1. **Haz una clasificación del software.**

   Los tipos de software que hay son:

   - **De sistema:** Sistema operativo, drivers
   - **De aplicación:** Suite ofimática, navegador, edición de imagen
   - **De desarrollo:** Editores, compiladores, interpretes...

2. **Describe la relación que existe entre los componentes hardware principales de un computador y el almacenamiento y ejecución del software.**

   La relación que existe es la siguiente:

   - **Disco duro**: Almacena de forma permanente los archivos ejecutables y los archivos de datos.
   - **Memoria RAM**: Almacena de forma temporal el código binario de los archivos ejecutables y los archivos de datos necesarios.
   - **CPU**: lee y ejecuta instrucciones almacenadas en memoria RAM, así como los datos necesarios.
   - **E/S:** recoge nuevos datos desde la entrada, se muestran los resultados, se leen/guardan a disco…

3. **Define los siguientes conceptos:**

   - **Código fuente.**

Archivo de texto legible escrito en un lenguaje de programación.

   - **Código objeto.**

(intermedio)Archivo binario no ejecutable

   - **Código ejecutable.**

Archivo binario ejecutable

Ciclo de vida del sofware

1. **Define "Ciclo de vida del software".**

   El ciclo de vida del software se refiere a las distintas fases y etapas por las que pasa un programa o sistema de software desde su concepción hasta su retirada o descontinuación. Estas etapas están diseñadas para guiar el desarrollo, mantenimiento y gestión eficientes del software.

2. **Nombra las fases principales del desarrollo de software y explica brevemente que se hace en cada una de ellas.**

   Las fases principales son:
   - Análisis
   - Diseño
   - Codificación
   - Pruebas
   - Mantenimiento

✓ **Análisis**

Se recopilan y analizan los requisitos del software. Esto implica comprender a fondo las necesidades del usuario, las funciones que debe realizar el software y las restricciones que debe cumplir.

✓ **Diseño**

En esta fase se crea una arquitectura y diseño detallado del software. Se definen las estructuras de datos, algoritmos y componentes del sistema

✓ **Codificación**

Se escribe el código fuente de cada componente. Pueden utilizarse distintos lenguajes informáticos:

Lenguaje de programación, C++, Java

Lenguaje de otro tipo:HTML,XML,JSON,…

✓ **Pruebas**
El principal objetico de las pruebas debe ser conseguir que el programa funcione incorrectamente y que se descubran defectos.
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.

3. Explica brevemente en qué consiste el modelo en cascada cuando hablamos de desarrollo de software.

El método de cascada se basa en que los equipos sigan una secuencia de pasos y nunca avancen hasta que se haya completado la fase anterior. Esta metodología, en su forma tradicional, no deja prácticamente ningún lugar para cambios o revisiones imprevistos.

4. Ventajas e inconvenientes del modelo en cascada.

Ventajas

- **Estructura Clara y Documentación Exhaustiva:** Cada fase está bien definida, lo que facilita la planificación y el seguimiento del progreso del proyecto. Además, se requiere una documentación detallada, lo que puede ser útil para futuras referencias y mantenimiento.
- **Fácil de Gestionar**: Debido a su naturaleza secuencial, es relativamente sencillo de gestionar y supervisar. Cada fase tiene hitos claros y se puede medir el progreso de manera concreta.
- **Adecuado para Proyectos Bien Definidos**: Funciona bien cuando los requisitos están bien entendidos y no se esperan cambios significativos durante el desarrollo.
- **Control de Costos y Tiempos:** Debido a su enfoque estructurado y predecible, puede ser más fácil de estimar y controlar los costos y los tiempos de entrega.

Desventajas

- **Poco Tolerante a Cambios:** El modelo en cascada es inflexible ante cambios en los requisitos después de que se haya iniciado la etapa de implementación. Introducir modificaciones puede ser costoso y complicado.
- **Validación Tardía del Producto Final:** Los usuarios no ven el producto final hasta que la fase de implementación está casi completa, lo que significa que cualquier error o malentendido en los requisitos puede no ser descubierto hasta tarde en el proceso.
- **Riesgo de No Cumplir con las Expectativas del Cliente**: Si los requisitos iniciales están mal entendidos o mal especificados, existe un riesgo significativo de que el producto final no cumpla con las expectativas del cliente.

- **No Adecuado para Proyectos Innovadores o Complejos:** No es el modelo más adecuado para proyectos donde los requisitos son inciertos o pueden cambiar rápidamente, o para proyectos de investigación y desarrollo donde la innovación es esencial.

