APUNTES EDD TEMA 1

//ACÁ ESTARÁ TODO EL CONTENIDO QUE PUEDE CAER EN EL EXAMEN

1. Conceptos básicos

Qué es un programa: es un conjunto de instrucciones escritas en un lenguaje de programación que le indica a una computadora qué pasos debe seguir para realizar una tarea. Un programa es la implementación de un algoritmo en un lenguaje de programación.

Qué es un algoritmo: es un conjunto finito de de instrucciones bien definidas que sirven para llevar a cabo una tarea

Qué es el software: El software es el conjunto de programas, aplicaciones y datos que se ejecutan en un sistema informático para realizar tareas específicas.

Que es el hardware: El hardware es la parte física de un sistema informático

Qué es el firmware: es un software de bajo nivel que está directamente integrado en un componente de hardware que sirve para realizar operaciones básicas con el hardware

principales diferencias entre el firmware y el software:

- El firmware se almacena en la ROM y el software se almacena en una unidad de almacenamiento masivo y se ejecuta en la memoria principal.
- La función del firmware es proporcionar las instrucciones básicas para para el funcionamiento y gestión del hardware y el software se emplea para realizar operaciones complejas con el hardware
- A diferencia del software, el firmware no interactúa con el usuario.

Tipos de software

SOFTWARE DE SISTEMAS: es el tipo de software que **permite controlar e interactuar con el hardware**, un ordenador sin software de sistema es inmanejable. Interactúa entre el hardware y software de aplicación. Estos programas realizan diversas tareas, algunas de las funciones típicas son la gestión de los recursos de software o la ejecución de tareas básicas del sistema operativo.

Algunos ejemplos de software de sistemas son:

El sistema operativo, que gestiona recursos del hardware, permite la ejecución de aplicaciones y proporciona interfaces para que los usuarios interactúen con el pc. Entre los más famosos encontramos Windows, Linux, MacOS, Android, iOS.

Controladores, permiten que el sistema operativo se comunique con otros dispositivos como las impresoras.

El Firmware, es un tipo de **software de bajo nivel que se ejecuta directamente en el hardware**. El firmware se utiliza para **controlar y gestionar hardware específico**, como firmware de BIOS en ordenadores, o firmware de un ratón.

SOFTWARE DE APLICACIÓN: Es un tipo de software diseñado para realizar funciones, tareas o actividades de alto nivel por el usuario. Puedes llevar a cabo operaciones de alto nivel que suponen gran cantidad de operaciones de bajo nivel. Un ejemplo sería el navegador, sirve para realizar ciertas tareas complejas en alto nivel. Una pequeña operación que hacemos en alto nivel equivalen a muchísimas en bajo nivel.

SOFTWARE DE DESARROLLO: sirve para desarrollar otro software, podemos incluir entornos de desarrollo, editores, compiladores e intérpretes.

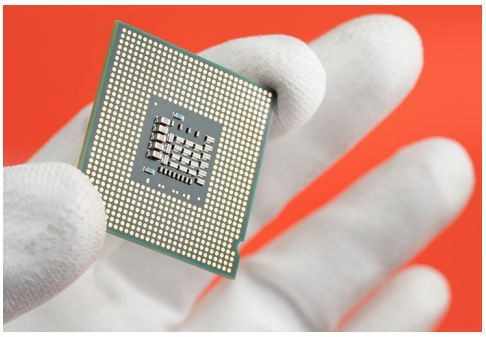
Hardware

El hardware es la parte física de un ordenador o sistema electrónico. En el caso de los ordenadores, se destacan componentes como el Procesador(CPU), Tarjeta gráfica, Unidad de almacenamiento masivo, Memoria RAM, Placa base, Fuente de alimentación(PSU).
A continuación se presentan las principales relaciones entre hardware y software componente por componente además de características técnicas acerca del funcionamiento de los componentes.

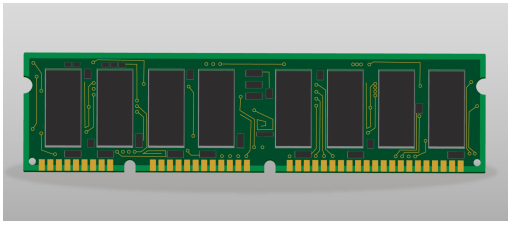
Unidad de almacenamiento: Memoria de almacenamiento masivo, es el almacenamiento secundario. Guarda permanentemente los datos. Algunos ejemplos de almacenamiento secundario son los discos duros (HDD), las unidades de estado sólido SSD y NVMe (memoria no volátil rápida). La velocidad a la que esta unidad lee o escribe datos se mide en mb/s. A día de hoy, los modelos comerciales alcanzan velocidades de aproximadamente 10.000mbps y ya hemos podido ver algunos a velocidades en lectura y escritura de hasta 21.000mbps

