiempo y bajo diferentes condiciones.
3. **Robustez** (*Robustness*): Se refiere a la capacidad del software para manejar situaciones inesperadas o errores sin colapsar o producir resultados incorrectos. Un software robusto es capaz de recuperarse de fallos y continuar su funcionamiento normal.
4. **Eficiencia** (*Performance*): Se refiere a la eficiencia y rapidez con la que el software realiza sus funciones. Un software con buen rendimiento es aquel que responde de manera rápida y eficiente a las solicitudes del usuario.
5. **Amigabilidad** (*Friendliness*): Se refiere a la facilidad de uso y la experiencia del usuario al interactuar con el software. Un software amigable es aquel que tiene una interfaz intuitiva, fácil de entender y utilizar, y que se adapta a las necesidades y preferencias del usuario.
6. **Verificabilidad** (*Verifiability*): Se refiere a la capacidad de verificar las propiedades del software de manera fácil y precisa. Un software verificable es aquel que permite comprobar y validar sus características, como la correctitud y el rendimiento.
7. **Mantenibilidad** (*Maintainability*): Se refiere a la facilidad con la que el software puede ser modificado, corregido o mejorado. Un software mantenible es aquel que está bien estructurado, documentado y diseñado de manera que los cambios puedan realizarse de forma eficiente.
8. **Reparabilidad** (*Reparability*): Se refiere a la capacidad del software de ser reparado o corregido en caso de fallos o errores. Un software reparable es aquel que permite identificar y solucionar problemas de manera rápida y efectiva.
9. **Evolucionabilidad** (*Evolvability*): Se refiere a la capacidad del software de adaptarse y evolucionar con el tiempo. Un software evolucionable es aquel que puede ser actualizado y mejorado para satisfacer nuevas necesidades y requerimientos.
10. **Reusabilidad** (*Reusability*): Se refiere a la capacidad del software de ser utilizado en diferentes contextos o proyectos. Un software reutilizable es aquel que puede ser aprovechado en múltiples aplicaciones, lo que ahorra tiempo y recursos en el desarrollo de nuevos sistemas.
11. **Portabilidad** (*Portability*): Se refiere a la capacidad del software de ser ejecutado en diferentes plataformas o entornos. Un software portátil es aquel que puede ser instalado y utilizado en diferentes sistemas operativos o dispositivos sin necesidad de modificaciones significativas.
12. **Comprensibilidad** (*Understandability*): Se refiere a la facilidad con la que el software puede ser comprendido y entendido por los desarrolladores y usuarios. Un software comprensible es aquel que tiene una estructura clara, un código legible y una documentación adecuada.
13. **Interoperabilidad** (*Interoperability*): Se refiere a la capacidad del software de interactuar y funcionar correctamente con otros sistemas o componentes. Un software interoperable es aquel que puede intercambiar datos y comunicarse de manera efectiva con otros sistemas.
14. **Productividad** (*Productivity*): Se refiere a la capacidad del software de mejorar la eficiencia y la productividad de los usuarios. Un software productivo es aquel que automatiza tareas, simplifica procesos y facilita el trabajo de los usuarios.
15. **Oportunidad** (*Timeliness*): Se refiere a la capacidad del software de cumplir con los plazos y tiempos establecidos. Un software oportuno es aquel que se desarrolla y entrega en el momento adecuado, sin retrasos ni demoras.
16. **Visibilidad** (*Visibility*): Se refiere a la capacidad del software de proporcionar información clara y comprensible sobre su funcionamiento y estado. Un software con visibilidad permite a los usuarios y desarrolladores monitorear y comprender lo que está sucediendo en el sistema.

**Ingeniería del Software**

La **ingeniería de software** es una disciplina formada por un conjunto de métodos, herramientas y técnicas que se utilizan en el **desarrollo de los programas informáticos** ([**software**](https://definicion.de/software/)).

Esta [**disciplina**](https://definicion.de/disciplina/) trasciende la actividad de **programación**, que es el pilar fundamental a la hora de crear una aplicación. El ingeniero de software se encarga de toda la gestión del proyecto para que éste se pueda desarrollar en un plazo determinado y con el presupuesto previsto.

**Visión general del Proceso de Desarrollo de Software**

* Análisis
* Diseño
* Codificación
* Prueba
* Validación
* Instalación
* Mantenimiento

**Participantes en el Proceso de Desarrollo de Software**

* Jefe de Proyecto
* Analista de Software
* Arquitecto de Software
* Desarrollador de Software
* Programador

**Modelos de Desarrollo de Software**

Existen varios modelos de desarrollo de software, cada uno con sus propias características, enfoques y estrategias. Algunos de los modelos más comunes incluyen:

**Modelo en Cascada:** Este modelo sigue un enfoque lineal y secuencial, donde cada fase debe completarse y verificarse antes de pasar a la siguiente. Es adecuado para proyectos con requisitos bien claros y definidos y estables sin cambios de principio a fin. Las empresas que trabajan en proyectos de software a gran escala, como sistemas de gestión empresarial o aplicaciones financieras, se las ve más beneficiadas cuando optan por este enfoque.

