Unidad de trabajo 2: El desarrollo del software

Apuntes de entorno de desarrollo

Nº	UNIDAD DE TRABAJO	DURACIÓN (HORAS)	EVALUACIÓN		
			1ª	2ª	3ª
2	El desarrollo de software.	10	X		

OBJETIVOS	1. **Entender** conceptos básicos de software. 2. **Conocer y utilizar** terminología propia de software. 3. **Conocer** los diferentes tipos de lenguajes de programación y sus principales características.
CONTENIDOS	• Concepto de programa informático. • Código fuente, código objeto y código ejecutable; tecnologías de virtualización. • Tipos de lenguajes de programación. Paradigmas. • Características de los lenguajes más difundidos. • Fases del desarrollo de una aplicación: análisis, diseño, codificación, pruebas, documentación, explotación y mantenimiento, entre otras. • Proceso de obtención de código ejecutable a partir del código fuente; herramientas implicadas. • Metodologías ágiles. Técnicas. Características.
RESULTADOS DE APRENDIZAJE Y CRITERIOS DE EVALUACIÓN	RA1. **Reconocer** los elementos y herramientas que intervienen en el desarrollo de un programa informático, analizando sus características y las fases en las que actúan hasta llegar a su puesta en funcionamiento. CE a) **Reconocer** la relación de los programas con los componentes del sistema informático: memoria, procesador, periféricos, entre otros. CE b) **Identificar** las fases de desarrollo de una aplicación informática. CE c) **Diferenciar** los conceptos de código fuente, objeto y ejecutable. CE d) **Reconocer** las características de la generación de código intermedio para su ejecución en máquinas virtuales. CE e) **Clasificar** los lenguajes de programación, identificando sus características. CE f) **Evaluar** la funcionalidad ofrecida por las herramientas utilizadas en el desarrollo de software. CE g) **Identificar** las características y escenarios de uso de las metodologías ágiles de desarrollo de software.

Materiales complementarios y vídeos:

https://wirtzIDE.blogspot.com/

Fernando Rodríguez Dieges
rdf@fernandowirtz.com
Versión 2024-07-23

1. Software y proyectos de desarrollo de software

1.1 Software

La Real Academia Galicia define la informática como "la ciencia del procesamiento automático de la información a través de máquinas electrónicas". Este tratamiento requiere:

1. Manejar el soporte físico o hardware formado por todos los componentes electrónicos tangibles involucrados. Según la Real Academia Galega, el soporte físico es preferible al hardware y lo define como la maquinaria o elemento que constituye una computadora.
2. El soporte lógico o software que hace que el hardware funcione, compuesto por todos los componentes intangibles involucrados en el proceso de procesamiento: programas, datos y documentación. Según la Real Academia Galicia, el soporte lógico es preferible al software y lo define como un conjunto de comandos y programas que permiten el uso de una computadora.
3. Equipo humano o personal informático que maneja los dispositivos físicos y lógicos para realizar todos los tratamientos.

El concepto de un programa informático está estrechamente relacionado con el concepto de un algoritmo. Un algoritmo es un conjunto ordenado y finito de instrucciones o reglas bien definidas que permiten resolver un problema.

Algoritmos para cambiar las ruedas del coche.

Información: Posición de la rueda pinchada y el gato, desde la rueda de repuesto y hasta la clave inglesa

Paso 1. Afloje el tornillo de la rueda cónica con una llave inglesa

PASO 2. Ponga el gato mecánico bajo el coche

PASO 3. Levante el gato hasta que la rueda pinchada pueda girar

Paso 4. Retire los tornillos

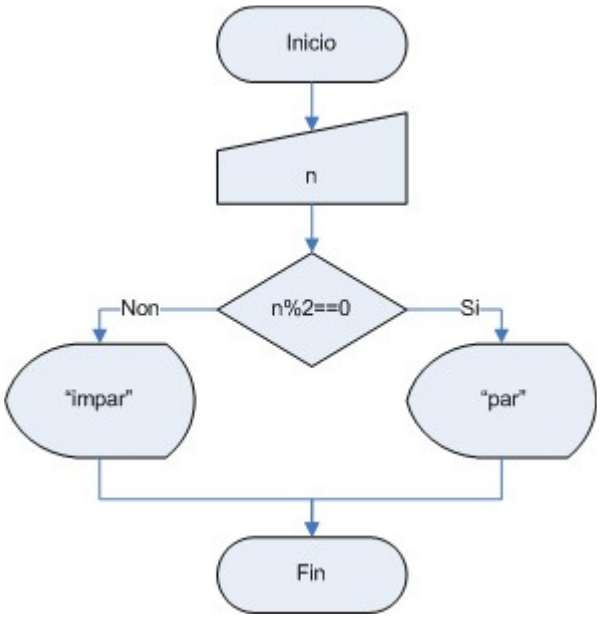
PASO 5. Retire las ruedas atadas

PASO 6. Ponte una llanta de repuesto

PASO 7. Coloque el tornillo y apriete suavemente el paso 8. Deje el gato hasta ue pueda ser liberado en el paso 9. Eliminar los gatos de su sitio web

PASO 10. Apriete los tornillos de la llave inglesa

Ejemplo de algoritmo representado a través de un diagrama de flujo que permite ver si los números escribidos son pares o impares, y trata a 0 como pares:



EL OBJETIVO FINAL Y LA SOLUCIÓN DE UN PROBLEMA. ¿CÓMO RESOLVERLO? EL PROBLEMA DE USAR UN ALGORITMO (INSTRUCCIÓN) Y DATOS.

Las computadoras no entienden el lenguaje natural: cómo podemos decirle a una computadora lo que necesita hacer: Los algoritmos se implementan escribiendo programas en algún lenguaje de programación.

El algoritmo puede escribirse en un lenguaje de programación usando algún tipo de herramienta de edición, dando como resultado que el código tenga que ser escrito en la memoria externa no volátil para perderse con el tiempo. Ejemplo de código escrito en lenguaje Java correspondiente al algoritmo anterior:

```
package parimpar;

import   java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int n;
        Scanner teclado = new Scanner(System.in);
        System.out.print("Teclee un número enteiro:");
        n = teclado.nextInt();
        if(n%2==0){
            System.out.println(n + " es par");
        }
        else{
            System.out.println (n + " es impar");
        }
    }
}
```

El código debe ser modificado hasta cierto punto para finalmente obtener un programa que se pueda ejecutar en una computadora. Para ser ejecutado, el programa necesita ser almacenado en la memoria interna y volátil del ordenador; Así que el procesador puede ir a recoger cada pedido

Qué forma el procedimiento, lo resuelve, supervisa su ejecución. Si es necesario, se accede a la memoria interna para procesar variables, o periféricos de entrada, salida o almacenamiento, para realizar cálculos o resolver expresiones lógicas hasta que se completa con éxito la última instrucción o se encuentra un error irresoluble.

Al ejecutar el programa correspondiente al código de ejemplo anterior, el procesador necesitará un número entero suministrado por el teclado (dispositivo de entrada o periférico), obtener el resultado de una operación aritmética (el resto del entero n dividido por 2), resolver una expresión lógica (averiguar si el resto anterior es 0), y ejecutar una instrucción alternativa para emitir n como un número par o impar desde la pantalla (dispositivo de salida o periférico) si el resto es 0. Ejemplo de ejecución del código anterior, y la apariencia del resultado de la ejecución en la pantalla de texto cuando se escribe el número 11:

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ ejercicios ---
Teclee un número enteiro:11
11 é impar
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 6.113 s
Finished at: 2020-07-29T10:33:00+02:00
Final Memory: 7M/27M
------------------------------------------------------------------------
```

Un programa informático es necesario para que una computadora funcione y es un conjunto de instrucciones que, una vez ejecutadas, realizan una o más tareas. El software es un conjunto de programas, y dentro de este concepto también se incluyen los datos y documentos que trabajan con estos programas.

Pseudocódigo e organigramas

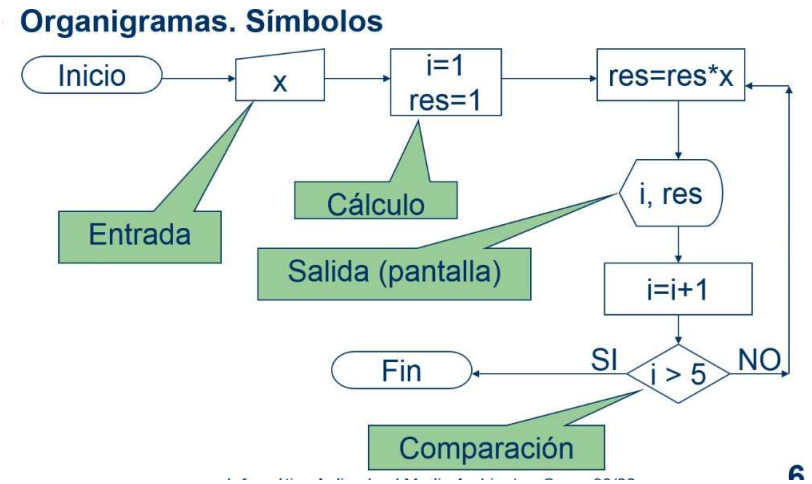
Antes de programar el algoritmo, esto puede ser representado de una manera más "natural" para entenderlo mejor y hacer que el proceso de programación sea más simple y libre de errores. Para iso, podemos usar pseudocódigo u organigramas.

El pseudocódigo es una descripción verbal que muestra el proceso a seguir en cada paso en un lenguaje casi natural.

Ejemplo: programa para escribir 5 primeras potencias de un número (pseudocódigo):

