CURSO 2022-2023
CICLO SUPERIOR DE DESARROLLO DE APLICACIONES WEB
IES AGUADULCE

Entornos de Desarrollo

Francisco Javier Sueza Rodríguez

24 de octubre de 2022

Índice general

Índice de figuras

Tema 1

Desarrollo de Software

En esta unidad vamos a realizar una introducción al concepto de software, así como a los diferentes tipos de software que podemos encontrar y al proceso de desarrollo de éste. También hablaremos de los lenguajes de programación, en que consiste y que características tiene. Por último, veremos que son los entornos de desarrollo, cuales su función y su evolución histórica.

1.1 Software y Tipos de Software

Un ordenador se compone de dos partes bien diferenciadas, el **hardware** y el **software**.

El **hardware**, **equipo** o **soporte físico** en informática se refiere a las partes físicas, tangibles de un sistema informático, sus componentes eléctricos, electrónicos y electromecánicos. Los cables, así como los muebles o cajas de todo tipo también se incluyen dentro de esta categoría. [1]

Por otro lado, el **software** es el conjunto de componentes lógicos que hace posible la realización de tareas específicas [2]. Dicho de otra forma, es el conjunto de programas informáticos que actúa sobre el hardware para ejecutar lo que el usuario desee.

Según su funcionalidad, podemos diferencias tres tipos principales de software:

- **Sistema Operativo**: conjunto de programas de un sistema informático que gestiona los recursos hardware y provee servicios a las aplicaciones informáticas para su funcionamiento. Ejemplos de sistemas operativos son: Microsoft Windows, Linux, macOS...
- **Software de Programación**: conjunto de herramientas y utilidades que permiten a los programadores desarrollar programas informáticos.
- **Aplicaciones Informáticas**: programas o conjunto de programas que tienen una aplicación concreta. Algunos ejemplos de aplicaciones informáticas son los procesadores de texto, reproductores multimedia, juegos...

Aunque estos son los tipos principales de software, podemos ampliar esta clasificación incluyendo los siguientes tipos, los cuales son, en mayor medida, subdivisiones de las aplicaciones informáticas [3]:

- **Software de Tiempo Real**: son aquellos programas que se usan en sistemas de tiempo real y que reaccionan con el entorno físico respondiendo a los estímulos del éste en un tiempo limitado. Ejemplos de este software son los sistemas de control de procesos, aplicaciones de robótica,..

- **Software de Gestión**: son programas que utilizan grandes cantidades de información almacenadas en bases de datos para ayudar a la administración y toma de decisiones. Un ejemplo de estos sistemas son los **ERP**.
- **Software Científico o de Ingeniería**: software que se encarga de realizar complejos cálculos numéricos de todo tipo, siendo la corrección y exactitud en estos cálculos uno de los requisitos básicos. También incluye los sistemas de diseño, ingeniería y fabricación asistida por ordenador (CAD, CAE y CAM), simuladores gráficos,..
- **Software Empotrado**: software que por norma general va instalado en memorias ROM y sirve para controlar productos y sistemas de los mercados industriales. Se aplica a todo tipo de productos como neveras, reproductores de vídeo, misiles, sistemas de control de automóviles,...
- **Software de Inteligencia Artificial**: software que basándose en el uso de lenguajes declarativos, sistemas expertos y redes neuronales, para simular comportamientos humanos como el aprendizaje, razonamiento y la resolución de problemas para la realización de forma fiable y rápida operaciones que para el ser humano son tediosas o incluso inabordables. Ejemplos de estas aplicaciones son las aplicaciones de *machine learning*, chatbots,...

En este tema, nos centraremos en las **aplicaciones informáticas**, como se desarrollan y cuales son las fases de este desarrollo. También veremos el **ciclo de vida de una aplicación informática** y los diferentes tipos de **lenguajes de programación** y sus características.

1.2 Relación Hardware-Software

Como hemos comentado en el punto anterior, un sistemas informático se compone de hardware y software. Existe una relación indisoluble entre estos dos componentes ya que se necesita que estén instalados y configurados correctamente para que el ordenador funcione.

