


Informe de Lenguajes, Paradigmas y Estándares de Programación



Javier Carrillo

28/11/2023

PORTADA

Nombre Alumno / DNI	Javier Carrillo Martin/ 06325710C
Título del Programa	1ºPHE Applied Computing & Artificial Intelligence
Nº Unidad y Título	Entrega parcial parte 4
Año académico	2032-2024
Profesor de la unidad	Gabriela García
Título del Assignment	UNIT 1 Programing and coding
Día de emisión	18/09/2023
Día de entrega	31/01/2024
Nombre IV y fecha	29/01/2024
Declaración del estudiante	<p>Certifico que la presentación del assignment es completamente mi propio trabajo y entiendo completamente las consecuencias del plagio. Entiendo que hacer una declaración falsa es una forma de mala práctica.</p> <p>Fecha: 31/01/2024</p> <p>Firma del alumno:</p> 

INTRODUCCIÓN

Los lenguajes de programación refieren a distintos tipos de expresiones y reglas de estructuración lógica que sirven para generar tareas recurrentes y sistemáticas. Los mismos son de gran importancia porque permiten generar distintos sistemas que sirven para tareas que satisfacen las necesidades de los usuarios.

Un paradigma de programación, por tanto, es un método para resolver un problema o realizar una tarea. Si bien existen diferentes lenguajes de programación se necesita de una estrategia cuando se implementa, es decir, un camino, y ese puede ser los paradigmas.

Los paradigmas más populares son:

- La programación estructurada
- La programación orientada a objetos
- La programación funcional

Pero en realidad existen cientos más. Afortunadamente, podemos estudiarlos más fácilmente clasificándolos en dos categorías: paradigmas imperativos y paradigmas declarativos. Estos paradigmas son fundamentales en el desarrollo de aplicaciones web online.

TIPOS DE LENGUAJE DE PROGRAMACIÓN

Lenguaje máquina -) 01001001 (Menos abstracto)

Lenguaje ensamblador -) mov, mov 1, mov 2...

Lenguaje de bajo nivel -) C, Perl, Cobol

Lenguaje medio nivel -) JAVA, C++

Lenguaje de alto nivel -) Python, JS, Ruby (Más abstracto)

1

El lenguaje de más bajo nivel (*más próximo a la máquina y más alejado de la comunicación humana*) es el lenguaje máquina, que son series de ceros y unos y se comunican directamente con la máquina sin necesidad de traducción. Esto implica un rápido procesamiento pero un difícil manejo de los programas por parte de los humanos, lo que hizo surgir nuevos tipos de

abstracción.

Tras él, se encuentra el **lenguaje ensamblador** que trabaja con registros de memoria de la computadora de forma directa. Cada lenguaje ensamblador es distinto y propio del procesador con el que trabaja.

Lenguaje de bajo nivel. Se trata de lenguajes de programación que están diseñados para un hardware específico y que por lo tanto no pueden migrar o exportarse a otros computadores. Sacan el mayor provecho posible al sistema para el que fueron diseñados, pero no aplican para ningún otro.

Lenguaje de alto nivel. Se trata de lenguajes de programación que aspiran a ser un lenguaje más universal, por lo que pueden emplearse indistintamente de la arquitectura del hardware, es decir, en diversos tipos de sistemas. Los hay de propósito general y de propósito específico.

Lenguaje medio nivel. Este término no siempre es aceptado, ya que propone lenguajes de programación que se ubican en un punto medio entre los dos anteriores: pues permite operaciones de alto nivel y a la vez la gestión local de la arquitectura del sistema.

Paradigmas de programación

Un paradigma de programación es una manera o estilo de programación de software. Existen diferentes formas de diseñar un lenguaje de programación y varios modos de trabajar para obtener los resultados que necesitan los programadores. Se trata de un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de programas para resolver problemas computacionales.

Paradigma imperativo

Los paradigmas imperativos consisten en una sucesión de instrucciones o conjunto de sentencias, como si el programador diera órdenes concretas. El desarrollador describe en el código paso por paso todo lo que hará su programa. Algunos lenguajes:, **COBOL**, **FORTRAN**, **C**, **C++**,.

