

# Ingeniería de software

La **ingeniería de software** (el término es discutido por cuanto el desarrollo de software no es en muchas ocasiones considerado como una ingeniería) es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software,<sup>1</sup> y el estudio de estos enfoques, es decir, el estudio de las aplicaciones de la ingeniería al software.<sup>2</sup> Integra matemáticas, ciencias de la computación y prácticas cuyos orígenes se encuentran en la ingeniería.<sup>3</sup>

Se citan las definiciones más reconocidas, formuladas por prestigiosos autores:

- Ingeniería de *software* es el estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas *software* (Zelkovitz, 1978).
- Ingeniería de *software* es la aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos. Se conoce también como desarrollo de *software* o producción de *software* (Bohem, 1976).
- La ingeniería de *software* trata del establecimiento de los principios y métodos de la ingeniería a fin de obtener *software* de modo rentable, que sea fiable y trabaje en máquinas reales (Bauer, 1972).
- La ingeniería de *software* es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación, y mantenimiento del *software*.<sup>1</sup>

En 2004, la U. S. Bureau of Labor Statistics (Oficina de Estadísticas del Trabajo de Estados Unidos) contó 760 840 ingenieros de *software* de computadora.<sup>4</sup> <sup>[*actualizar*]</sup>

El término "ingeniero de *software*", sin embargo, se utiliza de manera genérica en el ambiente empresarial, y no todos los que se desempeñan en el puesto de ingeniero de *software* poseen realmente títulos de ingeniería de universidades reconocidas.<sup>5</sup>

Algunos autores consideran que "desarrollo de *software*" es un término más apropiado que "ingeniería de *software*" para el proceso de crear *software*. Personas como Pete McBreen (autor de *Software Craftmanship*) cree que el término IS implica niveles de rigor y prueba de procesos que no son apropiados para todo tipo de desarrollo de *software*.

Indistintamente se utilizan los términos "ingeniería **de** *software*" o "ingeniería **del** *software*"; aunque menos común también se suele referenciar como "ingeniería **en** *software*".<sup>6 7 8</sup> En Hispanoamérica los términos más comúnmente usados son los dos primeros.

La creación del *software* es un proceso intrínsecamente creativo y la ingeniería del *software* trata de sistematizar este proceso con el fin de acotar el riesgo de fracaso en la consecución del objetivo, por medio de diversas técnicas que se han demostrado adecuadas sobre la base de la experiencia previa.

La ingeniería de *software* se puede considerar como la ingeniería aplicada al *software*, esto es, por medios sistematizados y con herramientas preestablecidas, la aplicación de ellos de la manera más eficiente para la obtención de resultados óptimos; objetivos que siempre busca la ingeniería. No es solo de la resolución de problemas, sino más bien teniendo en cuenta las diferentes

## Ingeniería de software

**Áreas del saber**

Software

**Campo de aplicación**

Desarrollo y mantenimiento de *software*

**Subárea de**

Ciencias de la computación



Un ingeniero de *software* (Trevor Parscal) programando en la sede de San Francisco de la Fundación Wikimedia.

soluciones, elegir la más apropiada.

La producción de *software* utiliza criterios y normas de la ingeniería de *software*, lo que permite transformarlo en un producto industrial usando bases de la ingeniería como métodos, técnicas y herramientas para desarrollar un producto innovador regido por metodologías y las buenas prácticas. Dicho producto es un medio que interviene en las funciones de sus usuarios para obtener un proceso productivo más eficaz y eficiente; hoy en día las empresas no podrían funcionar sin *software* porque este es un producto de uso masivo; por lo cual, el nivel de una empresa está determinado por la calidad de su infraestructura tecnológica y los productos desarrollados o adquiridos de acuerdo a sus necesidades.

# Índice

---

## Historia

## Objetivos

## Recursos

- Recursos humanos
- Recursos de entorno

## Implicaciones socioeconómicas

- Económicamente
- Socialmente

## Notaciones

- LUM (lenguaje unificado de modelado) o UML
- BPMN (notación para el modelado de procesos de negocios)
- Diagrama de flujo de datos (DFD)

## Herramienta CASE

## Metodología

- Etapas del proceso
  - Obtención de los requisitos
  - Análisis de requisitos
  - Limitaciones<sup>[20]</sup>
  - Especificación
  - Arquitectura
  - Programación
  - Desarrollo de la aplicación
  - Pruebas de *software*
  - Implementación
  - Documentación
  - Mantenimiento
- Ventajas<sup>[24]</sup>
  - Desde el punto de vista de gestión
  - Desde el punto de vista de los ingenieros de *software*
  - Desde el punto de vista de cliente o usuario final

## Modelos y ciclos de vida del desarrollo de *software*

- Modelo en cascada o clásico
- Modelo de prototipos
- Modelo en espiral
- Modelo de desarrollo por etapas
- Modelo incremental o iterativo
  - Modelo estructurado
  - Modelo orientado a objetos

Modelo RAD (rapid application development)

Modelo de desarrollo concurrente

Proceso unificado del desarrollo de *software*

## Producto

### Naturaleza de la ingeniería de *software*

Matemáticas

Creación

Gestión de Proyecto

### Participantes y papeles

Cliente

Desarrolladores

Gestores

Usuarios finales

Código ético de un ingeniero de *software*

### Educación ética

Organizaciones

### Véase también

### Referencias

### Bibliografía

### Enlaces externos

## Historia

---

Cuando aparecieron las primeras computadoras digitales en la década de 1940,<sup>9</sup> el desarrollo de *software* era algo tan nuevo que era casi imposible hacer predicciones de las fechas estimadas de finalización del proyecto y muchos de ellos sobrepasaban los presupuestos y tiempo estimados.. Los desarrolladores tenían que volver a escribir todos sus programas para correr en máquinas nuevas que salían cada uno o dos años, haciendo obsoletas las ya existentes.

El término ingeniería del *software* apareció por primera vez a finales de la década de 1950. La ingeniería de *software* fue estimulada por la crisis del *software* de las décadas de entre 1960 y 1980. La ingeniería del *software* viene a ayudar a identificar y corregir mediante principios y metodologías los procesos de desarrollo y mantenimiento de sistemas de *software*.