El primer modelo de arquitectura de hardware con programa almacenado fue propuesto por **John Von Neumman** en 1946. A continuación se muestra una imagen con las diferentes partes de esta arquitectura:

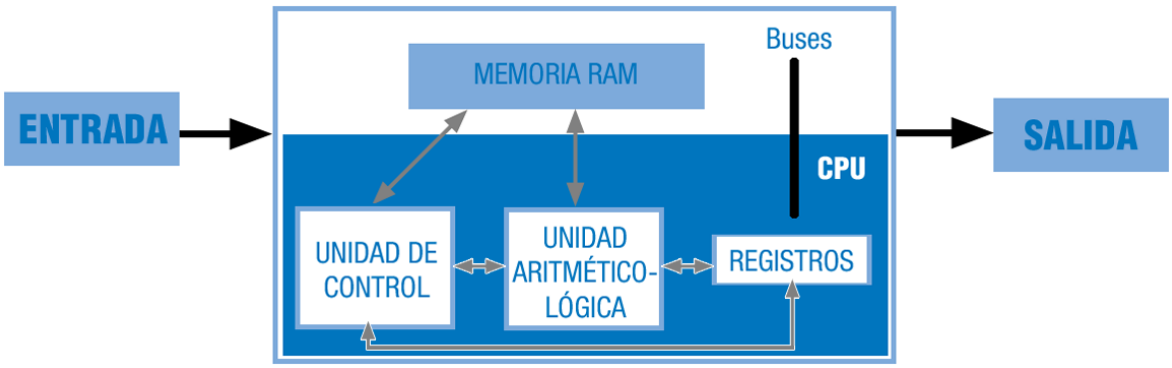

Figura 1.2.1: Arquitectura John Von Neuman

Esta relación software-hardware la podemos analizar desde dos puntos de vista diferentes:

a **Desde el punto de vista del sistema operativo**: el sistema operativo es el encargado de coordinar el funcionamiento del ordenador, actuando entre este y las aplicaciones que están corriendo en ese momento, gestionando los diferentes recursos hardware que requieren estas aplicaciones (CPU, RAM, interrupciones, dispositivos de E/S..).

b **Desde el punto de vista de las aplicaciones**: una aplicación solo es un conjunto de programas escritos en algún lenguaje de programación. Hay multitud de lenguajes de programación, con la característica de que todos están escritos en un idioma que el ser humano puede entender. El hardware por otro lado solo es capaz de interpretar señales eléctricas que se traducen en secuencias de 0 y 1 (código binario). Para que estás aplicaciones puedan ejecutarse en el hardware debe darse un proceso de "traducción"que veremos mas adelante.

1.3 Desarrollo de Software

Entendemos por **Desarrollo de Software** todo el proceso desde que se concibe la idea hasta que el programa está implementado y funcionando en el ordenador. Aunque en principio pueda parecer una tarea simple, consta de una series de pasos de obligado cumplimiento, pues solo así podemos garantizar que las aplicaciones creadas son eficientes, seguras, fiables y responden a las necesidades de los usuarios finales.

Como veremos más adelante en la unidad, esta serie de pasos se le suele denominar **Etapas** en el desarrollo de software. Según el orden y la forma en la que se llevan a cabo estas etapas hablaremos de diferentes ciclos de vida del software.

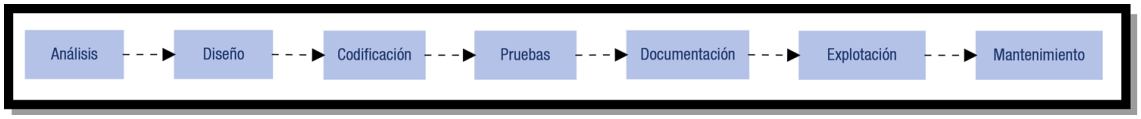

Figura 1.3.1: Etapas del Desarrollo de Software

Cabe destacar que la construcción de software es un proceso muy complejo y que requiere de una gran coordinación y disciplina del grupo de trabajo que lo desarrolle.

1.3.1 Ciclos de Vida del Software

Ya hemos visto que para crear software debemos seguir un número de pasos conocidos como ciclo de vida del software. Más adelante en esta unidad veremos en que consiste cada paso mas detalladamente. Por ahora, vamos a centrarnos en ver los diferentes ciclos de vida del software que existen atendiendo a como se desarrollan dichos pasos.