5. ¿Qué se entiende por verificación? ¿Y por validación?

La verificación se refiere al proceso de evaluar si el software cumple con las especificaciones y requisitos establecidos durante la fase de desarrollo. Es un proceso orientado a asegurarse de que el software se está construyendo correctamente. Esto implica revisar y analizar el código fuente, así como realizar pruebas para garantizar que el software se adhiere a las normas y estándares predefinidos.

La validación, por otro lado, se refiere al proceso de evaluar si el software cumple con las necesidades y expectativas del cliente. Es un proceso orientado a asegurarse de que el software cumple con los requerimientos y proporciona el valor esperado al usuario final. Esto implica realizar pruebas funcionales y de aceptación del usuario para verificar si el software satisface los objetivos de negocio.

6. **Explica como funciona el modelo de desarrollo mediante creación de prototipos**.

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 probarlo y dar su opinión sobre distintos aspectos como la usabilidad, la utilidad o el rendimiento, entre otras.

7.Explica como funciona el modelo espiral cuando se aplica al desarrollo orientado a objetos.

El modelo de desarrollo en Espiral es una combinación entre el modelo waterfall y un modelo por iteraciones. El proceso pasa por distintas etapas, desde la de conceptualización, siguiendo el desarrollo, luego una fase de mejoras, para finalizar con el mantenimiento.

7. **¿Qué cuatro principios rigen el desarrollo ágil expresados en el Manifiesto Ágil?**

- **Individuos e interacciones:**

Este principio destaca la importancia de las personas y la comunicación efectiva en el proceso de desarrollo. Se enfoca en la colaboración activa entre los miembros del equipo y la comprensión mutua de los objetivos y desafíos.

- **Software funcionando:**

Significa que el énfasis debe estar en la entrega de software funcional y utilizable, en lugar de enfocarse excesivamente en la documentación detallada. Aunque la documentación es importante, no debe convertirse en un obstáculo para el progreso.

- **Colaboración con el cliente:**

Se destaca la importancia de la comunicación cercana y la colaboración continua con el cliente para comprender y adaptarse a sus necesidades en constante cambio. Es esencial mantener una relación cercana y abierta con el cliente en lugar de depender únicamente de acuerdos contractuales formales.

- **Respuesta ante el cambio:**

Reconoce que los requisitos y las circunstancias pueden cambiar a lo largo del proyecto, y que el equipo debe estar preparado para adaptarse y ajustar su enfoque en consecuencia. La capacidad de responder de manera ágil a los cambios es un aspecto clave del desarrollo ágil.

8. **¿Qué es una historia de usuario? Consulta el siguiente enlace:**
    - https://es.wikipedia.org/wiki/Historias_de_usuario

Una historia de usuario es una representación de un requisito escrito en una o dos frases utilizando el lenguaje común del usuario. Las historias de usuario son utilizadas en las metodologías de desarrollo ágiles para la especificación de requisitos (acompañadas de las discusiones con los usuarios y las pruebas de validación).

Las historias de usuario son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Las historias de usuario permiten responder rápidamente a los requisitos cambiantes.

9. **Haz un resumen sobre que se entiende por Lean software y qué principios lo rigen. Consulta el siguiente enlace:**
    - https://es.wikipedia.org/wiki/Lean_software_development

Las metodologías de desarrollo de software lean (traducción aproximada en este contexto de lean: «austero», «firme», «seguro» o «eficiente») es una traducción de los principios y las prácticas de la forma de producir lean, hacia el área del desarrollo de software. Inicialmente, originado en el Sistema de Producción de Toyota y ahora, apoyado por una corriente que está surgiendo desde la comunidad Ágil. Este método ofrece todo un marco teórico sólido y basado en la experiencia, para las prácticas ágiles de gestión.