Aparte de la *crisis del *software** de las décadas de entre 1960 y 1980, la ingeniería de *software* se ve afectada por accidentes que conllevaron a la muerte de tres personas; esto sucedió cuando la máquina de radioterapia Therac-25 emite una sobredosis masiva de radiación y afecto contra la vida de estas personas.<sup>10</sup> Esto remarca los riesgos de control por *software*,<sup>11</sup> afectando directamente al nombre de la ingeniería de *software*.

A principios de los 1980,<sup>12</sup> la ingeniería del *software* ya había surgido como una genuina profesión, para estar al lado de las ciencias de la computación y la ingeniería tradicional. Antes de esto, las tareas eran corridas poniendo tarjetas perforadas como entrada en el lector de tarjetas de la máquina y se esperaban los resultados devueltos por la impresora.

Debido a la necesidad de traducir frecuentemente el *software* viejo para atender las necesidades de las nuevas máquinas, se desarrollaron lenguajes de orden superior. A medida que apareció el *software* libre, las organizaciones de usuarios comúnmente lo liberaban.

Durante mucho tiempo, solucionar la crisis del *software* fue de suma importancia para investigadores y empresas que se dedicaban a producir herramientas de *software*.

Para la década de 1980, el costo de propiedad y mantenimiento del *software* fue dos veces más caro que el propio desarrollo del *software*, y durante la década de 1990, el costo de propiedad y mantenimiento aumentó 30 % con respecto a la década anterior. En 1995, muchos de los proyectos de desarrollo estaban operacionales, pero no eran considerados exitosos. El proyecto de *software* medio sobrepasaba en un 50 % la estimación de tiempo previamente realizada, además, el 75 % de todos los grandes productos de *software* que eran entregados al cliente tenían fallas tan graves, que no eran usados en lo absoluto o simplemente no cumplían con los requerimientos del cliente.<sup>11</sup>

Algunos expertos argumentaron que la crisis del *software* era debido a la falta de disciplina de los programadores.

Cada nueva tecnología y práctica de la década de 1970 a la de 1990 fue pregonada como la única solución a todos los problemas y el caos que llevó a la crisis del *software*. Lo cierto es que la búsqueda de una única clave para el éxito nunca funcionó. El campo de la ingeniería de *software* parece un campo demasiado complejo y amplio para una única solución que sirva para mejorar la mayoría de los problemas, y cada problema representa solo una pequeña porción de todos los problemas de *software*.

El auge del uso del Internet llevó a un vertiginoso crecimiento en la demanda de sistemas internacionales de despliegue de información en la World Wide Web. Los desarrolladores se vieron en la tarea de manejar ilustraciones, mapas, fotografías y animaciones, a un ritmo nunca antes visto, con casi ningún método para optimizar la visualización y almacenamiento de imágenes. También fueron necesarios sistemas para traducir el flujo de información en múltiples idiomas extranjeros a lenguaje natural humano, con muchos sistemas de *software* diseñados para uso multilenguaje, basado en traductores humanos.

La ingeniería de *software* contribuyó alrededor de 90 000 millones de dólares por año, ya que entró en juego el Internet. Esto hace que los desarrolladores tuviesen que manejar imágenes mapas y animaciones para optimizar la visualización/almacenamiento de imágenes (como el uso de imágenes en miniatura). El uso de los navegadores y utilización de lenguaje HTML cambia drásticamente la visión y recepción de la información.

Las amplias conexiones de red causaron la proliferación de virus informáticos y basura o *spam* en los correos electrónicos (E-mail). Esta situación puso en una carrera contra el tiempo a los desarrolladores con el fin de crear nuevos sistemas de bloqueo o seguridad de dichas anomalías en la informática, ya que se volvían sumamente tediosas y difíciles de arreglar<sup>11</sup>

Después de una fuerte y creciente demanda surge la necesidad de crear soluciones de software a bajo costo, lo que conlleva al uso de metodologías más simples y rápidas que desarrollan *software* funcional. Cabe señalar que los sistemas más pequeños tenían un enfoque más simple y rápido para poder administrar el desarrollo de cálculos y algoritmos de *software*.

## Objetivos

---

La ingeniería de *software* aplica diferentes normas y métodos que permiten obtener mejores resultados, en cuanto al desarrollo y uso del *software*, mediante la aplicación correcta de estos procedimientos se puede llegar a cumplir de manera satisfactoria con los objetivos fundamentales de la ingeniería de *software*.

Entre los objetivos de la ingeniería de *software* están:

- Mejorar el diseño de aplicaciones o *software* de tal modo que se adapten de mejor manera a las necesidades de las organizaciones o finalidades para las cuales fueron creadas.
- Promover mayor calidad al desarrollar aplicaciones complejas.
- Brindar mayor exactitud en los costos de proyectos y tiempo de desarrollo de los mismos.
- Aumentar la eficiencia de los sistemas al introducir procesos que permitan medir mediante normas específicas, la calidad del *software* desarrollado, buscando siempre la mejor calidad posible según las necesidades y resultados que se quieren generar.
- Una mejor organización de equipos de trabajo, en el área de desarrollo y mantenimiento de *software*.
- Detectar a través de pruebas, posibles mejoras para un mejor funcionamiento del *software* desarrollado.<sup>13</sup>

# Recursos

---

## Recursos humanos

Son todas aquellas personas que intervienen en la planificación de cualquier instancias de *software* (por ejemplo: gestor, ingeniero de *software* experimentado, etc.), El número de personas requerido para un proyecto de *software* solo puede ser determinado después de hacer una estimación del esfuerzo de desarrollo...

## Recursos de entorno

Es el entorno de las aplicaciones (*software* y *hardware*) el *hardware* proporciona el medio físico para desarrollar las aplicaciones (*software*), este recurso es indispensable.<sup>14</sup>

# Implicaciones socioeconómicas

---

## Económicamente

En los Estados Unidos, el *software* contribuyó a una octava parte de todo el incremento del PIB durante la década de 1990 (alrededor de 90 000 millones de dólares por año), y un noveno de todo el crecimiento de productividad durante los últimos años de la década (alrededor de 33.000 millones de dólares estadounidenses por año). La ingeniería de *software* contribuyó a US\$ 1 billón de crecimiento económico y productividad en esa década. Alrededor del globo, el *software* contribuye al crecimiento económico de maneras similares, aunque es difícil de encontrar estadísticas fiables. <sup>[*cita requerida*]</sup>

Además, con la industria del lenguaje está hallando cada vez más campos de aplicación a escala global.