Paradigma declarativo

En cuanto a la programación declarativa, no es necesario definir algoritmos, solo le indicas al sistema el problema y automáticamente determina la vía de solución. Es un lenguaje de programación que se adapta a este paradigma es el SQL.

Los 3 sub-paradigmas o subordinados al paradigma de programación declarativa son:

- Programación Funcional
- Programación Lógica
- Programación Reactiva

Programación orientada a objetos

En este modelo de programación, tenemos elementos que denominamos objetos, que poseen características y funciones. Lo importante es que nos permite crear varios objetos y conectarlos entre ellos para crear una aplicación y si algo llegase a fallar, sabríamos cuál de los objetos nos está generando problemas y solucionarlo. Además, podemos agregar un nuevo objeto o datos y funciones a los objetos ya existentes.

3

En la programación orientada a objetos se toman en cuenta diferentes conceptos como:

- Abstracción de datos
- Encapsulación
- Eventos

La programación funcional

Corresponde al patrón de programación que se fundamenta en el concepto de enunciar el proceder de un programa como un modelo funcional matemático más que como secuencias explícitas de instrucciones a un procesador, que es el concepto principal en la programación imperativa.

La programación lógico

Es un paradigma de programación que utiliza los circuitos lógicos en lugar de solamente las funciones matemáticas para controlar cómo se enuncian los hechos y las reglas.

En lugar de un flujo de control cuidadosamente estructurado que dictamina cuándo ejecutar y cómo evaluar las llamadas a funciones u otras instrucciones, las reglas lógicas del programa se escriben como cláusulas o predicados lógicos.

Los estándares de programación

Son conjuntos de reglas y pautas que los programadores siguen al escribir código para garantizar consistencia, legibilidad y calidad en el software. Estos estándares son ampliamente aceptados por la comunidad de desarrollo y suelen estar documentados para facilitar su aplicación.

4 Algunos estándares comunes incluyen:

1. **Estilo de codificación:** Define cómo se estructura el código, incluyendo la indentación, el uso de espacios en blanco, el formato de los comentarios y la longitud de las líneas de código.
2. **Convenciones de nomenclatura:** Establecen reglas para nombrar variables, funciones, clases y otros elementos del código. Pueden incluir reglas sobre el uso de mayúsculas, minúsculas, camelCase, snake_case, entre otros.
3. **Manejo de errores y excepciones:** Especifica cómo se deben manejar las excepciones y errores en el código, incluyendo el uso de try-catch o estructuras similares.
4. **Documentación del código:** Describe cómo se deben documentar las funciones, métodos y clases utilizando comentarios para explicar su propósito, parámetros, valores de retorno, entre otros detalles relevantes.
5. **Prácticas de seguridad:** Incluyen pautas para escribir código seguro y resistente a vulnerabilidades conocidas, como la validación de entradas, el manejo seguro de contraseñas, la prevención de inyecciones de código, entre otros.
6. **Principios de diseño:** Establecen directrices para escribir código que siga principios de diseño sólidos, como el principio de responsabilidad única, la modularidad, la legibilidad y el mantenimiento del código.

Algunos ejemplos de estándares de programación ampliamente utilizados incluyen:

- **PEP 8 (Python Enhancement Proposal 8):** Es el estándar de estilo de codificación para Python.

- **Google Java Style Guide:** Ofrece directrices para escribir código Java en Google.

- **Airbnb JavaScript Style Guide:** Define reglas para escribir código JavaScript en el entorno de desarrollo de Airbnb.

5

- **Microsoft .NET Naming Guidelines:** Proporciona convenciones de nomenclatura para el desarrollo en el entorno .NET de Microsoft.

Estos estándares no son rígidos y pueden variar según el lenguaje de programación, la comunidad de desarrollo o las necesidades específicas del proyecto. Sin embargo, seguir estándares de programación ayuda a mantener un código consistente, facilita la colaboración entre programadores y mejora la legibilidad y mantenibilidad del software.

Entender los diferentes lenguajes de programación, paradigmas y estándares es fundamental para los desarrolladores de software por diversas razones que impactan directamente en la calidad, eficiencia y mantenibilidad del código, así como en la colaboración efectiva dentro de equipos de desarrollo. Aquí hay algunas reflexiones sobre su importancia:

1. **Flexibilidad y versatilidad:** Los diferentes lenguajes de programación y paradigmas ofrecen enfoques únicos para abordar problemas. Comprender una variedad de ellos permite a los desarrolladores elegir la mejor herramienta para cada situación y adaptarse a diferentes contextos y requerimientos del proyecto.