CPU: Central Processing Unit, unidad central de proceso, su función es leer y ejecutar instrucciones. Los componentes de un procesador son la **ALU** (Arithmetic Logic Unit, Unidad Aritmética y Lógica), la **CU** (Control Unit, Unidad de Control) y el **Conjunto de Registro.** Los

registros son pequeñas áreas de almacenamiento de datos que trabajan a velocidades muy altas, además desempeñan funciones fundamentales para el funcionamiento de un procesador. Además de los registros los procesadores cuentan con la memoria caché, una memoria de acceso rápido y alta velocidad, almacena datos e instrucciones que se utilizan con mucha frecuencia, está estratificada en 3 niveles: Caché L1, L2 y L3. La memoria caché L1 es la más rápida de las tres y a su vez la de menor capacidad (Cuanto más rápida es la memoria, más alto es su coste, por ello que sea la de menor capacidad) y la memoria caché L3 es la menos rápida y a su vez la de mayor capacidad. Otro factor a tener en cuenta es la ubicación de estas memorias en el procesador, por ejemplo la caché L1 está más cerca del núcleo que el resto.



Memoria RAM: es la **memoria principal**, Random Access memory, todos los programas que se ejecutan en el pc se buscan en la memoria RAM. **Siempre que se ejecuta un programa lo ejecuto en la memoria RAM**. Imagina que voy a ejecutar Photoshop, se buscan los datos en la memoria RAM, si no se encuentra, accede a la memoria secundaria, se transfiere a la memoria RAM y ya sí lo ejecuto. **Es una memoria volátil**, cuando lo desconecto de la corriente se pierde todo lo que contiene.



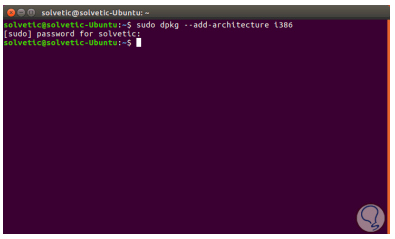
Dispositivos de entrada o salida de datos: **periféricos de entrada** como el ratón o una tableta gráfica **y de salida** los monitores o los cascos. También existen **dispositivos de entrada y salida** de datos como los monitores táctiles.

TIPOS DE CÓDIGO

- **Código fuente**: es el código que ha empleado un programador para desarrollar un programa. Se emplean lenguajes de alto nivel y el resto de programadores pueden leer y entender el código, algunos de los lenguajes que se suelen emplear pueden ser Java, C, C + +, Python, Ruby. Este código se escribe en un software de desarrollo y el código está compuesto por instrucciones, datos, comentarios y sigue una estructura y orden.

- **Código objeto:** Es un paso intermedio entre el código fuente y el código ejecutable, no es ejecutable y procede de una primera traducción del código fuente al lenguaje de máquinas . Posteriormente se incorporarán funciones, métodos o procedimientos alojados en bibliotecas.

- **Código ejecutable:** archivo final ejecutable escrito en lenguaje de máquinas. En función de cada sistema operativo el código ejecutable se almacena con una extensión u otra, por ejemplo en Microsoft Windows el archivo ejecutable se almacena en con la extensión .exe, en linux no se requiere una extensión en específico pero debe indicarse que tiene permisos de ejecución.



En los lenguajes compilados se realiza una traducción completa en que se divide en dos etapas, la primera es la compilación y la segunda es el “linkado”

El ciclo de vida del software

Es un concepto que se refiere a las fases por las que pasa un software desde un concepto inicial hasta su mantenimiento y posterior descontinuación.
Las fases del ciclo de vida del software están diseñadas para servir de guía en el proceso del desarrollo de software y pueden variar dependiendo de la metodología de desarrollo (como el modelo en cascada o el modelo en v)
Las fases principales son:

- Análisis
- Diseño
- Codificación
- Pruebas

- Mantenimiento

Análisis
Se determina y define claramente las necesidades del cliente y se especifica los requisitos que debe cumplir el software a desarrollar

- La especificación de requisitos debe:
    - Ser completa y sin omisiones
    - Ser concisa y sin trivialidades
    - Evitar ambigüedades. Utilizar lenguaje formal.
    - Evitar detalles de diseño o implementación
    - Ser entendible por el cliente
    - Separar requisitos funcionales y no funcionales
    - Dividir y jerarquizar el modelo
    - Fijar criterios de validación

Diseño
Durante esta etapa, se crea un diseño detallado del software, especificando cómo se verá y cómo funcionará. Se descompone y organiza el sistema en elementos componentes que pueden ser desarrollados por separado.
Las actividades habituales son las siguientes:

- Diseño arquitectónico
- Diseño detallado
- Diseño de datos
- Diseño de interfaz de usuario

Codificación
En esta fase, los programadores escriben el código fuente del software de cada componente. Es la etapa en la que se traducen los requisitos y el diseño en código ejecutable.