## Socialmente

La ingeniería de *software* cambia la cultura del mundo debido al extendido uso de la computadora. El correo electrónico (e-mail), la WWW y la mensajería instantánea permiten a la gente interactuar de nuevas maneras. El software baja el costo y mejora la calidad de los servicios de salud, los departamentos de bomberos, las dependencias gubernamentales y otros servicios sociales. Los proyectos exitosos donde se han usado métodos de ingeniería de *software* incluyen a GNU/Linux, el *software* del transbordador espacial, los cajeros automáticos y muchos otros.

# Notaciones

---

## LUM (lenguaje unificado de modelado) o UML

Es un lenguaje de modelado muy reconocido y utilizado actualmente que se utiliza para describir o especificar métodos. También es aplicable en el desarrollo de *software*.

Las siglas UML significan lenguaje unificado de modelado esto quiere decir que no pretende definir un modelo estándar de desarrollo, sino únicamente un lenguaje de modelado.<sup>15</sup>

Un lenguaje de modelado consta de vistas, elementos de modelo y un conjunto de reglas sintácticas, semánticas y pragmáticas que indican cómo utilizar los elementos.

En esta materia nos encontramos con varios diagramas que se pueden usar tales como: casos de uso, de clases, componentes, despliegue, etc.

## BPMN (notación para el modelado de procesos de negocios)

El objetivo de la notación para el modelado de procesos de negocios es proporcionar de una manera fácil de definir y analizar los procesos de negocios públicos y privados simulando un diagrama de flujo. La notación ha sido diseñada específicamente para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes del mismo, con un conjunto de actividades relacionadas. Características básicas de los elementos de BPMN

- Objetos de flujo: eventos, actividades, rombos de control de flujo (gateways).
- Objetos de conexión: flujo de secuencia, flujo de mensaje, asociación.
- Swimlanes (carriles de piscina): pool, lane.
- Artefactos: objetos de datos, grupo, anotación.<sup>15</sup>

## Diagrama de flujo de datos (DFD)

Un diagrama de flujo de datos permite representar el movimiento de datos a través de un sistema por medio de modelos que describen los flujos de datos, los procesos que transforman o cambian los datos, los destinos de datos y los almacenamientos de datos a la cual tiene acceso el sistema.

Su inventor fue Larry Constantine, basado en el modelo de computación de Martin y Estrin: flujo gráfico de datos. Con los diagramas de flujo de datos determina la manera en que cualquier sistema puede desarrollarse, ayuda en la identificación de los datos de la transacción en el modelo de datos y proporciona al usuario una idea física de cómo resultarán los datos a última instancia.<sup>16</sup>

## Herramienta CASE

---

Las Herramienta CASE son herramientas computacionales (*software*) que están destinadas a asistir en los procesos de ciclo de vida de un *software*, facilitan la producción del *software*, varias se basan principalmente en la idea de un modelo gráfico.<sup>17</sup>

## Metodología

---

Un objetivo de décadas ha sido el encontrar procesos y metodologías, 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. 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 salen 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.<sup>18</sup>

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.<sup>19</sup>

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. Asimismo, se define un diagrama de entidad/relación, 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.

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.<sup>19</sup>

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

## Limitaciones<sup>20</sup>

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
- 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 lo 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
- Diagrama de base de datos
- Diagrama de despliegue
- 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 (*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 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, 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.<sup>21</sup>

## 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), y luego probarlo de manera integral (pruebas de integración), 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, 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 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.<sup>22</sup>

## **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, 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.<sup>23</sup>

## **Documentación**

Es todo lo concerniente a la documentación del propio desarrollo del *software* y de la gestión del proyecto, pasando por modelaciones (UML), 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**

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<sup>14</sup> 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.

## **Ventajas<sup>24</sup>**

### **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

## Modelos y ciclos de vida del desarrollo de *software*

---

La ingeniería de *software*, con el fin de ordenar el caos que era anteriormente el desarrollo de *software*, dispone de varios modelos, paradigmas y filosofías de desarrollo, estos los conocemos principalmente como modelos o ciclos de vida del desarrollo de *software*, esto incluye el proceso que se sigue para construir, entregar y hacer evolucionar el *software*, desde la concepción de una idea hasta la entrega y el retiro del sistema y representa todas las actividades y artefactos (productos intermedios) necesarios para desarrollar una aplicación.<sup>25</sup>

El ciclo de vida de un *software* contiene los siguientes procedimientos:

- **Definición de objetivos:** definir el resultado del proyecto y su papel en la estrategia global.<sup>26</sup>
- **Análisis de los requisitos y su viabilidad:** recopilar, examinar y formular los requisitos del cliente y examinar cualquier restricción que se pueda aplicar.<sup>26</sup>
- **Diseño general:** requisitos generales de la arquitectura de la aplicación.<sup>26</sup>
- **Diseño en detalle:** definición precisa de cada subconjunto de la aplicación.<sup>26</sup>
- **Programación** (programación e implementación): es la implementación de un lenguaje de programación para crear las funciones definidas durante la etapa de diseño.<sup>26</sup>
- **Prueba de unidad:** prueba individual de cada subconjunto de la aplicación para garantizar que se implementaron de acuerdo con las especificaciones.<sup>26</sup>
- **Integración:** para garantizar que los diferentes módulos se integren con la aplicación. Este es el propósito de la *prueba de integración* que está cuidadosamente documentada.<sup>26</sup>
- **Prueba beta** (o *validación*), para garantizar que el *software* cumple con las especificaciones originales.<sup>26</sup>
- **Documentación:** sirve para documentar información necesaria para los usuarios del *software* y para desarrollos futuros.<sup>26</sup>
- **Implementación**
- **Mantenimiento:** para todos los procedimientos correctivos (mantenimiento correctivo) y las actualizaciones secundarias del *software* (mantenimiento continuo).<sup>26</sup>

### Modelo en cascada o clásico

En ingeniería de *software* el modelo en cascada —también llamado desarrollo en cascada o ciclo de vida clásico— se basa en un enfoque metodológico que ordena rigurosamente las etapas del ciclo de vida del *software*, esto sugiere una aproximación sistemática secuencial hacia el proceso de desarrollo del *software*, que se inicia con la especificación de requisitos del cliente y continúa con la planificación, el modelado, la construcción y el despliegue para culminar en el soporte del *software* terminado.<sup>27</sup>

### Modelo de prototipos

En ingeniería de *software*, el modelo de prototipos pertenece a los modelos de desarrollo evolutivo. Este permite que todo el sistema, o algunos de sus partes, se construyan rápidamente para comprender con facilidad y aclarar ciertos aspectos en los que se aseguren que el desarrollador, el usuario, el cliente estén de acuerdo en lo que se necesita así como también la solución que se propone para dicha necesidad y de esta manera minimizar el riesgo y la incertidumbre en el desarrollo, este modelo se encarga del desarrollo de diseños para que estos sean analizados y prescindir de ellos a medida que se adhieran nuevas especificaciones, es ideal para medir el alcance del producto, pero no se asegura su uso real.