```
1 programa Potencias;
2 leer(x)
3 i = 1; res = 1;
4 res = res*x;
5 escribir(x " elevado a " i " es " res);
6 i = i + 1
7 si (i > 5) entonces
    terminar
8 ir al paso 4
9 fin.
```

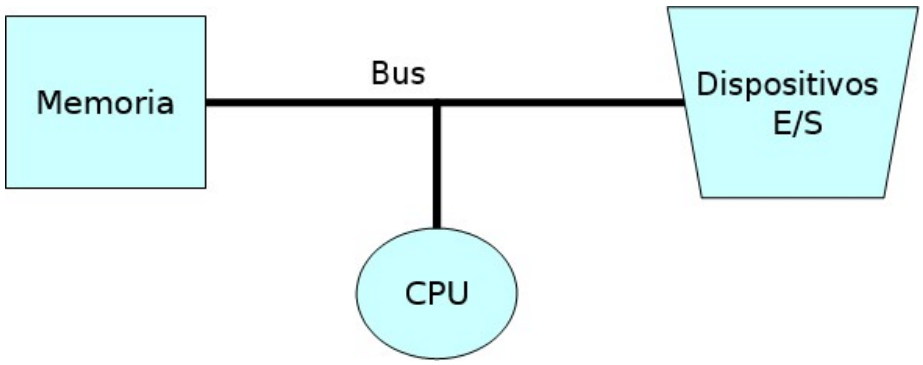
Un organigrama es una descripción gráfica de un problema, donde cada símbolo representa una acción (entrada, salida, comparación, etc.).



b)

1.2 Hardware y software

El 99% de las computadoras actuales tienen la siguiente arquitectura:

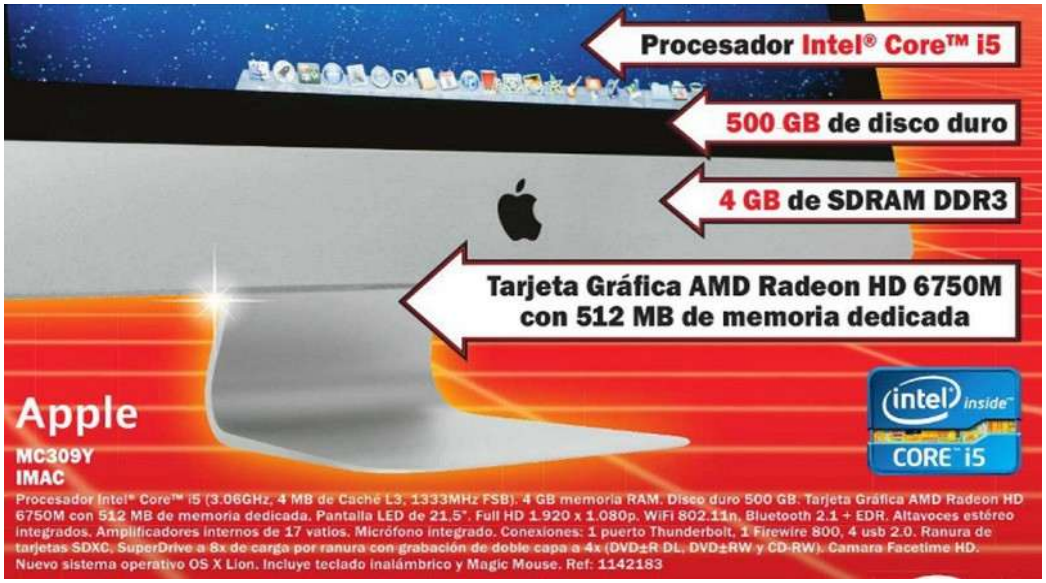


Memoria compartida de datos e instrucciones: conocida como arquitectura von Neumann , aunque fue propuesta originalmente por Eckert y Mauchly.

Componentes

- Unidad Central de Procesamiento (CPU): principal responsable de la ejecución de instrucciones y la coordinación del resto
- Memoria: almacena datos, instrucciones y resultados. Clasificación:
  - Permanente/volátil: información que se guarda en un almacenamiento permanente (por ejemplo, un disco duro) después de que la computadora se apagó, en un
    - almacenamiento no volátil (por ejemplo, RAM)
    - Primario/secundario: donde se ejecutan los procesos primarios y secundarios (RAM) y los datos y programas secundarios (disco). El primero es volátil y el segundo es permanente. La memoria principal es más pequeña, más rápida y más cara que la memoria secundaria.
    -
- Dispositivos de entrada/salida: proporciona datos e instrucciones y recibe resultados
- Bus de datos: para compartir información entre los componentes mencionados anteriormente

Ejemplos de computadoras reales



1.3 Clasificación de software

El software se puede dividir en dos tipos:

- **Software de sistema.** Controla y gestiona el hardware haciéndolo funcionar, ocultando el flujo interno de este trabajo al personal informático. Ejemplos de este tipo de software incluyen:
    - -Sistema operativo
    - Controladores de dispositivos
    - -Herramientas de diagnóstico
    - Servidores (correo, web, DNS, etc.)
    - Utilidades del sistema (compresión de información, rastreo de ejes defectuosos en el soporte.)

- **Software de aplicación.** Permite realizar una o más tareas específicas en cualquier área de actividad desde la instalación del software básico del sistema. Ejemplos de este tipo de software son:
    - Aplicaciones de control de sistemas y automatización industrial
    - Aplicaciones de oficina
    - Software educativo
    - Software empresarial: contabilidad, nómina, almacén,...
    - Bases de datos
    - -Videojuegos
    - Software de comunicación: navegador web, cliente de correo,...
    - -Software médico
    - Software de cálculo numérico
    - Software de diseño asistido por computadora

- El **software** de programación es el conjunto de herramientas que permiten la programación Desarrollo de programas informáticos tales como:
    - Editor de texto
    - Compiladore
    - sIntérpretes
    - Enlazadore
    - sScrubber
    - Un marco de desarrollo integrado (IDE), que agrupa las herramientas anteriores Es conveniente para los programadores completar la tarea de desarrollar programas informáticos en el mismo entorno y tener una avanzada interfaz gráfica de usuario (GUI).

El software de aplicación se puede dividir internamente:

- · **Software horizontal o general** que se puede utilizar en una variedad de entornos, como aplicaciones de oficina.
- **Software vertical o personalizado**, solo pueden ser utilizados en un contexto específico, como la contabilidad adaptada a una empresa específica.

> Misión 1.2. Busque en línea los nombres comerciales de algunos de los software del sistema y software de aplicaciones anteriores. Haz una tabla: tipo de software,nombre comercial, tipo de licencia.

1.4 Desarrollo de software

El proceso de desarrollo de software varía considerablemente dependiendo del grado de complejidad del software. Por ejemplo, desarrollar un sistema operativo requiere un equipo disciplinado, recursos, herramientas, un proyecto a seguir y alguien para gestionar todo. En el otro extremo, para hacer un programa que pueda mostrar si los números enteros entrados son pares o impares, solo se necesita un programador o entusiasta de la programación y un algoritmo de solución.

Fritz Bauer propuso por primera vez una definición de ingeniería de software en una reunión del Comité Científico de la OTAN en 1969, como el establecimiento y uso de principios de ingeniería robustos destinados a obtener económicamente software que funcione de manera confiable y eficiente en máquinas reales. Esta definición fue posteriormente refinada por muchas figuras prominentes en la comunidad del software y estandarizada en el estándar IEEE de 1993, que define la ingeniería de software como la aplicación de un enfoque sistemático, disciplinado y mensurable al desarrollo, operación y mantenimiento de software.

Durante décadas, los ingenieros de software han estado desarrollando y mejorando paradigmas (métodos, herramientas y procedimientos para describir modelos) que son sistemáticos, predecibles y repetibles, lo que aumenta la productividad y la calidad del software. La ingeniería de software es necesaria en proyectos de software a gran escala, debe aplicarse en proyectos de tamaño mediano y se recomienda aplicar algunos de sus procesos en proyectos pequeños.

Hay muchos patrones a seguir para desarrollar software. El más clásico es el modo en cascada.

Entre ellos destacan el modelo espiral **basado** en prototipos, la programación extrema como representante de la metodología de programación ágil y el modelo de métrica 3 para su aplicación a gran escala en proyectos relacionados con instituciones públicas.

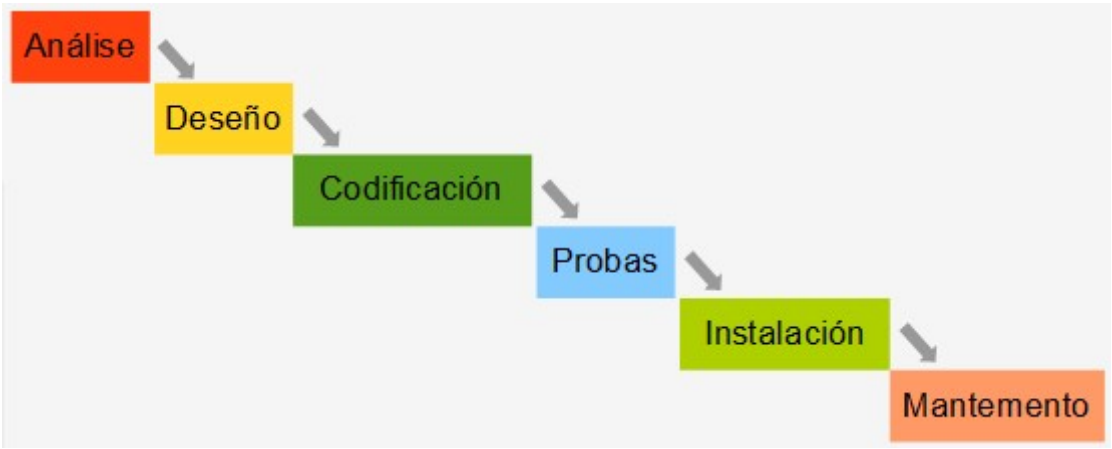
1.5 Paradigma clásico del ciclo de vida o modelo de caso por caso

La Royal Academy of Languages define un paradigma como "una teoría o conjunto de teorías cuyo núcleo central es aceptado sin duda y que proporciona la base y el modelo para resolver problemas y avanzar en el conocimiento".

Para la ingeniería de software, un paradigma es la combinación de métodos, herramientas y procedimientos para describir un modelo. Cada metodología de desarrollo de software tiene su propia metodología de desarrollo de software en mayor o menor medida.

El paradigma clásico del ciclo de vida del software, también conocido como modelo en cascada, consta de fases: análisis, diseño, codificación, pruebas, instalación y mantenimiento, siendo la documentación un aspecto implícito en cada fase. Algunos autores nombran las etapas con nombres ligeramente diferentes, o pueden combinar algunas para crear una nueva etapa, o nombrar una etapa con más detalle, pero esencialmente las etapas son todas las mismas.

Algunas de estas fases también se utilizan en otros modelos, con ligeras diferencias. Las fases se ejecutan secuencialmente con la información obtenida al final de una fase paracomenzar la siguiente fase. De esta manera, el cliente no puede ver el software en ejecución hasta la etapa final, y se necesita mucho trabajo para corregir un error detectado en la etapa final pero afectando a la etapa anterior.



ANALISIS

En esta etapa, el analista captura, analiza y especifica las necesidades que el software debe cumplir. Debe obtener información de los clientes o usuarios del software a través de entrevistas planificadas y inteligentemente estructuradas en un lenguaje comprensible para el usuario. El resultado de esta captura de información depende básicamente de las habilidades y experiencia del analista, incluso si tiene acceso a una guía o software específico.

Al final de esta fase, debe existir un documento de especificación de requisitos de software (ERS), en el que se detallarán los requisitos que el software necesita cumplir, se evaluarán los costos del proyecto y se planificará su duración. Toda esta información necesita ser comunicada al cliente para que la acepte.

El lenguaje utilizado para describir ERP (Enterprise Resource Planning: es el sistema

informático utilizado para la gestión de recursos en una organización) puede ser descriptivo o más formal y riguroso utilizando casos de uso en el lenguaje de modelado UML. El estándar IEEE 830-1998 recopila un estándar de prácticas recomendadas para la especificación de requisitos de software.

Diseño

En esta etapa, los diseñadores necesitan descomponer y organizar todo el sistema de software en partes que se pueden desarrollar por separado para aprovechar el desarrollo de software en equipo.

Los resultados de esta etapa se reflejan en el documento de diseño de software (SDD = Descripción del diseño de software), que contiene la estructura general del sistema, el trabajo que deben realizar las partes y cómo combinarlas, y es una guía que los programadores y probadores de software deben leer, comprender y seguir. Este artículo incluirá diseño de lógica de datos, arquitectura y diseño estructural, programación, organización y compilación del código fuente y diseño de interfaz hombre- máquina.

Codificación

Esta fase, llamada de programación o implementación, consiste en convertir el diseño lógico en código utilizando el lenguaje elegido, asegurando que cumpla los requisitos y sea ejecutable. Durante este proceso, se depura y revisa el código para corregir errores sintácticos, semánticos y lógicos.

Pruebas

Esta fase permite aplicar métodos o técnicas al código para asegurarse de que todas estas declaraciones han sido probadas y funcionan correctamente.

Las pruebas requieren planificación, diseño, ejecución y evaluación de los resultados. Una prueba se considera buena si revela errores que no se habían detectado anteriormente. Las pruebas realizadas inmediatamente después de la codificación pueden ser:

- Pruebas unitarias, cuando permiten realizar pruebas en pequeños trozos de código con una funcionalidad específica.
- Las pruebas de integración son las que permiten probar un conjunto de fragmentos de código que han pasado con éxito las pruebas unitarias correspondientes e influyen entre sí.

➢ Otras pruebas sobre todo el sistema podrían ser:

- Pruebas de verificación o aceptación para verificar que el sistema cumple con los requisitos de software establecidos en el ERS.
- Prueba de recuperación, que comprueba cómo reacciona el sistema frente a fallas generales y cómo se recupera de ellas.

- Pruebas de seguridad para verificar que los datos están protegidos contra ataques externos.

- Prueba de durabilidad, que prueba la reacción del sistema a los intentos de bloqueo o caída.
- Pruebas de rendimiento que permiten que el sistema satisfaga las necesidades extremas del usuario y pruebe el tiempo de respuesta del sistema.
-

Las pruebas deben ser realizadas primero por los desarrolladores del software, pero las recomendaciones también deben ser realizadas por expertos que no han participado en la creación y, en última instancia, por los usuarios.

El software puede colocarlo en el escritorio cuando el usuario aún no ha terminado, y luego nombrarlocon el nombre de la empresa y el texto que indica el grado de finalización. Este texto puede ser:

- Versión Alfa. Una versión inestable en la que aún se pueden añadir nuevas características y que aún no ha sido lanzada por la compañía que la desarrolló o por usuarios conocidos (testers)

- Versión beta. Las versiones inestables no añaden nuevas características, pero pueden tener errores. Ha sido lanzado al público, pero indica que se trata de una versión que aún está en beta.

- RC (Candidato de Versión) Es casi la versión final, pero todavía puede haber errores

- Por lo general, hay pequeños errores porque ha pasado por una versión beta.

- Versión RTM (Release to Manufacturing). V ersión estable para comercializar. Windows em

> Misión 1.3. Busque en Internet las versiones Alpha, beta, RC o RTM de aplicaciones de software disponibles para el usuario.

Instalación

Esta fase, también conocida como despliegue o despliegue, consiste en mover el software del sistema a la computadora de destino y configurarlo para que pueda ser utilizado por los usuarios finales. Esta etapa puede consistir en una simple copia de archivos, o puede ser más compleja, por ejemplo: copiar programas y datos comprimidos en una ubicación específica en el disco duro, crear accesos directos en el escritorio, crear una base de datos en una ubicación específica, etc.

Hay algunas herramientas de software que automatizan este proceso, llamadas instaladores. En el caso de una instalación simple, el instalador puede generar algunos archivos que permitan al usuario final automatizar el arranque y la instalación simple; En otro caso, se requiere la instalación por personal especializado.

El software puede ponerse en producción después de resolver el proceso de instalación, es decir, puede ser utilizado y operado por los clientes.

Mantenimiento

Esta fase permite mejorar y optimizar el software que se está produciendo. El mantenimiento permitirá

realizar cambios en el código para corregir los errores encontrados, perfeccionarlo (optimización del rendimiento y la eficiencia, reestructuración del código, perfeccionamiento de la documentación, etc.), evolucionarlo continuamente (añadir, modificar o eliminar funciones según las necesidades del mercado, etc.), o adaptarlo (cambios en el hardware utilizado, cambios en el software del sistema de administración de bases de datos, cambios en el sistema de comunicación, etc.).

Alrededor de dos tercios del tiempo invertido en ingeniería de software se dedica a tareas de mantenimiento, y a veces estas tareas son tan numerosas y complejas que el costo de rediseñar el código es más barato.

La versión del software generada por el mantenimiento varía según el proveedor. Por ejemplo:

Debian 7. 6, NetBeans IDE 6. 5, NetBeans IDE 7. 0. 1, NetBeans 8. 0, Java SE 8u20 (versión 8 actualizada 20). Algunos fabricantes ofrecen aplicaciones que son parches que mejoran el software instalado, como Service Pack 1 (SP1) para Windows Server 2008 R2, o Service Pack 2 (SP 2) para Windows Server 2008 R2.

> Misión 1.4. Busque en la web las últimas versiones de Debian, Wordpress, Java, Windows.

Documentación

La creación de documentación se asocia con todas las fases anteriores, en particular con las fases de codificación, prueba e instalación. Para esta última etapa, se puede distinguir:

 Documentación interna. Aparece en el código del programa para que el programador encargado de mantenerlo pueda obtener información adicional sin esfuerzo y puede extraer automáticamente esta información del código en formatos legibles como HTML, PDF, CHM, etc.

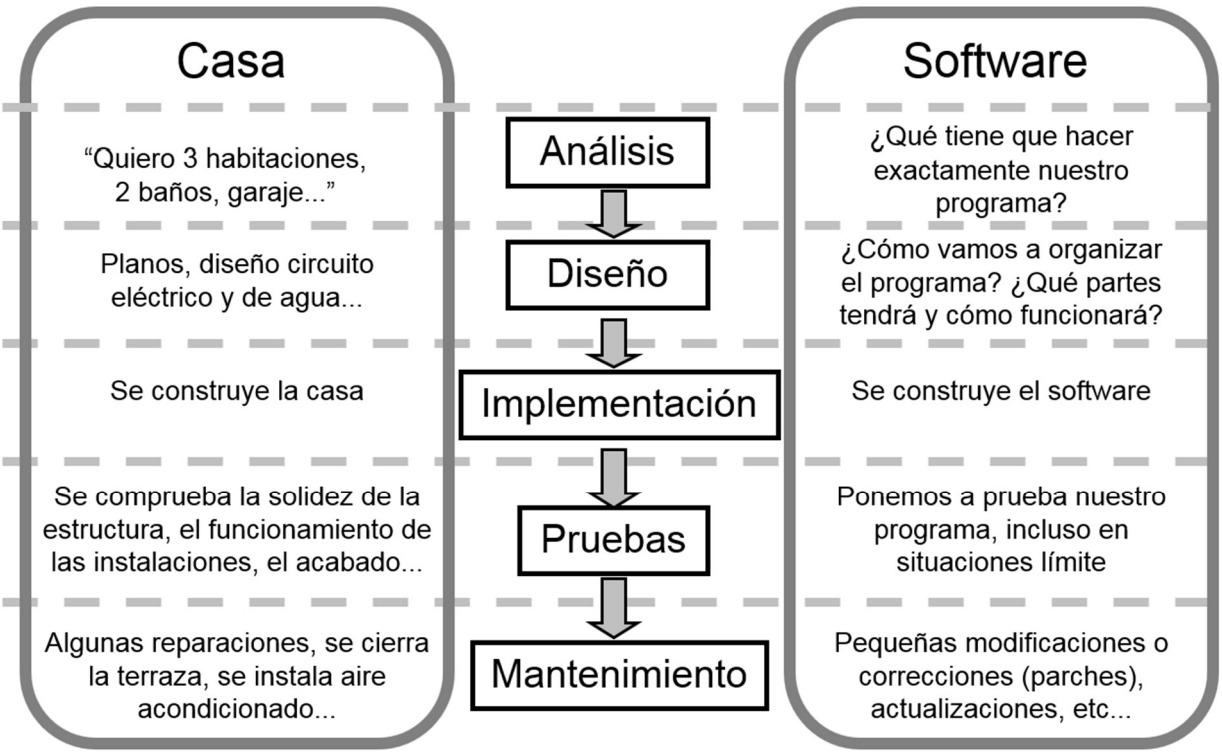
Ejemplos de código Java con comentarios Javadoc que proporcionan información adicional y permiten que algunas aplicaciones generen automáticamente páginas web usando la

información que extraen de los comentarios:

```
/** Este método calcula el número de reintentos de una
    determinada opción.
    @param opcion Cadena de texto con la opción a parsear
           para extraer el número de reintentos. La sintaxis es
           la siguiente: "reintentos:XXX", donde XXX será una
           cadena de texto que se interpretará como un
           número (valor retornado).
    @param similitud Flag que si vale true (valor por defecto)
           hará que el método no distinga las letras mayúsculas
           a la hora de intentar parsear la sintaxis, de modo
           que "reintentos:XXX" sea igual de válido que
           "ReinteNTos:XXX".
    @return Devuelve el número de reintentos o -1 en caso
            de error. */
int CalcularReintentos (const char *opcion,  bool similitud = true);
```

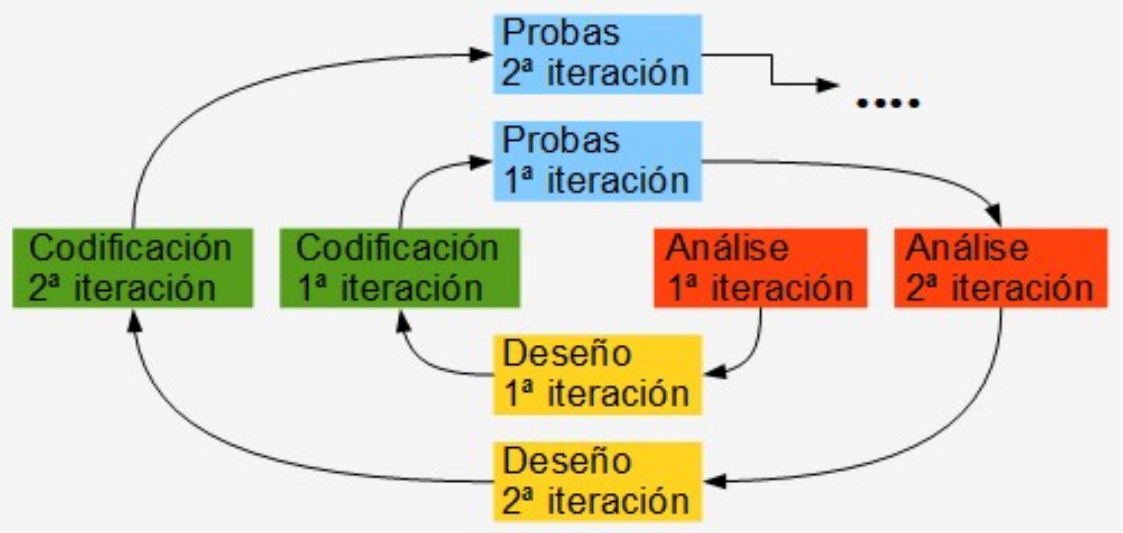
La documentación externa se adjunta al programa y se puede dirigir a:

- Programadores. Ejemplos: código fuente, interpretación de algoritmos complejos, especificaciones de datos y formatos de entrada/salida, lista de archivos utilizados o generados por la aplicación, lenguaje de programación, descripción de las condiciones o valores predeterminados utilizados,diagramas de diseño.
- El usuario. Ejemplos: requisitos del sistema (tipo de computadora que se ejecuta, sistema operativo requerido, recursos de hardware requeridos, etc.), detalles de instalación, una explicación de cómo utilizar mejor el software, una descripción de posibles errores de funcionamiento o de instalación y cómo corregirlos .
- La documentación propia es la documentación a la que se accede durante la ejecución del programa y puede contener: índice de contenido, guía de manejo del programa, asistente, ayuda contextual, autor del programa, versión, dirección de contacto, enlaces de referencia.



1.6 Modelo en espiral

El modelo se basa en la creación de prototipos de proyectos que se perfeccionan en iteraciones sucesivas a medida que se añaden nuevos requisitos, a través de procesos de análisis, diseño, codificación y prueba descritos en el modelo en cascada. Al final de cada iteración, el equipo de desarrollo de software y el cliente analizan el prototipo resultante y acuerdan si comenzar una nueva iteración. Siempre estás trabajando en un prototipo, por lo que cuando decides no hacer nuevas iteraciones y terminar el producto, tendrás que refinar el prototipo para obtener una versión finalmente terminada, estable y robusta.



1.7 Programación eXtrema[2]

La programación extrema o programación eXtreme es una metodología ágil de desarrollo de software que se basa en iteraciones en las fases de planificación, diseño, codificación y prueba.

Planificación

Cada reunión de planificación es coordinada por el Gerente de Proyecto y se celebra en:

- El cliente, sin el uso de ninguna tecnología o herramienta especial, explica en oraciones cortas las características que debe tener el software.
- El desarrollador de software convertirá cada entregable en una tarea que requiere hasta 3 días de tiempo óptimo de programación, de modo que el entregable completo no tome más de 3 semanas. Si estas cifras no se alcanzan, el rendimiento y las tareas se modifican de acuerdo con los deseos del cliente.donde se determina la cantidad de rendimientos que formarán parte de cada iteración, denominada velocidad del proyecto.
- Al final de cada iteración se lleva a cabo una reunión de planificación para permitir que el cliente evalúe los resultados; Si no lo acepta, debe agregar la funcionalidad no aceptada en la siguiente iteración, y el cliente necesita reorganizar la funcionalidad faltante para garantizar lavelocidad del proyecto.

Lo importante es la movilidad de las personas, es decir, en cada iteración los desarrolladores trabajan en diferentes partes del proyecto, de modo que cada dos o tres iteraciones los desarrolladores trabajan en varias partes del sistema.

Diseño

A diferencia del modelo en cascada, en esta etapa, cada objeto del sistema utiliza una tarjeta manual de tipo CRC (clase, Res-Ponabilidades, colaboración) que muestra el nombre de la clase, el nombre de la superclase, el nombre de la subclase, las responsabilidades de la clase y los objetos con los que colabora. Las tarjetas se colocan sobre una superficie para formar una estructura que refleja las dependencias entre ellas. Estas tarjetas serán terminadas y reposicionadas a mano a medida que avance el proyecto. Los desarrolladores se reúnen regularmente para comprender el conjunto y los detalles a través de las tarjetas.

CODIFICACIÓN Y PRUEBA

Una diferencia entre esta etapa y la etapa de codificación del modelo en cascada es que los desarrolladores deben acordar algunos estándares de codificación (nombres de variables, sangrías y alineaciones, etc.) y cumplir con estos estándares, ya que todos trabajarán en todo el proyecto.

Otra diferencia es que es aconsejable crear pruebas unitarias antes del propio código que se va a probar, ya que entonces tienes una idea más clara de lo que necesitas escribir.

La última diferencia es que se recomienda que los programadores desarrollen su trabajo en pares, ya que resulta que dos programadores trabajan juntos frente al mismo monitor y cada vez que pasan por teclado, ejecutan uno al lado del otro a la misma velocidad, pero el resultado final es de mucha mayor calidad, ya que mientras uno se enfoca en el método que

se está codificando, el otro piensa en cómo afecta el método a otros objetos, las dudas y sugerencias resultantes reducen drásticamente el número de errores y posteriores problemas de integración.

1.8 Indicadores v.3

Metrica Version 3 es una metodología de planificación, desarrollo y mantenimiento de sistemas de información promovida por el Departamento del Tesoro de los Estados Unidos y la Secretaría de Estado de Administración Pública que cubre el desarrollo estructurado y orientado a objetos.

La metodología se basa en el modelo del ciclo de vida del desarrollo propuesto en la norma ISO 12207, Tecnología de la información-Procesos del ciclo de vida del software.
Incluye tres procesos principales: planificación, desarrollo y mantenimiento. Cada proceso se divide en actividades que no se ejecutan necesariamente en secuencia, y cada actividad se divide en tareas. En el portal de gobierno electrónico (http://administracionelectronica. gob. es/) se puede acceder a un pdf detallando todas las metodologías y el personal informático involucrado en cada actividad.

Planificación

El proceso de planificación de sistemas de información (PSI) tiene por objeto proporcionar un marco de referencia para el desarrollo de sistemas de información acordes con los objetivos estratégicos de la organización, y la participación de la alta dirección de la organización es esencial. Las actividades incluyen:

- Descripción de la situación actual.
- Conjunto de modelos con arquitectura de información.
- Recomendar proyectos a desarrollar en los próximos años y las prioridades de cada uno de ellos.
- Propuesta de cronograma para la ejecución del proyecto.
- Evaluar los recursos necesarios para desarrollar los proyectos del próximo año.
- Programa de seguimiento y cumplimiento de la totalidad de la propuesta.

Desarrollo

Cada uno de los proyectos descritos en la planificación debe pasar por un proceso de desarrollo de sistemas de información, que consiste en las siguientes actividades:

Estudio de Viabilidad Sistemática (EVS), en el que se analizan los aspectos económicos, técnicos, legales y operativos del proceso y se toma la decisión de continuar o abandonar el proceso. En el primer caso será necesario describir la solución encontrada: descripción, costos, beneficios, riesgos, planificación de la solución. La solución puede ser el desarrollo de software hecho a medida, el uso de software estándar en el mercado, soluciones manuales o una combinación de ambos.

ANÁLISIS DE SISTEMAS DE INFORMACIÓN (ASI) para obtener una especificación de requisitos de software que contendrá las funcionalidades que proporcionará el sistema y las limitaciones a las que estará sujeto, para

Analice los casos de uso, las clases y las interacciones entre ellos para especificar la interfaz de usuario y desarrollar un plan de prueba.

- Diseño de Sistemas de Información (DSI), donde para cada caso de uso se realiza el diseño del comportamiento del sistema, diseño de la interfaz de usuario, diseño de clases, diseño de datos físicos (también diseño de migración de datos y carga inicial si es necesario), especificación técnica del plan de pruebas y establecimiento de requisitos de despliegue (despliegue del sistema, capacitación del usuario final, infraestructura, etc.).

Construcción de Sistemas de Información (CSI), en la que se prepara la base de datos física, se prepara el entorno de construcción, se genera el código, se realizan pruebas unitarias, de integración y pruebas del sistema, se redacta el manual de usuario, se define la capacitación del usuario final y se construyen los componentes y procedimientos para la migración y carga inicial de datos.

- Despliegue y aceptación del sistema (IAS), cuyo objetivo es entregar y aceptar todo el sistema y realizar todas las actividades necesarias para la transición a la producción. Para ello, se deben seguir los siguientes pasos: capacitar al equipo de implementación, capacitar a los usuarios finales, realizar la instalación, realizar la migración de datos y la carga inicial, realizar pruebas de implementación (verificar si el sistema está funcionando correctamente en el entorno operativo), realizar pruebas de aceptación del sistema (verificar si el sistema cumple con los requisitos iniciales del sistema), establecer los niveles de mantenimiento y servicio cuando el producto entre en producción. El último paso es el de producción, en el que se analizan los componentes necesarios para integrar el sistema en el entorno de producción, según las características y condiciones del entorno en el que se realizan las pruebas, y se instalan los componentes necesarios evaluando la necesidad de realizar una nueva carga de datos, inicialización o restauración, determinando la fecha de activación del sistema y eliminando los antiguos.

Mantenimiento

El objetivo de este proceso es obtener una nueva versión del sistema de información desarrollado utilizando la métrica 3 a partir de las solicitudes de mantenimiento realizadas por los usuarios debido a problemas detectados en el sistema o necesidades de mejora.

1.9 Metodología ágil: Scrum

La agilidad se define como la capacidad de reaccionar versátil al cambio y, por lo tanto, maximizar los beneficios. Respuesta rápida y buen efecto.

La metodología ágil es una forma de trabajar en equipo que requiere mucha interacción entre los miembros del equipo de desarrollo de software y el cliente para obtener resultados rápidos.

A continuación elaboramos la metodología Scrum, que es uno de los métodos más utilizados en la actualidad.

¿Qué es Scrum?

Scrum es una forma de trabajar, no un proceso, en el que se aplican de manera regular un conjunto de procesos para trabajar en equipo, y obtener el mejor resultado posible de un proyecto. Se aplica generalmente a desarrollo de software, pero puede ser aplicado para el desarrollo de cualquier pro- ducto que implique trabajo intelectual y relación con el cliente. Trabajos en cadena no requieren scrum.

Decimos que no es un proceso, dice “qué hay que hacer” y no “cómo hay que hacerlo”. La metodología es fácil de entender, lo importante es implantarla correctamente ya que la interacción entre los participantes es fundamental.

Ideas fundamentales del Scrum son:

- Adoptar una idea total de la realización del producto, en lugar de la planificación y ejecución completa del producto.
- Enfocarse más en las zonas de solapamiento, en lugar de realizar una tras otra en un ciclo de cascada. Entregas progresivas del producto a modo de prototipos.
- Pone por encima a los individuos y su interacción por encima de procesos y herramientas, colaboración con el cliente por encima de la negociación contractual.
- Eficaz respuesta ante cambios, ya sean cambios de requerimientos o incidencias.
- Otras metodologías se centran en la documentación, para hacer el proceso lo más independiente de las personas involucradas (cada uno tiene totalmente definido lo que tiene que ha- cer y cómo hacerlo). Scrum se centra en lo contrario, las personas, la comunicación, la inter- acción.

Historia

Este modelo fue identificado y definido por Ikujiro Nonaka e Hirotaka Takeuchi a principios de los 80, al analizar cómo desarrollaban los nuevos productos las principales empresas de manufactura tec- nológica.

En su estudio, Nonaka y Takeuchi compararon la nueva forma de trabajo en equipo, con el avance en formación de melé (scrum en inglés) de los jugadores de Rugby, a raíz de lo cual quedó acuñado el término “scrum” para referirse a ella.

¿Cómo se usa?