2. **Eficiencia y rendimiento:** Al conocer diferentes lenguajes y paradigmas, los desarrolladores pueden seleccionar el más adecuado para optimizar el rendimiento y la eficiencia del software. Algunos lenguajes son más eficientes para tareas específicas, como el procesamiento de datos, la manipulación de interfaces de usuario o la computación científica.

3. **Mejora en la resolución de problemas:** Los distintos paradigmas de programación ofrecen diferentes enfoques para abordar problemas. Entender varios paradigmas, como la programación orientada a objetos, funcional o lógica, brinda a los desarrolladores un conjunto diverso de herramientas para resolver desafíos de manera más efectiva y creativa.

4. **Facilita la comunicación y colaboración:** Los estándares de programación establecen reglas comunes para la escritura de código, lo que facilita la comprensión y colaboración entre desarrolladores. Un código coherente y legible según estándares bien definidos permite que múltiples programadores trabajen en el mismo proyecto de manera más eficiente.

6

5. **Adaptabilidad y actualización:** El mundo de la tecnología cambia constantemente. Entender diferentes lenguajes y estándares hace que los desarrolladores sean más adaptables a nuevas tecnologías, tendencias y mejores prácticas en el desarrollo de software.

6. **Mantenibilidad del código:** El cumplimiento de estándares de codificación y el uso adecuado de lenguajes y paradigmas contribuyen significativamente a la mantenibilidad del código. Un código bien estructurado y documentado es más fácil de mantener y actualizar a medida que el software evoluciona.

Reflexión / Conclusión

En resumen, comprender los diferentes lenguajes, paradigmas y estándares de programación es esencial para los desarrolladores, ya que les permite elegir las herramientas adecuadas, resolver problemas de manera eficiente, colaborar efectivamente con otros programadores y mantener el software de manera más efectiva a lo largo del tiempo. Esto conduce a la creación de software de mayor calidad y adaptabilidad a medida que evolucionan las necesidades y tecnologías.

¿Qué es un framework?, ¿Qué es una librería?

Un framework sirve para acometer un proyecto en menos tiempo, y en el sector de la programación, con un código más limpio y consistente, de manera rápida y eficaz. El framework ofrece una estructura base que los programadores pueden complementar o modificar según sus

objetivos.

El uso de frameworks permite, principalmente, agilizar procesos de desarrollo porque podemos reutilizar herramientas o módulos: ya tienes el 'esqueleto' sobre el que trabajar. El hecho de escribir código o desarrollar una aplicación más fácilmente te sirve para tener una mejor organización y control de todo el código elaborado, pudiendo usarlo nuevamente en el futuro.

Una librería es uno o varios archivos escritos en un lenguaje de programación determinado, que proporcionan diversas funcionalidades. A diferencia de un framework, una librería no aporta la estructura sobre cómo realizar el desarrollo, sino que proporciona funcionalidades comunes, que ya han sido resueltas previamente por otros programadores y evitan la duplicidad de código. Además reducen el tiempo de desarrollo y aumentan la calidad del mismo.

Comparación de ventajas de frameworks vs librerías

Frameworks y librerías son componentes clave en el desarrollo de software, pero tienen propósitos y características distintas. Aquí hay una comparación de las ventajas de ambos:

Frameworks:

1. Mayor estructura y arquitectura:

Ventaja: Los frameworks proporcionan una estructura predefinida y una arquitectura que guía el desarrollo. Esto puede acelerar la creación de aplicaciones al ofrecer patrones y convenciones.

2. Desarrollo más rápido:

Ventaja: Al tener características y funcionalidades preconstruidas, los frameworks pueden acelerar el proceso de desarrollo. Los desarrolladores pueden enfocarse en la lógica específica de la aplicación en lugar de preocuparse por la infraestructura base.

3. Consistencia y estandarización:

Ventaja: Los frameworks suelen imponer un conjunto de reglas y convenciones, lo que garantiza una mayor consistencia en el código y facilita la colaboración en equipos.