Pruebas
El principal objetivo de las pruebas debe ser conseguir que el programa funcione incorrectamente y que se descubran los fallos que tiene para poder arreglarlos. Deberemos someter al programa al máximo número de situaciones diferentes

Mantenimiento
Durante la explotación del sistema software es necesario realizar cambios ocasionales. Para ello hay que rehacer parte del trabajo realizado en las fases previas.
los tipos de mantenimiento son:

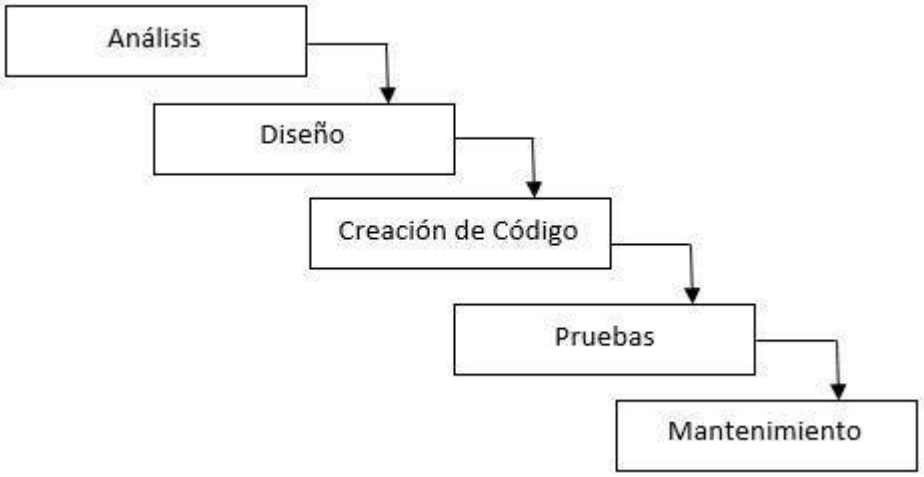
- Correctivo: se corrigen defectos.
- Perfectivo: se mejora la funcionalidad.
- Evolutivo: se añade funcionalidades nuevas.
- Adaptativo: se adapta a nuevos entornos.

MODELO EN CASCADA

El modelo en cascada es un modelo de desarrollo de software con enfoque lineal que identifica las fases principales del desarrollo software y éstas se realizan siguiendo un orden rígido y establecido, de manera que el resultado de una fase es el comienzo de la siguiente
Las fases del modelo en cascada son las siguientes:

1. Análisis

2. Diseño
3. Codificación
4. Pruebas
5. Mantenimiento



Ventajas del modelo en cascada:

- Claridad en su estructura
- Se documentan todas las etapas del desarrollo facilitando la documentación completa del proyecto.
- Fácil de gestionar al ser un modelo lineal en el que el resultado de una fase es el comienzo de la siguiente
- Funciona muy bien en proyectos pequeños cuyos requisitos estén bien definidos

Inconvenientes del modelo en cascada:

- No permite el desarrollo de más de una fase simultáneamente.
- Carece de flexibilidad para adaptarse a cambios en el desarrollo.
- El cliente no ve el producto hasta etapas finales y puede que no cumpla con las expectativas del cliente..
- Si los requisitos no están bien definidos el producto final no será el esperado.

Verificación se refiere al proceso de evaluación de un software para determinar si cumple con los requisitos establecidos. Durante la verificación, se revisan documentos, códigos, planes y especificaciones para garantizar que se sigan los estándares y procesos adecuados. Con ello podemos encontrar posibles errores en el desarrollo del software.

Validación evalúa un software durante o al final del proceso de desarrollo para determinar si satisface las necesidades del usuario y se ajusta a su propósito previsto. La validación implica pruebas dinámicas del sistema en un entorno simulado o real para verificar que el software se comporte según lo esperado y cumpla con los requisitos del cliente.