Aunque podemos encontrar diferentes clasificaciones sobre los distintos tipos de ciclos de vida del software los mas conocidos y utilizados son los siguientes:

1. **Modelo en Cascada**: es el modelo de vida clásico del software. Actualmente es prácticamente imposible de utilizar, salvo para desarrollos muy pequeños, ya que es necesario conocer todos los requisitos con antelación y las etapas pasan de una a otra sin retorno, presuponiendo que no ha habido errores en la etapa anterior.
2. **Modelo en Cascada con Realimentación**: es uno de los modelos más utilizados. Proviene del modelo anterior, pero se introduce una realimentación entre etapas, permitiendo que podamos volver hacia atrás en cualquier momento para corregir, depurar o modificar algún aspecto. Es el modelo perfecto si el proyecto es es rígido (pocos cambios y poco evolutivo) y los requisitos están claros. No obstante, si se prevén muchos cambios durante el desarrollo no es el modelo más idóneo.
3. **Modelos Evolutivos**: son los modelos más modernos y tienen en cuenta el aspecto cambiante y evolutivo del software. Dentro de este modulo podemos encontrar dos variantes:

3.1. **Modelo Iterativo Incremental**: ésta basado en el modelo en cascada con realimentación, donde las fases se repiten y refinan, propagando su mejora a las fases siguientes. El proyecto se realiza en pequeñas porciones (**incrementa**) en sucesivas iteraciones (**sprints**) al final de las cuales se ve lo que se ha desarrollado, pudiendo hacer correcciones o modificaciones antes de la siguiente iteración o incluso añadir nuevos requerimientos (**adaptativo**). Cada sprint debe proporcional un resultado completo preparado para entregárselo al clientes.

3.2. **Modelo en Espiral**: es una combinación del modelo anterior con el modelo en cascada. En éste, el software se va construyendo repetidamente en forma de versiones que son cada vez mejores, debido a que se va incrementando la funcionalidad. Es un modelo muy complejo.

En la actualidad, la metodologías de desarrollo ágil, enmarcadas dentro de los modelos evolutivos, están en auge, y metodologías como **Scrum**, **TDD** o **BDD**, así como metodologías híbridas basadas en estas, se están convirtiendo en el estándar de la industria. [4]

1.3.2 Herramientas de Apoyo al Desarrollo De Software

En la práctica, vamos a contar con un conjunto de herramientas que nos van a facilitar llevar a cabo las diferentes etapas del ciclo de vida de desarrollo, automatizando las tareas, ganando con ello fiabilidad y tiempo, y permitiéndonos centrarnos en los requerimientos del sistema y el análisis del mismo.

Las herramientas **CASE** son un conjunto de aplicaciones que se usan en el desarrollo de software con el propósito de reducir costes y tiempo de proceso, aumentando la productividad. Estas herramientas pueden ayudarnos prácticamente en cualquier etapa del proceso de desarrollo.

El desarrollo rápido de aplicaciones o **RAD**, es un proceso de desarrollo de software que comprende el desarrollo iterativo, la construcción de prototipos y el uso de herramientas CASE. Hoy en día se usa para referirnos al desarrollo rápido de interfaces de usuario o entornos de desarrollo integrados (**IDE**) completos.

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

- Mejorar la planificación del proyecto.
- Darle agilidad al proceso.
- Generar software mas reutilizable.
- Creación de aplicaciones más estandarizadas.
- Mejorar las tareas de mantenimiento de las aplicaciones.
- Permiten visualizar todo proceso de desarrollo de forma gráfica.

Las herramientas CASE se pueden clasificar según su funcionalidad o la función que desempeñan dentro del proceso de desarrollo.

Atendiendo a la función que desempeñan en cada fase del proceso, las herramientas CASE pueden ser:

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

Si tenemos en cuenta su funcionalidad, se pueden diferenciar algunas como:

- Herramientas de generación semiautomáticas de código.
- Editores **UML**.
- Herramientas de refactorización de código.
- Herramientas de mantenimiento, como los sistemas de control de versiones.

Algunos ejemplos de herramientas **CASE libres** son: ArgoUML, Use Case Maker, ObjectBuilder,...

1.4 Lenguajes de Programación

Podemos definir un **lenguaje de programación** como un idioma creado artificialmente, que se compone de un conjunto de símbolos y normas que se aplican sobre un alfabeto para obtener un código que el hardware de la computadora pueda entender y procesar. Es el instrumento que tenemos para que el ordenador realice las tareas que necesitamos, dicho de otra forma.