4. Soporte para tareas complejas:

Ventaja: Los frameworks a menudo ofrecen soluciones para tareas complejas como la gestión de la base de datos, la seguridad y el enrutamiento, simplificando el desarrollo de características avanzadas.

5. Comunidad y documentación:

Ventaja: Los frameworks generalmente tienen comunidades activas y una amplia documentación, lo que facilita encontrar ayuda, recursos y soluciones a problemas comunes.

Ventajas de las Librerías:

1. Mayor flexibilidad:

Ventaja: Las librerías son más flexibles y no imponen una estructura específica. Los desarrolladores tienen más libertad para integrar solo las partes necesarias en lugar de adoptar un conjunto completo de características.

2. Menos restricciones:

Ventaja: Al utilizar una librería, los desarrolladores tienen menos restricciones en términos de decisiones de arquitectura y elección de tecnologías. Esto puede ser beneficioso en proyectos donde la libertad es crucial.

3. Tamaño y rendimiento:

Ventaja: Las librerías tienden a ser más ligeras en comparación con los frameworks completos, lo que puede resultar en aplicaciones más pequeñas y eficientes, especialmente cuando solo se necesitan ciertas funcionalidades.

4. Adopción gradual:

Ventaja: Puedes adoptar librerías de forma gradual, integrándolas según sea necesario, lo que puede ser beneficioso en proyectos existentes o donde solo se requieren funcionalidades específicas.

5. Menor curva de aprendizaje:

Ventaja: Al ser menos prescriptivas, las librerías a menudo tienen una curva de aprendizaje más baja, permitiendo a los desarrolladores adoptar y utilizar solo las partes que necesitan sin tener que aprender una arquitectura completa.

¿Qué framework consideras que sería óptimo para una tienda online y por qué?

He elegido el Framework: E-Commerce Express:

Sus características principales son las siguientes

1. Gestión de Productos:

Descripción: Capacidad para agregar, editar y eliminar productos fácilmente.

Funcionalidad: Soporte para diversas categorías, variaciones de productos y gestión de inventario.

2. Carrito de Compras y Proceso de Pago:

Descripción: Facilitar la experiencia de compra del usuario.

Funcionalidad: Integración con pasarelas de pago populares, gestión de carritos de compras, seguimiento de pedidos y opciones de pago seguras.

3. Seguridad:

Descripción: Garantizar la seguridad de las transacciones y la información del cliente.

Funcionalidad: Uso de protocolos de seguridad como HTTPS, gestión de sesiones seguras, y prácticas recomendadas para proteger datos sensibles.

4. Personalización de Tiendas:

Descripción: Permitir a los propietarios de tiendas personalizar la apariencia y la experiencia de usuario.

Funcionalidad: Soporte para temas y plantillas, opciones de personalización de diseño y gestión de contenido.

5. Gestión de Usuarios:

Descripción: Administrar usuarios, clientes y roles.

Funcionalidad: Registro de usuarios, perfiles de clientes, autenticación segura y opciones de gestión de cuentas.

6. Optimización para Móviles:

Descripción:* Garantizar una experiencia de usuario fluida en dispositivos móviles.

Funcionalidad:* Diseño responsivo, navegación intuitiva en dispositivos móviles y rendimiento optimizado para velocidades de carga rápidas.

7. Analíticas y Reportes:

Descripción: Proporcionar información clave sobre el rendimiento de la tienda.

Funcionalidad: Integración con herramientas analíticas, generación de informes sobre ventas, tendencias de productos y comportamiento del usuario.

8. Escalabilidad:

Descripción: Preparar la tienda para el crecimiento futuro.

Funcionalidad: Diseño modular, escalabilidad horizontal y soporte para un aumento de tráfico y catálogo de productos.

9. Soporte Multilingüe y Moneda:

Descripción: Permitir la expansión a mercados internacionales.

Funcionalidad: Traducciones de contenido, selección de monedas y ajustes de impuestos por ubicación.

Este framework hipotético se enfoca en ofrecer una solución completa para el desarrollo de tiendas online, abordando aspectos clave como la usabilidad, la seguridad, la personalización y la escalabilidad.