Con la metodología Scrum el cliente se entusiasma y se compromete con el proyecto dado que lo ve crecer parte por parte. Asimismo, le permite en cualquier momento realinear el software con los objetivos de negocio de su empresa, ya que puede introducir cambios funcionales o de prioridad en el inicio de cada nueva iteración sin ningún problema.

Esta forma de trabajo promueve la motivación y compromiso del equipo que forma parte del proyecto, por lo que los profesionales encuentran un ámbito propicio para desarrollar sus capacidades.

Los principios de la metodología Scrum son:

- Concentración
- Priorización
- Autoorganización
- Ritmo

Beneficios de Scrum

1. El cliente puede empezar a utilizar los resultados más importantes del proyecto antes de que esté finalizado por completo.
2. El cliente establece sus expectativas indicando el valor que le aporta cada requisito
3. Reducción de riesgos
4. Predicciones de tiempos
5. Mayor productividad

¿Qué es un Sprint?

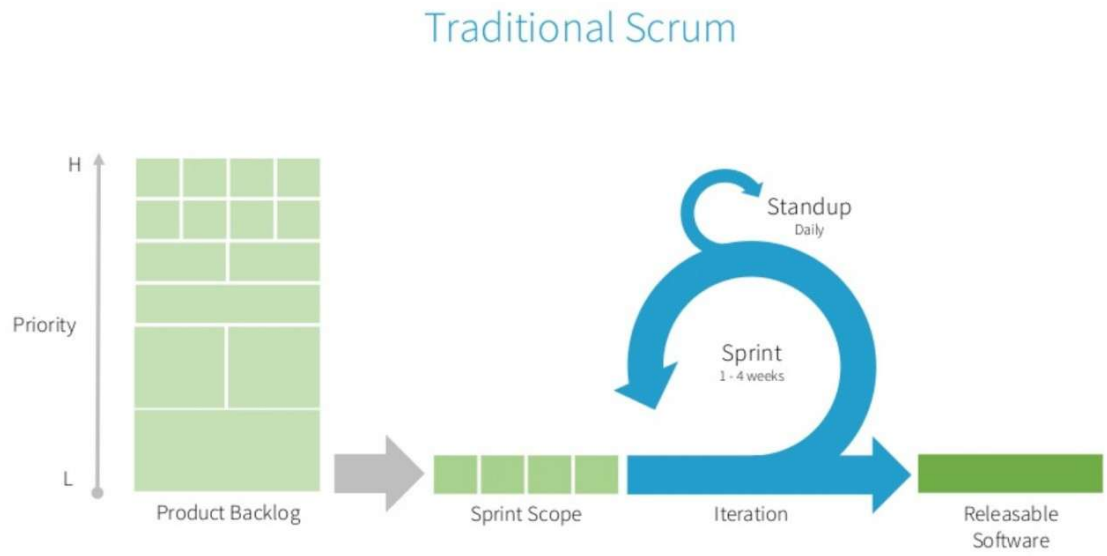
Sprint es el nombre que va a recibir cada uno de los ciclos o iteraciones que vamos a tener dentro de dentro de un proyecto Scrum.

Nos van a permitir tener un ritmo de trabajo con un tiempo prefijado, siendo la duración habitual de un Sprint va entre una y cuatro semanas, aunque lo que la metodología dice es que debería estar entre dos semanas y un máximo de dos meses.

En cada Sprint o cada ciclo de trabajo lo que vamos a conseguir es lo que se denomina un entregable o incremento del producto, que aporte valor al cliente.

La idea es que cuando tenemos un proyecto bastante largo, por ejemplo, un proyecto de 12 meses , vamos a poder dividir ese proyecto en doce Sprints de un mes cada uno. En cada uno de esos Sprints vamos a ir consiguiendo un producto, que siempre, y esto es muy importante, sea un producto que esté funcionando.

Vamos a verlo de forma más clara en esta imagen:

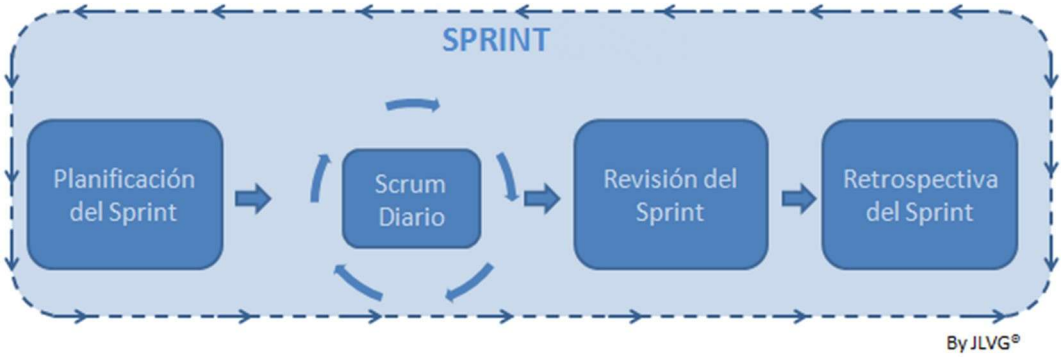


En la misma tenemos una pila o product backlog, que serían todos los requisitos que nos pide el cliente, es decir, el año completo de trabajo. La idea es ir seleccionando esos requisitos en los que tenemos la pila dividida, y los vamos a ir haciendo en diferentes Sprints, y realizamos todos los pasos que conforman un Sprint, es decir, la toma de requisitos, diseño, implementación, pruebas y desplie- gue en el plazo establecido, y así vamos a tener siempre un software que sea válido, un software funcionando.

Cuando hablamos de Sprint, hablamos de la parte más importante de Scrum, por eso vamos a ver las etapas y cómo se desarrollan en profundidad.

Etapa sprint

Cuando hablamos de Sprint en Scrum técnico, engloba todo el proceso, es decir, desde que decidi- mos qué vamos a hacer para ese Sprint, hasta que estudiamos cómo hemos trabajando en ese Sprint.



Cuando estamos en un proyecto Scrum para cada Sprint vamos a tener una serie de reuniones:

Reunión de Planificación del Sprint. Es la primera reunión del sprint, y en ella vamos a decidir lo que vamos a hacer y cómo lo vamos a hacer, el número de tareas o de historias de usuario que vamos a realizar en el Sprint.

Reuniones de Scrum diario, que van a ser pequeñas reuniones con los miembros del equipo.

Revisión del Sprint, en la que vamos a aceptar o denegar el Sprint.

Reunión de retrospectiva, dónde vamos a ver cómo ha trabajado el equipo y qué problemas ha tenido durante el desarrollo y cómo lo podemos corregir.

El Spring engloba todo lo anterior, desde que comienza el mismo hasta que es aceptado o denegado, y el equipo se pregunta cómo ha trabajado.

¿Qué es el Planning Póker?

El Planning Póker es un proceso de estimación de duración o esfuerzo en las tareas de un proyecto por parte de todo el equipo implicado. Para llevar a cabo esta técnica, cada participante cuenta con una baraja de cartas con números (basados en Fibonacci: 1,2,3,5,8,13, o similar) representando el grado de complejidad (esfuerzo) necesario para completar una tarea.

Comienza el cliente explicando un objetivo. De forma individual, cada miembro del equipo pone sobre la mesa la carta que representa la puntuación que él considera necesaria para llevar a cabo dicha tarea del proyecto en cuestión. De haber puntuaciones muy dispares, cada participante explica los motivos de su decisión. Y si fuese necesario, puede repetirse la votación hasta llegar a un consenso. Así, el resultado es una estimación consensuada y validada por todo el equipo.

Tiene muchas ventajas: todos los miembros del equipo expresan su opinión sin sentirse condicionados por el resto, se es más consciente del esfuerzo que requiere una tarea, lo que mejora la implicación. Al sentirse partícipes, el grado de compromiso con el proyecto también aumenta.

1.- Reunión de Planificación del Sprint

- Se lleva a cabo al inicio del ciclo
- Seleccionar qué trabajo se hará (tareas del Product backlog)
- El Product Owner presenta las funcionalidades (user stories) y sus prioridades (valoradas en story points). El equipo acuerda con él las actividades a desarrollar en ese Sprint. Tiene que

haber consenso (¿somos capaces de hacer todas esas actividades en el período que dura este sprint?)

- Esas actividades seleccionadas se almacenan en el Sprint Backlog y conforman el objetivo (Goal) del sprint actual. Hasta aquí la primera parte de la reunión.
- Ahora el equipo planifica las actividades, stories, y puede ser necesario volver a reestimar las actividades que se pueden llegar a realizar en este sprint. Cada una de estas actividades se "despiezan" en tareas que pueden ser desarrolladas por cada miembro técnico del equipo. Es cada miembro del equipo el que toma la tarea o tareas que va a realizar.
- Esta reunión tiene una duración máxima de ocho horas.
- A partir de este momento, todo el equipo se pone a trabajar en las tareas asignadas. Queda

claro que el feedback de la parte cliente, con el product owner, es una parte fundamental de esta metodología. El seguimiento constante impide que se llegue a una entrega final de producto y el cliente no esté satisfecho, como puede ocurrir con una metodología no ágil.

2.- Scrum diario

Es una reunión diaria con las siguientes características:

- La reunión comienza puntualmente a su hora.
- Todos los asistentes deben mantenerse de pie.
- La reunión debe ocurrir en la misma ubicación y a la misma hora todos los días.
- Tiempo límite de la reunión 15 minutos.
- Durante la reunión, cada miembro del equipo contesta a tres preguntas:

1. ¿Qué has hecho desde ayer?
2. ¿Qué es lo que estás planeando hacer hoy?
3. ¿Has tenido algún problema que te haya impedido alcanzar tu objetivo? Esta última pregunta puede modificar el trabajo de ese día de algún miembro del equipo, si esto es nece- sario para que otro miembro pueda realizar sus tareas.

3.- Reunión de Revisión del Sprint

Es una reunión para revisar el trabajo que fue completado y no completado.

Debe estar presente el Propietario del Producto. La relación con el cliente es más intensa que en metodologías tradicionales.

Presentar el trabajo al Propietario del Producto, mediante una demo para su aprobación. Aunque tenga algún error subsanable, el producto puede ser aceptado.

El trabajo incompleto no puede ser demostrado.

La falta de éxito de un sprint viene dada porque el producto presentado no corresponde a las necesidades del cliente o bien que no ha dado tiempo a resolverlo completamente. Habrá que tenerlo en cuenta para que no ocurra en el siguiente sprint.

Cuatro horas como límite.

4.- Revisión de Sprint

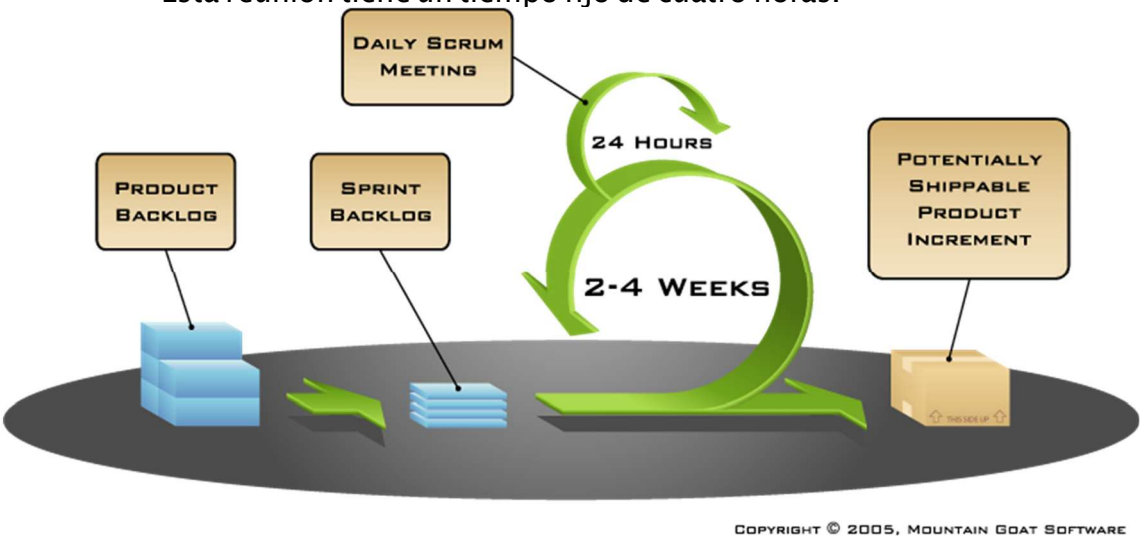
Después de cada sprint, se lleva a cabo una retrospectiva del sprint, en la cual todos los miembros del equipo dejan sus impresiones sobre el sprint recién superado. El propósito de la retrospectiva es realizar una mejora continua del proceso. Las tres preguntas de esta reunión son:

1. ¿En qué hemos mejorado desde nuestro último sprint? Es decir, que debemos seguir ha-

ciendo.

2. ¿En qué Podemos mejorar para los próximos sprints?
3. ¿Qué deberíamos parar de hacer?

Esta reunión tiene un tiempo fijo de cuatro horas.



Documentos del Scrum

Productos atrasados

El product backlog es un documento de alto nivel para todo el proyecto. Contiene descripciones ge - néricas de todos los requerimientos, funcionalidades deseables, etc. priorizadas según su valor para el negocio (business value). Es abierto y cualquiera puede modificarlo, aunque el Producto Owner su responsable. Es frecuente que al final de un sprint sufra alguna modificación.

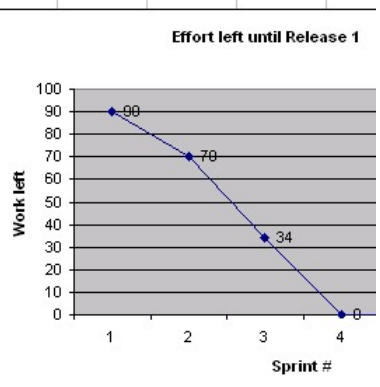
Weather on Mobile

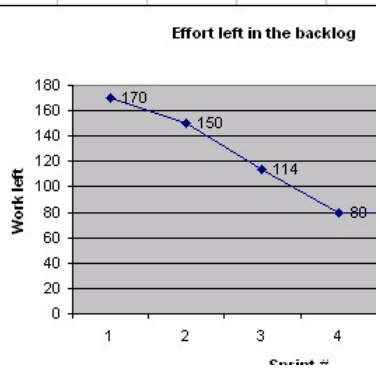
http://agilesoftwaredevelopment.com/scrum/simple-product-backlog

ID	Description	Sprint #	1	2	3	4	5	6
	Effort needed for Release 1 as in the beginning of the sprint		90	70	34	0	0	0
1	Set up continuous integration system		5	0	0	0	0	0
2	Create compilable application skeleton		5	0	0	0	0	0
3	Display current temperature in a simplest possible way		13	0	0	0	0	0
4	Set up the web server for serving weather data		3	0	0	0	0	0
5	Implement stubby WeatherML support on the server side		13	0	0	0	0	0
Sprint 1	*Make sample data go from server to device*							
6	Graphics support on the client side		20	0	0	0	0	0
16	Make the graphics library draw some icon and sample temperature text		-	13	0	0	0	0
17	Draw the real weather screen		-	8	0	0	0	0
7	Implement support for several days		8	8	0	0	0	0
8	Implement support for rain, snow, etc. icons		2	2	0	0	0	0
9	City changing support		-	5	0	0	0	0
Sprint 2	*Minimal working version*							
10	Fetch one day temperature data from the weather provider system		?					
11	Fetch rain, snow, etc details from the provider							
12	Fetch several days data from the provider							
13	Auto-refresh feature							
Sprint 3	*Plug in the real weather data*							
Release 1	**Sellable version**							
14	Inject simulated ads from the test server							
15	Plug real ads in							
18	Change current city automatically according to the cell info		40	40	40	40	40	40
Sprint 4	*Advertisements support*							
Release 2	**Ad-supported version**							
	Effort in the whole backlog		170	150	114	80	80	80

In a real product backlog, items are more likely to be in the form of user stories. E.g. this item could be stated as "As a user I want to see the real weather data so that I could know what to wear before leaving home"

Backlog state taken after the end of sprint 3 = after release 1





Sprint Backlog

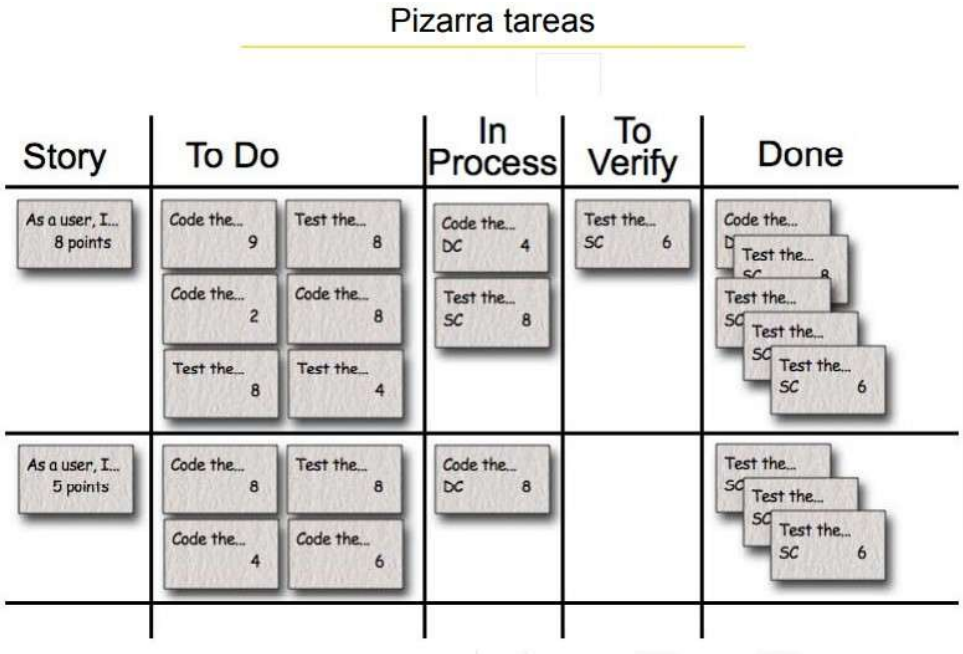
El sprint backlog es un documento detallado donde se describe el cómo el equipo va a implementar los requisitos durante el siguiente sprint. Consiste en tomar las actividades de mayor prioridad del Product Backlog. Las tareas se ordenan por prioridad, con esfuerzo estimado de dedicación.

Para medir el esfuerzo asignado a cada tarea se suelen asignar valores relativos, de forma que com- paramos una tarea con otra, ya que es más fácil que asignarle un número exacto de horas. Entre estas técnicas se puede usar la serie de Fibonnacci (1, 2, 3, 5, 8, 13, 21 34) para asignarle una puntua- ción a cada una, siendo 1 lo que asignamos a una tarea básica.

Las tareas en el sprint backlog nunca son asignadas, son tomadas por los miembros del equipo del modo que les parezca oportuno. Se puede añadir a este documento el porcentaje de tarea realizada, para ver de forma fácil y visual el progreso del sprint.

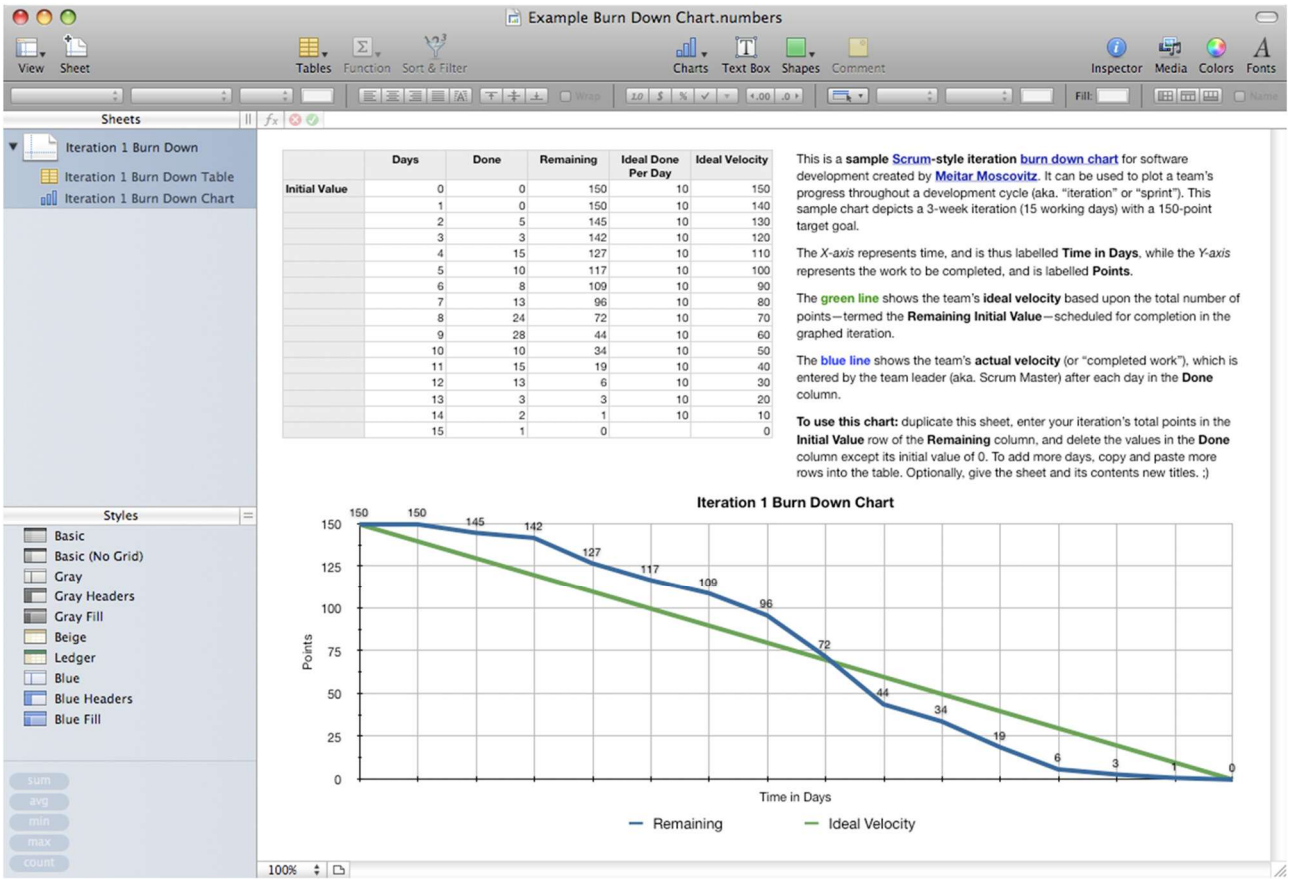
User Story	Tasks	Day 1	Day 2	Day 3	Day 4	Day 5	...
As a member, I can read profiles of other members so that I can find someone to date.	Code the ...	8	4	8	0		
	Design the ...	16	12	10	4		
	Meet with Mary about ...	8	16	16	11		
	Design the UI	12	6	0	0		
	Automate tests ...	4	4	1	0		
	Code the other ...	8	8	8	8		
As a member, I can update my billing information.	Update security tests	6	6	4	0		
	Design a solution to ...	12	6	0	0		
	Write test plan	8	8	4	0		
	Automate tests ...	12	12	10	6		
	Code the ...	8	8	8	4		

Otra forma de mostrar las tareas de sprint es mediante una pizarra de tareas:



Quemado

La burn down chart es una gráfica mostrada públicamente que mide la cantidad de requisitos en el Backlog del proyecto pendientes al comienzo de cada Sprint. Dibujando una línea que conecte los puntos de todos los Sprints completados, podremos ver el progreso del proyecto. Lo normal es que esta línea sea descendente, hasta llegar al eje horizontal, momento en el cual el proyecto se ha ter- minado.



En la figura superior de un Burn Down Chart, la línea azul representa la cantidad de requisitos pen - dientes y la línea verde representa la "velocidad ideal", basada en sprints previos. La velocidad se correspondería a la cantidad de requerimientos (user stories) del sprint, divididas por el número de días del mismo.

Roles

ScrumMaster

El ScrumMaster es el responsable de gestionar el proyecto, facilitar la comunicación entre los com- ponentes y que se sigan las pautas de la metodología. También tiene que resolver todas las inciden- cias que se vayan presentando.