Este modelo principalmente se aplica cuando un cliente define un conjunto de objetivos generales para el *software* a desarrollarse sin delimitar detalladamente los requisitos de entrada procesamiento y salida, es decir cuando el responsable no está seguro de la eficacia de un algoritmo, de la adaptabilidad del sistema o de la manera en que interactúa el hombre y la máquina.

Este modelo se encarga principalmente de ayudar al ingeniero de sistemas y al cliente a entender de mejor manera cuál será el resultado de la construcción cuando los requisitos estén satisfechos.<sup>28</sup>

## Modelo en espiral

El modelo en espiral, que Barry Boehm propuso originalmente en 1986, es un modelo de proceso de *software* evolutivo que conjuga la naturaleza iterativa de la construcción de prototipos con los aspectos controlados y sistemáticos del modelo en cascada, es decir, cuando se aplica este modelo, el *software* se desarrolla en una serie de entregas evolutivas (ciclos o iteraciones), cada una de estas entregando prototipos más completas que el anterior, todo esto en función del análisis de riesgo y las necesidades del cliente. Aunque el modelo espiral representa ventajas por sobre el desarrollo lineal, el cálculo de los riesgos puede ser muy complicado y por lo cual su uso en el ámbito real es muy escaso.<sup>29</sup>

## Modelo de desarrollo por etapas

Es un modelo en el que el *software* se muestra al cliente en etapas refinadas sucesivamente. Con esta metodología se desarrollan las capacidades más importantes reduciendo el tiempo necesario para la construcción de un producto; el modelo de entrega por etapas es útil para el desarrollo de la herramienta debido a que su uso se recomienda para problemas que pueden ser tratados descomponiéndolos en problemas más pequeños y se caracteriza principalmente en que las especificaciones no son conocidas en detalle al inicio del proyecto y por tanto se van desarrollando simultáneamente con las diferentes versiones del código.

En este modelo pueden distinguirse las siguientes fases:

- Especificación conceptual.
- Análisis de requisitos.
- Diseño inicial.
- Diseño detallado (codificación, depuración, prueba y liberación).

Cuando es por etapas, en el diseño global estas fases pueden repetirse según la cantidad de etapas que sean requeridas.

Entre sus ventajas tenemos:

- Detección de problemas antes y no hasta la única entrega final del proyecto.
- Eliminación del tiempo en informes debido a que cada versión es un avance.
- Estimación de tiempo por versión, evitando errores en la estimación del proyecto general.
- Cumplimiento a la fecha por los desarrolladores.

## Modelo incremental o iterativo

Desarrollo iterativo y creciente (o incremental) es un proceso de desarrollo de *software*, creado en respuesta a las debilidades del modelo tradicional de cascada, es decir, este modelo aplica secuencias lineales como el modelo en cascada, pero de una manera iterativa o escalada según como avance el proceso de desarrollo y con cada una de estas secuencias lineales se producen incrementos (mejoras) del *software*.<sup>30</sup>

Se debe tener en cuenta que el flujo del proceso de cualquier incremento puede incorporar el paradigma de construcción de prototipos, ya que como se mencionó anteriormente, este tipo de modelo es iterativo por naturaleza, sin embargo se diferencia en que este busca la entrega de un producto operacional con cada incremento que se le realice al *software*.

Este desarrollo incremental es útil principalmente cuando el personal necesario para una implementación completa no está disponible.

## Modelo estructurado

Este modelo —como su nombre lo indica— utiliza las técnicas del diseño estructurado o de la programación estructurada para su desarrollo, también se utiliza en la creación de los algoritmos del programa. Este formato facilita la comprensión de la estructura de datos y su control.<sup>31</sup> Entre las principales características de este modelo se encuentran las siguientes:

- Generalmente se puede diferenciar de una manera más clara los procesos y las estructuras de datos.
- Existen métodos que se enfocan principalmente en ciertos datos.
- La abstracción del programa es de un nivel mucho mayor.
- Los procesos y estructuras de datos son representados jerárquicamente.<sup>31</sup>

Este modelo también presenta sus desventajas entre las cuales podemos mencionar algunas:

- Se podía encontrar datos repetidos en diferentes partes del programa.<sup>31</sup>
- Cuando el código se hace muy extenso o grande su manejo se complica demasiado.<sup>32</sup>

En el modelo estructurado las técnicas que comúnmente se utilizan son:

- El modelo entidad-relación, esta técnica se relaciona principalmente con los datos.
- El diagrama de flujo de datos, esta es utilizada principalmente para los procesos.<sup>33</sup>

## Modelo orientado a objetos

Estos modelos tienen sus raíces en la programación orientada a objetos y como consecuencia de ella gira entorno al concepto de clase, también lo hacen el análisis de requisitos y el diseño. Esto además de introducir nuevas técnicas, también aprovecha las técnicas y conceptos del desarrollo estructurado, como diagramas de estado y transiciones. El modelo orientado a objetos tiene dos características principales, las cuales ha favorecido su expansión:

- Permite la reutilización de *software* en un grado significativo.
- Su simplicidad facilita el desarrollo de herramientas informáticas de ayuda al desarrollo, el cual es fácilmente implementada en una notación orientada a objetos llamado UML.<sup>34</sup>

## Modelo RAD (rapid application development)

El RAD (*rapid application development*: ‘desarrollo rápido de aplicaciones’), es un modelo de proceso de *software* incremental, desarrollado inicialmente por James Maslow en 1980, que resalta principalmente un ciclo corto de desarrollo.