Hay multitud de lenguajes cada uno con su estructura y símbolos. Además cada lenguaje está enfocado en la realización de tareas o áreas determinadas. Por ello, es muy importante la elección del lenguajes o lenguajes de programación en un proyecto.

Los lenguajes de programación han ido evolucionando a través de la historia, podemos ver esta evolución de forma simplificada en la siguientes figura:

Figura 1.4.1: Evolución de los lenguajes de programación

Las principales características de estos lenguajes de son las siguientes:

- **Lenguajes Máquina**:
  - Sus instrucciones están compuestas por unos y ceros.
  - Es el único lenguaje que entiende el ordenador, no necesita traducción.
  - Fue el primer lenguaje utilizado.
  - Es único para cada procesador, es decir, no es portable de un equipo a otro.
  - Hoy en día nadie lo usa.
- **Lenguaje Ensamblador**:
  - Sustituyó el lenguaje máquina para facilitar la programación.
  - Se programa usando mnemotécnicos (instrucciones complejas).
  - Necesita traducción al lenguaje máquina para poder ejecutarse.
  - Sus instrucciones hacen referencia a la ubicación física de los archivos y registros.
  - Es difícil de utilizar.
- **Lenguajes de alto nivel**:

- Sustituyeron al ensamblador para facilitar la programación.
- Se utilizan sentencias y órdenes derivadas el inglés. Necesita traducción al lenguaje máquina.
- Son más cercanos al razonamiento humano.
- Son los más utilizados hoy en día.

- **Lenguajes Visuales**:
  - Están sustituyendo a los lenguajes de alto nivel basados en código.
  - Se programa gráficamente usando y diseñando directamente la apariencia del software.
  - Su correspondiente código se genera automáticamente.
  - Necesitan traducción al lenguaje máquina.
  - Son completamente portables de un equipo a otro.

1.4.1 Concepto y Características

Como ya hemos dicho, un **lenguaje de programación** es un lenguaje formal que le proporciona a una persona, en este caso el programador, la capacidad de escribir una serie de **instrucciones** o secuencia de órdenes en forma de algoritmo con el fin de controlar el comportamiento lógico o físico de un sistema informático, de manera que se pueden obtener diferentes formas de dato o realizar tareas. [5]

Un lenguaje de programación esta compuesto por:

- **Alfabeto**: conjunto de símbolos permitidos.
- **Sintaxis**: normas de construcción para los símbolos del lenguaje.
- **Semántica**: significado de las construcciones para realizar acciones válidas.

Los lenguajes de programación se pueden clasificar de varias formas en base de distintas características:

- **Según lo cercano que este al lenguaje humano**:
  - **Lenguajes de Alto Nivel**: son lenguajes que por su esencia, son mas cercanos al razonamiento humano.
  - **Lenguajes de Bajo Nivel**: estos lenguajes están mas próximos al lenguaje que entiende el computador. Los principales son: **Ensamblador** y **Lenguaje Máquina**.
- **Según la Técnica de Programación**:
  - **Lenguajes de Programación Estructurados**: usan la técnica de programación estructurada. Ejemplos son Pascal, C, etc...
  - **Lenguajes de Programación Orientada a Objetos**: usan las técnica de programación orientada a objetos. Ejemplos son Java, Ruby, Ada, C++, etc...
  - **Lenguajes de Programación Funcional**: basan su técnica en el uso de verdaderas funciones matemáticas. Ejemplos son Haskell, Elixir, etc...
  - **Lenguajes de Programación Visual**: basados en las técnicas anteriores, permiten programar gráficamente, generando automáticamente el código correspondiente. Ejemplos son Visual Basic.Net, Borland, etc..

A pesar de la cantidad de lenguajes de programación que existen, Python, C, Java, C++, C#, Visual Basic y Javascript, concentran alrededor del 60 % del interés de la comunidad mundial de programadores [6], aunque en última instancia, la **elección del lenguaje de programación** para un proyecto dependerá de las características del problema a resolver.

1.4.2 Lenguajes de Programación Estructurados

Los lenguajes estructurados fueron los primeros lenguajes de alto nivel que surgieron, y a partir de ellos, se evolucionó a los diferentes tipos de lenguajes que tenemos hoy en día.