MODELO DE PROTOTIPOS

Un modelo prototipo o modelo de desarrollo evolutivo es utilizado principalmente en el desarrollo de software para ofrecer al usuario una visión previa de cómo será el programa o

sistema. Se le dice de desarrollo evolutivo al modelo de prototipo porque evoluciona hasta convertirse en el producto final.
En un modelo de prototipos las características fundamentales son:

- Tiempo. El prototipo se desarrolla en menos tiempo para poder ser probado y testeado.
- Coste. La inversión en un modelo de prototipo es ajustada, lo que requiere un uso óptimo de los recursos.
- Conciso. El prototipo debe incluir los requisitos y características básicas de la aplicación para poder evaluar su funcionamiento y utilidad.
- Evolutivo. El prototipo evoluciona gracias a la interacción con los usuarios.
- Funcional. El prototipo es una aplicación que funciona.

El modelo de prototipos se centra en un diseño rápido que representa las características principales del programa que el usuario podrá ver o utilizar. De esta manera pueden
Metodologías ágiles

MÉTODOS INCREMENTALES

Modelo en espiral
El modelo en espiral aplicado al desarrollo orientado a objetos implica establecer objetivos, analizar y diseñar clases y objetos clave, implementarlos en código, realizar pruebas y evaluar resultados. Basándose en esa evaluación, se planifica la siguiente iteración, permitiendo ajustar los objetos según los requisitos cambiantes del proyecto. Este enfoque iterativo y flexible se alinea con los principios del desarrollo orientado a objetos.

Metodologías ágiles

KANBAN

Kanban es una metodología ágil que procede del japonés que significa letrero o señal visual, esta metodología deriva de Lean manufacturing aplicado en la producción de Toyota. La idea general es que todos puedan ver el proceso de trabajo en cada uno de los procesos de producción definidos por el equipo. En Kanban el trabajo se visualiza en un tablero físico o digital. Este tablero está dividido en columnas que representan diferentes etapas del proceso de trabajo, como: por hacer, en proceso y terminado.

Los elementos de una pizarra kanban son:

- Carriles, que representan pasos definidos en el proceso.
- Tarjetas, que representan elementos de trabajo que avanzan a lo largo del proceso.

Kanban es una forma de organizar el trabajo visualmente. Los equipos que usan Kanban limitan cuántas tareas pueden hacer al mismo tiempo y se esfuerzan por trabajar de manera más eficiente. Usan un tablero para ver y organizar el trabajo, y siempre están buscando maneras de hacerlo mejor.

Por otro lado, Scrum es otra forma de gestionar el trabajo. Los equipos de Scrum se comprometen a hacer ciertas tareas en un tiempo específico llamado "sprint". Su objetivo es aprender rápido y mejorar escuchando a los clientes. Tienen roles, documentos y reuniones específicas para ayudarles a mantenerse en el camino correcto.

Algunas de las ventajas de usar una pizarra kanban son:

- Es una herramienta visual que nos permite ver de manera clara y en tiempo real el proceso de trabajo.
- Kanban es altamente adaptable a diferentes tipos de proyectos. Los equipos pueden introducir cambios y ajustes en el proceso sin interrupciones drásticas.
- Los tableros Kanban eliminan en gran medida la necesidad de reuniones de estado al transmitir información de estado a través de la información y la posición de la tarjeta.
- Identificación rápida de problemas, una pizarra kanban permite la identificación de problemas de manera ráìda gracias a que si se invierte más tiempo del estimado en una tarjeta, el equipo de trabajo podrá tomar medidas correctivas para mantener el flujo de trabajo
- Al visualizar el trabajo y alentar la transparencia, Kanban fomenta la colaboración entre los miembros del equipo.

Algunas desventajas de usar kanban son:

- En equipos grandes o con proyectos complejos, el tablero Kanban puede conllevar a una sobrecarga visual
- Kanban no proporciona roles específicos, pudiendo llevar a una falta de estructura, especialmente para equipos nuevos en Kanban, y dificultando la organización y la toma de decisiones.
- Kanban se centra en la gestión del trabajo a corto plazo y puede no ser óptimo para proyectos que requieren una planificación a largo plazo

En conclusión la metodología Kanban es muy útil y ofrece grandes ventajas aunque no es aplicable a todos los proyectos

Kanban es una metodología ágil que procede del japonés que significa letrero o señal visual, esta metodología deriva de Lean manufacturing aplicado en la producción de Toyota. La idea general es que todos puedan ver el proceso de trabajo en cada uno de los procesos de producción definidos por el equipo. En Kanban el trabajo se visualiza en un tablero físico o digital. Este tablero está dividido en columnas que representan diferentes etapas del proceso de trabajo, como: por hacer, en proceso y terminado.

Los elementos de una pizarra kanban son:

- Carriles, que representan pasos definidos en el proceso.
- Tarjetas, que representan elementos de trabajo que avanzan a lo largo del proceso.