Principios clave del Lean Software Development:

- **Eliminación del Desperdicio (Waste Elimination):** Se busca identificar y eliminar actividades que no añaden valor al cliente, como la duplicación de esfuerzos, la sobreproducción de características innecesarias o la espera por la aprobación de tareas.
- **Amplificación del Aprendizaje (Amplify Learning):** Se promueve un ciclo continuo de experimentación, retroalimentación y ajuste, permitiendo a los equipos aprender y mejorar de manera constante a lo largo del desarrollo.
- **Toma de Decisiones Basada en el Último Momento Posible (Decide as Late as Possible):** Se busca retrasar las decisiones hasta que se disponga de la información más actualizada y relevante, lo que reduce el riesgo de tomar decisiones prematuras o basadas en suposiciones.
- **Entregas Rápidas (Deliver as Fast as Possible):** Se enfoca en minimizar los tiempos de ciclo y llevar funcionalidades al cliente tan pronto como estén listas, lo que permite obtener retroalimentación temprana y adaptarse a las necesidades cambiantes.
- **Empoderamiento de los Equipos (Empower the Team):** Se promueve la autonomía y la toma de decisiones por parte de los equipos de desarrollo, lo que fomenta la responsabilidad y la creatividad en la resolución de problemas.
- **Construcción de la Integridad (Build Integrity In):** Se enfoca en la calidad desde el principio, asegurándose de que cada componente o módulo sea de alta calidad antes de pasar al siguiente.
- **Optimización del Todo (Optimize the Whole):** Se busca maximizar el rendimiento del sistema en su conjunto, en lugar de optimizar partes individuales. Esto implica coordinar y alinear esfuerzos en toda la organización.

- **Añadir Valor (Add Value):** Se prioriza la entrega de funcionalidades y características que aporten un valor real y tangible al cliente, evitando la sobreproducción de características innecesarias
- **Respeto por las Personas (Respect People):** Se reconoce la importancia de crear un ambiente de trabajo donde las personas se sientan valoradas y motivadas, lo que contribuye a un entorno más productivo y colaborativo.

10. KANBAN. Estudia las ventajas e inconvenientes de tener una pizarra web digital para la metodología Kanban. Puedes consultar los siguientes enlaces:

- https://leankit.com/learn/kanban/kanban-board/
- https://trello.com/es
- https://taiga.io/
- https://kanbantool.com/es/

Ventajas de tener una pizarra web digital para la metodología Kanban:

- **Acceso Remoto:** Permite a los miembros del equipo acceder a la pizarra Kanban desde cualquier ubicación con una conexión a Internet, lo que facilita el trabajo remoto y la colaboración entre equipos distribuidos.

- **Actualizaciones en Tiempo Real:** Los cambios realizados en la pizarra son visibles para todos los miembros del equipo al instante, lo que facilita la comunicación y la toma de decisiones basadas en la información más reciente.

- **Facilita la Gestión de Tareas**: Permite mover, reasignar y actualizar tareas de forma rápida y sencilla, lo que agiliza la gestión del flujo de trabajo y la asignación de responsabilidades.

- **Integración con Herramientas y Aplicaciones:** Muchas pizarras web para Kanban ofrecen integraciones con otras herramientas de gestión de proyectos, herramientas de comunicación y aplicaciones de colaboración, lo que facilita la sincronización de información y la automatización de procesos.

- **Seguimiento de Métricas y Estadísticas:** Algunas pizarras web ofrecen capacidades de análisis y generación de informes, lo que permite a los equipos realizar un seguimiento del rendimiento y la eficiencia del proceso Kanban.

Inconvenientes de tener una pizarra web digital para la metodología Kanban:

- **Dependencia de la Tecnología:** Si hay problemas con la plataforma o la conexión a Internet, los miembros del equipo pueden enfrentar dificultades para acceder y actualizar la pizarra.

- **Posible Curva de Aprendizaje:** Para algunos miembros del equipo, especialmente aquellos menos familiarizados con la tecnología, puede llevar tiempo acostumbrarse a la plataforma y aprender a utilizarla de manera efectiva.

- **Posible Falta de Flexibilidad:** Algunas pizarras web pueden estar diseñadas con características y funciones específicas que pueden no adaptarse completamente a las necesidades o preferencias del equipo.

- **Costo Asociado:** Algunas pizarras web para Kanban pueden tener un costo, especialmente si ofrecen características avanzadas o integraciones con otras herramientas.