1. Breve descripción sobre la relevancia del testing y pruebas de código en el ciclo de vida del desarrollo de software.

■ Definición y diferencia entre testing y pruebas de código.

El testing o pruebas de software

es, básicamente, un proceso por el que se comprueba que algo funciona como esperamos que lo haga. En el mundo del desarrollo de software se trata de probar que una pieza de

nuestro código funciona correctamente.

	Pruebas de código	Testing
Alcance	Se centra en la revisión del código fuente y la identificación de posibles problemas,	Puede abarcar diferentes aspectos del software, incluyendo funcionalidad, rendimiento y seguridad
Enfoque	Es realizado por otros miembros del equipo de desarrollo y herramientas automatizadas que analizan el código en busca de problemas específicos.	Puede involucrar a varias partes del equipo, desde desarrolladores hasta probadores y usuarios finales.
Tipos	Puede incluir revisiones de código formales o revisiones informales.	Incluye diferentes tipos de pruebas, como pruebas unitarias y pruebas de integración.

■ Objetivos y beneficios de realizar pruebas.

Las pruebas de código tienen varios objetivos principales que buscan asegurar la calidad del software, detectar errores y mejorar la eficacia del proceso de desarrollo. Aquí están algunos de los objetivos principales de realizar pruebas de código:

Identificar y Corregir Errores:

Uno de los objetivos principales es descubrir y corregir errores en el código antes de que el software se despliegue en entornos de producción. Esto ayuda a garantizar que el software funcione según lo esperado.

Garantizar la Calidad del Software:

Las pruebas de código buscan garantizar que el software cumpla con los estándares de calidad definidos y que cumpla con los requisitos y expectativas del usuario.

Validar la Funcionalidad:

Asegurarse de que todas las funciones y características del software funcionen correctamente y de acuerdo con los requisitos establecidos.

Facilitar el Mantenimiento:

Las pruebas contribuyen a la creación de un código más mantenible, facilitando futuras modificaciones y actualizaciones sin introducir errores no deseados.

Documentar el Comportamiento Esperado:

Las pruebas actúan como una forma de documentación viva del código, describiendo el comportamiento esperado de las funciones y componentes del software.

Aumentar la Confianza en el Software:

Al demostrar que el código ha sido probado exhaustivamente, se incrementa la confianza en la

estabilidad y eficacia del software.

Mejorar la Colaboración:

Las pruebas proporcionan una base sólida para la colaboración entre miembros del equipo de desarrollo, ya que todos pueden entender cómo se supone que debe comportarse el software.

Facilitar la Identificación de Problemas Tempranos:

Las pruebas, especialmente las pruebas automatizadas, permiten identificar problemas tempranos en el ciclo de desarrollo, lo que facilita la corrección antes de que los errores se propaguen.

Optimizar el Rendimiento:

Las pruebas también pueden estar orientadas a evaluar y mejorar el rendimiento del software, identificando posibles cuellos de botella o problemas de eficiencia.

Cumplir con Estándares y Normas:

Las pruebas pueden asegurar que el software cumpla con estándares y normativas específicas, especialmente en industrias reguladas como la salud, finanzas o aviación.

Reducir Costos a Largo Plazo:

Aunque las pruebas pueden requerir inversión inicial de tiempo y recursos, a largo plazo ayudan a reducir costos al prevenir defectos costosos en fases posteriores del desarrollo o incluso después del despliegue.

Los beneficios que tienen son los siguientes:

Reducción de Costos:

Identificar y corregir errores en etapas tempranas del desarrollo reduce los costos asociados con la corrección de errores en etapas más avanzadas.

Entrega más Rápida:

El proceso de pruebas ayuda a identificar y solucionar problemas antes, lo que acelera el desarrollo y permite entregas más rápidas y consistentes.

Mejora de la Calidad:

Al validar continuamente el código, se mejora la calidad general del software, lo que lleva a una mayor satisfacción del usuario.

Mantenibilidad Mejorada:

Las pruebas proporcionan una red de seguridad al realizar cambios en el código, asegurando que las nuevas implementaciones no rompan el código existente.