Kanban es una forma de organizar el trabajo visualmente. Los equipos que usan Kanban limitan cuántas tareas pueden hacer al mismo tiempo y se esfuerzan por trabajar de manera más eficiente. Usan un tablero para ver y organizar el trabajo, y siempre están buscando maneras de hacerlo mejor.

Por otro lado, Scrum es otra forma de gestionar el trabajo. Los equipos de Scrum se comprometen a hacer ciertas tareas en un tiempo específico llamado "sprint". Su objetivo es aprender rápido y mejorar escuchando a los clientes. Tienen roles, documentos y reuniones específicas para ayudarles a mantenerse en el camino correcto.

QUÉ SON LAS HISTORIAS DE USUARIOS

Una historia de usuario es una breve descripción de una funcionalidad del sistema desde el punto de vista del usuario final. Está escrita en un lenguaje sencillo y comprensible para cualquier persona. Las historias de usuario son utilizadas en las metodologías de desarrollo ágiles para la especificación de requisitos. Deben regirse a una serie de características que son las siguientes:

1. Independientes unas de otras : Cada tarea que necesitamos hacer debe poder hacerse por sí sola.

2. Negociables: Las tareas no son reglas estrictas. Hablar con los usuarios ayuda a entender qué necesitan realmente
3. Valoradas por los clientes o usuarios: Cada tarea debe ser importante para alguien, ya sea para los clientes o los usuarios finales.
4. Estimables: Hablar sobre las tareas nos ayuda a saber cuánto tiempo tomará hacer cada una.de manera que podremos hacer una estimación de cuánto tiempo nos puede tomar un proyecto.
5. Pequeñas: Las tareas no deben ser demasiado largas. Si son muy largas, es difícil saber cuánto tiempo tomarán.
6. Verificables: Las tareas deben ser cosas que se pueden comprobar y probar, especialmente en términos de funciones.

SCRUM

Scrum es una metodología de trabajo ágil en la que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto.

En Scrum un proyecto se ejecuta en ciclos temporales cortos y de duración fija denominados sprints que suelen durar entre una y cuatro semanas. Cada sprint tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite.

En un equipo de trabajo existen los siguientes roles:

- Product Owner: Su principal misión es encargarse de que exista una priorización clara de los objetivos a conseguir, con el propósito de maximizar el valor del trabajo que lleva a cabo el equipo. Las responsabilidades del Product Owner son:
    - Conocer el mercado y los comportamientos de los clientes o usuarios finales, con muy buena visión de Negocio.
    - Ser el representante de todas las personas interesadas para conseguir una buena definición de los objetivos del producto o proyecto y de los resultados esperados.

  El Product Owner es responsable de definir las historias de usuario, priorizar el trabajo de la lista de tareas pendientes del proyecto y tomar decisiones sobre qué funcionalidades se

desarrollan y en qué orden. Además, el Product Owner está disponible para el equipo de desarrollo durante el sprint para proporcionar claridad sobre los requisitos y tomar decisiones rápidas.

- Scrum Master: lidera al equipo y se encarga de conseguir que el equipo conozca los principios y valores ágiles, así como la teoría y prácticas de Scrum, con el objetivo de que los usen en sus procesos de toma de decisiones. El Scrum Master es un facilitador y un líder servicial que trabaja para ayudar al equipo de Scrum a alcanzar su máximo potencial y a seguir las prácticas de Scrum de manera efectiva. De este modo, es el coach y líder al servicio del equipo, llevando a cabo las siguientes responsabilidades:
    - El Scrum Master actúa como entrenador para el equipo de Scrum velando por que todos los participantes del proyecto sigan los valores y principios ágiles, las reglas y proceso de Scrum y guiar la colaboración dentro del equipo y con Product Owner.
    - Proporcionar orientación y apoyo para mejorar el rendimiento del equipo.
    - Quitar impedimentos que el equipo tiene en su camino para conseguir el objetivo de cada iteración y poder finalizar el proyecto con éxito.
    - El Scrum Master facilita las reuniones y eventos de Scrum, como las reuniones de planificación del sprint, las reuniones diarias, la revisión del sprint y la retrospectiva del sprint. Ayuda a organizar estas reuniones y a garantizar que se sigan las reglas y los procesos de Scrum.
    - El Scrum Master lidera la mejora continua del equipo. Facilita la retrospectiva del sprint, una reunión al final de cada sprint donde el equipo reflexiona sobre su desempeño y busca maneras de mejorar en el próximo sprint.
- El equipo de desarrollo es el conjunto de personas que de manera conjunta desarrollan el producto del proyecto. Tienen un objetivo común, comparten la responsabilidad del trabajo que realizan así como de su calidad en cada iteración y en el proyecto. Son multidisciplinarios y autoorganizados, lo que significa que se organizan y asignan tareas entre ellos según las habilidades y las necesidades del proyecto. El tamaño del equipo debe ser de 5 a 9 personas y trabajan conjuntamente