La **programación estructurada** se define como la técnica para escribir lenguajes de programación que solo permite el uso de tres estructuras de control:

- Sentencias secuencias.
- Sentencias selectivas (condicionales).
- Sentencias iterativas (bucles).

Los lenguajes de programación que se basan en este estilo de programación se conocen como **lenguajes de programación estructurados**.

La programación estructurada fue un gran éxito debido a su sencillez a la hora de construir y leer programas. Aunque como todo, tiene sus ventajas e inconvenientes, que son los siguientes:

- **Ventajas**:
  - Los programas son fáciles de leer, sencillos y rápidos.
  - El mantenimiento de los programas se simplifica.
  - La estructura del programa es sencilla y clara
- **Inconvenientes**:
  - Todo el programa se concentra en un único bloque, si el código crece mucho se hace complejo de manejar.
  - No permite la reutilización de código, ya que todo el programa va en el mismo bloque.

Debido a estos inconvenientes, la **programación estructurada** evolucionó hacia la **programación modular**, que divide el programa en trozos de código llamados módulos, con una funcionalidad concreta, y que permiten su reutilización en otras aplicaciones.

Algunos de los lenguajes estructurados mas usados son C, FORTRAN, Pascal, etc...

1.4.3 Lenguajes de Programación Orientados a Objetos

Los **lenguajes de programación orientados a objetos** tratan los programas, no como un conjunto de instrucciones secuenciales, sino como un **conjunto de objetos** independientes, que interactúan entre sí, y que son altamente reutilizables en otras aplicaciones.

Su **principal desventaja** es que no es un paradigma de programación tan intuitivo como la programación estructurada. A pesar de ello, alrededor del 55 % del software que se produce emplea esta técnica. Esto es debido principalmente a que el código es **muy reutilizable** y a la facilidad de **depuración**.

Las principales **características** de estos lenguajes son:

- Los objetos del programa tendrán una serie de atributos que los diferencias unos de otros.

- Se definen clases como una colección de objetos con características similares.
- Mediante los llamados métodos, los objetos se comunican con otros produciendo un cambio de estado.
- Los objetos son unidades independientes e indivisibles que forma la base de este paradigma de programación.

Algunos de los lenguajes orientados a objetos más utilizados son C++, Java, Ruby, Delphi, etc..

1.5 Fases del Desarrollo de Software

Como hemos visto en puntos anteriores, podemos elegir entre diferentes modelos de desarrollo de software, pero independientemente de cual elijamos, siempre hay un número de etapas que debemos seguir. Estas etapas son las siguientes:

1. **Análisis de Requisitos**: en esta etapa se especifican los requisitos funcionales y no funcionales del sistema.
2. **Diseño**: se divide el sistema en partes y se determina la función de cada una.
3. **Codificación**: se elige un lenguaje de programación y se codifican los programas.
4. **Pruebas**: se prueban los programas para detectar errores y depurarlos.
5. **Documentación**: se documentan y guarda información de todas las etapas.
6. **Explotación**: instalamos, configuramos y ejecutamos la aplicación en los equipos del clientes.
7. **Mantenimiento**: se mantiene el contacto con el cliente para actualizar o modificar la aplicación.

En los siguientes puntos vamos a ver estas etapas con más detalle.

1.5.1 Análisis

Es la primera fase del proyecto y una de las de mayor importancia, ya que todo lo demás dependerá de lo bien detallada que este esta fase. También es una de las más complicadas, ya que no esta automatizada y dependerá en gran medida del análisis que hagamos del problema a resolver.

En esta fase se especifican los requisitos funcionales y no funcionales de la aplicación. Estos **requisitos** consisten en:

1. **Requisitos funcionales**: estos requisitos especifican que tendrá que realizar la aplicación, que respuesta dará ante todas las entradas, como se comportará en situaciones inesperadas, etc...
2. **Requisitos no funcionales**: estos especifican requisitos como el tiempo de respuesta, legislación aplicable, etc...

Es de vital importancia una **buena comunicación** entre el **analista** y el **cliente** para que los requisitos se puedan especificar con el máximo detalle y adecuados a los deseos del cliente.