■ **Descripción detallada de diferentes tipos de pruebas, incluyendo, pero no limitado a: pruebas unitarias, de integración, de sistema, de aceptación, de carga, estrés, etc. Herramientas populares asociadas a cada tipo de prueba. Cada tipo de prueba tiene un propósito específico dentro del proceso de desarrollo de software.**

Aquí tienes una breve descripción de algunas pruebas comunes:

Pruebas Unitarias:

Propósito: Verificar que unidades individuales de código (módulos, funciones, métodos) funcionen correctamente de manera aislada.

Alcance: Centrado en pequeñas unidades de código.

Herramientas Comunes: JUnit, NUnit, PyTest.

Pruebas de Integración:

Propósito: Evaluar la interacción y la correcta integración entre unidades de código.

Alcance: Se centra en cómo las unidades colaboran entre sí.

Herramientas Comunes: TestNG, Mocha, PHPUnit.

Pruebas de Sistema:

Propósito: Validar que el sistema completo funcione según los requisitos especificados.

Alcance: Evalúa el sistema en su totalidad.

Herramientas Comunes: Selenium, Appium, Cypress.

Pruebas de Aceptación:

Propósito: Confirmar que el sistema cumple con los requisitos del cliente y las expectativas del usuario.

Alcance: Centrado en la funcionalidad global del sistema.

Herramientas Comunes:** Cucumber, Behave, SpecFlow.

Pruebas de Carga:

Propósito: Evaluar el rendimiento del sistema bajo condiciones de carga máxima o esperada.

Alcance: Verificar la capacidad del sistema para manejar un volumen significativo de transacciones o usuarios.

Herramientas Comunes: Apache JMeter, LoadRunner, Gatling.

Pruebas de Estrés:

Propósito: Determinar la estabilidad del sistema bajo condiciones extremas o más allá de los límites normales.

Alcance: Evaluar la capacidad de recuperación del sistema después de situaciones adversas.

Herramientas Comunes: Siege, Stress-ng, Apache Bench.

Pruebas de Seguridad:

Propósito: Identificar vulnerabilidades de seguridad en el software.

Alcance: Evaluar la resistencia del sistema a ataques y la protección de datos.

Herramientas Comunes: OWASP ZAP, Nessus, Burp Suite.

Pruebas de Regresión:

Propósito: Asegurar que los cambios en el código no afecten negativamente las funcionalidades existentes.

Alcance: Centrado en áreas afectadas por nuevas implementaciones o modificaciones.

Herramientas Comunes: Selenium, TestNG, Jest.

■ Exploración de técnicas como TDD (Test Driven Development), BDD (Behavior Driven Development) y otras relevantes. Beneficios y retos asociados a cada técnica.

El Test-Driven Development, que en español significa desarrollo dirigido por pruebas, es una metodología de desarrollo de software centrada justamente en lo que su nombre indica: la realización de pruebas unitarias antes de escribir el código.

Principalmente, el TDD sirve para conseguir un código limpio y simple. A través de la realización de pruebas automatizadas, creadas específicamente para cada funcionalidad que se pretenda desarrollar, se van testeando y probando distintas posibilidades. Si se detecta algún fallo, el código se reescribe de nuevo libre de errores.

El BDD (*Behavior Driven Development*), es un proceso de desarrollo de software que busca un lenguaje común entre la parte técnica y la de negocio, y que se guía por el comportamiento esperado de la aplicación.

● Automatización de Pruebas:

- **Introducción a la automatización y sus ventajas.**
- **Herramientas y frameworks populares para la automatización de pruebas.**

La automatización de pruebas de software describe cualquier proceso que implique el uso de herramientas de software independientes para probar el software en desarrollo. Las ventajas que tiene son ahorrar tiempo y recursos al permitirnos mejorar la precisión y la cobertura de las pruebas, al tiempo que identifica fallos y errores de código de forma más eficaz y eficiente.

Selenium

Selenium es una herramienta muy eficaz con integración de otras herramientas y capacidad de pruebas cruzadas en diferentes navegadores y sistemas operativos.

Cypress

Es una herramienta utilizada principalmente para probar aplicaciones web modernas. Se recomienda para quienes están recién empezando a automatizar, pues es fácil de instalar y configurar. Se usa Javascript como lenguaje a la hora de crear script de pruebas.