TÉRMINOS

- Product backlog: En la metodología de trabajo scrum, el product owner realiza una lista priorizada de requisitos que recibe el nombre de Product backlog. Los requisitos en el Product Backlog están representados en historias de usuario que describen funcionalidades o características del producto desde la perspectiva del usuario. Estos requisitos adquieren mayor prioridad en función del valor que aporten para el cliente.
- Sprint backlog: es una lista de tareas identificadas por el equipo de scrum que deberá ser completada durante cada sprint. El sprint backlog es representado a través de un tablero de tareas que hace visible todo el trabajo necesario para alcanzar el compromiso que se hizo con el product owner para el sprint.

SINÓNIMOS DE

- Jefe de proyecto: Scrum Master
- Cliente: Product Owner
- Equipo de desarrollo: equipo de desarrollo

RESUMEN

- La cultura de la empresa debe fomentar el trabajo en equipo, la colaboración entre todas las personas implicadas en un proyecto, la creatividad, la transparencia y la mejora continua.
- Requiere una alta implicación y dedicación del cliente en la dirección de los resultados del proyecto, gestión del ROI y disponibilidad para poder colaborar.
- La dirección debe comprometerse a apoyar Scrum, identificar y eliminar obstáculos existentes y futuros, y tomar decisiones para facilitar la transición.
- Compromiso de la Dirección de la organización para resolver problemas endémicos y realizar cambios organizativos, formando equipos autogestionados y multidisciplinares y fomentando una cultura de gestión basada en la colaboración y en la facilitación llevada a cabo por líderes al servicio del equipo.
- Compromiso conjunto y colaboración de los miembros del equipo.
- La relación entre el cliente y el proveedor se basa en un enfoque ganar-ganar, donde se asume que habrá cambios controlados para satisfacer las necesidades reales del cliente

- Scrum permite la incorporación incremental de requisitos y cambios controlados en el producto del proyecto.
- Tamaño de cada equipo entre 5 y 9 personas
- Equipo trabajando en un mismo espacio común para maximizar la comunicación.
- Los miembros del equipo deben dedicarse al proyecto a tiempo completo para evitar la pérdida de productividad debido a cambios constantes de tarea.
- El equipo debe ser estable durante el proyecto para aprovechar las relaciones interpersonales y la organización del trabajo que se han establecido.

EXTREME PROGRAMING

La programación extrema es una metodología ágil de gestión de proyectos que se centra en la velocidad y la simplicidad con ciclos de desarrollo cortos y con menos documentación. La estructura del proceso está determinada por 5 valores fundamentales, 5 reglas y 12 prácticas

Sus cinco valores fundamentales son:

1. Simplicidad: la programación extrema aboga por mantener las cosas lo más simples posible. Los equipos deben buscar soluciones simples y directas para los problemas en lugar de crear soluciones complejas y excesivamente detalladas.
2. Comunicación: la programación extrema se basa en una respuesta rápida y una comunicación efectiva. Para trabajar de manera efectiva, el equipo debe ser abierto y honesto. Ante los problemas todos aportan sus comentarios e ideas, ya que probablemente alguno de ellos ya tenga una solución adecuada. Y si no la tiene, podrán resolver el problema más rápidamente como grupo de lo que podrían hacerlo solos.
3. Comentarios o retroalimentación: El enfoque de XP es producir trabajo de forma rápida y sencilla, para luego compartir los resultados para obtener comentarios de forma casi inmediata. Los equipos recopilan comentarios de los clientes y las pruebas de software para ajustar y mejorar el producto.
4. Valentía: la valentía en XP se refiere a la disposición del equipo para tomar decisiones audaces y arriesgadas en beneficio del producto y del cliente. Los miembros del equipo deben ser valientes para admitir errores, probar nuevas ideas y hacer cambios significativos cuando sea necesario para mejorar el producto
5. Respeto: para que los equipos se comuniquen y colaboren de manera efectiva, deben aprender a estar en desacuerdo. El respeto es una base importante que promueve la bondad y la confianza, incluso cuando se expresan las opiniones con total honestidad.