**Modelo Iterativo e Incremental:** Este enfoque divide el desarrollo del software en ciclos cortos y repetitivos, entregando nuevas versiones del producto con cada iteración. Es ideal para proyectos donde los requisitos son propensos a cambios y se requiere flexibilidad para trabajar.

**Modelo de Prototipo**: se basa en la construcción de un prototipo de software que se construye rápidamente para que los usuarios puedan probarlo y aportar feedback. Así, se puede arreglar lo que está mal e incluir otros requerimientos que puedan surgir. Es un modelo iterativo que se basa en el método de prueba y error para comprender las especificidades del producto.

**Modelo en Espiral:** Este modelo implica una serie de ciclos repetitivos a lo largo del desarrollo del software, cada uno de los cuales se centra en mitigar los riesgos del proyecto. Su enfoque es una combinación de elementos de los modelos en cascada e iterativo, permitiendo una mayor flexibilidad y adaptabilidad a medida que avanza el ciclo de vida del proyecto. Es adecuado para proyectos de mayor complejidad, riesgo y cambios.

**Paradigmas de la Programación**

* Programación Imperativa
  + Programación Estructurada: La programación estructurada es un tipo de programación imperativa donde el flujo de control se define mediante bucles anidados, condicionales y subrutinas, en lugar de a través de GOTO.
  + Programación Por Procedimientos: Este paradigma de programación consiste en basarse en un número muy bajo de expresiones repetidas, englobarlas todas en un procedimiento o función y llamarlo cada vez que tenga que ejecutarse.
  + Programación Orientada a Objetos: En este modelo de paradigma se construyen modelos de objetos que representan elementos (objetos) del problema a resolver, que tienen características y funciones. Permite **separar los diferentes componentes de un programa**, simplificando así su creación, depuración y posteriores mejoras.
* Programación Declarativa
  + Programación Funcional: La programación funcional es un paradigma de programación centrado en el uso de funciones matemáticas puras y la inmutabilidad de datos. Este paradigma enfatiza la aplicación de funciones sin efectos secundarios, lo que significa que las funciones siempre producirán el mismo resultado para los mismos argumentos y no alterarán el estado del programa ni interactuarán con el mundo exterior.
  + Programación Lógica: La programación lógica es un paradigma de programación que se basa en la lógica matemática y la lógica proposicional. Este paradigma permite que los problemas sean formulados en términos de relaciones, representadas como hechos y reglas dentro de un programa.

**Modelado de Sistemas**

Es el proceso de desarrollo de modelos abstractos de un sistema. El modelado de sistemas es la disciplina que permite analizar estructuras complejas, mediante una representación visual.

**Tipos de Modelos y enfoques de Modelado**

* [Modelos de caso de uso](https://www.ibm.com/docs/es/SS5JSH_9.5.0/com.ibm.xtools.prodover.doc/topics/cusecasemodel.html)  
  Los modelos de caso de uso proporcionan información detallada sobre los comportamientos del sistema o la aplicación de software que se está desarrollando. Los diagramas de caso de uso describen las relaciones ente los casos de uso y los actores, y los diagramas de actividad describen el flujo de los objetos y control en cada comportamiento identificado.
* [El modelo de análisis](https://www.ibm.com/docs/es/SS5JSH_9.5.0/com.ibm.xtools.prodover.doc/topics/canalysismodel.html)  
  El modelo de análisis describe la estructura del sistema o aplicación que está modelando. Consiste en diagramas de clase y diagramas de secuencia que describen la implementación lógica de los requisitos funcionales identificados en el modelo de caso de uso.
* [El modelo de diseño](https://www.ibm.com/docs/es/SS5JSH_9.5.0/com.ibm.xtools.prodover.doc/topics/cdesignmodel.html)  
  El modelo de diseño se basa sobre el modelo de análisis describiendo, en mayor detalle, la estructura del sistema y cómo será implementado el sistema. Las clases que fueron identificadas en el modelo de análisis son refinadas para incluir las construcciones de implementación.
* [Protocolos de gestión de contrato de diseño](https://www.ibm.com/docs/es/SS5JSH_9.5.0/com.ibm.xtools.prodover.doc/topics/c_dcmp_overview.html)  
  Los protocolos de gestión de contratos de diseño o DCMP tratan de la captura y el diseño de los requisitos formales, garantizan la automatización mediante la repetición y el cumplimiento y promueven el control. Mediante diferentes protocolos de gestión de contratos de diseño se proporcionan diferentes niveles de gestión y el control del modelo y el sistema. La elección del protocolo también depende del entorno de trabajo, del tiempo y del control de la arquitectura necesario del proceso de desarrollo de software.