El ScrumMaster tiene que explicar y convencer al quipo del uso de los procesos de Scrum, cuando en muchos casos no tiene autoridad jerárquica sobre ellos.

Las incidencias que tiene que resolver pueden estar relacionadas con el cliente, o con terceras partes involucradas en el producto, por ejemplo, proveedores.

Propietario del producto

El Product Owner representa la voz del cliente. Sabe exactamente qué es lo que el cliente quiere. En un mundo perfecto, sería el mismo cliente, pero en general suele ser un intermediario. El Product Owner tiene que conocer en detalle todos los requisitos que se tienen que desarrollar, con la priori- dad de cada uno de ellos y sus dependencias.
Debe ser también capaz de contestar a todo tipo de cuestiones relacionadas con el producto y es el responsable del mantenimiento del Product Backlog y la descripción de las actividades.

Otra responsabilidad es decidir, negociado con el equipo, las actividades de cada Sprint.

Al final de cada sprint, en la Reunión de Retrospectiva, el Product Owner aprueba o rechaza las características desarrolladas. Y también puede dar sugerencias para siguientes Sprints.

Finalmente, también es responsable de planificar las distintas entregas de producto, las distintas versiones o prototipos. Él sabe cómo tiene que ser el producto.

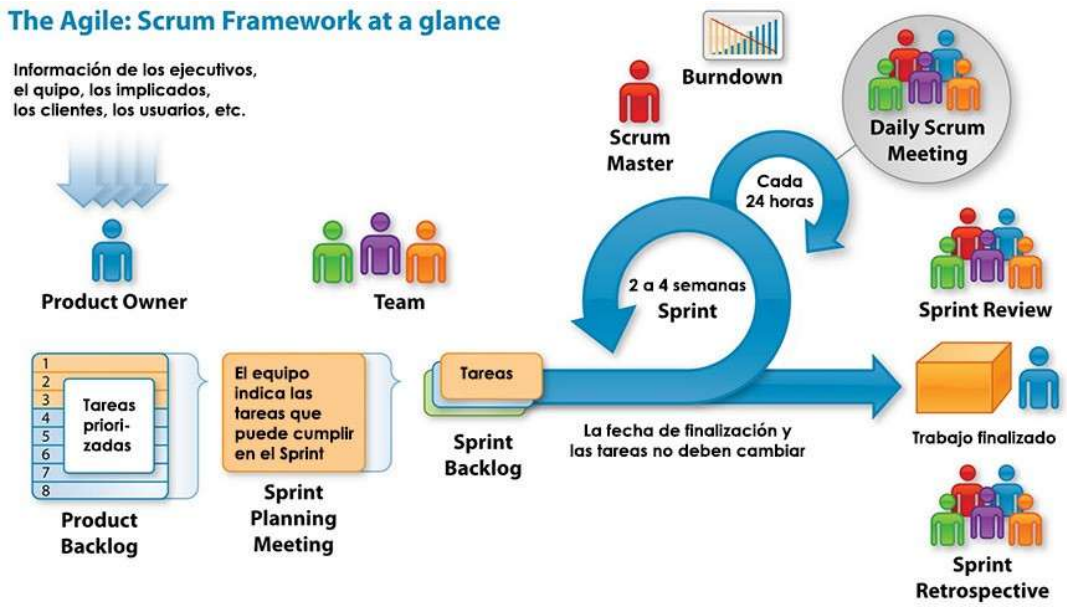
Tiene que haber mucha interacción con el equipo y retroalimentación mutua.

El equipo

Son las personas que desarrollan el equipo, son responsables de diseñar, desarrollar y probar el producto que están implementando.

El Product Owner alimenta con información al equipo y se asegura de que el equipo sigue el camino correcto y con la calidad adecuada.

El ScrumMaster lidera al equipo para que este siga la metodología Scrum.



Últimas Consideraciones

¿Scrum es adecuado para todos?

Definitivamente no. Algunas personas están demasiado acostumbradas a su forma de trabajar, y eso podría estar bien. Si lo que estás haciendo es bueno, puede que no haya ninguna razón para cambiar el proceso en absoluto. Sin embargo, si observa que los requisitos no se cumplen y que el producto final no es útil para el cliente, es posible que deba pensar más en usar Scrum.

¿Cuándo no usar Scrum?

Scrum no debe usarse para equipos pequeños, con un máximo de tres personas. En este caso, la experiencia ha demostrado que el marco no es tan útil por varias razones. Los miembros del equipo pueden hablar entre sí todo el tiempo, haciendo que las reuniones diarias no tengan sentido. Además, debido a que el equipo es bastante pequeño, las mejoras son fáciles de detectar y a menudo se implementan tan pronto como se identifican. Sin embargo, esto no significa que las buenas prácticas de desarrollo ágil no deban usarse. Esto simplemente significa que todo el marco Scrum es innecesario.

Otra situación en la que Scrum puede no ser necesario es que el alcance del producto es bien conocido y ciertamente no cambiará. Además, el equipo está bien informado de la tecnología utilizada, por lo que no hay sorpresas debido a la incertidumbre. Como se ha observado, esto es muy raro. Sin embargo, en este caso particular, no se requiere una retroalimentación continua, una mejora continua del proceso o un cambio de prioridad, ya que todo está bien determinado.

¿Scrum es perfecto, bala de oro?

Definitivamente no. Como se mencionó anteriormente, Scrum es difícil de poner en práctica debido a los posibles problemas de interacción humana. Además, es posible que las organizaciones no estén abiertas a todo lo necesario para

seguir Scrum. Además, Scrum no aborda el problema de las estimaciones a largo plazo. Todavía es muy difícil estimar con precisión un proyecto largo, lo que sigue siendo un problema para los gerentes que necesitan vender productos.

¿Cómo empiezo a usar Scrum?

Mi mejor consejo: paso a paso, en proyectos pequeños, preferiblemente con miembros que están acostumbrados a Scrum. Tal vez empiece con un Sprint, una reunión diaria y luego agregue un comentario. Lo más probable es que estos indiquen problemas en su proyecto, como errores en la comunicación y demasiadas características ya preparadas. Además, las sesiones diarias pueden no ser tan cortas como se espera, por lo que esto es algo que debe abordarse.

Más adelante, la introducción revisará qué procesos se mejorarán. Después de que el equipo

se acostumbra

a seguir todos los rituales de Scrum, es probable que el software en sí sea mejor. Sin embargo, notará que todavía hay espacio para mejoras, como pruebas automatizadas, mejor codificación y menos errores.

No te engañes: no es fácil, el cambio de mentalidad es dramático y algunas personas simplemente no se adaptan. Así que si crees que Scrum vale la pena, no te frustres e intenta lograrlo.

Además, recuerde que no adoptar todo el marco de Scrum no es algo malo. ¡Lo que funciona para ti es bueno! Tal vez en su empresa, la propiedad funciona mejor para una persona que para un equipo. O tal vez los sprints tienen que cambiar su marco de tiempo. Es importante utilizar las herramientas que Scrum ofrece para mejorar su proceso de desarrollo de software y hacer que su equipo sea ágil.

¿Es inútil el proceso de cascada?

Definitivamente no. No olvides el proceso de cascada, principalmente los relacionados con el CMM han existido durante mucho tiempo. A pesar de sus problemas ciertos, como se ha discutido ampliamente aquí, muchas empresas han tenido un gran éxito con ellas, y un número considerable de ellas todavía lo siguen siendo. Además, recuerde que no había nada para organizar el desarrollo de software antes de ellos, por lo que fueron un gran avance porque los procesos similares a CMM proporcionaron métodos para el desarrollo de software por primera vez.

Estos métodos permiten pensar y discutir los requisitos antes de la implementación del software, en lugar de la confusión absoluta que precede. Además, en mi opinión, la mayor ventaja: permitir tiempo para la arquitectura y el diseño del software, lo que mejora enormemente la calidad del software. Además, el diseño y la arquitectura permiten el desarrollo y el uso de patrones de diseño y buenas prácticas, lo que una vez más mejora enormemente la calidad del software.

Además, se desarrollaron estrategias de pruebas, así como automatización y pruebas unitarias,

que nuevamente contribuyeron a la mejora del software.

Así que el mensaje aquí es: no tomen a la ligera las metodologías de software de la vieja escuela, ya que nos dan grandes ideas para un buen desarrollo de software. Además: la arquitectura, las pruebas automatizadas, los patrones de diseño y más siguen siendo muy adecuados para los métodos ágiles.

Piense en los métodos ágiles como un paso en la dirección correcta para el desarrollo de software. Finalmente, no hay duda de que algún día empezaremos a ver problemas con los métodos ágiles y

, con suerte, salga algo mejor. Entonces, después de eso, surgen los problemas y el ciclo

continúa para siempre.

1.10 Licencias de software

Una licencia de software es un contrato entre el autor del software y el usuario final que lo utilizará. Los términos de este contrato establecen:

- -Condiciones de instalación y uso: tanto el costo como quién puede utilizarlo, en qué equipo, etc.
- -Posibilidad de modificar, transferir y distribuir el software.
- -El tiempo y la región en la que se aplica la licencia, ya que cada país o región puede tener leyes diferentes.
- -Responsabilidad por errores de software
- ——Atribución (diga quién es el autor)

Hay muchas licencias diferentes, y la primera gran división será software propietario o software libre con derechos de autor y no propietario.

Software propietario

No tenemos acceso a su código fuente, por lo que es accesible por sus desarrolladores y no puede ser modificado ni distribuido sin el permiso expreso de su titular. En general, los usuarios tendrán derecho a ejecutar el software y dentro de ciertas condiciones.

Software privado y dos tipos especiales que no debemos confundir con software libre son o (el software es libre, pero tiene derechos de autor y por lo tanto no puede ser modificado o distribuido) y o Shareware(El software gratuito también tiene derechos de autor y tiene restricciones en el teiempo de uso y las funciones. Generalmente, los programas comprados por los usuarios después de la prueba no están sujetos a restricciones).

Software libre

El software libre es el software en el que el titular de los derechos permite al usuario utilizar, alterar y redistribuir el software de cualquier manera, de cualquier manera y para cualquier propósito, independientemente de la forma modificada.

Original. No es que sea gratis. Quere Para decir respeto a la libertad de los usuarios, en Específicamente, los usuarios tienen cuatro libertades esenciales:

- Libertad para ejecutar programas para cualquier propósito (Libertad 0).
- Libertad para estudiar cómo funciona un programa y modificarlo (Libertad 1). Visitas El código fuente es necesario para lograr esto.
- La libertad de redistribuir copias para ayudar a otros (libertad 2).
- Libertad de distribuir sus versiones modificadas a terceros para (Libertad 3). Esto da a toda la sociedad la oportunidad de beneficiarse del cambio.

En otras palabras, el "software libre" es una cuestión de gratuidad, no de precio (la confusión

puede surgir del doble significado de "gratis": gratuito y gratuito). Cuando el usuario no controla el programa, decimos que no es "libre", o que es "privado". Pero incluso si lo piensas al revés, el software libre no tiene que ser gratuito, y de hecho hay compañías que cobran por distribuir copias de software libre.

Software libre con y sin copyleft: Hacer ambos son tipos de software libre. El primero (con copyleft) no puede agregar restricciones al redistribuir el programa, y el segundo (sin copyleft) si se pueden agregar restricciones, algunas modificaciones pueden no ser gratuitas en absoluto y convertirse en software propietario.

GPL-GNU

La GPL (General Public License), también conocida como GNU General Public License, es una licencia de código abierto con copyleft que los usuarios pueden usar, investigar, compartir y modificar Software, siempre y cuando los productos derivados sigan bajo licencia GPL. Consiste en unconjunto específico de cláusulas de distribución para publicar programas con copyleft. El Proyecto GNU utiliza esta licencia para la mayoría de los programas que publica. Esta licencia fue desarrollada por la Free Software Foundation.

Fuente abierta (Open Source)

En la práctica es lo mismo que el software libre, solo en algunas matizas de licencia muy específicas. La diferencia es más filosófica: la idea del software libre aboga por la libertad del usuario para la libertad y la equidad, mientras que la idea del código abierto se centra principalmente en las ventajas prácticas de este tipo de software y no aboga por principios.

Dominio público

Es el software que no tiene derechos de autor. Se El código fuente es público y podemos Digamos que también es software libre, pero no viceversa.

Por otro lado, la mayoría del software libre no está en el dominio público, de lo contrario está bajo sus derechos de autor, pero este último da una licencia legal para su uso libre. Por lo tanto , no confunda el dominio público (sin derechos de autor) con el software libre (con derechos de autor pero con licencia de uso libre)

> Misión 1.6. Busque en línea qué es Creative Commons y los tipos de licencias que lo componen (incluya diferentes iconos que representan diferentes licencias en la respuesta). Finalmente, busque la página de recursos gratuitos para incluir en nuestro programa: imágenes, música, etc. y comente sus requisitos de uso.

2. Lenguajes de programación y herramientas de desarrollo

2.1 Clasificación de los lenguajes informáticos

Un lenguaje informático es un lenguaje que permite la comunicación con un ordenador y consiste en un conjunto de símbolos y palabras que siguen ciertas reglas gramaticales.

No existe una clasificación de los lenguajes informáticos adoptados por la mayoría de los autores, a menos que exista una gran diferencia entre ellos. Una posible clasificación es: lenguaje de marcado, especificación, consulta, transformación y programación.

Lenguaje de marcado

Permiten colocar en el texto insignias o logotipos que serán interpretados como diferentes por la aplicación o proceso. Ejemplos de lenguaje de marcado:

- Lenguaje XML (eXtensible Markup Language), que es un metalenguaje extensible diseñado para la transmisión de información estructurada y que puede ser validado.

Ejemplo de código XML editado en NetBeans:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Exemplo básico de XML
-->
<alumnos>
    <alumno>
        <nome>Pepe</nome>
        <apelidos>Ruíz Arias</apelidos>
    </alumno>
    <alumno>
        <nome>María Dolores</nome>
        <apelidos>González Paz</apelidos>
    </alumno>
</alumnos>
```