Esta es una metodología que posibilita la construcción de sistemas computacionales que combinen técnicas y utilidades CASE (Computer Aided Software Engineering), la construcción de prototipos centrados en el usuario y el seguimiento lineal y sistemático de objetivos, incrementando la rapidez con la que se producen los sistemas mediante la utilización de un enfoque de desarrollo basado en componentes.<sup>35</sup>

Si se entienden bien los requisitos y se limita el ámbito del proyecto, el proceso RAD permite que un equipo de desarrollo cree un producto completamente funcional dentro de un periodo muy limitado de tiempo sin reducir en lo más mínimo la calidad del mismo.<sup>36</sup>

## Modelo de desarrollo concurrente

El modelo de desarrollo concurrente es un modelo de tipo de red donde todas las personas actúan simultáneamente o al mismo tiempo. Este tipo de modelo se puede representar a manera de esquema como una serie de actividades técnicas importantes, tareas y estados asociados a ellas.

El modelo de proceso concurrente define una serie de acontecimientos que dispararan transiciones de estado a estado para cada una de las actividades de la ingeniería del *software*. Por ejemplo, durante las primeras etapas del diseño, no se contempla una inconsistencia del modelo de análisis. Esto genera la corrección del modelo de análisis de sucesos, que disparara la actividad de análisis del estado hecho al estado cambios en espera. Este modelo de desarrollo se utiliza a menudo como el paradigma de desarrollo de aplicaciones cliente/servidor. Un sistema cliente/servidor se compone de un conjunto de componentes funcionales. Cuando se aplica a cliente/servidor, el modelo de proceso concurrente define actividades en dos dimensiones: una división de sistemas y una división de componentes. Los aspectos del nivel de sistemas se afrontan mediante dos actividades: diseño y realización.

La concurrencia se logra de dos maneras:

- Las actividades del sistema y de componente ocurren simultáneamente y pueden modelarse con el enfoque orientado a objetos descrito anteriormente;
- Una aplicación cliente/servidor típica se implementa con muchos componentes, cada uno de los cuales se pueden diseñar y realizar concurrentemente.

En realidad, el modelo de desarrollo concurrente es aplicable a todo tipo de desarrollo de *software* y proporciona una imagen exacta del estado actual de un proyecto. En vez de confinar actividades de ingeniería de *software* a una secuencia de sucesos, define una red de actividades, todas las actividades de la red existen simultáneamente con otras. Los sucesos generados dentro de una actividad dada o algún otro lado de la red de actividad inicia las transiciones entre los estados de una actividad.

## Proceso unificado del desarrollo de *software*

El proceso unificado es un proceso de *software* genérico que puede ser utilizado para una gran cantidad de tipos de sistemas de *software*, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de competencia y diferentes tamaños de proyectos.

Provee un enfoque disciplinado en la asignación de tareas y responsabilidades dentro de una organización de desarrollo. Su meta es asegurar la producción de *software* de muy alta calidad que satisfaga las necesidades de los usuarios finales, dentro de un calendario y presupuesto predecible.<sup>37</sup>

El proceso unificado tiene dos dimensiones:

- Un eje horizontal que representa el tiempo y muestra los aspectos del ciclo de vida del proceso a lo largo de su desenvolvimiento
- Un eje vertical que representa las disciplinas, las cuales agrupan actividades de una manera lógica de acuerdo a su naturaleza.

La primera dimensión representa el aspecto dinámico del proceso conforme se va desarrollando, se expresa en términos de fases, iteraciones e hitos (milestones).

La segunda dimensión representa el aspecto estático del proceso: cómo es descrito en términos de componentes del proceso, disciplinas, actividades, flujos de trabajo, artefactos y roles.

El refinamiento más conocido y documentado del proceso unificado es el RUP (proceso unificado racional).

El proceso unificado no es simplemente un proceso, sino un marco de trabajo extensible que puede ser adaptado a organizaciones o proyectos específicos. De la misma manera, el proceso unificado de racional, también es un marco de trabajo extensible, por lo que muchas veces resulta imposible decir si un refinamiento particular del proceso ha sido derivado del proceso unificado o del RUP. Por dicho motivo, los dos nombres suelen utilizarse para referirse a un mismo concepto.<sup>38</sup>

# Producto

---

El *software* se ha convertido en algo muy necesario en nuestra sociedad actual, es la máquina que conduce a la toma de decisiones comerciales, sirve para la investigación científica moderna, es un factor clave que diferencia productos y servicios modernos. Esto se da porque el *software* está inmerso en sistemas de todo tipo alrededor de nosotros.

El *software* de computadora es el producto que diseñan y construyen los ingenieros de *software*. Esto abarca programas que se ejecutan dentro de una computadora de cualquier tamaño y arquitectura, después de estar construido casi cualquier persona en el mundo industrializado, ya sea directa o indirectamente.

Los productos se pueden clasificar en:

- Productos genéricos: Son los producidos por una organización para ser vendidos al mercado.
- Productos hechos a medida: Sistemas que son desarrollados bajo pedido a un desarrollador específico.

Estos productos deben cumplir varias características al ser entregados, estas son:

- Mantenibles: El *software* debe poder evolucionar mientras cumple con sus funciones.
- Confiabilidad: No debe producir daños en caso de errores.
- Eficiencia: El *software* no debe desperdiciar los recursos.
- Utilización adecuada: Debe contar con una interfaz de usuario adecuada y su documentación.

Lo que constituye el producto final es diferente para el ingeniero y los usuarios, para el ingeniero son los programas, datos y documentos que configuran el *software* pero para el usuario el producto final es la información que de cierto modo soluciona el problema planteado por el usuario.

## Naturaleza de la ingeniería de *software*

---

La ingeniería de *software* es una disciplina que está orientada a aplicar conceptos y métodos de ingeniería al desarrollo de *software* de calidad.

### Matemáticas

Los programas tienen muchas propiedades matemáticas. Por ejemplo la corrección y la complejidad de muchos algoritmos son conceptos matemáticos que pueden ser rigurosamente probados. El uso de matemáticas en la IS es llamado *métodos formales*.

### Creación

Los programas son construidos en una secuencia de pasos. El hecho de definir propiamente y llevar a cabo estos pasos, como en una línea de ensamblaje, es necesario para mejorar la productividad de los desarrolladores y la calidad final de los programas. Este punto de vista inspira los diferentes procesos y metodologías que se encuentran en la IS.