1. **KANBAN. Haz un resumen de la metodología Kanban e indica sus diferencias frente a SCRUM. Puedes consultar el siguiente enlace:**
   - https://es.atlassian.com/agile/kanban

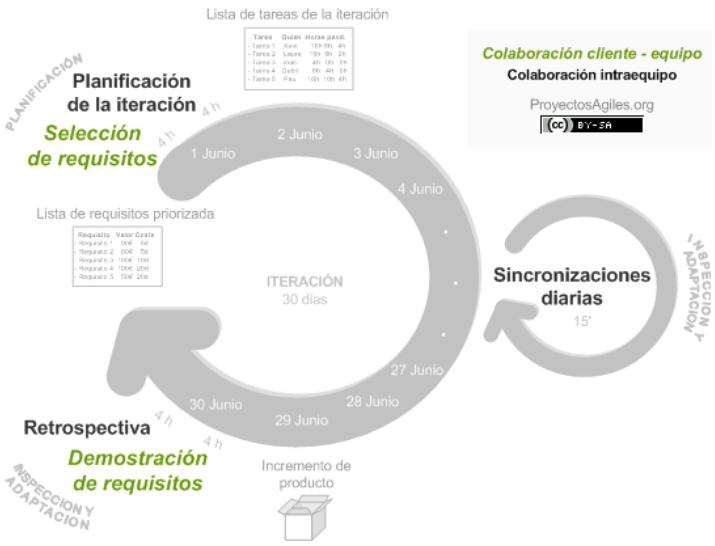
Un tablero de kanban es una herramienta ágil de gestión de proyectos diseñada para ayudar a visualizar el trabajo, limitar el trabajo en curso y maximizar la eficiencia (o el flujo). Puede ayudar tanto a los equipos ágiles como a los de DevOps a definir el orden de su trabajo diario. Los tableros de kanban utilizan tarjetas, columnas y la mejora continua para ayudar a los equipos tecnológicos y de servicios a comprometerse con la cantidad de trabajo adecuada y, por supuesto, a llevarla a cabo.

Elementos los tableros de kanban

Los tableros de kanban se pueden dividir en cinco componentes: señales visuales, columnas, límites del trabajo en curso, un punto de compromiso y un punto de entrega**.**

- **Señales visuales**: una de las primeras cosas que observarás en un tablero de kanban son las tarjetas visuales (adhesivos, tickets, etc.). Los equipos de kanban escriben todos sus proyectos y elementos de trabajo en tarjetas, generalmente uno por tarjeta
- **Columnas**: otra de las señas de identidad de los tableros de kanban son las columnas. Cada columna representa una actividad específica que, en conjunto, conforman un "flujo de trabajo".
- **Límites del trabajo en curso:** los límites del trabajo en curso son el número máximo de tarjetas que puede haber en una columna en un momento dado. Una columna con un límite de trabajo en curso de tres no puede contener más de tres tarjetas
- **Punto de compromiso:** los equipos de kanban suelen tener un backlog para su tablero. Es aquí donde los clientes y los compañeros de equipo plantean ideas para proyectos que el equipo puede adoptar cuando esté listo para ello.
- **Punto de entrega:** el punto de entrega es el final del flujo de trabajo de un equipo de kanban. Para la mayoría de los equipos, el punto de entrega es el momento en el que el producto o servicio está en manos del cliente

2. **SCRUM. Explica como funciona Scrum. Consulta los siguientes enlaces:**
    - https://proyectosagiles.org/que-es-scrum/
    - https://proyectosagiles.org/como-funciona-scrum/



En Scrum un proyecto se ejecuta en ciclos temporales cortos y de duración fija (iteraciones que normalmente son de 2 semanas, aunque en algunos equipos son de 3 y hasta 4 semanas, límite máximo de feedback de producto real y reflexión). Cada iteración 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.

El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente (Product Owner) prioriza los objetivos balanceando el valor que le aportan respecto a su coste (que el equipo estima considerando la Definición de Hecho) y quedan repartidos en iteraciones y entregas.

Las actividades que se llevan a cabo en Scrum son las siguientes (los tiempos indicados son para iteraciones de 2 semanas):

Planificación de la iteración

El primer día de la iteración se realiza la reunión de planificación de la iteración. Tiene dos partes:

- Selección de requisitos (2 horas)
- Planificación de la iteración (2 horas).

Ejecución de la iteración

Cada día el equipo realiza una reunión de sincronización (15 minutos), normalmente delante de un tablero físico o pizarra (Scrum Taskboard). En la reunión cada miembro del equipo responde a tres preguntas:

- ¿Qué he hecho desde la última reunión de sincronización para ayudar al equipo a cumplir su objetivo?
- ¿Qué voy a hacer a partir de este momento para ayudar al equipo a cumplir su objetivo?
- ¿Qué impedimentos tengo o voy a tener que nos impidan conseguir nuestro objetivo?

Durante la iteración el Facilitador (Scrum Master) se encarga de que el equipo pueda mantener el foco para cumplir con sus objetivos.