-HTML (lenguaje de marcado de hipertexto) o XHTML (lenguaje de marcado de hipertexto extensible) = XML + HTML, ambos lenguajes se utilizan para publicar hipertexto en la World Wide Web. El marcado modifica la apariencia del contenido, por ejemplo, dibujando tablas, tablas, etc.

Ejemplo de código XHTML editado en Notepad++:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Exemplo moi básico de XHTML con estilo externo -->
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Exemplo básico de xhtml</title>
        <meta   content="text/html; charset=utf-8"
                http-equiv="Content-Type" />
        <link   rel="stylesheet" href="exemplo_css.css"
                type="text/css" />
    </head>
    <body>
        <h1>Cabeceira principal</h1>
        <p class="principal">Este párrafo contén texto e un enlace a
            <a href="http://www.realacademiagalega.org/">Real Academia Galega</a>
        </p>
    </body></html>
```

Lenguaje canónico

Describen algo de forma precisa. Por ejemplo, CSS (Cascading Style Sheet) es un lenguaje formal que especifica cómo se renderizan o estilan documentos HTML, XML o XHTML. Los ejemplos de código CSS se pueden aplicar a los ejemplos XHTML editados anteriormente en Notepad ++:

```
body{
    font-family: "MS Sans Serif", Geneva, sans-serif;
    font-size: 15px;
    color: Black;
    border:black 2px double;
    padding: 40px;
    margin: 20px;}
h1 {
    font: 40px "Times New Roman", Times, serif;
    font-weight: bolder;
    word-spacing: 25px;}
p.principal{
    text-align: center;
    font: 10px "MS Serif", "New York", serif;}
```

Lenguaje de consulta

Permiten extraer o manipular información a partir de un conjunto de información, generalmente todos estos grupos de información son bases de datos. Por ejemplo, el lenguaje de consulta SQL (Standard Query Language) permite la búsqueda y manipulación de información en bases de datos relacionales, y el lenguaje XQuery permite la búsqueda y manipulación de información en bases de datos XML nativas. Ejemplo de script SQL:

```
CREATE DATABASE  `empresa`;

USE `empresa`;

CREATE TABLE `centros` (
  `cen_num` int(11) NOT NULL default '0',
  `cen_nom` char(30) default NULL,
  `cen_dir` char(30) default NULL,
  UNIQUE KEY `numcen` (`cen_num`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `centros` VALUES
    (10,'SEDE CENTRAL','C/ ALCALA, 820-MADRID'),
    (20,'RELACION CON CLIENTES','C/ ATOCHA, 405-MADRID');
 CREATE TABLE `deptos` (
  `dep_num` int(11) NOT NULL default '0',
  `dep_cen` int(11) NOT NULL default '0',
  `dep_dire` int(11) NOT NULL default '0',
  `dep_tipodir` char(1) default NULL,
  `dep_presu` decimal(9,2) default NULL,
  `dep_depen` int(11) default NULL,
  `dep_nom` char(20) default NULL,
  UNIQUE KEY `numdep` (`dep_num`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `deptos` VALUES
    (122,10,350,'F','60000.00',120,'PROCESO DE DATOS'),
    (121,10,110,'P','200000.00',120,'PERSONAL'),
    (120,10,150,'P','30000.00',100,'ORGANIZACION'),
    (112,20,270,'F','90000.00',110,'SECTOR SERVICIOS'),
    (111,20,400,'P','111000.00',110,'SECTOR INDUSTRIAL'),
    (200,20,600,'F','80000.00',100,'TRANSPORTES'),
    (100,10,260,'P','120000.00',NULL,'DIRECCION GENERAL');
```

Ejemplos de expresiones FLOWR en XQuery (for, let, order by, where, return) (https://es.wikipedia.org/wiki/XQuery):

```
Para  $libro  en  doc("books.xml")/bib/book
let $editorial: = $libro/editorial
Donde  $editorial="MCGRAW/HILL  "o  contiene  ($editorial,"Toxosoutos")
Volver a $Reservar
```

Lenguaje de transformación

Actúan sobre un mensaje inicial para obtener otro mensaje nuevo. Por ejemplo, XSLT (Extensible Style Sheet Language Transformations: https:// ~~es.wikipedia.org/wiki/extensible_styles-heet_language_transformations) le~~ permite describir las transformaciones que se realizarán en un documento XML para obtener otro archivo. Un simple ejemplo de transformación XSL editado en NetBeans, cuando manipula el ejemplo XML anterior, obtiene una página HTML con una lista de estudiantes:

```
<? ¿Versión xml = "1.0" codificación = "UTF-8"? >
<! --Ejemplo simple de transformación XSL-->
<  xsl:stylesheet  xmlns:  xsl="http://www.w3.org/1999/xsl/Transform  "
                Versión = "1.0" >
    < xsl:output method="html"/ >
    < xsl:template match="alumnos " >
        < html >
            < cabeza >
                < title>fundamentos.xsl</title >
            < /cabeza >
```

```
            < cuerpo principal >
                < h1>Alumnios</h1 >
                < ul >
                    < xsl:for-each select="alumno " >
                        < li >
                            < xsl:value-of select="apelidos"/ >,
                            < xsl:value-of select="nome " / >
                        < /li >
                    < /xsl : para cada uno >
                < /ul >
            < /cuerpo >
        < /html >
    < /xsl:plantilla >
< /xsl : hoja de estilo >
```

Lenguaje de programación

Permiten la comunicación con un dispositivo de hardware, permitiendo así la ejecución de un procedimiento dado, para lo cual pueden procesar estructuras de datos almacenadas en memorias internas o externas, y utilizar estructuras de control. Tienen vocabulario y están sujetos a reglas sintácticas y semánticas.

El vocabulario es una colección de símbolos que se pueden utilizar y pueden ser: identificadores (nombres de variables, tipos de datos, nombres de métodos, etc.), constantes, variables, operadores, instrucciones y comentarios.

Las reglas sintácticas especifican la secuencia de símbolos que forman una oración bien escrita

en este lenguaje, es decir, para que no tenga errores ortográficos.

Las reglas semánticas definen la construcción sintáctica y cómo deben ser las expresiones y los tipos de datos utilizados.

Ejemplo de código Java simple para editar en NetBeans:

```
Paquete parimpar;

Importar java.util.Scanner;
Clase pública Main {
    public static void main(String [] args) {
        Tú n;
Teclado del escáner = nuevo escáner (System.in);
System.out.print("Escriba un número entero "); n=
teclado.nextInt ();
        Si (n%2==0) {
System.out.println(n + "es par");}

        De lo contrario {
System.out.println(n + "es un número impar");}

    }
}
```

2.2 Clasificación de los lenguajes de programación

Los lenguajes de programación se pueden clasificar por proximidad al hardware, generación, paradigma de programación, cómo se traduce y ejecuta el lenguaje de máquina, y arquitectura cliente- servidor.

Ordenar por distancia desde el hardware

Se pueden dividir en lenguajes de bajo y alto nivel.

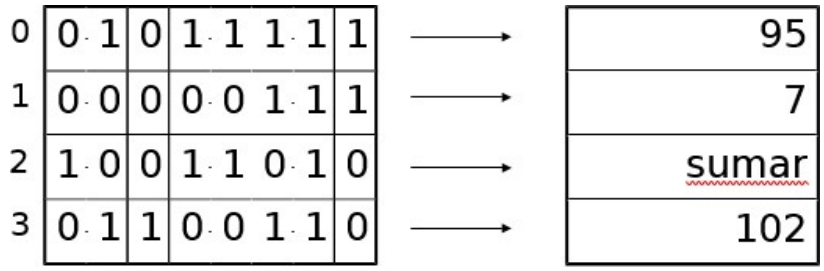
Idiomas de bajo nivel

Se basan directamente en los circuitos electrónicos de la máquina, por lo que un programa escrito para una máquina no podrá ser utilizado en otra máquina diferente. Pueden ser lenguaje de máquina o lenguaje ensamblador.

El lenguaje de máquina es un código binario (cero y uno) o hexadecimal (números 0 a 9 y letras

A a F) que actúa directamente sobre el hardware. Es el único idioma que no requiere traducción

, ya que el procesador puede reconocer directamente las instrucciones.



La codificación en lenguaje ensamblador es memotécnica, es decir, el uso de etiquetas para describir ciertas operaciones. El código de ensamblaje debe traducirse al lenguaje de la máquina para que el procesador pueda reconocer las instrucciones.

Ejemplos:

```
. Modelo pequeño
. pila
. Fecha
Cadena1 DB THola Mundo.$T
. Código

Procedimiento:
mov ax,@data mov ds, ax
mov dx, offset chain 1
mov ah, 9 int 21h mov ah, 4ch int 21h
Fin del programa
```

Decir que tiene la ventaja de ser rápido, que es el lenguaje del ordenador en sí, pero decirle que tiene la desventaja de ser muy propenso a errores, complicado de programar y solo para tareas

muy específicas de programar ciertos microprocesadores.

Misión 1.7. Busque en la web un fragmento de código escrito en lenguaje ensamblador.

Idiomas de alto nivel

Están tratando de acercarse al lenguaje humano y separarse del conocimiento interno de la máquina, por lo que necesitan traducir el lenguaje de la máquina. Esta traducción hace que se ejecute más lentamente que los lenguajes de bajo nivel, pero puede ejecutarse en diferentes computadoras debido a que no depende del procesador.

Ordenar por generación

Se pueden clasificar en 5 generaciones.

1. La primera generación (1GL) consiste en los lenguajes de programación utilizados en las primeras computadoras: lenguaje de máquina y lenguaje ensamblador.
2. Segunda generación (2GL) formada por lenguajes macroensambladores, es decir, lenguajes ensambladores que combinan instrucciones de procesamiento y control de datos más complejas. Estos lenguajes son específicos para una familia de procesadores y el hardware asociado con ellos. Todavía se utiliza para programar el kernel del sistema operativo y los controladores de ciertos dispositivos.
3. La tercera generación (3GL) formada por la mayoría de los lenguajes actuales de alto nivel. El código no tiene nada que ver con las máquinas y los lenguajes de programación son similares al lenguaje humano. Por ejemplo, Java, C, C++, PHP, JavaScript y Visual Basic.
4. Cuarta generación (4GL), que consiste en lenguajes y esquemas diseñados para tareas o propósitos muy específicos, como el acceso a bases de datos, generación de informes, generación de interfaces de usuario, etc. Por ejemplo, SQL, Informix 4GL y Progress 4GL.
5. La quinta generación (5GL) consiste en lenguajes en los que el programador establece los problemas a resolver y las condiciones a cumplir. Se utilizan en inteligencia artificial, sistemas basados en el conocimiento, sistemas expertos, mecanismos de inferencia o procesamiento del lenguaje natural. Ejemplos son Prolog, Smalltalk y Lisp.

Clasificación por paradigma de programación

Un paradigma de programación es una metodología o filosofía de programación que se sigue con un núcleo central incuestionable, es decir, un lenguaje que utiliza el mismo paradigma de programación programará con los mismos conceptos básicos. La evolución de los métodos de programación y los lenguajes ha ido de la mano con el tiempo. Se pueden dividir en dos grandes grupos: aquellos que siguen el paradigma imperativo y aquellos que siguen el paradigma declarativo.

Imperativo

El código consiste en una serie de pasos o instrucciones para completar una tarea organizando o cambiando los valores en la memoria. Las instrucciones se ejecutan en secuencia, es decir, no pasan a ejecutar la siguiente hasta que se haya ejecutado una. Por ejemplo, Java, C, C++, PHP, JavaScript y Visual Basic.

Dentro del lenguaje imperativo, se distingue entre aquellos que siguen una metodología estructurada y aquellos que siguen una metodología orientada a objetos.

A finales de la década de 1960, nació el enfoque estructurado, que permitió tres estructuras en el código: secuencial, alternativo (basado en decisiones) y repetitivo (cíclico). Los programas están hechos de datos y esas estructuras.

La **metodología estructurada** se desarrolló mediante la adición de módulos como componentes básicos, dando lugar a la programación estructurada y modular. Un programa consiste en módulos o procesos, por un lado, y datos, por el otro. Estos módulos requieren algunos datos de entrada y obtienen otros datos de salida, que a su vez pueden ser utilizados por otros módulos. C, por ejemplo, es un lenguaje estructurado y modular.

En la década de 1980 surgió una **metodología orientada a objetos** con objetos como elemento básico. Esta metodología comenzó a popularizarse a principios de la década de 1990 y actualmente es la más ampliamente utilizada. Por ejemplo, C ++ y Java son lenguajes orientados a objetos. Cada objeto sigue un patrón especificado en una clase. Los objetos incluidos en el programa están relacionados o colaboran con otros objetos de la misma clase u otras clases. Una clase consiste en propiedades (datos) y métodos (procedimientos) y puede tener las siguientes propiedades:

- Patrimonio. Puede declarar una subclase que hereda las propiedades y métodos de la clase padre. Los métodos heredados se pueden sobrecargar, es decir, el mismo método que define diferentes comportamientos puede aparecer dentro de la misma clase, con diferentes firmas (el mismo número y tipo de parámetros).

- Polimorfismo. Una propiedad que permite que un método se comporte de manera diferente dependiendo del objeto al que se aplica.

- Encapsulación. Permite ocultar los detalles internos de una clase.

Paradigma de declaración.

El código indica que es lo que quieres conseguir, no cómo tienes que conseguirlo, es decir, es un conjunto de condiciones, proposiciones, declaraciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución. Los programadores necesitan pensar en la lógica del algoritmo, no en el orden en el que se ejecuta. Por ejemplo, SQL, Lisp y Prolog son lenguajes declarativos.

En los últimos años, la programación declarativa se ha vuelto cada vez más popular, por lo que muchos lenguajes que normalmente usan el paradigma imperativo han incorporado funcionalidad declarativa. Java introdujo un "streaming" que permite esta codificación en su versión 8 (2014).

En el siguiente ejemplo, vemos la misma tarea desarrollada en Java con programación imperativa y programación declarativa (filtrar solo números pares de una colección de números y guardar su cuadrado):

```java
//programación imperativa
public class Main {
  public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        List<Integer> result = new ArrayList<>();

        for (Integer number : numbers) {
            if (number % 2 == 0) {
                result.add(number * number);
            }
        }
        System.out.println("Resultado con for (Imperativo): " + result);
  }
}
```

```java
//programación declarativa
public class Main {
  public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        List<Integer> result = numbers.stream()
                                      .filter(n -> n % 2 == 0)
                                      .map(n -> n * n)
                                      .collect(Collectors.toList());
        System.out.println("Resultado con Streams (Declarativo): " + result);
  }