La culminación de esta fase es un **documento ERS** (Especificación de Requisitos del Sistema) donde deben quedar especificado lo siguiente:

- La planificación de las reuniones que van a tener lugar.
- Relación de los objetivos del cliente y el sistema.
- Relación de los objetivos funcionales y no funcionales del sistema.

- Relación de objetivos prioritarios y temporalización.
- Reconocimiento de requisitos mal planteados, etc...

Una vez realizado este documento y con los requisitos especificados y detallados, pasaremos a la siguiente fase, la fase de diseño.

1.5.2 Diseño

En esta segunda fase, el sistema debe dividirse en diferentes partes y establecer que relación habrá entre ellas. Decidir exactamente que hará cada parte. En definitiva, debemos crear un modelo funcional-estructural de los requerimientos del sistema global, para poder dividirlo y afrontarlo por separado.

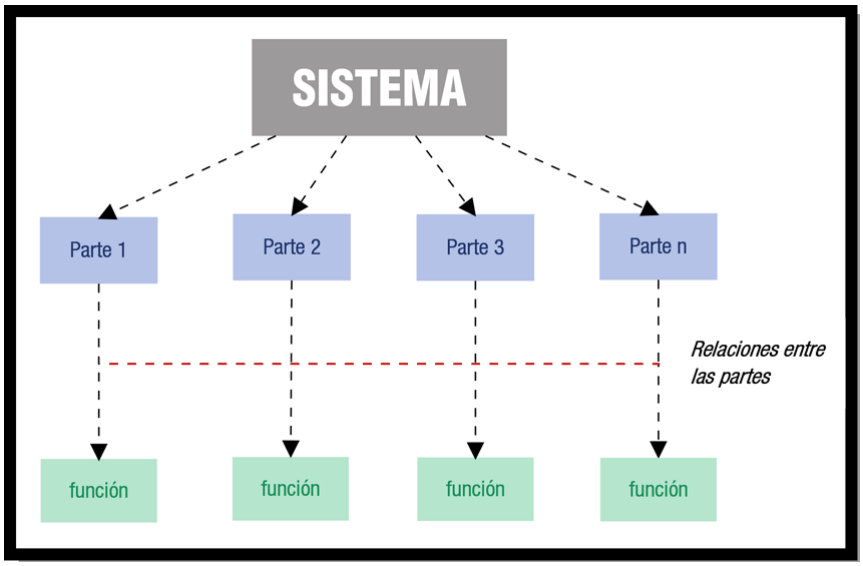

Figura 1.5.1: División de la aplicación en partes

Además, en esta etapa se deben tomar decisiones que van a afectar el desarrollo del software, como:

- Entidades y relaciones de la base de datos.
- Selección del lenguaje de programación que se va a utilizar.
- Selección del SGBD.
- Definición de diagrama de clases.
- Definición de diagrama de colaboración.
- Definición del diagrama de paso de mensajes.

1.5.3 Codficación

En esta fase, con el lenguajes de programación elegido, codificar toda la información anterior y llevarla a código fuente. Esta tarea la realiza el **programador** y tiene que cumplir con todos los requisitos y restricciones impuestos en la fase de análisis y diseño de la aplicación.

Ademas, el **código** deberá cumplir con las siguientes **características**:

- **Modularidad**: que este divido en trozos pequeños funcionales.

- **Corrección**: que haga lo que se pide realmente.
- **Fácil de leer**: que sea fácil de leer y comprender para facilitar su desarrollo y mantenimiento.
- **Eficiencia**: que haga un buen uso de los recursos.
- **Portabilidad**: que se pueda ejecutar en cualquier equipo.

El código pasará por un número de fases desde su implementación hasta que se pueda ejecutar. Estas fases son las siguientes:

1. **Código Fuente**: es el código escrito por los programadores usando algún edito o IDE. Se escribe usando algún lenguaje de programación de alto nivel y contiene el conjunto de instrucciones necesaria.
2. **Código Objeto**: es el código binario resultado de compilar el código fuente. La **compilación** es la traducción de una sola vez del programa y se realiza utilizando un compilador. La **traducción** es la interpretación y ejecución simultánea de un programa línea a línea.
3. **Código Ejecutable**: es el resultado de enlazar el código objeto con ciertas **rutina** y **biblioteca** necesarias. El sistema operativo se encarga de cargar el programa en la memoria RAM y proceder a ejecutarlo.