- Elimina los obstáculos que el equipo no puede resolver por sí mismo.
- Protege al equipo de interrupciones externas que puedan afectar el objetivo de la iteración o su productividad.

Inspección y adaptación

El último día de la iteración se realiza la reunión de revisión de la iteración. Tiene dos partes:

- Revisión (demostración) (1,5 horas). El equipo presenta al cliente los requisitos completados en la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo.
- Retrospectiva (1,5 horas). El equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad. El Facilitador se encargará de eliminar o escalar los obstáculos identificados que estén más allá del ámbito de acción del equipo.

3. SCRUM. Define los siguientes términos:

- Product backlog.: El backlog de un producto es una lista de trabajo ordenado por prioridades para el equipo de desarrollo que se obtiene de la hoja de ruta y sus requisitos. Los elementos más importantes se muestran al principio del backlog del producto para que el equipo sepa qué hay que entregar primero.

Sprint backlog: El sprint backlog es representado a través de un tablero de tareas; hace visible todo el trabajo necesario para alcanzar el compromiso que se hizo con el product owner para el sprint. Permite ver las tareas donde el equipo está teniendo problemas y no avanza, para tomar decisiones al respecto.

---

4. SCRUM. En la terminología Scrum qué terminos se utilizan como sinónimo de:

- Jefe de proyecto.
- Cliente.
- Equipo de desarrollo.

- Jefe de proyecto:En Scrum, no hay un rol específico llamado "Jefe de Proyecto". En cambio, hay un rol llamado Scrum Master, que es responsable de facilitar el proceso Scrum y asegurarse de que el equipo siga las prácticas y reglas de Scrum. El Scrum Master no es un jefe en el sentido tradicional, sino más bien un facilitador y líder de proceso.
- Cliente :En Scrum, el término que se utiliza para referirse a la entidad o persona que representa los intereses del negocio y que tiene la autoridad para tomar decisiones es el Product Owner. El Product Owner es

responsable de definir y priorizar los elementos del Product Backlog, así como de tomar decisiones sobre qué funcionalidades deben ser implementadas.

- Equipo de desarrollo: En Scrum, el término que se utiliza para referirse al grupo de personas responsables de llevar a cabo el trabajo de desarrollo es simplemente el Equipo de Desarrollo. Este equipo es autónomo y multidisciplinario, y se organiza de manera autónoma para completar el trabajo durante el Sprint. No hay un líder designado dentro del equipo de desarrollo en Scrum.

5. **SCRUM. Haz un resumen de los requisitos para poder utilizar Scrum. Consulta el siguiente enlace:**
   - https://proyectosagiles.org/requisitos-de-scrum/

   Los siguientes puntos son de especial importancia para la implantación de una gestión ágil de proyectos como Scrum:

1. Cultura de empresa basada en trabajo en equipo, delegación, creatividad y mejora continua.
2. Compromiso del cliente en la dirección de los resultados del proyecto, gestión del ROI y disponibilidad para poder colaborar.
3. 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.
4. Compromiso conjunto y colaboración de los miembros del equipo.
5. Relación entre proveedor y cliente basada en ganar-ganar, colaboración y transparencia.
6. Facilidad para realizar cambios en el proyecto.
7. Tamaño de cada equipo entre 5 y 9 personas (con técnicas específicas de planificación y coordinación cuando varios equipos trabajan en el mismo proyecto).
8. Equipo trabajando en un mismo espacio común para maximizar la comunicación.
9. Dedicación del equipo a tiempo completo.
10. Estabilidad de los miembros del equipo

6. **XP. Explica los 5 valores de la Programación Extrema.**

La Programación Extrema (XP, por sus siglas en inglés, eXtreme Programming) es una metodología ágil de desarrollo de software que se centra en la mejora de la calidad y la eficiencia en equipos pequeños. Está basada en cinco valores fundamentales que guían la forma en que se lleva a cabo el proceso de desarrollo. Estos valores son:

- Comunicación: La comunicación efectiva es esencial en XP. Los miembros del equipo deben estar en constante interacción, compartiendo información sobre el progreso, los desafíos y las soluciones. Esto incluye comunicarse con el cliente para entender y validar sus necesidades.

- Simplicidad: XP aboga por mantener las cosas simples y evitar la complejidad innecesaria. Esto se refleja en la forma en que se diseñan y desarrollan las soluciones. Se busca implementar la funcionalidad justa y necesaria para satisfacer los requisitos, evitando el exceso de complejidad que puede dificultar la mantenibilidad y comprensión del código.