}
```

Clasificado por traducción en lenguaje de máquina y modo de ejecución

El llamado código fuente se refiere al código escrito en un lenguaje de programación simbólico a través de herramientas de edición. Este código se guarda en un archivo conocido como archivo fuente, que debe ser traducido al lenguaje de la máquina para poder ejecutarse. La traducción puede hacerse a través de un compilador y/o intérprete.

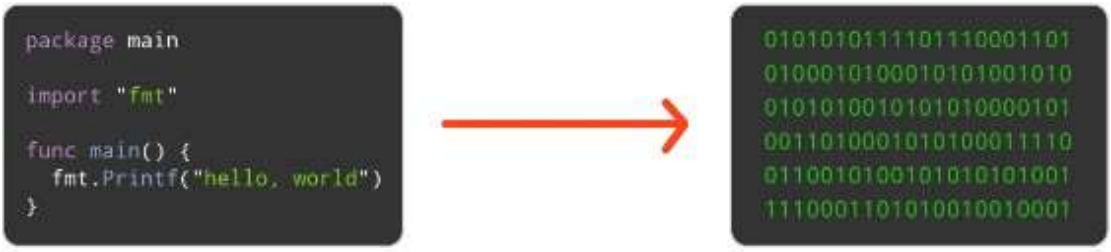
Teniendo en cuenta cómo se traduce y ejecuta el lenguaje de máquina, se puede pensar que hay tres grupos de lenguajes de programación: compilados, interpretados y lenguajes de ejecución administrados o de máquina virtual. Algunos lenguajes como Prolog pueden considerarse compilados o interpretados

porque tienen un compilador o intérprete.

Lenguaje compilado

Estos lenguajes tienen compiladores o programas que traducen el código fuente en código de máquina mediante la creación de archivos ejecutables en tiempo de compilación. En tiempo de ejecución, el ejecutable se puede iniciar en la plataforma que sirve el compilador. Algunas características
Dichos idiomas son:

- Hay un ejecutable de una máquina real.
- El proceso de traducción se divide en tiempo de compilación y una sola vez.
- La velocidad de ejecución es muy rápida porque el ejecutable es un lenguaje de máquina.
- El ejecutable solo se ejecuta en la plataforma en la que fue creado.
- El usuario en ejecución no necesita conocer el código fuente, solo necesita tener el código ejecutable No se puede manipular fácilmente para obtener el código fuente, y como resultado, el programador tiene el código fuente más protegido.
- Si hay errores léxicos, sintácticos o semánticos, no se generan ejecutables.
- La interrupción de la ejecución puede crear dificultades para el sistema operativo y puede afectar a la plataforma.
- La modificación del código fuente implica la regeneración del ejecutable.



Lenguaje de alto nivel que entiende el programador

Lenguaje de máquina que entiende el procesador

Lenguaje interpretado

Estos lenguajes requieren un intérprete o un programa en la memoria para traducir el código fuente al lenguaje de la máquina en tiempo de ejecución. Algunas de las características de este tipo de lenguaje son:

- No hay ejecutables.
- El proceso de traducción se realiza en cada ejecución.
- La ejecución es lenta, ya que se debe añadir la traducción al proceso de ejecución.
- Los archivos se pueden ejecutar en diferentes plataformas siempre que haya

un intérprete disponible.

- El usuario ejecuta el programa usando el código fuente.
- Posibles errores de tipo léxico, sintáctico o semántico en tiempo de ejecución.
- La ejecución de interrupciones suele afectar solo al intérprete y no a la plataforma.
- Se puede ejecutar en cualquier plataforma, siempre y cuando exista su intérprete.
- Modificar el código fuente no requiere ninguna operación adicional antes de que el programa se ejecute.

Máquina virtual o lenguaje de ejecución administrado

Estos lenguajes requieren inicialmente un compilador que traduce el código fuente en código intermedio multiplataforma mientras se programan, y luego otro software para traducir este código intermedio en código de máquina. En el caso de lenguajes como Java, sucede:

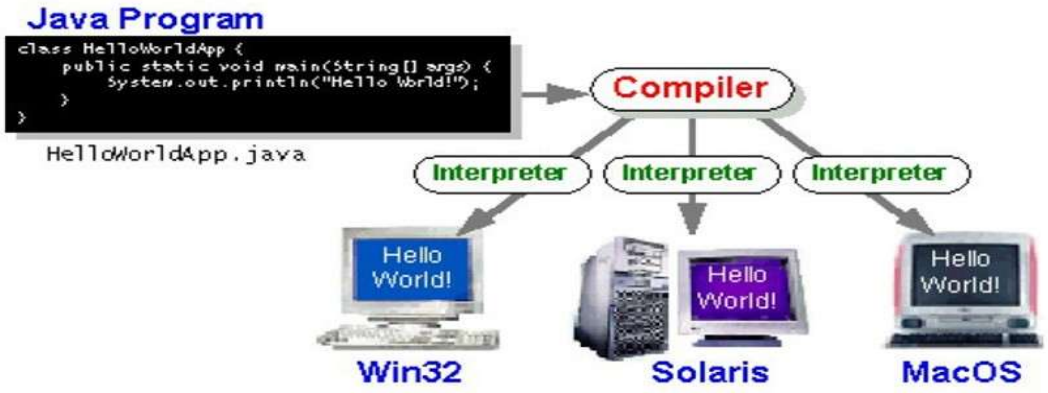
- En el momento de la compilación, el compilador Java genera un código intermedio llamado bytecode Java independiente de la plataforma.
- En tiempo de ejecución, necesita una máquina virtual Java (JVM) que interpreta el bytecode. Esta máquina virtual es un software que simula una máquina real con su propio sistema operativo y la hace un intermediario con la máquina real, por lo que las máquinas virtuales son diferentes para Linux, Windows, Mac y Solaris.

En el caso de un lenguaje de ejecución alojado, como la plataforma Microsoft. NET, ocurre

- En el momento de la compilación, el compilador genera código CIL (Common Intermediate Language) independiente de la plataforma.
- Para ejecutarlo hay que contar con la versión correcta de. NET Framework en la máquina cliente (por lo tanto también CRL) y luego realizar la última fase de compilación, donde CRL traducirá el código intermedio en código de máquina a través del compilador JIT (Just In Time). Algunas de las características de este tipo de máquina virtual administrada o lenguaje de ejecución son:
- La presencia de código intermedio que se ejecuta en un software específico pero no es un archivo ejecutable.
- El proceso de traducción inicial se completa antes de la ejecución y el proceso de traducción final se completa en cada ejecución.
- Ejecución más lenta que los lenguajes compilados, pero más rápida que los lenguajes interpretados.
- El archivo se puede ejecutar en diferentes plataformas siempre que exista un

software específico.

- El usuario no tiene código fuente, sólo el código intermedio que no puede ser manipulado fácilmente puede obtener el código fuente, de esta manera el programador tiene el código fuente más protegido.
- Se detectan errores de tipo léxico, sintáctico o semántico durante la fase de compilación y, si existen, no se genera código intermedio.
- La ejecución de interrupciones suele afectar solo al intérprete y no a la plataforma.
- Modificar el código fuente implica repetir el proceso de compilación.



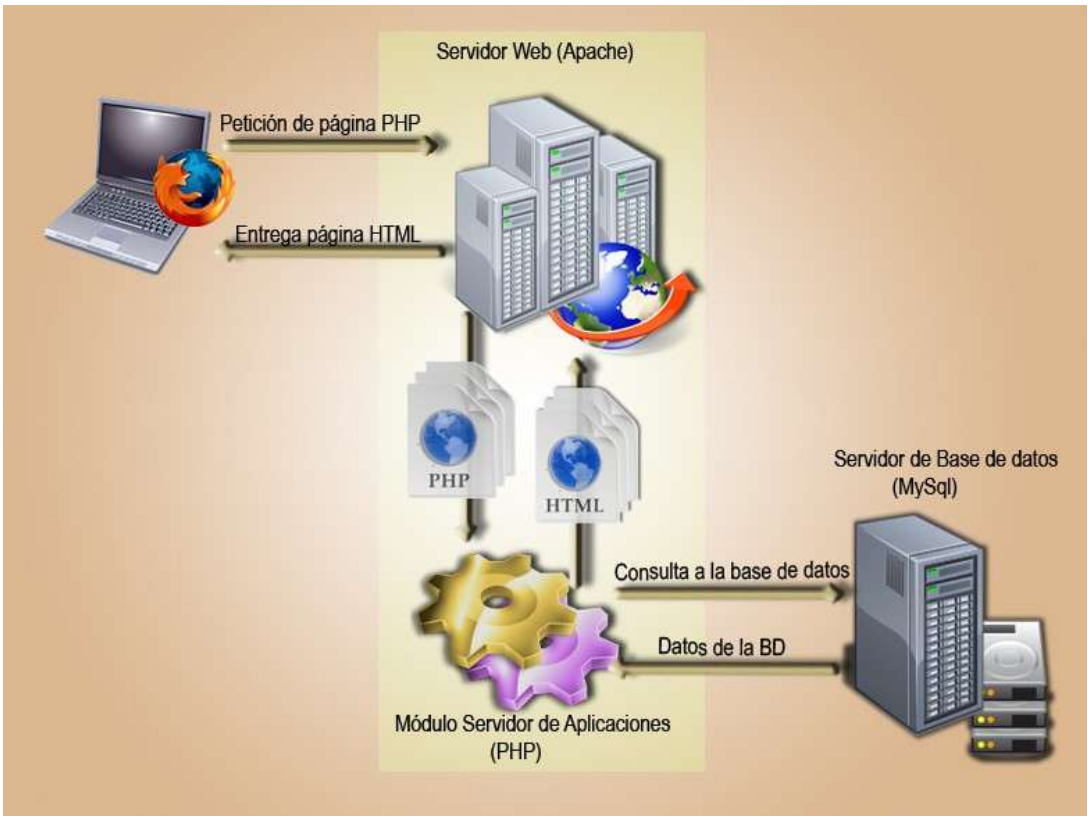
Operación de la arquitectura cliente-servidor

La arquitectura cliente- servidor consiste básicamente en un programa cliente que hace una solicitud al servidor. Esta arquitectura es más útil cuando el cliente y el servidor se comunican a través de una red, aunque también se pueden aplicar cuando están en la misma máquina. Este es el esquema utilizado por todas las aplicaciones web y hemos distinguido claramente entre dos tipos de esquemas:

- Cliente, como JavaScript (con HTML y CSS)
- En el lado del servidor, con lenguajes como PHP, C# o Java (y JavaScript con servidor Node.js)
- Para aclarar, estos lenguajes generalmente se usan con "frameworks" o frameworks, pero discutiremos esto más adelante.
- Para ilustrar esto, podemos suponer un navegador cliente con un intérprete PHP y un servidor web, como se muestra en los siguientes pasos:
- El usuario solicita una página HTML desde el navegador del cliente al servidor Web con código PHP y código JavaScript incrustados en ella.
- El servidor Web procesa la solicitud, interpreta el código PHP, se ejecuta colocando el resultado de la ejecución en el sitio donde se encuentra el código PHP y devuelve al cliente la página Web que contiene el código HTML y JavaScript. El servidor Web debe ejecutar comandos relacionados con el procesamiento de la base de datos en el servidor de base de datos, si tales instrucciones están disponibles en el código PHP.
- El navegador cliente interpreta el código JavaScript y muestra la página al

usuario.

- El usuario puede interactuar con la página de tal manera que cada vez que se hace una solicitud al servidor, la página recarga completamente la respuesta del servidor.



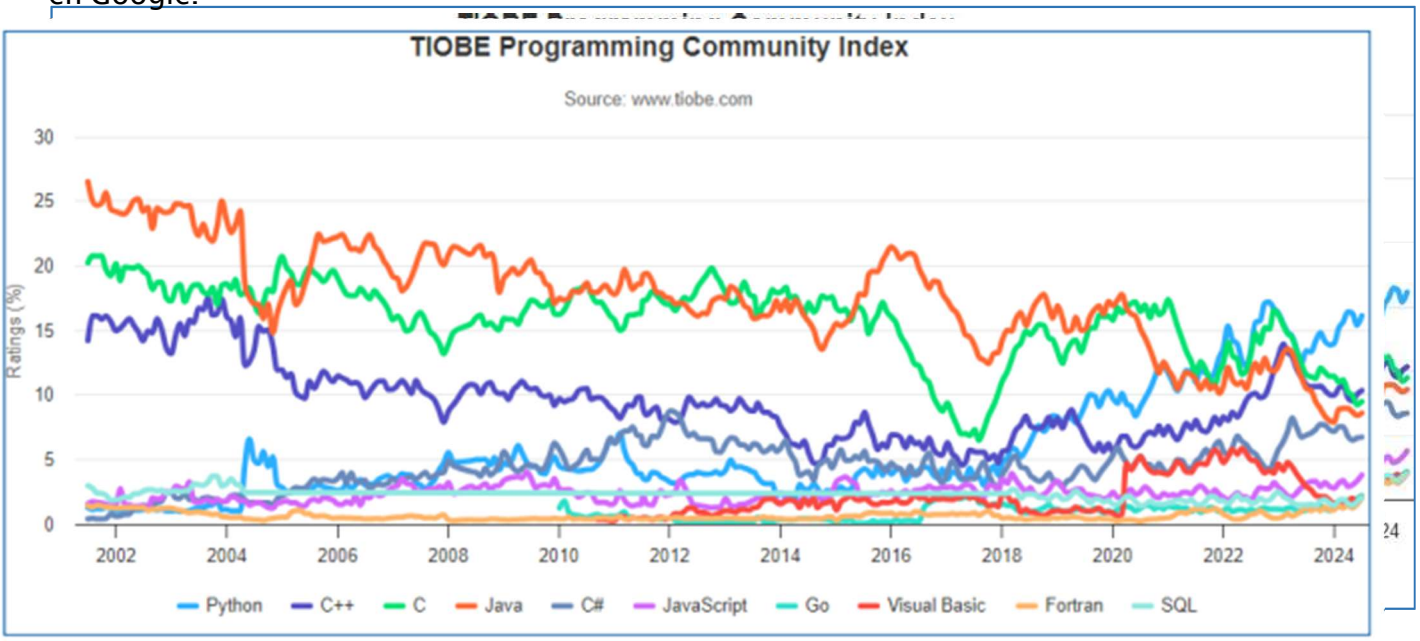
- El usuario final nunca podrá ver el código PHP o Java porque el servidor web lo ha convertido a HTML. Sin embargo, si puedes ver el código JavaScript, porque esto se ejecuta en la computadora del usuario, no en el servidor.Existe unha técnica denominada AJAX (Asynchronous JavaScript And XML), que permite que JavaS- cript procese algún evento iniciado polo usuario facendo unha petición ao servidor que este res- ponde co resultado en XML. JavaScript permite la interacción asíncrona entre servidor y cliente mediante la actualización de partes de la página para procesar la salida XML sin tener que volver a cargarla completamente.

Misión 1.8. Visite la página de Instagram (u otra) y compruebe si hay código Javascript.

2.3 Los lenguajes más usados

Para saber cuáles son los idiomas más hablados, puede consultar los siguientes índices o clasificaciones:

- Índice Tiobe (http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html), que se basa en el número de ingenieros cualificados en cada idioma en todo el mundo, los cursos de idiomas ofrecidos, los proveedores que trabajan en el idioma, las búsquedas realizadas en Google, Bing, Yahoo, Wikipedia, Amazon, YouTube y Baidu, y una serie de premisas como la presencia del lenguaje como lenguaje de programación en Wikipedia y al menos 10000 visitas en Google.



- Índice PYPL o Índice de Popularidad de Lenguajes de Programación (http://pypl.github.io/PYPL. html), que clasifica los lenguajes de programación basándose en el análisis de la frecuencia de búsqueda de títulos o guías del lenguaje utilizando Google Trends.
- Ranking Redmonk (https://redmonk.com/sogrady/2024/03/08/language-rankings-1-24/), que no se basa en motores de búsqueda, sino en proyectos alojados en repositorios de GitHub y problemas web de StackOverflow para programadores. Incluye lenguajes informáticos, no solo lenguajes de programación.
- El ranking de Trendyskills (https://trendyskills.com/) se basa en vacantes en lenguajes de programación de Estados Unidos, Reino Unido, Alemania, Suecia, España, Países Bajos, Irlanda, Suiza, Austria, Bélgica, Finlandia, República Checa, India y Grecia, incluyendo lenguajes informáticos y no solo lenguajes de programación.

C

C es un lenguaje creado por Bell Labs en 1972 para codificar el sistema operativo UNIX. Está clasificado como un lenguaje de alto nivel, estructurado y modular, pero tiene muchas características de bajo nivel, como el uso de punteros para hacer referencia a la ubicación física de la RAM. Estas características le permiten trabajar muy cerca de la máquina, pero el programa es mucho más complejo y propenso a más errores. Ha influido en el diseño de lenguajes como C++, C#, Java, PHP, JavaScript, entre otros. Ejemplo de código C:

```
/* Programa C para sumar números enteros de 1 a 20
 */
# incluyendo < stdio.h >
```

```
# incluyendo < stdlib.h >
int main(int argc, char**argv) {
    int i; /* Contador de enteros */
    Long int suma; /* Sumador de números enteros */
    Eres el último; Suma = 0;
    Para (i = 1; i < = 20; i ++) {
Importe = Importe i;
    }
printf("Programa    C\ntotal    e:%ld\n    ",    total);
printf("\nEscriba cualquier número para terminar... ");
scanf("%d ", & final);
fflush (stdin);
    Return (EXIT_SUCCESS);
}
```

C++

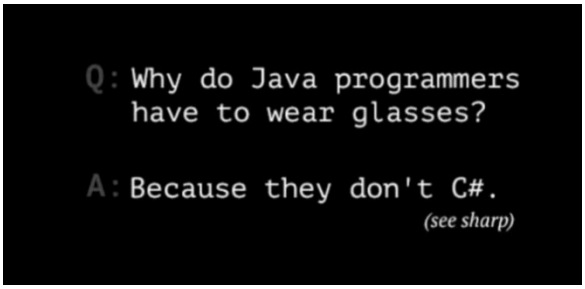
Diseñado por Bjarne Stroustrup en 1980 para extender C con mecanismos que permiten POO (programación orientada a objetos). Ejemplo de código C++ editado en NetBeans:

```
/*
* Programa C++ para sumar números enteros de 1 a 20
 */
#incluir < iostream >
int main(int argc, char**argv) {
    int i; /* Contador de enteros */
    Long int suma; /* Sumador de números enteros */
Suma = 0;
    Para (i = 1; i < = 20; i ++) {
Importe = Importe i;
    }
std::cout<<"a     suma     e<<<sum<<std::endl;
    Devuelve 0;
}
```

C#

Es un lenguaje de programación orientado a objetos desarrollado por Microsoft en 2000 como parte de la plataforma. NET. Visual C# ofrece un editor de código avanzado, un diseñador de interfaz de usuario y una gran cantidad de herramientas para facilitar el desarrollo de aplicaciones en C# y. NET Framework. Ejemplo de código C# creado con Visual Studio:

```
El uso del sistema;
Ejemplo de espacio de nombre_CSharp_Console {
    Programa de clase {
        Estático void Main(cadena [] args)
{
Console.   WriteLine("    Hola    Mundo    ");
        }
    }
}
```



Java

Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems en 1995. Utiliza mucha sintaxis de C y C ++, pero adopta un modelo de objetos más simple y elimina las herramientas de bajo nivel utilizadas en C. Ejemplos de código Java editado en NetBeans: paquete sumainteriros;

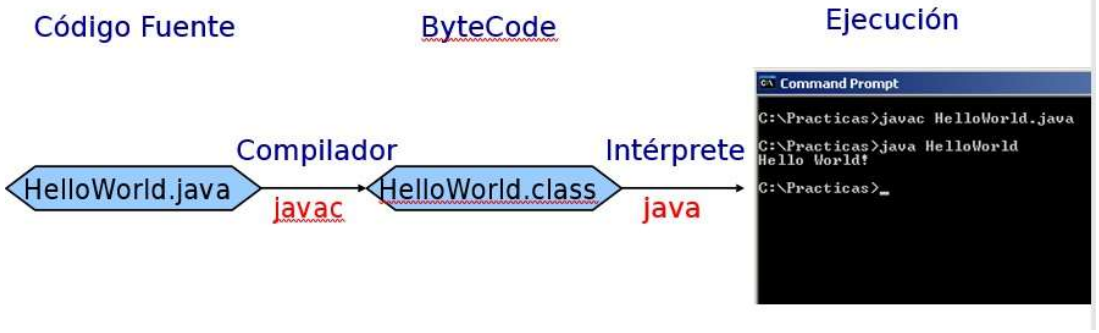
```
/**
* Archivo: Main.java
* Autor: Profesor
* Fecha: 14/08/2020 12:25:00
* Objetivo: ver la suma de los 20 primeros números naturales
 */


Clase pública Main {

    public static void main(String [] args) {
        Suma de números enteros = 0; /* Sumador de números enteros */
        Para (int i=1;i<=20;i++)
        {
Suma = suma + i;
        }
System.out.println("Ejemplos de Java ");
System.out.println("Suma de los primeros 20 números naturales = "+ sum);}

}
```

No sigue un patrón, ni compila ni interpreta. Tiene una compilación inicial que genera un código intermedio, casi ejecutable. Un intérprete (para la máquina virtual Java) luego ejecuta el código línea por línea. Esto logra la independencia de la plataforma (cada plataforma necesitará desarrollar su propia máquina virtual, pero el código fuente se aplicará a todas las plataformas).



Como puede ver en imaxe anterior, el archivo final a ejecutar será. class, que es el bytecode que interpreta la máquina virtual. En realidad, las aplicaciones Java se distribuyen en archivos de tipo. jar, que son archivos comprimidos que solo contienen bytecode y no los demás archivos que componen la aplicación: imágenes, audio, etc.
Es uno de los lenguajes más utilizados en la actualidad, en una variedad de dispositivos diferentes. La idea de una máquina virtual permite que los programas escritos en Java se ejecuten en diferentes hardware.

PHP

PHP (Hypertext PreProcessor) es un lenguaje interpretado del lado del servidor que se puede utilizar para crear cualquier tipo de programa, pero su lugar más popular es para crear páginas web dinámicas. El código PHP parece estar incrustado en el código HTML o XHTML y requiere un cliente para hacer una solicitud al servidor y un servidor web para ejecutar la solicitud.

Diseñado por Rasmus Lerdorf en 1995 y todavía desarrollado por el grupo PHP

(http://php.net/) Ejemplo de código PHP incrustado en una página XHTML editada con Note-Pad ++:

```
<! DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transicional//ES"
"http://www.w3.org/TR/xhtml1/dtd/xhtml1-transitional.dtd" >
< html xmlns="http://www.w3.org/1999/xhtml " >
    < cabeza >
        < title>Ejemplos de php</title >
        < meta content="text/html;charset=utf-8 " http-equiv="Content-Type"/>
    </head >
    < cuerpo principal >
        <? php
        $sum=0;
        Par ($i=1; $i<=20; $i=$i+1)
        á   $suma=$suma+$i;
        }
        Eco   < p>La cantidad total será de: ". $suma."</p >";
        ? >
    < /cuerpo >
< /html >
```

Python

Es un lenguaje de programación interpretado que permite la programación orientada a objetos con una sintaxis muy concisa que favorece la legibilidad del código. Fue diseñado por Guido Van Rossum en 1991 y actualmente es administrado por la Python Software Foundation (http://www.python.org/). Tiene una licencia de código abierto llamada Python Software Foundation License 1 y es compatible con la Licencia Pública General de GNU a partir de la versión 2.1.1. Es uno de los lenguajes en ascenso y ocupa una posición de liderazgo en el entorno de IA. Ejemplo de código de Python:

```
# Suma enteros de 1 a 20
def sumarEnteiros(n):
Suma = 0
    Para i en el intervalo (1, n+1):
sum=sum+i

    Devuelve la suma
A=sumarEnteiros(20)

Imprimir "Suma de números enteros de 1 a 20:"+str (A)
```

JavaScript

Es un lenguaje de programación dialectal para el estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, tipado débilmente (declaración de tipo) y dinámico. Generalmente se utiliza en el lado del cliente y se implementa como parte de un navegador web , aunque también existe JavaScript del lado del servidor (Server-side JavaScript o SSJS). Ejemplo de código JavaScript:

```
<! Tipo de documento >
< html xmlns="http://www.w3.org/1999/xhtml " >
    < head><title>Ejemplo de JavaScript</title >
          < meta content="text/html;charset=utf-8 " http-equiv="Content-Type"/>
    </head >
    < body><h1>Número natural impar ata 9</h1 > <
          script type="text/javascript " >
            <! --
            Transformarme;
            Para (i = 1; i < = 10; i +
= 2) document.write(i + "");
```

2.4 Proceso de generación de código

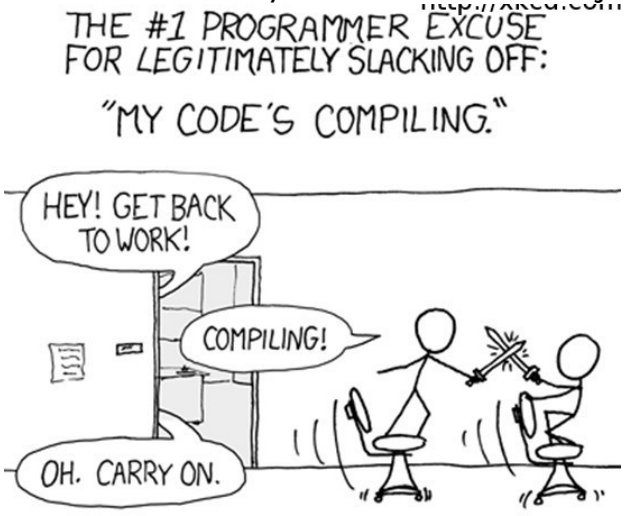
La generación de código consiste en procesos de edición, compilación y vinculación. Una vez completado el código, puede ejecutarlo para producir resultados.

Edición

Esta etapa consiste en escribir algoritmos de análisis en lenguajes de programación utilizando un editor de texto o herramientas de edición incluidas en el entorno de desarrollo. El código generado se denomina código fuente y el archivo correspondiente se denomina archivo fuente. Aunque el archivo tiene extensiones como. java,. c,. py, etc. dependiendo del idioma, siempre es texto sin formato (txt).
Los editores ampliamente utilizados (excepto para entornos de desarrollo) pueden ser DevC++ o Code::Blocks for C, Pycharm para Python o, más generalmente, Sublime Text. Estos editores completan el código, resaltando las diferentes estructuras del lenguaje con diferentes colores, algunos de ellos permiten que la compilación y la ejecución se realicen desde el mismo editor.

Compilación

Analiza y sintetiza el código fuente a través del compilador. Si no se encuentran errores, se obtiene el código objeto o el código intermedio multiplataforma. La gente no entiende tal código y no puede ejecutarlo directamente.
Esta etapa no se aplicará a los lenguajes interpretados, aunque éstos pueden tener herramientas que permitan el análisis léxico y sintáctico antes de la ejecución.



ANALISIS

Los análisis realizados incluyen:

- Análisis léxico, comprobando si los símbolos utilizados son correctos, incluidos los espacios.

Análisis sintáctico, en el que se comprueba que la agrupación de símbolos cumpla con las reglas lingüísticas.

- Análisis semántico en el que se realiza el resto de comprobaciones, como si la variable utilizada está declarada o no, o la coherencia entre el tipo de datos de la variable y el valor almacenado en ella, o la comparación del número y tipo de parámetros entre la definición y la llamada al método.

Síntesis

Esta síntesis permite:

- Generación de código intermedio independiente de la máquina. Algunos lenguajes compilados como C primero pasan por una fase de preprocesamiento en la que se realizan operaciones tales como reemplazar constantes con valores o incluir archivos de encabezado. Otros lenguajes, como Java, pueden ejecutar bytecode Java en la JVM, mientras que otros, como C #, pueden ejecutar código CIL en la CLR.
- Traducir el código intermedio anterior en el código de máquina para obtener el código objeto. Esta traducción también trae consigo una optimización del código. Este nuevo código no está listo para ejecutarse directamente.

Enlace

Esta etapa vincula el archivo de objeto obtenido en la compilación al módulo de objeto externo a través de un programa de enlace para obtener un archivo ejecutable si no se encuentra ningún error. El archivo de objeto resultante de la compilación puede contener referencias al código de objeto externo que es parte de una biblioteca externa estática o dinámica:

- Si las bibliotecas son estáticas, el vinculador agrega el código objeto de esas bibliotecas al archivo objeto, por lo que el ejecutable resultante aumenta en tamaño con respecto al archivo objeto, pero solo requiere el sistema operativo para ejecutarse.
- Si la biblioteca es dinámica (Biblioteca de enlace dinámico-DLL en Windows, objetos compartidos en Linux), el vinculador solo agrega referencias a la biblioteca, por lo que el ejecutable resultante apenas aumenta en tamaño con respecto al archivo objeto, pero la biblioteca dinámica debe ser accesible mientras el ejecutable se está ejecutando.

Ejecución

La ejecución requiere diferentes herramientas, dependiendo de si el lenguaje es interpretado, compilado o una máquina virtual o ejecución administrada.

Si el lenguaje se interpreta, entonces se necesitará un archivo fuente y un intérprete, lo que se hace traduciendo cada instrucción en el archivo fuente al lenguaje de máquina (análisis y síntesis) y ejecutándolo.

Por ejemplo: PHP.

Si el lenguaje es compilado, se necesitarán archivos ejecutables y, en algunos casos, bibliotecas dinámicas. Por ejemplo: C.

Si el lenguaje requiere una máquina virtual, entonces es necesario tener un código intermedio y una máquina virtual para que este último traduzca el código intermedio al lenguaje de la máquina y lo ejecute. Ejemplo: programas para plataformas Java o Android. Este último utiliza Java (adaptación) y una máquina virtual Dalvik (DVM) o una máquina ART (Android Runtime) con una estructura diferente a la JVM para interpretar el código intermedio.

Esquema de desarrollo integrado de IDE

El entorno de desarrollo integrado (Integrated Development Environment) es una aplicación que facilita las tareas anteriores. Soportan una variedad de lenguajes de programación diferentes, tanto nativos como implementados a través de complementos. Incluye las siguientes características:

-Editor de código fuente con ayuda para resaltar diferentes estructuras de código, completar texto automáticamente, detectar errores de sintaxis, etc.

-El depurador de código (debugger) permite ejecutar paso a paso el código escrito, permitiendo observar en cualquier momento el estado de los datos y variables del programa.

-Utilidades para generar interfaces gráficas de una manera sencilla e intuitiva.

-Control de versiones de los programas de desarrollo.

-Herramientas adicionales para facilitar otras tareas: migraciones, pruebas, etc.

Entre los más populares que admiten varios idiomas son Visual Studio Code de Microsoft y IntelliJ de JetBrains.

Misión 1.10. Preparar una presentación que describa brevemente la diferencia entre lenguajes compilados, interpretados y semi-compilados (basados en un programa sencillo que suma dos números en lenguajes C, Python y Java, respectivamente) y que muestre con capturas de pantalla el software involucrado en estos tres casos y vea el proceso completo de edición, compilación (si es así), ejecución (en estos tres casos).

Axuda a la tarea

2.5 Framework

Un marco (plataforma, entorno, marco de desarrollo rápido de aplicaciones) para los programadores sobre la base del cual podemos desarrollar proyectos sin tener que empezar desde cero. etc. Es una plataforma de software en la que se definen programas de soporte, bibliotecas, lenguajes interpretados que ayudan a desarrollar y conectar diferentes módulos o partes de un proyecto.

Mediante el uso de un framework, podemos dedicar más tiempo a analizar los requisitos del sistema y las especificaciones técnicas de la aplicación, ya que se resuelve la tediosa tarea de programar los detalles.

Ventajas del uso de marcos:

El rápido desarrollo del software.

Reutilice partes de código de otras aplicaciones.

o Deseño uniforme del software.

Inconvenientes:

El código depende en gran medida del marco utilizado (si no cambiamos el marco, una

gran parte de la aplicación tendrá que reescribirse).

Un nuevo entorno de aprendizaje, además del idioma que usted utiliza.

o Instalar e implementar el marco en nuestro equipo consume recursos del sistema.

Misión 1.11. Busque Internet en general, y Infojobs en particular, los marcos más demandados, agregue una pequeña descripción a ellos y haga una correspondencia con el lenguaje que utilizan. Puede utilizar el lenguaje que enfatizó en la tarea 1.9. Debes incluir al menos estos frameworks: Spring/ SpringBoot, Laravel, Symphony, CodeIgniter, Django, ASP. NET, Express, Angular, React.js, React Native, Vue, Flutter.