### Gestión de Proyecto

El desarrollo de *software* de gran porte requiere una adecuada gestión del proyecto. Hay presupuestos, establecimiento de tiempos de entrega, un equipo de profesionales que liderar. Recursos (espacio de oficina, insumos, equipamiento) por adquirir. Para su administración se debe tener una clara visión y capacitación en gestión de proyectos.

## Participantes y papeles

---

Para el desarrollo de un sistema de *software* es necesaria la colaboración de muchas personas con diversas competencias, capacidades e intereses. Al conjunto de personas involucradas en el proyecto se les conoce como participantes.

Al conjunto de funciones y responsabilidades que hay dentro del proyecto o sistema se le conoce como roles o papeles. Los roles están asociados a las tareas que son asignadas a los participantes, en consecuencia, una persona puede desempeñar uno o múltiples roles, así también un mismo rol puede ser representado por un equipo.<sup>39</sup>

## Cliente

Es frecuente el uso de los términos "usuarios", "usuarios finales" y "clientes" como sinónimos, lo cual puede provocar confusión; estrictamente, el cliente (persona, empresa u organización) es quién especifica los requisitos del sistema,<sup>40</sup> en tanto que el usuario es quien utiliza u opera finalmente el producto *software*, pudiendo ser o no el cliente.

## Desarrolladores

Esta clase de participantes están relacionados con todas las facetas del proceso de desarrollo del *software*. Su trabajo incluye la investigación, diseño, implementación, pruebas y depuración del *software*.<sup>41</sup>

## Gestores

En el contexto de ingeniería de *software*, el gestor de desarrollo de *software* es un participante, que reporta al director ejecutivo de la empresa que presta el servicio de desarrollo. Es responsable del manejo y coordinación de los recursos y procesos para la correcta entrega de productos de *software*, mientras participa en la definición de la estrategia para el equipo de desarrolladores, dando iniciativas que promuevan la visión de la empresa.<sup>42</sup>

## Usuarios finales

El usuario final es quien interactúa con el producto de *software* una vez es entregado.<sup>40</sup> Generalmente son los usuarios los que conocen el problema, ya que día a día operan los sistemas.

## Código ético de un ingeniero de *software*

Un ingeniero de *software* debe tener un código donde asegura, en la medida posible, que los esfuerzos realizados se utilizarán para realizar el bien y deben comprometerse para que la ingeniería de *software* sea una profesión benéfica y respetada. Para el cumplimiento de esta norma, se toman en cuenta ocho principios relacionados con la conducta y las decisiones tomadas por el ingeniero; donde estos principios identifican las relaciones éticamente responsables de los individuos, grupos y organizaciones donde participen. Los principios a los que deben sujetarse son sobre la sociedad, cliente y empresario, producto, juicio, administración, profesión, colegas y por último el personal.

- **Sociedad:** Los ingenieros de *software* deben actuar de manera congruente con el interés social, aceptando la responsabilidad total de su trabajo, moderando los intereses con el bienestar social, aprobando el *software* solamente si se tiene una creencia bien fundamentada, cooperando en los esfuerzos para solucionar asuntos importantes de interés social, ser justo y veraz en todas las afirmaciones relativas al *software* o documentos asociados.
- **Cliente y empresario:** Se debe actuar de manera tal que se llegue a conciliar los mejores intereses de los clientes y empresarios, congruentemente con el interés social. Estos deberán prestar servicios en sus áreas de competencia, siendo honestos y francos sobre las limitaciones, no utilizar un *software* que se obtenga ilegalmente o sin ética, usar la propiedad de los clientes o empresarios de manera autorizada, mantener secreto cualquier documento de información confidencial.
- **Producto:** Hay que asegurarse que los productos y sus modificaciones cumplan con los estándares profesionales más altos posibles, procurando la alta calidad, costos aceptables y una agenda razonable



asegurando que los costos y beneficios sean claros y aceptados por el empresario y el cliente. Asegurar que las metas y objetivos de cualquier proyecto sean adecuados y alcanzables.

- **Juicio:** Se debe mantener una integridad e independencia en el juicio profesional, moderando todo juicio técnico por la necesidad de apoyar y mantener los valores humanos, mantener la objetividad profesional con respecto a cualquier *software* o documento relacionado, no involucrarse en prácticas financieras fraudulentas.
- **Administración:** Se deberá asegurar una buena administración para cualquier proyecto en el cual se trabaje, utilizando procedimientos efectivos para promover la calidad y reducir riesgos, asegurándose también que se conozcan las políticas y procedimientos del empresario para proteger contraseñas, archivos e información confidencial.
- **Profesión:** Se debe incrementar la integridad y reputación de la profesión en conjunto con el interés social, ayudando al desarrollo de un ambiente organizacional favorable para actuar, promoviendo el conocimiento público de la ingeniería de *software*, extendiendo el conocimiento de la ingeniería de *software* por medio de participaciones en organizaciones, reuniones y publicaciones profesionales.
- **Colegas:** Cada ingeniero deberá apoyar y ser justos con los colegas, motivando a sus colegas sujetándose al código, ayudando también a su desarrollo profesional, reconocer los trabajos de otros y abstenerse a atribuirse de méritos indebidos, revisar los trabajos de manera objetiva, sincera y propiamente documentada.
- **Personal:** Los ingenieros de *software* participaran toda su vida en el aprendizaje con la práctica y promoverán un enfoque ético de la profesión, mejorando su conocimiento de los avances en el análisis, especificación, diseño, desarrollo, mantenimiento, pruebas del *software* y documentos relacionados en conjunto con administración del proceso de desarrollo.<sup>43</sup>

## Educación ética

---

### Organizaciones

- [IEEE Computer Society](#)
- [Association for Computing Machinery \(ACM\)](#).
- [Software Engineering Institute \(SEI\)](#).
- [British Computer Society \(BCS\)](#).
- [RUSOFT Association](#)
- [Society of Software Engineers \(SES\)](#). (<http://www.iacsit.org/show-49-305-1.html>)
- [Consejo General de Colegios Oficiales de Ingeniería Técnica en Informática \(CONCITI\)](#)

### Véase también

---

- [Anexo:Filosofías del desarrollo de software](#)
- [Ingeniería de sistemas](#)
- [Ingeniería informática](#)
- [Gestión de la configuración](#)
- [Proceso para el desarrollo de software](#)
- [Mantenimiento de software](#)
- [Fragilidad del software](#)
- [Error de software](#)
- [Usabilidad](#)
- [MÉTRICA](#)
- [Historia de la ingeniería del software](#)
- [Crisis del software](#)
- [No hay balas de plata](#)

## Referencias

---

1. "IEEE Standard Glossary of Software Engineering Terminology," *IEEE std 610.12-1990*, 1990. ISBN 155937067X.