- Retroalimentación (Feedback): La retroalimentación constante es un pilar de XP. Esto implica recibir y proporcionar comentarios sobre el trabajo realizado. Por ejemplo, las pruebas continuas y la integración continua proporcionan feedback inmediato sobre la calidad del código.

- Valentía (Courage): En XP, la valentía se refiere a la voluntad y la determinación para enfrentar los desafíos y tomar decisiones difíciles. Esto puede implicar, por ejemplo, admitir errores, cambiar de dirección en el desarrollo o sugerir mejoras en el proceso.

- Respeto (Respect): El respeto es fundamental en XP. Se trata de valorar y confiar en los miembros del equipo, así como en el cliente y otros stakeholders. También implica respetar el trabajo y las opiniones de los demás.

7. XP. ¿Cuáles son las características distintivas de XP frente a otras metodologías ágiles? Explícalas. Puedes consultar el siguiente enlace:

- http://www.davidvalverde.com/blog/introduccion-a-la-programacion-extrema-xp/

La programación extrema o XP es una metodología de desarrollo que se englobaría dentro de las denominadas metodologías Ágiles en la que se da máxima prioridad a la obtención de resultados y reduce la burocracia que se produce al utilizar otras 'metodologías pesadas'.

Objetivos de la programación extrema

El objetivo principal de la XP es la satisfacción del cliente. Se le trata de dar al cliente lo que quiere y cuando quiere. Por tanto, se debe responder rápidamente a las necesidades del cliente, aunque realice cambios en fases avanzadas del proyecto. Como metodología Ágil que es, se pueden producir modificaciones de los requisitos del proyecto a lo largo de su desarrollo, sin que esto produzca un buen dolor de cabeza.

Valores de la programación extrema

Para garantizar el éxito de un proyecto, los autores de XP han considerado como fundamentales cuatro valores:

- Comunicación. Muy importante. La XP ayuda mediante sus prácticas a la comunicación entre los integrantes del grupo de trabajo: jefes de proyecto, clientes y desarrolladores.
- Sencillez. Los programas deben ser los más sencillos posibles y tener la funcionalidad necesaria que se indican en los requisitos. No hay que añadir algo que no se necesite hoy. Si se necesita añadir más funcionalidad mañana pues ya se hará entonces.
- Retroalimentación. Las pruebas que se le realizan al software nos mantiene informados del grado de fiabilidad del sistema.
- Valentía. Asumir retos, ser valientes ante los problemas y afrontarlos. El intentar mejorar algo que ya funciona. Aunque gracias a las pruebas unitarias no existe el riesgo de 'meter la pata'.

Lenguajes de programación

1. **¿Qué diferencia existe entre los lenguajes declarativos y los imperativos?. Nombra al menos 2 de cada tipo.**

Los lenguajes declarativos y los lenguajes imperativos son dos paradigmas diferentes en programación que se utilizan para expresar la lógica de un programa de manera distinta.

Lenguajes Declarativos:

En un lenguaje declarativo, se describe lo que se quiere lograr sin especificar cómo hacerlo. El programador se enfoca en definir el problema y el sistema determina la mejor manera de resolverlo.

1. SQL (Structured Query Language): **SQL** se utiliza para realizar consultas y manipulación de bases de datos relacionales. Los comandos SQL describen qué datos se deben recuperar o modificar, pero no especifican cómo se deben llevar a cabo estas operaciones.
2. Prolog: **Prolog** es un lenguaje de programación lógica en el que se definen relaciones y se hacen consultas sobre ellas. Se basa en la lógica de primer orden y se centra en inferencias.

Lenguajes Imperativos:

En un lenguaje imperativo, se detalla paso a paso cómo se debe realizar una tarea. El programador instruye explícitamente al sistema sobre qué pasos debe seguir para llegar al resultado deseado.

1. **C**: C es un lenguaje de programación de propósito general que sigue un enfoque imperativo. En C, se especifican claramente los pasos que el programa debe seguir para llevar a cabo una operación.
2. **Python**: Aunque Python es un lenguaje multiparadigma, a menudo se utiliza de manera imperativa. Los programas en Python a menudo siguen un flujo de control explícito donde se especifican las operaciones en un orden secuencial.