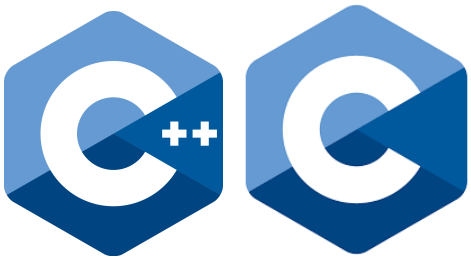
Las diferencias principales entre eXtreme Programming y otras metodologías ágiles son las siguientes:

- En XP no existen rangos no hay rangos jerárquicos formales como metodologías ágiles. XP promueve un enfoque colaborativo, en el que cada miembro del equipo es valorado por sus habilidades.
- En la Programación Extrema, el equipo de trabajo y el cliente colaboran para decidir qué funcionalidades o tareas deben priorizarse. La toma de decisiones sobre la priorización es un proceso colaborativo y participativo que involucra tanto a los miembros del equipo de desarrollo como al cliente. En otras metodologías ágiles se lleva a cabo de otras formas.
- XP es conocida por su flexibilidad extrema para adaptarse a los cambios en los requisitos. La toma de decisiones en XP puede ser altamente dinámica y puede ajustarse rápidamente según las necesidades cambiantes del cliente. XP se destaca por su capacidad para realizar cambios significativos incluso durante una iteración frente al resto de metodologías ágiles
- XP se enfoca en la entrega continua e incremental de software al cliente. Otras metodologías, como Kanban, también pueden permitir un flujo de trabajo continuo pero XP es conocida por su enfoque en la entrega rápida y regular.
- XP enfatiza la retroalimentación inmediata a través de prácticas como las revisiones de código y las pruebas automatizadas. Otras metodologías también valoran la retroalimentación, pero la frecuencia y el enfoque pueden variar.

LENGUAJES IMPERATIVOS Y DECLARATIVOS

Los lenguajes de programación declarativos tienen como principal objetivo lo que debe hacerse, sin tener en cuenta cómo hacerlo. En lenguaje declarativo, se debe definir el resultado deseado o las relaciones entre los datos, y el sistema se encarga de determinar la mejor manera de lograrlo. Algunos ejemplos son SQL y HTML.

Los lenguajes de programación imperativos se centran en cómo se debe hacer una tarea específica. En los lenguajes imperativos, se debe especificar detalladamente los pasos que el sistema debe seguir para alcanzar el resultado deseado. Algunos ejemplos son C y C++.



Diferencias clave entre lenguajes imperativos y declarativos

Legibilidad

- Imperativa: su comprensión tiende a ser más compleja, requieren de una curva de aprendizaje para implementarlos o interpretarlos.
- Declarativa: su comprensión tiende a ser más simple, relega en su nombre la funcionalidad que persigue.

Reusabilidad

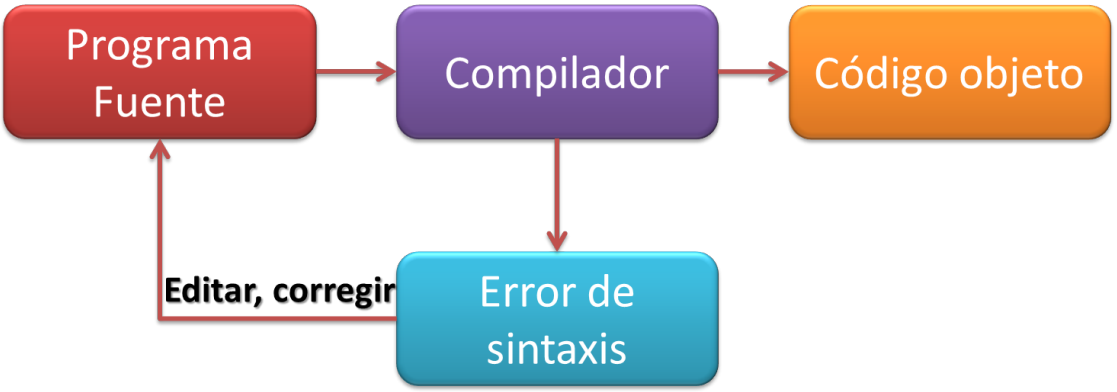
- Imperativo: Para reutilizar una misma función en diferentes partes del programa, la misma secuencia de instrucciones debe ser replicada o copiada.
- Declarativo: Los procesos y las operaciones se encapsulan. De modo que se define una operación compleja una vez y posteriormente se reutilizarla en múltiples partes del programa.