2. SWEBOK executive editors, Alain Abran, James W. Moore; editors, Pierre Bourque, Robert Dupuis. (2004). Pierre Bourque and Robert Dupuis, ed. *Guide to the Software Engineering Body of Knowledge - 2004 Version* (<http://www.swebok.org>). IEEE Computer Society. pp. 1-1. ISBN 0-7695-2330-7.
3. ACM (2006). «Computing Degrees & Careers» ([https://web.archive.org/web/20110617053818/http://computingcareers.acm.org/?page\\_id=12](https://web.archive.org/web/20110617053818/http://computingcareers.acm.org/?page_id=12)). ACM. Archivado desde el original ([http://computingcareers.acm.org/?page\\_id=12](http://computingcareers.acm.org/?page_id=12)) el 17 de junio de 2011. Consultado el 23 de noviembre de 2010.
4. Bureau of Labor Statistics, U.S. Department of Labor, *USDL 05-2145: Occupational Employment and Wages, November 2004* (<ftp://ftp.bls.gov/pub/news.release/ocwage.txt>) (enlace roto disponible en Internet Archive; véase el historial ([https://web.archive.org/web/\\*ftp://ftp.bls.gov/pub/news.release/ocwage.txt](https://web.archive.org/web/*ftp://ftp.bls.gov/pub/news.release/ocwage.txt)) y la última versión (<https://web.archive.org/web/2/ftp://ftp.bls.gov/pub/news.release/ocwage.txt>))., Table 1.
5. «La ciencia de hoy es la tecnología del mañana» (<https://desarrolloprogram.wordpress.com/ingenieria-de-software/>).
6. Universidad Siglo XXI, Argentina (<http://www.21.edu.ar/carreras/ingenieria-en-software/>)
7. «Universidad Autónoma de Guadalajara, México» (<https://web.archive.org/web/20140325030707/http://www.uag.mx/licenciatura/en-software/#>). Archivado desde el original (<http://www.uag.mx/licenciatura/en-software/#>) el 25 de marzo de 2014. Consultado el 11 de marzo de 2014.
8. Tecnológico de Antioquía, Colombia ([http://www.tdea.edu.co/index.php?option=com\\_content&view=article&id=174](http://www.tdea.edu.co/index.php?option=com_content&view=article&id=174)) (enlace roto disponible en Internet Archive; véase el historial ([https://web.archive.org/web/\\*http://www.tdea.edu.co/index.php?option=com\\_content&view=article&id=174](https://web.archive.org/web/*http://www.tdea.edu.co/index.php?option=com_content&view=article&id=174)) y la última versión ([https://web.archive.org/web/2/http://www.tdea.edu.co/index.php?option=com\\_content&view=article&id=174](https://web.archive.org/web/2/http://www.tdea.edu.co/index.php?option=com_content&view=article&id=174))).
9. Leondes (2002). *intelligent systems: technology and applications*. CRC Press. ISBN 978-0-8493-1121-5.
10. «An Investigation of Therac-25 Accidents» ([https://web.archive.org/web/20140502053916/http://courses.cs.vt.edu/professionalism/Therac\\_25/Therac\\_1.html](https://web.archive.org/web/20140502053916/http://courses.cs.vt.edu/professionalism/Therac_25/Therac_1.html)). Archivado desde el original ([http://courses.cs.vt.edu/professionalism/Therac\\_25/Therac\\_1.html](http://courses.cs.vt.edu/professionalism/Therac_25/Therac_1.html)) el 2 de mayo de 2014. Consultado el 11 de abril de 2014.
11. Association for Computing Machinery. «The Risks Digest» (<https://catless.ncl.ac.uk/Risks/>) (en inglés). Consultado el 10 de noviembre de 2018.
12. Sommerville, Ian (1985) [1982]. *Software Engineering* (en inglés). Addison-Wesley. ISBN 0-201-14229-5. «Software engineering... has recently emerged as a discipline in its own right.»
13. Universidad Politécnica de Madrid. «Objetivos de ingeniería del software» (<http://www.eui.upm.es/estudios/grados/software/objetivos>).
14. Pressman, Roger S. (2003). «El proceso». *Ingeniería del software, un enfoque práctico*. México: Mc Graw Hill, quinta edición.
15. Monografias.com. «Ingeniería de Software UML» (<http://www.monografias.com/trabajos5/insof/insof.shtml>). Consultado el 10 de noviembre de 2018.
16. Notaciones de Ingeniería de Software (<http://infoblog-ingsoftware.blogspot.com/2010/11/las-tres-notaciones-son-y-uml-lenguaje.html>).
17. Monografias.com Ingeniería del software (<http://www.monografias.com/trabajos5/inso/inso.shtml>)
18. [1] (<http://yaqui.mxl.uabc.mx/~molguin/as/IngReq.htm>)
19. [2] (<http://www.slideshare.net/marfonline/analisis-de-requerimientos-ingenieria-de-software>)
20. «Limitaciones de los software» (<http://mundokramer.wordpress.com/2011/05/19/requerimientos-caracteristicas-y-limitaciones-del-software/>), artículo en el sitio web *Mundo Kramer*.
21. «Unidad 2: Fundamentos de la ingeniería del software» (<http://ingsoftware072301.obolog.es/unidad-2-fundamentos-ingenieria-software-2006544>), artículo en el sitio web Ing Software.
22. [3] ([http://catarina.udlap.mx/u\\_dl\\_a/tales/documentos/lis/fuentes\\_k\\_jf/capitulo2.pdf](http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/fuentes_k_jf/capitulo2.pdf))
23. Sommerville, Ian. *Ingeniería de Software*.
24. Metodología de ingeniería de software (<http://www.slideshare.net/vdaniel20/metodologas-y-ciclos-de-vida>), artículo en el sitio web Slide Share.
25. Ingeniería de software: ciclos de vida y metodologías (<https://web.archive.org/web/20131203234610/http://sistemas.uniandes.edu.co/~isis2603/dokuwiki/lib/exe/fetch.php?media=principal:isis2603-modelosciclosdevida.pdf>), artículo publicado en el sitio web de la Facultad de Ingeniería de la Universidad de Los Andes.
26. «Ciclo de vida del software» (<http://es.ccm.net/contents/223-ciclo-de-vida-del-software>). Consultado el 14/12/16.
27. Pressman, Roger S.: *Ingeniería del software: un enfoque práctico*. Sexta edición, pág. 50-51.
28. Lawrence Peleeeger, Shari: *Ingeniería de software: modelo de prototipos*. Universidad Estatal de Milagro.
29. Pressman, Roger S.: *Ingeniería del software: un enfoque práctico*. Sexta edición, pág. 58-60.
30. Pressman, Roger S.: *Ingeniería del software: un enfoque práctico*. Sexta edición, pág. 52-53.
31. Diseño estructurado (<http://www.slideshare.net/waralivt/desarrollo-estructurado-15323143>), artículo en el sitio web Slide Share.