En los siguiente punto vamos a profundizar un poco más en los diferentes estados y fases en los que se encuentra el código en esta etapa.

1.5.3.1 Código Fuente

El código fuente es el conjunto de instrucciones que la computadora debe realizar, escritas por un programador en algún lenguaje de programación. Este conjunto de instrucciones no es directamente interpretable por la máquina, sino que deberá de pasar por un proceso de traducción para que pueda ser ejecutado.

Una parte importante de esta fase es la **elaboración de un algoritmo**, que definimos como un conjunto de pasos a seguir para obtener la solución a un problema. El algoritmo los diseñamos en pseudocódigo, y con el la codificación a un lenguaje de programación determinado será más rápida y directa.

Para obtener el código fuente de una aplicación se deben seguir estos pasos:

1. Se debe partir de la etapa anterior de análisis y diseño.
2. Se diseñara un algoritmo que simbolice los pasos a seguir para solucionar el problema.
3. Se elegirá un lenguaje de algo nivel apropiado para las características del software que se quiere codificar.
4. Se procederá a codificación del algoritmo diseñado.

La culminación de este proceso es la obtención de un documento con todos los **módulos**, **funciones**, **bibliotecas** y **procedimientos** necesarios para codificar la aplicación. Puesto que el código es inteligible por la máquina habrá que traducirlo, obteniendo así un código equivalente pero ya entendible por el ordenador.

También hay que tener en cuenta, en este punto, bajo que **licencia** vamos a crear el código, siendo las dos más comunes:

- **Licencias de Código Fuente Abierto**: son licencias que permiten que cualquier usuario o programador pueda estudiar el código, modificarlo y reutilizarlo.

- **Licencias de Código Fuente Cerrado**: estas licencias no dan permisos para editar ni modificar el código fuente.

1.5.3.2 Código Objeto

El código objeto es un código intermedio, traducido a unos y ceros, pero que aún no puede ser ejecutado directamente. Consiste en **bytecode** que esta distribuido en varios archivos, según la cantidad de módulos de software que compongan la aplicación. Este código se **debe generar** cuando el **código fuente** este **libre de errores** sintácticos y semánticos.

El proceso de transformación del código fuente a código puede realizarse de **dos formas**:

- **Compilación**: el proceso de traducción se realiza sobre todo el programa de una vez. El software que realiza esta operación se llama **compilador**.
- **Traducción**: el proceso de traducción del código fuente se realiza línea y línea y se ejecuta simultáneamente. No existe código objeto intermedio. El software utilizado se denomina **intérprete**. El proceso de traducción es más lento que el de compilación.

1.5.3.3 Ejecutable

El código ejecutable es el que obtenemos al enlazar el código objeto con las librerías y rutinas necesarias, resultando un único archivo que puede ser directamente ejecutado por la computadora. Este archivo es ejecutado y controlado por el sistema operativo.

Para obtener un solo archivo ejecutable habrá que enlazar todos los archivos de código objeto mediante una aplicación denominada **linker** (enlazador) y obtener así un único archivo que ya si es ejecutable directamente por la computadora.

En la siguiente figura vemos el proceso completo para generar archivos ejecutables.

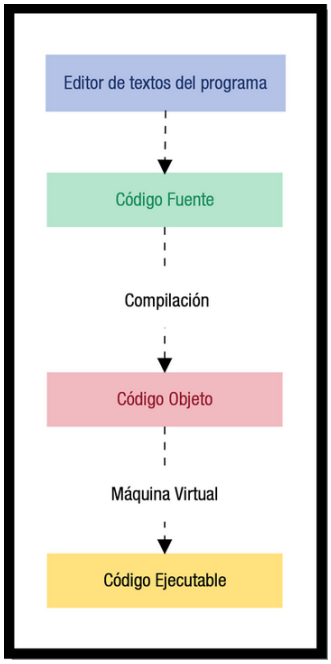

Figura 1.5.2: Generación de código ejecutable

1.5.4 Pruebas

Una vez obtenido el software, la siguiente fase del ciclo de vida es la realización de pruebas. Normalmente, estás se realizan sobre un conjunto de datos de prueba, que consiste en un conjunto de datos límite a los que la aplicación es sometida.