Diferencias clave:

- Enfoque: Los lenguajes declarativos se centran en "qué" se quiere lograr, mientras que los lenguajes imperativos se centran en "cómo" lograrlo.
- Nivel de abstracción: Los lenguajes declarativos tienden a ser más abstractos, dejando los detalles de implementación al sistema subyacente. Los lenguajes imperativos son más detallados y específicos.
- Optimización: Los lenguajes declarativos confían en el sistema para optimizar la ejecución. En los lenguajes imperativos, el programador tiene más control sobre la optimización.

Ambos paradigmas tienen sus propias ventajas y se eligen según el tipo de problema que se está abordando y las preferencias del programador.

2. ¿Explica qué es compilar? ¿Explica qué es interpretar?

Compilar: se denomina la fase de codificación en que un programa es traducido del código fuente al código máquina para que pueda ejecutarse**.** Como tal, la realiza un compilador virtual, cuya tarea consiste en llevar un programa fuente a programa objeto

Interpretar: Un lenguaje de programación interpretado es aquel cuyo código fuente es ejecutado directamente por un intérprete en lugar de ser compilado a un lenguaje de bajo nivel antes de ser ejecutado

3. Ventajas de los lenguajes compilados.

Los programas que son compilados a un código máquina nativo suelen ser más rápidos que los lenguajes interpretados. Esto es debido a que el proceso de traducción del código en tiempo de ejecución aumenta la sobrecarga y puede ocasionar que el programa sea más lento en general.

4. Ventajas de los lenguajes interpretados.

Los lenguajes interpretados suelen ser más flexibles, y a menudo ofrecen características como escritura dinámica y tamaño de programa más pequeño. Además, ya que los intérpretes ejecutan el código fuente del programa ellos mismos, el código en sí es independiente de la plataforma.

5. Nombra 2 lenguajes compilados y otros 2 interpretados.

Ejemplos de lenguajes compilados incluyen C, C++, Java, Go y Rust, entre muchos otros.

Ejemplos de lenguajes interpretados incluyen Ruby, Python y JavaScript, entre muchos otros. A todos estos lenguajes se les conoce como lenguajes de alto nivel.

6. **¿Puede considerarse código objeto el bytecode generado en Java tras la compilación? Explica la respuesta.**

Sí, el bytecode generado en Java tras la compilación puede considerarse como código objeto.

Cuando se compila un programa Java, el compilador traduce el código fuente Java a un lenguaje intermedio llamado "bytecode". Este bytecode no es código de máquina directamente ejecutable por la CPU, sino que está diseñado para ser interpretado por la Máquina Virtual de Java

7. **Pon un ejemplo de lenguaje de los siguientes tipos:**

- Bajo nivel.

  Ejemplo: Lenguaje ensamblador (Assembly language).

  Descripción: El lenguaje ensamblador es un lenguaje de programación de bajo nivel que utiliza mnemotécnicos y códigos de operación para representar las instrucciones de la CPU. Cada instrucción del lenguaje ensamblador se corresponde con una instrucción en lenguaje máquina específica de la arquitectura de la CPU.

- Nivel medio.

Ejemplo: C++

Descripción: C++ es una extensión del lenguaje C que incluye características de programación orientada a objetos. Combina las características de bajo nivel de C con abstracciones de alto nivel, permitiendo tanto programación estructurada como orientada a objetos.

- Alto nivel.

Ejemplo: Python.

Descripción: Python es un lenguaje de programación de alto nivel que se centra en la legibilidad y la simplicidad. Ofrece abstracciones de alto nivel que permiten a los programadores expresar ideas de manera más concisa y clara. Python es conocido por su facilidad de aprendizaje y amplia biblioteca estándar.

8. ¿Qué paradigma de programación siguen los siguientes lenguajes?

C:

- Paradigma: Procedural.

Descripción: C es un lenguaje de programación que sigue principalmente el paradigma procedural. Se centra en la estructuración del programa en procedimientos o funciones que manipulan datos. Sin embargo, C también puede admitir programación imperativa y, en menor medida, programación orientada a objetos a través de extensiones y prácticas de programación específicas.

C++:

- Paradigma: Multi-paradigma (Principalmente orientado a objetos, pero también soporta programación procedural y genérica).

Descripción: C++ es un lenguaje que extiende a C y añade características de la programación orientada a objetos (como clases y herencia), así como características de programación genérica (a través de plantillas). También sigue el paradigma procedural.

SQL:

- Paradigma: Declarativo (Específicamente, lenguaje de manipulación de datos).