Abstracción

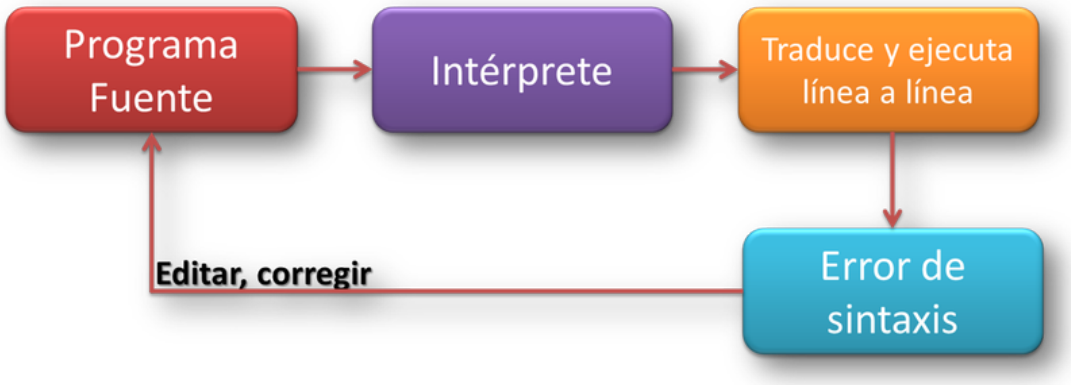
- Imperativo: suelen ser más detallados y cercanos a la máquina, permitiendo un control más preciso
- Declarativo: abstrae el desarrollo lógico a capas inferiores, siendo accedido a través de una función con un nombre descriptivo. No se tiene en cuenta cómo lo hace, lo importante es qué hace.

QUÉ ES COMPILAR E INTERPRETAR

Compilar es el proceso de traducir por completo el código fuente que ha sido escrito por un programador de alto nivel a un lenguaje que la máquina sea capaz de comprender y ejecutar (lenguaje de máquina). Se lleva a cabo mediante la acción de un compilador.



Interpretar es el proceso mediante el cual un programa escrito en un lenguaje de programación es ejecutado línea por línea por un intérprete, a diferencia de la compilación, donde el código se traduce completamente a un código ejecutable antes de ejecutarlo, la interpretación implica analizar y ejecutar el código fuente directamente



,

instrucción por instrucción, durante el tiempo de ejecución.

VENTAJAS DE LOS LENGUAJES COMPILADOS

Los lenguajes compilados son convertidos directamente a código máquina que el procesador puede ejecutar. Como resultado, suelen ser más rápidos y más eficientes al ejecutarse en comparación con los lenguajes interpretados.

Los compiladores pueden aplicar diversas técnicas de optimización al código fuente para mejorar su rendimiento.

Los errores en el código fuente se detectan durante la fase de compilación, lo que permite a los desarrolladores corregir los errores antes de que el programa se ejecute. Los programas compilados suelen utilizar menos recursos del sistema (como memoria y CPU) en comparación con los programas interpretados, ya que no hay un intérprete en ejecución que consuma recursos constantemente.

VENTAJAS DE LOS LENGUAJES INTERPRETADOS

La principal ventaja de los lenguajes interpretados es que son independientes de las máquinas y de los sistemas operativos ya que no contienen instrucciones propias

Los lenguajes interpretados suelen tener una sintaxis más sencilla y permiten el desarrollo más rápido.

Los lenguajes interpretados suelen ser más flexibles y dinámicos en términos de tipos de datos y manipulación de datos. Las variables pueden cambiar de tipo durante la ejecución del programa, lo que proporciona una mayor flexibilidad en el manejo de datos.

CRITERIOS PARA LA ELECCIÓN DE UN LENGUAJE DE PROGRAMACIÓN

- Campo de aplicación: El campo de aplicación se refiere a la naturaleza del problema que estás tratando de resolver con el software.
- Experiencia previa: La experiencia previa de los desarrolladores en un lenguaje particular es un criterio importante. Si un equipo de desarrollo ya tiene experiencia en un lenguaje específico, puede ser más eficiente y productivo utilizar ese lenguaje para nuevos proyectos.

- Herramientas de desarrollo: Las herramientas de desarrollo disponibles para un lenguaje pueden influir significativamente en la productividad y la calidad del código.
- Documentación disponible: La disponibilidad de documentación detallada, tutoriales y recursos educativos es esencial, especialmente para los desarrolladores principiantes o para proyectos complejos.
- Base de usuarios: A mayor cantidad de usuarios que lo vayan a poder usar mejor.
- Reusabilidad: Algunos lenguajes están diseñados con principios que fomentan la reutilización del código, como la modularidad y la orientación a objetos. Un lenguaje que permite la fácil reutilización de componentes y bibliotecas puede acelerar el desarrollo y mejorar la calidad del software.
- Portabilidad: capacidad del software para ejecutarse en diferentes plataformas y sistemas operativos sin modificaciones significativas.
- Imposición del cliente: En algunos casos, el cliente o la organización para la que se desarrolla el software puede imponer restricciones sobre el lenguaje a utilizar.