- **Casos de Uso y Ejemplos:**
- **Presenta algunos ejemplos prácticos o casos de uso donde se aplican distintos tipos de pruebas y técnicas de testing en proyectos reales.**

Pruebas Unitarias:

Caso de uso: Desarrollo de una nueva función en un sistema de gestión de pedidos en línea.

Ejemplo: Escribir pruebas unitarias para verificar que cada función individual (como el cálculo del total del pedido) produce los resultados esperados.

Pruebas de Integración:

Caso de uso: Integración de un nuevo módulo de pago en un sistema de comercio electrónico.

Ejemplo: Verificar que el módulo de pago interactúa correctamente con otras partes del sistema, como el carrito de compras y la base de datos de productos.

Pruebas Funcionales:

Caso de uso: Desarrollo de una aplicación de chat en tiempo real.

Ejemplo: Realizar pruebas para asegurarse de que las funciones principales, como enviar mensajes, recibir notificaciones y ver la lista de contactos, funcionen como se espera.

Pruebas de Regresión:

Caso de uso: Implementación de una corrección de errores en un sistema de gestión de inventario.

Ejemplo: Ejecutar pruebas de regresión para asegurarse de que la corrección de errores no haya introducido nuevos problemas en otras partes del sistema.

Pruebas de Aceptación del Usuario:

Caso de uso: Desarrollo de un portal en línea para la gestión de proyectos.

Ejemplo: Invitar a usuarios finales a probar el sistema y verificar que cumple con sus requisitos y expectativas antes de su lanzamiento.

Pruebas de Estrés:

Caso de uso: Implementación de una aplicación web para la venta de boletos en línea.

Ejemplo: Realizar pruebas de estrés para simular la carga máxima de usuarios y verificar cómo responde la aplicación bajo una carga pesada.

Pruebas de Seguridad:

Caso de uso: Desarrollo de un sistema de gestión de información confidencial.

Ejemplo: Realizar pruebas de seguridad para identificar posibles vulnerabilidades, como la validación inadecuada de entradas, y garantizar la protección de los datos sensibles.

Pruebas de Usabilidad:

Caso de uso: Creación de una aplicación móvil para la reserva de servicios de transporte.

Ejemplo: Conducir pruebas de usabilidad para evaluar la facilidad de uso, la navegación y la satisfacción del usuario durante el proceso de reserva.

Conclusión personal tras el trabajo

En forma de conclusión tras realizar este informe me gustaria resaltar la importancia de comprender los diferentes lenguajes de programación, paradigmas y estándares ya que es crucial en el desarrollo de software debido a su impacto directo en la eficiencia, mantenibilidad y escalabilidad de los sistemas. La elección adecuada de lenguajes y paradigmas permite adaptarse a los requisitos específicos del proyecto, optimizando el rendimiento y facilitando la colaboración entre equipos. Además, la adherencia a estándares promueve la interoperabilidad y la compatibilidad, facilitando la integración de componentes y el desarrollo de software sostenible a lo largo del tiempo. En última instancia, una comprensión profunda de estas dimensiones contribuye a la creación de software robusto, flexible y capaz de evolucionar con éxito en un entorno tecnológico en constante cambio.

¿Qué es un framework?, ¿Qué es una librería?

Un framework sirve para acometer un proyecto en menos tiempo, y en el sector de la programación, con un código más limpio y consistente, de manera rápida y eficaz. El framework ofrece una estructura base que los programadores pueden complementar o modificar según sus objetivos.

El uso de frameworks permite, principalmente, agilizar procesos de desarrollo porque podemos reutilizar herramientas o módulos: ya tienes el 'esqueleto' sobre el que trabajar. El hecho de escribir código o desarrollar una aplicación más fácilmente te sirve para tener una mejor organización y control de todo el código elaborado, pudiendo usarlo nuevamente en el futuro.

Una librería es uno o varios archivos escritos en un lenguaje de programación determinado, que proporcionan diversas funcionalidades. A diferencia de un framework, una librería no aporta la estructura sobre cómo realizar el desarrollo, sino que proporciona funcionalidades comunes, que ya han sido resueltas previamente por otros programadores y evitan la duplicidad de código. Además reducen el tiempo de desarrollo y aumentan la calidad del mismo.

Citas

[Wikipedia](#)

[Paradigmas](#)

[Chat gpt](#)

[Más fuentes](#)