La **realización de pruebas** es una fase **imprescindible** para asegurar la **validación** y **verificación** del software desarrollado y podemos clasificarlas en dos tipos:

- **Pruebas Unitarias**: consiste en probar una a una las diferentes partes del software y comprobar su funcionamiento de forma individual. Un ejemplo de librerías para realizar este tipo de pruebas es **JUnit** para el lenguaje Java o **Jest** para Javascript.
- **Pruebas de Integración**: se realiza una vez que se han realizado las pruebas unitarias y consistirán en probar el funcionamiento del sistema completo, con todas sus partes interrelacionadas.

La prueba final, también conocida como **Beta Test** se realizará sobre el entorno de producción del cliente.

1.5.5 Documentación

Para proporcionar toda la información posible sobre el uso del software, tanto a los usuarios como a otros desarrolladores, es necesario crear la **documentación** de la aplicación.

Debemos ir documentando cada fase del proyecto, ademas, para ir pasando de una otra de forma clara y definida. Una correcta documentación facilitará la reutilización de parte del software para futuros proyectos así como el mantenimiento y su utilización por otros desarrolladores. En la siguiente figura se muestra una tabla con los diferentes tipos de documentación que se pueden generar.

Documentos a elaborar en el proceso de desarrollo de software

	GUÍA TECNICA	GUÍA DE USO	GUÍA DE INSTALACIÓN
Quedan reflejados:	• El diseño de la aplicación. • La codificación de los programas. • Las pruebas realizadas.	• Descripción de la funcionalidad de la aplicación. • Forma de comenzar a ejecutar la aplicación. • Ejemplos de uso del programa. • Requerimientos software de la aplicación. • Solución de los posibles problemas que se pueden presentar.	Toda la información necesaria para: • Puesta en marcha. • Explotación. • Seguridad del sistema.
¿A quién va dirigido?	Al personal técnico en informática (analistas y programadores).	A los usuarios que van a usar la aplicación (clientes).	Al personal informático responsable de la instalación, en colaboración con los usuarios que van a usar la aplicación (clientes).
¿Cuál es su objetivo?	Facilitar un correcto desarrollo, realizar correcciones en los programas y permitir un mantenimiento futuro.	Dar a los usuarios finales toda la información necesaria para utilizar la aplicación.	Dar toda la información necesaria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa.

Figura 1.5.3: Tipos de documentación en el desarrollo de software

1.5.6 Explotación

Glosario

biblioteca Conjunto de subprogramas que sirven para desarrollar componentes software o que actúan como interfaz de comunicación entre componentes software. 13

CASE Computer Aided Software Engineerig. 7, 13

Desarrollo de Software Conjunto de procesos desde que nace una idea hasta que se convierte en software. 6, 13

ERP Enterprise Resource Planning o Sistema de Planificación de Recursos Empresariales, son programas que se usan para la gestión empresarial y que se hacen cargo de distintas operaciones internas, desde la producción, la distribución o incluso los recursos humanos. 5, 13

IDE Integrated Development Environment. 7, 13

Linux Sistema Operativo tipo Unix compuesto por software libre y de código abierto que sirve como base para multitud de distribuciones como Debian, Slackware, Ubuntu, Gentoo.... 4, 13

macOS Sistema Opertativo basado en Unix, más concretamente en FreeBSD, y que está desarrollado y comercializado por Apple desde 2001. 4, 13

Microsoft Windows Sistema Operativo desarrollado por Microsoft que actualmente se encuentra en la versión 11. 4, 13

RAD Rapid Application Development. 7, 13

UML Unified Modeling Language. 8, 13

Bibliografía

[1] Wikipedia - Hardware. https://es.wikipedia.org/wiki/Hardware.

[2] Wikipedia - Software. https://es.wikipedia.org/wiki/Software.

[3] Joaquin Marquéz y Sergio Ernesto Gastón Perez. Tipos de software. https://es.slideshare.net/susahhreyyhha/tipos-de-software-15979630.

[4] Amna Batool. A survey of key challenges of agile in global software development: A case study with malaysia perspective. *International Journal of Industrial and Manufacturing Engineering, Vol13, nº10.*

[5] Wikipedia - Lenguaje de programación. https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n.

[6] Indice TIOBE. https://www.tiobe.com/tiobe-index/.