Descripción: SQL (Structured Query Language) es un lenguaje diseñado para manejar bases de datos relacionales. En lugar de instruir explícitamente cómo realizar una operación, los usuarios de SQL describen el resultado deseado y el sistema de gestión de bases de datos (DBMS) se encarga de determinar la mejor forma de obtenerlo.

Java:

- Paradigma: Multi-paradigma

(Principalmente orientado a objetos, pero también soporta programación imperativa y funcional).

Descripción: Java es un lenguaje de programación que sigue principalmente el paradigma de programación orientada a objetos. Sin embargo, a partir de la versión 8, Java introdujo características de programación funcional con la inclusión de expresiones lambda y Streams API.

JavaScript:

- Paradigma: Multi-paradigma (Principalmente orientado a objetos y funcional, pero también soporta programación imperativa).

Descripción: JavaScript es conocido por ser un lenguaje de programación versátil que admite múltiples paradigmas. Es principalmente un lenguaje orientado a objetos y funcional, pero también permite la programación imperativa. Desde ECMAScript 6, JavaScript ha incorporado muchas características de programación funcional, como las funciones de flecha y los métodos de array de alto orden.

Lisp:

- Paradigma: Funcional.

Descripción: Lisp es uno de los lenguajes de programación más antiguos que se utilizan comúnmente. Es conocido por su fuerte adhesión al paradigma de programación funcional y por su capacidad para manipular código como datos.

Prolog:

- Paradigma: Lógico.

Descripción: Prolog es un lenguaje de programación lógica que se basa en la lógica de predicados. Los programas Prolog están compuestos por una base de hechos y reglas de inferencia, y la ejecución se basa en un motor de inferencia que resuelve consultas lógicas.

9. Explica qué criterios pueden seguirse a la hora de elegir un lenguaje de programación para el desarrollo software.

La elección del lenguaje de programación es una decisión importante en el desarrollo de software y puede tener un impacto significativo en la eficiencia, mantenibilidad y rendimiento del proyecto. A continuación, se presentan

algunos criterios que se pueden considerar al elegir un lenguaje de programación:

1. Objetivos del Proyecto:

Tipo de Aplicación: ¿Es una aplicación web, de escritorio, móvil o un sistema embebido? Cada tipo de aplicación puede requerir un conjunto diferente de herramientas y lenguajes.

2. Experiencia del Equipo:

Conocimientos Previos: ¿El equipo de desarrollo tiene experiencia previa en un lenguaje específico? Utilizar un lenguaje familiar puede acelerar el proceso de desarrollo.

3. Capacidad de Aprendizaje:

¿El equipo está dispuesto y tiene el tiempo para aprender un nuevo lenguaje?

4. Eficiencia y Rendimiento:

Requerimientos de Rendimiento: ¿La aplicación requiere un alto rendimiento y eficiencia? Algunos lenguajes pueden ser más eficientes en determinadas tareas o en ciertos contextos.

5. Bibliotecas y Frameworks:

Ecosistema de Desarrollo: ¿El lenguaje tiene una amplia gama de bibliotecas y frameworks que faciliten el desarrollo de la aplicación?

6. Portabilidad y Plataformas Soportadas:

Requisitos de Plataforma: ¿La aplicación debe ejecutarse en múltiples plataformas (Windows, macOS, Linux, etc.)? Algunos lenguajes son más portables que otros.

7. Seguridad:

Vulnerabilidades y Prácticas de Seguridad: ¿El lenguaje tiene características que faciliten la escritura de código seguro? ¿Es propenso a ciertos tipos de vulnerabilidades?.

8. Comunidad y Soporte:

Comunidad de Desarrolladores: ¿El lenguaje tiene una comunidad activa que proporciona soporte, tutoriales y soluciones a problemas comunes?

9. Escalabilidad:

Posibilidad de Escalabilidad: ¿El lenguaje y sus herramientas permiten escalar la aplicación a medida que crece la demanda?

10. Tiempo de Desarrollo:

Productividad del Desarrollador: ¿El lenguaje y sus herramientas permiten un desarrollo rápido y eficiente?.

11. Costo:

Licencias y Herramientas: ¿El lenguaje y las herramientas asociadas son de código abierto o requieren licencias costosas?.

12. Compatibilidad con Sistemas Existentes:

Integración con Sistemas Heredados: ¿Es necesario que el nuevo software se integre con sistemas o componentes existentes que están escritos en un lenguaje específico?

13. Tendencias del Mercado

14. Popularidad y Demanda Laboral: ¿El lenguaje está en alta demanda en el mercado laboral?