32. [4] (<http://cuadrocomparativodeprogramacion.blogspot.com/2012/11/cuadro-comparativo-de-programacion.html>), cuadro comparativo de programación estructurada y programación orientada objeto .
33. [5] ([http://books.google.com.pa/books?id=\\_tKTpr4Ah88C&printsec=frontcover&dq=ingenieria+de+software&hl=es&sa=X&ei=g3k8U6KDDKe0sQS8poDIAQ&redir\\_esc=y#v=onepage&q=ingenieria%20de%20software&f=false](http://books.google.com.pa/books?id=_tKTpr4Ah88C&printsec=frontcover&dq=ingenieria+de+software&hl=es&sa=X&ei=g3k8U6KDDKe0sQS8poDIAQ&redir_esc=y#v=onepage&q=ingenieria%20de%20software&f=false)), Benet Campderrich Falgueras, Editorial UOC, 2002 - 320 páginas.
34. Campderrich Falgueras, Benet (2002): *Ingeniería de software*. ([http://books.google.com.pa/books?id=\\_tKTpr4Ah88C&printsec=frontcover&dq=ingenieria+de+software&hl=es&sa=X&ei=g3k8U6KDDKe0sQS8poDIAQ&redir\\_esc=y#v=onepage&q=ingenieria%20de%20software&f=false](http://books.google.com.pa/books?id=_tKTpr4Ah88C&printsec=frontcover&dq=ingenieria+de+software&hl=es&sa=X&ei=g3k8U6KDDKe0sQS8poDIAQ&redir_esc=y#v=onepage&q=ingenieria%20de%20software&f=false)) Barcelona: Editorial UOC, 2002. 320 páginas.
35. [6] ([http://www.casemaker.com/download/products/totem/rad\\_wp.pdf](http://www.casemaker.com/download/products/totem/rad_wp.pdf))
- Archivado ([https://web.archive.org/web/20140824021938/http://www.casemaker.com/download/products/totem/rad\\_wp.pdf#](https://web.archive.org/web/20140824021938/http://www.casemaker.com/download/products/totem/rad_wp.pdf#)) el 24 de agosto de 2014 en la Wayback Machine., What is Rapid Application Development?
36. Pressman, Roger S.: *Ingeniería del software: un enfoque práctico*. Sexta edición, pág. 53-54.
37. «Proceso unificado del desarrollo de software» (<http://yaqui.mxl.uabc.mx/~molguin/as/RUP.htm>), artículo en el sitio web Yaqui.
38. Pressman, Roger S.: *Ingeniería del software: un enfoque práctico*. Sexta edición, pág. 67-72.
39. Bernd Bruegge & Allen H.Dutoit. Object-Oriented Software Engineering, Prentice Hall, Pag. 11.
40. Pressman, 2002, p. 39
41. «O\*NET Code Connector - Software Developers, Systems Software - 15-1133.00» (<https://www.onetcodeconnector.org/ccreport/15-1133.00>). Onetcodeconnector.org. Consultado el 4 de agosto de 2014.
42. «Software Development Manager Position Description» (<https://www.interfacing.com/bpm-jobs>). interfacing.com. Consultado el 4 de agosto de 2014.
43. «Ingeniería de Software Código de Ética y Práctica Profesional» (<http://seeri.etsu.edu/Codes/SpanishVersionSECode.htm>). SEERI, East Tennessee State University. 1999.


## Bibliografía

---

- *Ingeniería de software* (sexta edición), Ian Sommerville. Addison Wesley. Sitio en inglés (<https://web.archive.org/web/20170718055633/http://www.booksites.net/>)
- Pressman, Roger S.: *Ingeniería del software: un enfoque práctico* (información en inglés). ([http://highered.mcgraw-hill.com/sites/0072853182/information\\_center\\_view0/](http://highered.mcgraw-hill.com/sites/0072853182/information_center_view0/)) McGraw Hill Higher Education, sexta edición, pág. 50-51.

## Enlaces externos

---

-  Wikiversidad alberga proyectos de aprendizaje sobre **Ingeniería de software**.
- «Is software engineering actually engineering?» ([http://iwarrior.uwaterloo.ca/?module=displaystory&story\\_id=1051&format=html&edition\\_id=1](http://iwarrior.uwaterloo.ca/?module=displaystory&story_id=1051&format=html&edition_id=1)), artículo publicado en *The Iron Warrior*, publicación de la University of Waterloo Engineering Society.

---

Obtenido de «[https://es.wikipedia.org/w/index.php?title=Ingeniería\\_de\\_software&oldid=118749454](https://es.wikipedia.org/w/index.php?title=Ingeniería_de_software&oldid=118749454)»

---

**Esta página se editó por última vez el 31 ago 2019 a las 05:23.**

El texto está disponible bajo la Licencia Creative Commons Atribución Compartir Igual 3.0; pueden aplicarse cláusulas adicionales. Al usar este sitio, usted acepta nuestros [términos de uso](#) y nuestra [política de privacidad](#). Wikipedia® es una marca registrada de la [Fundación Wikimedia, Inc.](#), una organización sin ánimo de lucro.